GPS-GSM BASED VEHICULAR TRACKING SYSTEM

Srikar Devulapalli

INDEX

Abstract:

Initially the GPS continuously takes input data from the satellite and stores the values of the latitude and longitude values in the LPC2148 micro controller. If we have to track the vehicle, we need to send a message to the device by which it gets activated. Once if the GSM gets activated it takes the last received latitude and longitude position values from the buffer and sends the message back to our mobile phone which has been predefined int e program. Once the message has been sent GSM gets deactivated and the GPS gets activated. After we received the message as the latitude and longitudinal values insert the in the google maps or google search we can get the exact location up to 10 meters.

INTRODUCTION

Tracking System is used for the observing of persons or objects on the move and supplying a timely ordered sequence of respective location data to a model.
Vehicular tracking system has given very good improved security systems to the vehicles. This hardware is fitted on the vehicle in a manner it is not visible to anyone. When the vehicle has been stolen, the location data can be used to find or trace the vehicle can be informed to cops for further action. It can be even being used to detect the unauthorized movements in the vehicle and alert the owner.
When a request has been sent by the user is sent to the number in the module the system automatically sends a return reply to the particular mobile indicating the position of the altitude and longitude. A program has been used to determine exact location of vehicle.
Vehicle Tracking features;
It is mainly benefited for the companies which are based on transport system. This tracking system can store the whole data where the vehicle stopped, how much time it takes to reach destination, and all these systems are used for data capture, data storage and data analysis.

Few Applications of the Tracking systems in Real world:
Fleet management:
This uses the GPS (Global Positioning system) and identifies the position trucks location forms the map of all vehicle positions in the city.
Car Navigation
Attendance Management system

In this Project we have used
1.GSM MODULE
2.GPS MODULE
3.LPC2148 Microcontroller
4.MAX 232
5.LCD DISPLAY

WORKING METHODOLOGY

The project model consists of GPS receiver and GSM module with micro controller LPC2148 controller and this whole system is fitted in the vehicle. In the other end we will have a mobile phone. So the GPS system will send the longitudinal and latitude values corresponding to the position of the vehicle to the GSM module.

Imagine the bus left Bengaluru at 4'0 clock in the morning. If the in charge of the vehicle wants to know where the vehicle is at 10'o clock. We will send the message BUSTRACK "BUS NUMBER". This SMS sent would come through the GSM Service provider and then reach the vehicle which is travelling, because the vehicle has a GSM device with a sim card. This modem will receive the SMS and send to the microcontroller in the vehicle. The microcontroller will receive the message and compare the message and command. If everything Matches, then it will send a message back informing the latitude and longitude.

The GSM receiver in the office receives the data & gives to the microcontroller through the serial port.

DETAILED
DESCRIPTION OF THE COMPONENTS
USED

Microcontroller LPC2148

A controller which uses Arm7[Advanced Risk Machine] Processor.
LPC means Low Power Consumption

Specifications of LPC2148:

Operating volatage:3.3. volts

ASM1117 It converts 5 volts to 3 volts

Operating Frequency:12Mhz

Memory size:RAM:40KB

ROM:32-512KB

UARTS: UART 0 and UART1

TIMERS: TIMER1 and TIMER2

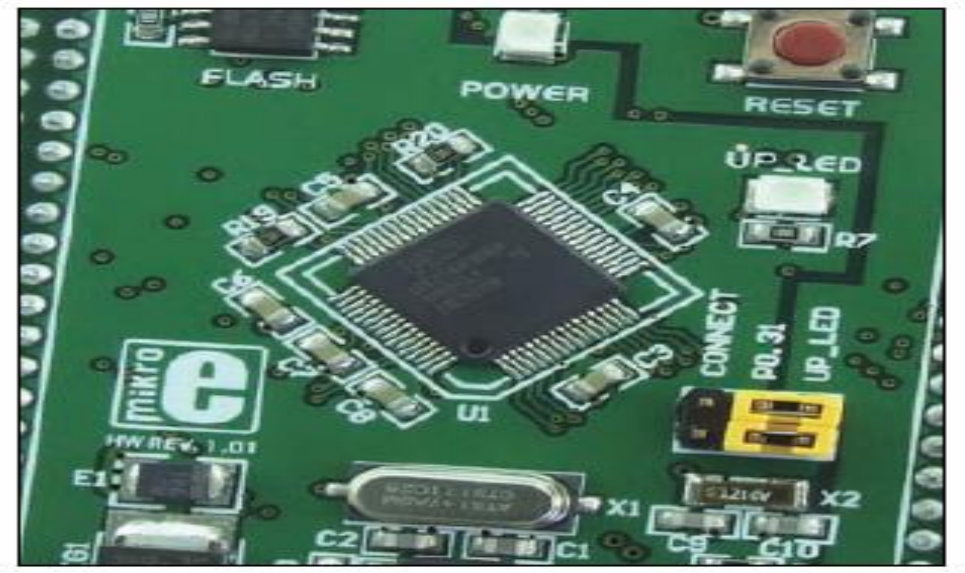
WATCHDOGTIMER-1

It has got a 2 10-bit A-D converter and 1 10-bit D-A converter

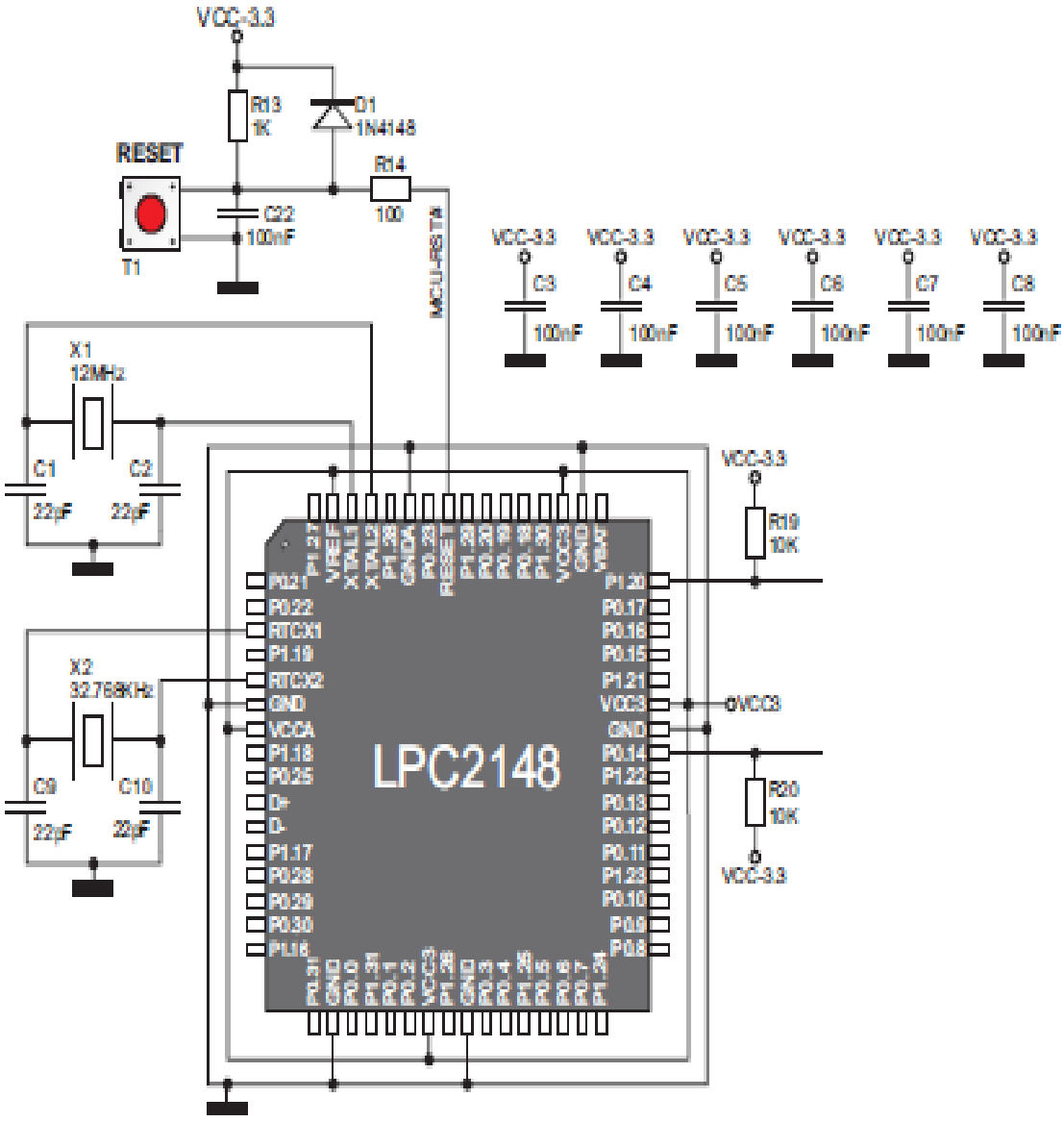
Low Power Real time clock with independent power and 32KHz clock input

USB 2.0 full speed compliant device controller with 2KB of endpoint RAM

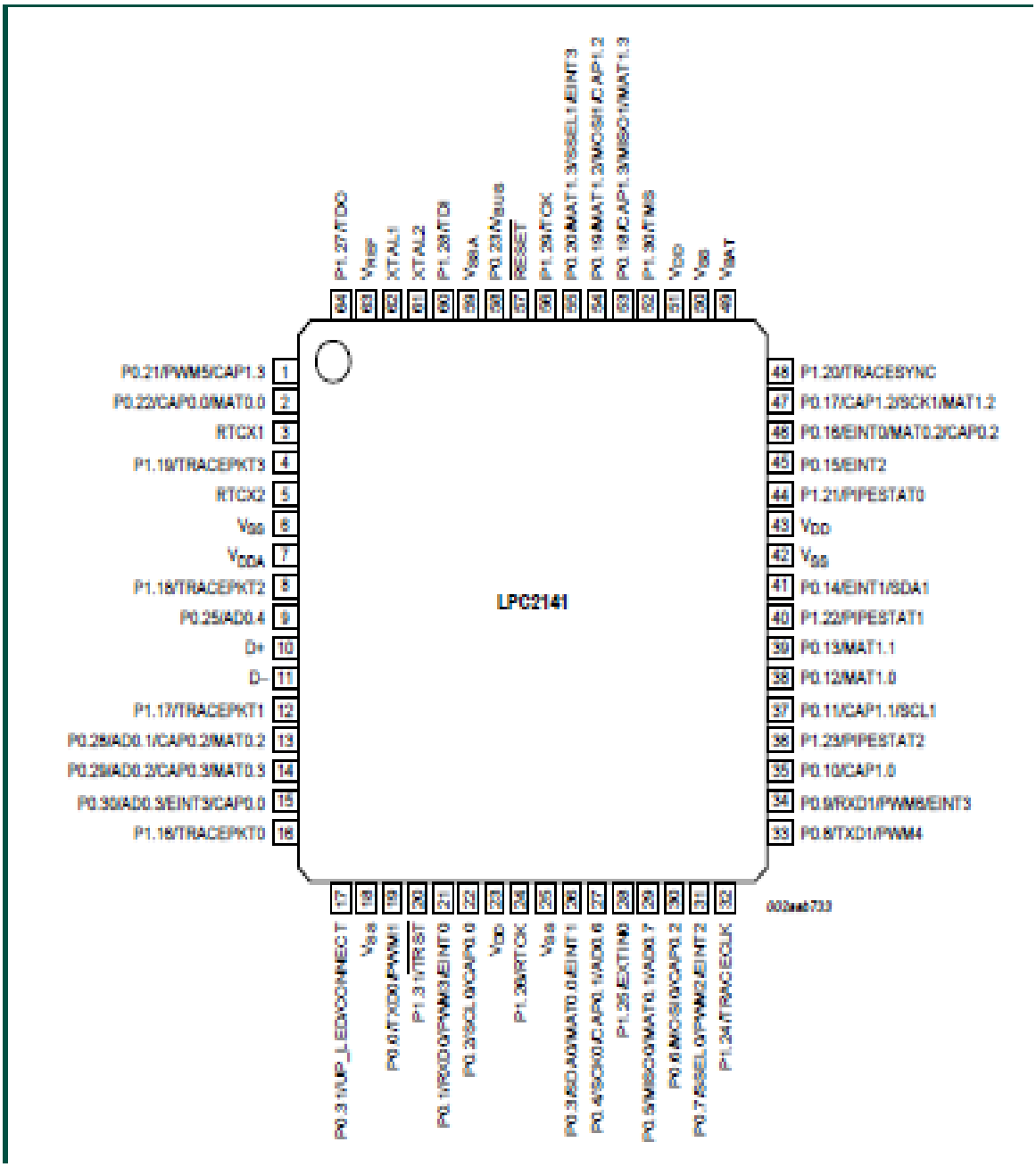
It has got 64 pins-QUAD package IC. Two ports: [0-31] and [32-63] pins



SCHEMATIC DIAGRAM OF MICROCONTROLLER LPC2148:



PINOUT of LPC2148 IC:



\

PIN Description of LPC2148 I.C:

Symbol	Pin	Type	Description
P0.0 to P0.31		I/O	**Port 0:** Port 0 is a 32-bit I/O port with individual direction controls for each bit. Total of 28 pins of the Port 0 can be used as a general purpose bi-directional digital I/Os while P0.31 provides digital output functions only. The operation of port 0 pins depends upon the pin function selected via the pin connect block. Pins P0.24, P0.26 and P0.27 are not available.
P0.0/TXD0/ PWM1	19[1]	I/O	**P0.0** — General purpose digital input/output pin
		O	**TXD0** — Transmitter output for UART0
		O	**PWM1** — Pulse Width Modulator output 1
P0.1/RxD0/ PWM3/EINT0	21[2]	I/O	**P0.1** — General purpose digital input/output pin
		I	**RxD0** — Receiver input for UART0
		O	**PWM3** — Pulse Width Modulator output 3
		I	**EINT0** — External interrupt 0 input
P0.2/SCL0/ CAP0.0	22[3]	I/O	**P0.2** — General purpose digital input/output pin
		I/O	**SCL0** — I^2C0 clock input/output. Open drain output (for I^2C compliance)
		I	**CAP0.0** — Capture input for Timer 0, channel 0
P0.3/SDA0/ MAT0.0/EINT1	26[3]	I/O	**P0.3** — General purpose digital input/output pin
		I/O	**SDA0** — I^2C0 data input/output. Open drain output (for I^2C compliance)
		O	**MAT0.0** — Match output for Timer 0, channel 0
		I	**EINT1** — External interrupt 1 input
P0.4/SCK0/ CAP0.1/AD0.6	27[4]	I/O	**P0.4** — General purpose digital input/output pin
		I/O	**SCK0** — Serial clock for SPI0. SPI clock output from master or input to slave
		I	**CAP0.1** — Capture input for Timer 0, channel 0
		I	**AD0.6** — A/D converter 0, input 6. This analog input is always connected to its pin
P0.5/MISO0/ MAT0.1/AD0.7	29[4]	I/O	**P0.5** — General purpose digital input/output pin
		I/O	**MISO0** — Master In Slave OUT for SPI0. Data input to SPI master or data output from SPI slave
		O	**MAT0.1** — Match output for Timer 0, channel 1
		I	**AD0.7** — A/D converter 0, input 7. This analog input is always connected to its pin
P0.6/MOSI0/ CAP0.2/AD1.0	30[4]	I/O	**P0.6** — General purpose digital input/output pin
		I/O	**MOSI0** — Master Out Slave In for SPI0. Data output from SPI master or data input to SPI slave
		I	**CAP0.2** — Capture input for Timer 0, channel 2
		I	**AD1.0** — A/D converter 1, input 0. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
P0.7/SSEL0/ PWM2/EINT2	31[2]	I/O	**P0.7** — General purpose digital input/output pin
		I	**SSEL0** — Slave Select for SPI0. Selects the SPI interface as a slave
		O	**PWM2** — Pulse Width Modulator output 2
		I	**EINT2** — External interrupt 2 input
P0.8/TXD1/ PWM4/AD1.1	33[4]	I/O	**P0.8** — General purpose digital input/output pin
		O	**TXD1** — Transmitter output for UART1
		O	**PWM4** — Pulse Width Modulator output 4
		I	**AD1.1** — A/D converter 1, input 1. This analog input is always connected to its pin. Available in LPC2144/6/8 only

Table 35. Pin description ...continued

Symbol	Pin	Type	Description
P0.9/RxD1/ PWM6/EINT3	34[2]	I/O	**P0.9** — General purpose digital input/output pin
		I	**RxD1** — Receiver input for UART1
		O	**PWM6** — Pulse Width Modulator output 6
		I	**EINT3** — External Interrupt 3 input
P0.10/RTS1/ CAP1.0/AD1.2	35[4]	I/O	**P0.10** — General purpose digital input/output pin
		O	**RTS1** — Request to Send output for UART1. Available in LPC2144/6/8 only.
		I	**CAP1.0** — Capture input for Timer 1, channel 0
		I	**AD1.2** — A/D converter 1, input 2. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
P0.11/CTS1/ CAP1.1/SCL1	37[3]	I/O	**P0.11** — General purpose digital input/output pin
		I	**CTS1** — Clear to Send input for UART1. Available in LPC2144/6/8 only.
		I	**CAP1.1** — Capture input for Timer 1, channel 1.
		I/O	**SCL1** — I^2C1 clock input/output. Open drain output (for I^2C compliance)
P0.12/DSR1/ MAT1.0/AD1.3	38[4]	I/O	**P0.12** — General purpose digital input/output pin
		I	**DSR1** — Data Set Ready input for UART1. Available in LPC2144/6/8 only.
		O	**MAT1.0** — Match output for Timer 1, channel 0.
		I	**AD1.3** — A/D converter input 3. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
P0.13/DTR1/ MAT1.1/AD1.4	39[4]	I/O	**P0.13** — General purpose digital input/output pin
		O	**DTR1** — Data Terminal Ready output for UART1. Available in LPC2144/6/8 only.
		O	**MAT1.1** — Match output for Timer 1, channel 1.
		I	**AD1.4** — A/D converter input 4. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
P0.14/DCD1/ EINT1/SDA1	41[3]	I/O	**P0.14** — General purpose digital input/output pin
		I	**DCD1** — Data Carrier Detect input for UART1. Available in LPC2144/6/8 only.
		I	**EINT1** — External Interrupt 1 input
		I/O	**SDA1** — I^2C1 data input/output. Open drain output (for I^2C compliance) **Note:** LOW on this pin while RESET is LOW forces on-chip boot-loader to take over control of the part after reset.
P0.15/RI1/ EINT2/AD1.5	45[4]	I/O	**P0.15** — General purpose digital input/output pin
		I	**RI1** — Ring Indicator input for UART1. Available in LPC2144/6/8 only.
		I	**EINT2** — External Interrupt 2 input.
		I	**AD1.5** — A/D converter 1, input 5. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
P0.16/EINT0/ MAT0.2/CAP0.2	46[2]	I/O	**P0.16** — General purpose digital input/output pin
		I	**EINT0** — External Interrupt 0 input.
		O	**MAT0.2** — Match output for Timer 0, channel 2.
		I	**CAP0.2** — Capture input for Timer 0, channel 2.
P0.17/CAP1.2/ SCK1/MAT1.2	47[1]	I/O	**P0.17** — General purpose digital input/output pin
		I	**CAP1.2** — Capture input for Timer 1, channel 2.
		I/O	**SCK1** — Serial Clock for SSP. Clock output from master or input to slave.
		O	**MAT1.2** — Match output for Timer 1, channel 2.

