Using the USART of AVR Microcontrollers.

Welcome to the third part of my RS232 serial communication tutorial. Till now we saw the basics of RS232 communication and made our level converter. Now its time to understand the USART of AVR microcontroller and write the code to initialize the USART and use it to send and receive data.

Like many microcontrollers AVR also has a dedicated hardware for serial communication this part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. This special hardware make your life as programmer easier. You just have to supply the data you need to transmit and it will do the rest. As you saw serial communication occurs at standard speeds of 9600,19200 bps etc and this speeds are slow compared to the AVR CPUs speed. The advantage of hardware USART is that you just need to write the data to one of the registers of USART and your done, you are free to do other things while USART is transmitting the byte.

Also the USART automatically senses the start of transmission of RX line and then inputs the whole byte and when it has the byte it informs you(CPU) to read that data from one of its registers.

The USART of AVR is very versatile and can be setup for various different mode as required by your application. In this tutorial I will show you how to configure the USART in a most common configuration and simply send and receive data. Later on I will give you my library of USART that can further ease you work. It will be little complicated (but more useful) as it will have a FIFO buffer and will use interrupt to buffer incoming data so that you are free to anything in your main() code and read the data only when you need. All data is stored into a nice FIFO(first in first out queue) in the RAM by the ISR.

USART of AVR Microcontrollers.

The USART of the AVR is connected to the CPU by the following six registers.

- **UDR** - USART Data Register : Actually this is not one but two register but when you read it you will get the data stored in receive buffer and when you write data to it goes into the transmitters buffer. **This important to remember it.**
- **UCSRA** - USART Control and status Register A : As the name suggests it is used to configure the USART and it also stores some status about the USART. There are two more of this kind the **UCSRB and UCSRC**.
- **UBRRH** and **UBRRH** : This is the USART Baud rate register, it is 16BIT wide so UBRRH is the High Byte and UBRRL is Low byte. But as we are using C language it is directly available as UBRR and compiler manages the 16BIT access.

So the connection of AVR and its internal USART can be visualized as follows.

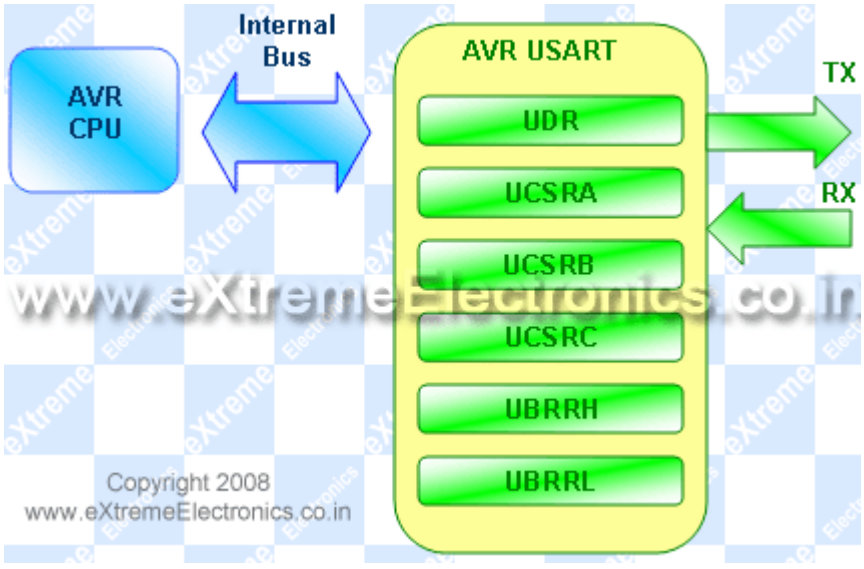


Fig- AVR USART registers.

Registers Explained

In order to write programs that uses the USART you need to understand what each register's importance. The scheme behind using the AVR USART is same as with any other internal peripheral (say ADC). So if you are new to this topic please see this tutorial, it shows you the basic idea of using peripherals.

I am not going to repeat what is already there in the datasheets, I will just tell about what is required for a quick startup. The datasheets of AVR provides you with all the details of every bit of every register so please refer to it for detailed info. Note bit names with **RED** background are of our interest here.

UDR: Explained above.

UCSRA: USART Control and Status Register A

****

Bit No	7	6	5	4	3	2	1	0
Name	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Initial Val	0	0	1	0	0	0	0	0

RXC this bit is set when the USART has completed receiving a byte from the host (may be your PC) and the program should read it from **UDR**

TXC This bit is set (1) when the USART has completed transmitting a byte to the host and your program can write new data to USART via UDR

UCSRB: USART Control and Status Register B

****

Bit No	7	6	5	4	3	2	1	0
Name	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Initial Val	0	0	0	0	0	0	0	0

RXCIE: Receive Complete Interrupt Enable - When this bit is written one the the associated interrupt is enabled.

TXCIE: Transmit Complete Interrupt Enable - When this bit is written one the the associated interrupt is enabled.

RXEN: Receiver Enable - When you write this bit to 1 the USART receiver is enabled. **The normal port functionality of RX pin will be overridden.** So you see that the associated I/O pin now switch to its secondary function,i.e. RX for USART.

TXEN: Transmitter Enable - As the name says!

UCSZ2: USART Character Size - Discussed later.

For our first example we won't be using interrupts so we set UCSRB as follows

```
UCSRB=(1<<RXEN)|(1<<TXEN);
```

UCSRC: USART Control And Status Register C

****

Bit No	7	6	5	4	3	2	1	0
Name	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Initial Val	0	0	0	0	0	0	0	0

IMPORTANT : The UCSRC and the UBRRH (discussed below) register shares same address so to determine which register user want to write is decided with the 7th(last) bit of data if its 1 then the data is written to UCSRC else it goes to UBRRH. This seventh bit is called the

URSEL: USART register select.

UMSEL: USART Mode Select - This bit selects between asynchronous and synchronous mode. As asynchronous mode is more popular with USART we will be using that.

UMSEL	Mode
0	Asynchronous
1	Synchronous

USBS: USART Stop Bit Select - This bit selects the number of stop bits in the data transfer.

