Report on

SECURE POSITION TRACKING SYSTEM

Project Report submitted in partial fulfilment of course on Embedded Systems (EC 411)

Under Guidance of Dr. Aparna P

Prakash Pali 11EC67
Ankit Shah 11EC86
Shubham Yadav 11EC92

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL
SRINIVASNAGAR 575025 KARNATAKA INDIA

November 2013

Table of Contents

INTRODUCTION

Data Logging is an important part in remote monitoring of a system and based on the data collected one can deduce suitable conclusion for a particular application. We have designed a secure position tracking system, where a code word is checked to communicate the data and this data is accessible to only one user who owns the device. Since the object under consideration is continuously monitored so we will attach the transmitter with the object that will continuously transmit the co-ordinates in terms of latitude and longitude of the object received by the receiver via GSM module. We are displaying the list of coordinates of the object with time and date on the LCD display. We will keep an update of time and date with GPS system.

This data includes date, time and position (latitude and longitude) is also possible to process for obtaining the speed of the object too. This system contains single-board embedded system that is equipped with GPS receiver and GSM modem along with ARM micro-controller (LPC2148). The data received by GPS is read, processed, analysed and stored in the memory of the micro-controller , Based on the code word if the user sends a message for example position to the SIM card present inside the GSM module.

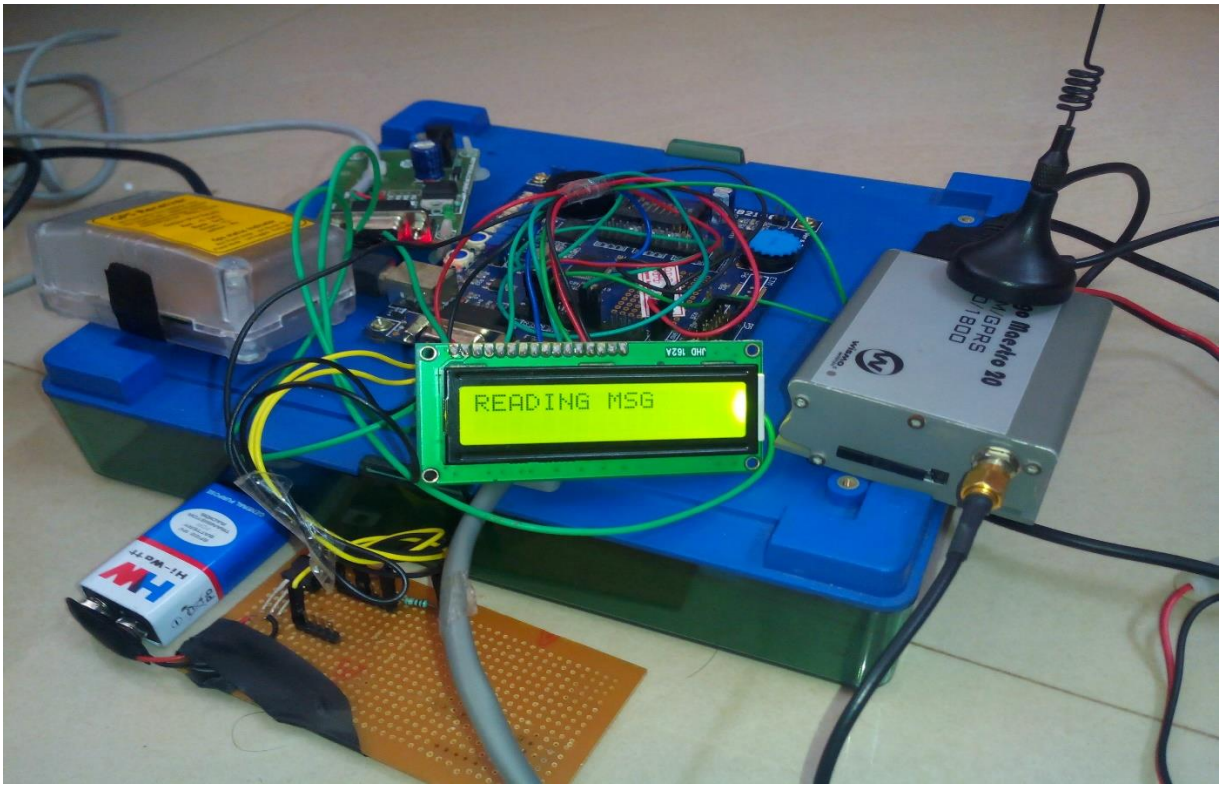

Figure 1 : Our Complete System.