Table 35. Pin description ...continued

Symbol	Pin	Type	Description
P0.18/CAP1.3/ MISO1/MAT1.3	53[1]	I/O	**P0.18** — General purpose digital input/output pin
		I	**CAP1.3** — Capture input for Timer 1, channel 3.
		I/O	**MISO1** — Master In Slave Out for SSP. Data input to SPI master or data output from SSP slave.
		O	**MAT1.3** — Match output for Timer 1, channel 3.
P0.19/MAT1.2/ MOSI1/CAP1.2	54[1]	I/O	**P0.19** — General purpose digital input/output pin
		O	**MAT1.2** — Match output for Timer 1, channel 2.
		I/O	**MOSI1** — Master Out Slave In for SSP. Data output from SSP master or data input to SSP slave.
		I	**CAP1.2** — Capture input for Timer 1, channel 2.
P0.20/MAT1.3/ SSEL1/EINT3	55[2]	I/O	**P0.20** — General purpose digital input/output pin
		O	**MAT1.3** — Match output for Timer 1, channel 3.
		I	**SSEL1** — Slave Select for SSP. Selects the SSP interface as a slave.
		I	**EINT3** — External Interrupt 3 input.
P0.21/PWM5/ AD1.6/CAP1.3	1[4]	I/O	**P0.21** — General purpose digital input/output pin
		O	**PWM5** — Pulse Width Modulator output 5.
		I	**AD1.6** — A/D converter 1, input 6. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
		I	**CAP1.3** — Capture input for Timer 1, channel 3.
P0.22/AD1.7/ CAP0.0/MAT0.0	2[4]	I/O	**P0.22** — General purpose digital input/output pin.
		I	**AD1.7** — A/D converter 1, input 7. This analog input is always connected to its pin. Available in LPC2144/6/8 only.
		I	**CAP0.0** — Capture input for Timer 0, channel 0.
		O	**MAT0.0** — Match output for Timer 0, channel 0.
P0.23/V_{BUS}	58[1]	I/O	**P0.23** — General purpose digital input/output pin.
		I	V_{BUS} — Indicates the presence of USB bus power.
P0.25/AD0.4/ Aout	9[5]	I/O	**P0.25** — General purpose digital input/output pin
		I	**AD0.4** — A/D converter 0, input 4. This analog input is always connected to its pin.
		O	**Aout** — D/A converter output. Available in LPC2142/4/6/8 only.
P0.28/AD0.1/ CAP0.2/MAT0.2	13[4]	I/O	**P0.28** — General purpose digital input/output pin
		I	**AD0.1** — A/D converter 0, input 1. This analog input is always connected to its pin.
		I	**CAP0.2** — Capture input for Timer 0, channel 2.
		O	**MAT0.2** — Match output for Timer 0, channel 2.
P0.29/AD0.2/ CAP0.3/MAT0.3	14[4]	I/O	**P0.29** — General purpose digital input/output pin
		I	**AD0.2** — A/D converter 0, input 2. This analog input is always connected to its pin.
		I	**CAP0.3** — Capture input for Timer 0, Channel 3.
		O	**MAT0.3** — Match output for Timer 0, channel 3.
P0.30/AD0.3/ EINT3/CAP0.0	15[4]	I/O	**P0.30** — General purpose digital input/output pin.
		I	**AD0.3** — A/D converter 0, input 3. This analog input is always connected to its pin.

Symbol	Pin	Type	Description
P0.31	17[5]	O	**P0.31** — General purpose output only digital pin (GPO).
		O	**UP_LED** — USB Good Link LED indicator. It is LOW when device is configured (non-control endpoints enabled). It is HIGH when the device is not configured or during global suspend.
		O	**CONNECT** — Signal used to switch an external 1.5 kΩ resistor under the software control (active state for this signal is LOW). Used with the Soft Connect USB feature. **Note:** This pin MUST NOT be externally pulled LOW when RESET pin is LOW or the JTAG port will be disabled.
P1.0 to P1.31		I/O	**Port 1:** Port 1 is a 32-bit bi-directional I/O port with individual direction controls for each bit. The operation of port 1 pins depends upon the pin function selected via the pin connect block. Pins 0 through 15 of port 1 are not available.
P1.16/ TRACEPKT0	16[5]	I/O	**P1.16** — General purpose digital input/output pin
		O	**TRACEPKT0** — Trace Packet, bit 0. Standard I/O port with internal pull-up.
P1.17/ TRACEPKT1	12[5]	I/O	**P1.17** — General purpose digital input/output pin
		O	**TRACEPKT1** — Trace Packet, bit 1. Standard I/O port with internal pull-up.
P1.18/ TRACEPKT2	8[5]	I/O	**P1.18** — General purpose digital input/output pin
		O	**TRACEPKT2** — Trace Packet, bit 2. Standard I/O port with internal pull-up.
P1.19/ TRACEPKT3	4[5]	I/O	**P1.19** — General purpose digital input/output pin
		O	**TRACEPKT3** — Trace Packet, bit 3. Standard I/O port with internal pull-up.
P1.20/ TRACESYNC	48[5]	I/O	**P1.20** — General purpose digital input/output pin
		O	**TRACESYNC** — Trace Synchronization. Standard I/O port with internal pull-up. **Note:** LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.25:16 to operate as Trace port after reset
P1.21/ PIPESTAT0	44[5]	I/O	**P1.21** — General purpose digital input/output pin
		O	**PIPESTAT0** — Pipeline Status, bit 0. Standard I/O port with internal pull-up.
P1.22/ PIPESTAT1	40[5]	I/O	**P1.22** — General purpose digital input/output pin
		O	**PIPESTAT1** — Pipeline Status, bit 1. Standard I/O port with internal pull-up.
P1.23/ PIPESTAT2	36[5]	I/O	**P1.23** — General purpose digital input/output pin
		O	**PIPESTAT2** — Pipeline Status, bit 2. Standard I/O port with internal pull-up.
P1.24/ TRACECLK	32[5]	I/O	**P1.24** — General purpose digital input/output pin
		O	**TRACECLK** — Trace Clock. Standard I/O port with internal pull-up.
P1.25/EXTIN0	28[5]	I/O	**P1.25** — General purpose digital input/output pin
		I	**EXTIN0** — External Trigger Input. Standard I/O with internal pull-up.
P1.26/RTCK	24[5]	I/O	**P1.26** — General purpose digital input/output pin
		I/O	**RTCK** — Returned Test Clock output. Extra signal added to the JTAG port. Assists debugger synchronization when processor frequency varies. Bi-directional pin with internal pull-up. **Note:** LOW on this pin while $\overline{\text{RESET}}$ is LOW enables pins P1.31:26 to operate as Debug port after reset
P1.27/TDO	64[5]	I/O	**P1.27** — General purpose digital input/output pin
		O	**TDO** — Test Data out for JTAG interface.
P1.28/TDI	60[5]	I/O	**P1.28** — General purpose digital input/output pin
		I	**TDI** — Test Data in for JTAG interface.

Table 35. Pin description ...continued

Symbol	Pin	Type	Description
P1.29/TCK	56[5]	I/O	**P1.29** — General purpose digital input/output pin
		I	**TCK** — Test Clock for JTAG interface. This clock must be slower than 1/6 of the CPU clock (CCLK) for the JTAG interface to operate.
P1.30/TMS	52[5]	I/O	**P1.30** — General purpose digital input/output pin
		I	**TMS** — Test Mode Select for JTAG interface.
P1.31/$\overline{\text{TRST}}$	20[5]	I/O	**P1.31** — General purpose digital input/output pin
		I	**$\overline{\text{TRST}}$** — Test Reset for JTAG interface.
D+	10[7]	I/O	USB bidirectional D+ line.
D-	10[7]	I/O	USB bidirectional D- line.
$\overline{\text{RESET}}$	57[8]	I	**External reset input**: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. TTL with hysteresis, 5 V tolerant.
XTAL1	62[9]	I	Input to the oscillator circuit and internal clock generator circuits.
XTAL2	61[9]	O	Output from the oscillator amplifier.
RTCX1	3[10]	I	Input to the RTC oscillator circuit. Can be left floating if the RTC is not used.
RTCX2	5[10]	O	Output from the RTC oscillator circuit. Can be left floating if the RTC is not used.
V_{SS}	6, 18, 25, 42, 50	I	**Ground**: 0 V reference
V_{SSA}	59	I	**Analog Ground**: 0 V reference. This should nominally be the same voltage as V_{SS}, but should be isolated to minimize noise and error. This pin must be grounded if the ADC/DAC are not used.
V_{DD}	23, 43, 51	I	**3.3 V Power Supply**: This is the power supply voltage for the core and I/O ports.
V_{DDA}	7	I	**Analog 3.3 V Power Supply**: This should be nominally the same voltage as V_{DD} but should be isolated to minimize noise and error. This voltage is used to power the ADC(s) and DAC (where available). This pin must be tied to V_{DD} when the ADC/DAC are not used.
V_{REF}	63	I	**A/D Converter Reference**: This should be nominally the same voltage as V_{DD} but should be isolated to minimize noise and error. Level on this pin is used as a reference for A/D convertor and DAC (where available). This pin must be tied to V_{DD} when the ADC/DAC are not used.
V_{BAT}	49	I	**RTC Power Supply**: 3.3 V on this pin supplies the power to the RTC.

[1] 5 V tolerant pad providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control.

[2] 5 V tolerant pad providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control. If configured for an input function, this pad utilizes built-in glitch filter that blocks pulses shorter than 3 ns.

[3] Open-drain 5 V tolerant digital I/O I2C-bus 400 kHz specification compatible pad. It requires external pull-up to provide an output functionality.

[4] 5 V tolerant pad providing digital I/O (with TTL levels and hysteresis and 10 ns slew rate control) and analog input function. If configured for an input function, this pad utilizes built-in glitch filter that blocks pulses shorter than 3 ns. When configured as an ADC input, digital section of the pad is disabled.

[5] 5 V tolerant pad providing digital I/O (with TTL levels and hysteresis and 10 ns slew rate control) and analog output function. When configured as the DAC output, digital section of the pad is disabled.

[6] 5 V tolerant pad with built-in pull-up resistor providing digital I/O functions with TTL levels and hysteresis and 10 ns slew rate control. The pull-up resistor's value typically ranges from 60 kΩ to 300 kΩ.

[7] Pad is designed in accordance with the Universal Serial Bus (USB) specification, revision 2.0 (Full-speed and Low-speed mode only).

[8] 5 V tolerant pad providing digital input (with TTL levels and hysteresis) function only.

[9] Pad provides special analog functionality.

Pin Description:

Port 0:32 bit I/O port with individual direction controls for each bit. Of total 32 pins of port 0-31 can be used a general purpose bidirectional digital I/O while P0.31 is only O/P pin.
Operation of the Port 0 pins depends upon the pin function selected via pin connect block.

Port 1: 32 Bidirectional I/O port with individual direction controls for each bit.

Port 1.0 to Port 1.15 are not available

Port 1.16 to Port 1.31 are available

Fast General Purpose Parallel I/O registers

Device Pins that are not connected to a Specific Peripheral function are controlled by GPIO register Ports can be Dynamically configured as the I/O.
Now we came to know the description of the pins. We need to know how to configure them for communicating with the microcontroller.

PIN CONFIGURATION:

There is a block called as Pin connect block which helps us to the configure the microcontroller pins to do the desired functions.

The pin connect block allows selected pins of the microcontroller to have more than one function. Configuration registers control the multiplexers to allow connection between the pin and the on chip peripherals.

Selection of single function on a port pin completely excludes all other functions otherwise available on the same pin.

PinSEL0 –Port0.0 to port 0.15

PinSEL1 –Port0.16 to port 0.31

PinSEL0 –Port1.16 to port 1.31

All the above registers are 16 Bit

Port1.0 to port1.15 are for future reference. they are not used here We have three special function registers which help us to clear or set the data in a particular register. They are

IOXCLR-16-bit Register to clear the content. If we give LOGIC 1 It has no effect. But if we give Logic 0 it clears the required pins of the register

IOXSET-16-bit register to set the content. If we configure with Logic 1 it sets all the pins

IOXDIR- X=>0 UART 0

X=>1 UART 1

We will understand this pin configuration by blinking of LED programs clearly. For this purpose, we need to know how to program the controller

PROGRAMMING THE MICROCONTROLLER:

First of all, we simulator called Keil4.We simulate in this and then check if we are getting the required output. Type the below code in Keil4 software

Write an Embedded C Program for blinking of a 2 LEDS which are connected to PORT0.1 and PORT0.2 with a delay of 10milliseconds?

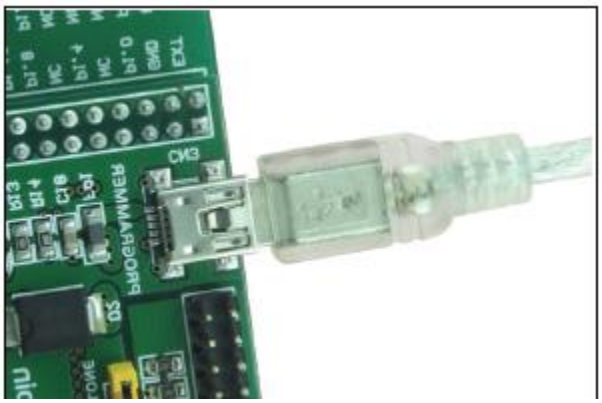
```
#include<PC214X.h>
Void delay(int)
Int void main ()
{
While (1)
{

IOX PIR 0x00000006;
PINSE0=0x0000000;
IOXSET=0x0000006;
delay (10);
IOXCLR=0x00000006;
}
}

Void delay (int x)

{

Int i,j;
for(i=0;i<=2;i++)
{
For(j=0;j<=1275;j++)
{
return (0);
}
}
}
```

Now after you check the output in the registers if we get the desired output. Then we can program the controller

The Microcontroller can be programmed with a boot loader or the JTAG programmer. The use of the BOOT loader is enabled due to the boot loader code that is loaded into the microcontroller. In order to program the microcontroller with the Boot loader, it is necessary to connect the board to PC via a CN3 Connector and USB cable. The HEX code obtained by running the code in Keil software is transferred from the PC to the Microcontroller by using some of the BOOTLOADER programs Here we are using the FLASH Magic.

Below are the Steps to followed to upload the code into the Microcontroller

Step1:

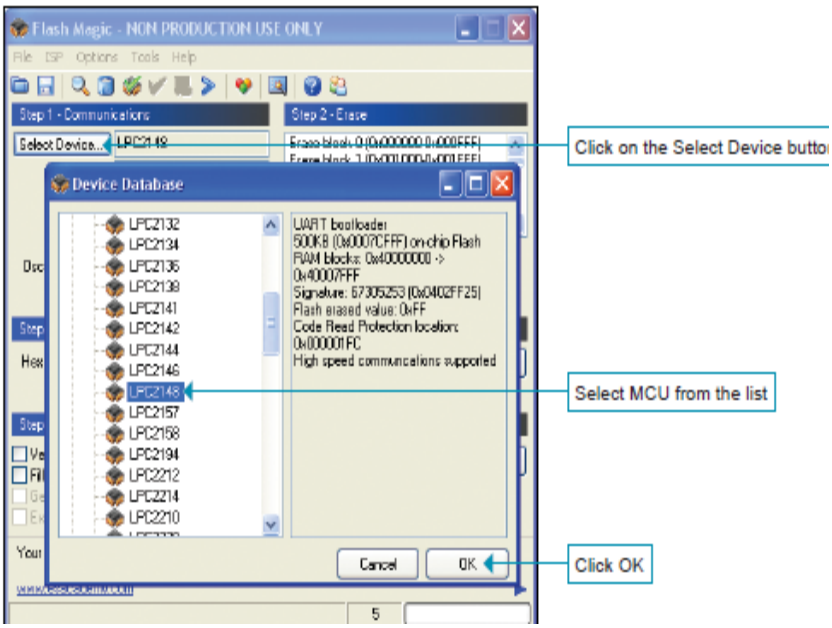


Connect the ARM 64 pin microboard to the PC

Step2:

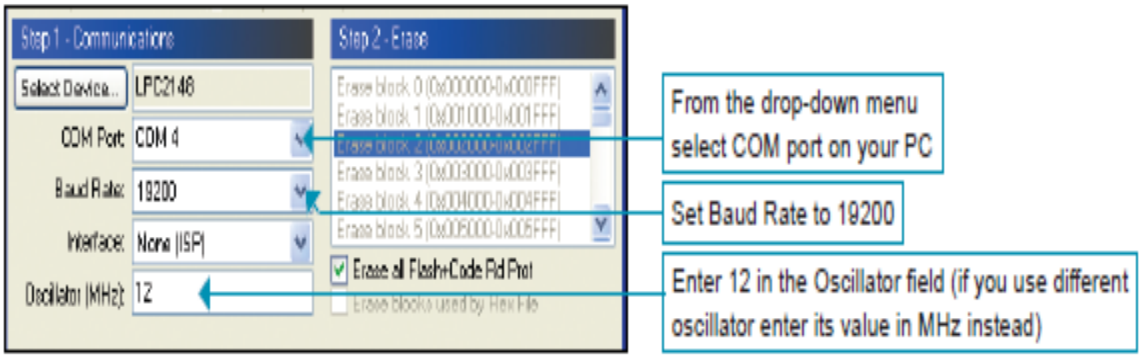
Start the FLASH magic in Your PC by installing and double clicking the icon Flash Magic
If you don't have download the flash magic from the link given below
http://www.flashmagictool.com/download.html&d=FlashMagic.exe

