CHAPTER 4
SELECTION OF SENSORS, COMMUNICATION PROTOCOLS AND MICROCONTROLLERS

4.1 Selection of Sensors:
Initial work was carried out to show the demonstration of cell phone based remote monitoring capability using PC and output devices were simple LEDs and input devices were dip switches. The messages / commands were transferred between system control cell phone (3315/3210/6610) and user cell phone based on changes in input status or desired control of outputs. At a later stage, microcontroller based remote temperature monitoring application was developed using temperature sensor DS 1820. Finally a cell phone based remote monitoring system for motor pump based irrigation in agriculture and home automation systems were developed and sensors and protection schemes were designed for this application.

4.1.1 Three Phase Voltage Sensing:
Three single phase transformers have been used to sense power failure, unbalanced phase voltages, single phasing, under-voltage and overvoltage conditions.
Fig 4.1 Three Phase Supply Voltage Sensing Circuit

Fig 4.1 shows the circuit for 3 phase supply voltage sensing. Three single phase transformers with secondary windings of 9V, 500 mA are used in system. The corresponding voltages are given to analog inputs of microcontroller based system. Analog voltages are read periodically by microcontroller and occurrence of any abnormal conditions is relayed in form of message to user cell phone through system cell phone.

4.1.2 Overcurrent/ Bearing Failure Sensing:
Whenever overcurrent or bearing failure of induction motor occurs, there is rise in temperature of motor enclosure (Schoen et al. 1995; Izzet Y. Onel et al., 2006, Devaney and Eren, 2004). The basic overcurrent protection for motor is provided by bi-metallic strip of starter. However, in order to ensure maximum reliability, one temperature sensor is mounted on body of motor and another temperature sensor is mounted at a suitable location to measure ambient temperature. Whenever temperature difference exceeds specified safety limits, signal is sent to switch off pump along with error indication of thermal overheating. This arrangement ensures that catastrophic event like burning of motor due to any fault like overcurrent, bearing blockage, insulation failure, etc. are avoided and preventive maintenance can be carried out at substantially lower cost.
The chosen temperature sensor is DS 18S20 manufactured by Dallas Semiconductor (Maxim) (DS 18S20 Datasheet). It has operating temperature range of –55 °C to + 125 °C. It communicates over 1-wire bus and has alarm function with non-volatile user programmable upper and lower trigger points. The major advantages offered by this sensor over other similar IC sensors like AD 590, LM 35, LM 335, etc. are

1. There is no need of external ADC in DS 18S20 based circuit as output is directly in form of serial data.
2. The system has inherent noise immunity without need of amplification, filtering as the signal is in digital form with CRC checksum facility to detect errors during transmission.
3. It can easily be applied in cases where multiple temperature sensors are needed in the system due to its unique 64-bit serial code and 1-wire connectivity.

The block diagram of sensor is shown in Fig 4.2. It has 8 bytes scratchpad. Byte 0 and byte 1 of the scratchpad contain the LSB and the MSB of the temperature register, respectively. These bytes are read-only. Bytes 2 and 3 provide access to TH and TL registers. Bytes 4 and 5 are reserved for internal use by the device. Bytes 6 and 7 contain the COUNT REMAIN and COUNT PER °C registers, which can be used to calculate extended resolution results.

In our system, due to high cost of DS 18S20 temperature sensor, it was decided to use a single temperature sensor and latest temperature reading corresponding to Motor OFF condition is treated as ambient temperature and subsequent readings during motor ON conditions are measured with reference to this baseline (ambient) temperature. The temperature sensor DS 18S20 is mounted on the enclosure of induction motor (Kovačić et al, 2010). Whenever
temperature difference exceeds specified limits, fault occurrence message is sent to user cell phone. DS18S20 is based on single wire transaction protocols and data pin is interfaced to one port pin of microcontroller to obtain temperature related data (http://www.maxim-ic.com/an162).

4.1.2.1 One-Wire Protocols:
Communication with the DS1820 is achieved through the use of “time slots”, which allow data to be transmitted over the 1-Wire TM bus. Fig. 4.3 shows the interface timing specifications for reset, read and write time slots.

The transaction sequence for accessing single DS18S20 using 1-wire interface is as follows:
1. **Reset Pulse** – Microcontroller (master) sends active low pulse of at least 480 µs to start the initialization process using port bit as output. It then releases the bus by configuring port bit as input.

2. **Presence Pulse** – On rising edge of reset pulse, DS 18S20 (slave) waits for 15-60 µs and sends low pulse of 60 to 240 µs on 1-wire bus. At end of this interval, port bit is again reconfigured as output.

3. **Skip ROM command** – Microcontroller then sends skip ROM command (code CC) on 1-wire bus.

4. **Write Scratchpad command** – Microcontroller then sends write scratchpad command (code 4E) to enable writing of upper and lower temperature setting (TH & TL) in scratchpad of DS 18S20.

5. **Temperature trigger points** – Microcontroller then sends 2 data bytes through 1-wire bus to scratchpad of 18S20 for temperature trigger settings.

6. **Reset Pulse** – Microcontroller issues reset pulse (>480µs) and reconfigures port bit as input.

7. **Presence Pulse** – DS 18S20 responds with a presence pulse. At the end of this interval, port bit is reconfigured as output.

8. **Skip ROM command** – Microcontroller then issues skip ROM command (code CC)

9. **Convert T command** – Microcontroller then issues convert temperature command (code 44h)

10. **Delay** – Microcontroller then waits for at least 750 ms. (During this period, sensing of other parameters is carried out). Timer based interrupt can also be used to provide this delay.

11. **Reset Pulse** – Microcontroller issues reset pulse (>480µs) and reconfigures port bit as input.

12. **Presence Pulse** – DS 18S20 responds with a presence pulse. At the end of this interval port bit is reconfigured as output.

13. **Skip ROM command** – Microcontroller then issues skip ROM command (code CC).

14. **Read scratchpad command** – Microcontroller then issues read scratchpad command (code BE).
15. Reading of 9 bytes – Here at the start of each bit interval, Microcontroller configures port bit as o/p and send ‘0’ for 1 µsec and then releases bus (port bit as input). It reads the entire scratchpad of DS 18S20 including its CRC byte. It then calculates CRC of first 8 bytes of scratchpad and compares with calculated CRC, if they do not match, microcontroller repeats reading operation once more. Then steps 6 to 15 are repeated to continuously read temperature till microcontroller receives new SMS command. In this case, microprocessor starts from step 1. It also sends SMS to user indicating fault due to overcurrent / bearing fault if temperature rises beyond the upper threshold after switching OFF the motor.

4.1.3 Dry Running Sensing:
For detection of dry running condition of induction motor pump, water level of well are sensed at five different levels. Lowest level indicates the point at which pump should be switched off.
Fig. 4.4 shows the dry running sensing circuit. Quad op-amp IC 324 is used as a comparator. Present level of water is read by microcontroller and during pump-on conditions micro-controller continuously checks the level of water through its port. If water level does not fall within specified time or if water level falls below level 1 (Foot valve), micro-controller switches OFF the pump and sends message to user cell phone indicating dry running condition. The rate of fall of water level after pump is switched ON depends upon the diameter of well, pump rating, suction pipe head and depth of well (rate of influx of water internally into well). A trial run is
carried out by switching ON the motor when there is sufficient water in the well (above Level 5) in circuit and noting the time of change of state of input ports to determine optimum time durations.