USBS	Stop Bit(s)
0	1 BIT
1	2 BIT

UCSZ: USART Character size - These three bits (one in the UCSRB) selects the number of bits of data that is transmited in each frame. Normally the unit of data in MCU is 8BIT (C type "char") and this is most widely used so we will go for this. Otherwise you can select 5,6,7,8 or 9 bit frames!

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5Bit
0	0	1	6Bit
0	1	0	7Bit
0	***1***	***1***	***8Bit***
1	0	0	Reserved

1	0	1	Reserved
1	1	0	Reserved
1	1	1	9Bit

So we set UCSRC as follows

```
UCSRC=(1<<URSEL)|(3<<UCSZ0);
```

UBRR: USART Baud Rate Register:

****

This is the USART Baud rate register, it is 16BIT wide so **UBRRH** is the High Byte and **UBRRL** is Low byte. But as we are using C language it is directly available as UBRR and compiler manages the 16BIT access. This register is used by the USART to generate the data transmission at specified speed (say 9600Bps). To know about baud rate see the previous tutorial. UBRR value is calculated according to following formula.

$$UBRR = \frac{f_{osc}}{16 \times \text{Baud Rate}} - 1$$

Where fosc is your CPU frequency say 16MHz

Baud Rate is the required communication speed say 19200 bps (see previous tutorial for more info).

Example:

For above configuration our UBRR value comes to be 51.083 so we have to set

```
UBRR=51;
```

in our initialization section. Note UBRR can hold only integer value. So it is better to use the baud rates that give UBRR value that are purely integer or very close to it. So if your UBRR value comes to be 7.68 and you decided to use UBRR=8 then it has high error percentage, and **communication is unreliable**!

You may also use our **Androd App** for **calculating the UBRR** much easily ! It can run on Smartphones and Tablets running Android OS.

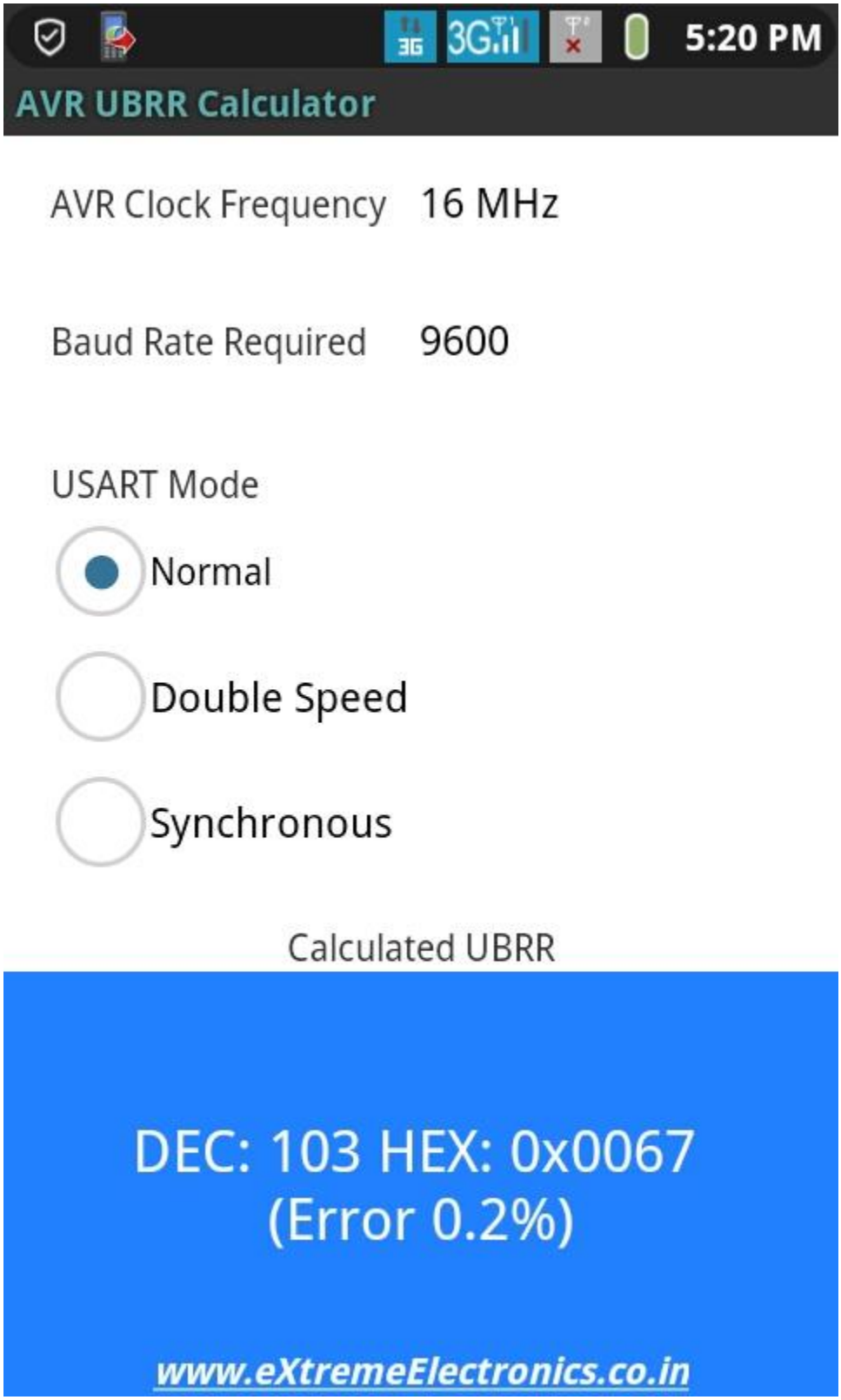


Fig. - AVR UBRR Calculator for Android

Download

- AVR UBRR Calculator for Android 2.2 or Higher (apk file)

Initialization of USART

Before using the USART it must be initialized properly according to need. Having the knowledge of RS232 communication and Internal USART of AVR you can do that easily. We will create a function that will initialize the USART for us.

```
#include <avr/io.h>
#include <inttypes.h>

void USARTInit(uint16_t ubrr_value)
{

   //Set Baud rate
   UBRR= ubrr_value;

   /*Set Frame Format


   >> Asynchronous mode
   >> No Parity
   >> 1 StopBit
   >> char size 8

   */

   UCSRC=(1<<URSEL)|(3<<UCSZ0);
```

```
    //Enable The receiver and transmitter
    UCSRB=(1<<RXEN)|(1<<TXEN);


}
```

Now we have a function that initializes the USART with a given UBRR value.