Step3:

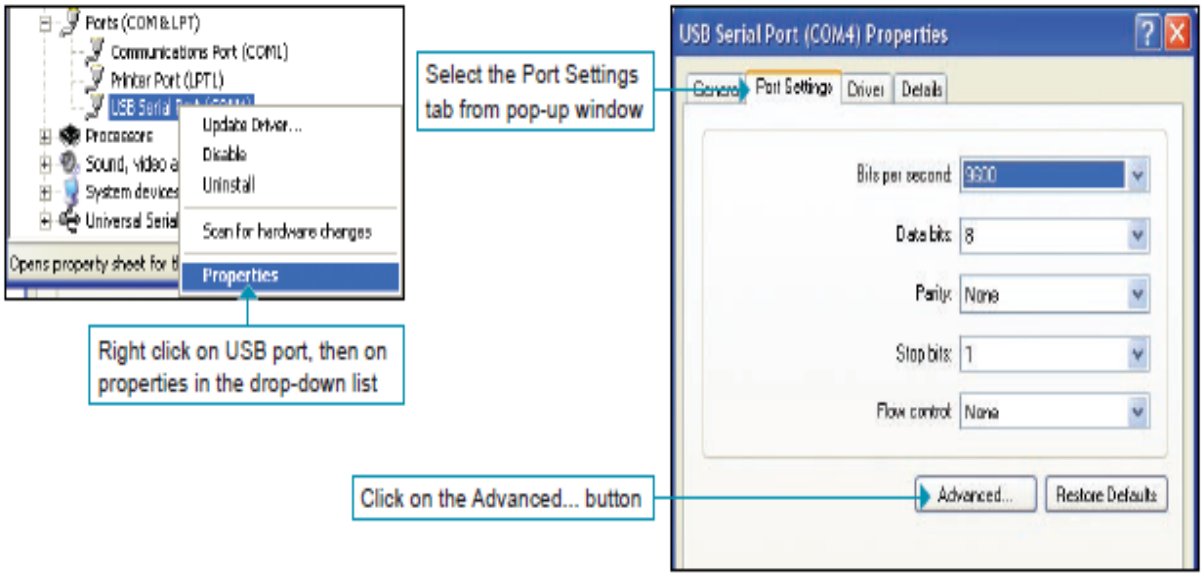


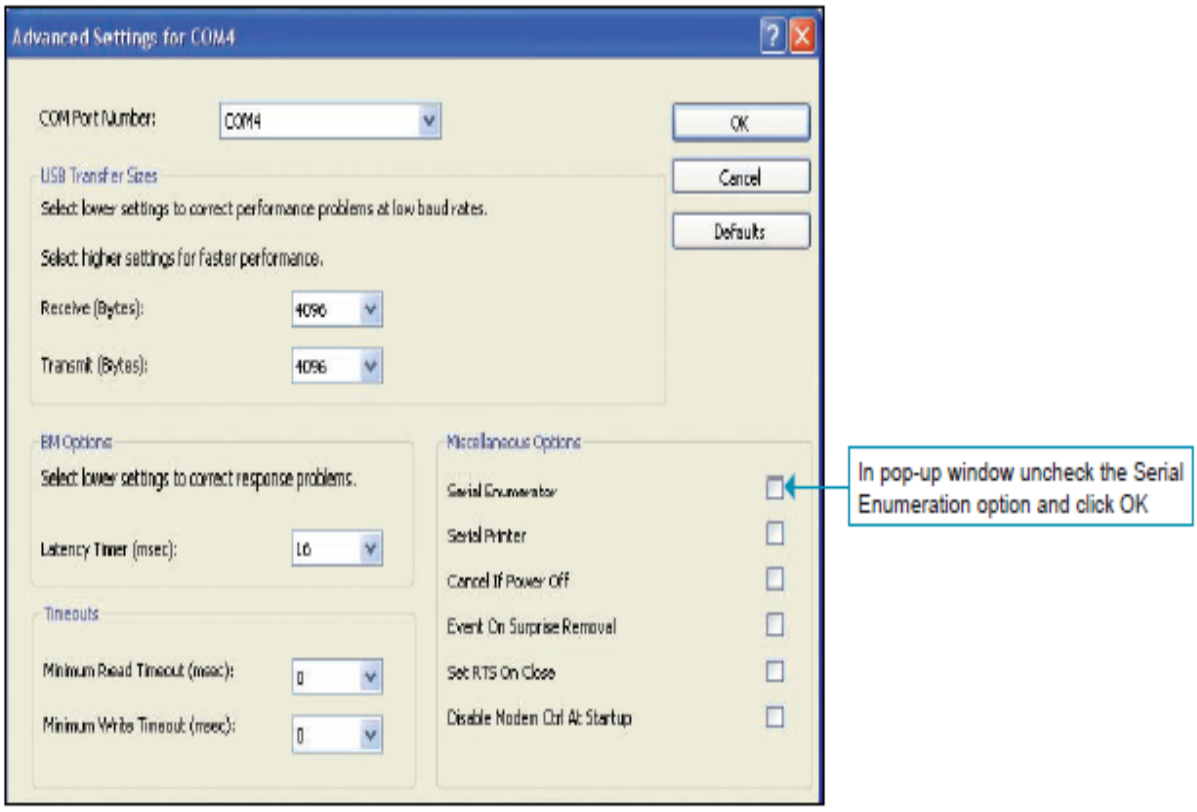
Select the Microcontroller Unit

Step4:

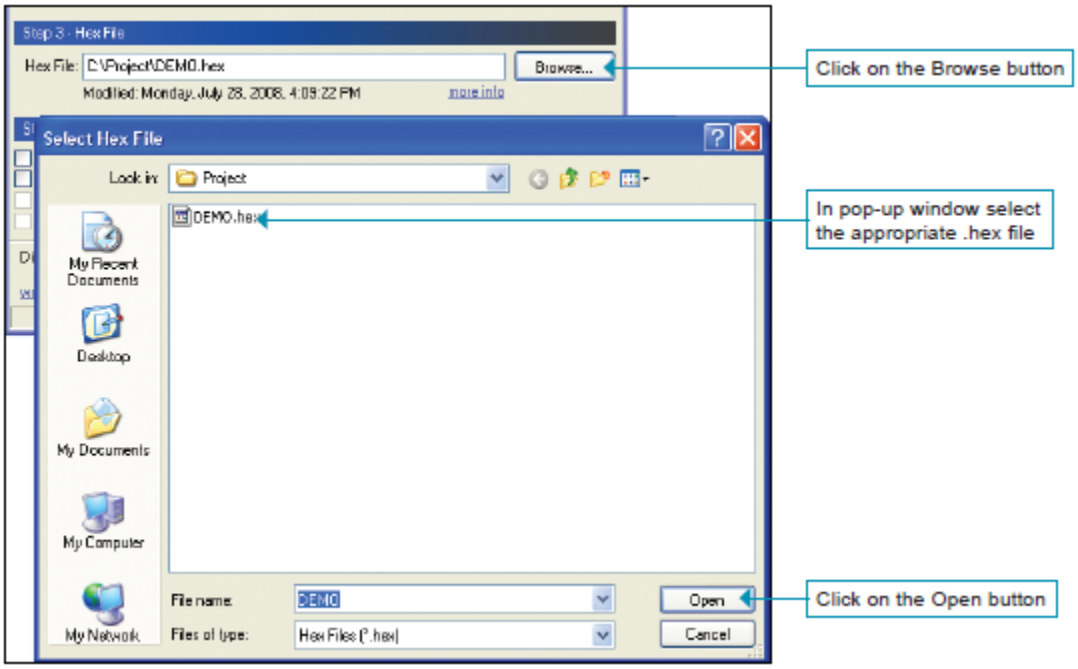


Device Manager on your PC contains information on which COM port is used for USB communication with the mikroBoard for ARM 64-pin. In this case the COM4 port is used.


Ports (COM & LPT)
Communications Port (COM1)
Printer Port (LPT1)
Processors
Update Driver...
Disable
Uninstall
Scan for hardware changes
Properties
Right click on USB port, then on properties in the drop-down list
Select the Port Settings tab from pop-up window
USB Serial Port (COM4) Properties
General
Port Settings
Driver
Details
Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
Advanced...
Restore Defaults
Click on the Advanced... button

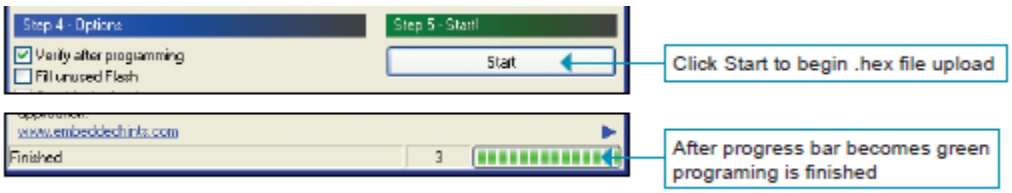

Advanced Settings for COM4
COM Port Number: COM4
OK
Cancel
Defaults
USB Transfer Sizes
Select lower settings to correct performance problems at low baud rates.
Select higher settings for faster performance.
Receive (Bytes): 4096
Transmit (Bytes): 4096
BM Options
Select lower settings to correct response problems.
Latency Timer (msec): 16
Timeouts
Minimum Read Timeout (msec): 0
Minimum Write Timeout (msec): 0
Miscellaneous Options
Serial Enumerator
Serial Printer
Cancel If Power Off
Event On Surprise Removal
Set RTS On Close
Disable Modem Ctrl At Startup
In pop-up window uncheck the Serial Enumeration option and click OK

Step5:



Browse for the HEX file of required function

Step6:



Finally Upload the HEX file into the Microcontroller.

In this we can upload the files and program the Controller.

LCD Unit in LPC2148:

A liquid crystal display (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. Each pixel consists of a column of liquid crystal molecules suspended between two transparent electrodes, and two polarizing filters, the axes of polarity of which are perpendicular to each other. Without the liquid crystals between them, light passing through one would be blocked by the other. The liquid crystal twists the polarization of light entering one filter to allow it to pass through the other.

A program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to a controller is an LCD display. Some of the most common LCDs connected to the controllers are 16X1, 16x2 and 20x2 displays. This means 16 characters per line by 1 line 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively.

Shapes and S

available. Line lengths of 8, 16, 20, 24, 32 and 40 characters are all standard, in one, two

Many microcontroller devices use 'smart LCD' displays to output visual information. LCD displays designed around LCD NT-C1611 module, are inexpensive, easy to use, and it is even possible to produce a readout using the 5X7 dots plus cursor of the display. They have a standard ASCII set of characters and mathematical symbols. For an 8-bit data bus, the display requires a +5V supply plus 10 I/O lines (RS RW D7 D6 D5 D4 D3 D2 D1 D0). For a 4-bit data bus it only requires the supply lines plus 6 extra lines (RS RW D7 D6 D5 D4). When the LCD display is not enabled, data lines are tristate and they do not interfere with the operation of the microcontroller.

Features:

(1) Interface with either 4-bit or 8-bit microprocessor.
(2) Display data RAM
(3) 80x□8 bits (80 characters).
(4) Character generator ROM
(5) 160 different 5 □□7 dot-matrix character patterns.
(6) Character generator RAM (7) □8 different user programmed 5 □□7 dot-matrix patterns.
(8) Display data RAM and character generator RAM may be Accessed by the microprocessor.
(9) Numerous instructions
(10) Clear Display, Cursor Home, Display ON/OFF, Cursor ON/OFF, Blink Character, Cursor Shift, Display Shift.

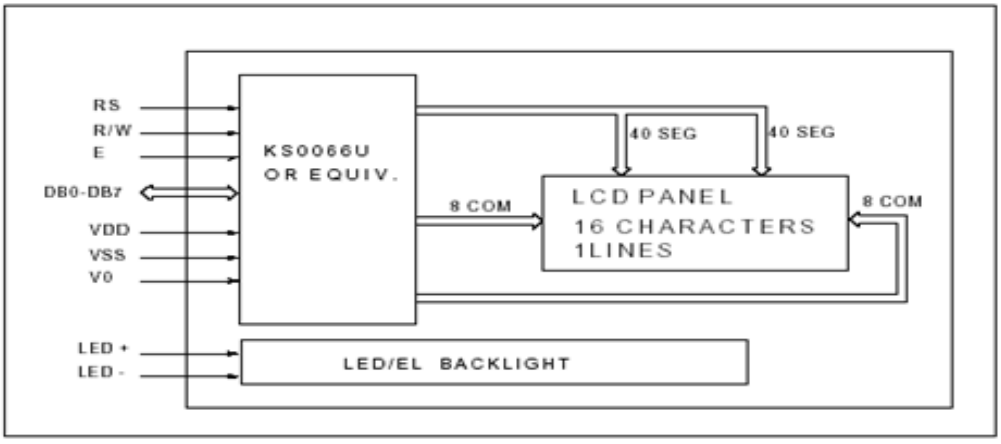
(11). Built-in reset circuit is triggered at power ON.
(12). Built-in oscillator.

- Data can be placed at any location on the LCD. For 16×1 LCD, the address locations are:

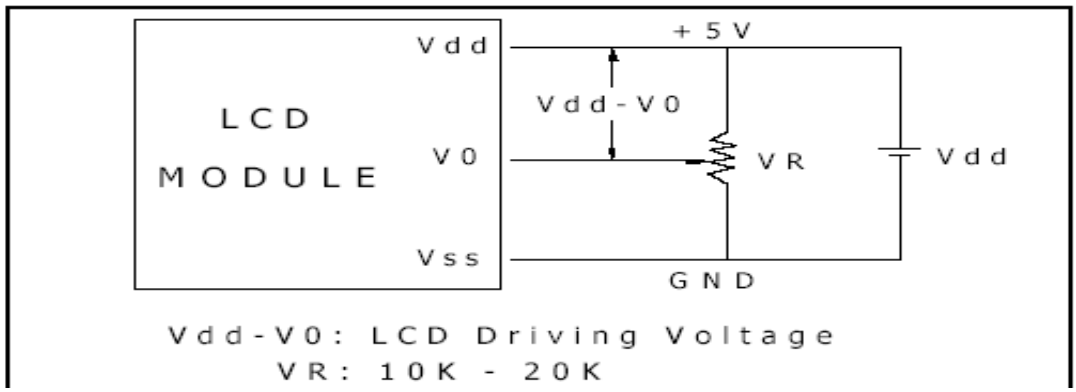
POSITION		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ADDRESS	LINE1	00	01	02	03	04	05	06	07	40	41	42	43	44	45	46	47

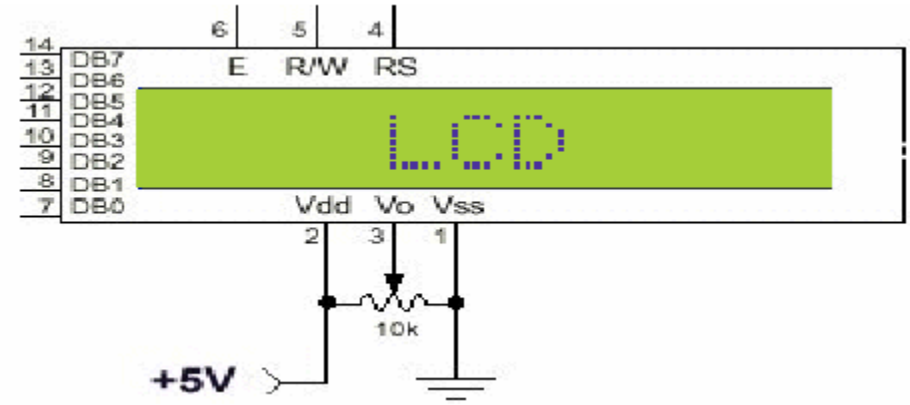
Figure representing the Address locations for a 1x16 line LCD

Electrical blockdiagram:



Power supply for lcd driving:





The above diagram gives the schematic view and pins in it clearly.

Pin Description:

PIN	SYMBOL	FUNCTION
1	Vss	Power Supply(GND)
2	Vdd	Power Supply(+5V)
3	Vo	Contrast Adjust
4	RS	Instruction/Data Register Select
5	R/W	Data Bus Line
6	E	Enable Signal
7-14	DB0-DB7	Data Bus Line
15	A	Power Supply for LED B/L(+)
16	K	Power Supply for LED B/L(-)

CONTROL LINES:

EN:

Line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

RS:

Line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

RW:

Line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands, so RW will almost always be low.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7.