4.1.4 Soil Moisture Content Measurement:
The soil moisture content measurement circuit determines whether sufficient water has percolated through soil in the region under test.

Fig. 4.5 Soil Moisture Measurement Circuit

Fig. 4.5 shows the soil moisture measurement circuit. Two copper leads acts as sensor probes. They are immersed in the region where water is to be distributed. The conductivity of soil depends on the amount of moisture present which increases with water content in the soil. Transistor 2N2222 is used as current amplifier. Under dry condition of soil between the probes, there is no conduction path between two copper leads. In this case, voltage output of circuit is between 0 to 0.9 V. When water is added to the soil, it percolates through successive layers and spread across wider region due to capillary force. This causes increase in the moisture content of soil leading to rise in conductivity between the probes. Under this conditions when sufficient water has spread the output of the circuit varies from 2.0 to 3.4 V. For extremely drenched soil, the voltage may increase beyond 4V. The output of circuit is given to one of the analog input channels of microcontroller board. This circuit is used in surface based irrigation scheme to switch off the motor pump when optimum quantity of water has been distributed in the specified region and send message/ alarm to indicate the completion of work to the user.

4.1.5 Time Based Control
For sprinkle based irrigation scheme, time duration of on-state of pump is specified for water distribution depending on the type of soil. For timing functions, it is preferable to use Real Time Clock (RTC) chip with battery back-up to ensure that the pump is kept ON for specified time
interval even after intermittent power failure period. Serial RTC DS1307 is chosen for the application.

DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM manufactured by Dallas Semiconductor (DS 1307 Datasheet). Address and data are transferred serially through an $I^2C$™, bidirectional bus (Two Wire Interface - TWI). The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the battery supply.

![Block Diagram of DS1307](source: Dallas Semiconductor (Maxim) Data sheet)

![Interfacing of DS1307 with microcontroller](source: Dallas Semiconductor (Maxim) Data sheet)
Figure 4.6 shows internal block diagram of DS 1307 and Figure 4.7 shows it's interfacing to microcontroller system. Table 4.1 shows the addresses and formats of various internal NV SRAM locations.

Initially with the help of keyboard and LCD display, present date and time are written into corresponding internal memory locations of this IC using I²C protocol (Master Transmitter mode). This IC helps in time-keeping functions. Whenever on-time duration for pump is specified, the two variables are defined to store time duration in minutes and hours. The pump is switched on through relays using ports of µc and present time is read from internal locations of RTC serially by microcontroller at regular intervals using I²C slave transmitter mode. After passage of every minute (known through RTC) variables values are updated if normal conditions exists and when their values reach zero values, pump is switched off through relay using output port bit. Accurate time based operations can be carried out by regularly reading registers of this IC and power failure and fault durations (Motor OFF state duration) can be known and taken into account for optimum water distribution.

### 4.1.5.1 I²C Protocols

I²C bus must be controlled by a master device (micro-controller) that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave on the I²C bus. Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between START and STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit. The DS1307 operates in the standard mode (100 kHz) only.

### Table 4.1 Internal Locations of DS1307 (Source: Maxim Datasheet)

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>Bit7</th>
<th>Bit6</th>
<th>Bit5</th>
<th>Bit4</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0</th>
<th>FUNCTION</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>CH</td>
<td>10S</td>
<td>Secs</td>
<td>Secs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00–59</td>
</tr>
<tr>
<td>01H</td>
<td>0</td>
<td>10M</td>
<td>Min</td>
<td>Min</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00–59</td>
</tr>
<tr>
<td>02H</td>
<td>0</td>
<td>12</td>
<td>10H</td>
<td>10H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01–12</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>PM/AM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+AM/PM</td>
<td>00–23</td>
</tr>
<tr>
<td>03H</td>
<td>0</td>
<td>0</td>
<td>10D</td>
<td>10D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAY</td>
<td>01–07</td>
</tr>
<tr>
<td>04H</td>
<td>0</td>
<td>0</td>
<td>10D</td>
<td>10D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DAY</td>
<td>01–31</td>
</tr>
<tr>
<td>05H</td>
<td>0</td>
<td>0</td>
<td>10M</td>
<td>10M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Month</td>
<td>01–12</td>
</tr>
<tr>
<td>06H</td>
<td>10</td>
<td>Year</td>
<td>Year</td>
<td>Year</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00–99</td>
</tr>
<tr>
<td>07H</td>
<td>OUT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SOWE</td>
<td>0</td>
<td>0</td>
<td>RS1</td>
<td>RS0</td>
<td>Control—</td>
</tr>
<tr>
<td>08H-3FH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RAM 55 x 8</td>
</tr>
</tbody>
</table>
Fig. 4.8 Data Transfer on I^2C Serial Bus (Source: Dallas Semiconductor (Maxim) DS1307 Datasheet)

Fig. 4.9(a) Data Write - Slave Receiver Mode (Source: Dallas Semiconductor (Maxim) DS1307 Datasheet)

Fig. 4.9(b) Data Read - Slave Transmitter Mode (Source: Dallas Semiconductor (Maxim) DS1307 Datasheet)

Fig. 4.8 shows the typical I^2C bus transaction. Fig. 4.9 (a) and (b) show data transfer operation for data-write Slave Receiver Mode and data-read Slave Transmitter mode respectively.

Many microcontrollers have internal hardware register support for I^2C bus support. Figure 4.10 shows the overview of TWI (I^2C) module of AVR microcontroller ATmega32. This module contains group of registers which should be initialized properly for two-wire protocols.
Various Registers associated with TWI interface are described below:

**TWI Bit Rate Register – TWBR**: selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master mode.

**TWI Control Register – TWCR**: used to control the operation of the TWI. It is used to enable the TWI (TWEN bit), to initiate a Master access by applying a START condition to the bus (TWSTA bit), to generate a receiver acknowledge (TWEA bit), to generate a stop condition (TWSTO bit), and to control halting of the bus while the data to be written to the bus are written to the TWDR (TWWC bit). It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible. TWINT (TW Interrupt) Flag bit is set by the hardware when TWI has completed current task and expects application software response.

**TWI Status Register – TWSR**: Most significant five bits reflect the status of the TWI logic and the Two-wire Serial Bus.
TWI Data Register – TWDR: In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when TWINT is set by hardware.

TWI (Slave) Address Register – TWAR: should be loaded with the 7-bit Slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a Slave Transmitter or receiver.

The transaction sequence for writing data into DS 1307 registers using I2C interface with ATmega32 microcontroller as master is as follows:

1. Transmission of Start condition is initiated by setting TWEN, TWSTA and TWINT bits of TWCR register.
2. 5 most significant bits of TWSR are read and if there is match with 0x08 (Start Condition) then slave address write operation is carried out by sending 0xD0 (DS1307 address) to TWDR register and TWINT & TWEN bits of TWCR register are set.
3. 5 most significant bits of TWSR are read and if there is match with 0x18 (Slave Address – WR mode with acknowledgement) then address from which write operation is to be carried out is sent (00 for seconds register of DS1307) to TWDR register and TWINT & TWEN bits of TWCR register are set.
4. 5 most significant bits of TWSR are read and if there is match with 0x28 (Data Transmitted with acknowledgement) then present value of time in seconds is sent to TWDR register and TWINT & TWEN bits of TWCR register are set.
5. Step No. 4 is repeated six more times to write present values of minutes, hours, day, date, month and year into subsequent registers of DS1307.
6. Now STOP condition is initiated on the Two wire interface by setting TWSTO and TWEN bits of TWCR register. Thus process of writing into all registers of RTC DS1307 is completed.