That's it for now friends. In next tutorial we will learn how to send and receive data over RS232 channel. ***Don't forget to post your opinion or doubts or any suggestion I would be very happy to see them. So don't wait post a comment now !***

Other Parts of this Tutorial

- RS232 Communication – The Basics
- RS232 Communication – The Level Conversion
- Using the USART of AVR Microcontrollers.
- Using the USART of AVR Microcontrollers : Reading and Writing Data
- Visualize ADC data on PC Screen using USART – AVR Project

Using the USART of AVR Microcontrollers : Reading and Writing Data

Posted by Avinash on December 29th, 2008 11:52 AM. Under AVR Tutorials, Code Libraries

Till now we have seen the basics of RS232 communication, the function of level converter and the internal USART of AVR micro. After understanding the USART of AVR we have also written a easy to use function to initialize the USART. That was the first step to use RS232. Now we will see how we can actually send/receive data via rs232. As this tutorial is intended for those who are never used USART we will keep the things simple so as to just concentrate on the "USART" part. Of course after you are comfortable with usart you can make it more usable my using interrupt driven mechanism rather than "polling" the usart.

So lets get started! In this section we will make two functions :-

- USARTReadChar() : To read the data (char) from the USART buffer.
- USARTWriteChar(): To write a given data (char) to the USART.

This two functions will demonstrate the use of USART in the most basic and simplest way. After that you can easily write functions that can write strings to USART.

Reading From The USART : USARTReadChar() Function.

This function will help you read data from the USART. For example if you use your PC to send data to your micro the data is automatically received by the USART of AVR and put in a buffer and bit in a register (UCSRA) is also set to indicate that data is available in buffer. Its now your duty to read this data from the register and process it, otherwise if new data comes in the previous one will be lost. So the funda is that wait until the RXC bit (bit no 7) in UCSRA is SET and then read the UDR register of the USART.

(See the full description of USART registers)

```
char USARTReadChar()
{
   //Wait untill a data is available

   while(!(UCSRA & (1<<RXC)))
   {
      //Do nothing
   }

   //Now USART has got data from host
   //and is available is buffer

   return UDR;
}
```

Writing to USART : USARTWriteChar()

This function will help you write a given character data to the USART. Actually we write to the buffer of USART and the rest is done by USART, that means it automatically sends the data over RS232 line. One thing we need to keep in mind is that before writing to USART buffer we must first check that the buffer is free or not. It its not we simply wait until it is free. If its not free it means that USART is still busy sending some other data and once it finishes it will take the new data from buffer and start sending it.

Please not that the data held in the buffer is not the current data which the USART is busy sending. USART reads data from the buffer to its shift register which it starts sending and thus the buffer is free for your data. Every time the USART gets it data from buffer and thus making it empty it notifies this to the CPU by telling "**U**SART **D**ata **R**egister Ran

Empty" (UDRE) . It does so by setting a bit (UDRE bit no 5) in UCSRA register.

So in our function we first wait until this bit is set (by the USART), once this is set we are sure that buffer is empty and we can write new data to it.

(See the full description of USART registers)

```
void USARTWriteChar(char data)
{
   //Wait until the transmitter is ready

   while(!(UCSRA & (1<<UDRE)))
   {
      //Do nothing
   }

   //Now write the data to USART buffer

   UDR=data;
}
```

Note: Actually their are two separate buffers, one for transmitter and one for receiver. But they share common address. The trick is that all "writes" goes to the Transmitter's buffer while any "read" you performs comes from the receiver's buffer.

That means if we read UDR we are reading from receivers buffer and when we are writing to UDR we are writing to transmitters buffer.

UDR=some_data; //Goes to transmitter

some_data=UDR; //Data comes from receiver

(See the full description of USART registers)

Sample program to use AVR USART

The following program makes use of the two functions we developed. This program simply waits for data to become available and then echoes it back via transmitter but with little modification. For example if you send "A" to it, it will send you back "[A]" that is input data but surrounded by square bracket. This program is enough to test the USART yet easy to understand.

```
/*

A simple program to demonstrate the use of USART
of AVR micro

*********************************************************
*************

See: www.eXtremeElectronics.co.in for more info

Author : Avinash Gupta
E-Mail: me@avinashgupta.com
Date : 29 Dec 2008

Hardware:
   ATmega8 @ 16MHz

   Suitable level converter on RX/TX line
   Connected to PC via RS232
   PC Software :  Hyper terminal @ 19200 baud
                  No Parity,1 Stop Bit,
                  Flow Control = NONE
```

```
*/

#include <avr/io.h>
#include <inttypes.h>

//This function is used to initialize the USART
//at a given UBRR value
void USARTInit(uint16_t ubrr_value)
{

   //Set Baud rate

   UBRRL = ubrr_value;
   UBRRH = (ubrr_value>>8);

   /*Set Frame Format


   >> Asynchronous mode
   >> No Parity
   >> 1 StopBit

   >> char size 8

   */

   UCSRC=(1<<URSEL)|(3<<UCSZ0);


   //Enable The receiver and transmitter

   UCSRB=(1<<RXEN)|(1<<TXEN);
```

```
}


//This function is used to read the available
data
//from USART. This function will wait untill
data is
//available.
char USARTReadChar()
{
   //Wait untill a data is available

   while(!(UCSRA & (1<<RXC)))
   {
      //Do nothing
   }

   //Now USART has got data from host
   //and is available is buffer

   return UDR;
}


//This fuction writes the given "data" to
//the USART which then transmit it via TX line
void USARTWriteChar(char data)
{
   //Wait untill the transmitter is ready

   while(!(UCSRA & (1<<UDRE)))
   {
      //Do nothing
   }
```

```
   //Now write the data to USART buffer

   UDR=data;
}

void main()
{
   //This DEMO program will demonstrate the use
of simple

   //function for using the USART for data
communication

   //Varriable Declaration
   char data;

   /*First Initialize the USART with baud rate =
19200bps

   for Baud rate = 19200bps

   UBRR value = 51

   */

   USARTInit(51);     //UBRR = 51

   //Loop forever

   while(1)
   {
      //Read data
      data=USARTReadChar();
```

```
        /* Now send the same data but but surround
it in
        square bracket. For example if user sent
'a' our
        system will echo back '[a]'.

        */

        USARTWriteChar('[');
        USARTWriteChar(data);
        USARTWriteChar(']');

    }
}
```

Download Sample Program

Running the USART Demo

You can run the above program in a ATmega8, ATmega16, ATmega32 cpu running at **16MHz** without any modification. If you are using different clock frequency you have to change the UBRR value that we are passing to USARTInit() function. See previous tutorial for calculating UBRR value. AVR running the USART demo program can be interface to PC using following three ways.