Logic status on control lines:

• E - 0 Access to LCD disabled

- 1 Access to LCD enabled

• R/W - 0 Writing data to LCD

- 1 Reading data from LCD

• RS - 0 Instructions

1 Character

Writing data to the LCD:

1) Set R/W bit to low

2) Set RS bit to logic 0 or 1 (instruction or character)

3) Set data to data lines (if it is writing)

4) Set E line to high

5) Set E line to low

Read data from data lines (if it is reading) on LCD:

1) Set R/W bit to high

2) Set RS bit to logic 0 or 1 (instruction or character)

3) Set data to data lines (if it is writing)

4) Set E line to high

5) Set E line to low

CODE FOR DISPLAYING IN LCD UNIT in LPC2148:

```
#include "gpio.h"

#define  LCD_RS     P1_24                                          // P1.24
#define  LCD_EN     P1_22                                          // P1.22

#define  LCD_D4     P0_10                                          // P0.10
#define  LCD_D5     P0_11                                          // P0.11
#define  LCD_D6     P0_12                                         // P0.12
#define  LCD_D7     P0_13                                          // P0.13

#define  LCD_RS_on    (IOSET1 |= LCD_RS)
#define  LCD_RS_off   (IOCLR1 |= LCD_RS)

#define  LCD_EN_on  (IOSET1 |= LCD_EN)
#define  LCD_EN_off   (IOCLR1 |= LCD_EN)
void delay( unsigned int a)
{
 int i,j;
 for(i=0;i<a;i++)
 for(j=0;j<5000;j++);
}

void clcd(unsigned char val)
{
 unsigned int lcd_ch;
 unsigned int lcd_i;

 IOCLR1 = LCD_RS ;

 lcd_ch=((val>>4)&0x0f);

 IOCLR0 = (LCD_D7|LCD_D6|LCD_D5|LCD_D4);
 IOSET0 = (lcd_ch<<10);
 IOSET1 = LCD_EN ;
 for (lcd_i=0;lcd_i<100;lcd_i++);
 IOCLR1 = LCD_EN ;
 lcd_ch=(val&0x0F);

 IOCLR0 = (LCD_D7|LCD_D6|LCD_D5|LCD_D4);
 IOSET0 = (lcd_ch<<10);
 IOSET1 = LCD_EN ;                                    // EN = 1 (Enable)
 for (lcd_i=0;lcd_i<100;lcd_i++);          //delay
 IOCLR1 = LCD_EN ;                                    // EN = 0 (Disable)

 for (lcd_i=0;lcd_i<100;lcd_i++);           //delay
}
```

```
void dlcd(unsigned char val)
{
 unsigned int lcd_ch;                  // LCD Initial Data
 unsigned int lcd_i;                   // LCD Initial Delay Count

 IOSET1 = LCD_RS ;                          // RS = 1

 lcd_ch=((val>>4)&0x0F);

 IOCLR0 = (LCD_D7|LCD_D6|LCD_D5|LCD_D4); // Reset 4-Bit Pin Data
 IOSET0 = (lcd_ch<<10);
 IOSET1 = LCD_EN ;                                      // EN = 1 (Enable)
 for (lcd_i=0;lcd_i<100;lcd_i++);          // delay
 IOCLR1 = LCD_EN ;                                      // EN = 0 (Disable)

 lcd_ch=(val&0x0F);

 IOCLR0 = (LCD_D7|LCD_D6|LCD_D5|LCD_D4);
 IOSET0 = (lcd_ch<<10);

 IOSET1 = LCD_EN ;                                      // EN = 1 (Enable)
 for (lcd_i=0;lcd_i<100;lcd_i++);          //delay
 IOCLR1 = LCD_EN ;                                      // EN = 0 (Disable)

 for (lcd_i=0;lcd_i<100;lcd_i++);           //delay
}

void initlcd()
{
 unsigned int i;
 PINSEL0 |= 0x00000000;
 PINSEL1 |= 0x00000000;
 PINSEL2 |= 0x00000000;
 IODIR1  |= 0x01400000;
 IODIR0  |= 0x00003c00;


 for (i=0;i<10000;i++);

 IOCLR0 = ((LCD_D7|LCD_D6|LCD_D5|LCD_D4));
 IOCLR1 =((LCD_RS|LCD_EN));
 for (i=0;i<10000;i++);
 LCD_EN_off;
 for (i=0;i<10000;i++);

 IOCLR0 = ((LCD_D7|LCD_D6|LCD_D5|LCD_D4));
  IOCLR1 =((LCD_RS|LCD_EN));
 IOSET0= (LCD_D5|LCD_D4);
 LCD_EN_on;
 for (i=0;i<10000;i++);
  LCD_EN_off;
```

```
 for (i=0;i<10000;i++);

 IOCLR0 = ((LCD_D7|LCD_D6|LCD_D5|LCD_D4));

  IOCLR1 =((LCD_RS|LCD_EN));
 IOSET0 = (LCD_D5|LCD_D4);
 LCD_EN_on;
 for (i=0;i<10000;i++);
 LCD_EN_off;
 for (i=0;i<10000;i++);

 IOCLR0 = ((LCD_D7|LCD_D6|LCD_D5|LCD_D4));

  IOCLR1 =((LCD_RS|LCD_EN));
 IOSET0 = (LCD_D5);
 LCD_EN_on;
 for (i=0;i<10000;i++);
 LCD_EN_off;
 for (i=0;i<10000;i++);


 clcd(0x28); delay(1000);
 clcd(0x28);delay(1000);

 clcd(0x0e);delay(1000);
  clcd(0x01);delay(1000);
 clcd(0x06);delay(1000);
 clcd(0x80);delay(1000);

}


void stringlcd(unsigned char ch,unsigned char *str)
{

 if(ch==0x80){clcd(0x01);}
 clcd(ch);delay(50);
 while(*str)
 {
  dlcd(*str++);delay(50);

 }
}

// CONVERT HEX DECIMAL TO ASCII VALUE

 void conv(unsigned int temp1_value)
                        {
           unsigned char value,d1,d2,d3,d4,val1;
             value=temp1_value/10;
             d4=temp1_value%10;
```

```
 val1=value/10;
 d3=value%10;
                        d2=val1%10;
                        d1=val1/10;

dlcd(d1+48);delay(10);
dlcd(d2+48);delay(10);
dlcd(d3+48);delay(10);
        dlcd(d4+48);delay(10);
```

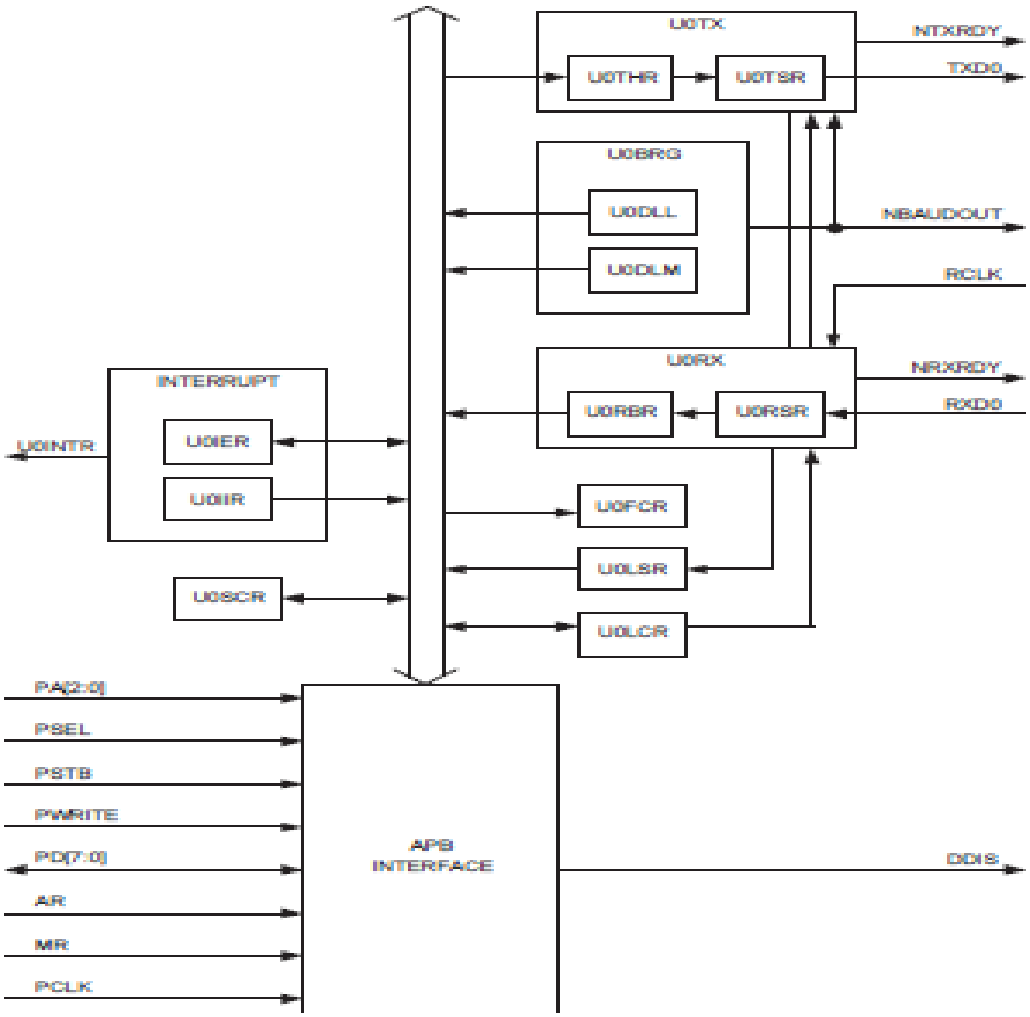
SERIAL COMMUNICATION:

There are two UARTS in the LPC2148.

Features:

- 16 byte Receive and Transmit FIFOs
- Register locations conform to '550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Built-in fractional baud rate generator with auto bauding capabilities.
- Mechanism that enables software and hardware flow control implementation.

UART0 BLOCK DIAGRAM



UART0 PIN DESCRIPTION:

Table 160: UART0 pin description

Pin	Type	Description
RXD0	Input	**Serial Input.** Serial receive data.
TXD0	Output	**Serial Output.** Serial transmit data.

UART0 Register Map:

Table 161: UART0 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U0RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE000 C000 (DLAB=0)
U0THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE000 C000 (DLAB=0)
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000 C014

Line Select Register/Line Status Register:

DR: Data Ready
If receive Bit is ready, then It contains 1
If Receive Bit is not ready, then it contains 0

OE: Over run error
If overrun occurs, then it contains 1
If no overrun occurs, then it contains 0

FE: Frame error [Error occurred at start/stop bit]
If there exists start and stop bit in framing it contains 1
If any of start/stop bit is missing, then FE has 0

BI: Break Interrupt
Continuous data is being received
If there is a break in data, then it contains 1
If there is no break in data, then it contains 0

THRE: Transmit Hold Register Empty
If there is no data in Register, then it contains 1
If there is Data in THR, then it has 0

THR Register: Transmit Hold Register
TEMT-It checks the THR
THR=1 No data
THR=0 Data available

Rx: -`
If there is error in receiving bit, then it has 1
If there's no error in receiving bit,then it has 0

Line Control Register:
Baud: This is used to set Baud rate
Parity:
00->Odd parity
01->Even Parity

Stop Bit:
If there is 0 in Stop bit –It means it has1 Stop Bit
If there is 1 in Stop bit-2Stop Bits are there

Parity enable:
0-No parity generate
1-Parity generate

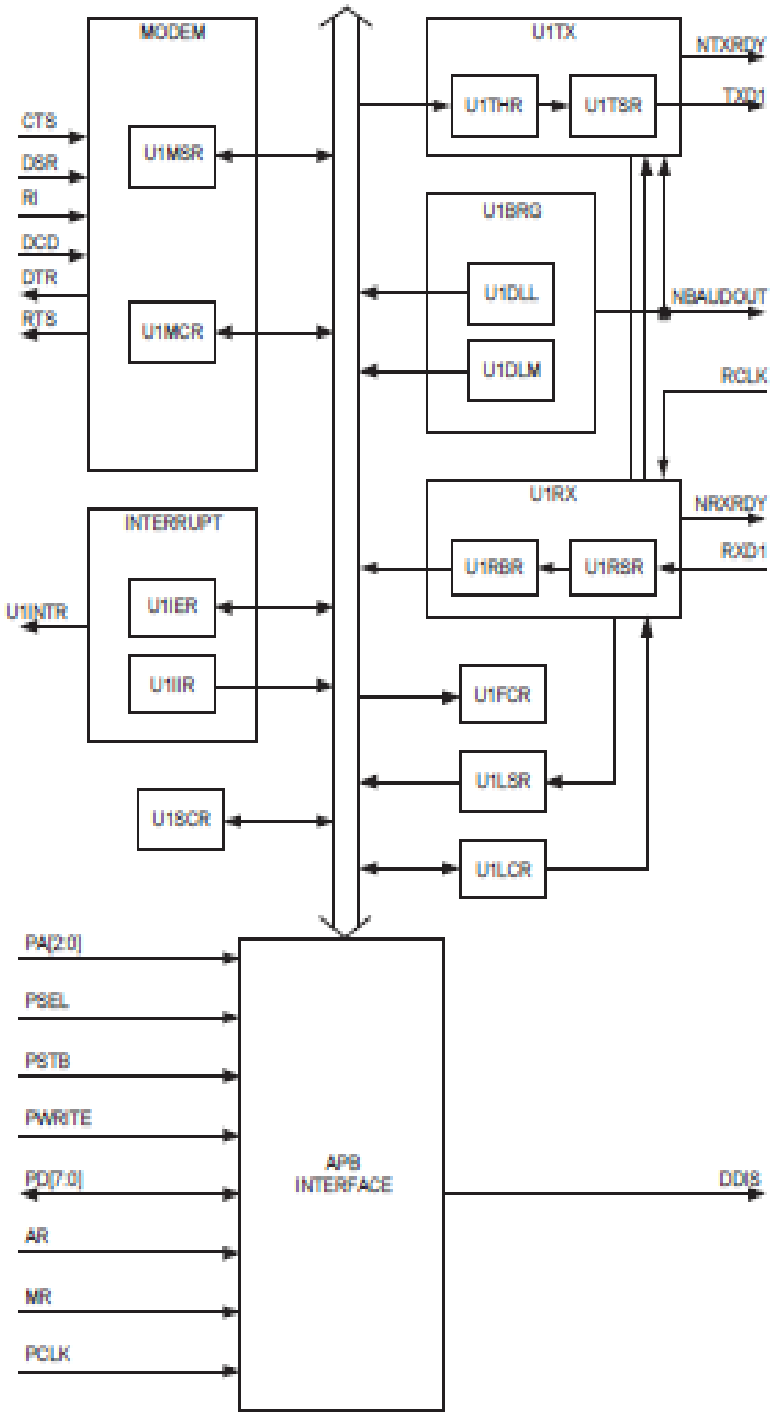
Divisor Latch
0-Disable
1-Enable

UART1:

Features:

• UART1 is identical to UART0
• 16 byte Receive and Transmit FIFOs.
• Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
• Built-in fractional baud rate generator with autobauding capabilities.
• Mechanism that enables software and hardware flow control implementation.

Block diagram of UART1



Register Map of UART1:

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U1RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE001 0000 (DLAB=0)
U1DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE001 0000 (DLAB=1)
U1DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE001 0004 (DLAB=1)
U1IER	Interrupt Enable Register	-	-	-	-	-	-	En.ABTO	En.ABEO	R/W	0x00	0xE001 0004 (DLAB=0)
		En.CTS Int[2]	-	-	-	E.Modem St.Int[2]	En. RX Lin.St. Int	Enable THRE Int	En. RX DatAv.Int			
U1IIR	Interrupt ID Reg.	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE001 0008
		FIFOs Enabled		-	-	IIR3	IIR2	IIR1	IIR0			
U1FCR	FIFO Control Register	RX Trigger		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE001 000C
U1MCR[2]	Modem Ctrl. Reg.	CTSen	RTSen	-	LoopBck.	-	-	RTS	DTR	R/W	0x00	0xE001 0010
U1LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE001 0014
U1MSR[2]	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing Edge RI	Delta DSR	Delta CTS	RO	0x00	0xE001 0018
U1SCR	Scratch Pad Reg.	8-bit Data								R/W	0x00	0xE001 001C
U1ACR	Auto-baud Control Register	-	-	-	-	-	-	ABTO IntClr	ABEO IntClr	R/W	0x00	0xE001 0020
		-	-	-	-	-	Aut.Rstrt.	Mode	Start			
U1FDR	Fractional Divider Register	Reserved[31:8]								R/W	0x10	0xE001 0028
		MulVal				DivAddVal						
U1TER	TX. Enable Reg.	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE001 0030

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

[2] Modem specific features are available in LPC2144/6/8 only.

Pin description of UART1:

Pin	Type	Description
RXD1	Input	**Serial Input.** Serial receive data.
TXD1	Output	**Serial Output.** Serial transmit data.
DSR1[1]	Input	**Data Set Ready.** Active low signal indicates if the external modem is ready to establish a communications link with the UART1. In normal operation of the modem interface (U1MCR[4] = 0), the complement value of this signal is stored in U1MSR[5]. State change information is stored in U1MSR[1] and is a source for a priority level 4 interrupt, if enabled (U1IER[3] = 1).
DTR1[1]	Output	Data Terminal Ready. Active low signal indicates that the UART1 is ready to establish connection with external modem. The complement value of this signal is stored in U1MCR[0].

Finally, if we want to communicate we need to set the Baud rate

Formulas to calculate the BAUD rate:

In case of UART0:

$$UART0_{baudrate} = \frac{PCLK}{16 \times (256 \times U0DLM + U0DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

U0DLM and U0DLL are standard UART0 Baud rate divider registers

DIVADDVAL and MULVAL are UART1 fractional baud rate generator specific parameters.

In case of UART1:

$$UART1_{baudrate} = \frac{PCLK}{16 \times (256 \times U1DLM + U1DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

PCLK is Peripheral clock

U1DLM and U1DLL are standard UART1 Baud rate divider registers

DIVADDVAL and MULVAL are UART1 fractional baud rate generator specific parameters.

We generally set the baud rate as 96000bps.

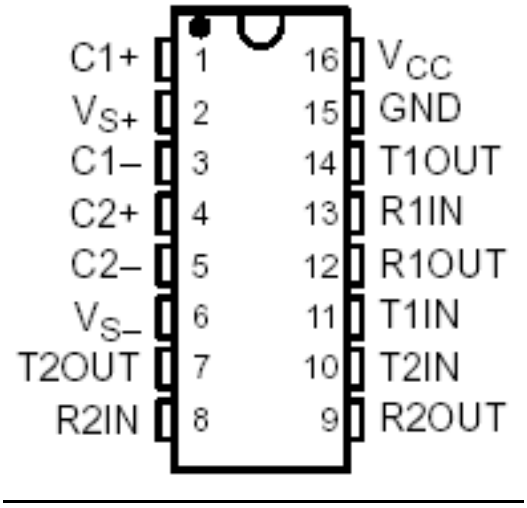
Embedded C code for receiving and transmitting a single character:

```
#include<PC241x.h>
Char
PINSE0=0x00000005;
U0LCR=0x83;
U0Dll=0x61;
D0DLM=0x00;
U0CLR=0x03;
While(1)
UOTHR=’A’;
While(!LSR&0x02);
}
While(!LSR & 0x01);
A=U0RBR;
}
```

MAX 232

Introduction:

A standard serial interface for PC, RS232C, requires negative logic, i.e., logic 1 is -3V to -12V and logic 0 is +3V to +12V. To convert TTL logic, say, TxD and RxD pins of the microcontroller thus need a converter chip. A MAX232 chip has long been using in many microcontrollers boards. It is a dual RS232 receiver / transmitter that meets all RS232 specifications while using only +5V power supply. It has two onboard charge pump voltage converters which generate +10V to -10V power supplies from a single 5V supply. It has four level translators, two of which are RS232 transmitters that convert TTL/CMOS input levels into +9V RS232 outputs. The other two level translators are RS232 receivers that convert RS232 input to 5V. Typical MAX232 circuit is shown below.