The transaction sequence for reading data (present time) from RTC DS 1307 registers using I2C interface with ATmega32 microcontroller as master is as follows:

1. Transmission of Start condition is initiated by setting TWEN, TWSTA and TWINT bits of TWCR register.
2. 5 most significant bits of TWSR are read and if there is match with 0x08 (Start Condition) then slave address write operation is carried out by sending 0xD0 (DS1307 address) to TWDR register and TWINT & TWEN bits of TWCR register are set.

3. 5 most significant bits of TWSR are read and if there is match with 0x18 (Slave Address -WR mode with acknowledgement) then address from which read operation is to be carried out is sent (00 for seconds register of DS1307) to TWDR register and TWINT & TWEN bits of TWCR register are set.

4. 5 most significant bits of TWSR are read and if there is match with 0x28 (Data Transmitted with acknowledgement) then repeated start condition is initiated by setting TWEN, TWSTA and TWINT bits of TWCR register.

5. 5 most significant bits of TWSR are read and if there is match with 0x08 (Start Condition) then slave address read operation is carried out by sending 0xD1 (DS1307 address) to TWDR register and TWINT & TWEN bits of TWCR register are set.

6. 5 most significant bits of TWSR are read and if there is match with 0x40 (Slave address read mode with acknowledgement) then TWINT, TWEN and TWEA bits of TWCR are set.

7. 5 most significant bits of TWSR are read and if there is match with 0x50 (Data byte read with acknowledgement); then TWDR is read to receive present time in seconds from DS1307 seconds registers and TWINT, TWEN and TWEA bits of TWCR are set.

8. Step No. 7 is repeated six times to read values of minutes, hours, day, date, month and year and then STOP condition is initiated on the Two wire interface by setting TWSTO and TWEN bits of TWCR register. Thus process of reading all registers of RTC DS1307 is completed.

4.1.6 Voice Messaging & Control

As SMS response is non-deterministic and may vary depending on network traffic, an optional feature of voice messaging and control is provided for the application using Single-Chip Voice Recording & Playback Device IC APR 9600 and DTMF receiver IC M-8870. This approach is based on making voice call instead of messaging wherein microphone of system cell phone receives selected voice message from speaker connected to IC APR 9600 relating present state of system. Moreover, it also allows user to send commands in form of DTMF code by pressing
relevant number keys. DTMF receiver IC is connected to speaker of system cell phone to decode the received code and send it to microcontroller to carry out specified operation.

### 4.1.6.1 Voice Recording and Playback Circuit

Aplus Integrated Circuits Inc.’s APR 9600 device offers single chip voice recording, non-volatile storage and playback capability for 40 to 60 seconds. The device supports both random and sequential access of multiple messages.

![Block diagram APR 9600](image)

Fig 4.11 Block diagram APR 9600 (Source: Aplus Integrated Circuit Inc. APR 9600 Datasheet)

Fig 4.11 shows the block diagram of this chip. A differential microphone amplifier, including integrated AGC, is included. The amplified microphone signal is fed into the device by connecting the Ana_Out pin to the Ana_In pin through an external DC blocking capacitor. After anti-aliasing filtering, the signal is clocked into the memory array through Sample and Hold and Analog Write/Read circuits. When playback is desired the previously stored recording is retrieved from memory, low pass filtered, amplified and heard by connecting speaker to the SP+ and SP- pins. Chip management is accomplished through the device control block. Message management is controlled through the message control block.

In our application, APR 9600 is configured in random access mode and message selection pins are controlled through 3 to 8 line decoder from ports of microcontroller. This arrangement permits microcontroller to choose the voice message to be sent to speaker based on present status of the system. Initially APR 9600 IC is kept in record mode by grounding /RE pin and 8 different
voice messages are recorded through microphone to denote different conditions of monitoring system by selection of message pins. Now microphone is disconnected and device is moved to playback mode and depending on change in status, microprocessor sends appropriate code through 3 to 8 decoder to send voice message through speaker. Fig. 4.12 shows the interfacing of APR 9600 for the application.

**Fig. 4.12 Interfacing of IC APR 9600 with Microcontroller**

### 4.1.6.2 DTMF Receiver Circuit
Clare's M-8870 is a full DTMF Receiver that integrates both band split filter and decoder functions into a single package. Figure 4.13 shows the Block Diagram of M-8870 DTMF Receiver IC. M-8870 IC contain a band split filter that separates the high and low tones of the received pair, and a digital decoder that verifies both the frequency and duration of the received tones before passing the resulting 4-bit code to the output bus. When the detector recognizes the simultaneous presence of two valid tones, it raises the Early Steering flag (ESt). Any subsequent loss of signal condition will cause ESt to fall. If ESt signal remains high for validation period, after a short delay, the delayed steering output flag (StD) goes high, signaling that a received tone pair has been registered. The contents of the output latch are made available on the 4-bit output bus by raising the three state control input (OE) to logic high.
Fig. 4.14 shows the interfacing of DTMF receiver IC with microcontroller ports. Whenever DTMF tones are received through microphone, StD goes high after some delay and microcontroller can scan this output and whenever it is high, 4-bit BCD code corresponding to DTMF tones is read and necessary action can be initiated depending upon the value received.

4.2 Selection of Communication Protocols
Remote monitoring involves transfer of information between control system and user. The selection of communication protocols is predominantly dependent on the technical features of the selected system cell phone and choice of control system chosen. PC based control system results in greater flexibility as even USB based data link cables can be directly used due to availability of software drivers for conversion into serial COM ports. However, simple micro-controller based system have only RS232C Serial communication port and so additional hardware in form of converters have to be added for data link connection with cell phone. The most common method of communication is through serial port using data link cable through AT commands and other protocols. This method has been extended through USB-to-serial converters and Bluetooth.
serial adaptors to current era cell phones also. Many earlier Nokia cell phone models did not support AT commands. These models used to communicate using F-Bus protocols.

4.2.1 F-Bus Protocols

Nokia has developed its own proprietary F-Bus protocols which contain series of commands that allow users to make calls, send and receive SMS messages, and perform many other functions using PC. Number of researchers / engineers have probed the activity on serial bus and provided unofficial information about the list of commands through trials for Nokia cell phone (http://www.mygnokii.org). F-Bus i.e. Fast Bus is high-speed full duplex bus. It uses one pin for transmitting data and one pin for receiving data plus the ground pin. It is fast and works at 115.2 kbps, 8 data-bits, no parity and one stop bit. For setting the F-Bus operation, DTR pin should be set and RTS pin should be cleared. In F-Bus protocol, the information has to be sent in specific frame format from source (PC / mobile phone) and response/ acknowledgement be received in specific manner from destination (Mobile phone / PC) to indicate information is correctly received. This protocol was used for system cell phone models based on 3310, 3315 and 3120.

