NEAR EAST UNIVERSITY

GRADUATED SCHOOL OF APPLIED AND SOCIAL SCIENCES

# WIRELESS EMERGENCY WARNING SYSTEMS DESIGN AND IMPLEMENTATION

Mehmet UĞURLU

Master Thesis

Department of Computer Engineering

Nicosia-2003

*"Communities that do not develop information and technology, which is the immediate product of information, lose their independence and as its consequence lose their happiness."*

"Bilgi ve onun ürünü olan teknolojiyi üretmeyen toplumlar bağımsızlıklarını dolayısıyla mutluluklarını yitirirler."

Mehmet Uğurlu: Wireless Emergency Warning Systems Design & Implementation

Approval of the Graduate School of Applied & Social Sciences

Prof.Dr.Fakhraddin Mamedov
Director

We certify that this thesis is satisfactory for the award of the degree of
Master of Science in Computer Engineering

Prof.Dr.Şenol Bektaş
Supervisor

Examining Committee in charge:

Prof.Dr.Fakhraddin Mamedov,Committe Chairman.Dean of
Engineering Faculty,NEU

Assoc.Prof.Dr. Doğan İbrahim,Committe Member,Chairman
Computer Engineering Department,NEU

Assoc.Prof.Dr. Rahib Abiyev,Committee Member,
Computer Engineering Department,NEU

## ACKNOWLEDGMENTS

I express sincere appreciation to Prof.Dr.Fahreddin Mamedov for his guidence and insight thoroughout the research. Thanks go to the other faculty members, Prof.Dr.Şenol Bektaş for his suggestions and comments and thanks to Research Assistant Aylin Aytaç for her help.Also special thanks to my colleagues.

## ABSTRACT

The life is not only becoming easy, but gaining speed also by technological improvements. Especially the wireless technology has become an indispensable factor in communication in which a great advance has been established in last several years. In many fields, people are racing with time and requiring correct information, in appropriate location and time. Administrations are making a loss, since they can't reach information instantly when they need. Mobile phones, Internet, fax-data communication, SMS (short message service) and WAP are becoming widespread in popular fields rapidly. By widening in usage, the GSM (Global System for Mobile Communications) technology is taking part in industrial organizations, hotels, financial associations, residences, media and shopping centers.

The "Wews System"(wireless emergency warning system) that was developed in this project, provides an SMS communication from PC's fed by sensor signals to mobile phones of the users. The messages include information about failures, dangerous situations and production processes in industrial organizations, residences and shopping centers, firstly. In this study, content information is given on usage of sensors, technical features, montage, electrical and logical connections, pc ports and low level programming and GSM network infrastructure.

Keywords: sensors,wews,pc ports,sms,gsm

# TABLE OF CONTENTS

## LIST OF FIGURES

# INTRODUCTION

The life is not only becoming easy, but gaining speed also by technological improvements. Especially the wireless technology has become an indispensable factor in communication in which a great advance has been established in last several years. In many fields, people are racing with time and requiring correct information, in appropriate location and time. Administrations are making a loss, since they can't reach information instantly when they need. Mobile phones, Internet, fax-data communication, SMS (short message service) and WAP are becoming widespread in popular fields rapidly. By widening in usage, the GSM (Global System for Mobile Communications) technology is taking part in industrial organizations, hotels, financial associations, residences, media and shopping centers.

The aim of this thesis is to develop architectural model for organization's Wireless Emergency Warning Systems by using GSM technology.

Thesis consist of five chapter, conclusion and two appendixes.

In first chapter we informed about sensors technology, configuration of sensors, types of sensors and sensor connections.

In second chapter we informed about serial and parallel ports configuration, port communication and port connections.

In third chapter the main subject is wireless emergency warning system design&implementation .This software program used AT commands and functions is coded on the $C^{++}$ programming language. This program check the sensors and then predefined phone numbers in the **Phone Numbers Text File** send SMS from GSM that the connected ports than SMS result report sent to **Sonuc Text File** as deliver if it is success, not deliver if it is not success.

In four chapter we informed about the base of GSM network ,GSM technology ,SMS , GSM Subscriber Services.

In five chapter we informed about improvement of wireless technology and it's market.

In conclusion I wish to say that all results ( what I mentioned in all chapters) indicated that this technology effect a lot of area and moreover WEWS system evaluate the signals that it takes from the automation system or organization and transfer  the necessary information to the predefined mobile telephone number by making use of SMS.

In addition, we give figure of product like our work in first appendix. We also added informations about AT commands from ETSI (Europe Telecommunication Standard Institute).

# CHAPTER 1

# SENSORS TECHNOLOGY

## 1.1 Overview

Microsensors have become an essential element of process control and analytical measurement systems, finding countless applications in, for example, industrial monitoring, factory automation, the automotive industry, transportation, telecommunications, computers and robotics, environmental monitoring, health care, and agriculture; in other words, in almost all spheres of our life. The main driving force behind this progress comes from the evolution in the signal processing. With the development of microprocessors and application-specific integrated circuits (1C), signal processing has become cheap, accurate, and reliable—and it increased the intelligence of electronic equipment. In the early 1980s a comparison in performance/ price ratio between microprocessors and sensors showed that sensors were behind. This stimulated research in the sensor area, and soon the race was on to develop sensor technology and new devices. New products and companies have emerged from this effort, stimulating further advances of microsensors. Application of sensors brings new dimensions to products in the form of convenience, energy savings, and safety . Today, we are witnessing an explosion of sensor applications. Sensors can be found in many products, such as microwave and gas ovens, refrigerators, dishwashers, dryers, carpet cleaners, air conditioners, tape recorders, TV and stereo sets, compact and videodisc players. And this is just a beginning.

## 1.2 Sensor Classification

Sensing the real world requires dealing with physical and chemical quantities that are diverse in nature. From the measurement point of view, all physical and chemical quantities (measurands) can be divided into six signal domains.

The thermal signal domain: the most common signals are temperature, heat, and heat flow.

The mechanical signal domain: the most common signals are force, pressure, velocity, acceleration, and position.

The chemical signal domain: the signals are the internal quantities of the matter such as concentration of a certain material, composition, or reaction rate.

The magnetic signal domain: the most common signals are magnetic field in tensity, flux density, and magnetization.

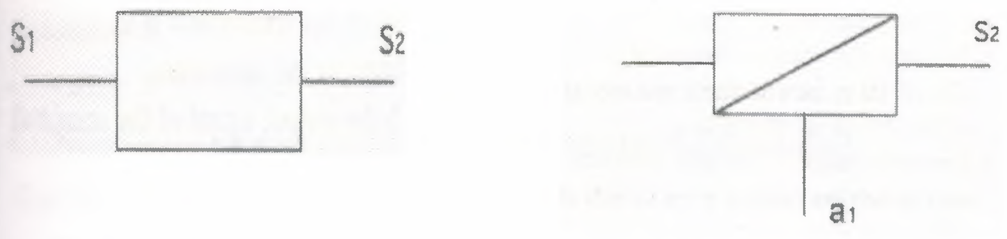
The radiant signal domain: the signals are quantities of the electromagnetic waves such as intensity, wavelength, polarization, and phase.

The electrical signal domain: the most common signals are voltage, current, and charge.

As mentioned, sensors convert nonelectrical physical or chemical quantities into electrical signals. It should be also noted that the principle of operation of a particular sensor is dependent on the type of physical quantity it is designed to sense. Therefore, it is no surprise that a general classification of sensors follows the classification of physical quantities. Accordingly, sensors are classified as thermal, mechanical, chemical, magnetic, and radiant.

There is also a classification of sensors based on whether they use an auxiliary energy source or not. Sensors that generate an electrical output signal without an auxiliary energy source are called *self-generating* or *passive*. An example of this type of sensor is a thermocouple. Sensors that generate an electrical output signal with the help of an auxiliary energy source are called *modulating* or *active*. Figure 1.1 shows symbolic presentations of self-generating and modulating sensors. Here, *Si* represents the input signal, $s_2$ is the output signal, and a, is the auxiliary energy source. In modulating sensors, the auxiliary energy serves as a main source for the output signal, and the measured physical quantity modulates it. This class of sensors includes magnetotransistors and phototransistors. Modulating sensors are the best choice for the measurement of weak signals.

In addition to the preceding classifications, there are many others based on some common features. A good example is automotive, where



A B

**Figure 1.1** Symbolic presentation of self-generating and modulating sensor: (a) self-generating sensor; (b) modulating sensor, where $s_1$ is the input signal, $s_2$ is the output signal, $a_1$ is the auxiliary energy source.

the common feature is the application in automobiles for engine and vehicle control. A curious reader can find more information about the classification of sensors in a recently published book on silicon sensors

### 1.3 Sensor Parameters

Performance of sensors, like other electronic devices, is described by parameters.

- Absolute sensitivity is the ratio of the change of the output signal to the change of the measurand (physical or chemical quantity).
- Relative sensitivity is the ratio of a change of the output signal to a change in the measurand normalized by the value of the output signal when the measurand is 0.
- Cross sensitivity is the change of the output signal caused by more than one measurand.
- Direction dependent sensitivity is a dependence of sensitivity on the angle be tween the measurand and the sensor.
- Resolution is the smallest detectable change in the measurand that can cause a change of the output signal.
- Accuracy is the ratio of the maximum error of the output signal to the full-scale output signal expressed in a percentage.
- Linearity error is the maximum deviation of the calibration curve of the output signal from the best fitted straight line that describes the output signal.
- Hysteresis is a lack of the sensor's capability to show the same output signal at a given value of measurand regardless of the direction of the change in the

measurand.

- Offset is the output signal of the sensor when the measurand is 0.
- Noise is the random output signal not related to the measurand.

- Cutoff frequency is the frequency at which the output signal of the sensor drops to 70.7% of its maximum.
- Dynamic range is the span between the two values of the measurand (maximum and minimum) that can be measured by sensor.
- Operating temperature range is the range of temperature over which the output signal of the sensor remains within the specified error.

It should be pointed out that in addition to these common parameters, other parameters are often used to describe other unique properties of sensors.

## 1.4 A Seamless Sensor System

Sensing systems are generally used for process control and measurement instrumentation. A simple block diagram of a sensing system is shown in Figure 2 As can be seen, the term *transducer* is used for both the input and the output blocks of the sensing system. The role of the input transducer is to get information from the real world about a physical or chemical quantity; in other words, tó "sense the world." This is the reason why input transducers are commonly called *sensors.* Often the electrical signals generated by sensors are weak and have to be amplified or processed in some way. This is done by the signal processing part of the sensing system. Finally, the role of the output transducer is to convert an electrical signal into a form acceptable for our senses or to initiate some "action," for example, opening or closing a valve. For this reason, output transducers are often called *actuators.* A simple block diagram of the sensing system, as just described, helps to grasp the basic concept of sensing, but it really does not tell the whole story.

Much has been written about the phenomenal development of microelectronics and the strong influence of microprocessors and other integrated circuits on sensing systems. Figure 3 shows a typical sensing system composed of the many devices

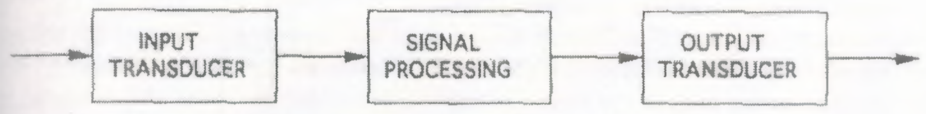


**Figure 1. 2** Simple block diagram of the sensing system.

of modern microelectronics . Following the signal path in Figure 3, one can see that the electrical signals created by sensors are amplified, converted to digital form, and transferred to a microprocessor. The microprocessor also controls a variety of actuators through the interface circuits, where the signals are converted back to analog form and used to drive the actuators. The entire sensing system thus can form a closed control loop.

Also, the microprocessor may communicate with a higher level control computer, making the sensing system, shown in Figure 1.3, part of a larger system. Currently, the type of sensing system shown in Figure 1.3 is spatially distributed and made of separate functional blocks. Point-to-point wiring is typically used for the electrical connection between the blocks. Many experts expect in the future that such sensing systems will be integrated into a single chip, forming a "smart" sensor or "seamless" sensor system, where boundaries between the functional blocks will not be apparent.

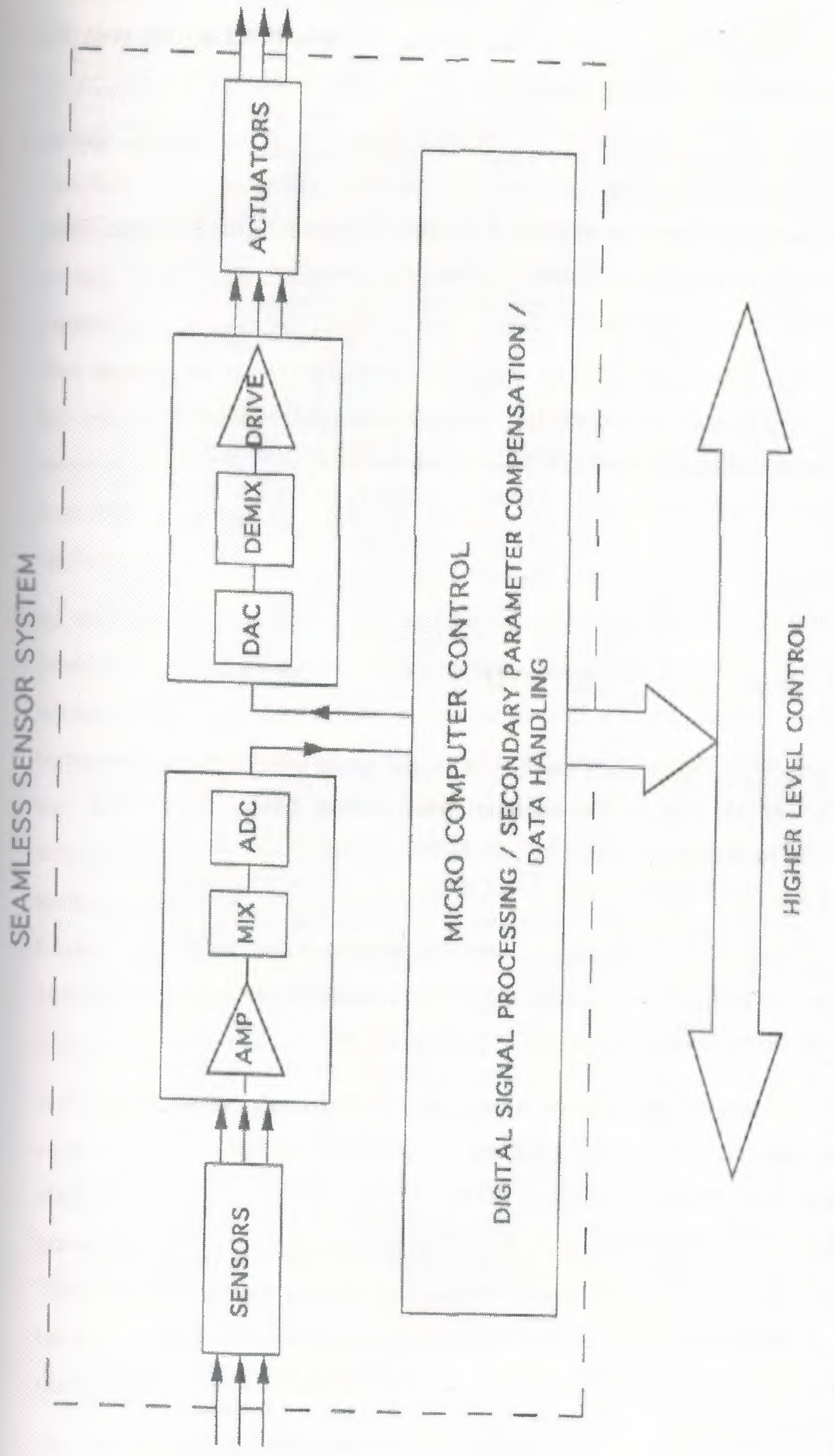


**Figure 1.3** Seamless sensor system on the chip.

## 1.5 Semiconductor Sensor

Semiconductor sensors are transducers that convert mechanical signals into electrical signals. These devices are widely used for the measurement and control of physical variables. Microphones are used in audio systems. Pressure sensors are used in fluidic, pneumatic, and tactile detection systems. Accelerometers are used in navigational and air-bag deployment. Magnetic sensors are used in positional control. Infrared and visible light sensors are used in cameras and night-vision systems. Temperature and flow sensors are used in air conditioning and automotive systems. Chemical sensors are used in biological diagnostic systems. The list of applications of these devices is enormous, and it is growing on a yearly basis. Currently, there is a large demand for low-cost, accurate, and reliable sensors for industrial and consumer product applications.

In the past twenty years, the application of microelectronic technology to the fabrication of mechanical devices greatly stimulated research in semiconductor sensors. Such microfabricated devices are micromachined sensors. Micromachining technology takes advantage of the benefits of semiconductor technology to address the manufacturing and performance requirements of the sensor industry. The versatility of semiconducting materials and the miniaturization of VLSI patterning techniques promise new sensors with better capabilities and improved performance-to-cost ratio over those of conventionally machined devices. Figure 4 shows an example of a microelectromechanical sensing system (MEMS) used in the deployment of air-bags which illustrates the integration of electrical and mechanical devices.

A major factor that contributes to the cost of manufactured products is the overhead expense on production facilities. Technology-based products such as precision electronic and mechanical devices require expensive facilities and highly skilled laborers. These costs are largely independent of the number of products produced. Therefore, the per-unit cost of manufactured goods decreases as the production volume increases. Maximizing throughputs without sacrificing product quality is one of the major goals of manufacturers.

An example that illustrates this point occurs in the microelectronics industry. Integrated-circuit technology allows thousands of electronic circuits to be batch-fabricated simultaneously through a single pass of processing sequences. Batch-fabrication of microelectronic circuits was made possible through the invention of planar technology. In the planar manufacturing process, three-dimensional devices are built on a wafer substrate using stacked layers of planar materials with different but coordinated two-dimensional patterns.

Analog Devices' ADXL-50, the industry's first surface micromachined accelerometer, includes signal conditioning on chip.

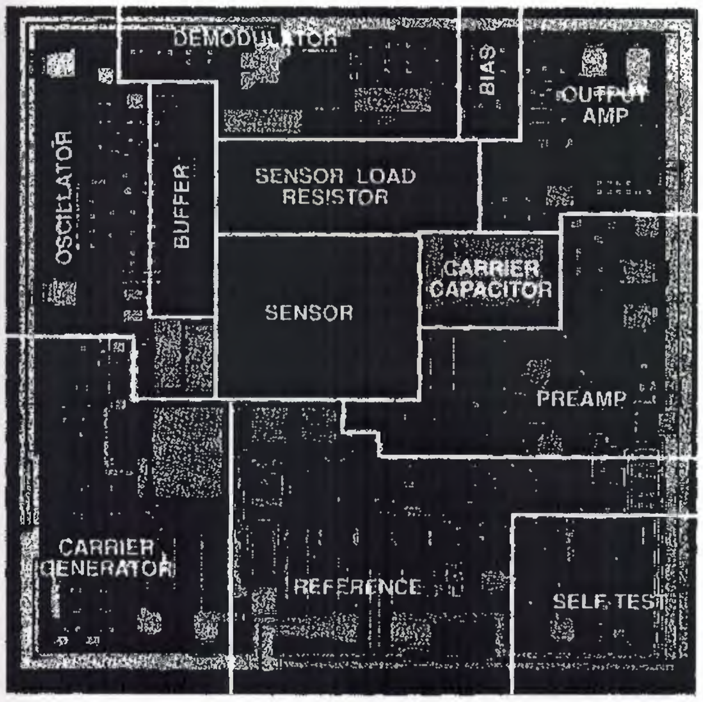


**Fig 1.4** State of the art surface micromachined accelerometer that integrates micro-mechanical sensors with BICMOS technology. (Courtesy of Analog Devices.)