Features:

1. Operates with Single 5-V Power Supply

2.LinBiCMOSE Process Technology

3.Two Drivers and Two Receivers

4.±30-V Input Levels

5.Low Supply Current. 8 mA (Typical)

6.Meets or Exceeds TIA/EIA-232-F and ITU

Recommendation V.28

7.Designed to be Interchangeable With

Maxim MAX232

8.Applications

- TIA/EIA-232-F
- Battery-Powered Systems
- Terminals
- Modems
- Computers

9.ESD Protection Exceeds 2000 V Per

MIL-STD-883, Method 3015

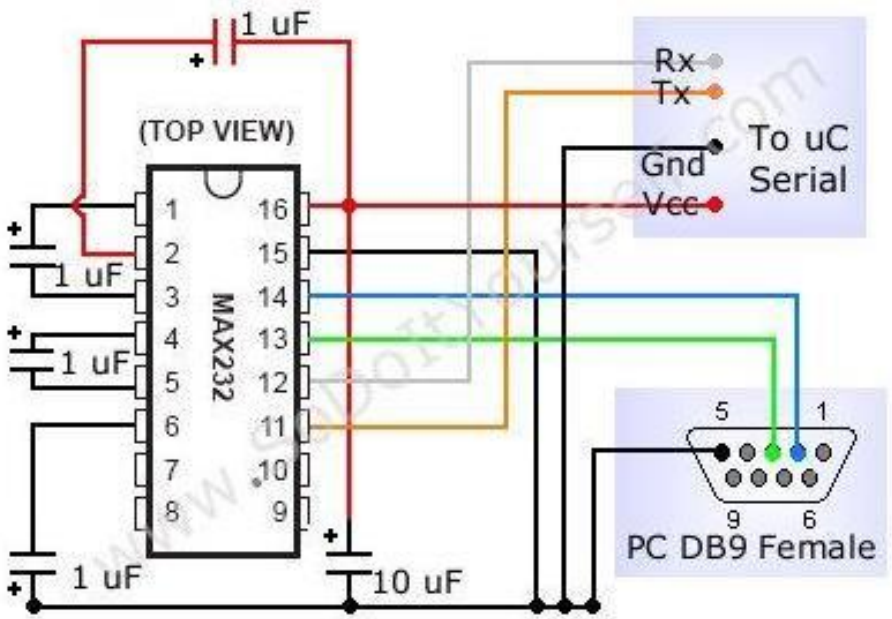
10.Package Options Include Plastic

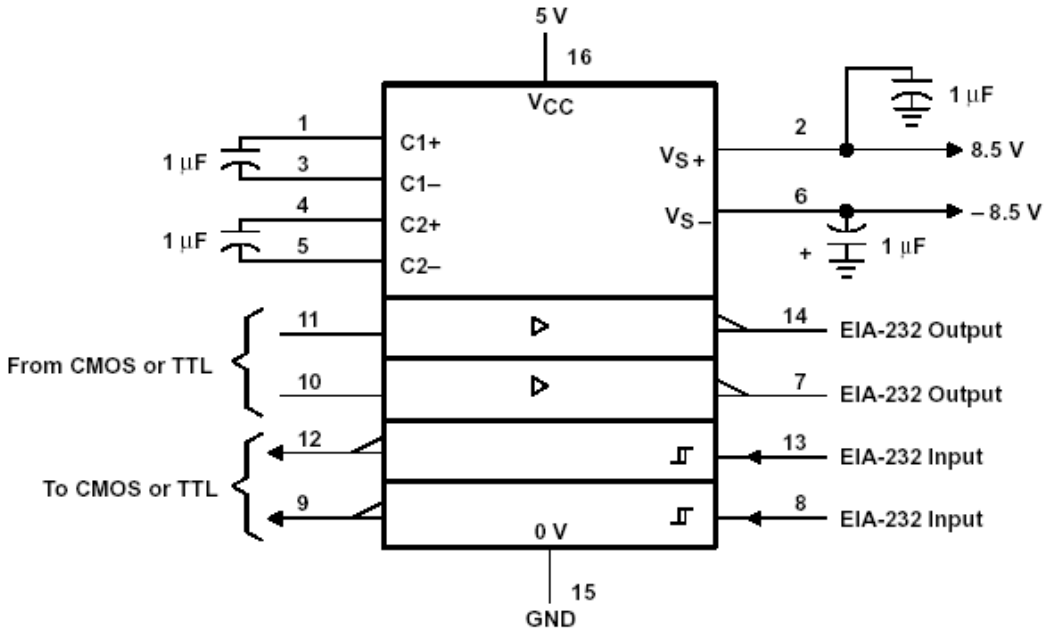
Small-Outline (D, DW) Packages and

Standard Plastic (N) DIPs

Circuit connections:

A standard serial interfacing for PC, RS232C, requires negative logic, i.e., logic '1' is -3V to -12V and logic '0' is +3V to +12V. To convert a TTL logic, say, TxD and RxD pins of the uC chips, thus need a converter chip. A MAX232 chip has long been using in many uC boards. It provides 2-channel RS232C port and requires external 10uF capacitors. Carefully check the polarity of capacitor when soldering the board. A DS275 however, no need external capacitor and smaller. Either circuit can be used without any problems.


1 uF
(TOP VIEW)
MAX232
1 uF
1 uF
1 uF
10 uF
Rx
Tx
Gnd
Vcc
To uC
Serial
PC DB9 Female


5 V
16
VCC
C1+
C1–
C2+
C2–
VS+
VS–
1 µF
1 µF
1 µF
1 µF
8.5 V
– 8.5 V
From CMOS or TTL
To CMOS or TTL
EIA-232 Output
EIA-232 Output
EIA-232 Input
EIA-232 Input
0 V
15
GND

GPS MODULE:

A GPS tracking unit is a device, normally carried by a moving vehicle or person, that uses the Global Positioning System to determine and track its precise location, and hence that of its carrier, at intervals. The recorded location data can be stored within the tracking unit, or it may be transmitted to a central location database, or Internet-connected computer, using a cellular (GPRS or SMS), radio, or satellite modem embedded in the unit. This allows the asset's location to be displayed against a map backdrop either in real time or when analyzing the track later, using GPS tracking software. Data tracking software is available for smartphones with GPS capability.

WORKING: -

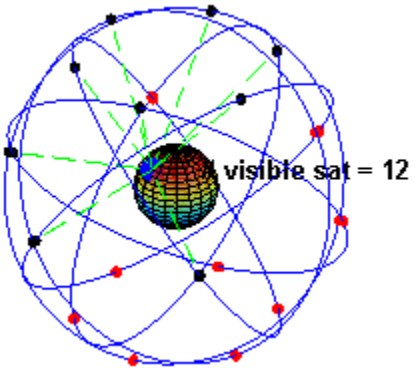
GPS receivers use a constellation of satellites and ground stations to compute position and time almost anywhere on earth.

At any given time, there are at least 24 active satellites orbiting over 12,000 miles above earth. The positions of the satellites are constructed in a way that the sky above your location will always contain at most 12 satellites. The primary purpose of the 12 visible satellites is to *transmit* information back to earth over radio frequency (ranging from 1.1 to 1.5 GHz). With this information and some math, a ground based *receiver* or GPS module can calculate its position and time.

The data sent down to earth from each satellite contains a few different pieces of information that allows your GPS receiver to accurately calculate its position and time. An important piece of equipment on each GPS satellite is an extremely accurate atomic clock. The time on the atomic clock is sent down to earth along with the satellite's orbital position and arrival times at different points in the sky. In other words, the GPS module receives a timestamp from each of the visible satellites, along with data on where in the sky each one is located (among other pieces of data). From this information, the GPS receiver now knows the distance to each satellite in view. **If the GPS receiver's antenna can see at least 4 satellites, it can accurately calculate its position and time.**

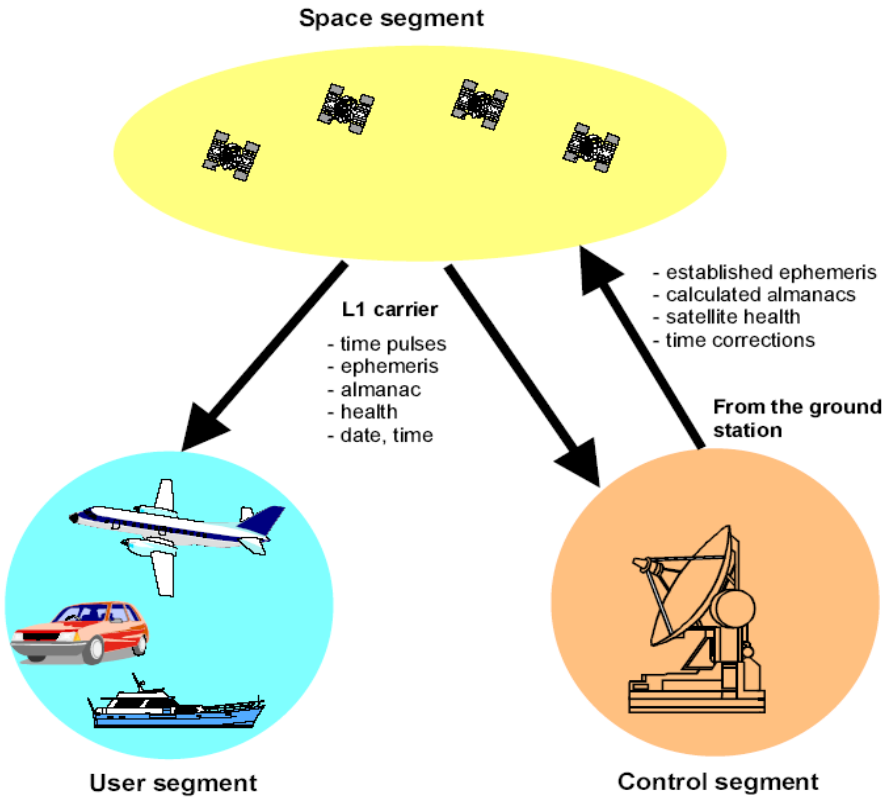
There is another piece of the global positioning system we haven't talked about. Along with satellites and GPS receivers, there are ground based stations that can communicate with the satellite network and some GPS receivers. This system is formally called the control segment and increases the accuracy of your GPS receiver. Common systems that use the control segment to improve accuracy are WAAS and DGPS. WAAS is common on most GPS receivers and improves accuracy to about 5 meters. DGPS requires a specific type of GPS receiver and gets centimeter accuracy. DGPS units are also expensive and tend to be larger because they require an additional antenna.

Space segment



A visual example of the GPS constellation in motion with the Earth rotating. Notice how the number of satellites in view from a given point on the Earth's surface, in this example at 45°N, changes with time.

The space segment (SS) comprises the orbiting GPS satellites, or Space Vehicles (SV) in GPS parlance. The GPS design originally called for 24 SVs, eight each in three circular orbital planes, but this was modified to six planes with four satellites each. The orbital



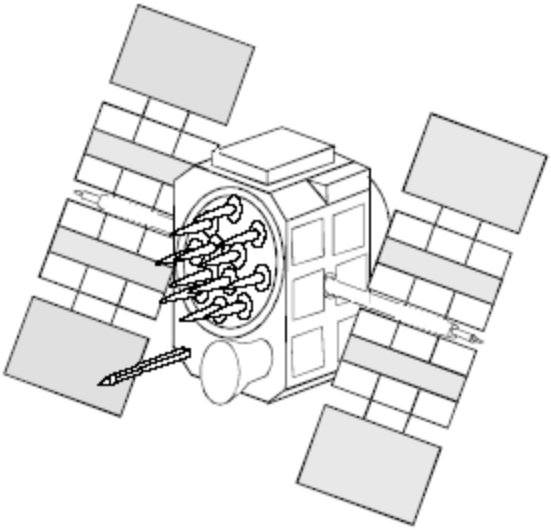


Figure 11: A GPS satellite

3.2.2.2 The communication link budget analysis

planes are centered on the Earth, not rotating with respect to the distant stars. The six planes have approximately 55° inclinations (tilt relative to Earth's equator) and are separated by 60° right ascension of the ascending node (angle along the equator from a reference point to the orbit's intersection). The orbits are arranged so that at least six satellites are always within line of sight from almost everywhere on Earth's surface.

Orbiting at an altitude of approximately 20,200 kilometers (12,600 miles or 10,900 nautical miles; orbital radius of 26,600 km (16,500 mi or 14,400 NM)), each SV makes two complete orbits each sidereal day. The ground track of each satellite therefore repeats each (sidereal) day. This was very helpful during development, since even with just four satellites, correct alignment means all four are visible from one spot for a few hours each day. For military operations, the ground track repeat can be used to ensure good coverage in combat zones.

As of March 2008, there are 31 actively broadcasting satellites in the GPS constellation. The additional satellites improve the precision of GPS receiver calculations by providing redundant measurements. With the increased number of satellites, the constellation was changed to a non-uniform arrangement. Such an arrangement was shown to improve reliability and availability of the system, relative to a uniform system, when multiple satellites fail.

Some reports in 2008 indicated that the 32nd satellite was causing difficulties for some GPS receivers.

Control segment

The flight paths of the satellites are tracked by US Air Force monitoring stations in Hawaii, Kwajalein, Ascension Island, Diego Garcia, and Colorado Springs, Colorado, along with monitor stations operated by the National Geospatial-Intelligence Agency (NGA). The tracking information is sent to the Air Force Space Command's master control station at Schriever Air Force Base in Colorado Springs, which is operated by the 2nd Space Operations Squadron (2 SOPS) of the United States Air Force (USAF). Then 2 SOPS contacts each GPS satellite regularly with a navigational update (using the ground antennas at Ascension Island, Diego Garcia, Kwajalein, and Colorado Springs). These updates synchronize the atomic clocks on board the satellites to within a few nanoseconds of each other, and adjust the ephemeris of each satellite's internal orbital model. The updates are created by a Kalman filter which uses inputs from the ground monitoring stations, space weather information, and various other inputs.

Satellite maneuvers are not precise by GPS standards. So to change the orbit of a satellite, the satellite must be marked 'unhealthy', so receivers will not use it in their calculation. Then the maneuver can be carried out, and the resulting orbit tracked from the ground. Then the new ephemeris is uploaded and the satellite marked healthy again.

User segment

PS receivers come in a variety of formats, from devices integrated into cars, phones, and watches, to dedicated devices such as those shown here from manufacturers Trimble, Garmin and Leica (left to right).

The user's GPS receiver is the user segment (US) of the GPS. In general, GPS receivers are composed of an antenna, tuned to the frequencies transmitted by the satellites, receiver-processors, and a highly-stable clock (often a crystal oscillator). They may also include a display for providing location and speed information to the user. A receiver is often described by its number of channels: this signifies how many satellites it can monitor simultaneously. Originally limited to four or five, this has progressively increased over the years so that, as of 2007, receivers typically have between 12 and 20 channels.

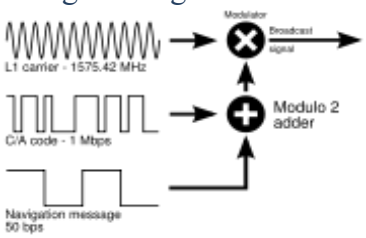
A typical OEM GPS receiver module measuring 15×17 mm.

GPS receivers may include an input for differential corrections, using the RTCM SC-104 format. This is typically in the form of a RS-232 port at 4,800 bit/s speed. Data is actually sent at a much lower rate, which limits the accuracy of the signal sent using RTCM. Receivers with internal DGPS receivers can outperform those using external RTCM data. As of 2006, even low-cost units commonly include Wide Area Augmentation System (WAAS) receivers.

A typical GPS receiver with integrated antenna.

Many GPS receivers can relay position data to a PC or other device using the NMEA 0183 protocol, or the newer and less widely used NMEA 2000. Although these protocols are officially defined by the NMEA, references to these protocols have been compiled from public records, allowing open source tools like gpsd to read the protocol without violating intellectual property laws. Other proprietary protocols exist as well, such as the SiRF and MTK protocols. Receivers can interface with other devices using methods including a serial connection, USB or Bluetooth.

Navigation signals



GPS broadcast signal

Each GPS satellite continuously broadcasts a **Navigation Message** at 50 bit/s giving the time-of-week, GPS week number and satellite health information (all transmitted in the first part of the message), an ephemeris (transmitted in the second part of the message) and an almanac (later part of the message). The messages are sent in frames, each taking 30 seconds to transmit 1500 bits.

The first 6 seconds of every frame contains data describing the satellite clock and its relationship to GPS time. The next 12 seconds contain the **ephemeris** data, giving the satellite's own precise orbit. The ephemeris is updated every 2 hours and is generally valid for 4 hours, with provisions for updates every 6 hours or longer in non-nominal conditions. The time needed to acquire the ephemeris is becoming a significant element of the delay to first position fix, because, as the hardware becomes more capable, the time to lock onto the satellite signals shrinks, but the ephemeris data requires 30 seconds (worst case) before it is received, due to the low data transmission rate.

The **almanac** consists of coarse orbit and status information for each satellite in the constellation, an ionosphere model, and information to relate GPS derived time to Coordinated Universal Time (UTC). A new part of the almanac is received for the last 12 seconds in each 30 second frame. Each frame contains 1/25th of the almanac, so 12.5 minutes are required to receive the entire almanac from a single satellite. The almanac serves several purposes. The first is to assist in the acquisition of satellites at power-up by allowing the receiver to generate a list of visible satellites based on stored position and time, while an ephemeris from each satellite is needed to compute position fixes using that satellite. In older hardware, lack of an almanac in a new receiver would cause long delays before providing a valid position, because the search for each satellite was a slow process. Advances in hardware have made the acquisition process much faster, so not having an almanac is no longer an issue. The second purpose is for relating time derived from the GPS (called GPS time) to the international time standard of UTC. Finally, the almanac allows a single frequency receiver to correct for ionosphere error by using a global ionosphere model. The corrections are not as accurate as augmentation systems like WAAS or dual frequency receivers. However, it is often better than no correction since ionosphere error is the largest error source for a single frequency GPS receiver. An important thing to note about navigation data is that each satellite transmits only its own ephemeris, but transmits an almanac for all satellites.

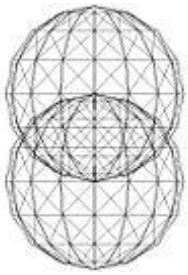
Each satellite transmits its navigation message with at least two distinct spread spectrum codes: The **Coarse / Acquisition (C/A) code**, which is freely available to the public, and the **Precise (P) code**, which is usually encrypted and reserved for military applications. The C/A code is a 1023 length Gold code at 1.023 million chips per second so that it repeats every millisecond. As pointed out in a chip is essentially the same thing as a bit and chips per second are the same as bits per second. The justification for coming up with this new term, chip, is that in some cases a sequence of bits is used as a type of Modulation and contains no information.