Features of LPC 2148

The LPC2141/42/44/46/48 microcontrollers are based on a 16-bit/32-bit ARM7TDMI-S CPU with real-time emulation and embedded trace support that combine the microcontroller with embedded high-speed flash memory ranging from 32 kB to 512 kB.

A 128-bit wide memory interface and a unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty. Due to their tiny size and low power consumption, LPC2141/42/44/46/48 are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. Serial communications interfaces ranging from a USB 2.0 Full-speed device, multiple UARTs, SPI, SSP to I2C-bus and on-chip SRAM of 8 kB up to 40 kB, make these devices very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, single or dual 10-bit ADC(s), 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.

Key features

- 16-bit/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
- 8 kB to 40 kB of on-chip static RAM and 32 kB to 512 kB of on-chip flash memory.
- 128-bit wide interface/accelerator enables high-speed 60 MHz operation.
- In-System Programming/In-Application Programming (ISP/IAP) via on-chip bootloader software. Single flash sector or full chip erase in 400 ms and programming of 256B in 1 ms.
- Embedded ICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high-speed tracing of instruction execution.
- USB 2.0 Full-speed compliant device controller with 2 kB of endpoint RAM.
- Addition, the LPC2146/48 provides 8 kB of on-chip RAM accessible to USB by DMA.
- One or two (LPC2141/42 vs. LPC2144/46/48) 10-bit ADCs provide a total of 6/14 analog inputs, with conversion times as low as 2.44 µs per channel.
- Single 10-bit DAC provides variable analog output.
- Two 32-bit timers/external event counters (with four capture and four compare channels each), PWM unit (six outputs) and watchdog.

GSM (Global System for Mobile Communication)

GSM (Global System for Mobile Communications, is a standard set developed by the European Telecommunications Standards Institute (ETSI) to describe technologies for second generation (2G) digital cellular networks

GSM networks operate in a number of different carrier frequency ranges (separated into GSM frequency ranges for 2G and UMTS frequency bands for 3G), with most 2G GSM networks operating in the 900 MHz or 1800 MHz bands.
Each band available for the system is divided into channels with bandwidth of 200 kHz. For the GSM 900 system there are 124 available channels (separate for the uplink and downlink direction), and for the GSM 1800 374 channels, where these bands were already allocated, the 850 MHz and 1900 MHz bands were used instead (for example in Canada and the United States).
In GSM system three basic subsystems can be distinguished:

1. Base station subsystem (BSS)
2. Core Network (CN)
3. User Equipment (UE)

· Between particular elements of the system the interfaces are defined
Base station subsystem – includes system of base stations and their controllers
· Base station provide optimum radio coverage of a given area and communicates with user equipment over air interface
· The operation of the base station subsystem is controlled by the base station controller (BSC)
· This manages radio resources allocation, controls the setting-up of calls, gathers results of measurements carried out by base station and mobile station
· The BSC is also responsible for power control and handover control
· Interface A enables the BSS system to be connected to mobile switching centre (MSC)
· Interface connects BSS with packet switching element GSM system – architecture 3/6

Figure 2: Fargo Maestro GSM Module

The main elements of the core network are:

- mobile switching centre (MSC)
- visitor's location register (VLR)
- home location register (HLR)
- authentication centre (AUC)
- equipment identification register (EIR)
- serving GPRS support node (SGSN)
- gateway GPRS support node (GGSN)

Subscriber Identity Module, commonly known as a **SIM card**. The SIM is a detachable smart card containing the user's subscription information and phone book. This allows the user to retain his or her information after switching handsets.

Security

GSM is designed with a moderate level of service security. It uses several cryptographic algorithms for security. The A5/1 and A5/2 stream ciphers are used for ensuring over-the-air voice privacy. A5/1 was developed first and is a stronger algorithm used within Europe and the United States; A5/2 is weaker and used in other countries. Serious weaknesses is found in both algorithms: it is possible to break A5/2 in real-time with a cipher text-only attack, and in January 2007, The Hacker's Choice started the A5/1 cracking project with plans to use FPGAs that allow A5/1 to be broken with a rainbow table attack.