By optically repeating the patterns on the wafer, many units are fabricated with just one pass of the process . Micromachined sensors benefit from the same planar manufacturing processes.

Because sensors receptive to different physical variables are structurally different, in general, there is no single technology that allows for the fabrication of a wide variety of sensors. However, there are two major classifications of microsensor technologies. *Bulk-micromachined* sensors are primarily made by the accurate machining of a relatively thick substrate. ***Swface-micrimachined*** sensors are primarily constructed from stacked thin films. Both technologies use materials and processes borrowed from VLSI technology. The three processes of deposition, lithography, and etching are sufficient to construct a wide variety of mechanical structures required for specific sensors. A fundamental sensor-fabrication problem is the development of a suitable fabrication-process sequence of these basic machining steps that define the desired shape and function of the device.

## 1.6 Sensor Types

**a.** Acoustic sensors

Acoustic sensors are devices that employ elastic waves at frequencies in the megahertz to low gigahertz range to measure physical, chemical, or biological quantities. Their high sensitivity makes these devices particularly attractive for chemical vapor and gas sensing. In many cases, the output of these sensors is a frequency, which can be measured simply and accurately with an electronic counter. With proper design, these sensors can be quite stable, permitting a \ large dynamic range to be realized.

**b.** Mechanical Semiconductor Sensors

Silicon is used for mechanical sensors, because it combines well-established electronic properties with excellent mechanical properties. Other advantages of silicon include drastically reduced dimensions and mass, batch fabrication and easy interfacing or even integration with electronic circuits and microprocessors. Interest in the mechanical properties of silicon and its use for sensors started with the discovery of its piezoresistivity. The first mechanical sensor was the piezoresistive pressure sensor, but since the development of this sensor, a very wide variety of sensors has been conceived and produced.

**c.** Magnetic sensor

A magnetic sensor is capable of converting a magnetic field into a useful electrical / signal. A magnetic sensor is also needed whenever a nonmagnetic signal is / detected by means of an intermediary conversion into a magnetic signal in a so-called tandem transducer. Examples are the detection of a current through its magnetic field or the mechanical displacement of a magnet. Thus, we can distinguish two groups of direct and indirect magnetic-sensor applications.[3]

In *direct* applications, the magnetic sensor is part of a magnetometer. Examples are the measurement of the geomagnetic field, the reading of magnetic data storage media, the identification of magnetic patterns in cards or banknotes, and the control of magnetic apparatus.

In *indirect* applications, the magnetic field is used as an intermediary carrier for detecting nonmagnetic signals. Examples are potential-free current detection for overload protection, integrated watt-hour meters, and contactless linear or angular position, displacement, or velocity detection using a permanent magnet.

These applications require the detection of magnetic fields in the micro- and millitesla range, which can be achieved by integrated semiconductor sensors.

Contactless switching for keyboards or collectorless DC motor control, displacement detection for proximity switches or crankshaft position sensors, and current detection seem to comprise most of the large-scale applications of magnetic sensors. It is for these large-scale applications that inexpensive batch-fabricated semiconductor magnetic sensors are highly desirable. It is unlikely that integrated silicon magnetic sensors will ever replace nuclear magnetic resonance (NMR) magnetometry with resolution in the nanotesla region, let alone the superconducting quantum interference devices (SQUID) resolving picotesla fields occurring in biomagnetometry.

With respect to the above ranges of magnetic resolution, we recall the following magnetic units. As a measure for the magnetic field strength *H* we use the related magnetic induction *B,* whose unit is 1 tesla = 1 V-s/m$^2$. This is the inverse of the unit of carrier mobility, namely 1 m$^2$/V-s = 10$^4$cm$^2$/V-s=l T. The product of magnetic induction and mobility is a dimensionless number which controls the strength of the galvanomagnetic effects.

Semiconductor magnetic sensors including integrated silicon and GaAs • sensors are useful in the range between 1 /iT and 1 T. Here are some examples II of magnetic induction within that range:

- geomagnetic field 30-60 /*T
- magnetic storage media about 1 mT
- permanent magnets in switches 5-100 mT
- conductor carrying a 10 A current 1 mT
- superconducting coils 10-20 T

**d.** Radiationsensors

Radiation sensors_transform incident radiant signals into standard electrical output signals to be used for data collection,processing and storage.Radiant signals can be categorized into one of the following types: electromagnetic, neutrons, fast electrons, or heavy-charge particles. Electromagnetic radiation and neutrons are uncharged, while fast electrons and heavy-charged particles are charged-particulate radiation.All radiant signals originate in atomic or nuclear processes, and similar techniques are used for their detection

**e.** Thermal sensors

The operation of thermal sensors generally can be described in three steps. In the first step the non-thermal quantity is transduced into a thermal quantity by either transducing the power of the non-thermal quantity directly into a heat flow (the self-generating sensors), or by exerting influence by the non-thermal signal on a heat-flow generated by the sensor itself (the modulating sensors). In the second step, the heat flow in the sensor is converted into a temperature difference by means of a thermal resistance. In silicon sensors, micromachining has proved to be a powerful tool for obtaining optimized thermal structures. Closed membranes, cantilever beams and bridges, and floating membranes are often encountered structures in which thermal resistances ahd parallel conductances can be defined very accurately in a simple way. In the third step, the temperature difference is transduced into an electrical signal. The main elements used for this step are transistors or resistors that measure the absolute temperature and are suited for smart sensors, and thermocouples which are interesting for measuring temperature differences, as they can do this without offset and will not spoil the offsetless character of self-generating sensors.

f. Chemical sensors

All the forms of semiconductor chemical sensors have one major problem. In order to detect the chemical species of interest, the sensors must be exposed, unprotected, to the ambient solution or gas. It is difficult to make them reversibly reactive to the gases of interest and nonreactive with respect to all other possible chemical species that may appear in the atmosphere or liquid. Fortunately, in most cases, the form of interference is known and an ideal sensor is not required. For example, the degrading effect of $H_2S$ or $Cl_2$ on some sensors is no problem if the user is sure these particular species will not be present.

Sensors from semiconducting metal oxides have the desired feature of low cost, good sensitivity, and convenient form of response (a simple change in resistance). These features have made, and undoubtedly will continue to make, these sensors popular. However, the sensors have problems in reproducibility, stability and selectivity. Every improvement in these aspects will undoubtedly increase the usage of the devices.

g. Biosensor

Biosensors are a special class of chemical sensors that take advantage of the high selectivity and sensitivity of biologically active materials. This high selectivity and sensitivity of the biological material is a result of millions of years of evolution of life on earth, since much of the communication among /biological organisms is based on chemical signals, whether the senses of smell and taste, or immunological reactions, or pheromones, or "hunting" of single-celled organisms. Even the senses of vision, hearing, and touch are transmitted by chemical communication through the nervous system. These communication processes can be considered to be "bio-recognition" processes. Thus, the potential to use these bio-recognition processes as inputs to a sensor is apparent. The diversity of life is reflected in the large variety of biosensors, since there are biological chemicals, organelles, cells, tissues, and organisms that react to everything from small inorganic molecules, such as oxygen, to large, complicated proteins and carbohydrates.

# CHAPTER 2

# PORTS COMMINICATIONS

## 2.1 Parallel Port

DB25 (Figure 2.1) connector with an 8 bit data bus (Pin 2-7) which is more popularly used for computer printers while is still used for other devices.

The standard length of Printer Parallel cables is a maximum of 15 feet although there are 50 foot cables it is not recommended that these cables be used as it can create poor connection and data signals.

## 2.2 Types Of Parallel Ports

**Unidirectional -** 4-bit standard port which by factory default did not have the capability of transferring data both ways.

**Bi-directional -** 8-bit standard port which was released with the introduction of the PS/2 port in 1987 by IBM and are still found in computers today. The Bi-directional port is cable of sending 8-bits input and output. Today on multifunction printers this port can be referred to as a bi-directional, Centronics, PS/2 type or standard port.

**EPP -** The Enhanced Parallel Port (EPP) was developed in 1991 by Intel, Xircom and Zenith Data Systems and operates close to ISA bus speed and can achieve transfer rates up to 1 to 2MB/sec of data.

EPP version 1.7 released in 1992 and later adapted into the IEEE 1284 standard. All additional features are adapted into the IEEE standard.

EPP version 1.9 never existed.

**ECP -** The Enhanced Capabilities Port (ECP) was developed by Microsoft and Hewlett-Packard and announced in 1992 is an additional enhanced Parallel port. Unfortunately with ECP it requires an additional DMA channel which can cause resource conflicts.

## 2.3 Parallel Port Devıces

**Printer** - The most common use for the Parallel port.

**Scanner** - Another commonly used parallel device is the Parallel scanner. Parallel scanners are a popular alternative to SCSI scanners because of how easy they are to to install.

**External Drives** - Another popular use of the Parallel ports are external drives such as the,sensors and other devices  which can be easily removed from one computer and placed onto another.

| Port Name | Adress(Hex) | (Hex) |
|---|---|---|
| DATA(D) | 378 | 378 |
| STATUS(S) | 378+1 | 379 |
| CONTROL(C ) | 378+2 | 37A |

**Layout**

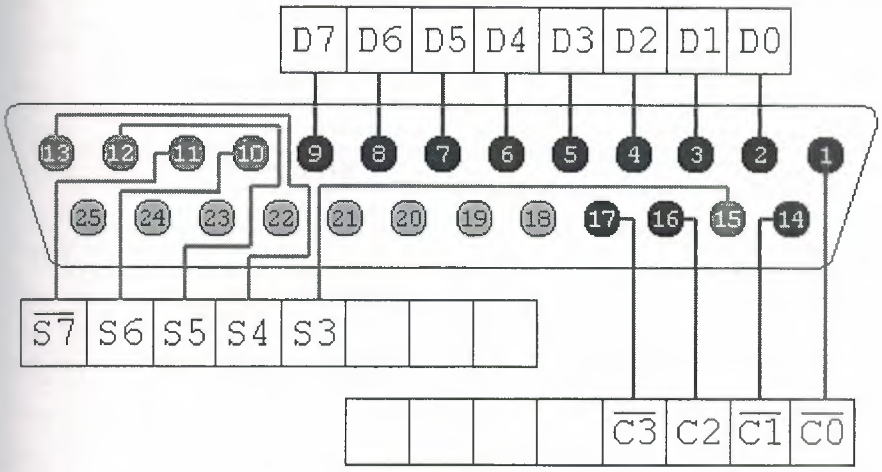


**Figure 2.1** DB25 Connector

| Signal Name | BIT | PIN | |
|---|---|---|---|
| -Strobe | ¬C0 | 1 | Output |
| +Data Bit 0 | D0 | 2 | Output |
| +Data Bit 1 | D1 | 3 | Output |
| +Data Bit 2 | D2 | 4 | Output |
| +Data Bit 3 | D3 | 5 | Output |
| +Data Bit 4 | D4 | 6 | Output |
| +Data Bit 5 | D5 | 7 | Output |
| +Data Bit 6 | D6 | 8 | Output |
| +Data Bit 7 | D7 | 9 | Output |
| -Acknowledge | S6 | 10 | Input |
| +Busy | ¬S7 | 11 | Input |
| +Paper End | S5 | 12 | Input |
| +Select In | S4 | 13 | Input |
| -Auto Feed | ¬C1 | 14 | Output |
| -Error | S3 | 15 | Input |
| -Initialize | C2 | 16 | Output |
| -Select | ¬C3 | 17 | Output |
| Ground | - | 18-25 | Ground |

| PIN | PURPOSE |
|---|---|
| Pin 1 | -Strobe |
| Pin 2 | +Data Bit 0 |
| Pin 3 | +Data Bit 1 |
| Pin 4 | +Data Bit 2 |
| Pin 5 | +Data Bit 3 |
| Pin 6 | +Data Bit 4 |
| Pin 7 | +Data Bit 5 |
| Pin 8 | +Data Bit 6 |
| Pin 9 | +Data Bit 7 |
| Pin 10 | -Acknowledge |
| Pin 11 | +Busy |
| Pin 12 | +Paper End |
| Pin 13 | +Select |
| Pin 14 | -Auto Feed |
| Pin 15 | -Error |
| Pin 16 | -Initialize Printer |
| Pin 17 | -Select Input |
| Pin 18 | -Data Bit 0 Return (GND) |
| Pin 19 | -Data Bit 1 Return (GND) |
| Pin 20 | -Data Bit 2 Return (GND) |
| Pin 21 | -Data Bit 3 Return (GND) |
| Pin 22 | -Data Bit 4 Return (GND) |
| Pin 23 | -Data Bit 5 Return (GND) |
| Pin 24 | -Data Bit 6 Return (GND) |
| Pin 25 | -Data Bit 7 Return (GND) |

The following is an explanation of each of the above purposes.

**Pin1** = Data acknowledgement when the signal is low.
**Pin 2 - 9** = Data transfer pins.
**Pin 10** = Acknowledge that the data has finished processing and when the signal is high indicates ready for more.
**Pin 11** = When the signal goes high indicate that the printer has accepted the data and is processing it. Once this signal goes low and Pin 10 goes high will accept additional data.
**Pin 12** = Printer paper jam when signal is high or no signal if printer jam.
**Pin 13** = When high signal printer is indicating that it is on-line and ready to print.
**Pin 14** = When low signal PC has indicated that the printer inset a line feed after each line.
**Pin 15** = Printer sends data to the computer telling it that an error has occurred.
**Pin 16** = When low signal PC has requested that the printer initiate a internal reset.
**Pin 17** = When low signal the PC has selected the printer and should in return prepare for data being sent.
**Pin 18 - 25** = Ground.

## 2.4 Serıal Port

The serial port is an Asynchronous port which transmits one bit of data at a time, usually connecting to the UART Chip. Serial Ports are commonly found on the majority of PC Compatible computers. Usually referred to as a DB9 or DB25 connection both of which adhere to the RS-232c interface standard and defined in ISO 2110 and ISO 4902. D represents the shape of the connector if placed vertically as shown in the below illustrations. The number 9 / 25 indicating the number of pins found on the connector. DB9 Serial connections are now commonly found on modern PC's where DB25 is commonly found on older computers.

## 2.5 Serıal Port Devıces

The following is a listing of various hardware components which can be purchased and used with your Serial port.

**Mouse** - One of the most commonly used devices for serial ports, usually used with computers with no PS/2 Ports or laptop computers.

**Modem** - Another commonly used device for serial ports. Used commonly with older computers however is also commonly used with computers for its ease of use.

**Network** - One of the original uses of the serial port, which allowed two computers to connect together and allow large files to be transferred between the two.

**Printer** - Today is not commonly used device for serial ports (not applicable to the DB25 or Parallel Port). However was frequently used with older printers and plotters.

**External Drives** – In this project we use celuler phone.

## 2.6 DB9 Informatıon

In the illustration below you can notice several factors to help correctly identify the DB9 Serial connection. First you will notice that the DB9 connection has 9 pins which are each described in the below chart. The illustration below is an example of the female serial connector which would usually be located on the connector that would connect to the computer. each serial connector generally has two screws measuring .3 cm to allow the serial connection to be securely connected to the back of the computer.

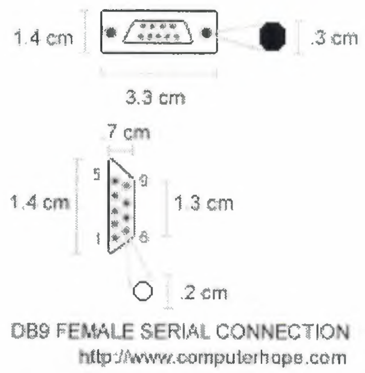


**Figure 2.2** DB9 Femail Serial connection

**Identifying:**

The DB9 serial connection is identified first by its 9 pins.
The DB9 is shaped like a D.
The DB9 will generally be a male connector on the back of the computer.

The following is a listing of each of the pins located on the DB9 connector and what each of these pins are for.

| PIN | PURPOSE | SIGNAL NAME |
|---|---|---|
| Pin 1 | Data Carrier Detect | DCD |
| Pin 2 | Received Data | RxData |
| Pin 3 | Transmitted Data | TxData |
| Pin 4 | Data Terminal Ready | DTR |
| Pin 5 | Signal Ground | Gnd |
| Pin 6 | Data Set Ready | DSR |
| Pin 7 | Request To Send | RTS |
| Pin 8 | Clear To Send | CTS |
| Pin 9 | Ring Indicator | RI |

## 2.7 DB25 Informatıon

In the illustration below you can notice several factors to help correctly identify the DB25 port. First you will notice that the DB25 connection has 25 pins which are each illustrated in the below chart.

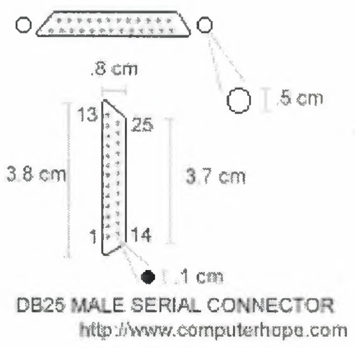


**Figure 2.3** Mail Serial Connection

**Identifying:**

The DB25 serial connection is identified first by its 25 pins.

The DB25 is shaped like a D.

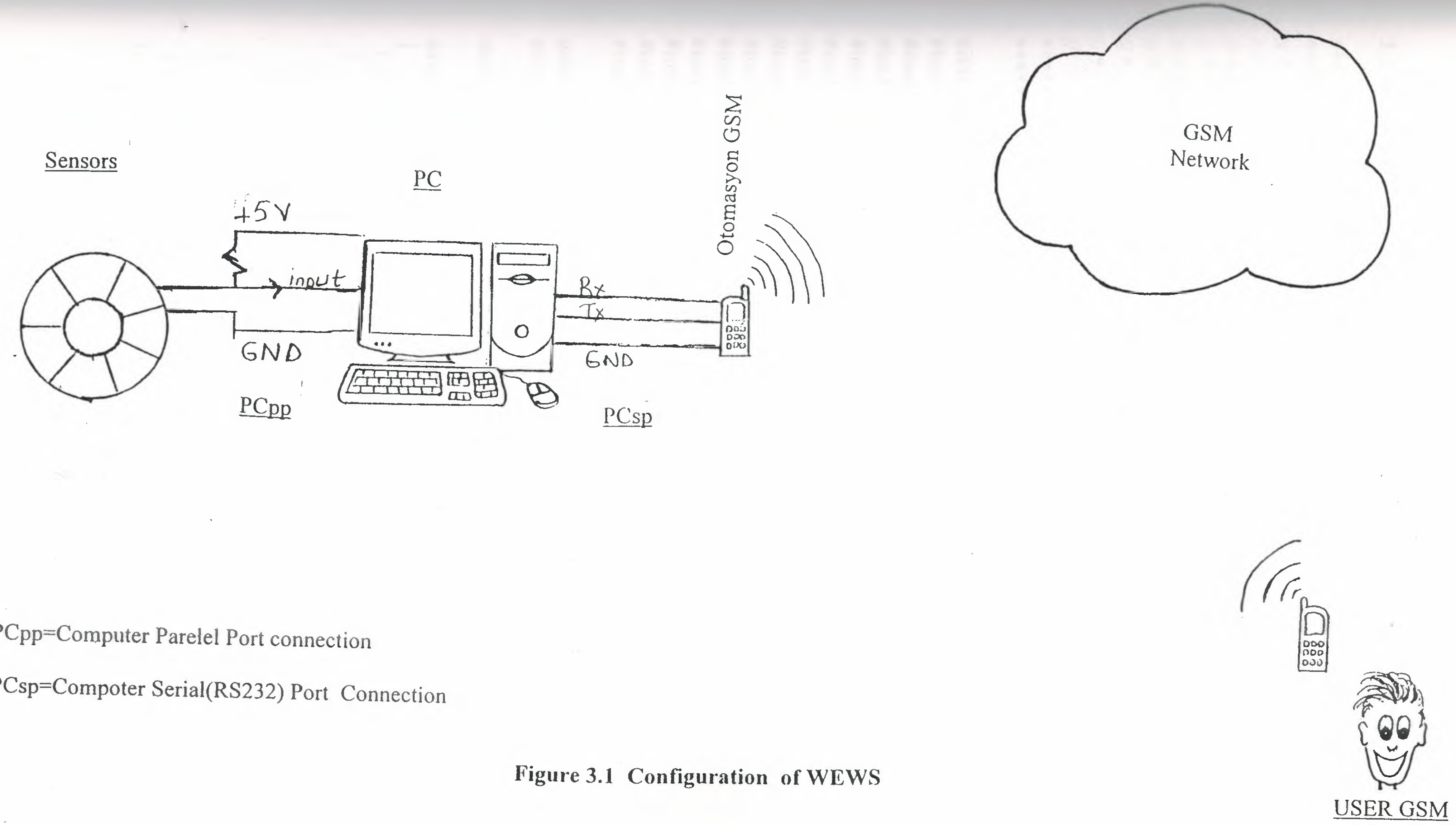
The DB25 is generally be a male connector on the back of the computer.

| DB25 | Signal Name |
|---|---|
| 8 | Data Carrier Detect (DCD) |
| 3 | Reive Data (RxD) |
| 2 | Transmite Data (TxD) |
| 20 | Data Terminal Ready (DTR) |
| 7 | Signal Ground (GND) |
| 6 | Data Set Ready (DSR) |
| 4 | Request to Sent (RTS) |
| 5 | Clear to Sent (CTS) |
| 22 | Ring Indicator (RI) |

Note:1, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 24 and 25 pins are not use in DB25.