4.2.1.1 Basic Format for F-Bus (version 2)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameID</td>
<td>0x1E - F-Bus serial comm.</td>
</tr>
<tr>
<td>DestDev</td>
<td>0x00 – mobile phone</td>
</tr>
<tr>
<td>SrcDev</td>
<td>0x0C – TE (PC/microcontroller)</td>
</tr>
<tr>
<td>MsgType</td>
<td>Type of command</td>
</tr>
<tr>
<td>FrmeLen</td>
<td>{Block} + 2 (+1 if padding byte exists)</td>
</tr>
<tr>
<td>Framestogo</td>
<td>0x01 indicates last frame</td>
</tr>
<tr>
<td>Seq. No.</td>
<td>0xAB where A - 4/6 for first Block, 0 for continuing block; B – sequence No.</td>
</tr>
<tr>
<td>Paddingbyte</td>
<td>0x00 if FrmeLen is odd number otherwise does not exist</td>
</tr>
<tr>
<td>Chksum1</td>
<td>Exclusive ORing on frame’s odd numbers</td>
</tr>
<tr>
<td>Chksum2</td>
<td>Exclusive ORing on frame’s even numbers</td>
</tr>
</tbody>
</table>

Depending upon type of command to be carried out, MsgType and {Block} data parameters will vary. In our case, system needs to send SMS message relating device status / acknowledge incoming SMS, check SMS message for any valid command relating to control of devices and delete incoming SMS messages after interpretation to ensure sufficient SIM / Phonebook memory is available in mobile phone for future incoming SMS messages.

4.2.1.2 Acknowledgement Format for F-Bus (version 2)
4.2.1.3 Sending SMS using F-Bus

SMS messages have to be sent in specific format described by GSM 3.40 standard [10]. The MsgType for submitting SMS message is 0x02. The format of data block for SMS-submit in FBUS protocol is:

| 0x01 | 0x00 | 0x02 | 0x00 | sl | St | {sca} | fo | 0x00 | pi | dcs | ml | dl | dt | {da} | {vp} | {msg} |

Where
- **sl**: SMS Centre No. Length in octets including st
- **st**: SMS Centre No. Type
  - 0x81 – Normal, 0x91 – International, 0xA1 - National
- **sca**: SMS Centre No. packed in BCD format (10 byte reverse nibble format)
- **fo**: first octet of Transfer Protocol Data Unit (TPDU), holds flags such as validity period format, message type indicator, etc.
- **pi**: protocol id
- **dcs**: data coding scheme
- **ml**: message length in 7 bit character
- **dl**: Destination address length in semi octets
- **dt**: Destination address type (same as st)
- **da**: Destination number packed in BCD (same format as sca)
- **vp**: Validity period (1 integer type or 7 octets of vpf)
- **msg**: 7-bit character packed into bytes (GSM 3.38)

The best approach to decide pi, dcs, ml, vp and msg is to send SMS message from mobile to PC and debug the format corresponding to these fields and insert these values in the data block. This technique results in saving lot of time involved in making trials with various formats.

After sending the SMS-submit message from microcontroller to system cell phone, microcontroller will receive following acknowledgment of SMS-Submit frame:

| 0x1E | 0x0C | 0x00 | 0x7F | 0x00 | 0x02 | 0x1C | 0x72 |

After a short time, the system cell phone will inform with “SMS message sent” frame:

| 0x1E | 0x00 | 0x0C | 0x02 (MsgType) | 0x00 | FrmLen | 0x01 | 0x08 | 0x00 | 0x02 (Message sent) |

| 0x64 | 0x12 | 0x00 | 0x01 | Seq No. | [00] (padding byte, if it exists) | chksum1 | chksum2 |

Microcontroller can now send acknowledgment to this frame:

| 0x1E | 0x0C | 0x00 | 0x7F | 0x00 | 0x02 | 0x04 | 0x10 | 0x79 |

4.2.1.4 Receiving SMS messages

Whenever mobile phone receives SMS message, it sends a ‘SMS received frame’ in following format (similar to SMS Submit Frame)

| 0x10 | 0x00 | 0x02 (SMS Handlg) | MsgLen | 0x01 | 0x08 | 0x00 | 0x10 (Msg recd) | 0x02 (Mem typ) |
PC should send acknowledge frame

In order to ensure memory is free for future incoming messages, PC must delete message from mobile phone memory by sending delete message frame

4.2.2 AT Commands

AT (attention) commands were initially developed by Hayes to carry out communication between computers and modems. These commands have been extended by cell phone manufacturers to carry out specified tasks through serial communication interface with terminal equipment (TE) (ETSI TS 127 007 V5.3.0 (2003-03); AT Commands ..., 2006). However, cell phone manufacturers do not provide support to all AT Commands. Now-a-days extensive list of mobile AT commands are available for carrying out various activities like sending SMS, using GPRS services, sending fax, controlling speaker volume, battery status indication etc. AT commands are sent by sending text strings ‘A’, ‘T’, along with specified command strings through serial port to mobile and are executed on receipt of carriage return. The result codes are sent by mobile to TE indicate the status after execution of command. AT Commands supported for particular cell phone can be known by checking the response to specified AT commands with addition of character "?" instead of parameters. OK response indicates the support of specified AT command whereas ERROR code indicates non-support. Table 4.2 shows major AT Commands used in SMS Transmission and Reception.

| TABLE 4.2 MAJOR AT COMMANDS FOR SMS TRANSMISSION & RECEPTION (TEXT MODE) |
| AT Command | Function |
| AT + CMGF = 1 | Select Text mode for SMS |
| AT + CNMI = 2, 2, 0, 0, 0 | New message indications to TE with buffering on data mode, direct forwarding with unsolicited result code +CMT to TE without SMS-DELIVER status to TE |
| AT + CSDH = 1 | Show all headers in Text mode |
| AT + CSMP = 49, 167, 0, 0 | Used to select additional parameters for SMS sending |
| AT + CMGS = "9921237724" | sends message "Device 1 ON" from TE Through the network to cell phone No. 9921237724 |
| >Device 1 ON ^Z | |
Whenever SMS is received by cell phone with incoming message settings as referred to Table 4.2, unsolicited result code with message received by TE is shown below:

+CMT: "9921237724", "Vasif", "11/03/06, 22:10:00+01"
Device 1 Switched ON

It was observed that Nokia 6610 model supports all the AT commands related to SMS while later mobile models 3500 classic, 2700 classic, etc. do not support incoming message indication command (CNMI).

Table 4.3 shows list of major AT commands used with voice call (Miscall) establishment and detection.

**Table 4.3 Major AT Commands For Voice Call Establishment And Detection**

<table>
<thead>
<tr>
<th>AT Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATV1</td>
<td>Select Verbose response mode</td>
</tr>
<tr>
<td>AT + CLIP = 1</td>
<td>Enables called subscriber to get the calling line identity (CLI) of the calling party when receiving a mobile terminated call. Set command enables the presentation of the CLI at TE</td>
</tr>
<tr>
<td>AT + CLOP = 1</td>
<td>Enables calling subscriber to get connected line identity (COL) of called party after setting up a mobile originated call. Set command enables the presentation of the COL at the TE.</td>
</tr>
<tr>
<td>ATD 9921237724</td>
<td>Dial voice call to specified number 9921237724. The possible responses are OK/BUSY/NO CARRIER/NO ANSWER/ERROR</td>
</tr>
<tr>
<td>AT + CHUP</td>
<td>Execution command causes the TA to hang up the current GSM call of the Mobile Equipment</td>
</tr>
</tbody>
</table>

Whenever system cell phone initialized with above commands receives incoming call, unsolicited code RING is received along with following response

RING
+CLIP, "9921237724", 147

Similarly when system cell phone carries dialing command ATD <number>; following unsolicited result code is obtained on successful response from called party