Each satellite has its own C/A code so that it can be uniquely identified and received separately from the other satellites transmitting on the same frequency. The P-code is a 10.23

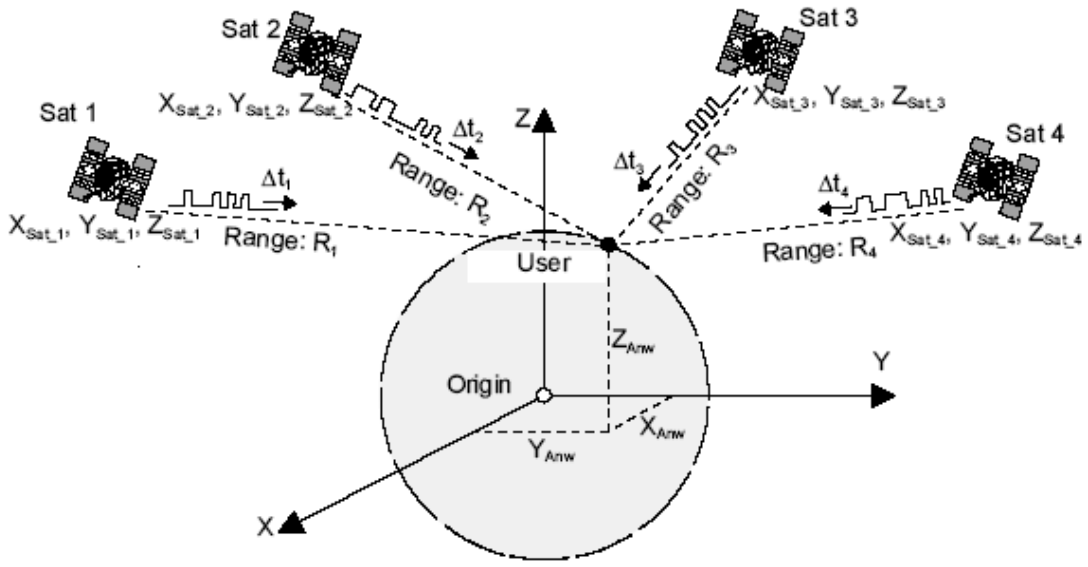
mega chip per second PRN code that repeats only every week. When the "anti-spoofing" mode is on, as it is in normal operation, the P code is encrypted by the **Y-code** to produce the **P(Y)** code, which can only be decrypted by units with a valid decryption key. Both the C/A and P(Y) codes impart the precise time-of-day to the user.

Position determination

Before providing a more mathematical description of position calculation, the introductory material on these topics is reviewed. To describe the basic concept of how a GPS receiver works, the errors are at first ignored. Using messages received from four satellites, the GPS receiver is able to determine the satellite positions and time sent. The x, y, and z components of position and the time sent are designated as $[x_i, y_i, z_i, t_i]$where the subscript i denotes which satellite and has the value 1, 2, 3, or 4. Knowing the indicated time the message was receivedtr_i, the GPS receiver can compute the indicated transit time, $(tr_i - t_i)$. of the message. Assuming the message traveled at the speed of light, c, the distance traveled, $p_i$can be computed as $(tr_i - t_i)\, c$. Knowing the distance from GPS receiver to a satellite and the position of a satellite implies that the GPS receiver is on the surface of a sphere centered at the position of a satellite. Thus we know that the indicated position of the GPS receiver is at or near the intersection of the surfaces of four spheres. In the ideal case of no errors, the GPS receiver will be at an intersection of the surfaces of four spheres. The surfaces of two spheres if they intersect in more than one point intersect in a circle. A figure, Two Sphere Surfaces Intersecting in a Circle, is shown below depicting this which hopefully will aid the reader in visualizing this intersection.



Two Sphere Surfaces Intersecting in a Circle



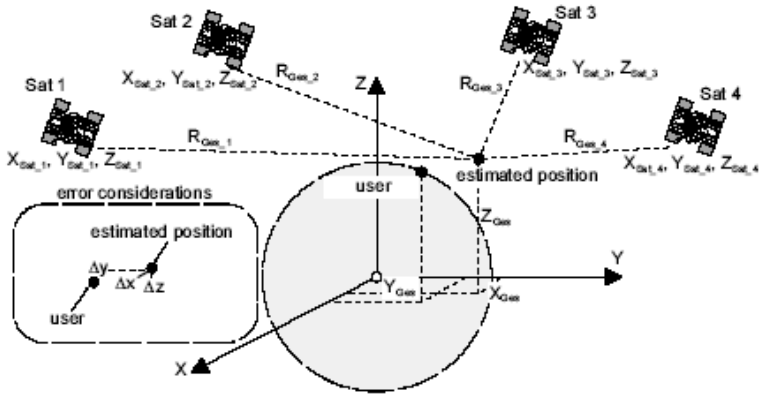
Three dimensional co-ordinate system

Generally (with $\Delta x = x - x_0$): $$f(x) = f(x_0) + \frac{f'}{1!}(x_0)\cdot\Delta x + \frac{f''}{2!}(x_0)^2\cdot\Delta x + \frac{f'''}{3!}(x_0)^3\cdot\Delta x + \ldots$$

Simplified (1st part only): $$f(x) = f(x_0) + f'(x_0)\cdot\Delta x \qquad (7a)$$

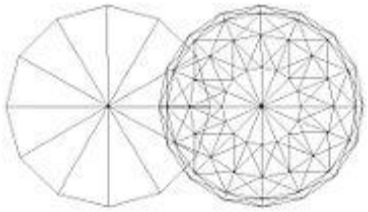
In order to linearise the four equations (6a), an arbitrarily estimated value x_0 must therefore be incorporated in the vicinity of x.

For the GPS system, this means that instead of calculating X_{Anw}, Y_{Anw} and Z_{Anw} directly, an estimated position X_{Ges}, Y_{Ges} and Z_{Ges} is initially used (Figure 23).



The article, trilateration, shows mathematically how the equation for a circle is determined. A circle and sphere surface in most cases of practical interest intersects at two points, although it is conceivable that they could intersect in 0 or 1 point. Another figure, Surface of Sphere Intersecting a Circle (not disk) at Two Points, is shown below to aid in visualizing this intersection. Again trilateration clearly shows this mathematically. The correct position of the GPS receiver is the one that is closest to the fourth sphere. This paragraph has described the

basic concept of GPS while ignoring errors. The next problem is how to process the messages when errors are present.



Surface of Sphere Intersecting a Circle (not disk) at Two Points

Let b denote the clock error or bias, the amount by which the receiver's clock is slow. The GPS receiver has four unknowns, the three components of GPS receiver position and the clock bias $[x, y, z, b]$. The equation of the sphere surfaces is given by:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = \big((tr_i + b - t_i)c\big)^2,\ i = 1, 2, 3, 4.$$

Another useful form of these equations is in terms of the pseudo range*s*, which are simply the ranges approximated based on GPS receiver clock's indicated (i.e. uncorrected) time so that $p_i = (tr_i - t_i)\, c$. Then the equations become:

$$p_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2)} - bc,\ i = 1, 2, 3, 4.$$

Two of the most important methods of computing GPS receiver position and clock bias are (1) trilateration followed by one dimensional numerical root finding and (2) multidimensional Newton-Raphson. These two methods along with their advantages are discussed.

- Solve by trilateration followed by one dimensional numerical root finding. This method involves using Trilateration to determine the intersection of the surfaces of three spheres. It is clearly shown in trilateration that the surfaces of three spheres intersect in 0, 1, or 2 points. In the usual case of two intersections, the solution which is nearest the surface of the sphere corresponding to the fourth satellite is chosen. The surface of the earth can also sometimes be used instead, especially in the case of civilian GPS receivers since it is illegal in the United States to track vehicles of more than 60,000 feet in altitude. The bias, b is then computed based on the distance from the solution to the surface of the sphere corresponding to the fourth satellite. Using an updated received time based on this bias, new spheres are computed and the process is repeated. One advantage of this method is that it involves one dimensional as opposed to multidimensional numerical root finding.

- Utilize multidimensional Newton-Raphson. Linearize around an approximate solution say $\left[x^{(k)}, y^{(k)}, z^{(k)}, b^{(k)}\right]$ from iteration k, then solve four linear equations derived from the quadratic equations above to obtain $\left[x^{(k+1)}, y^{(k+1)}, z^{(k+1)}, b^{(k+1)}\right]$. The radii are large and so the sphere surfaces are close to flat. This near flatness may cause the iterative procedure to converge rapidly in the case where $b$is near the correct value and the primary change is in the values of $x, y, \ and \ z$, since in this case the problem is merely to find the intersection of nearly flat surfaces and thus close to a linear problem. However when $b$is changing significantly, this near flatness does not appear to be advantageous in producing rapid convergence, since in this case these near flat surfaces will be moving as the spheres expand and contract. This possible fast convergence is an advantage of this method. Also it has been claimed that this method is the "typical" method used by GPS receivers. A disadvantage of this method is that according to, "There are no good general methods for solving systems of more than one nonlinear equation."

- Other methods include: Solve for the intersection of the expanding signals form light cones in 4-space cones, solve for the intersection of hyperboloids determined by the time difference of signals received from satellites utilizing multi lateration, Solve the equations in accordance with.

More than four satellites should be used, if available. This results in an over-determined system of equations with no unique solution, which must be solved by least-squares or a similar technique. If all visible satellites are used, the results are always at least as good as using the four best, and usually better. Also the errors in results can be estimated through the residuals. With each combination of four or more satellites, a geometric dilution of precision (GDOP) vector can be calculated, based on the relative sky positions of the satellites used. As more satellites are picked up, pseudo ranges from more combinations of four satellites can be processed to add more estimates to the location and clock offset. The receiver then determines which combinations to use and how to calculate the estimated position by determining the weighted average of these positions and clock offsets. After the final location and time are calculated, the location is expressed in a specific coordinate system such as latitude and longitude, using the WGS 84 geodetic datum or a local system specific to a country.

Finally, results from other positioning systems such as GLONASS or the upcoming Galileo can be used in the fit, or used to double check the result. (By design, these systems use the same bands; so much of the receiver circuitry can be shared, though the decoding is different.)

P(Y) code

Calculating a position with the P(Y) signal is generally similar in concept, assuming one can decrypt it. The encryption is essentially a safety mechanism: if a signal can be successfully decrypted, it is reasonable to assume it is a real signal being sent by a GPS satellite. In comparison, civil receivers are highly vulnerable to spoofing since correctly formatted C/A

signals can be generated using readily available signal generators. RAIM features do not protect against spoofing, since RAIM only checks the signals from a navigational perspective.

GPS APPLICATIONS: -

- Personal tracking
- Military-grade geotagging
- Vehicle tracking applications

USAGE IN OUR PROJECT: -

GPS Module

Latitude and longitude are usually provided in the geodetic datum on which GPS is based (WGS-84).

- ❖ Receivers can often be set to convert to other user-required datum.
- ❖ Receiver position is computed from the SV positions, the measured pseudo-ranges, and a receiver position estimate.
- ❖ Four satellites allow computation of three position dimensions and time.
- ❖ Three satellites could be used determine three position dimensions with a perfect receiver clock.
- ❖ In practice this is rarely possible and three SVs are used to compute a two-dimensional, horizontal fix (in latitude and longitude) given an assumed height.
- ❖ This is often possible at sea or in altimeter equipped aircraft.
- ❖ Five or more satellites can provide position, time and redundancy.
- ❖ Twelve channel receivers allow continuous tracking of all available satellites, including tracking of satellites with weak or occasionally obstructed signals.

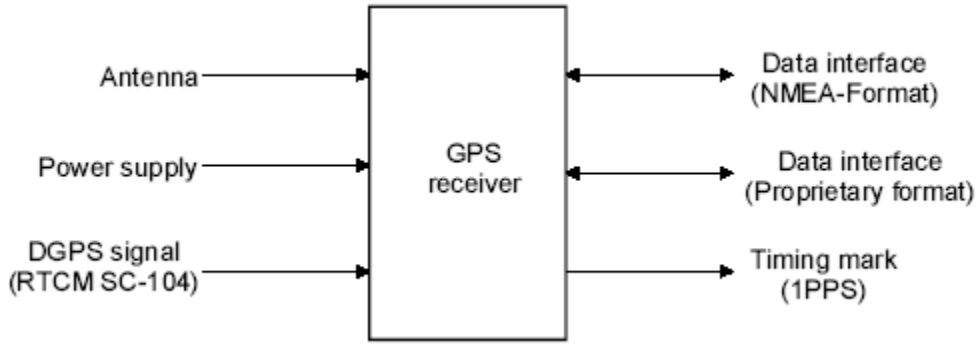
The GPS used follows NMEA 0183 version 3.0. The parameters like latitude, longitude, altitude and speed are received from GPS via RS232 these parameters are compare with the predefine wave points and when status of success is displayed on the LCD along with these instant latitude, longitude, altitude and speed parameters are also displayed on the LCD.

To achieve this, we are going to use one GPS module and one GSM module, which are connected to a micro controller unit. Whenever the user sends an SMS to the GSM modem, micro controller unit will get the appropriate request by sending standard AT commands, and sends request to GPS module from the micro controller in the form of NMEA standard command sentences, to get the vehicle longitude and latitude. After that by processing the received data in micro controller, the reverse SMS with position will send to the user's number from GSM modem.

The GPS module in this system will communicate with the satellite and receives its current position (vehicle position). If any theft occours we can lock the doors remotely again by sending SMS.

GPS Receivers

GPS receivers require different signals in order to function figure. These variables are broadcast after position and time have been successfully calculated and determined. To ensure that the different types of appliances are portable there are either international standards for data exchange (NMEA and RTCM), or the Manufacturer provides defined (proprietary) formats and protocols.



Block diagram of a GPS receiver with interfaces, Data interfaces, The NMEA-0183 data interface, In order to relay computed GPS variables such as position, velocity, course etc. to a peripheral (e.g. computer, screen, transceiver), GPS modules have a serial interface (TTL or RS - 232level).

The most important elements of receiver information are broad cast via this interface in a special data format. ThisformatisstandardisedbythenationalMarineElectronicsAssociation (NMEA) to ensure that data exchange takes place without any problems. Now-a-days, data is relayed according to the NMEA-0183 specification. NMEA has specified data sets for various applications e.g. GNSS (Global Navigation Satellite System), GPS, Loran, Omega, Transit and also for various manufacturers.

The following seven datasets are widely used with GPS module store lay GPS information [xv]:

1. GGA (GPS Fix Data, fixed data for the Global Positioning System)

2. GLL (Geographic Position–Latitude/Longitude)

3. GSA (GNSSDOP and Active Satellites, degradation of accuracy and the number of

   active satellites in the Global Satellite Navigation System)

4. GSV (GNSS Satellites in View, satellites in view in the Global

   Satellite Navigation System)

5. RMC (Recommended Minimum Specific GNSSD at a)

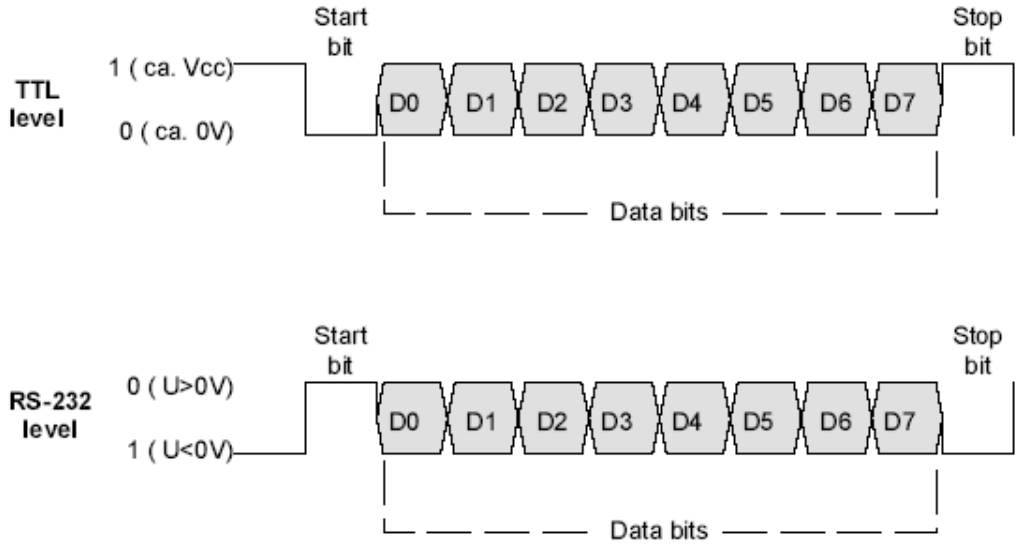
6. VTG (Course over Ground and Ground Speed, horizontal course and horizontal velocity)

7. ZDA (Time & Date)

   Structure of the NMEA protocol

In the case of NMEA, there at which data is transmitted is 4800 Baud using printable 8bit ASCII characters.

Transmission begin switch start bit (logical zero), followed by eight data bits and as to bit (logical one) added at the end. No parity bits are used.



NMEA format (TTL and RS-232 level)

The different levels must be take into consideration depending on whether the GPS receiver used has a TTL or RS-232interface (Figure)

• In the case of a TTL level interface, a logical zero corresponds to approx. 0V and a logical one roughly to the operating voltage of the system (+3.3V.+5V)

• In the case of an RS232 interface a logical zero corresponds to a positive voltage (+3V_. _+15V) and a Logical one a negative voltage (-3V... –15V).

If a GPS module with a TTL level interface is connected to an appliance with an RS-232 interface, a level conversion must be effected A few GPS modules all the baud rate to be increased (upto38400bitsper_second).

COMMANDS: -

- AT+CREG - Network registration

Set command controls the presentation of an unsolicited result code +CREG: when =1 and there is a change in the ME network registration status, or code +CREG: [,,] when =2 and there is a change of the network cell. Read command returns the status of result code presentation and an integer which shows whether the network has currently indicated the registration of the ME. Location information elements and are returned only when =2 and ME is registered in the network.

- AT+CMGF – Set SMS Text Mode or SMS PDU Mode

The AT+CMGF command sets the GSM modem in SMS Text Mode or SMS PDU Mode.

In Text Mode, SMS messages are represented as readable text. In PDU Mode, all SMS messages are represented as binary strings encoded in hexadecimal characters like **31020B911326880736F40000A900**.
Although Text Mode is easier to use, PDU Mode is more consistent on different GSM Modems.