# CHAPTER 3

# WIRELESS EMERGENCY WARNING SYTEMS DESIGN&IMPLEMENTATION



PCpp=Computer Parelel Port connection

PCsp=Compoter Serial(RS232) Port  Connection

Figure 3.1  Configuration  of WEWS

## 3.1 Serial port communication with gsm

```
*************************************************************************
*****
// * GSM TA/ME library
// *
// * File:    gsm_win32_port.cc
// *
// * Purpose: WIN32 serial port implementation
// *
// *
// *
//
*************************************************************************
*****

#ifdef HAVE_CONFIG_H
#include <gsm_config.h>
#endif
#include <winsock.h>
#include <gsmlib/gsm_nls.h>
#include <gsmlib/gsm_win32_serial.h>
#include <gsmlib/gsm_util.h>
#include <fcntl.h>
#include <iostream>
#include <strstream>
#include <errno.h>
#include <stdio.h>
#include <assert.h>
#include <signal.h>

using namespace std;
using namespace gsmlib;

static long int timeoutVal = TIMEOUT_SECS;

struct ExceptionSafeOverlapped: public OVERLAPPED
{
 ExceptionSafeOverlapped()
 {
  memset((OVERLAPPED*)this,0,sizeof(OVERLAPPED));
  hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
  if (hEvent == INVALID_HANDLE_VALUE)
   throw GsmException(_("error creating event"),OSError,GetLastError());
 }
 ~ExceptionSafeOverlapped()
 { CloseHandle(hEvent); }
};
```

```
typedef BOOL (WINAPI *TCancelIoProc)(HANDLE file);
TCancelIoProc CancelIoProc = NULL;
BOOL CancelIoHook(HANDLE file)
{
 if (CancelIoProc)
  return CancelIoProc(file);

 HMODULE hmodule = GetModuleHandle("KERNEL32");
 if (hmodule)
 {
  CancelIoProc = (TCancelIoProc)GetProcAddress(hmodule,"CancelIo");
  if (CancelIoProc)
   return CancelIoProc(file);
 }

 return TRUE;
}
#define CancelIo CancelIoHook

// Win32SerialPort members

void Win32SerialPort::throwModemException(string message)
throw(GsmException)
{
 ostrstream os;
 os << message << " (errno: " << errno << "/" << strerror(errno) << ")"
   << ends;
 char *ss = os.str();
 string s(ss);
 delete[] ss;
 throw GsmException(s, OSError, errno);
}

void Win32SerialPort::putBack(char c)
{
 assert(_oldChar == -1);
 _oldChar = c;
}

int Win32SerialPort::readByte() throw(GsmException)
{
 if (_oldChar != -1)
 {
  int result = _oldChar;
  _oldChar = -1;
  return result;
 }
```

```
char c;
int timeElapsed = 0;
bool readDone = true;
ExceptionSafeOverlapped  over;

DWORD initTime = GetTickCount();
DWORD dwReaded;
if (!ReadFile(_file,&c,1,&dwReaded,&over))
{
 readDone = false;
 if (GetLastError() != ERROR_IO_PENDING)
 {
  throwModemException(_("reading from TA"));
 }

 while(!readDone)
 {
  if (interrupted())
   throwModemException(_("interrupted when reading from TA"));

  // wait another second
  switch(WaitForSingleObject(over.hEvent,1000))
  {
  case WAIT_TIMEOUT:
   break;
  case WAIT_OBJECT_0:
  case WAIT_ABANDONED:
         // !!! do a infinite loop if (bytesWritten < lenght) ?
   GetOverlappedResult(_file,&over,&dwReaded,TRUE);
   readDone = true;
   break;
  case WAIT_FAILED:
   throwModemException(_("reading from TA"));
  }

  timeElapsed = (GetTickCount() - initTime)/1000U;

  // timeout elapsed ?
  if (timeElapsed >= timeoutVal)
  {
   CancelIo(_file);
   break;
  }

 }
}
```

```
  if (! readDone)
   throwModemException(_("timeout when reading from TA"));

#ifndef NDEBUG
  if (debugLevel() >= 2)
  {
   // some useful debugging code
   if (c == LF)
     cerr << "<LF>";
   else if (c == CR)
     cerr << "<CR>";
   else cerr << "<'" << (char) c << "'>";
   cerr.flush();
  }
#endif
  return c;
}

Win32SerialPort::Win32SerialPort(string device, int lineSpeed,
                               string initString, bool swHandshake)
  throw(GsmException) :
  _oldChar(-1)
{
  int holdoff[] = {2000, 1000, 400};

  // open device
  _file = CreateFile(device.c_str(),GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_OVERLAPPED, NULL );
  if (_file == INVALID_HANDLE_VALUE)
   throwModemException(stringPrintf(_("opening device '%s'"),
                                  device.c_str()));

  int initTries = 3;
  while (initTries-- > 0)
  {
   // flush all pending output
   FlushFileBuffers(_file);

   // toggle DTR to reset modem
   if (!EscapeCommFunction(_file,CLRDTR))
     throwModemException(_("clearing DTR failed"));
   Sleep(holdoff[initTries]);
   if (!EscapeCommFunction(_file,SETDTR))
     throwModemException(_("setting DTR failed"));

   DCB dcb;
   // get line modes
  if (!GetCommState(_file,&dcb))
    throwModemException(stringPrintf(_("GetCommState device '%s'"),
                                  device.c_str()));

//  if (tcgetattr(_fd, &t) < 0)
//    throwModemException(stringPrintf(_("tcgetattr device '%s'"),
//                                  device.c_str()));

  // set the device to a sane state
  dcb.fBinary = TRUE;
  dcb.BaudRate = lineSpeed;

  // n,8,1
  dcb.fParity = FALSE;
  dcb.Parity = 0;
  dcb.ByteSize = 8;
  dcb.StopBits = 0;

  if (!swHandshake)
  {
    dcb.fInX = FALSE;
    dcb.fOutX = FALSE;
    dcb.fOutxDsrFlow = FALSE;
    dcb.fOutxCtsFlow = FALSE;
  }
  else
  {
    dcb.fInX  = TRUE;
    dcb.fOutX = TRUE;
    dcb.fOutxDsrFlow = FALSE;
    dcb.fOutxCtsFlow = FALSE;
  }
  dcb.fDtrControl = DTR_CONTROL_ENABLE;
  dcb.fRtsControl = RTS_CONTROL_ENABLE;

//  t.c_iflag |= IGNPAR;
//  t.c_iflag &= ~(INPCK | ISTRIP | IMAXBEL |
//             (swHandshake ? CRTSCTS : IXON |  IXOFF)
//             | IXANY | IGNCR | ICRNL | IMAXBEL | INLCR | IGNBRK);
//  t.c_oflag &= ~(OPOST);
//  // be careful, only touch "known" flags
//  t.c_cflag&= ~(CSIZE | CSTOPB | PARENB | PARODD);
//  t.c_cflag|= CS8 | CREAD | HUPCL |
//    (swHandshake ? IXON |  IXOFF : CRTSCTS) |
//    CLOCAL;
//  t.c_lflag &= ~(ECHO | ECHOE | ECHOPRT | ECHOK | ECHOKE | ECHONL |
//             ECHOCTL | ISIG | IEXTEN | TOSTOP | FLUSHO | ICANON);
//  t.c_lflag |= NOFLSH;
```

```
//
//   t.c_cc[VMIN] = 1;
//   t.c_cc[VTIME] = 0;
//
//   t.c_cc[VSUSP] = 0;

  // write back
  if (!SetCommState(_file,&dcb))
   throwModemException(stringPrintf(_("SetCommState device '%s'"),
                              device.c_str()));

  Sleep(holdoff[initTries]);

  if (!SetupComm(_file,1024,1024))
   throwModemException(stringPrintf(_("SetupComm device '%s'"),
                              device.c_str()));


  // flush all pending input
  PurgeComm(_file,PURGE_RXABORT|PURGE_RXCLEAR);

  try
  {
   // reset modem
   putLine("ATZ");
   bool foundOK = false;
   int readTries = 5;
   while (readTries-- > 0)
   {
    string s = getLine();
    if (s.find("OK") != string::npos ||
       s.find("CABLE: GSM") != string::npos)
    {
     foundOK = true;
     readTries = 0;          // found OK, exit loop
    }
   }

   if (foundOK)
   {
    // init modem
    readTries = 5;
    // !!! no not declare this in loop, compiler error on Visual C++
    // (without SP and with SP4)
    string s;
    putLine("AT" + initString);
    do
    {
        s = getLine();
        if (s.find("OK") != string::npos ||
            s.find("CABLE: GSM") != string::npos)
          return;                // found OK, return
      } while(--readTries);
    }
   }
   catch (GsmException &e)
   {
    if (initTries == 0)
      throw e;
   }
  }
  // no response after 3 tries
  throw GsmException(stringPrintf(_("reset modem failed '%s'"),
                                  device.c_str()), OtherError);
}

string Win32SerialPort::getLine() throw(GsmException)
{
  string result;
  int c;
  while ((c = readByte()) > 0)
  {
    while (c == CR)
    {
      c = readByte();
    }
    if (c == LF)
      break;
    result += c;
  }

#ifndef NDEBUG
  if (debugLevel() >= 1)
    cerr << "<-- " << result << endl;
#endif

  return result;
}

void Win32SerialPort::putLine(string line,
                              bool carriageReturn) throw(GsmException)
{
#ifndef NDEBUG
  if (debugLevel() >= 1)
    cerr << "--> " << line << endl;
#endif
```

```
if (carriageReturn) line += CR;
// !!! BUG, mantain this pointer isn't corrent, use iterator !!!
const char *l = line.c_str();

FlushFileBuffers(_file);      // flush all pending input and output

int timeElapsed = 0;

DWORD bytesWritten = 0;

ExceptionSafeOverlapped over;

DWORD initTime = GetTickCount();
if (!WriteFile(_file,l,line.length(),&bytesWritten,&over))
{
 if (GetLastError() != ERROR_IO_PENDING)
 {
  throwModemException(_("writing to TA"));
 }

 while(bytesWritten < (DWORD)line.length())
 {
  if (interrupted())
   throwModemException(_("interrupted when writing to TA"));

  // wait another second
  switch(WaitForSingleObject(over.hEvent,1000))
  {
  case WAIT_TIMEOUT:
   break;
  case WAIT_OBJECT_0:
  case WAIT_ABANDONED:
      // !!! do a infinite loop if (bytesWritten < lenght) ?
   GetOverlappedResult(_file,&over,&bytesWritten,TRUE);
   break;
  case WAIT_FAILED:
   throwModemException(_("writing to TA"));
  }

  timeElapsed = (GetTickCount() - initTime)/1000U;

  // timeout elapsed ?
  if (timeElapsed >= timeoutVal)
  {
   CancelIo(_file);
   throwModemException(_("timeout when writing to TA"));
  }
  }
 }

 return;
/*
 // empty buffer
 SetCommMask(_file,EV_TXEMPTY);
 DWORD dwEvent;
 ResetEvent(over.hEvent);
 if( WaitCommEvent(_file,&dwEvent,&over) )
   return; // already empty

 // check true errors
 if (GetLastError() != ERROR_IO_PENDING)
   throwModemException(_("error comm waiting"));

 while(timeElapsed < timeoutVal)
 {
   if (interrupted())
     throwModemException(_("interrupted when flushing to TA"));

   switch( WaitForSingleObject( over.hEvent, 1000 ) )
   {
   case WAIT_TIMEOUT:
     break;

   // successfully flushed
   case WAIT_ABANDONED:
   case WAIT_OBJECT_0:
     return;

   default:
     throwModemException(_("error waiting"));
   }
   timeElapsed = (GetTickCount() - initTime)/1000U;
 }

 CancelIo(_file);
 throwModemException(_("timeout when writing to TA"));
*/

 // echo CR LF must be removed by higher layer functions in gsm_at because
 // in order to properly handle unsolicited result codes from the ME/TA
}

bool Win32SerialPort::wait(GsmTime timeout) throw(GsmException)
{
  // See differences from UNIX
  // Why do I use Windows ?
  ExceptionSafeOverlapped over;

  SetCommMask(_file,EV_RXCHAR);
  DWORD dwEvent;
  if( !WaitCommEvent(_file,&dwEvent,&over) )
  {
    // check true errors
    if (GetLastError() != ERROR_IO_PENDING)
      throwModemException(_("error comm waiting"));

    switch( WaitForSingleObject( over.hEvent, timeout->tv_sec*1000U+timeout-
>tv_usec ) )
    {
    case WAIT_TIMEOUT:
      CancelIo(_file);
      return false;

    case WAIT_ABANDONED:
    case WAIT_OBJECT_0:
      return true;

    default:
      throwModemException(_("error waiting"));
    }
  }

  return true;
}

void Win32SerialPort::setTimeOut(unsigned int timeout)
{
  timeoutVal = timeout;
}

Win32SerialPort::~Win32SerialPort()
{
  if ( _file != INVALID_HANDLE_VALUE)
    CloseHandle(_file);
}

int gsmlib::baudRateStrToSpeed(string baudrate) throw(GsmException)
{
  if (baudrate == "300")
    return 300;
  else if (baudrate == "600")
    return 600;
```

```
 else if (baudrate == "1200")
   return 1200;
 else if (baudrate == "2400")
   return 2400;
 else if (baudrate == "4800")
   return 4800;
 else if (baudrate == "9600")
   return 9600;
 else if (baudrate == "19200")
   return 19200;
 else if (baudrate == "38400")
   return 38400;
 else if (baudrate == "57600")
   return 57600;
 else if (baudrate == "115200")
   return 115200;
 else
   throw GsmException(stringPrintf(_("unknown baudrate '%s'"),
                                   baudrate.c_str()), ParameterError);
}
```