+COLP, "", 147

### 4.2.3 Java ME application

Due to lack of complete support of AT commands in many recent mobile models, it was decided to use Java 2 Micro Edition (J2ME) application for messaging (James Keogh, 2003; www.forum.nokia.com/java). With J2ME, Sun has adapted the Java platform for consumer
products that incorporate or are based on small computing devices. J2ME defines elements for building complete Java runtime environments that meet the requirements for a broad range of devices. The Java ME platform has two defined subsets of libraries, called configurations: the more capable Connected Device Configuration (CDC) and the more restricted Connected Limited Device Configuration (CLDC), which is compliant with smaller memory requirements and Virtual Machine capabilities from the mobile device. On top of these configurations are profiles, which concentrate even further on a specific group of devices. Mobile devices use the Mobile Information Device Profile (MIDP), which adds class libraries specific to mobile devices such as cell phones and PDAs. Any embedded system with a CLDC configuration and MIDP profile can run applications written for any other platform with corresponding libraries. The core of the J2ME is the MIDlet; each application has its own MIDlet. Most of present cell phones are Java enabled, so programs can be developed in Java ME and this application in Javabyte code can be downloaded into cell phone (Files having .jar extension). Java ME provides APIs for supporting various activities like UI & graphics, wireless messaging, multimedia, networking etc. So it is possible to receive / send messages to other cell phones as well as communicate with PC using serial communication ports by writing Java based programs. However, the Java based applications are basically designed for user based inputs and display and so certain additional modifications are needed to automate the running of application. As signed certificates for higher authorization level are costly to implement, it was decided to control the pressing of the soft keys of user interface of cell phone through analog switches controlled by microcontroller ports. Eclipse version 3.2.2 with Java ME plug-in was used to develop program in Java ME. Sun Wireless Tool Kit for CLDC WTK2.2 was used for simulation and debugging of Java program and Nokia PC Suite software was used to download generated file with extension .jar into cell phone model.

4.2.3.1 Java ME MIDlet

The MIDlet forms the application framework that executes on CLDC devices under the Mobile Information Device Profile (MIDP). Every application must extend the MIDlet class found in the javax.microedition.midlet package. The application management software (AMS) manages the MIDlet itself. The AMS is a part of the device’s operating environment and guides the MIDlet through its various states during the execution process. When a MIDlet suite is installed on a
device, its classes, resource files, arguments, and persistent storage are kept on the device and ready for use.

Every MIDlet must have three methods:

1. **startApp method**: is called when the MIDlet is started. After startApp, the MIDlet is in *Active* state and the AMS allows it to hold resources. If a runtime exception occurs during startApp, the MIDlet will be destroyed immediately.

2. **pauseApp method**: Even if pauseApp() is not called, it still needs to be included in S60 or Series 40 MIDP implementations

3. **destroyApp method**: destroyApp signals the MIDlet to terminate and places it in Destroyed state. During termination, all the MIDlet's resources are released and objects deleted. The MIDlet has five seconds to handle the destroyApp call, after which the AMS closes the application itself.

MIDP 2.0 compliant devices implement a push registry that allows inactive MIDlets to be launched in response to external events, such as received SMS messages, or previously set timers. With Push, MIDlets can in a sense be run like services on a mobile device to handle certain tasks without the user's explicit interaction. While the MIDlet is active, the MIDlet itself is responsible for dealing with any push events that may occur. When the AMS detects an incoming connection or an alarm based activation associated with a MIDlet, the AMS starts the MIDlet and calls the `startApp()` method. At this point the MIDlet takes over responsibility for the connection and performs the steps necessary for handling the incoming connection.

### 4.2.3.2 Java ME Messaging

The messaging API is based on the Generic Connection Framework (GCF), which is defined in CLDC1.0 specification. The package `javax.microedition.io` defines the framework and supports input/output and networking functionality in J2ME profiles. Messaging provides the notion of opening a connection based on a string address and that the connection can be opened in either client or server mode. The interfaces for the messaging API have been defined in the `javax.wireless.messaging` package.

A message can be thought of as having an address part and a data part. A message is represented by a class that implements the interface defined for messages in the API. This interface provides methods that are common for all messages. In the `javax.wireless.messaging` package, the
base interface that is implemented by all messages is named Message. It provides methods for addresses and timestamps.

For the data part of the message, the API is designed to handle text, binary and multipart messages. These are represented by three subinterfaces of Message: TextMessage, BinaryMessage and MultipartMessage. These sub interfaces provide ways to manipulate the payload of the message as Strings, byte arrays and message parts, respectively.

Message sending and receiving functionality is implemented by a Connection interface, in this case, MessageConnection. To make a connection, the application obtains an object implementing the MessageConnection from the Connector class by providing a URL connection string that identifies the address.

If the application specifies a full destination address that defines a recipient to the Connector, it gets a MessageConnection that works in a "client" mode. This kind of Connection can only be used for sending messages to the address specified when creating it.

The application can create a "server" mode MessageConnection by providing a URL connection string that includes only an identifier that specifies the messages intended to be received by this application. Then it can use this MessageConnection object for receiving and sending messages. The format of the URL connection string that identifies the address is specific to the messaging protocol used.

For sending messages, the MessageConnection object provides factory methods for creating Message objects. For receiving messages, the MessageConnection supports an event listener-based receiving mechanism, in addition to a synchronous blocking receive() method.

The system cell phone is configured as server with port ID 3333 for receiving and sending messages. The major aspect of the source code for receiving incoming message is shown below

```java
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet. *
import javax.wireless.messaging.*;
public class SMSReceive extends MIDlet implements CommandListener, Runnable,
MessageListener {
Thread thread;
String[] connections;
boolean done;
boolean mesrec= false;
String smsPort;
MessageConnection smsconn;
Message msg;
String senderAddress;
```
public SMSReceive() {
    smsPort = getAppProperty("SMS-Port");
}

public void startApp() {
    String smsConnection = "sms://:" + smsPort;
    if (smsconn == null) {
        try {
            smsconn = (MessageConnection) Connector.open(smsConnection);
            smsconn.setMessageListener(this);
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }

    connections = PushRegistry.listConnections(true);
    done = false;
    thread = new Thread(this);
    thread.start();
}

public void notifyIncomingMessage(MessageConnection conn) {
    if (thread == null) {
        done = false;
        thread = new Thread(this);
        thread.start();
    }
}

public void run() {
    /** Check for sms connection. */
    try {
        msg = smsconn.receive();
        if (msg != null) {
            mesrec = true;
            senderAddress = msg.getAddress();
            content.setTitle("From: " + senderAddress);
            if (msg instanceof TextMessage) {
                content.setString(((TextMessage) msg).getPayloadText());
                TextMessage tmsg = (TextMessage) msg;
                String msgText = tmsg.getPayloadText();
                System.out.println("Message text: " + msgText);
                System.out.println("Message has been received successfully! Ok.");
            } else {
                System.out.println("Message not a Text message");
            }
        } catch (IOException e) {
            // e.printStackTrace();
        }
    }
}

public void pauseApp() {
    done = true;
    mesrec = false;
    thread = null;
    resumeScreen = display.getCurrent();
}
public void destroyApp(boolean unconditional) {
    done = true;
    thread = null;
    mesrec = false;
    if (smsconn != null) {
        try {
            smsconn.close();
            System.out.println("closing application");
        } catch (IOException e) {
            // Ignore any errors on shutdown
        }
    }
}