GSM Modem SIM900 V7.03

The GSM modem is a specialized type of modem which accepts a SIM card operates on a subscriber's mobile number over a network, just like a cellular phone. GSM Modem is RS232-logic level compatible, i.e., it takes-3v to -15v as logic high and +3v to +15 as logic low.MAX232 is used to convert TTL into RS232 logic level converter used between the microcontroller and the GSM board. The signal at pin 11 of the microcontroller is sent to the GSM modem through pin 11 of max232.this signal is received at pin2 (RX) of the GSM modem. The GSM modem transmits the signal from pin3 (TX) to the microcontroller through MAX232, which is received at pin 10 of IC1.

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 possible 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.

AT COMMANDS

The "AT" or "at" prefix must be set at the beginning of each command line. To terminate a command line enter <CR>

The following are the AT commands used in our code to configure GSM module:

1. ATE
    a. ATE is used to SET COMMAND ECHO MODE.
    b. We have set echo mode off by initialising to ATE0 so that commands work in normal mode.
2. AT+CSDH

    This command shows the TEXT Mode Parameters.

    AT+CSDH=1 Sets it to show the output in TEXT mode. This is particularly necessary as we are using only test mode.

Note: - GSM module SIM 900 supports SMS message sending in two mode that is TEXT mode and PDU (Protocol Data Unit) mode.

3. AT + CMGR

    This command is used to read message from a particular location in the storage memory of the SIM card

    AT+CMGR =1 sets the GSM module to read message from position 1 in the memory storage area

4. AT + CMGD

    This Command used to delete message from a particular location from the memory of SIM Card.

    Ex AT+CMGD =1 deletes the message from position 1 of the SIM card

5. AT+ CMGS

    It is used to write a message to the receiver that needs to be send. After typing the message, we need to send “Control + Z “in order to send the message that is done with the ASCII value of it that is “0x1A” in the code.

6. AT+CNMI

    Used for indicating a new message.

GPS (Global Positioning System)

An embedded GPS application requires two support processes:

- reception and buffering of raw strings from the GPS receiver and,
- parsing select elements from target GPS strings as required by the application.

A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above the Earth. Each satellite continually transmits messages that include

- The time the message was transmitted
- Precise orbital information (the ephemeris)
- The general system health and rough orbits of all GPS satellites (the almanac).

The receiver uses the messages it receives to determine the transit time of each message and computes the distance to each satellite. These distances along with the satellites' locations are used with the possible aid of trilateration, depending on which algorithm is used, to compute the position of the receiver. This position is displayed, perhaps with a moving map display or latitude and longitude; elevation information may be included. Many GPS units show derived information such as direction and speed, calculated from position changes.

Three satellites might seem enough to solve for position since space has three dimensions and a position near the Earth's surface can be assumed. However, even a very small clock error multiplied by the very large speed of light— the speed at which satellite signals propagate — results in a large positional error. Therefore, receivers use four or more satellites to solve for both the receiver's location and time. The very accurately computed time is effectively hidden by most GPS applications, which use only the location. A few specialized GPS applications do however use the time; these include time transfer, traffic signal timing, and synchronization of cell phone base stations.

Although four satellites are required for normal operation, fewer apply in special cases. If one variable is already known, a receiver can determine its position using only three satellites. For example, a ship or aircraft may have known elevation. Some GPS receivers may use additional clues or assumptions (such as reusing the last known altitude, dead reckoning, inertial navigation, or including information from the vehicle computer) to give a less accurate (degraded) position when fewer than four satellites are visible.

GPS RECEIVER

The hardware interfaces for GPS units are designed to meet NMEA requirements. The GPS receiver provides data in NMEA 0183 format with a 1Hz update rate. Generally message received by GPS is in NMEA [National Marine Electronics Association} message format and NMEA protocol which is most commonly used is NMEA0183 protocol. GPS sentences beginning with the following specifications: $GPGGA, $GPGSA, $GPGSV, $GPRMC, and $GPVTG.