## 3.2 Send SMS code

```
// c:\gsm\gsmsmsd -r -b 38400 -d :COM1 –spooll //
**************************************************************************
*****
// * GSM TA/ME library
// *
// * File:    gsm_sms.cc
// *
// * Purpose: SMS functions
// *          (ETSI GSM 07.05)
// *
// *
// *
**************************************************************************
*****

#ifdef HAVE_CONFIG_H
#include <gsm_config.h>
#endif
#include <gsmlib/gsm_nls.h>
#include <gsmlib/gsm_sysdep.h>
#include <gsmlib/gsm_sms.h>
#include <gsmlib/gsm_util.h>
#include <gsmlib/gsm_parser.h>
#include <gsmlib/gsm_me_ta.h>
#include <strstream>
#include <string>

using namespace std;
using namespace gsmlib;

// local constants

static const string dashes =
"---------------------------------------------------------------------------";

// SMSMessage members

Ref<SMSMessage> SMSMessage::decode(string pdu,
                                    bool SCtoMEdirection,
                                    GsmAt *at) throw(GsmException)
{
  Ref<SMSMessage> result;
  SMSDecoder d(pdu);
  d.getAddress(true);
  MessageType messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
  if (SCtoMEdirection)
  // TPDUs from SC to ME
  switch (messageTypeIndicator)
  {
  case SMS_DELIVER:
    result = new SMSDeliverMessage(pdu);
    break;

  case SMS_STATUS_REPORT:
    result = new SMSStatusReportMessage(pdu);
    break;

  case SMS_SUBMIT_REPORT:
    // observed with Motorola Timeport 260, the SCtoMEdirection can
    // be wrong in this case
    if (at != NULL && at->getMeTa().getCapabilities()._wrongSMSStatusCode)
      result = new SMSSubmitMessage(pdu);
    else
      result = new SMSSubmitReportMessage(pdu);
    break;

  default:
    throw GsmException(_("unhandled SMS TPDU type"), OtherError);
  }
 else
  // TPDUs from ME to SC
  switch (messageTypeIndicator)
  {
  case SMS_SUBMIT:
    result = new SMSSubmitMessage(pdu);
    break;

  case SMS_DELIVER_REPORT:
    result = new SMSDeliverReportMessage(pdu);
    break;

  case SMS_COMMAND:
    result = new SMSCommandMessage(pdu);
    break;

  default:
    throw GsmException(_("unhandled SMS TPDU type"), OtherError);
  }
 result->_at = at;
 return result;
}

Ref<SMSMessage> SMSMessage::decode(istream& s)
throw(gsmlib::GsmException)
```

```
{
 string pdu;
 unsigned char ScToMe;

 s >> ScToMe;
 s >> pdu;

 return decode(pdu,ScToMe=='S');
}

unsigned char SMSMessage::send(Ref<SMSMessage> &ackPdu)
 throw(GsmException)
{
 if (_messageTypeIndicator != SMS_SUBMIT &&
     _messageTypeIndicator != SMS_COMMAND)
   throw GsmException(_("can only send SMS-SUBMIT and SMS-COMMAND
TPDUs"),
                      ParameterError);

 if (_at.isnull())
   throw GsmException(_("no device given for sending SMS"), ParameterError);

 string pdu = encode();
 Parser p(_at->sendPdu("+CMGS=" +
                      intToStr(pdu.length() / 2 - getSCAddressLen()),
                      "+CMGS:", pdu));
 unsigned char messageReference = p.parseInt();

 if (p.parseComma(true))
 {
   string pdu = p.parseEol();

   // add missing service centre address if required by ME
   if (! _at->getMeTa().getCapabilities()._hasSMSSCAprefix)
     pdu = "00" + pdu;

   ackPdu = SMSMessage::decode(pdu);
 }
 else
   ackPdu = SMSMessageRef();

 return messageReference;
}

unsigned char SMSMessage::send() throw(GsmException)
{
 SMSMessageRef mref;
 return send(mref);
}

unsigned int SMSMessage::getSCAddressLen()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 return e.getLength();
}

unsigned char SMSMessage::userDataLength() const
{
 unsigned int udhl = _userDataHeader.length();
 if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
  return _userData.length() + (udhl ? ((1 + udhl) * 8 + 6) / 7 : 0);
 else
  return _userData.length() + (udhl ? (1 + udhl) : 0);
}

ostream& SMSMessage::operator<<(ostream& s)
{
 unsigned char ScToMe;

 if (dynamic_cast<SMSDeliverMessage*>(this) ||
   dynamic_cast<SMSStatusReportMessage*>(this) ||
   dynamic_cast<SMSSubmitReportMessage*>(this))
 {
  ScToMe = 'S';
 }
 else if (dynamic_cast<SMSSubmitMessage*>(this) ||
      dynamic_cast<SMSCommandMessage*>(this) ||
      dynamic_cast<SMSDeliverReportMessage*>(this))
  {
   ScToMe = 'M';
  }
  else
  {
   throw GsmException(_("unhandled SMS TPDU type"), OtherError);
  }

  s << ScToMe;
  return s << encode();
}

// SMSMessage::SMSMessage(SMSMessage &m)
// {
//   _at = m._at;

// }
// SMSMessage &SMSMessage::operator=(SMSMessage &m)
// {
// }

SMSMessage::~SMSMessage() {}

// SMSDeliverMessage members

void SMSDeliverMessage::init()
{
 _messageTypeIndicator = SMS_DELIVER;
 _moreMessagesToSend = false;
 _replyPath = false;
 _statusReportIndication = false;
 _protocolIdentifier = 0;
}

SMSDeliverMessage::SMSDeliverMessage()
{
 init();
}

SMSDeliverMessage::SMSDeliverMessage(string pdu) throw(GsmException)
{
 SMSDecoder d(pdu);
 _serviceCentreAddress = d.getAddress(true);
 _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
 assert(_messageTypeIndicator == SMS_DELIVER);
 _moreMessagesToSend = d.getBit(); // bit 2
 d.getBit();              // bit 3
 d.getBit();              // bit 4
 _statusReportIndication = d.getBit(); // bit 5
 bool userDataHeaderIndicator = d.getBit(); // bit 6
 _replyPath = d.getBit();     // bit 7
 _originatingAddress = d.getAddress();
 _protocolIdentifier = d.getOctet();
 _dataCodingScheme = d.getOctet();
 _serviceCentreTimestamp = d.getTimestamp();
 unsigned char userDataLength = d.getOctet();
 d.markSeptet();

 if (userDataHeaderIndicator)
 {
  _userDataHeader.decode(d);
  if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
   userDataLength -= ((_userDataHeader.length() + 1) * 8 + 6) / 7;
  else
```

```
    userDataLength -= ((string)_userDataHeader).length() + 1;
  }
  else
   _userDataHeader = UserDataHeader();

  if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
  {                        // userDataLength is length in septets
   _userData = d.getString(userDataLength);
   _userData = gsmToLatin1(_userData);
  }
  else
  {                        // userDataLength is length in octets
   unsigned char *s =
    (unsigned char*)alloca(sizeof(unsigned char) * userDataLength);
   d.getOctets(s, userDataLength);
   _userData.assign((char*)s, (unsigned int)userDataLength);
  }
}

string SMSDeliverMessage::encode()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 e.set2Bits(_messageTypeIndicator); // bits 0..1
 e.setBit(_moreMessagesToSend); // bit 2
 e.setBit();               // bit 3
 e.setBit();               // bit 4
 e.setBit(_statusReportIndication); // bit 5
 e.setBit(_userDataHeader.length() != 0); // bit 6
 e.setBit(_replyPath);        // bit 7
 e.setAddress(_originatingAddress);
 e.setOctet(_protocolIdentifier);
 e.setOctet(_dataCodingScheme);
 e.setTimestamp(_serviceCentreTimestamp);
 e.setOctet(userDataLength());
 e.markSeptet();
 if (_userDataHeader.length()) _userDataHeader.encode(e);
 if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
   e.setString(latin1ToGsm(_userData));
 else
   e.setOctets((unsigned char*)_userData.data(), _userData.length());
 return e.getHexString();
}

string SMSDeliverMessage::toString() const
{
 ostrstream os;
 os << dashes << endl
```

```
    << _("Message type: SMS-DELIVER") << endl
    << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
    << _("More messages to send: ") << _moreMessagesToSend << endl
    << _("Reply path: ") << _replyPath << endl
    << _("User data header indicator: ")
    << (_userDataHeader.length()!=0) << endl
    << _("Status report indication: ") << _statusReportIndication << endl
    << _("Originating address: '") << _originatingAddress._number
    << "'" << endl
    << _("Protocol identifier: 0x") << hex
    << (unsigned int)_protocolIdentifier << dec << endl
    << _("Data coding scheme: ") << _dataCodingScheme.toString() << endl
    << _("SC timestamp: ") << _serviceCentreTimestamp.toString() << endl
    << _("User data length: ") << (int)userDataLength() << endl
    << _("User data header: 0x")
    << bufToHex((unsigned char*)
            ((string)_userDataHeader).data(),
            ((string)_userDataHeader).length())
    << endl
    << _("User data: '") << _userData << "'" << endl
    << dashes << endl << endl
    << ends;
  char *ss = os.str();
  string result(ss);
  delete[] ss;
  return result;
}

Address SMSDeliverMessage::address() const
{
  return _originatingAddress;
}

Ref<SMSMessage> SMSDeliverMessage::clone()
{
  Ref<SMSMessage> result = new SMSDeliverMessage(*this);
  return result;
}

// SMSSubmitMessage members

void SMSSubmitMessage::init()
{
  // set everything to sensible default values
  _messageTypeIndicator = SMS_SUBMIT;
  _validityPeriodFormat = TimePeriod::Relative;
  _validityPeriod._format = TimePeriod::Relative;
  _validityPeriod._relativeTime = 168; // 2 days
```

```
 _statusReportRequest = false;
 _replyPath = false;
 _rejectDuplicates = true;
 _messageReference = 0;
 _protocolIdentifier = 0;
}

SMSSubmitMessage::SMSSubmitMessage()
{
 init();
}

SMSSubmitMessage::SMSSubmitMessage(string pdu) throw(GsmException)
{
 SMSDecoder d(pdu);
 _serviceCentreAddress = d.getAddress(true);
 _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
 assert(_messageTypeIndicator == SMS_SUBMIT);
 _rejectDuplicates = d.getBit(); // bit 2
 _validityPeriodFormat = (TimePeriod::Format)d.get2Bits(); // bits 3..4
 _statusReportRequest = d.getBit(); // bit 5
 bool userDataHeaderIndicator = d.getBit(); // bit 6
 _replyPath = d.getBit();      // bit 7
 _messageReference = d.getOctet();
 _destinationAddress = d.getAddress();
 _protocolIdentifier = d.getOctet();
 _dataCodingScheme = d.getOctet();
 if (_validityPeriodFormat != TimePeriod::NotPresent)
  _validityPeriod = d.getTimePeriod(_validityPeriodFormat);
 unsigned char userDataLength = d.getOctet();
 d.markSeptet();

 if (userDataHeaderIndicator)
 {
  _userDataHeader.decode(d);
  if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
   userDataLength -= ((_userDataHeader.length() + 1) * 8 + 6) / 7;
  else
   userDataLength -= ((string)_userDataHeader).length() + 1;
 }
 else
  _userDataHeader = UserDataHeader();

 if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
 {                    // userDataLength is length in septets
  _userData = d.getString(userDataLength);
  _userData = gsmToLatin1(_userData);
 }
 else
 {                          // _userDataLength is length in octets
   unsigned char *s =
     (unsigned char*)alloca(sizeof(unsigned char) * userDataLength);
   d.getOctets(s, userDataLength);
   _userData.assign((char*)s, userDataLength);
 }
}

SMSSubmitMessage::SMSSubmitMessage(string text, string number)
{
 init();
 _destinationAddress = Address(number);
 _userData = text;
}

string SMSSubmitMessage::encode()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 e.set2Bits(_messageTypeIndicator); // bits 0..1
 e.setBit(_rejectDuplicates); // bit 2
 e.set2Bits(_validityPeriodFormat); // bits 3..4
 e.setBit(_statusReportRequest); // bit 5
 bool userDataHeaderIndicator = _userDataHeader.length() != 0;
 e.setBit(userDataHeaderIndicator); // bit 6
 e.setBit(_replyPath);       // bit 7
 e.setOctet(_messageReference);
 e.setAddress(_destinationAddress);
 e.setOctet(_protocolIdentifier);
 e.setOctet(_dataCodingScheme);
 e.setTimePeriod(_validityPeriod);
 e.setOctet(userDataLength());
 e.markSeptet();
 if (userDataHeaderIndicator) _userDataHeader.encode(e);
 if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
   e.setString(latin1ToGsm(_userData));
 else
   e.setOctets((unsigned char*)_userData.data(), _userData.length());
 return e.getHexString();
}

string SMSSubmitMessage::toString() const
{
 ostrstream os;
 os << dashes << endl
   << _("Message type: SMS-SUBMIT") << endl
   << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
   << _("Reject duplicates: ") << _rejectDuplicates << endl
   << _("Validity period format: ");
 switch (_validityPeriodFormat)
 {
 case TimePeriod::NotPresent:
   os << _("not present");
   break;
 case TimePeriod::Relative:
   os << _("relative");
   break;
 case TimePeriod::Absolute:
   os << _("absolute");
   break;
 default:
   os << _("unknown");
   break;
 }
 os << endl
   << _("Reply path: ") << _replyPath << endl
   << _("User data header indicator: ")
   << (_userDataHeader.length()!=0) << endl
   << _("Status report request: ") << _statusReportRequest << endl
   << _("Message reference: ") << (unsigned int)_messageReference << endl
   << _("Destination address: '") << _destinationAddress._number
   << "'" << endl
   << _("Protocol identifier: 0x") << hex
   << (unsigned int)_protocolIdentifier << dec << endl
   << _("Data coding scheme: ") << _dataCodingScheme.toString() << endl
   << _("Validity period: ") << _validityPeriod.toString() << endl
   << _("User data length: ") << (int)userDataLength() << endl
   << _("User data header: 0x") << bufToHex((unsigned char*)
                                   ((string)_userDataHeader).data(),
                                   _userDataHeader.length())
   << endl
   << _("User data: '") << _userData << "'" << endl
   << dashes << endl << endl
   << ends;
 char *ss = os.str();
 string result(ss);
 delete[] ss;
 return result;
}

Address SMSSubmitMessage::address() const
{
 return _destinationAddress;
}
```

```
Ref<SMSMessage> SMSSubmitMessage::clone()
{
 Ref<SMSMessage> result = new SMSSubmitMessage(*this);
 return result;
}

// SMSStatusReportMessage members

void SMSStatusReportMessage::init()
{
 _messageTypeIndicator = SMS_STATUS_REPORT;
 _moreMessagesToSend = false;
 _statusReportQualifier = false;
 _messageReference = 0;
 _status = SMS_STATUS_RECEIVED;
}

SMSStatusReportMessage::SMSStatusReportMessage(string pdu)
throw(GsmException)
{
 SMSDecoder d(pdu);
 _serviceCentreAddress = d.getAddress(true);
 _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
 assert(_messageTypeIndicator == SMS_STATUS_REPORT);
 _moreMessagesToSend = d.getBit(); // bit 2
 d.getBit();              // bit 3
 d.getBit();              // bit 4
 _statusReportQualifier = d.getBit(); // bit 5
 _messageReference = d.getOctet();
 _recipientAddress = d.getAddress();
 _serviceCentreTimestamp = d.getTimestamp();
 _dischargeTime = d.getTimestamp();
 _status = d.getOctet();
}

string SMSStatusReportMessage::encode()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 e.set2Bits(_messageTypeIndicator); // bits 0..1
 e.setBit(_moreMessagesToSend); // bit 2
 e.setBit();              // bit 3
 e.setBit();              // bit 4
 e.setBit(_statusReportQualifier); // bit 5
 e.setOctet(_messageReference);
 e.setAddress(_recipientAddress);
 e.setTimestamp(_serviceCentreTimestamp);
 e.setTimestamp(_dischargeTime);
```

```
  e.setOctet(_status);
  return e.getHexString();
}

string SMSStatusReportMessage::toString() const
{
  ostrstream os;
  os << dashes << endl
    << _("Message type: SMS-STATUS-REPORT") << endl
    << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
    << _("More messages to send: ") << _moreMessagesToSend << endl
    << _("Status report qualifier: ") << _statusReportQualifier << endl
    << _("Message reference: ") << (unsigned int)_messageReference << endl
    << _("Recipient address: '") << _recipientAddress._number << "'" << endl
    << _("SC timestamp: ") << _serviceCentreTimestamp.toString() << endl
    << _("Discharge time: ") << _dischargeTime.toString() << endl
    << _("Status: 0x") << hex << (unsigned int)_status << dec
    << " '" << getSMSStatusString(_status) << "'" << endl
    << dashes << endl << endl
    << ends;
  char *ss = os.str();
  string result(ss);
  delete[] ss;
  return result;
}

Address SMSStatusReportMessage::address() const
{
  return _recipientAddress;
}

Ref<SMSMessage> SMSStatusReportMessage::clone()
{
  Ref<SMSMessage> result = new SMSStatusReportMessage(*this);
  return result;
}

// SMSCommandMessage members

void SMSCommandMessage::init()
{
  _messageTypeIndicator = SMS_COMMAND;
  _messageReference = 0;
  _statusReportRequest = true;
  _protocolIdentifier = 0;
  _commandType = EnquireSM;
  _messageNumber = 0;
  _commandDataLength = 0;
```

```
}

SMSCommandMessage::SMSCommandMessage(string pdu) throw(GsmException)
{
 SMSDecoder d(pdu);
 _serviceCentreAddress = d.getAddress(true);
 _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
 assert(_messageTypeIndicator == SMS_COMMAND);
 d.getBit();              // bit 2
 d.getBit();              // bit 3
 d.getBit();              // bit 4
 _statusReportRequest = d.getBit(); // bit 5
 _messageReference = d.getOctet();
 _protocolIdentifier = d.getOctet();
 _commandType = d.getOctet();
 _messageNumber = d.getOctet();
 _destinationAddress = d.getAddress();
 _commandDataLength = d.getOctet();
 unsigned char *s =
    (unsigned char*)alloca(sizeof(unsigned char) * _commandDataLength);
 d.getOctets(s, _commandDataLength);
}

string SMSCommandMessage::encode()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 e.set2Bits(_messageTypeIndicator); // bits 0..1
 e.setBit();              // bit 2
 e.setBit();              // bit 3
 e.setBit();              // bit 4
 e.setBit(_statusReportRequest); // bit 5
 e.setOctet(_messageReference);
 e.setOctet(_protocolIdentifier);
 e.setOctet(_commandType);
 e.setOctet(_messageNumber);
 e.setAddress(_destinationAddress);
 e.setOctet(_commandData.length());
 e.setOctets((const unsigned char*)_commandData.data(),
          (short unsigned int)_commandData.length());
 return e.getHexString();
}

string SMSCommandMessage::toString() const
{
 ostrstream os;
 os << dashes << endl
   << _("Message type: SMS-COMMAND") << endl
```

```
    << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
    << _("Message reference: ") << (unsigned int)_messageReference << endl
    << _("Status report request: ") << _statusReportRequest << endl
    << _("Protocol identifier: 0x") << hex
    << (unsigned int)_protocolIdentifier << dec << endl
    << _("Command type: 0x") << hex << (unsigned int)_commandType
    << dec << endl
    << _("Message number: ") << (unsigned int)_messageNumber << endl
    << _("Destination address: '") << _destinationAddress._number
    << "'" << endl
    << _("Command data length: ") << (unsigned int)_commandDataLength << endl
    << _("Command data: '") << _commandData << "'" << endl
    << dashes << endl << endl
    << ends;
  char *ss = os.str();
  string result(ss);
  delete[] ss;
  return result;
}

Address SMSCommandMessage::address() const
{
  return _destinationAddress;
}

Ref<SMSMessage> SMSCommandMessage::clone()
{
  Ref<SMSMessage> result = new SMSCommandMessage(*this);
  return result;
}

// SMSDeliverReportMessage members

void SMSDeliverReportMessage::init()
{
  _messageTypeIndicator = SMS_DELIVER_REPORT;
  _protocolIdentifierPresent = false;
  _dataCodingSchemePresent = false;
  _userDataLengthPresent = false;
}

SMSDeliverReportMessage::SMSDeliverReportMessage(string pdu)
  throw(GsmException)
{
  SMSDecoder d(pdu);
  _serviceCentreAddress = d.getAddress(true);
  _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
  assert(_messageTypeIndicator == SMS_DELIVER_REPORT);
  d.alignOctet();              // skip to parameter indicator
  _protocolIdentifierPresent = d.getBit(); // bit 0
  _dataCodingSchemePresent = d.getBit(); // bit 1
  _userDataLengthPresent = d.getBit(); // bit 2
  if (_protocolIdentifierPresent)
    _protocolIdentifier = d.getOctet();
  if (_dataCodingSchemePresent)
    _dataCodingScheme = d.getOctet();
  if (_userDataLengthPresent)
  {
    unsigned char userDataLength = d.getOctet();
    d.markSeptet();
    if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
    {
      _userData = d.getString(userDataLength);
      _userData = gsmToLatin1(_userData);
    }
    else
    {                          // userDataLength is length in octets
      unsigned char *s =
        (unsigned char*)alloca(sizeof(unsigned char) * userDataLength);
      d.getOctets(s, userDataLength);
      _userData.assign((char*)s, userDataLength);
    }
  }
}

string SMSDeliverReportMessage::encode()
{
  SMSEncoder e;
  e.setAddress(_serviceCentreAddress, true);
  e.set2Bits(_messageTypeIndicator); // bits 0..1
  e.alignOctet();              // skip to parameter indicator
  e.setBit(_protocolIdentifierPresent); // bit 0
  e.setBit(_dataCodingSchemePresent); // bit 1
  e.setBit(_userDataLengthPresent); // bit 2
  if (_protocolIdentifierPresent)
    e.setOctet(_protocolIdentifier);
  if (_dataCodingSchemePresent)
    e.setOctet(_dataCodingScheme);
  if (_userDataLengthPresent)
  {
    unsigned char userDataLength = _userData.length();
    e.setOctet(userDataLength);
    if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
      e.setString(latin1ToGsm(_userData));
    else
      e.setOctets((unsigned char*)_userData.data(), userDataLength);
  }
  return e.getHexString();
}

string SMSDeliverReportMessage::toString() const
{
  ostrstream os;
  os << dashes << endl
     << _("Message type: SMS-DELIVER-REPORT") << endl
     << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
     << _("Protocol identifier present: ") << _protocolIdentifierPresent
     << endl
     << _("Data coding scheme present: ") << _dataCodingSchemePresent << endl
     << _("User data length present: ") << _userDataLengthPresent << endl;
  if (_protocolIdentifierPresent)
    os << _("Protocol identifier: 0x") << hex
       << (unsigned int)_protocolIdentifier
       << dec << endl;
  if (_dataCodingSchemePresent)
    os << _("Data coding scheme: ") << _dataCodingScheme.toString() << endl;
  if (_userDataLengthPresent)
    os << _("User data length: ") << (int)userDataLength() << endl
       << _("User data: '") << _userData << "'" << endl;
  os << dashes << endl << endl
     << ends;
  char *ss = os.str();
  string result(ss);
  delete[] ss;
  return result;
}

Address SMSDeliverReportMessage::address() const
{
  assert(0);                    // not address, should not be in SMS store
  return Address();
}

Ref<SMSMessage> SMSDeliverReportMessage::clone()
{
  Ref<SMSMessage> result = new SMSDeliverReportMessage(*this);
  return result;
}

// SMSSubmitReportMessage members

void SMSSubmitReportMessage::init()
{
  _messageTypeIndicator = SMS_SUBMIT_REPORT;
```

```
 _protocolIdentifierPresent = false;
 _dataCodingSchemePresent = false;
 _userDataLengthPresent = false;
}

SMSSubmitReportMessage::SMSSubmitReportMessage(string pdu)
throw(GsmException)
{
 SMSDecoder d(pdu);
 _serviceCentreAddress = d.getAddress(true);
 _messageTypeIndicator = (MessageType)d.get2Bits(); // bits 0..1
 assert(_messageTypeIndicator == SMS_SUBMIT_REPORT);
 _serviceCentreTimestamp = d.getTimestamp();
 _protocolIdentifierPresent = d.getBit(); // bit 0
 _dataCodingSchemePresent = d.getBit(); // bit 1
 _userDataLengthPresent = d.getBit(); // bit 2
 if (_protocolIdentifierPresent)
   _protocolIdentifier = d.getOctet();
 if (_dataCodingSchemePresent)
   _dataCodingScheme = d.getOctet();
 if (_userDataLengthPresent)
 {
  unsigned char userDataLength = d.getOctet();
  d.markSeptet();
  if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
  {
   _userData = d.getString(userDataLength);
   _userData = gsmToLatin1(_userData);
  }
  else
  {                     // _userDataLength is length in octets
   unsigned char *s =
     (unsigned char*)alloca(sizeof(unsigned char) * userDataLength);
   d.getOctets(s, userDataLength);
   _userData.assign((char*)s, userDataLength);
  }
 }
}

string SMSSubmitReportMessage::encode()
{
 SMSEncoder e;
 e.setAddress(_serviceCentreAddress, true);
 e.set2Bits(_messageTypeIndicator); // bits 0..1
 e.setTimestamp(_serviceCentreTimestamp);
 e.setBit(_protocolIdentifierPresent); // bit 0
 e.setBit(_dataCodingSchemePresent); // bit 1
 e.setBit(_userDataLengthPresent); // bit 2
```

```
  if (_protocolIdentifierPresent)
    e.setOctet(_protocolIdentifier);
  if (_dataCodingSchemePresent)
    e.setOctet(_dataCodingScheme);
  if (_userDataLengthPresent)
  {
    e.setOctet(userDataLength());
    if (_dataCodingScheme.getAlphabet() == DCS_DEFAULT_ALPHABET)
      e.setString(latin1ToGsm(_userData));
    else
      e.setOctets((unsigned char*)_userData.data(), _userData.length());
  }
  return e.getHexString();
}

string SMSSubmitReportMessage::toString() const
{
  ostrstream os;
  os << dashes << endl
     << _("Message type: SMS-SUBMIT-REPORT") << endl
     << _("SC address: '") << _serviceCentreAddress._number << "'" << endl
     << _("SC timestamp: ") << _serviceCentreTimestamp.toString() << endl
     << _("Protocol identifier present: ") << _protocolIdentifierPresent
     << endl
     << _("Data coding scheme present: ") << _dataCodingSchemePresent << endl
     << _("User data length present: ") << _userDataLengthPresent << endl;
  if (_protocolIdentifierPresent)
    os << _("Protocol identifier: 0x") << hex
       << (unsigned int)_protocolIdentifier
       << dec << endl;
  if (_dataCodingSchemePresent)
    os << _("Data coding scheme: ") << _dataCodingScheme.toString() << endl;
  if (_userDataLengthPresent)
    os << _("User data length: ") << (int)userDataLength() << endl
       << _("User data: '") << _userData << "'" << endl;
  os << dashes << endl << endl
     << ends;
  char *ss = os.str();
  string result(ss);
  delete[] ss;
  return result;
}

Address SMSSubmitReportMessage::address() const
{
  assert(0);                // not address, should not be in SMS store
  return Address();
}
```

```
Ref<SMSMessage> SMSSubmitReportMessage::clone()
{
 Ref<SMSMessage> result = new SMSSubmitReportMessage(*this);
 return result;
}
```