Here system.out.print function directly causes the string in argument to be sent serially to PC/
Microcontroller through COM port to which data cable of cell phone is connected. Push registry
initialization in JAD file allows launching of this MIDlet by AMS in response to incoming SMS
message. JAD file for this midlet is shown below

MIDlet-1: SMSReceive,, SMSReceive
MIDlet-Jar-URL: vamarvcr.jar
MIDlet-Permissions-Opt: javax.microedition.io.PushRegistry,
javax.microedition.io.Connector.sms, javax.wireless.messaging.sms.receive
MicroEdition-Configuration: CLDC-1.1
MIDlet-Push-1: sms://:3333, SMSReceive, *
MIDlet-Version: 1.0.0
MIDlet-Name: vamarvcr Midlet Suite
MIDlet-Vendor: Midlet Suite Vendor
MicroEdition-Profile: MIDP-2.0
SMS-Port: 3333

In order to ensure that incoming message from user cell phone is accepted by our application
instead of being sent to inbox, application must be developed in user cell phone which will
directly address server ID port (in our case 3333) of system cell phone. The relevant portion of
source code for this application on user cell phone is shown below

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.wireless.messaging.*;

public class Smssend extends MIDlet implements CommandListener {
    private Display display;
    private TextField textMesg;
    private Command send1Command;
    private String serverPort;
    public Smssend () throws Exception {
        display = Display.getDisplay(this);
        send1Command = new Command("SEND1", Command.SCREEN, 1);
        serverPort = getAppProperty("serverPort");
    }
    public void startApp() {
        try {
            displayBlankForm ();
            System.out.println("Start Midlet for sending");

```
} catch (Exception e) {
    System.out.println("Error in start");
    e.printStackTrace();
}

public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}
public void commandAction(Command command, Displayable screen) {
    if (command == send1Command) {
        try {
            String addr = "sms://9764996778:" + serverPort;
            MessageConnection conn = MessageConnection.Connector.open(addr);
            TextMessage msg = (TextMessage) conn.newMessage(MessageConnection.TEXT_MESSAGE);
            msg.setPayloadText(textMesg.getString());
            conn.send(msg);
            conn.close();
            destroyApp(false);
            notifyDestroyed();
        } catch (Exception e) {
            System.out.println("Error in sending");
            e.printStackTrace();
        }
    }
}

private void displayBlankForm() throws Exception {
    Form form = new Form("mesg_send");
    textMesg = new TextField("Message", ",", 100, TextField.ANY);
    form.append(textMesg);
    form.addCommand(send1Command);
    form.setCommandListener((CommandListener) this);
    display.setCurrent(form);
}

The JAD file associated with this midlet is shown

MIDlet-Version: 1.0.0
MIDlet-Vendor: Midlet Suite Vendor
MIDlet-Jar-URL: vawma_n1.jar
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0
MIDlet-1: Smssend,,vawma_1.Smssend
serverPort: 3333
MIDlet-Name: vawma_n1 Midlet Suite
Fig. 4.15 Screen shots of execution of SMS sending MIDlet (a) Display of Blank form (b) Message entry (c) Activation of Send1 command results in asking of permission for sending of message by AMS

Fig. 4.16 Screen shots of SMS Reception Midlet (a) Display of Incoming message alert by AMS (b) Activation of Start command results in display of received message on display and sending of message to microcontroller through serial com port.

Fig. 4.15 shows the screen shots of SMS sending MIDlet from user cell phone where initially Blank Display form is shown in fig. 4.15(a) and relevant message can be typed by user. Fig. 4.15(b) shows message "Device 2 OFF" typed by user and pressing of central key results in execution of Send1 command which causes messaging API to be called for sending of message to system cell phone. AMS asks permission of user to finally transmit the message as shown in Fig. 4.15(c). Fig. 4.16 shows the screen shots of system cell phone where fig (a) shows incoming message alert by AMS due to push registry initialization. However AMS waits for response from central key of mobile 'start' command. Only after response is obtained, the MIDlet is launched and incoming message is displayed on the screen as shown in Fig 4.16(b) and sent to microcontroller through data cable connected to COM port.

Remote monitoring application requires automated response on system cell phone, so there is need to develop a mechanism which will result in automated pressing of central key whenever incoming message is received. It is observed that whenever incoming message is received, AMS provide audio alert in form of standard message tone. This alert tone was recorded for three Nokia Models 6681, 3500 classic and 2700 classic and frequency analysis was carried out to determine dominant tones.
Fig. 4.17(a) Plot of standard message tone for Nokia Model 6681

Fig. 4.17(b) Frequency Response of Standard Message Tone for Nokia Model 6681
Fig. 4.17(c) Plot of standard message tone for Nokia 3500 classic

Fig. 4.17(d) Frequency Response of Standard Message Tone for Nokia 3500 classic
From Fig. 4.17, it is observed that dominant frequency for standard message tone is 1860 Hz for Nokia 6681 model, 1550 Hz for Nokia 3500 classic and 1750 Hz for Nokia 2700 classic models respectively.
Figure 4.18 shows the Block Diagram of SMS reception MIDlet activation circuit. The microphone is coupled to speaker of system cell phone. Active Band pass filter is second-order Butterworth filter implemented using LF351 IC consisting of two op-amps with choice of components based on lower cut-off frequency of 1300 Hz and higher cut-off frequency of 2000 Hz. Tone decoder is implemented using NE567 IC and free-running frequency is set by external components based on cell phone model chosen. CD4066 IC is used as analog controlled switch. Whenever SMS is received by cell phone, AMS generates audio standard message alert tone which is picked by microphone and passed through band pass filter circuit to Tone decoder. Tone decoder output goes low due to input frequency tone match with its free running frequency. Low input on interrupt pin of microcontroller causes interrupt service routine to be executed which causes output port pin to be active for specified duration. This causes closing of central switch of cell phone resulting in activation of SMS MIDlet which results in reception of message and communication of message to microcontroller through serial COM port.

4.2.4 Bluetooth Serial Port Adaptor:

Simple embedded systems with RS232 serial connection cannot be directly connected with modern cell phones having USB based cables. USB-to-RS232C converters can be used. However, Bluetooth based connection can offer greater flexibility and mobility at lower cost. Bluetooth can be considered cable replacement technology and has been developed by consortium of companies including Ericsson, IBM, Intel, Nokia, Toshiba, etc. to provide royalty free, open specification for short range wireless connectivity (Cano et al., 2005, Sairam et al., 2002, Haartsen et al., Shepherd R, 2001). It is radio-frequency technology that uses 2.4 GHz Industrial-Scientific-Medical (ISM) band. For providing Bluetooth connectivity to the
microcontroller system, Bluetooth Serial Port Adaptor (SPA) from Connect Blue cB-OEMSPA312 was chosen.

![Image of Bluetooth SPA](http://www.connectblue.com)

Connect Blue's OEM SPA are based on the PCB cB-0902. Fig. 4.19 shows the image of Bluetooth SPA chosen in our work. cB-0902 is a small size Bluetooth module based on the Phillips BGB203 system in package (SiP). The BGB203 has on chip SRAM and FLASH stacked in the same package. The modules are available in many variants with different antenna / connectors and output power combinations. The cB-0902 has an RS232 interface.

![Block Diagram of cB-0902](http://www.connectblue.com)
Fig. 4.20 shows the Block Diagram of cB-902 PCB. In our case, Power Amplifier and Co-axial connector for External Antenna is not used and Bluetooth range is limited to 100m (Class B). The presence of four connectors permits SPA to be connected in different ways. We have used connector J1 for interfacing RS232 and power signals.

**Table 4.4 Signals on J1 Connector (Source: cB E_M_Datasheet_OemSPA)**

<table>
<thead>
<tr>
<th>Pin Nr</th>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>VSS</td>
<td>GND</td>
</tr>
<tr>
<td>3-4</td>
<td>VCC_3V3</td>
<td>3.0 to 6.0V DC</td>
</tr>
<tr>
<td>5</td>
<td>RS232-CTS</td>
<td>h/w flow control(i/p)</td>
</tr>
<tr>
<td>6</td>
<td>RS232-TxD</td>
<td>Transmit Data (o/p)</td>
</tr>
<tr>
<td>7</td>
<td>RS232-RTS</td>
<td>h/w flow control(o/p)</td>
</tr>
<tr>
<td>8</td>
<td>RS232-RxD</td>
<td>Receive Data (i/p)</td>
</tr>
<tr>
<td>9</td>
<td>RS232-DTR</td>
<td>Data Terminal Ready(o/p)</td>
</tr>
<tr>
<td>10</td>
<td>RS232-DSR</td>
<td>Data Set Ready(i/p)</td>
</tr>
<tr>
<td>11</td>
<td>RED/Mode</td>
<td>Multiplexed. i/p(mode) selects RS232/UART mode during 500ms after start-up. o/p (RED) logic signal for RED LED</td>
</tr>
<tr>
<td>12</td>
<td>Switch-0</td>
<td>i/p used for &quot;connect on external signal” function</td>
</tr>
<tr>
<td>13</td>
<td>GREEN/</td>
<td>Multiplexed. i/p(mode) '0' causes system to move to default serial settings</td>
</tr>
<tr>
<td></td>
<td>Switch-1</td>
<td>during 500ms after start-up. o/p (GREEN) logic signal for GREEN LED</td>
</tr>
<tr>
<td>14</td>
<td>BLUE</td>
<td>o/p (BLUE) logic signal for BLUE RED</td>
</tr>
<tr>
<td>15-20</td>