- AT+CNMI - flash message to read message
- AT+CMGS – message sending

4 GPS receiver

Chip	MediaTek MT3318	
Frequency	L1 1575.42MHz, C/A code	
Channels	Support 32 channels	
Update rate	1Hz default, up to 5Hz	
Acquisition Time	Hot start (Open Sky)	2s (typical)
	Cold Start (Open Sky)	36s (typical)
Position Accuracy	Autonomous	3m (2D RMS)
	SBAS	2.5m (depends on accuracy of correction data)
Datum	WGS-84 (default)	
Max. Altitude	< 18,000 m	
Max. Velocity	< 515 m/s	

GSM MODULE: -

GSM/GPRS MODEM is a class of wireless modem devices that are designed for communication of a computer with the GSM and GPRS network. It requires a SIM (Subscriber Identity Module) card just like mobile phones to activate communication with the network.

A GSM/GPRS MODEM can perform the following operations:

1. Receive, send or delete SMS messages in a SIM.
2. Read, add, search phonebook entries of the SIM.
3. Make, Receive, or reject a voice call.

Wireless MODEMs are the MODEM devices that generate, transmit or decode data from a cellular network, for establishing communication between the cellular network and the computer

Wireless MODEMs like other MODEM devices **use serial communication** to interface with and need **Hayes compatible AT commands** for communication with the computer (any microprocessor or microcontroller system).

GSM system was developed as a digital system using time division multiple access (TDMA) technique for communication purpose. A GSM digitizes and reduces the data, then sends it down through a channel with two different streams of client data, each in its own particular time slot. The digital system has an ability to carry 64 kbps to 120 Mbps of data rates.

GSM Architecture

A GSM network consists of the following components:

- **A Mobile Station:** It is the mobile phone which consists of the transceiver, the display and the processor and is controlled by a SIM card operating over the network.
- **Base Station Subsystem:** It acts as an interface between the mobile station and the network subsystem. It consists of the Base Transceiver Station which contains the radio transceivers and handles the protocols for communication with mobiles. It also consists of the Base Station Controller which controls the Base Transceiver station and acts as an interface between the mobile station and mobile switching centre.
- **Network Subsystem:** It provides the basic network connection to the mobile stations. The basic part of the Network Subsystem is the Mobile Service Switching Centre which provides access to different networks like ISDN, PSTN etc. It also consists of the Home Location Register and the Visitor Location Register which provides the call routing and roaming capabilities of GSM. It also contains the Equipment Identity Register which maintains an account of all the mobile equipment's wherein each mobile is identified by its own IMEI number. IMEI stands for International Mobile Equipment Identity.

SIM 300 MODULE: -

SIM 300 is a GSM modem with a simple serial interface. SIM 300 modem can accept any GSM network operator SIM card and act just like a mobile phone with its own unique phone number. With this module one can send/receive sms, connect to internet via GPRS and receive calls. The modem can either be connected to PC serial port directly or to any microcontroller.

SIM300C with a tiny configuration can fit almost all the space requirements in your industrial applications, such as telemetry, telemetric and other mobile data communication systems.

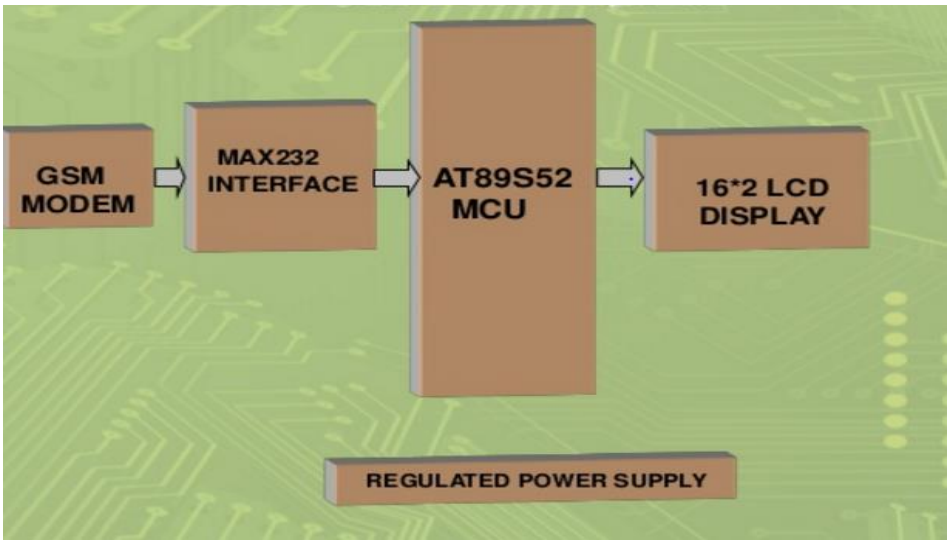
AT commands while using GSM/GPRS modules: -

Command	Description
AT+CMGD	DELETE SMS MESSAGE
AT+CMGF	SELECT SMS MESSAGE FORMAT
AT+CMGL	LIST SMS MESSAGES FROM PREFERRED STORE
AT+CMGR	READ SMS MESSAGE
AT+CMGS	SEND SMS MESSAGE
AT+CMGW	WRITE SMS MESSAGE TO MEMORY
AT+CMSS	SEND SMS MESSAGE FROM STORAGE
AT+CMGC	SEND SMS COMMAND
AT+CNMI	NEW SMS MESSAGE INDICATIONS
AT+CPMS	PREFERRED SMS MESSAGE STORAGE
AT+CRES	RESTORE SMS SETTINGS
AT+CSAS	SAVE SMS SETTINGS
AT+CSCA	SMS SERVICE CENTER ADDRESS
AT+CSCB	SELECT CELL BROADCAST SMS MESSAGES
AT+CSDH	SHOW SMS TEXT MODE PARAMETERS
AT+CSMP	SET SMS TEXT MODE PARAMETERS
AT+CSMS	SELECT MESSAGE SERVICE

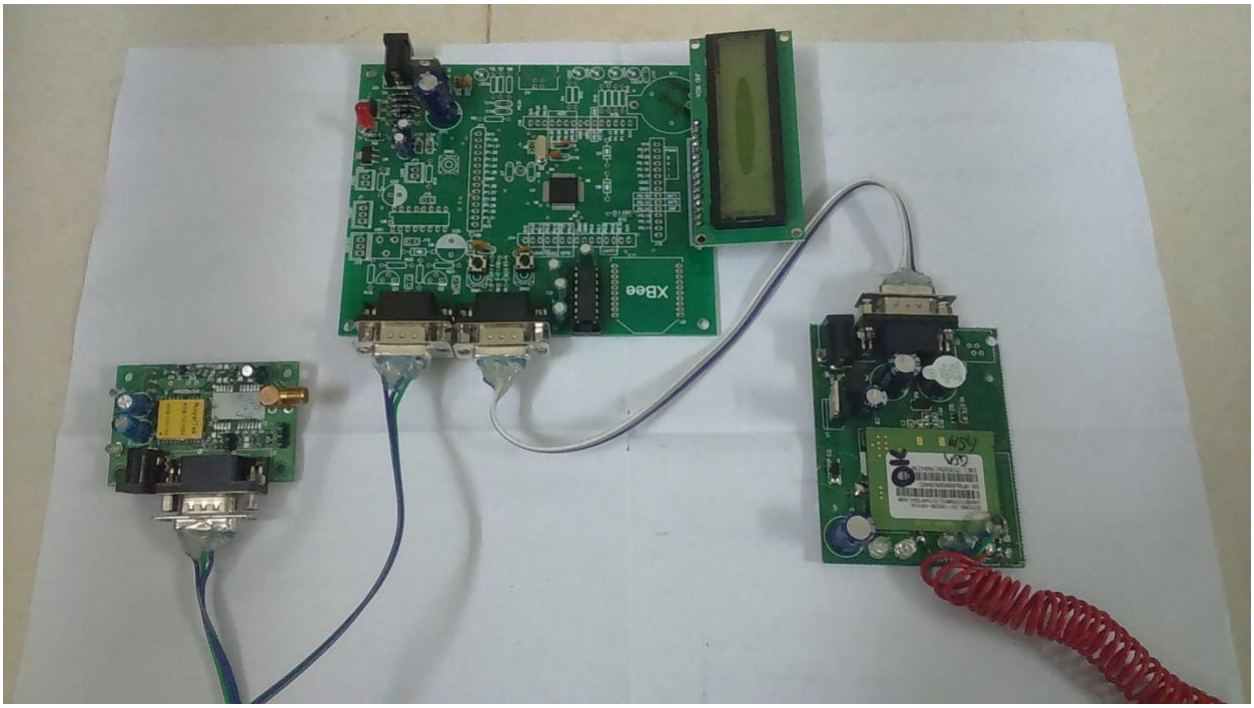
Working of GSM Module:

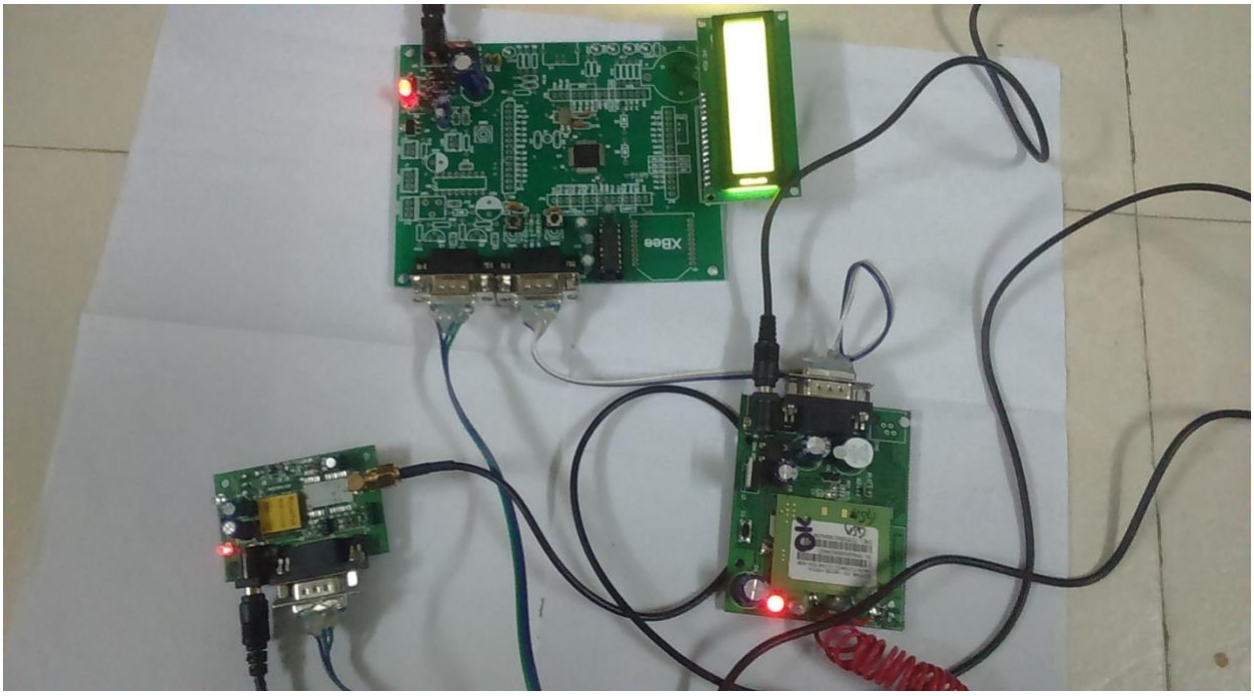
A GSM modem duly interfaced to the microcontroller(MC) through the level shifter IC Max232. The SIM card mounted GSM modem upon receiving digit command by SMS from any cell phone send that data to the MC through serial communication. While the program is executed, the GSM modem receives command ‘STOP’ to develop an output at the MC, the contact point of which are used to disable the ignition switch. The command so sent by the user is based on an intimation received by him through the GSM modem ‘ALERT’ a programmed message only if the input is driven low. The complete operation is displayed over 16×2 LCD display.

Block diagram:



CONNECTIONS:





WORKING: -

After making all the connections, we switch on the modules i.e. GPS, GSM, LCD display.Upload the code given below

```
#include<lpc214x.h>
#include<string.h>
#include<stdio.h>
////////////////////////////////////////////////////////////////////////////
#define Fosc            12000000
#define Fcclk           (Fosc * 5)
#define Fcco            (Fcclk * 4)
#define Fpclk           (Fcclk / 4) * 1
#define UART_BPS1 9600                  //Set buadrate here     ///gsm modem
#define UART_BPS0 9600                  //Set buadrate here     ///gps modem
////////////////////////////////////////////////////////////////////////////
unsigned int C[6]={0x20,0x28,0x01,0x0e,0x06,0x80};
unsigned int RS=0X01000000;                             ////P1^24 pin is connected to lcd rs pin
unsigned int EN=0X00400000;
////P1^22 pin is connected to lcd en pin
unsigned int PORT=0X00003C00;                           ///lcd data pins are P0^10,11,12,13
////////////////////////////////////////////////////////////////////////////
void LCD_INIT(void);                                    ///lcd initalization
void LCD_CMD(unsigned  char );                          //comand mode rs=0;en=0;and en=1;
void LCD_DATA(unsigned char );                          //data mode rs=1;en=0;and en=1;
void LCD_DAISPLAY(unsigned char * );                    //lcd display string data
////////////////////////////////////////////////////////////////////////////
void  Init_UART1(void);                                 //uart1 for gsm modem intiliztion and set the baudrate for 9600
void  UART1_SendByte(unsigned char data);           //commad should passing
void  UART1_SendStr(const unsigned char *str);     //gsm send the string files to the mobiles
////////////////////////////////////////////////////////////////////////////
void  Init_UART0(void);                                 //uarto for gps modem intiatiztion to set the baudrate for 9600
void  UART0_SendByte(unsigned char data);           //gps modem to trace the data $GPRMC,$GPGGA    DATA
void  UART0_SendStr(const unsigned char *str); //GPS DATA PASS TO GSM
////////////////////////////////////////////////////////////////////////////
void sendsms(unsigned char *,unsigned char *);      /// SENDING THE INFORMATION TO MOBILES
unsigned char msgdelete(unsigned char sim);         ///CONTROLLER SEND THE COMMAND FOR SIM TO DELETE THE MESSAGES
void msgread(unsigned char sim);    // READ THE MEEAGES IN SIM CARD FIRST MESSAGE
unsigned char msgindicator(void);   ///SIM SHOW THE MESSAGES TO THE MODEM AND MICROCONTROOLER
unsigned char  ONE (unsigned char count);   ///FIRST MESSAGE
void gps(void);                                 ///GPS RECIVE THE DATA FROM SATILITE
```

```
unsigned char LAT[12];                                  //        12 DIGIT  LATITUDE
unsigned char LONG[12];                                 //  12 DITAL LONGITUDE
void dis(void);                                 ///DISPLAY THE GPS DATA
void gsm_fun(void);                             ///GSM FUNCTION SHOULD SEND  THE
void gsm_init(void);                    // GSM INTIALIZTION COMMANS
unsigned char flag;
unsigned char str1[30];
unsigned char str2[30];

void DELAY(unsigned int);

unsigned char *CMD[]={{"AT"},
                                        {"AT+CMGF=1"},
                                        { "AT+CNMI=1,1,0,0,0"},
                                         {"AT+CMGR="},
                                         {"AT+CMGD="},
                                         {"AT+CMGS="},
                                         "ATE0"         };

unsigned char *usr[]={          {"AT+OK"},
                                                {"AT+CMGF=1+OK"},
                                                {"AT+CNTI+OK"},
                                                {"AT+CMGD+OK"},
                                                {"MESSAGE SENT"},
                                                {"IN VALID MESSGE"}
                                        };

unsigned char NMI[]={"NEW MESSAGE  "};
unsigned char MOBILE[]={"  MOBILE  NO   "};
unsigned char MSG[]={"MESSAGE  "};
unsigned char sent[]={"  MESSAGE SENT  "};

void SEND_UART1(unsigned char *);
void SEND_CHAR(unsigned char);

int main(void)
{
PINSEL0 = 0x00050005;       // UART 1 and 0
PINSEL1=0x15400000;
IODIR0|=0X00003c00;   /*(pin p0.10,11,12,13 are used for LCD data line)*/
IODIR1=0XF3000000;   /*(pin p1.24,p1.25 are used for reg:select and Enable)*/
                /*(0/P P1.28 TO P1.31 DEVICES) */

Init_UART1();
Init_UART0();
LCD_INIT();
```

```
DELAY(100);

LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY("GPS--GSM BASED");
LCD_CMD(0xC0);
LCD_DAISPLAY("INTEGRATION FOR");

DELAY(500);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY("ENHANCING PUBLIC");
LCD_CMD(0xC0);
LCD_DAISPLAY("TRANSPORTATION SYS");
gsm_init();
DELAY(500);
while(1)
{
gsm_fun();
}
}
////////////////////////////////////////////////////////////////////////////
void gsm_fun()
{
unsigned char sim,check;
LCD_CMD(0x01);
LCD_DAISPLAY("WAITING FOR SMS");
sim=msgindicator();
msgread(sim);
do
{
check=msgdelete(sim);
}while(check==0);
UART1_SendStr("AT+CMGD=1");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=2");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=3");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=4");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=5");
```

```
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=6");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=7");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=8");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
UART1_SendStr("AT+CMGD=9");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
DELAY(500);
if(!strcmp(str2,"BUSTRACK"))
{
gps();
sendsms(LAT,LONG);
DELAY(2000);
}
}
////////////////////////////////////////////////////////////////////////
void SEND_CHAR(unsigned char p)
{
U1THR=p;
while(!(U1LSR&0X20));
}
////////////////////////////////////////////////////////////////////////
void SEND_UART1(unsigned char *p)
{
unsigned int N,len;
len=strlen(p);
for(N=0;N<=len;N++)
{
U1THR=*p;
while(!(U1LSR&0X20));
p++;
}
}
////////////////////////////////////////////////////////////////////////
/*********************************************/
unsigned char ONE (unsigned char count)
{

unsigned char ch;
```

```
unsigned  char x=0;
unsigned char str0[50];
DELAY(200);
SEND_UART1(CMD[count]);
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
while(!(U1LSR&0X01));
ch=U1RBR;
while(ch!='K')
{
str0[x++]=ch;
while(!(U1LSR&0X01));
ch=U1RBR;
}
if(ch!='K')
{
return(0);
}
return(1);
}
//////////////////////////////////////////////////////////////////////
void gsm_init(void)
{
unsigned char check;
do
{
check=ONE(0);
}while(check==0);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(usr[0]);
DELAY(100);
do
{
check=ONE(6);
}while(check==0);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(usr[1]);
DELAY(100);
do
{
check=ONE(1);
}while(check==0);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
```

```
LCD_DAISPLAY(usr[1]);
DELAY(100);
do
{
check=ONE(2);
}while(check==0);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(usr[2]);
DELAY(100);
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY("GSM Initialized");
DELAY(200);
}
////////////////////////////////////////////////////////////////////////
unsigned char msgindicator(void)
{
unsigned char str0[50];
unsigned char ch,sim;
unsigned  char x=0;
for(x=0;x<13;x++)
{
while(!(U1LSR&0X01));
ch=U1RBR;
str0[x]=ch;
}
while(ch!=0x0d)
{
str0[x++]=ch;
while(!(U1LSR&0X01));
ch=U1RBR;
}
str0[x]='\0';
LCD_CMD(0x01);                                          //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(NMI);
DELAY(200);
sim=str0[--x];
return(sim);
}
////////////////////////////////////////////////////////////////////////
void msgread(unsigned char sim)
{
unsigned char ch;
unsigned  char x=0;
```

```
DELAY(200);
SEND_UART1(CMD[3]);
SEND_CHAR(sim);
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
while(!(U1LSR&0X01));
ch=U1RBR;
while(ch!=':')
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
while(ch!=':')
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
while(ch!=':')
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
for(x=0;x<15;x++)
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
for(x=0;x<13;x++)
{
while(!(U1LSR&0X01));
ch=U1RBR;
str1[x]=ch;
}
str1[x]='\0';
while(ch!=0x0d)
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
while(!(U1LSR&0X01));
ch=U1RBR;
x=0;
while(ch!=0x0d)
{
while(!(U1LSR&0X01));
ch=U1RBR;
```

```
str2[x++]=ch;
}
x--;
str2[x]='\0';
LCD_CMD(0x01);                                                    //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(MOBILE);
                                                                  //title 1line
LCD_CMD(0xC0);
LCD_DAISPLAY(str1);
DELAY(200);DELAY(200);DELAY(200);

LCD_CMD(0x01);                                                    //title 1line
LCD_CMD(0x80);
LCD_DAISPLAY(MSG);
DELAY(200);
LCD_CMD(0xC0);
LCD_DAISPLAY(str2);
DELAY(200);
LCD_CMD(0x01);
 LCD_CMD(0x80);
LCD_DAISPLAY("MESSAGE RECEIVED");
DELAY(200);DELAY(200);DELAY(200);
}
////////////////////////////////////////////////////////////////////////////
unsigned char msgdelete(unsigned char sim)
{
unsigned char ch;
unsigned  char x=0;
unsigned char str0[50];
DELAY(100);
SEND_UART1(CMD[4]);
SEND_CHAR(sim);
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
while(!(U1LSR&0X01));
ch=U1RBR;
while(ch!='K')
{
str0[x++]=ch;
while(!(U1LSR&0X01));
ch=U1RBR;
}
if(ch!='K')
{
return(0);
```

```
}
return(1);
}
////////////////////////////////////////////////////////////////////////////
void sendsms(unsigned char *s,unsigned char *p)
{
unsigned char ch;
unsigned  char x=0;
unsigned char send[30];
unsigned char N,len;
SEND_CHAR('A');
SEND_CHAR('T');
SEND_CHAR('+');
SEND_CHAR('C');
SEND_CHAR('M');
SEND_CHAR('G');
SEND_CHAR('S');
SEND_CHAR('=');
SEND_CHAR("");
//////////////////////////////
SEND_CHAR(str1[3]);
SEND_CHAR(str1[4]);
SEND_CHAR(str1[5]);
SEND_CHAR(str1[6]);
SEND_CHAR(str1[7]);
SEND_CHAR(str1[8]);
SEND_CHAR(str1[9]);
SEND_CHAR(str1[10]);
SEND_CHAR(str1[11]);
SEND_CHAR(str1[12]);
//////////////////////////////
SEND_CHAR("");
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
while(!(U1LSR&0X01));
ch=U1RBR;
while(ch!='>')
{
while(!(U1LSR&0X01));
ch=U1RBR;
}
UART1_SendByte('L');
UART1_SendByte('T');
len=strlen(s);
for(N=0;N<=len;N++)
{
```

```
U1THR=*s;
while(!(U1LSR&0X20));
s++;
}
UART1_SendByte('\r');
UART1_SendByte('\n');
UART1_SendByte('L');
UART1_SendByte('G');
len=strlen(p);
for(N=0;N<=len;N++)
{
U1THR=*p;
while(!(U1LSR&0X20));
p++;
}
UART1_SendByte('\r');
UART1_SendByte('\n');
UART1_SendByte('B');
UART1_SendByte('U');
UART1_SendByte('S');
SEND_CHAR(0X1A);
SEND_CHAR(0X0D);
SEND_CHAR(0X0A);
while(!(U1LSR&0X01));
ch=U1RBR;
while(ch!=0x0d)
{
send[x++]=ch;
while(!(U1LSR&0X01));
ch=U1RBR;
}
send[x]='\0';
LCD_CMD(0x01);
 LCD_CMD(0x80);
LCD_DAISPLAY(sent);
DELAY(200);DELAY(200);DELAY(200);
}
////////////////////////////////////////////////////////////////////////////////////////////
void  Init_UART1(void)                                  //This function setups UART1
{
unsigned int Baud16;
U1LCR = 0x83;                           // DLAB = 1
Baud16 = (Fpclk / 16) / UART_BPS1;
U1DLM = Baud16 / 256;
U1DLL = Baud16 % 256;
U1LCR = 0x03;
```

```
}
void  UART1_SendByte(unsigned char data)                    //A  function  to  send  a  byte  on
UART1
{
U1THR = data;
while( (U1LSR&0x40)==0 );
DELAY(20);
}
void  UART1_SendStr(const unsigned char *str)               //A  function  to  send  a  string  on
UART1
{
while(1)
{
if( *str == '\0' ) break;
UART1_SendByte(*str++);
}
}
////////////////////////////////////////////////////////////////////////////////////////
void  Init_UART0(void)                                      //This function setups UART1
{
unsigned int Baud16;
U0LCR = 0x83;                           // DLAB = 1
Baud16 = (Fpclk / 16) / UART_BPS0;
U0DLM = Baud16 / 256;
U0DLL = Baud16 % 256;
U0LCR = 0x03;
}
void  UART0_SendByte(unsigned char data)                    //A  function  to  send  a  byte  on
UART1
{
U0THR = data;
while( (U0LSR&0x40)==0 );
}
void  UART0_SendStr(const unsigned char *str)               //A  function  to  send  a  string  on
UART1
{
while(1)
{
if( *str == '\0' ) break;
UART0_SendByte(*str++);
}
}
////////////////////////////////////////////////////////////////////////////////////////
void LCD_INIT()
{
unsigned int i;
```