## 3.3 Phone Numbers Text File

```
c:\gsm\msg.txt -f --store sm --action php
Fire numbers
+905325405555 [Mr. Adams]
"Emergency Message: House on fire. Get immediately home."
+90532110 [Fire Department]
"Emergency Message: House on fire. Address: Çetin Emeç Blv. No:144/2 Dikmen,
Ankara"

Thief Numbers

+905325405555 [Mr. Adams]
"Emergency Message: Thief. Get immediately home."
+90532111 [Police]
"Emergency Message: Thief. Address: Çetin Emeç Blv. No:144/2 Dikmen, Ankara"
```

## 3.4 Sonuc Text File

```
c:\gsm\sonuc.php
----------------------------------------------------------------------------
Message type: SMS-DELIVER
SC address: '491710762100'
More messages to send: 1
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: '171'
Protocol identifier: 0x39
Data coding scheme: default alphabet
SC timestamp: 04/16/1999 08:09:44 AM (+0200)
User data length: 160
```

```
Message type: SMS-DELIVER
SC address: '491710762100'
More messages to send: 1
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: '01805000102'
Protocol identifier: 0x39
Data coding scheme: default alphabet
SC timestamp: 12/17/1998 02:10:55 PM (+0100)
User data length: 159


------------------------------------------------------------------------


------------------------------------------------------------------------
Message type: SMS-DELIVER
SC address: '41794991200'
More messages to send: 1
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: 'dialing.de '
Protocol identifier: 0x39
Data coding scheme: default alphabet
SC timestamp: 04/21/2001 12:15:28 PM (+0000)
User data length: 0
User data header: 0x
User data: ''
------------------------------------------------------------------------


------------------------------------------------------------------------
Message type: SMS-DELIVER
SC address: '41794991200'
More messages to send: 1
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: 'dialing.de '
Protocol identifier: 0x39
Data coding scheme: default alphabet
SC timestamp: 04/21/2001 12:15:28 PM (+0000)
User data length: 0
------------------------------------------------------------------------
```

```
Message type: SMS-DELIVER
SC address: ''
More messages to send: 0
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: ''
Protocol identifier: 0x0
Data coding scheme: default alphabet
SC timestamp: 00/00/2000 12:00:00 AM (+0000)
User data length: 0
User data header: 0x
User data: ''
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Message type: SMS-DELIVER
SC address: ''
More messages to send: 0
Reply path: 0
User data header indicator: 0
Status report indication: 0
Originating address: ''
Protocol identifier: 0x0
Data coding scheme: default alphabet
SC timestamp: 00/00/2000 12:00:00 AM (+0000)
User data length: 0
User data header: 0x
User data: ''
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Message type: SMS-DELIVER-REPORT
SC address: ''
Protocol identifier present: 0
Data coding scheme present: 0
User data length present: 0
--------------------------------------------------------------------------
```

```
Message type: SMS-COMMAND
SC address: ''
Message reference: 0
Status report request: 1
Protocol identifier: 0x0
Command type: 0x0
Message number: 0
Destination address: ''
Command data length: 0
Command data: ''
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Message type: SMS-SUBMIT
SC address: ''
Reject duplicates: 1
Validity period format: relative
Reply path: 0
User data header indicator: 0
Status report request: 0
Message reference: 0
Destination address: ''
Protocol identifier: 0x0
Data coding scheme: default alphabet
Validity period: 2 days
User data length: 35
User data header: 0x
User data: 'This is a submit message, isn't it?'
```

## 3.5 Check Sensors & Send Emergency SMS

```
*****************************************************************
*****
// * Purpose:Check Sensors & send emergency sms
// *
//*
// *
//
*****************************************************************
*****
#include <dos.h>
 #include <stdio.h>
 #define DATA    0x0378
 #define STATUS  DATA+1
 #define CONTROL DATA+2
 void main()
 {
   int  veri;

          do while(1)
          {
                 veri = inp(DATA);
                 if (veri = 0x1 )
                 {
                   system('c:\gsm\gsmsmsd -r -b 38400 -d :COM1 --spool
                          c:\gsm\msg.txt -f --store sm --action php
                          c:\gsm\sonuc.php');
                        delay(5000);
                 };
          };
   return 0;
 }
```

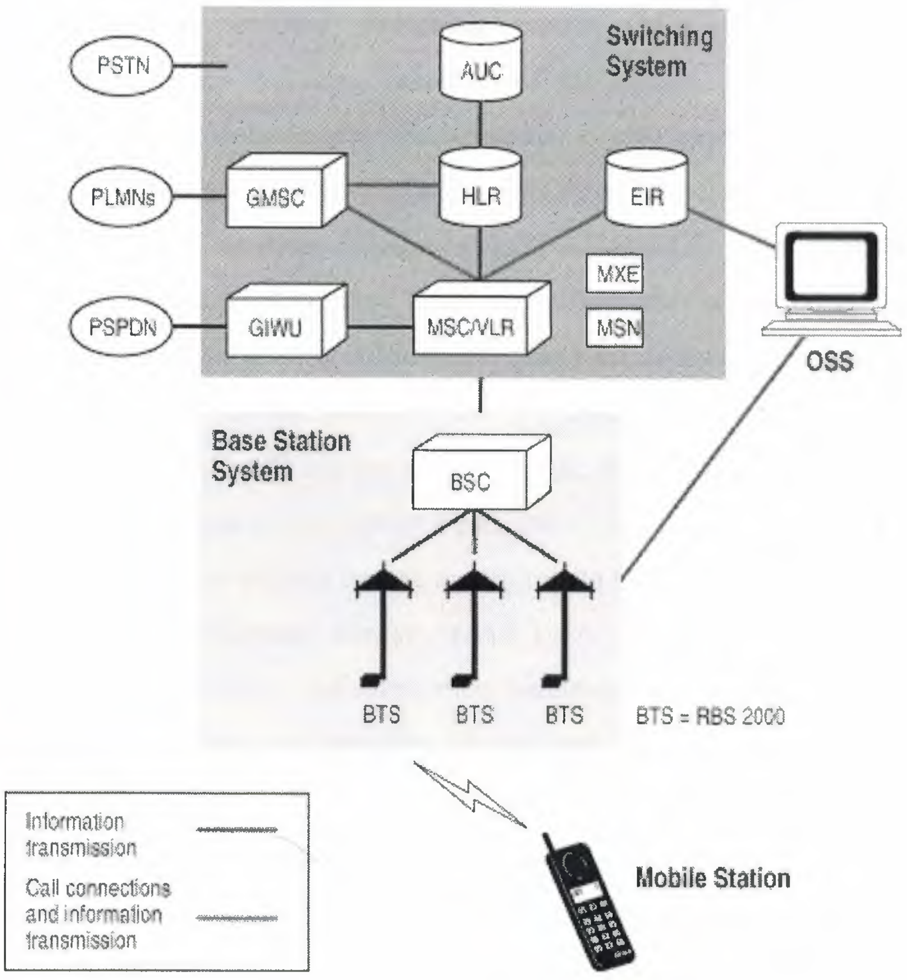
# CHAPTER 4

# GLOBAL SYSTEM FOR MOBILE NETWORK

## 4.1 The GSM Network

GSM provides recommendations, not requirements. The GSM specifications define the functions and interface requirements in detail but do not address the hardware. The reason for this is to limit the designers as little as possible but still to make it possible for the operators to buy equipment from different suppliers. The GSM network is divided into three major systems: the switching system (SS), the base station system (BSS), and the operation and support system (OSS). The basic GSM network elements are shown in *Figure 4.1*.

**Figure 4.1** GSM Network Elements



## 4.2 The Switching System

The switching system (SS) is responsible for performing call processing and subscriber-related functions. The switching system includes the following functional units.

- **home location register (HLR)**—The HLR is a database used for storage and management of subscriptions. The HLR is considered the most important database, as it stores permanent data about subscribers, including a subscriber's service profile, location information, and activity status. When an individual buys a subscription from one of the PCS operators, he or she is registered in the HLR of that operator.
- **mobile services switching center (MSC)**—The MSC performs the telephony switching functions of the system. It controls calls to and from other telephone and data systems. It also performs such functions as toll ticketing, network interfacing, common channel signaling, and others.
- **visitor location register (VLR)**—The VLR is a database that contains temporary information about subscribers that is needed by the MSC in order to service visiting subscribers. The VLR is always integrated with the MSC. When a mobile station roams into a new MSC area, the VLR connected to that MSC will request data about the mobile station from the HLR. Later, if the mobile station makes a call, the VLR will have the information needed for call setup without having to interrogate the HLR each time.
- **authentication center (AUC)**—A unit called the AUC provides authentication and encryption parameters that verify the user's identity and ensure the confidentiality of each call. The AUC protects network operators from different types of fraud found in today's cellular world.
- **equipment identity register (EIR)**—The EIR is a database that contains information about the identity of mobile equipment that prevents calls from stolen, unauthorized, or defective mobile stations. The AUC and EIR are implemented as stand-alone nodes or as a combined AUC/EIR node.

## 4.3 The Base Station System (BSS)

All radio-related functions are performed in the BSS, which consists of base station controllers (BSCs) and the base transceiver stations (BTSs).

- **BSC**—The BSC provides all the control functions and physical links between the MSC and BTS. It is a high-capacity switch that provides functions such as handover, cell configuration data, and control of radio frequency (RF) power levels in base transceiver stations. A number of BSCs are served by an MSC.
- **BTS**—The BTS handles the radio interface to the mobile station. The BTS is the radio equipment (transceivers and antennas) needed to service each cell in the network. A group of BTSs are controlled by a BSC.

## 4.4 The Operation and Support System

The operations and maintenance center (OMC) is connected to all equipment in the switching system and to the BSC. The implementation of OMC is called the operation and support system (OSS). The OSS is the functional entity from which the network operator monitors and controls the system. The purpose of OSS is to offer the customer cost-effective support for centralized, regional, and local operational and maintenance activities that are required for a GSM network. An important function of OSS is to provide a network overview and support the maintenance activities of different operation and maintenance organizations.

## 4.5 Additional Functional Elements

Other functional elements shown in *Figure 2* are as follows:

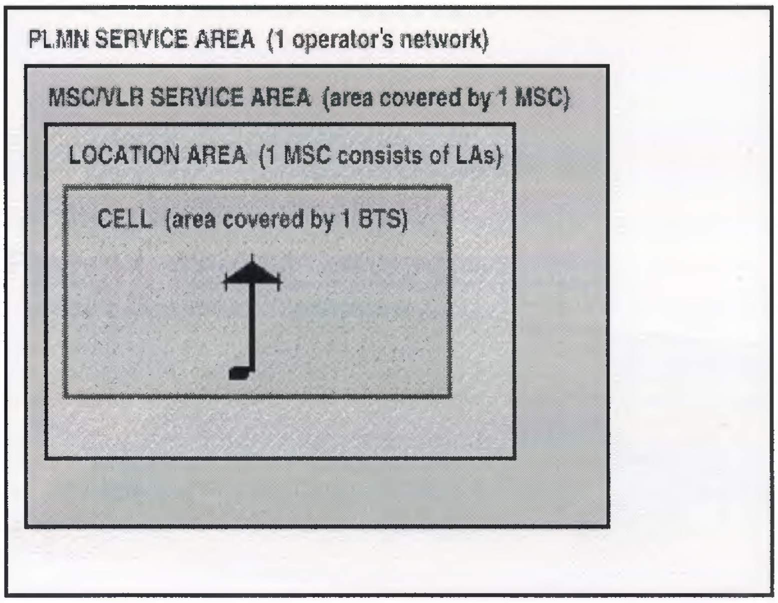
- **message center (MXE)**—The MXE is a node that provides integrated voice, fax, and data messaging. Specifically, the MXE handles short message service, cell broadcast, voice mail, fax mail, e-mail, and notification.
- **mobile service node (MSN)**—The MSN is the node that handles the mobile intelligent network (IN) services.

- **gateway mobile services switching center (GMSC)**—A gateway is a node used to interconnect two networks. The gateway is often implemented in an MSC. The MSC is then referred to as the GMSC.
- **GSM interworking unit (GIWU)**—The GIWU consists of both hardware and software that provides an interface to various networks for data communications. Through the GIWU, users can alternate between speech and data during the same call. The GIWU hardware equipment is physically located at the MSC/VLR.

## 4.6 GSM Network Areas

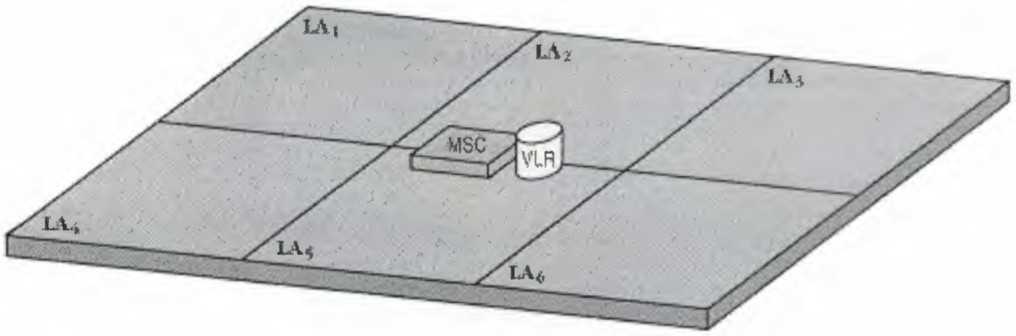
The GSM network is made up of geographic areas. As shown in *Figure 4.2*, these areas include cells, location areas (LAs), MSC/VLR service areas, and public land mobile network (PLMN) areas.

**Figure 4.2** Network Areas


PLMN SERVICE AREA (1 operator's network)
MSC/VLR SERVICE AREA (area covered by 1 MSC)
LOCATION AREA (1 MSC consists of LAs)
CELL (area covered by 1 BTS)

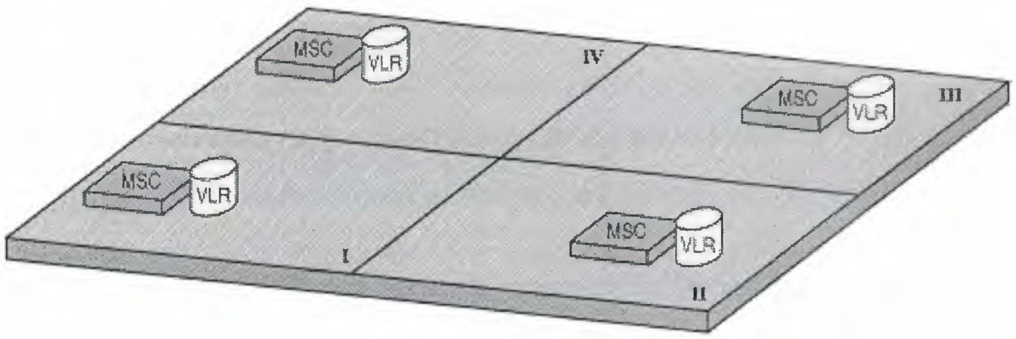
The cell is the area given radio coverage by one base transceiver station. The GSM network identifies each cell via the cell global identity (CGI) number assigned to each cell. The location area is a group of cells. It is the area in which the subscriber is paged. Each LA is served by one or more base station controllers, yet only by a single MSC (see *Figure 4.3*). Each LA is assigned a location area identity (LAI) number.

**Figure 4.3** Location Areas



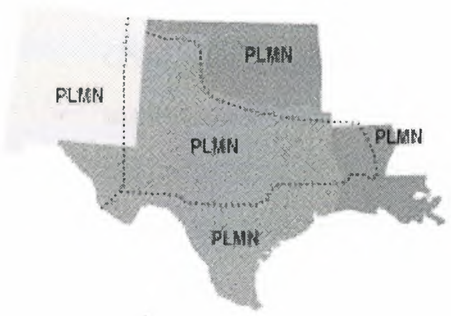
An MSC/VLR service area represents the part of the GSM network that is covered by one MSC and which is reachable, as it is registered in the VLR of the MSC (see *Figure 4.4*).

**Figure 4.4** MSC/VLR Service Areas



The PLMN service area is an area served by one network operator (see *Figure 4.5*).

**Figure 4.5** PLMN Network Areas



## 4.7 GSM Specifications

Before looking at the GSM specifications, it is important to understand the following basic terms:

- **bandwidth**—the range of a channel's limits; the broader the bandwidth, the faster data can be sent
- **bits per second (bps)**—a single on-off pulse of data; eight bits are equivalent to one byte
- **frequency**—the number of cycles per unit of time; frequency is measured in hertz (Hz)
- **kilo (k)**—kilo is the designation for 1,000; the abbreviation kbps represents 1,000 bits per second
- **megahertz (MHz)**—1,000,000 hertz (cycles per second)
- **milliseconds (ms)**—one-thousandth of a second
- **watt (W)**—a measure of power of a transmitter

Specifications for different personal communication services (PCS) systems vary among the different PCS networks. Listed below is a description of the specifications and characteristics for GSM.