<td>UART signals (CTS to DSR)</td>
<td>Not used in our application</td>
</tr>
</tbody>
</table>

Table 4.4 shows the pins of J1 connector of SPA along with their functional description. Bluetooth SPA being optimized for size does not contain LEDs. However, for testing and debugging, it is important to have indication of present mode of operation. J1 connector contains pins as logic signals for three LEDs for this purpose. It also provides both UART and RS232 connection signals for serial communication.
Fig. 4.21 Signal sensing, conditioning & Power supply circuit for SPA (Source: http://www.connectblue.com)

Fig. 4.21 shows the arrangement for LED indications for mode, switches for default settings (reset) and serial connection on external signal and power supply connections to SPA through J1 connector. The popular quad comparator IC LM339 is used to translate low level signals to TTL levels and provide sufficient current driving capabilities for LEDs. Table 4.5 shows the signal states in different modes of operation of SPA with corresponding ON/OFF status of the three LEDs.

**TABLE 4.5 SIGNALS STATES IN DIFFERENT SPA MODES**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Green LED</th>
<th>Blue LED</th>
<th>Red LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Mode</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>AT Mode</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
</tr>
</tbody>
</table>
Fig. 4.22 shows the image of designed PCB based on the circuit in Fig. 4.20 along with 9-pin D connector for RS232 COM port connection. 20-pin 2mm pitch female FRC connector was provided on PCB for connection to SPA through J1 connector.

Bluetooth SPA cB-OEMSPA312 is connected to AVR Microcontroller board through RS232C serial interface. The serial communication parameters are 57.6 kbps, 8 data bits, no parity, one stop bit and no flow control mode. SPA works in data and AT modes and needs to be properly configured before it can start data communication with other Bluetooth devices. On power-on condition, SPA is initially in data mode and by sending “///” characters within 3 seconds, the device can be moved into AT mode for configuration. In AT mode, series of commands can be sent for proper configuration.

Table 4.6 gives list of major AT commands used for configuring SPA. During first time usage, commands needs to be sent for writing local device name and reading BT address of SPA. Later on, only device discovery needs to be performed using AT*AGDD command and SPA can directly connect to serial service by using connect to serial data service command AT*ADCP with BT address of device to which it wants connection. If match is found, it can start data communication between micro-controller system and specified cell phone using AT*ADDM command. Serial communication parameters can be optionally changed using AT*AMRS command.

<table>
<thead>
<tr>
<th>AT Command</th>
<th>Description</th>
<th>Response</th>
</tr>
</thead>
</table>

**TABLE 4.6 MAJOR AT CONFIGURATION COMMANDS FOR BT-SPA**
AT*AGLN="Bluetooth SPA BNCOE", 1
Write Local Device Name 'Bluetooth SPA BNCOE' and store in non-volatile memory

OK or ERROR

AT*AIlba?
Read BT address of local device

*AILBA: <bd_addr>, OK or ERROR

AT*AGDD=40,2
Performs device discovery for 50 secs

*AGDD: <no_of_devices>
*AGDDE:<bd_addr>, <cod>, <device_name_valid>,<device_name> (for each device)

AT*ADCP=<bd_addr>, 0, 0, 0
Connect to a serial service enabled on a remote device using SPP, automatic service search and master/slave role decided by remote device

ADCP:<connection_handle> OK

AT*ADCC=<connection_handle>
Close existing data mode connection

OK or ERROR

AT*ADDM
Request the Serial Port Adapter to move to data mode.

OK or ERROR

SPA is configured in Bluetooth Serial Port Profile and provided with BT address of system cell phone and then moved back to data mode. Now SPA scans for Bluetooth enabled devices in the neighborhood and checks if the discovered device address matches with its stored BT address. If match is found, it can start data communication between microcontroller system and specified cell phone.

4.3 Selection of Microcontroller
The selection of microcontroller depends on computational speed, power consumption, memory requirements, available on-chip peripherals, cost and application’s specific requirements. There are plenty of popular families of microcontrollers.

4.3.1 89C51/89C52 Microcontroller:
89c51/89c52 is very popular 8-bit microcontroller initially manufactured by Intel and then licensed to many other microcontroller manufacturers. It has 4K /8K bytes of in-system programmable flash, 256 bytes of internal RAM, 2/3 16-bit Counter /Timers, 32 I/O ports, 5/6 vector two-level interrupt architecture and full duplex serial port. It provides facility to interface external program and data memory of maximum 64 KB. However in this case, number of ports available for General Purpose I/O is drastically reduced. In addition, this µc supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode...
saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset. Fig. 4.23 shows the Block Diagram for this microcontroller.

Fig. 4.23 Block Diagram of 89C51/52 Microcontroller (Source: Atmel 89C51 Datasheet)

However, non-availability of internal ADC, I^2C interface, SPI interface and inability to work at 57.6 kpbs baud rate with 12.00 MHz crystal resulted in rejection of this device in our application.

4.3.2 MSP430F2013 Microcontroller:
MSP430F2013 Microcontroller is a 16 bit ultra low power mixed signal microcontroller from Texas Instruments. It has 2k Flash, 128 bytes of RAM, watchdog timer, a 16-bit Sigma Delta
Analog-to-Digital converter, a 16-bit timer, brownout detector, a USI module supporting SPI and I²C, and five low power modes drawing as little as 0.1 μA standby, etc. with JTAG debugging and emulation support. MSP430 family of processors' is becoming popular in WSN applications where motes are intended to run for long intervals > 1 year on battery power. Fig. 4.24 shows the functional Block Diagram of MSP430F2013 microcontroller.

![Functional Block Diagram of MSP430F2013 Microcontroller](Source: www.ti.com/msp430)

The eZ430-F2013 MSP430 development tool in form of USB stick and supporting software was used to develop application using MSP 430F2013 microcontroller. eZ430- F2013 uses the IAR Embedded Workbench Integrated Development Environment (IDE) to provide full emulation with the option of designing a stand-alone system or detaching the removable target board to integrate into an existing design. The USB port provides power to operate the ultra-low-power MSP430 so no external power supply is required. All 14-pins on the MSP430F2013 are accessible on the MSP-EZ430D target board for easy debugging and interfacing with peripherals. Additionally, one of the digital I/O pins is connected to an LED for visual feedback. Fig. 4.25 shows the image of eZ430 development tool.
Because of its powerful features, humidity and temperature measurement application was developed using this microcontroller. However, it was difficult to develop an efficient C program for application using only 2K of flash. Moreover, application envisaged uses battery power only during power supply failure period. So low power consumption is not the essential requirement for the system. Therefore, this microcontroller was not used further.

4.3.3 ATmega32 Microcontroller:

ATmega32 is part of AVR series of microcontroller manufactured by Atmel with 8-bit RISC architecture, 32 kB of in-system programmable Flash, 1k E2PROM, 2k SRAM, 32-bit multi-function General purpose I/O, 8 channel 10-bit ADC, Two 8-bit Timer / Counters with Separate Prescalers and Compare Modes, One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode, TWI, USART, SPI, JTAG interface support and six software selectable low power modes to reduce power consumption (URL: http://www.atmel.com/dyn/resources/prod_documents/2503S.pdf, Barnett et al., 2007). The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. Atmel provides free software support in form of AVR ‘C’ compiler, AVR Studio for software debugging and
Simulation in assembly language and Ponyprog software is also available for flash programming. Functional Block Diagram of AVR ATmega32 is shown in Fig. 4.26.

The ATmega32 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs from the pins of Port A. The device also supports 16 differential voltage input combinations with programmable gains on some inputs. The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown.
in Figure 4.27. The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX register. ADMUX register also contains bits for selecting internal or external reference voltage and right or left justified format for result. The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted.

Fig. 4.27 AVR ATmega32 ADC Block Diagram (Source: Atmel ATmega32 Datasheet)
A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR (Fig. 4.28). Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished (free running mode).

The successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. The ADC module contains a prescaler, which generates an acceptable ADC clock frequency. The prescaling is set by the ADPS bits in ADCSRA. Fig. 4.29 shows the Block Diagram of ATmega32 ADC prescalar circuit.

![Auto Trigger Logic Diagram](Source: Atmel ATmega32 Datasheet)
So, ADC conversion using ATmega32 Microcontroller requires proper initialization of internal ADC registers. A sample C code for our application for reading analog voltage from ADC0 pin is shown below.