Sentence ID	Description
$GPGGA	GPS Fix Date
$ GPGSA	GPS Dilution of Precision and active satellites
$GPGSV	GPS Satellite in view
$GPRMC	Recommended minimum specific GPS/Transit data
$GPVTG	Track made good and ground speed
$GPMSS	Beacon Receiver status
$GPZDA	UTC Date/Time and Local time Zone Offset

Table 1 : Various Formats of GPS DATA

Figure 3:GPS Module from NSK Electronics

LIQUID CRYSTAL DISPLAY

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on.

A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.
The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD. Click to learn more about internal structure of a LCD.

Features:

- 16 characters wide, 2 rows.
- White text on blue background.
- Connection port is 0.1" pitch, single row for easy breadboarding and wiring.
- Pins are documented on the back of the LCD to assist in wiring it up
- Single LED backlight included can be dimmed easily with a resistor or PWM and uses much less power than LCD with EL (electroluminescent) backlights
- Can be fully controlled with only 6 digital lines!
- Built in character set supports most English/European/Japanese text, see the HD44780 datasheet for the full character set.
- Up to 8 extra characters can be created for custom glyphs or 'foreign' language support
- Comes with strip of header pins.
- Can adjust contrast with the addition of a potentiometer (not included).

Figure 4: 16x2 LCD Display.

Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; through a variable resistor	VEE
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight Vcc (5V)	Led+
16	Backlight Ground (0V)	Led-

Table 2 LCD Pins and its Functionality

UART

UART (Universal Asynchronous Receive Transmitter) is an asynchronous serial communication in which we serially transmit and receive data. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires

A UART usually contains the following components:

- A clock generator, usually a multiple of the bit rate to allow sampling in the middle of a bit period.
- input and output shift registers
- transmit/receive control
- read/write control logic
- transmit/receive buffers (optional)
- parallel data bus buffer (optional)
- First-in, first-out (FIFO) buffer memory (optional)

The LPC24xx devices currently have two on-chip UARTS. Both have a built-in baud rate generator and 16 byte transmit and receive FIFOs. The putchar () and getchar functions are used to read/write a single character to the UART. Both the putchar() and getchar() functions read the Link Status Register (LSR) to check on UART error conditions and to check the status of the receive and transmit FIFO

HARDWARE DESIGN

Hardware framework for tracking system is shown. System contains high Performance ARM controller, a GPS, and GSM modem. A tracking system will provide effective real time vehicle location reporting .Tracking system will inform where your vehicle is and where it has been, how longer it has been there. The basic function of in vehicle unite is to acquire, Monitor and transmit the position latitude, longitude, time to management centre either at fixed interval or on demand. Microcontroller unit form the heart of tracking unit, which acquires and process the position data from the GPS module. The GPS receiver of vehicle terminal receives and resolves the navigation message broadcasted by GPS position satellites, computes the longitude and latitude of vehicle coordinates, transforms it into the GSM message form by GSM Communication controller and sends the message to monitoring centre via the GSM network.

DEBUGGING AND TESTING PROCESS

Steps we followed in the debugging process:-

1. First, we got the readings on the Keil debugger of the locations of a sample value of co-ordinates by reading those values through UART in the code.
2. After this, we learnt on how GPS works and understood about the NMEA protocol for locating the data of latitude and longitude.
3. To get this data on the UART terminal of Keil our next step was to check whether GPS is working or not (by working we mean changing values of location in real time).
4. After successful testing of GPS we burnt the modified code on the LPC board to view the same results by connecting the GPS to one of the UART.
5. Then once the GPS data is available in the terminal we had to interface an LCD screen, as it would make the system easier to debug and user friendly to detect any errors in the device.
6. So getting two parts of the projects running we learnt about GSM module and how GSM works in detail and about the AT commands to configure GSM module.
7. For the AT commands we need to check whether a message is received and whether the message is in the correct location.
8. To continuous keep receiving the data we are using poling technique in our code, as the data needs to be checked continuously if GSM is in the text mode for receiving the message.
9. To identify the user, mobile number is stored in a buffer from the data received through the GPS and the code word is to be checked after verification of the user.
10. Longitude and Latitude data is send to the user by the command generated from the GSM module.

HOW WE HAVE MADE THE SYSTEM SECURE?