- **frequency band**—The frequency range specified for GSM is 1,850 to 1,990 MHz (mobile station to base station).

- **duplex distance**—The duplex distance is 80 MHz. Duplex distance is the distance between the uplink and downlink frequencies. A channel has two frequencies, 80 MHz apart.
- **channel separation**—The separation between adjacent carrier frequencies. In GSM, this is 200 kHz.
- **modulation**—Modulation is the process of sending a signal by changing the characteristics of a carrier frequency. This is done in GSM via Gaussian minimum shift keying (GMSK).
- **transmission rate**—GSM is a digital system with an over-the-air bit rate of 270 kbps.
- **access method**—GSM utilizes the time division multiple access (TDMA) concept. TDMA is a technique in which several different calls may share the same carrier. Each call is assigned a particular time slot.
- **speech coder**—GSM uses linear predictive coding (LPC). The purpose of LPC is to reduce the bit rate. The LPC provides parameters for a filter that mimics the vocal tract. The signal passes through this filter, leaving behind a residual signal. Speech is encoded at 13 kbps.

## 4.8 GSM Subscriber Services

There are two basic types of services offered through GSM: telephony (also referred to as teleservices) and data (also referred to as bearer services). Telephony services are mainly voice services that provide subscribers with the complete capability (including necessary terminal equipment) to communicate with other subscribers. Data services provide the capacity necessary to transmit appropriate data signals between two access points creating an interface to the network. In addition to normal telephony and emergency calling, the following subscriber services are supported by GSM:

- **dual-tone multifrequency (DTMF)**—DTMF is a tone signaling scheme often used for various control purposes via the telephone network, such as remote control of an answering machine. GSM supports full-originating DTMF.

- **facsimile group III**—GSM supports CCITT Group 3 facsimile. As standard fax machines are designed to be connected to a telephone using analog signals, a special fax converter connected to the exchange is used in the GSM system. This enables a GSM–connected fax to communicate with any analog fax in the network.
- **short message services**—A convenient facility of the GSM network is the short message service. A message consisting of a maximum of 160 alphanumeric characters can be sent to or from a mobile station. This service can be viewed as an advanced form of alphanumeric paging with a number of advantages. If the subscriber's mobile unit is powered off or has left the coverage area, the message is stored and offered back to the subscriber when the mobile is powered on or has reentered the coverage area of the network. This function ensures that the message will be received.
- **cell broadcast**—A variation of the short message service is the cell broadcast facility. A message of a maximum of 93 characters can be broadcast to all mobile subscribers in a certain geographic area. Typical applications include traffic congestion warnings and reports on accidents.
- **voice mail**—This service is actually an answering machine within the network, which is controlled by the subscriber. Calls can be forwarded to the subscriber's voice-mail box and the subscriber checks for messages via a personal security code.
- **fax mail**—With this service, the subscriber can receive fax messages at any fax machine. The messages are stored in a service center from which they can be retrieved by the subscriber via a personal security code to the desired fax number.

## 4.9 Supplementary Services

GSM supports a comprehensive set of supplementary services that can complement and support both telephony and data services. Supplementary services are defined by GSM and are characterized as revenue-generating features. A partial listing of supplementary services follows.

- **call forwarding**—This service gives the subscriber the ability to forward incoming calls to another number if the called mobile unit is not reachable, if it is busy, if there is no reply, or if call forwarding is allowed unconditionally.
- **barring of outgoing calls**—This service makes it possible for a mobile subscriber to prevent all outgoing calls.
- **barring of incoming calls**—This function allows the subscriber to prevent incoming calls. The following two conditions for incoming call barring exist: baring of all incoming calls and barring of incoming calls when roaming outside the home PLMN.
- **advice of charge (AoC)**—The AoC service provides the mobile subscriber with an estimate of the call charges. There are two types of AoC information: one that provides the subscriber with an estimate of the bill and one that can be used for immediate charging purposes. AoC for data calls is provided on the basis of time measurements.
- **call hold**—This service enables the subscriber to interrupt an ongoing call and then subsequently reestablish the call. The call hold service is only applicable to normal telephony.
- **call waiting**—This service enables the mobile subscriber to be notified of an incoming call during a conversation. The subscriber can answer, reject, or ignore the incoming call. Call waiting is applicable to all GSM telecommunications services using a circuit-switched connection.
- **multiparty service**—The multiparty service enables a mobile subscriber to establish a multiparty conversation—that is, a simultaneous conversation between three and six subscribers. This service is only applicable to normal telephony.
- **calling line identification presentation/restriction**—These services supply the called party with the integrated services digital network (ISDN) number of the calling party. The restriction service enables the calling party to restrict the presentation. The restriction overrides the presentation.
- **closed user groups (CUGs)**—CUGs are generally comparable to a PBX. They are a group of subscribers who are capable of only calling themselves and certain numbers.

## 4.10 The Short Message Service (SMS)

The Short Message Service (SMS) allows text messages to be sent and received to and from mobile telephones. The text can comprise words or numbers or an alphanumeric combination. SMS was created as part of the GSM Phase 1 standard. The first short message is believed to have been sent in December 1992 from a PC to a mobile phone on the Vodafone GSM network in the UK. Each short message is up to 160 characters in length when Latin alphabets are used, and 70 characters in length when non-Latin alphabets such as Arabic and Chinese are used.

There is no doubting the success of SMS. The market in Europe alone had reached over three billion short messages per month as of December 1999, despite little in proactive marketing by network operators and phone manufacturers. Key market drivers over the next two years, such as the Wireless Application Protocol (WAP), will continue this growth path.

Typical uses of SMS include notifying a mobile phone owner of a voicemail message, alerting a salesperson of an inquiry and telling a driver the address of the next pickup.

## 4.11 Sms Technology

SMS is essentially similar to paging, but SMS messages do not require the mobile phone to be active and within range, as they will be held for a number of days until the phone is active and within range. SMS messages are transmitted within the same cell or to anyone with roaming capability. They can also be sent to digital phones from a web site equipped with a PC Link or from one digital phone to another. An SMS gateway is a web site that lets you enter an SMS message to someone within the cell served by that gateway or acts as an international gateway for users with roaming capability.

The SMS is a store and forward service. In other words, short messages are not sent directly from sender to recipient, but via an SMS Center. Each mobile telephone network that supports SMS has one or more messaging centers to handle and manage the short messages.

The SMS features confirmation of message delivery. This means that, unlike paging, users do not simply send a short message and trust and hope that it gets delivered. Instead the sender of the short message can receive a return message back notifying them whether the short message has been delivered or not.

Short messages can be sent and received simultaneously with GSM (Global System for Mobile Communications) voice, data and fax calls. This is possible because whereas voice, data and fax calls take over a dedicated radio channel for the duration of the call, short messages travel over and above the radio channel using the signaling path. As such, users of SMS rarely, if ever, get a busy or engaged signal as they can do during peak network usage times.

Ways of sending multiple short messages are available. SMS concatenation (stringing several short messages together) and SMS compression (getting more than 160 characters of information within a single short message) have been defined and incorporated in the GSM SMS standards.

The network operator needs to purchase its first generation SMS Center as part of the network commissioning plan. The initial SMS Center may simply be a voice mail platform module or a stand-alone SMS Center. It is not possible to make the SMS available without an SMS Center since all short messages pass through the SMS Center.

## 4.12 Recent Sms Developments

Because simple person-to-person messaging is such an important component of total SMS traffic volumes, anything that simplifies message generation is an important enabler of SMS. Predictive text input algorithms significantly reduce the number of key strokes that need to be made to input a message. T9, from Tegic, anticipates which word the user is trying to generate. Widespread incorporation of such algorithms into the installed base of mobile phones will typically lead to an average uplift in SMS traffic of 25% per enabled user. These predictive text algorithms support multiple languages.

The introduction of standardised protocols such as SIM Application Toolkit and the Wireless Application Protocol (WAP) contribute to an increase in messaging usage by providing a standard service development and deployment environment for application developers and business partners. These protocols also make it easier for users to reply to and otherwise access messaging services through custom menus on the phone. While these protocols are only a means to an end and not new messaging destinations or services, they are likely to lead to a 10-15% uplift in total SMS volumes.

The introduction of more user-friendly terminals contributes to increases in messaging usage. Terminals such as smart phones make it easier for users to originate, reply to and otherwise access messaging services through the provision of a QWERTY keyboard, rather than the limited keypad on standard mobile phones.

# CHAPTER 5

# WIRELESS TECHNOLOGY

## 5.1 Wireless Technology

Wireless technology uses individual radio frequencies over and over again by dividing a service area into separate geographic zones called cells. Cells can be as small as an individual building (say an airport or arena) or as big as 20 miles across, or any size in between. Each cell is equipped with its own radio transmitter/receiver antenna.

Because the system operates at such a low power, a frequency being used to carry a phone conversation in one cell can be used to carry another conversation in a nearby cell without interference. (this allows much greater capacity than radio systems like Citizens Band (CB) in which all users must try to get their messages on the same limited channels.)

When a customer using a wireless phone - car phone or portable - approaches the boundary of one cell, the wireless network senses that the signal is becoming weak and automatically hands off the call to the antenna in the next cell into which the caller is traveling. When subscribers travel beyond their home geographical area, they can still make wireless calls. The wireless carrier in the area where they are traveling provides the service. This is called roaming

Each cellular antenna is linked to a mobile switching center (MSC), which connects your wireless call to the local "wired" telephone network. Wireless carriers own MSCs.

The mobile telephone industry is limited to 45 megahertz MHz of spectrum bandwidth, which without frequency-reuse, would limit each cellular carrier to 396 frequencies or voice channels. In order to increase calling capacity, these low power facilities "reuse" frequencies on the radio spectrum. The manner in which providers organize, or "configure," their cells is an important factor in increasing frequency reuse and establishing an area's calling capacity.

- Analog cellular operates in the 800MHz frequency range and is available across 95 percent of the United States. Analog cellular service sends a voice through the air using continuous radio waves. As the voice signals travel through the air they get weaker with distance. Equipment in the cellular network returns the signal to its original strength, or amplifies it. This technology is the predominant system in use today. The operating system (called the air interface) for analog is called Advanced Mobile Phone Service (AMPS).
- AMPS stands for advanced mobile phone service. AMPS transmits voices as FM radio signals. The original cellular standard, AMPS is still the most widely used system in the U.S., although digital networks are catching up quickly. A variation on AMPS is narrow-band advanced mobile phone service, or NAMPS, which uses a narrower bandwidth and has greater data capabilities.
- Digital cellular shares the 800MHz frequency band with analog and is usually available where analog service is offered. In digital transmissions, a conversation is converted into the ones and zeros of computer code. Unlike analog transmissions that are sent out as a continuously varying electrical signal in the shape of a wave, digital transmissions are a combination of on-and-off pulses of electricity. Several incompatible air interfaces are used to implement digital cellular networks, including Code Division Multiple Access (CDMA) and Time Division Multiple Access (TDMA).
- CDMA is also known as spread spectrum technology because it uses a low-power signal that is "spread" across a wide bandwidth. With CDMA, a phone

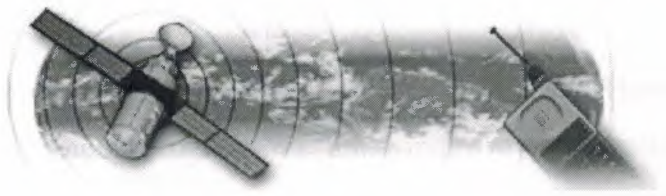
call is assigned a code, which identifies it to the correct receiving phone. Using the identifying code and a low-power signal, a large number of calls can be carried simultaneously on the same group of channels.

- TDMA is a digital air interface technology designed to increase the channel capacity by chopping the signal into pieces and assigning each one to a different time slot, each lasting a fraction of a second. Using TDMA, a single channel can be used to handle simultaneous phone calls.
- GSM stands for global system for mobile communications. It is a type of time division multiple access (TDMA) digital wireless network that has encryption features. GSM is rapidly being deployed worldwide and is the standard in Europe at 900MHz. In the U.S., carriers are deploying GSM at 1900MHz, making GSM phones sold in the U.S. incompatible with European GSM phones, and vice versa.
- Personal Communications Service (PCS) is an all-digital service specifically designed for U.S. operations and is available in metropolitan areas. PCS is a term coined by the Federal Communications Commission to describe a digital, two-way, wireless telecommunications system licensed to operate between 1850-1990 MHz, although the FCC's rules describe "PCS" as a broad family of wireless services without reference to spectrum band or technology. PCS is capable of increased call capacity. PCS networks are CDMA, TDMA and global system for mobile communications (GSM).
- Enhanced Specialized Mobile Radio (ESMR) service is also a digital service, formed by the application of digital systems to traditional dispatch "specialized mobile radio" service spectrum, in the 800 and 900 MHz bands. By aggregating this spectrum, and applying a cellular-like digital network, an ESMR company is able to provide a cellular- or PCS-like voice and data messaging service. NEXTEL is one such company, using Motorola's iDEN (TDMA-based) technology to deliver ESMR services in towns and cities across the U.S.

Satellite systems are another viable type of wireless telecommunications service. Instead of sending and receiving signals from a ground-based antenna, wireless phones will communicate via satellites circling the earth.

- GeoSynchronous Satellites: Geosynchronous satellites represent yet another way of providing wireless communications. These satellites, located 22,300 miles above the earth, revolve around the earth once each twenty-four hours- the same as the earth itself. Consequently they appear to be stationary. Communications between two places on earth can take place by using these satellites; one frequency band is used for the uplink, and another for the downlink. Such satellite systems are excellent for the transmission of data, but they leave something to be desired for voice communications. This is a result of the vast distance and the time it takes for an electrical signal to make an earth-satellite-earth round trip. That time amounts to one quarter of a second. A reply from the called subscriber takes another quarter of a second, and the resultant half a second is definitely noticeable. Consequently, voice communications is seldom carried via geosynchronous satellites

Low Earth Orbit (LEO) satellite system. LEOs are satellites that communicate directly with handheld telephones on earth. Because these satellites are relatively low-less than 900 miles-they move across the sky quite rapidly. In a LEO system the communications equipment on a satellite acts much like the cell site of a cellular system. It catches the call from earth and usually passes it to an earth-based switching system. Because of the speed of the satellite, it is frequently necessary to hand off a particular call to a second satellite just rising over the horizon. This is akin to a cellular system, except that in this case it is the cell site that is moving rather than the subscriber.



## CONCLUSION

Information and Communication technologies begins to affect the other branches of life. Recently developed SMS technology which provide usage of WEWS technology widespread is a clear example. As Security and truthfulness of the knowledge at industrial foundations, police-traffic stations, organizations, shopping centers and at houses becomes important, WEWS applications that makes use of popular technologies seems to be necessary then ever.

Some of the areas that WEWS system can be applicable :

. Industrial Foundations,
. Electricity, water, natural gas distributor systems,
. Smart buildings,
. villas,
. shopping centers,
. houses,
. hotels,
. police-traffic stations,
. automobile industry.

At above areas important processes occur other than alarms and crashes. These are applying periodical test scenarios at fire and security systems, providing security at the energy distributor center, operating feedback systems like generator. It provides necessary profits when necessary person become aware of the status of these kind of processes.

Various methods are used so as to inform the necessary places/person with true data. However most of the time human does the informing process. We must improve the system by automation and remove the human effect. For secure and safe delivery of the necessary information of critical alarms, important events, summary reports ( stock, product, user, organization information etc..) to the authorized person, organizations prefers WEWS system. WEWS system evaluate the signals that it takes from the automation system or organization and transfer the necessary information to the predefined mobile telephone number by making use of SMS.

## REFERENCES:

1. **Mobile Messaging Technologies and Services: SMS, EMS and MMS**
   Gwenael Le Bodic, January 2003
2. **The GSM Evolution: Mobile Packet Data Services**
   Peter Stuckmann, October 2002
3. **The Essential Guide to RF and Wireless**
   Carl J. Weisman, January 2002
4. **The Essential Guide to RF and Wireless**
   Carl J. Weisman, January 2002
5. **3G Wireless Networks**
   Clint Smith, September 2001
6. **Mobile Application Development with SMS and the SIM Toolkit**
   Scott C. Guthery, Mary Cronin, November 2001
7. **Complete Wireless Design**
   Cotter W. Sayre, January 2001
8. **Sensor Technologies and Data Requirements for ITS Applications**
   Lawrence A. Klein, June 2001
9. **Mirosensors,MEMS Smart Devices**
   Julian W. Gardner, J. W. Gardner, Vijay K. Varadan, December 2001
10. **Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering**
    Joseph Mitola, September 2000
11. **Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks with Disk**
    Janet Louise Axelson, Jan Axelson, February 1999
12. **Programming the Parallel Port**
    Dhananjay V. Gadre, Dhananjay V. Garde, January 1998
13. **Electromechanical Sensors and Actuators**
    Ilene J. Busch-Vishniac, I. Busch-Vishniac, October 1998

14. **Fundamentals of Programmable Logic Controllers, Sensors, and Communications**
Jon Stenerson, June 1998

15. **GSM and Personal Communications Handbook**
Siegmund Redl, Matthias Weber, Malcolm Oliphant, Malcolm W. Oliphant, May 1998

16. **Multi-Sensor Fusion : Fundamentals and Applications with Software**
Richard R. Brooks, Sundararaja Iyengar, S. S. Iyengar, November 1997

17. **Parallel Port Complete**
Janet Louise Axelson, Jan Axelson, March 1997

18. **Parallel Programming Using C++**
Gregory V. Wilson, Paul Lu (Editor), Foreword by Bjane Stroustrup, August 1996

19. **Mobile Radio Communications**
Raymond Steele, January 1996

20. **Acoustic Wave Sensors: Theory, Design, Physico-Chemical Applications**
David Stephen Ballantine, D. S. Ballantine, Greg C. Frye, December 1995

21. **Introduction to GSM**
Siegmund Redl, Malcolm W. Oliphant, Mathias K. Weber, Mathias Weber, Matthias Weber, March 1995

22. **Sensors and Control Systems in Manufacturing**
Sabrie Soloman,1994

23. **Expert C Programming**
Peter van der Linden, P. Van Der Linden, June 1994

24. **C Programming Language**
Brian W. Kernighan, Dennis M. Ritchie, Dennis Ritchie, March 1989

25. **Digital Cellular Mobile Radio Links And Networks**
Y:F:Ko,December 1989

26. **Electrochemical Sensors in Immunological Analysis**
T. Ngo, June 1987

**27. Advanced Mobile PHOne services:The Cellular Concept**

J.Vol,january 1979

**28. ETSI(European Telecommunication Standarts Institue) ,**

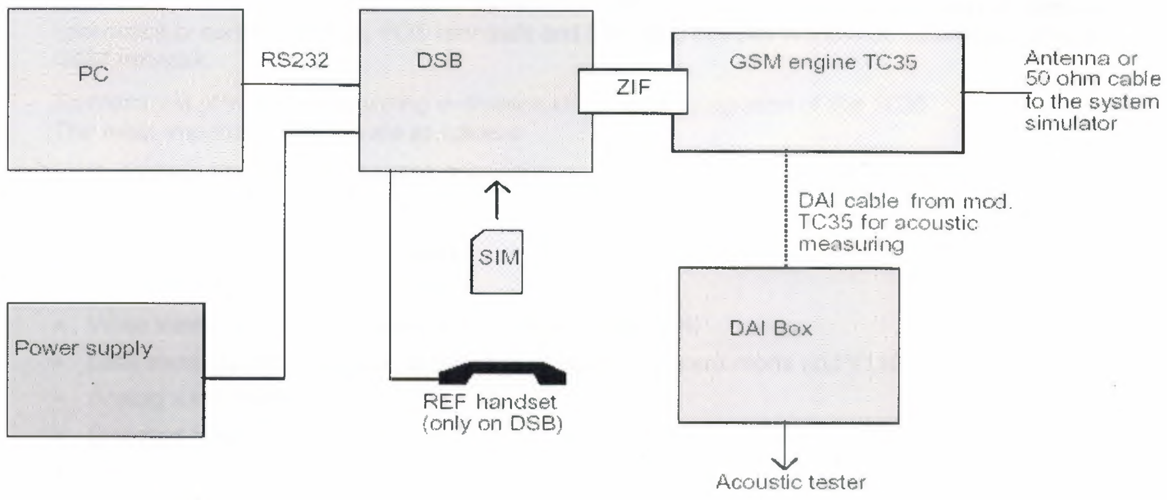
http://www.etsi.com/

**29. AT Command Set For Nokia GSM Products ,**

http://3ton.com/besik/ATNOKIA.pdf

## APPENDIX 1

**App.1** .Genaral Sample Configuration from simens company;


PC
RS232
DSB
ZIF
GSM engine TC35
Antenna or 50 ohm cable to the system simulator
SIM
DAI cable from mod. TC35 for acoustic measuring
Power supply
DAI Box
REF handset (only on DSB)
Acoustic tester

The reference configuration, consisting of

GSM Engine
Development Support Box
Sim Card reader integrated on the DSB
Handset
Pc

## General Product Description TC35

The Siemens TC35 represents the new generation of GSM modules in dual band technology EGSM900/GSM1800. TC35 is lightweight and small, has optimal power consumption and transmits data, voice, SMS and fax. The TC35 is suitable for complex industrial applications such as telemetry, telematics or communication, POS terminals and handheld devices worldwide wherever there is a GSM network.