- **Connect to a Physical COM Port.**

  If you are lucky and own a really old PC then you may find a Physical COM port on your PC's back. It is a 9 pin D type male connector. In this case you have to make a **RS232 to TTL converter** and connect the MCU to COM port via it.

- **Connect to a Virtual COM Port.**

Those who are not so lucky may buy a Virtual COM port. Again in this case too you need to built a **RS232 to TTL converter** and connect the MCU to COM port via it.

Virtual COM Port can be connect to USB Port.

RS232 to TTL Converter attached.

- **Connect Via a Chips like CP2102.**

  CP2102 is single chip USB to UART Bridge by SiLabs. This chip can be used to connect your embedded applications to USB port and enable them to transfer data with PC. It is the easiest path to build PC interfaced projects, like a PC controlled robot. We have a very good CP2102 module that can be used right out of the box. We have done all PCBs and fine SMD soldering for you.

CP2102 Based Module.

CP2102 Module Can be hooked to USB Directly!

Flywires used for interconnects.

Connection is very easy.

Female sides provide easy connection to headers.

xBoard MINI v2.0 with ATmega8 MCU

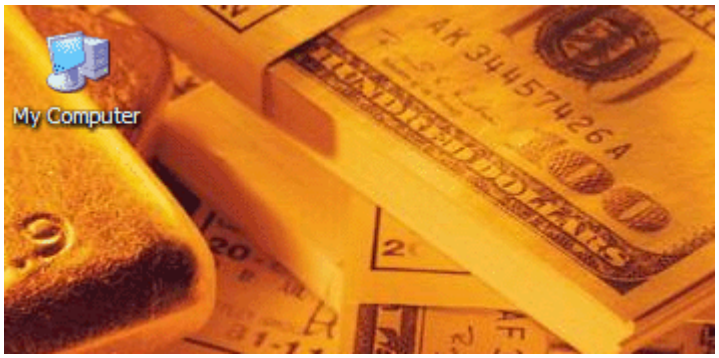
ATmega8 Connected to CP2102

Complete setup for ATmega to USB Connection

Finding the COM port number of Serial Port

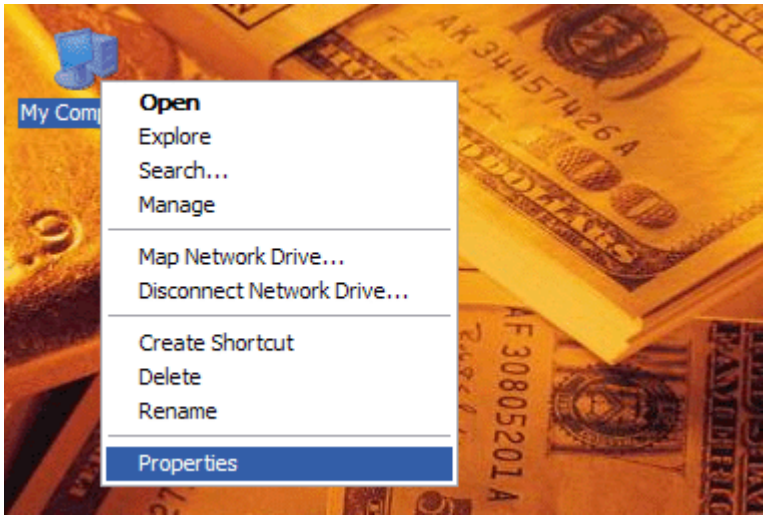
A PC can have several COM ports, each may have some peripheral connected to it like a Modem. Serial Ports on PC are numbered like COM1, COM2 ... COMn etc. You first need to figure out in which COM port you have connected the AVR. Only after you have a correct COM port number you can communicate with the AVR using tools such as Hyperterminal. The steps below shows how to get COM port number in case of Virtual COM Ports.

Right Click on ***"My Computer"*** icon in Windows Desktop.