1. We have used an identifier in our code to identify the owner by his/her mobile number.
2. The owner has to enter mobile number for the first use.
3. Now the owner will be able to communicate with the device with a set of encrypted code words to retrieve data from the GPS Receiver about the position and different locations in previous five set time intervals.
4. Our system is not accessible through any other mobile number to retrieve information.

ADVANTAGES

- Easy to operate.
- Simple design.
- Isolate both GPS and GSM signal.
- Simple and reliable design with secure communication.
- Remote Communication using GSM Modem.
- Sends location information in the form of Latitude and Longitude.
- Reliable for Remote Tracking.

FUTURE IMPROVEMENTS

- Many features like time, velocity, direction of the system, altitude of the system are possible to implement.
- We can interface Google maps with our system.
- Velocity constraints and location constraints would give higher functionality.
- The Packaging of the complete system is reducible by using miniaturised components for portable applications.

LCD interfacing code:

```
#include    "clcd.h"
#include     string.h"
#include    "config.h"

//**********  Macro Definitions*******************
#define _SET_RS()          CTRL_PORT_SET |= ((unsigned long)(1)<<CTRL_RS)
#define _SET_EN()          CTRL_PORT_SET |= ((unsigned long)(1)<<CTRL_EN)
#define _CLEAR_RS()        CTRL_PORT_CLR |= ((unsigned long)(1)<<CTRL_RS)
#define _CLEAR_EN()        CTRL_PORT_CLR |= ((unsigned long)(1)<<CTRL_EN)
#define _EnToggle()        {_SET_EN();__asm{nop;nop;}_CLEAR_EN();}
#define _TYPE_CMD           0
#define _TYPE_DATA          1
//**********  End of Macro Definitions*************

volatile unsigned char uSDelay;
void _Itoa(int n, char *s);
void _Reverse(char *s);
void _ClcdDelayMs(unsigned int delay);
void _ClcdDelay45Us(void);

void ClcdInit(void)
{
   DATA_DIR |= (unsigned long)(DATA_PORT);            //initialize D4:D7 pins as output
   CTRL_DIR |= ((unsigned long)(1)<<CTRL_RS);         //initialize RS pins as output
   CTRL_DIR |= ((unsigned long)(1)<<CTRL_EN);         //initialize EN pins as output
   _CLEAR_EN();                                       //clear EN
   _CLEAR_RS();                                       //clear RS
   _ClcdDelayMs(300);
   ClcdSendByte(0X03,0);                               //Configure bus width as 8-bit
   _ClcdDelayMs(50);
   ClcdSendByte(0X02,0);                              //Configure bus width as 4-bit, 1 line,5X7 dots
   _ClcdDelayMs(50);
   ClcdSendByte(0X28,0);                             //Configure bus width as 4-bit, 2 line,5X7 dots
   _ClcdDelayMs(50);
   ClcdSendByte(0X10,0);                             //Cursor move and Shift to left
   _ClcdDelayMs(1);
   ClcdSendByte(0x0D,0);                              // DisplayOn,CursorOff
   _ClcdDelayMs(1);
   ClcdSendByte(0x06,0);                             // EntryMode,Automatic Increment - No Display
shift.
   _ClcdDelayMs(1);
   ClcdSendByte(0x01,0);                            //Clear Display and set address DDRAM with 0X00
   _ClcdDelayMs(5);
}

void ClcdSendByte(unsigned char byte,unsigned char type)
{
   _CLEAR_EN();
   if(type==_TYPE_DATA)
         {
      _SET_RS();      // Selects data Register for read / write
         }
   else if(type==_TYPE_CMD)
         {
      _CLEAR_RS();    // Selects cmd  Register for write
         }
```

```
  _ClcdDelay45Us();

   DATA_PORT_CLR |= DATA_PORT;                                    //Clear Data Port
   DATA_PORT_SET |= ((((unsigned long)byte >> 4) & 0x0F)<<D4);    //Send byte to Data port
   _EnToggle();
   _ClcdDelay45Us();
   DATA_PORT_CLR |= DATA_PORT;                                    //Clear Data Port
   DATA_PORT_SET |= (((unsigned long)byte & 0x0F)<<D4);           //Send byte to Data port
   _EnToggle();
   _ClcdDelay45Us();

}

void ClcdPutS(void *str)
{
   unsigned char *str1 = (unsigned char*) str;
   while(*str1)                                                   //Check for valid character
         {
             ClcdSendByte(*str1++,_TYPE_DATA);                    //Send out the current byte pointed to
         }
}

void ClcdPutS_P(const char *str)
{
   while(*str)                                                    //Check for valid character
         {
      ClcdSendByte(*str++,_TYPE_DATA);                          //Send out the current byte pointed to
          }
}

void _Reverse(char *s)
{
   unsigned char i, j;
   char c;

   for (i = 0, j = strlen(s)-1; i<j; i++, j--)
   {
      c = s[i];
      s[i] = s[j];
      s[j] = c;
   }
}

void ClcdGoto(unsigned char x,unsigned char y)
{
   switch(y)
         {
      case 1:
          y=0x80 ;                                              // position for line 1
         break;

      case 2:
          y=0xC0 ;                                              // position for line 2
          break;
     case 3:
        y=0x94 ;                                                // position for line 3
        break;

      case 4:
        y=0xD4 ;                                                 // position for line 4
        break;
```

```
        default:
        break;
          }
   ClcdSendByte((x-1+y),_TYPE_CMD);
}

void ClcdClear()
{
   ClcdSendByte(0x01,_TYPE_CMD);                    //Clear Display and set address DDRAM with
0X00
   _ClcdDelayMs(2);                                 // delay after sending clear command
}

void _ClcdDelayMs(unsigned int count)
{
   volatile unsigned int j,k;
   for (j=0;j<count;j++)
        {
      for (k=0;k<6000;k++)
                {
                 __asm
                        {
                            nop;
                             nop;
                         }
                }
        }
}

void _ClcdDelay45Us(void)
{
        volatile unsigned int k;
        for(k=0;k<409;k++)
        {
        __asm
                {
                   nop;
                }
        }
```