Siemens will offer a corresponding evaluation kit for rapid integration of the TC35.
The most important features are as follows:

- Frequency EGSM900/GSM1800 Phase 2+
- GSM Class: Small MS
- 2W – Power for Class 4/ EGSM900
- 1W – Power for Class 1/ GSM1800
- Voice transmission with Triple Rate Codec (HR, FR, EFR)
- Data transmission in CSD up to 9.6 kbps in non-transparent mode and V110
- Analog audio interface
- Standard handsfree function
- FAX
- SMS (text and PDU mode) /MT/MO/CB
- Software download possible via RS232 and SIM card reader
- BOF with AT operations set compatible with M20/A20 with the reference status: M20 Technical Description Version 7.0 and M20 SW Version 3.3 as well as A20 Technical Description Version 1.0
- Electrical interface via 40-pin ZIF connector (AVX 04-6240 or similar structure)
- HF connector: coax jack 50Ω Murata GSC, MM9329-2700
- SIM card reader interface 3V via 40-pin ZIF connector
- Reset using ATC or PD (Power Down) via line at the ZIF connector
- RS 232 autobauding (4.8 kbps – 115 kbps)
- Power ON using ignition line at the ZIF connector
- Power OFF using PD (Power Down) via line at the ZIF connector or ATC
- Locating and position services possible with AT^MONI, AT^MONP
- Temperature range from –20°C to +55°C
- Supply voltage from 3.3V to 5.5V
- Power consumption idle mode < 3.5mA
- Power consumption active mode ~ 300mA
- Power consumption peak 1.6 – 2.3A
- Power consumption in the OFF state ~ 100µA
- Dimensions (L,B,H) 54.5x36x6.7 mm
- Volume < 13 cm$^3$
- Weight ~ 18 g

**APPENDIX 2**

GSM AT-Commands for SMS

This AT-Commands related to ETSI (European Telecommunications Standards Institute)

| **5.1.1 AT+CMGC Send an SMS command** | |
|---|---|
| Test command<br>AT+CMGC=? | Response<br>OK |
| Write command<br><br>if text mode (AT+CMGF=1):<br>AT+CMGC=<fo>,<ct>[,<pid>[,<mn>[,<da>[,<toda>]]]]<CR><br>text is entered<br><ctrl-Z/ESC> | Response<br><br>if text mode (+CMGF=1) and sending successful:<br>**+CMGC: <mr>[,<scts>]**<br>if sending fails:<br>**+CMS ERROR: <err>** |
| if PDU mode (AT+CMGF=0):<br>AT+CMGC=<length><CR><br>PDU is given<br><ctrl-Z/ESC><br>+CMGC=? | Response<br><br>if PDU mode (+CMGF=0) and sending successful:<br>**+CMGC: <mr>[,<ackpdu>]**<br>if sending fails:<br>**+CMS ERROR: <err>**<br><br>Parameter<br>`<length>` Length of PDU<br>`<pdu>` See "AT+CMGL"<br>`<mr>` Message reference<br><fo> depending on the command or result code: first octet of GSM 03.40 SMS-DELIVER, SMS-SUBMIT (default 17), SMS-STATUS-REPORT, or SMS-COMMAND (default 2) in integer format<br><ct> GSM 03.40 TP-Command-Type in integer format (default 0)<br><pid> GSM 03.40 TP-Protocol-Identifier in integer format (default 0)<br><toda> GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <da> is + (IRA 43) default is 145, otherwise default is 129)<br><da> GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <toda> |
| Reference<br>GSM 07.05 | Note |

## 5.1.2 AT+CMGD Delete SMS message

| | |
|---|---|
| Test command<br>AT+CMGD=? | Response<br>**OK**<br>Parameter |
| Execute command<br>AT+CMGD=<br><index> | Response<br>TA deletes message from preferred message storage **<mem1>** location **<index>**.<br>**OK**<br>If error is related to ME functionality:<br>**+CMS ERROR <err>**<br>Parameter<br>**<index>** integer type; value in the range of location numbers supported by the associated memory |
| Reference<br>GSM 07.05 | Note |

## 5.1.3 AT+CMGF Select SMS message format

| | |
|---|---|
| Test command<br>AT+CMGF=? | Response<br>**+CMGF:** (list of supported **<mode>s**) **OK**<br>Parameter<br>see set command |
| Read command<br>AT+CMGF? | Response<br>**+CMGF: <mode> OK**<br>Parameter<br>see set command |
| Set command<br>AT+CMGF =<br>[<mode>] | Response<br>TA sets parameter which specifies the input and output format of messages to be used.<br>**OK**<br>Parameter<br>**<mode>** 0 PDU mode<br>1 text mode |
| Reference<br>GSM 07.05 | Note |

## 5.1.4 AT+CMGL List SMS messages from preferred store

| | |
|---|---|
| Test command AT+CMGL=? | Response +CMGL: (list of supported <stat>s) OK Parameter See execute command |
| Execute command AT+CMGL [=<stat>] | Parameter **1) If text mode:** <stat> "REC UNREAD" Received unread messages (default) "REC READ" Received read messages "STO UNSENT" Stored unsent messages "STO SENT" Stored sent messages "ALL" All messages **2) If PDU mode:** <stat> 0 Received unread messages (default) 1 Received read messages 2 Stored unsent messages 3 Stored sent messages 4 All messages Response TA returns messages with status value <stat> from message storage <mem1> to the TE. If status of the message is 'received unread', status in the storage changes to 'received read'. *Note:* if the selected <mem1> can contain different types of SMs (e.g. SMS-DELIVERs, SMS-SUBMITs, SMS-STATUS-REPORTs and SMS-COMMANDs), the response may be a mix of the responses of different SM types. TE application can recognize the response format by examining the third response parameter. |
| | Response 1) If text mode (+CMGF=1) and command successful: for SMS-SUBMITs and/or SMS-DELIVERs: **+CMGL: <index>,<stat>,<oa/da>,[<alpha>],[<scts>][,<tooa/toda>, <length>]<CR><LF><data>[<CR><LF>** **+CMGL: <index>,<stat>,<da/oa>,[<alpha>],[<scts>][,<tooa/toda>, <length>]<CR><LF><data>[...]] OK** for SMS-STATUS-REPORTs: **+CMGL: <index>,<stat>,<fo>,<mr>,[<ra>],[<tora>],<scts>,<dt>,<st> [<CR><LF>** **+CMGL: <index>,<stat>,<fo>,<mr>,[<ra>],[<tora>],<scts>,<dt>,<st> [...]] OK** for SMS-COMMANDs: +CMGL: <index>,<stat>,<fo>,<ct>[<CR><LF> +CMGL: <index>,<stat>,<fo>,<ct>[...]] OK for CBM storage: +CMGL: <index>,<stat>,<sn>,<mid>,<page>,<pages> <CR><LF><data>[<CR><LF> +CMGL: <index>,<stat>,<sn>,<mid>,<page>,<pages> <CR><LF><data>[...]]OK 2) If PDU mode (+CMGF=0) and command successful: +CMGL: <index>,<stat>,[<alpha>],<length><CR><LF><pdu> [<CR><LF>+CMGL: <index>,<stat>,[alpha],<length><CR><LF><pdu> |

[...]] OK

3) If error is related to ME functionality:

+CMS ERROR: <err>

Parameter

| | |
|---|---|
| <alpha> | string type alphanumeric representation of <da> or <oa> corresponding to the entry found in MT phonebook; implementation of this feature is manufacturer- specific |
| <ct> | GSM 03.40 TP-Command-Type in integer format (default 0) |
| <da> | GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <toda> |

<data>

In the case of SMS: GSM 03.40 TP-User-Data in text mode responses; format:

| | |
|---|---|
| -if <dcs> | indicates that GSM 03.38 default alphabet is used and <fo> indicates that GSM 03.40 TP-User-Data-Header-Indication is not set: ME/TA converts GSM alphabet into current TE character set according to rules of Annex A |
| -if <dcs> | indicates that 8-bit or UCS2 data coding scheme is used, or <fo> indicates that GSM 03.40 TP-User-Data-Header-Indication is set: ME/TA converts each 8-bit octet into hexadecimal numbers containing two IRA characters (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)) |

In the case of CBS: GSM 03.41 CBM Content of Message in text mode responses; format:

| | |
|---|---|
| - if <dcs> | indicates that GSM 03.38 default alphabet is used: ME/TA converts GSM alphabet into current TE character set according to rules of Annex A |
| -if <dcs> | indicates that 8-bit or UCS2 data coding scheme is used: ME/TA converts each 8-bit octet into hexadecimal numbers containing two IRA characters |

Parameter

| | |
|---|---|
| <dt> | GSM 03.40 TP-Discharge-Time in time-string format: "yy/MM/dd,hh:mm:ss±zz", where characters indicate year (two last digits), month, day, hour, minutes, seconds and time zone. For example, 6th of May 1994, 22:10:00 GMT+2 hours equals "94/05/06,22:10:00+08" |
| <fo> | depending on the command or result code: first octet of GSM 03.40 SMS-DELIVER, SMS-SUBMIT (default 17), SMS-STATUS-REPORT, or SMS-COMMAND (default 2) in integer format |
| <length> | integer type value indicating in the text mode (+CMGF=1) the length of the message body <data> (or <cdata>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length) |
| <index> | integer type; value in the range of location numbers supported by the associated memory |
| <mid> | GSM 03.41 CBM Message Identifier in integer format |
| <mr> | GSM 03.40 TP-Message-Reference in integer format |
| <oa> | GSM 03.40 TP-Originating-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tooa> |
| <pages> | GSM 03.41 CBM Page Parameter bits 0-3 in integer format |
| <pdu> | In the case of SMS: GSM 04.11 SC address followed by GSM 03.40 TPDU in hexadecimal format: ME/TA converts each octet of TP data unit into hexadecimal numbers containing two IRA characters (e.g. octet with |

| | | |
|---|---|---|
| | | integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)). In the case of CBS: GSM 03.41 TPDU in hexadecimal format. |
| | <page> | GSM 03.41 CBM Page Parameter bits 4-7 in integer format |
| | <ra> | GSM 03.40 TP-Recipient-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tora> |
| | <scts> | GSM 03.40 TP-Service-Centre-Time-Stamp in time-string format (refer <dt>) |
| | <sn> | GSM 03.41 CBM Serial Number in integer format |
| | <st> | GSM 03.40 TP-Status in integer format |
| | <toda> | GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <da> is + (IRA 43) default is 145, otherwise default is 129) |
| | <tooa> | GSM 04.11 TP-Originating-Address Type-of-Address octet in integer format (default refer<toda>) |
| | <tora> | GSM 04.11 TP-Recipient-Address Type-of-Address octet in integer format (default refer<toda>) |
| Reference<br>GSM 07.05 | Note | |

## 5.1.5 AT+CMGR Read SMS message

| | |
|---|---|
| Test command<br>AT+CMGR=? | Response<br>OK<br>Parameter |
| Execute command<br>AT+CMGR=<br><index> | Parameter<br>**<index>** integer type; value in the range of location numbers supported by the associated memory<br>Response<br>TA returns SMS message with location value <index> from message storage **<mem1>** to the TE. If status of the message is 'received unread', status in the storage changes to 'received read'.<br><br>1) If text mode (**+CMGF=1**) and command successful:<br><br>for SMS-DELIVER:<br>**+CMGR: <stat>,<oa>,[<alpha>],<scts> [,<tooa>,<fo>,<pid>,<dcs>, <sca>,<tosca>,<length>]<CR><LF><data>**<br><br>for SMS-SUBMIT:<br>**+CMGR: <stat>,<da>,[<alpha>] [,<toda>,<fo>,<pid>,<dcs>,[<vp>], <sca>,<tosca>,<length>]<CR><LF><data>**<br><br>for SMS-STATUS-REPORT:<br>**+CMGR: <stat>,<fo>,<mr>,[<ra>],[<tora>],<scts>,<dt>,<st>**<br><br>for SMS-COMMAND:<br>**+CMGR: <stat>,<fo>,<ct> [,<pid>,[<mn>],[<da>],[<toda>],<length> <CR><LF><cdata>]**<br><br>for CBM storage:<br>**+CMGR: <stat>,<sn>,<mid>,<dcs>,<page>,<pages><CR><LF><data>**<br><br>2) If PDU mode (+CMGF=0) and command su ccessful:<br><br>**+CMGR: <stat>,[<alpha>],<length><CR><LF><pdu> OK**<br><br>3)If error is related to ME functionality:<br>**+CMS ERROR: <err>**<br><br>Parameter<br>**<alpha>** string type alphanumeric representation of <da> or <oa> corresponding to the entry found in MT phonebook; implementation of this feature is manufacturer-specific<br>**<stat>** integer type in PDU mode (default 0), or string type in text mode (default "REC UNREAD"); indicates the status of message in memory: defined values:<br>0 "REC UNREAD" received unread message (i.e. new message)<br>1 "REC READ" received read message<br>2 "STO UNSENT" stored unsent message (only applicable to SMs)<br>3 "STO SENT" stored sent message (only applicable to SMs)<br>4 "ALL" all messages (only applicable to AT+CMGL List SMS messages from preferred store command) |

| | | |
|---|---|---|
| | <ct> | GSM 03.40 TP-Command-Type in integer format (default 0) |
| | <da> | GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <toda> |
| | <data> | |
| | | In the case of SMS: GSM 03.40 TP-User-Data in text mode responses; format: |
| | -if <dcs> | indicates that GSM 03.38 default alphabet is used and <fo> indicates that GSM 03.40 TP-User-Data-Header-Indication is not set: ME/TA converts GSM alphabet into current TE character set according to rules covered in Annex A |
| | -if <dcs> | indicates that 8-bit or UCS2 data coding scheme is used, or <fo> indicates that GSM 03.40 TP-User-Data-Header-Indication is set: ME/TA converts each 8-bit octet into hexadecimal numbers containing two IRA characters (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)) |
| | | In the case of CBS: GSM 03.41 CBM Content of Message in text mode responses; format: |
| | Parameter | |
| | - if <dcs> | indicates that GSM 03.38 default alphabet is used: ME/TA converts GSM alphabet into current TE character set according to rules covered in Annex A |
| | -if <dcs> | indicates that 8-bit or UCS2 data coding scheme is used: ME/TA converts each 8-bit octet into hexadecimal numbers containing two IRA characters |
| | <dcs> | depending on the command or result code: GSM 03.38 SMS Data Coding Scheme (default 0), or Cell Broadcast Data Coding Scheme in integer format |
| | <cdata> | GSM 03.40 TP-Command-Data in text mode responses; ME/TA converts each 8-bit octet into two IRA character long hexadecimal number (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)) |
| | <dt> | GSM 03.40 TP-Discharge-Time in time-string format: "yy/MM/dd,hh:mm:ss±zz", where characters indicate year (two last digits), month, day, hour, minutes, seconds and time zone. For example, 6th of May 1994, 22:10:00 GMT+2 hours equals "94/05/06,22:10:00+08" |
| | <fo> | depending on the command or result code: first octet of GSM 03.40 SMS- DELIVER, SMS-SUBMIT (default 17), SMS-STATUS-REPORT, or SMS-COMMAND (default 2) in integer format |
| | <length> | integer type value indicating in text mode (+CMGF=1) the length of the message body <data> (or <cdata>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length) |
| | <index> | integer type; value in the range of location numbers supported by the associated memory |
| | <mid> | GSM 03.41 CBM Message Identifier in integer format |
| | <mr> | GSM 03.40 TP-Message-Reference in integer format |
| | <oa> | GSM 03.40 TP-Originating-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tooa> |
| | <page> | GSM 03.41 CBM Page Parameter bits 4-7 in integer format |
| | <pages> | GSM 03.41 CBM Page Parameter bits 0-3 in integer format |

| | | |
|---|---|---|
| | <pdu> | In the case of SMS: GSM 04.11 SC address followed by GSM 03.40 TPDU in hexadecimal format: ME/TA converts each octet of TP data unit into hexadecimal numbers containing two IRA characters (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)). In the case of CBS: <ra> GSM 03.40 TP-Recipient-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tora> |
| | <pid> | GSM 03.40 TP-Protocol-Identifier in integer format (default 0) |
| | <ra> | GSM 03.40 TP-Recipient-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted to characters of the currently selected TE character set (refer command AT+CSCS Select TE character set.); type of address given by <tora> |
| | <sca> | GSM 04.11 RP SC address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted to characters of the currently selected TE character set (refer command AT+CSCS Select TE character set.); type of address given by <tosca> |
| | <scts> | GSM 03.40 TP-Service-Centre-Time-Stamp in time-string format (refer <dt>) |
| | <sn> | GSM 03.41 CBM Serial Number in integer format |
| | <st> | GSM 03.40 TP-Status in integer format |
| | <toda> | GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <da> is + (IRA 43) default is 145, otherwise default is 129) |
| | <tooa> | GSM 04.11 TP-Originating-Address Type-of-Address octet in integer format (default refer<toda>) |
| | <tora> | GSM 04.11 TP-Recipient-Address Type-of-Address octet in integer format (default refer<toda>) |
| | <tosca> | GSM 04.11 RP SC address Type-of-Address octet in integer format (default refer <toda>) |
| | <vp> | depending on SMS-SUBMIT <fo> setting: GSM 03.40 TP-Validity-Period either in integer format (default 167) or in time-string format (refer <dt>) |
| Reference<br>GSM 07.05 | Note | |