My Computer Icon on Windows Desktop

Select ***"Properties"***

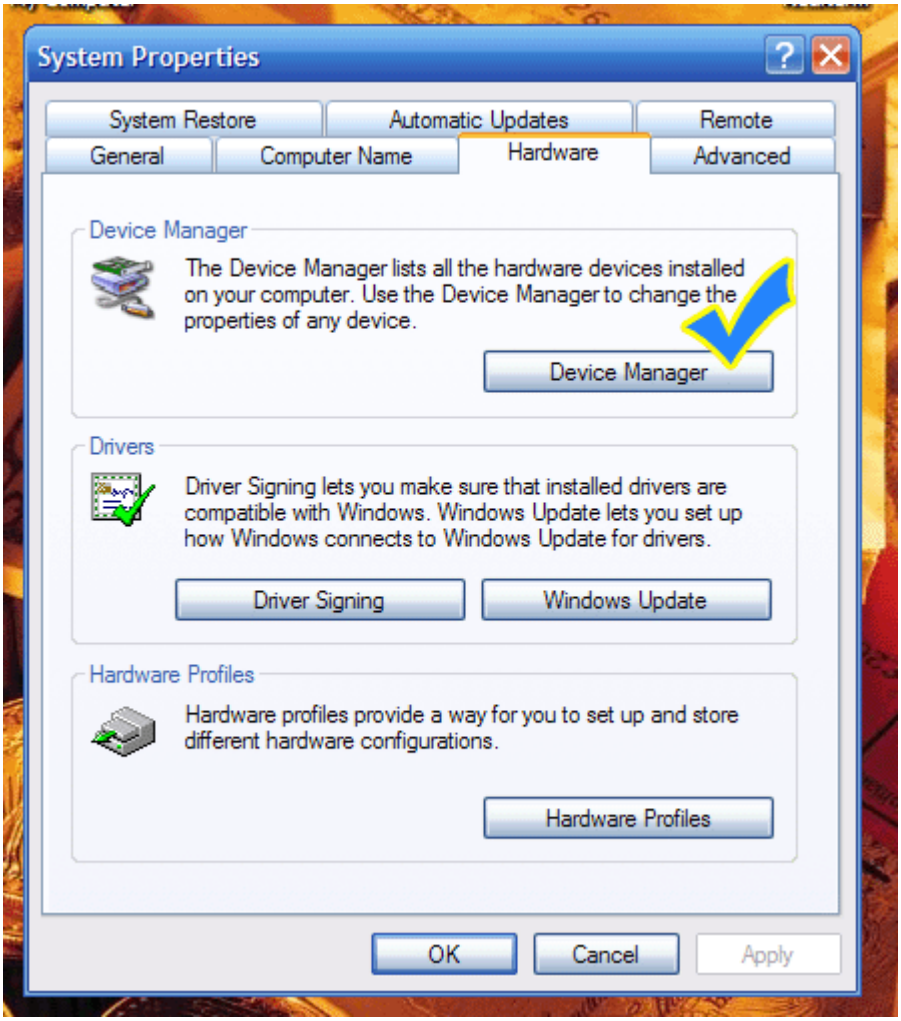


System Context Menu

The System Properties will open up. Go to the ***"Hardware"*** Tab.

System Properties.

In Hardware tab select ***"Device Manager"*** button. It will open up device manager.



Open Device Manager

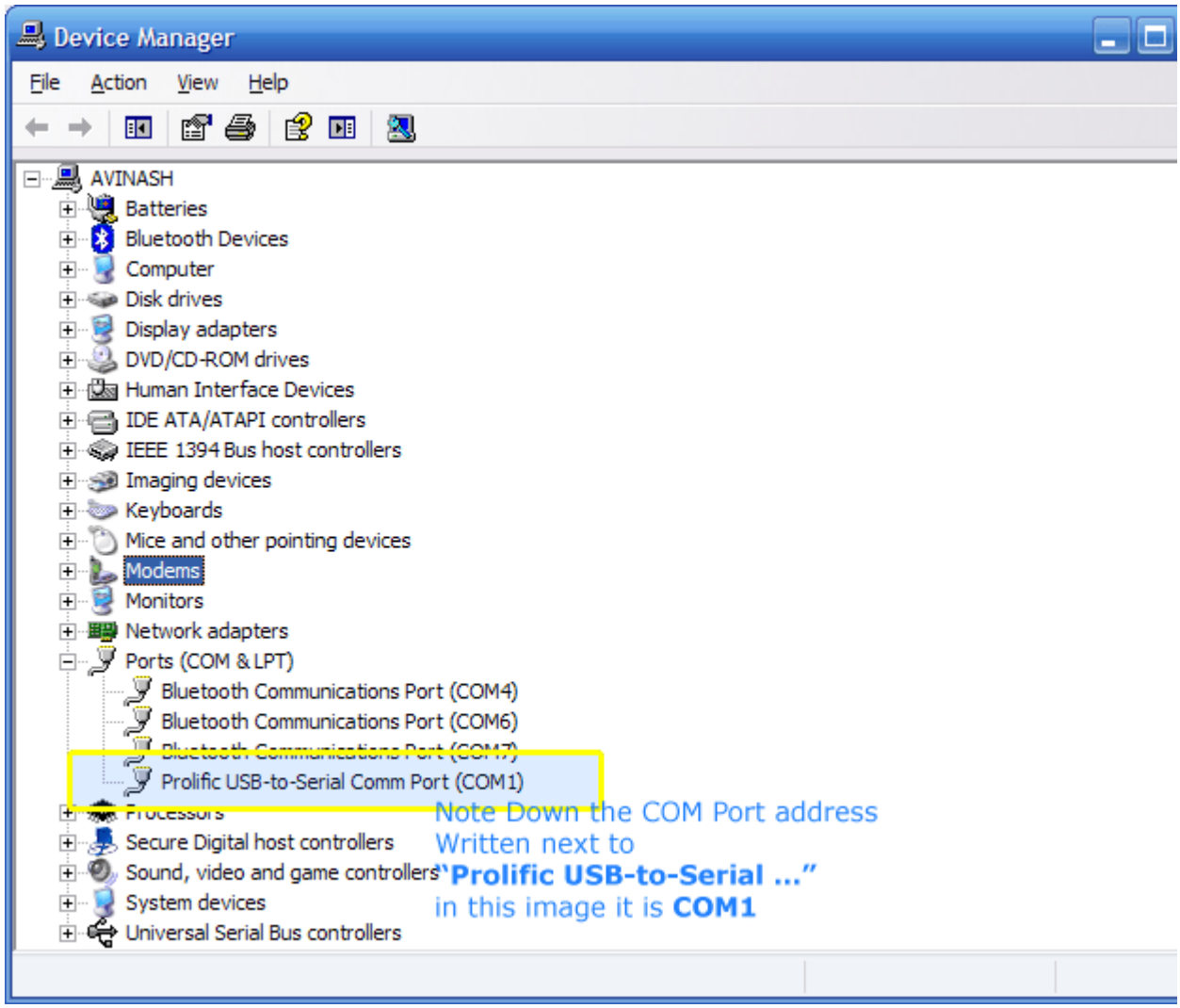
In Device Manager Find the Node ***"Ports (COM & LPT)"***