```c
#include <string.h>
#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>

int main(void)
{
    unsigned char adc_pin=0; // channel 0 select
    int k;
    long int temp;
    union _adc_data {
        unsigned char dt[2]; // unsigned short data;
    } adc_data;
    ADMUX = (0x40+adc_pin); //Select channel 0 (single-ended) & external AVCC
    for(;;)
    {
        ADCSR = 0xC0; //Control Word for enabling ADC, activating SOC & clk divisor of 128
        loop_until_bit_is_set (ADCSR, ADIF);
        //Wait For EOC or while(ADIF==0)
        adc_data.dt[0] = ADCL; //full 8bit
        //Read Lower Byte
        adc_data.dt[1] = ADCH; //lsb 2
        //Read Higher Byte
        full=adc_data.data & 0x03ff;
        temp=(long)full*500;
    }
}
```

Fig. 4.29 AVR ATmega32 ADC Prescalar Circuit (Source: Atmel ATmega32 Datasheet)
temp/=1024;
k=(int)temp;
/* functions for displaying digital value on LCD
   Comparing with threshold values, sending message
   Based on comparison etc.*/
delay1(10);
}

ATmega32 supports serial communication through USART. The simplified block diagram of
USART is shown in Fig. 3.30. It consists of clock generator, transmitter and receiver sections.
The clock generation logic generates the base clock for the Transmitter and Receiver. The
UMSEL bit in UCSRC selects between asynchronous and synchronous operation. Double Speed
(Asynchronous mode only) is controlled by the U2X found in the UCSRA Register. UBRR is
used as a programmable prescaler or baud rate generator. The down-counter, running at system
clock (f_{osc}), is loaded with the UBRR value each time the counter has counted down to zero or
when the UBRRL Register is written. A clock is generated each time the counter reaches zero.
This clock is the baud rate generator clock output (= f_{osc}/(UBRR+1)). The Transmitter divides
the baud rate generator clock output by 2, 8 or 16 depending on mode.
USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization. TXC Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. TXC Flag must be cleared before each transmission. It is preferable to use external crystal of frequency 1.8432 MHz to ensure baud rate generation without any error.
The generalized code for serial communication using 57.6K baud rate through serial port is shown.

```c
void uart_init()
{
    UBRRH = 0x0;
    UBRRL = 0x01; // crystal 1.8432MHZ baud rate = 57.6kbps
    UCSRB=0x0D8; // enable TX RX & enable TX & RX Complete INTs
    UCSRC = 0x86; // asynchronous mode with 8-bit data mode
    sei();       // set global interrupt enable
    delay(1000);
}
```

INTERRUPT(SIG_UART_TRANS)
{
    TXC = 0;
    // send message through UDR To COM port
}

INTERRUPT(SIG_UART_RECV)
{
    // RECEIVE message from COM port through UDR
}

Most of the work was carried out on this microcontroller due to presence of several desirable features and flash of sufficient capacity for application development.

### 4.3.4 LPC2148 Microcontroller:

LPC2148 microcontroller from Philips (now ‘NXP’) semiconductors is based on a 16-bit/32-bit ARM7TDMI core with flash memory of 512 kB, on chip S-RAM of 40 kB, Serial communications interfaces ranging from a USB 2.0 Full-speed device, multiple UARTs, SPI, SSP to I2C. Various 32-bit timers, RTC, single or dual 10-bit ADCs, 10-bit DAC, PWM channels and 45 fast GPIO lines with up to nine edge or level sensitive external interrupt pins make these microcontrollers suitable for industrial control and medical systems. The ARM7TDMI-S is a general purpose 32-bit RISC microprocessor with high performance and very low power consumption. RISC approach results in a high instruction throughput and impressive real-time interrupt response. Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The ARM7TDMI-S processor also employs a unique architectural strategy known as Thumb, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue. The Thumb set’s 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM’s performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because
Thumb code operates on the same 32-bit register set as ARM code. Fig. 4.31 shows the Functional Block Diagram of LPC 2148 microcontroller.

![Fig. 4.31 NXP's LPC 2148 Microcontroller (Source: NXP LPC2148 Datasheet)](image)

It was observed that USB interface cannot directly used for connectivity with Nokia models as drivers are available only for PC with Windows OS and need tremendous software efforts for implementation on non-OS based embedded systems. Since the application envisaged requires less computation efforts it was decided to continue with ATmega32 micro-controller for the work.