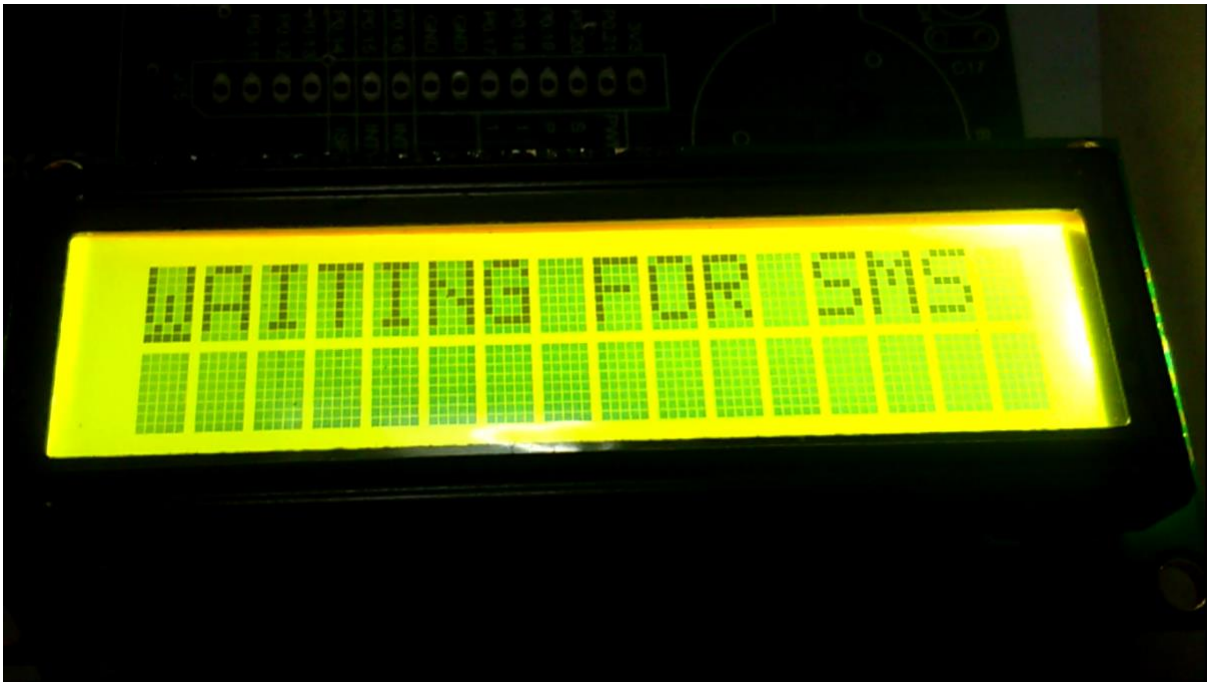
```
IODIR0|=PORT;
IODIR1|=0X01400000;

for(i=0;i<6;i++)
{
LCD_CMD(C[i]);
}
}
////////////////////////////////////////////////////////////////////////
void LCD_CMD(unsigned  char x)
{
unsigned  int data;
IOCLR0=PORT;
data=x;
data=data & 0xf0;
data=data<<6;
IOCLR1=RS;
IOSET0=data;
IOSET1=EN;
DELAY(100);
IOCLR1=EN;
IOCLR0=PORT;
DELAY(100);
data=x;
data=data & 0x0F;
data=data<<10;
IOCLR1=RS;
IOSET0=data;
IOSET1=EN;
DELAY(100);
IOCLR1|=EN;
IOCLR0|=PORT;
}
////////////////////////////////////////////////////////////////////////
void LCD_DATA(unsigned char x)
{
unsigned int data;
IOCLR0=PORT;
data=x;
data=data & 0xf0;
data=data<<6;
IOSET1=RS;
IOSET0=data;
IOSET1=EN;
DELAY(100);
DELAY(5000);
```

```
IOCLR1=EN;
IOCLR0=PORT;
DELAY(100);
data=x;
data=data & 0x0F;
data=data<<10;
IOSET1=RS;
IOSET0=data;
IOSET1=EN;
DELAY(100);
IOCLR1=EN;
IOCLR0=PORT;
}
////////////////////////////////////////////////////////////////////////////
void LCD_DAISPLAY(unsigned char *data)
{
while(*data)
{
LCD_DATA(*data);
data++;
}
LCD_CMD(0x80);
}
////////////////////////////////////////////////////////////////////////////
void DELAY(unsigned int val )
{
unsigned int i;
while(val--)
for(i=0;i<500;i++);
}
////////////////////////////////////////////////////////////////////////////
void gps(void)
{
unsigned char gp='\0',i,CH1;
while(gp!='$')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
gp='\0';
while(gp!='G')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
gp='\0';
```

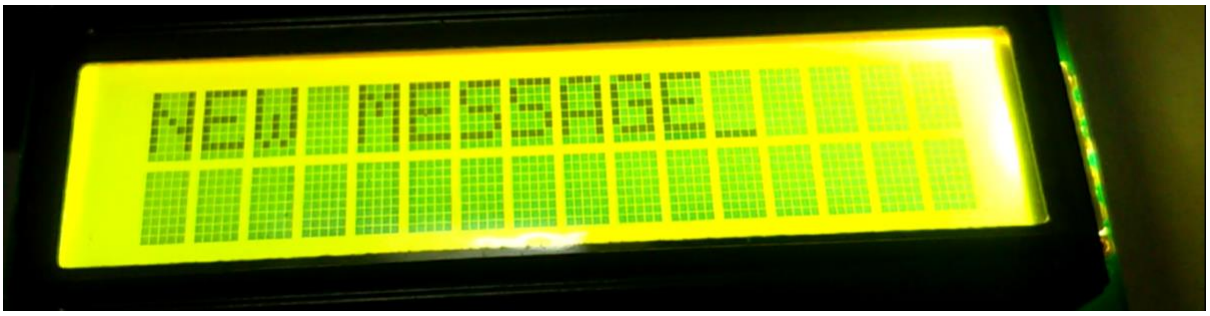
```
while(gp!='P')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
gp='\0';
while(gp!='R')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
gp='\0';
while(gp!='M')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
gp='\0';
while(gp!='C')
{
while(!(U0LSR&0x01));
gp=U0RBR;
}
while(!(U0LSR&0x01));
CH1=U0RBR;
for(i=0;i<13;i++)
{
while(!(U0LSR&0x01));
CH1=U0RBR;
}
for(i=0;i<11;i++)
{
while(!(U0LSR&0x01));
LAT[i]=U0RBR;
}
while(!(U0LSR&0x01));
CH1=U0RBR;
while(!(U0LSR&0x01));
CH1=U0RBR;
for(i=0;i<11;i++)
{
while(!(U0LSR&0x01));
LONG[i]=U0RBR;
}
LAT[11]='\0';
LONG[11]='\0';
```

```
for(i=0;i<11;i++)
{
while(!(U0LSR&0x01));
}
for(i=0;i<6;i++)
{
while(!(U0LSR&0x01));
}
dis();
}
////////////////////////////////////////////////////////////////////////////////
void dis(void)
{
LCD_CMD(0x80);
LCD_DAISPLAY("LAT: ");
LCD_CMD(0x85);
LCD_DAISPLAY(LAT);
LCD_CMD(0xC0);
LCD_DAISPLAY("LON: ");
LCD_CMD(0xC5);
LCD_DAISPLAY(LONG);
DELAY(100);DELAY(100);DELAY(100);
}
```

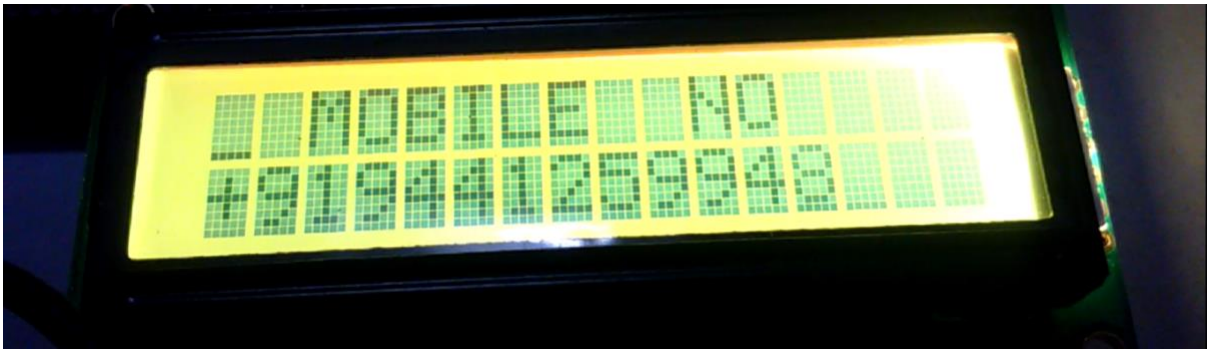
Then it is



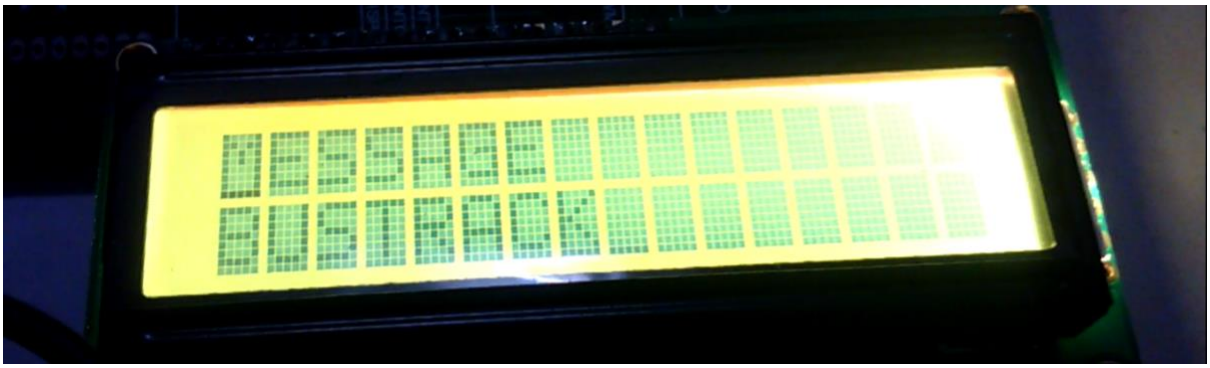
When we send a message “BUSTRACK” to the phone number of the sim card in GSM.



The number from which message has been sent will appear:



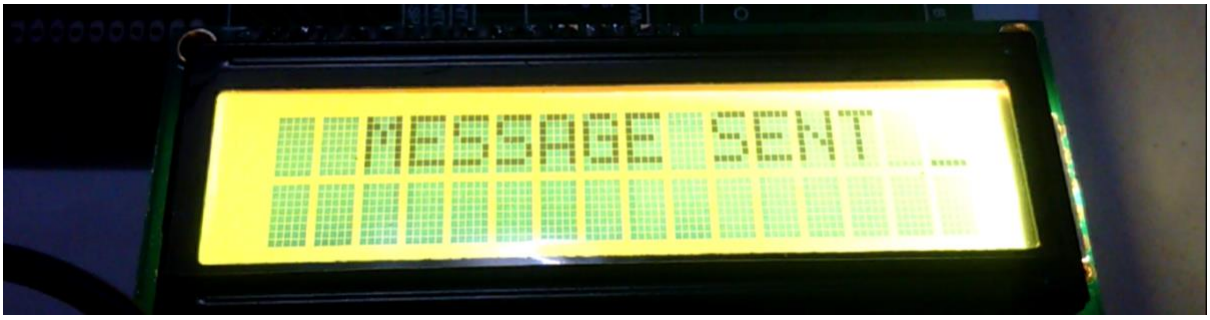
The received message will be displayed:



The acknowledgement will be displayed:

Latitude and longitudinal values will be sent through the GSM module to the mobile phone from which the message was sent:


LAT: 1253.8572,N
LON: 7703.5933,E


MESSAGE SENT...