Expand the PORT node in Device Manager

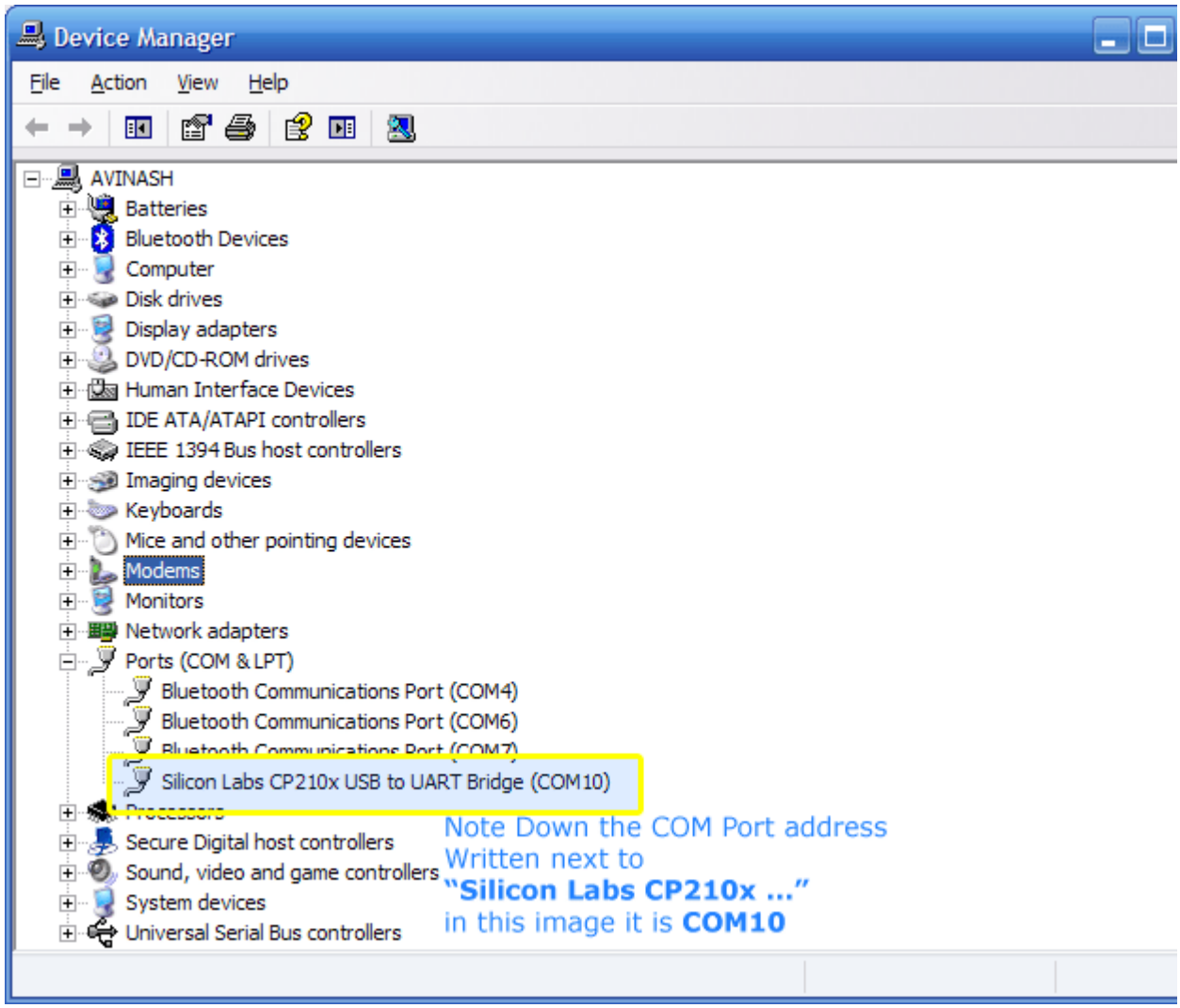
Depending on whether you are using a "USB to Serial Converter" or "CP2102 USB/USART Bridge Module" you have to find the port with following name.

- **Prolific USB-to-Serial** if you are using Bafo USB to Serial Converter.
- **Silicon Labs CP210x** if you are using CP2102 chip.

Note down the COM port number next to the port name. You need to open this Port in Hyperterminal.



COM Port Number



COM Port Number

Communication using a Terminal Program on PC.

Since this is the introductory article about serial communication, we won't be going in much detail on PC end COM port programming. For this reason we will be using a ready made software for sending and receiving serial data. I will be showing how to use two different terminal program to exchange data with embedded application.

Windows Hyperterminal

This is a default terminal program shipped with Windows OS. You can start it from

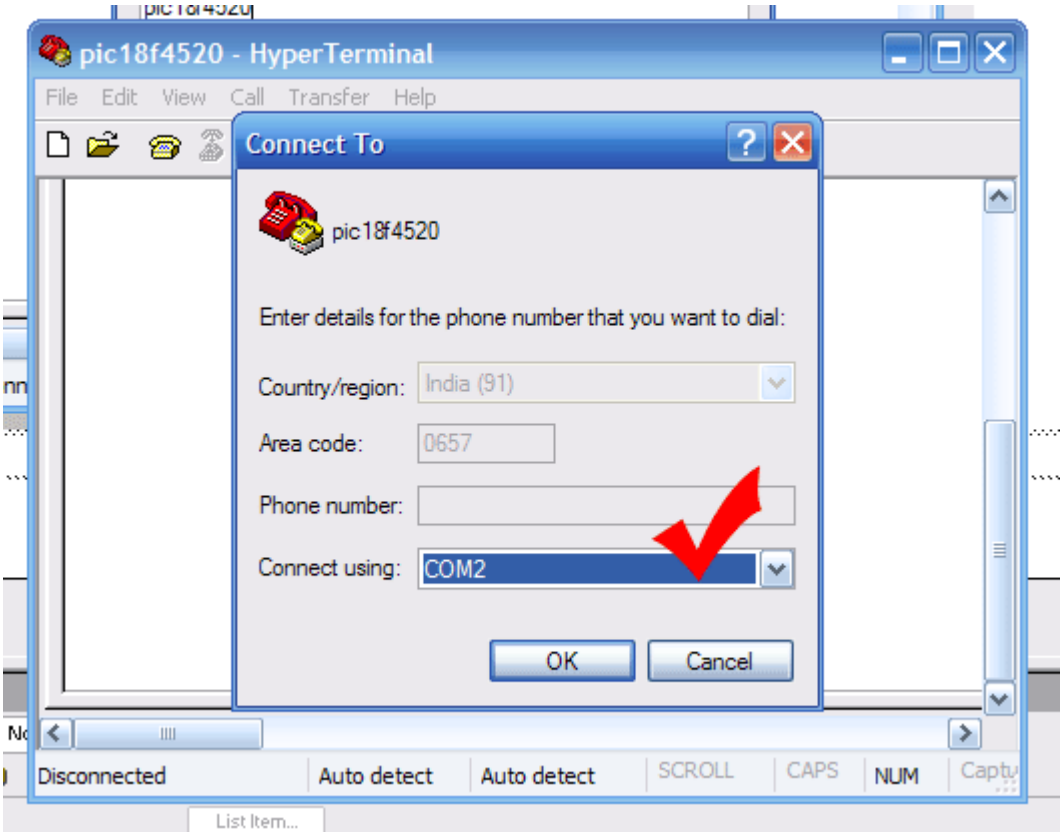
Start Menu->All Programs->Accessories->Communication->Hyperterminal.