Initialisation Code:

```
#include "config.h"
void Uart0Init (void)                              // Initialize Serial Interface
{
  PINSEL0 |= 0x00000005;                            //Enable RxD0 and TxD0
   U0LCR = 0x83;                                    // 8 bits, no Parity, 1 Stop bit
   U0DLL = 97;                                     // 9600 Baud Rate @ 15MHz VPB Clock
   U0LCR = 0x03;                                   // DLAB = 0
}
void Uart0PutCh (unsigned char ch)                // Write character to Serial Port
{
  while (!(U0LSR & 0x20));
  U0THR = ch;
}
        unsigned char Uart0GetCh (void)           // Read character from Serial Port
```

```
        {
         while ((U0LSR&0x20) == 0);
        return (U0RBR);
        }
        void Uart1Init (void)                    // Initialize Serial Interface
        {
         PINSEL0 |= 0x00050000;                  //Enable RxD1 and TxD1
         U1LCR = 0x83;                            // 8 bits, no Parity, 1 Stop bit
         U1DLL = 97;                              // 9600 Baud Rate @ 15MHz VPB Clock
         U1LCR = 0x03;                            // DLAB = 0
        }
        void Uart1PutCh (unsigned char ch)       // Write character to Serial Port
        {
        U1THR = ch;
        while ((U1LSR&0x20) == 0);
        }
unsigned char Uart1GetCh (void)          // Read character from Serial Port
{
  while (!(U1LSR & 0x01));
  return (U1RBR);
}
        void  Uart0PutS(unsigned char *str)      //A function to send a string on UART0
{
   while(*str)
   {
      Uart0PutCh(*str++);
   }
}
void  Uart1PutS(unsigned char *str)              //A function to send a string on UART1
{
   while(*str)
   {
      Uart1PutCh(*str++);
   }
```