## 5.1.6 AT+CMGS Send SMS message

| | |
|---|---|
| Test command<br>AT+CMGS=? | Response<br>**OK**<br>Parameter |
| Execute command<br>1) If text mode (+CMGF=1):<br>+CMGS=<da>[,<toda>]<CR><br>text is entered<br><ctrl-Z/ESC><br>2) If PDU mode (+CMGF=0):<br>+CMGS=<length><CR><br>PDU is given <ctrl-Z/ESC><br>ESC aborts message | Response<br>TA transmits SMS message from a TE to the network (SMS-SUBMIT). Message reference value <mr> is returned to the TE on successful message delivery. Value can be used to identify message upon unsolicited delivery status report result code.<br>1) If text mode (+CMGF=1) and sending successful:<br>**+CMGS: <mr>[,scts>] OK**<br>2) If PDU mode (+CMGF=0) and sending successful:<br>**+CMGS: <mr>[,ackpdu>] OK**<br>3) If error is related to ME functionality:<br>**+CMS ERROR: <err>**<br><br>Parameter<br>**<da>** GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <toda><br>**<toda>** GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <da> is + (IRA 43) default is 145, otherwise default is 129)<br>**<length>** integer type value indicating in the text mode (+CMGF=1) the length of the message body <data> (or <cdata>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length)<br>**<mr>** GSM 03.40 TP-Message-Reference in integer format<br>**<scts>** GSM 03.40 TP-Service-Centre-Time-Stamp in time-string format (refer <dt>)<br>**<dt>** GSM 03.40 TP-Discharge-Time in time-string format: "yy/MM/dd,hh:mm:ss±zz", where characters indicate year (two last digits), month, day, hour, minutes, seconds and time zone. For example, 6th of May 1994, 22:10:00 GMT+2 hours equals "94/05/06,22:10:00+08"<br>**<ackpdu>** GSM 03.40 RP-User-Data element of RP-ACK PDU; format is same as for <pdu> in case of SMS, but without GSM 04.11 SC address field and parameter shall be enclosed in double quote characters like a normal string type parameter<br>**<pdu>** In the case of SMS: GSM 04.11 SC address followed by GSM 03.40 TPDU in hexadecimal format: ME/TA converts each octet of TP data unit into hexadecimal numbers containing two IRA characters (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)). In the case of CBS: GSM 03.41 TPDU in hexadecimal format. |

## 5.1.7 AT+CMGW Write SMS message to memory

| | |
|---|---|
| Test command<br>AT+CMGW=? | Response<br>OK<br>Parameter |
| Execute command<br>1) If text mode (+CMGF=1):<br>+CMGW[=<oa/da>[,<tooa/toda>[,stat>]]]<br><CR> text is entered ctrl-Z/ESC><br><ESC> quits without sending<br>2) If PDU mode (+CMGF=0):<br>+CMGW=<length>[,stat]<CR><br>PDU is given <ctrl-Z/ESC> | Response<br>TA transmits SMS message (either SMS-DELIVER or SMS-SUBMIT) from TE to memory storage <mem2>. Memory location <index> of the stored message is returned. Message status will be set to 'stored unsent' unless otherwise given in parameter <stat>.<br>*Note:* SMS-COMMANDs and SMS-STATUS-REPORTs can not be stored in text mode.<br>If writing is successful:<br>**+CMGW: <index> OK**<br>If error is related to ME functionality:<br>**+CMS ERROR: <err>**<br>Parameter<br><oa> GSM 03.40 TP-Originating-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tooa><br><da> GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <toda><br><tooa> GSM 04.11 TP-Originating-Address Type-of-Address octet in integer format (default refer <toda>)<br><toda> GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of <da> is + (IRA 43) default is 145, otherwise default is 129)<br><length> integer type value indicating in the text mode (+CMGF=1) the length of the message body <data> (or <cdata>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length)<br><stat> 0 "REC UNREAD" Received unread messages (default)<br>1 "REC READ" Received read messages<br>2 "STO UNSENT" Stored unsent messages<br>3 "STO SENT" Stored sent messages<br>4 "ALL" All messages<br><pdu> In the case of SMS: GSM 04.11 SC address followed by GSM 03.40 TPDU in hexadecimal format: ME/TA converts each octet of TP data unit into hexadecimal numbers containing two IRA characters (e.g. octet with integer value 42 is presented to TE as two characters 2A (IRA 50 and 65)). In the case of CBS: GSM 03.41 TPDU in hexadecimal format.<br><index> Index of message in selected storage <mem2><br>*Note:* ctrl-Z sends/writes message, Returns Ok ESC aborts input, message NOT sent/written. Returns Ok |

## 5.1.8 AT+CMSS Send SMS message from storage

| | |
|---|---|
| Test command<br>AT+CMSS=? | Response<br>**OK**<br>Parameter |
| Execute command<br>+CMSS=<br><index>[,<da><br>[,<toda>]] | Response<br>TA sends message with location value **<index>** from message storage **<mem2>** to the network (SMS-SUBMIT or SMS-COMMAND). If new recipient address **<da>** is given for SMS-SUBMIT, it shall be used instead of the one stored with the message. Reference value **<mr>** is returned to the TE on successful message delivery. Values can be used to identify message upon unsolicited delivery status report result code. This command should be abortable.<br><br>1) If text mode (**+CMGF=1**) and send successful:<br>**+CMSS: <mr>[,scts>] OK**<br><br>2) If PDU mode (**+CMGF=0**) and send successful:<br>**+CMSS: <mr>[,ackpdu>] OK**<br><br>3) If error is related to ME functionality:<br>**+CMS ERROR: <err>**<br><br>Parameter<br>**<ackpdu>** GSM 03.40 RP-User-Data element of RP-ACK PDU; format is same as for <pdu> in case of SMS, but without GSM 04.11 SC address field and parameter shall be bounded by double quote characters like a normal string type parameter.<br>**<index>** integer type; value in the range of location numbers supported by the associated memory<br>**<da>** GSM 03.40 TP-Destination-Address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by **<toda>**<br>**<scts>** GSM 03.40 TP-Service-Centre-Time-Stamp in time-string format.<br>**<toda>** GSM 04.11 TP-Destination-Address Type-of-Address octet in integer format (when first character of **<da>** is + (IRA 43) default is 145, otherwise default is 129)<br>**<mr>** GSM 03.40 TP-Message-Reference in integer format |

## 5.1.9 AT+CNMA New SMS message acknowledge to ME/TE, only phase 2+

| | |
|---|---|
| Test command<br>AT+CNMA=? | Response<br>1) If text mode (+CMGF=1):<br>**OK**<br><br>2) If PDU mode (+CMGF=0):<br>+CNMA: (list of supported <n>s) OK<br><br>Parameters<br>see execute command |
| Execute command<br>1) If text mode:<br>AT+CNMA<br><br>2) If PDU mode:<br>AT+CN-MA[=<n>[,<length>[<CR><br>**PDU is given**<ctrl-Z/ESC>]]] | Response<br>TA confirms successful receipt of a new message (SMS-DELIVER or SMS-STATUS-REPORT) which is routed directly to the TE. TA shall not send another +CMT or +CDS result code to TE until previous one is acknowledged.<br>If ME does not receive acknowledgment within required time (network timeout), ME should send RP-ERROR to the network. TA shall automatically disable routing to TE by setting both <mt> and <ds> values of +CNMI to zero.<br><br>*Note:* the command shall o n l y be used when +CSMS parameter <service> equals 1 (= phase 2+).<br><br>1) If text mode:<br>**OK**<br><br>2) If PDU mode:<br>**OK**<br><br>3) If error is related to ME functionality:<br>+CMS ERROR: <err><br><br>Parameters<br>**<n>** 0 command operates similarly as defined for the text mode<br>1 send RP-ACK (or buffered result code received correctly)<br>2 send RP-ERROR (if PDU is not given, ME/TA shall send SMS-DELIVER-REPORT with GSM 03.40 TP-FCS value set to 'FF' (unspecified error cause))<br>**<length>** integer type value indicating in the text mode (+CMGF=1) the length of the message body **<data>** (or <cdata>) in characters; or in PDU mode (+CMGF=0), the length of the actual TP data unit in octets (i.e. the RP layer SMSC address octets are not counted in the length) |

| **5.1.10 AT+CNMI New SMS message indications** | |
|---|---|
| Test command<br>AT+CNMI=? | Response<br>**+CNMI:** (list of supported <mode>s), (list of supported <mt>s), (list of supported <bm>s), (list of supported <ds>s), (list of supported <bfr>s) **OK**<br>Parameter<br>see set command |
| Read command<br>AT+CNMI? | Response<br>**+CNMI:** <mode>,<mt>,<bm>,<ds>,<bfr> **OK**<br>Parameter<br>see set command |
| Set command<br>AT+CNMI = [<mode><br>[,<mt>[,<bm><br>[,<ds>[,<bfr>]]]]] | Response<br>TA selects the procedure, how the receipt of new SMS messages from the network is indicated to the TE when TE is active, e.g. DTR signal is ON. If TE is inactive (e.g. DTR signal is OFF), message receiving should be done as specified in GSM 03.38.<br>*__Note1:__* when DTR signal is not available or the state of the signal is ignored (V.25ter command &D0), reliable message transfer can be assured by using +CNMA acknowledgment procedure.<br>*__Note2:__* the rules <mt>=2 and <mt>=3 for storing received SM are possible *__only if phase 2+__* compatibility is activated with +CSMS=1<br>**OK**<br>If error is related to ME functionality:<br>**+CMS ERROR:** <err><br>Parameter<br>**<mode>** 0 Buffer unsolicited result codes in the TA. If TA result code buffer is full, indications can be buffered in some other place or the oldest indications may be discarded and replaced with the new received indications.<br>1 Discard indication and reject new received message unsolicited result codes when TA-TE link is reserved (e.g. in on-line data mode). Otherwise forward them directly to the TE.<br>2 Buffer unsolicited result codes in the TA when TA-TE link is reserved (e.g. in on-line data mode) and flush them to the TE after reservation. Otherwise forward them directly to the TE.<br>3 Forward unsolicited result codes directly to the TE. TA-TE link specific inband technique used to embed result codes and data when TA is in on-line data mode.<br>**<mt>** (the rules for storing received SMs depend on the relevant data coding method (refer to GSM 03.38 [2]), preferred memory storage (+CPMS) setting and this value<br>*__Note__*: if AT command interface is acting as the only display device, the ME must support storage of class 0 messages and messages in the message waiting indication group (discard message)<br>0 No SMS-DELIVER indications are routed to the TE.<br>1 If SMS-DELIVER is stored in ME/TA, indication of the memory location is routed to the TE using unsolicited result code: +CMTI: <mem>,<index> |
| | 2 SMS-DELIVERs (except class 2 messages and messages in the message waiting indication group (store message)) are routed directly to the TE using unsolicited result code: +CMT: [<alpha>],<length><CR><LF><pdu> (PDU mode enabled)<br>3 Class 3 SMS-DELIVERs are routed directly to the TE using unsolicited result codes defined in <mt>=2. Messages of other data coding schemes result in indication as defined in <mt>=1. |

| | | | |
|---|---|---|---|
| | **<bm>** | | (the rules for storing received CBMs depend on the relevant data coding method (refer to GSM 03.38 [2]), the setting of Select CBM Types (+CSCB) and this value: |
| | | 0 | No CBM indications are routed to the TE. |
| | | 1 | If CBM is stored in ME/TA, indication of the memory location is routed to the TE using unsolicited result code: **+CBMI: <mem>,<index>** |
| | | 2 | New CBMs are routed directly to the TE using unsolicited result code: **+CBM: <length><CR><LF><pdu>** (PDU mode enabled) or **+CBM: <sn>,<mid>,<dcs>,<page>,<pages><CR><LF><data>** (text mode enabled) If ME supports data coding groups which define special routing also for messages other than class 3 (e.g. SIM specific messages), ME may choose not to route messages of such data coding schemes into TE (indication of a stored CBM may be given as defined in **<bm>**=1). |
| | | 3 | Class 3 CBMs are routed directly to TE using unsolicited result codes defined in <bm>=2. If CBM storage is supported, messages of other classes result in indication as defined in **<bm>**=1. |
| | **<ds>** | 0 | No SMS-STATUS-REPORTs are routed to the TE. |
| | | 1 | SMS-STATUS-REPORTs routed to TE not supported. |
| | | 2 | indication of memory location routed to TE not supported. |
| | **<bfr>** | 0 | TA buffer of unsolicited result codes defined within this command is flushed to the TE when <mode> 1...3 is entered (OK response shall be given before flushing the codes). |
| | | 1 | TA buffer of unsolicited result codes defined within this command is cleared when <mode> 1...3 is entered. |
| | Unsolicited result code | | |
| | **+CMTI: <mem>,<index>** | | Indication that new message has been received |
| | **+CBMI: <mem>,<index>** | | Indication that new CB-message has been received |
| | **+CMT: ,<length><CR><LF><pdu>** | | Short message is output directly |
| | **+CBM: <length><CR><LF><pdu>** | | Cell broadcast message is output directly |
| Reference | Note | | |
| GSM 07.05 | Parameters can only be set to provider supported values | | |

## 5.1.11 AT+CPMS Preferred SMS message storage

| | |
|---|---|
| Test command<br>AT+CPMS=? | Response<br>**+CPMS:** (list of supported **<mem1>**s),(list of supported **<mem2>**s) ,(list of supported **<mem3>**s)<br>Parameter<br>see set command |
| Read command<br>AT+CPMS? | Response<br>**+CPMS: <mem1>,<used1>,<total1>,<mem2>,<used2>,<total2>, <mem3>,<used3>,<total3> OK**<br>If error is related to ME functionality:<br>**+CMS ERROR**<br>Parameter<br>see set command |
| Set command<br>AT+CPMS = <mem1><br>[,<mem2><br>[,<mem3>]] | Response<br>TA selects memory storages **<mem1>**, **<mem2>** and **<mem3>** to be used for reading, writing, etc.<br>**+CPMS: <used1>,<total1>,<used2>,<total2>,<used3>,<total3> OK**<br>If error is related to ME functionality:<br>**+CMS ERROR:<err>**<br>Parameter<br>**<mem1>** Messages to be read and deleted from this memory storage "SM" SIM message storage<br>**<mem2>** Messages will be written and sent to this memory storage "SM" SIM message storage<br>**<mem3>** Received messages will be placed in this memory storage if routing to PC is not set ("+CNMI") "SM" SIM message storage<br>**<usedx>** Number of messages currently in **<memx>**<br>**<totalx>** Number of messages storable in **<memx>** |
| Reference<br>GSM 07.05 | Note |

## 5.1.12 AT+CSCA SMS service centre address

| | |
|---|---|
| Test command<br>AT+CSCA=? | Response<br>**OK** |
| Read command<br>AT+CSCA? | Response<br>**+CSCA: <sca>,<tosca> OK**<br>Parameter<br>see set command |
| Set command<br>AT+CSCA = <sca>[,<tosca>] | Response<br>TA updates the SMSC address, through which mobile originated SMs are transmitted. In text mode, setting is used by send and write commands. In PDU mode, setting is used by the same commands, but only when the length of the SMSC address coded into <pdu> parameter equals zero.<br>*Note:* this command writes the service centre address to non-volatile memory.<br>**OK**<br>Parameter<br><sca> GSM 04.11 RP SC address Address-Value field in string format; BCD numbers (or GSM default alphabet characters) are converted into characters; type of address given by <tosca><br><tosca> Service centre address format GSM 04.11 RP SC address Type-of-Address octet in integer format (default refer <toda>)<br>*Note:* Parameter field <tosca> is ignored, national/international call center numbers are recognized by the leading + in the number. |
| Reference<br>GSM 07.05 | Note |

## 5.1.13 AT+CSCB Select cell broadcast messages

| | |
|---|---|
| Test command<br>AT+CSCB=? | Response<br>`+CSCB: (list of supported <mode>s)`<br>Parameter<br>`<mode>` 0 Accepts messages that are defined in <mids> and <dcss><br>1 Does not accept messages that are defined in <mids> and <dcss> |
| Read command<br>AT+CSCB?<br><br>Write command<br>AT+CSCB=[<mode>[,<mids>[,<dcss>]]] | Response<br>`+CSCB: <mode>,<mids>,<dcss>`<br>Parameter<br>`<mode>` See Test command<br>`<mids>` String type; combinations of CBM message IDs<br>`<dcss>` String type; combinations of CBM data coding schemes |
| Reference<br>GSM 07.05 | Note<br>It is possible that this command will be changed in case of deviation from U35! |

## 5.1.14 AT+CSDH Show SMS text mode parameters

| | |
|---|---|
| Test command<br>AT+CSDH=? | Response<br>**+CSDH:** (list of supported **<show>**s) **OK**<br>Parameter<br>see set command |
| Read command<br>AT+CSDH? | Response<br>**+CSDH:<show> OK**<br>Parameter<br>see set command |
| Set command<br>AT+CSDH=<br><show> | Response<br>TA sets whether or not detailed header information is shown in text mode result codes.<br>**OK**<br>Parameter<br>**<show>** 0 do not show header values defined in commands **+CSCA** and **+CSMP** (**<sca>**, **<tosca>**, **<fo>**, **<vp>**, **<pid>** and **<dcs>**) nor **<length>**, **<toda>** or **<tooa>** in **+CMT**, **+CMGL**, **+CMGR** result codes for SMS-DELIVERs and SMS-SUBMITs in text mode; for SMS-COMMANDs in **+CMGR** result code, do not show **<pid>**, **<mn>**, **<da>**, **<toda>**, **<length>** or **<cdata>**<br>1 show the values in result codes |
| Reference<br>GSM 07.05 | Note |

## 5.1.15 AT+CSMP Set SMS text mode parameters

| | |
|---|---|
| Test command<br>AT+CSMP=? | Response<br>**+CSMP:** (list of supported **<fo>**s), (list of supported **<vp>**s) **OK**<br>Parameter<br>see set command |
| Read command<br>AT+CSMP? | Response<br>**+CSMP:<fo>,<vp> OK**<br>Parameter<br>see set command |
| Set command<br>AT+CSMP=<br>[<fo>[<vp>[,pid><br>[,<dcs>]]]] | Response<br>TA selects values for additional parameters needed when SM is sent to the network or placed in a storage when text format message mode is selected. It is possible to set the validity period starting from when the SM is received by the SMSC (**<vp>** is in range 0... 255) or define the absolute time of the validity period termination (**<vp>** is a string).<br>Parameter<br>**<fo>** depending on the command or result code: first octet of GSM 03.40 SMS-DELIVER, SMS-SUBMIT (default 17), , or SMS-COMMAND (default 2) in integer format<br>**<vp>** depending on SMS-SUBMIT <fo> setting: GSM 03.40 TP-Validity-Period either in integer format (default 167)<br>**<pid>** Protocol-Identifier in integer format (default 0), refer GSM 03.40<br>**<dcs>** SMS Data Coding Scheme (default 0), or Cell Broadcast Data Coding Scheme in integer format depending on the command or result code: GSM 03.38 |
| Reference<br>GSM 07.05 | Note<br>The command writes the parameters in NON-VOLATILE memory. |

## 5.1.16 AT+CSMS Select Message Service

| | |
|---|---|
| Test command<br>AT+CSMS=? | Response<br>**+CSMS:** (list of supported **<service>**s) **OK**<br>Parameter<br>see set command |
| Read command<br>AT+CSMS? | Response<br>**+CSMS: <service>,<mt>,<mo>,<bm> OK**<br>Parameter<br>see set command |
| Set command<br>AT+CSMS=<br><service> | Response<br>**+CSMS: <mt>,<mo>,<bm> OK**<br>If error is related to ME functionality:<br>**+CMS ERROR: <err>**<br>Parameter<br>**<service>**<br>0 GSM 03.40 and 03.41 (the syntax of SMS AT commands is compatible with GSM 07.05 Phase 2 version 4.7.0; Phase 2+ features which do not require new command syntax may be supported (e.g. correct routing of messages with new Phase 2+ data coding schemes))<br>1 GSM 03.40 and 03.41 (the syntax of SMS AT commands is compatible with GSM 07.05 Phase 2+ version; the requirement of **<service>** setting 1 is mentioned under corresponding command descriptions) Currently not available with the TC35.<br><br>128 Compatibility to Phase 1 and to device type M1 (manufacturer specific)<br><br>**<mt>** Mobile Terminated Messages:<br>**0** Type not supported<br>**1** Type supported<br>**<mo>** Mobile Originated Messages:<br>**0** Type not supported<br>**1** Type supported<br>**<bm>** Broadcast Type Messages:<br>**0** Type not supported<br>**1** Type supported |
| Reference<br>GSM 07.05 | Note |

# VITA

Mehmet Uğurlu was born in Tortum, Erzurum on September 18, 1975 .He received his B.Sc degree in Computer Engineering on January 1998 as a honour student . He works in the Goverment Company as a Software Engineer since 1999. His main areas of interest are Wireless Technology, Software Programming.