Hyperterminal is not available in Windows Vista or Windows 7 so you have to use other terminal programs like RealTerm.

On startup it will ask for a connection name. Here we will enter **AVR**

Create New Connection

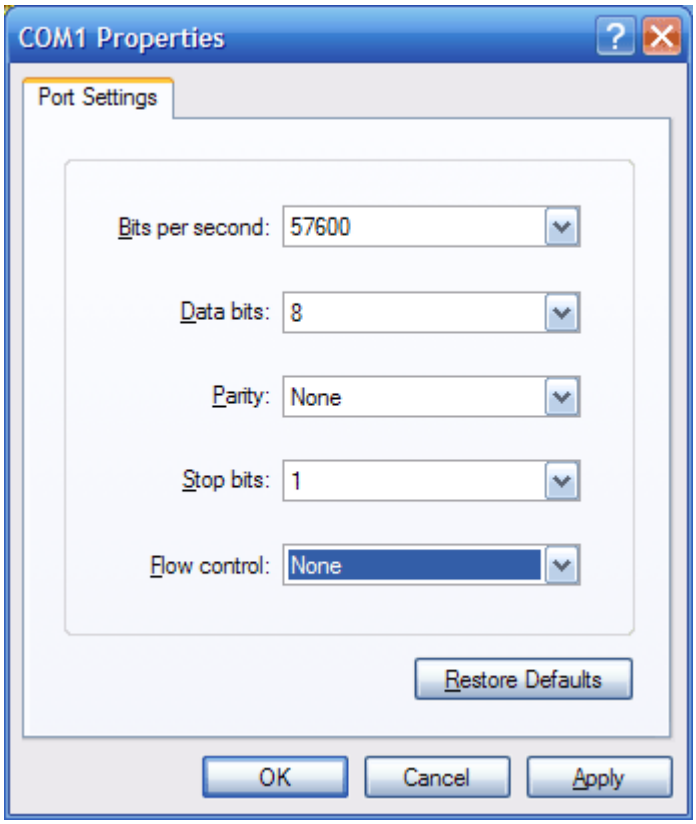
After that select a COM port you want to use. If you are using USB to serial adaptor please confirm which COM port number it is using. Other COM ports are usually connected to some device say an Internal modem etc. While some others are Bluetooth COM ports. Don't use them. If you have a physical com port then most probably it will be COM1. If you select wrong COM port during this step you won't be able to communicate with the AVR MCU and won't get expected results.



Select COM Port

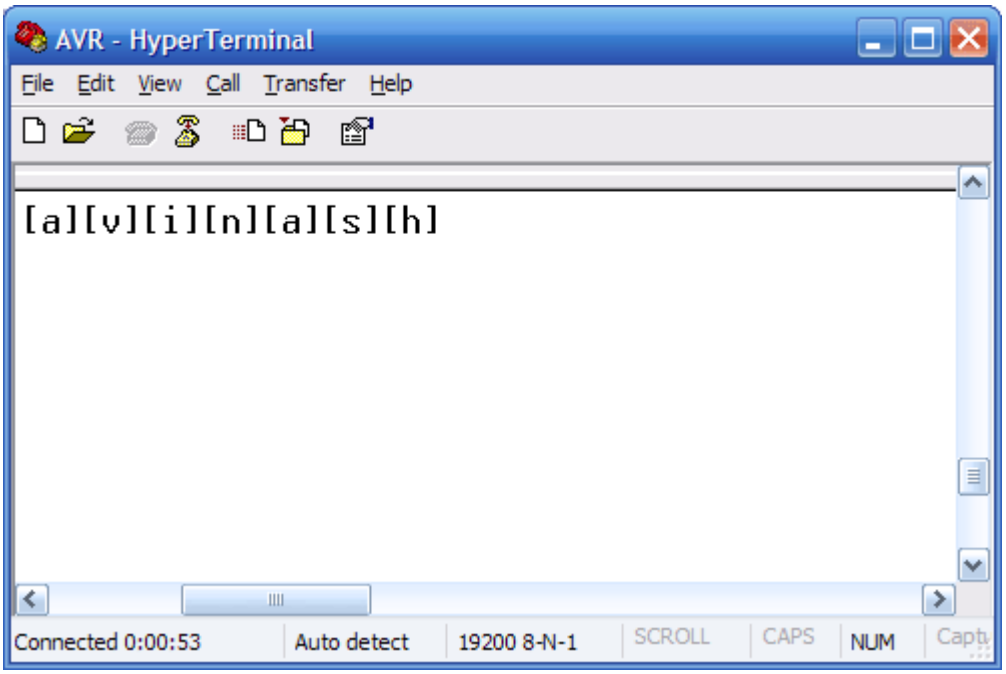
Now setup the COM port parameters as follows.

- Bits per second: 19200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None



Setting up the COM port

HyperTerminal is ready for communication now! If everything went right HyperTerminal and AVR will talk happily and AVR will send the following message as we have programmed it.



Screenshot of Hyperterminal Showing the message received from AVR

If the screen shows similar message then you have successfully created a link between PC and your AVR micro. It shows that PC can read the data sent by AVR. To test if the AVR can also read Hyperterminal, press some keys on PC keyboard. Hyperterminal will send them over COM port to the AVR mcu where AVR will process the data. In the simple test program this processing includes returning the same data but enclosed inside [and], so if you press '**k**' then AVR will return **[k]**. If you are able to see this on PC screen then you are sure that AVR is receiving the data correctly.