MAIN CODE:

```
    void _DelayMs(unsigned int);
    unsigned char GsmSendMsg(unsigned
    char *msgStr);
    unsigned char GsmReadMsg(unsigned
    char *msgstr);
    void _DelayMs(unsigned int count);
    void reset();
    int handle_byte(int bytegps);

    int counter1=0;
    int counter2=0;
    int bytegps;
    char cmd[7]="$GPRMC";
    char lng[25]=" Long:";
    char lat[7]="Lat:";
    int offsets[13];
    char buffer[300]="";
    char data1[20]="1300.3835";
    char data2[20]="07447.2656";
    char unuser[50]="Unauthorised User
    no : ";
    char lastfive[100]="";
    int offset;
    int i=0,j=0,cp=0;
    unsigned char msg;
    int length=0;
    int k=200;
    int a=0,b=7,c=7;
    int count0a=0;
    int countcomma=0;
    int count_lf=0;
    unsigned char
    usrNumStr[20]="\"+919035220425\"";/
    /Receiptent Mobile Number
    unsigned char msgno='1';
    unsigned char rcmsg[150]="";
    unsigned char rcmsg1[20]="";
    unsigned char rcmsg2[20]="";

    unsigned char
    msgformat[20]="position";//codeword-
    1
unsigned char
msgformat1[20]="lastfive";//codeword-2

    int main(void)
    {

    IODIR1|=0xFFFF0000;

//Turn Off the LEDs on board
IOPIN1&=~ (0xFFFF0000);

    Uart0Init();
            Uart1Init();
    ClcdInit();
    ClcdCursorOff();
    ClcdPutS("REAL TIME POSITN");
    ClcdGoto(1,2);
    ClcdPutS("    TRACKING!    ");
    _DelayMs(500);
    ClcdClear();
    ClcdPutS(" GETTING READY");
            _DelayMs(500);
            ClcdClear();
    ClcdGoto(1,2);
    ClcdPutS("     DONE!!");

    // Echo off
    Uart1PutS("ATE0\r");
    _DelayMs(500);

    //  For getting sms length during
sms read
    Uart1PutS("AT+CSDH=1\r");
    _DelayMs(500);


            while(1)
            {
              bytegps=Uart0GetCh();
                    if
(!handle_byte(bytegps))
     reset();
            }
  }


void _DelayMs(unsigned int count)
{
    volatile unsigned int j,k;
    for (j=0;j<count;j++)
            {
       for (k=0;k<6000;k++)
                    {
            __asm
                                {
               nop;
               nop;
```

```
            }
        }
    }
}


unsigned char GsmSendMsg(unsigned
char *msgStr)
{
                ClcdClear();
                ClcdPutS("SENDING
MSG");
                _DelayMs(500);


        Uart1PutS("AT+CMGS=");
    Uart1PutS(usrNumStr);
    Uart1PutCh('\r');
    _DelayMs(100);
    Uart1PutS(msgStr);
    _DelayMs(100);
    Uart1PutCh(0x1A);
    _DelayMs(500);

        Uart1PutS("AT+CMGD=1\r");
    _DelayMs(500);
          Uart1PutCh(0XD);
                ClcdClear();
                ClcdPutS("MSG SENT");
                _DelayMs(500);
                ClcdClear();
                ClcdPutS("DELETING
MSG");
                _DelayMs(500);
                ClcdClear();

        Uart1PutS("AT+CMGD=1\r");
                _DelayMs(500);
                Uart1PutCh(0XD);
                reset();
                        return 1;

}


void reset()
{
        counter1=0;
        counter2=0;
        a=0;
        b=7;
        c=7;
        k=200;
        i=0;
        j=0;
        return;
}

int get_size(int offset)
{
return offsets[offset+1]-offsets[offset]-
1;
}

int handle_byte( int bytegps)
{       char lat[7]="Lat:";
        char lng[25]="Long:";
        char unuser[50]="Unauthorised
User no : ";

buffer[counter1]=bytegps;
counter1++;


        if(counter1==300)
{
return 0;
}

if (bytegps==',')
{
counter2++;
offsets[counter2]=counter1;

if (counter2==13)
{
return 0;
}
}//if bytegps==',' ends


if (bytegps=='*')
{

offsets[12]=counter1;
}//if bytegps=='*' ends


if (bytegps==0x0a)
{
if(counter2!=12 || (get_size(0)!=6))
{
return 0;
}
```

```
for(j=0;j<6;j++)
{
if (buffer[j] != cmd[j])
{
return 0;
}
}

//second case in if bytegps==''
if (get_size(3) != 9)
{

return 0;
}


//third case in if bytegps==''
if (get_size(5)!=10)
{
return 0;
}


for (j=0;j<9;j++)
{
data1[j]=buffer[offsets[3]+j];
}

for (j=0; j<10;j++)
{
data2[j]=buffer[offsets[5]+j];

}
ClcdClear();
ClcdGoto(1,1);
ClcdPutS("Lat:");
ClcdPutS(data1);
ClcdGoto(1,2);
ClcdPutS("Long:");
ClcdPutS(data2);
_DelayMs(1000);
ClcdClear();
count_lf++;
if(count_lf/30==1)
{
strcat(lastfive,lat);
strcat(lastfive,data1);
strcat(lastfive,lng);
strcat(lastfive,data2);
if (count_lf==150)
{count_lf=0;
memset(lastfive,0,strlen(lastfive));
}
}

GsmReadMsg("1");

if (a==15 & b==15)
{
strcat(lat,data1);
strcat(lat,lng);
strcat(lat,data2);
GsmSendMsg(lat);
ClcdClear();
ClcdPutS("RESETTING");
_DelayMs(500);
ClcdClear();
memset(lat,0,strlen(lat));
}

if(a!=15 & b==15)
{
strcat(unuser,rcmsg1);
GsmSendMsg(unuser);
memset(unuser,0,strlen(unuser));
}
if(a==15 & c==15)
{
GsmSendMsg(lastfive);

}

return 0;
}
return 1;
}



unsigned char GsmReadMsg(unsigned
char *msgstr)
{   int counter0a=0;

ClcdClear();
ClcdPutS("READING MSG");
DelayMs(500);
Uart1PutS("AT+CMGR=");
Uart1PutCh(msgno);
_DelayMs(500);
Uart1PutCh(0XD);


i=0;
k=200;
count0a=0;
```

```
while(k>0)
{
msg=Uart1GetCh();
rcmsg[i]=msg;
i++;
if (msg==0x0a)
{
count0a++;
}
if (count0a==5)
{k=0;
count0a=0;
}
if (count0a==2)
if (i<50)
{
ClcdClear();
ClcdPutS("ERROR");
_DelayMs(100);
count0a=0;
return;
}
k--;
}


length=strlen(rcmsg);
j=0;
countcomma=0;
for(i=0;i<length;i++)
{                    Uart0PutCh(rcmsg[i]);
if(rcmsg[i]>96 & rcmsg[i]<123)
{
rcmsg2[j]=rcmsg[i];
j++;
}

if (rcmsg[i]==44)
countcomma=countcomma+1;

if(countcomma==0)
{
for(k=0;k<15;k++)
rcmsg1[k]=rcmsg[i+k+2];
}
}

_DelayMs(500);
ClcdClear();
ClcdGoto(1,1);
ClcdPutS("RECEIVED MSG:");
ClcdGoto(1,2);
ClcdPutS(rcmsg2);
_DelayMs(1000);

ClcdClear();
ClcdGoto(1,1);
ClcdPutS("SENDERS NO:");
ClcdGoto(1,2);
ClcdPutS(rcmsg1);
_DelayMs(1000);

a=0;
for(j=0;j<15;j++)
{
if(usrNumStr[j]==rcmsg1[j])
a++;
}
b=7;
length=strlen(rcmsg2);
for(i=0;i<length;i++)
{
if(msgformat[i]==rcmsg2[i])
b++;
}
c=7;
for(i=0;i<length;i++)
{
if(msgformat1[i]==rcmsg2[i])
c++;
}
memset(rcmsg1,0,strlen(rcmsg1));
memset(rcmsg2,0,strlen(rcmsg2));

_DelayMs(30);  // timepass
return 1;
}
```

REFERENCES:

[1] "GSM Enhanced GPS Based Vehicle Tracking System" in 2nd *National Conference in Intelligent Computing & Communication.*

[2] "GPS – GSM Based Tracking System" in *International Journal of Engineering Trends and Technology- Volume3Issue2- 2012.*

[3] Fargo Maestro 20 User Guide *from Fargo Telecom.*

[4] User Manual of LPC214x from *NXP Semiconductors.*

[5] SIM 900 AT Command SET from *WaveCom*.

[6] www.wikipedia.com/GSM

[7] www.wikipedia.com/GPS

[8] www.embeddedlaboratory.com