That's it! It fully tests the Serial Communication Routine and your hardware setup.

Setting Up Realterm and using it to communicate with AVR

If you are running Windows Vista or Windows 7 then the Hyperterminal Program may not be available. So in place of it you can use **Realterm**. It can be downloaded from here.

- http://realterm.sourceforge.net/

Start Realterm from its Desktop Icon. You will get a screen similar to this. Increase the Value of **Row** to 40 to see whole message.

Screenshot of Realterm Showing the message received from AVR

Setup Realterm as follows. Go to the **Port** Tab and set it as follows.

- Baud: 19200
- Port: Port where you have connected the AVR
- Data bits: 8
- Parity: None
- Stop bits: 1
- Hardware Flow Control: None

Screenshot of Realterm Setup

GSM Module SIM300 Interface with AVR Amega32

Posted by Avinash on July 29th, 2012 12:17 PM. Under AVR Tutorials Share on facebook Share on twitter Share on delicious Share on digg Share on stumbleupon Share on reddit Share on email More Sharing Services

A **GSM/GPRS Module** like SIM300 can be used for any embedded application that requires a long range communication, like a robot in Chennai controlled by a person sitting in New Delhi! Or simply a water pump in a rice field turned on in the morning by a farmer sitting in his house few kilometers away! You have few communication options depending on the application, they may be as follows.

- Simple SMS based communication
  - Turn on/off loads using simple SMS commands, so the controlling device is a standard handset. You can use any mobile phone to control the device.
  - A intruder alarm/fire alarm that informs about the panic situation to the house owner on his/her mobile via SMS.
- Call based communication
  - A smart intruder alarm/fire alarm that calls the police or fire station and plays a pre recorded audio message to inform about the emergency.
- Internet Based Communication (GPRS)
  - User can control the end application using any PC/Tablet/Mobile with internet connection. Example: LED Message Displays installed on highways/expressways controlled from a central control room to inform users or traffic conditions ahead.
  - A robot controlled over internet. That means the robot can be accessed from any device having internet any where in the world.
  - A portable device installed on vehicles that connects to internet using the GPRS Module SIM300 and uploads current position(using Global Position System) to a server. The server stores those location in a database with the ID of vehicle. Then a client(using a PC) can connect with the server using World Wide Web to see the route of the vehicle.

Advantage of using SIM300 Module.

The SIM300 KIT is a fully integrated module with SIM card holder, power supply etc. This module can be easily connected with low cost MCUs like AVR/PIC/8051. The basic communication is over asynchronous serial line. This is the most basic type of serial communication that's why it is very popular and hardware support is available in most MCUs. The data is transmitted bit by bit in a frame consisting a complete byte. Thus at high level it is viewed as a simple

text stream. Their are only two streams one is from MCU to SIM300 and other is from SIM300 to MCU. Commands are sent as simple text. Their are several tutorials that describes how to send and receive strings over the serial line.

- Using the USART of AVR MCU
- Using the USART of AVR MCU - Sending and Receiving data.

If you have never heard of serial communication and never did it in practice then it is highly recommend to go and understand clearly using some thing simpler (experiments given in above links).

Communication with SIM300 Module using AVR UART.

The hardware(inside the AVR MCU Chip) that is used to to serial communication is called the UART, we use this UART to communicate with the SIM300 module (the UART can also be used to communicate with other devices like RFID Readers, GPS Modules, Finger Print Scanner etc.). Since UART is such a common method of communication in embedded world that we have made a clean and easy to use library that we use in all our UART based projects.

Since a byte can arrive to MCU any time from the sender (SIM300), suppose if the MCU is busy doing something else, then what happens? To solve this, we have implemented a interrupt based buffering of incoming data. A buffer is maintained in the RAM of MCU to store all incoming character. Their is a function to inquire about the number of bytes waiting in this queue.

Following are the functions in AVR USART library

```
void USARTInit(uint16_t ubrrvalue)
```

Initializes the AVR USART Hardware. The parameter ***ubrrvalue*** is required to set desired baud rate for communication. By default SIM300 communicates at 9600 bps. For an AVR MCU running at 16MHz the ***ubrrvalue*** comes to be **103**.

```
char UReadData()
```

Reads a single character from the queue. Returns 0 if no data is available in the queue.

```
void UWriteData(char data)
```

Writes a single byte of data to the Tx Line, used by UWriteString() function.

```
uint8_t UDataAvailable()
```

Tells you the amount of data available in the FIFO queue.

```
void UWriteString(char *str)
```

Writes a complete C style null terminated string to the Tx line.

Example #1:

```
UWriteString("Hello World !");
```

Example #2:

```
char name[]="Avinash !";
    UWriteString(name);
```

```
void UReadBuffer(void *buff,uint16_t len)
```

Copies the content of the fifo buffer to the memory pointed by ***buff***, the amount of data to be copied is specified by ***len*** parameter. If UART incoming fifo buffer have fewer data than required (as per ***len*** parameter) then latter areas will contain zeros.

Example:

```
char gsm_buffer[128];
    UReadBuffer(gsm_buffer,16);
```

The above example will read 16 bytes of data (if available) from the incoming fifo buffer to the variable ***gsm_buffer***. Please note that we have allocated gsm_buffer as 128 byte array because latter we may need to read more than 16 bytes. So the same buffer can be used latter to read data up to 128 bytes.

The above function is used generally along with ***UDataAvailable()*** function.

```
while(UDataAvailable()<16)
    {
     //Do nothing
    }

char gsm_buffer[128];
    UReadBuffer(gsm_buffer,16);
```

The above code snippet waits until we have 16 bytes of data available in the buffer, then read all of them.

```
 void UFlushBuffer()
```

Removes all waiting data in the fifo buffer. Generally before sending new command to GSM Module we first clear up the data waiting in the fifo.

The above functions are used to send and receive text stream from the GSM Module SIM300.

SIM300 AT Command Set

Now you know the basics of AVR USART library and how to use it to initialize the USART, send and receive character data, its time for us to move ahead and look what commands are available with SIM300 module and how we issue commands and check the response. SIM300 supports several functions like sending text messages, making phone call etc. Each of the tasks is done using a command, and sim300 has several commands known as the ***command set***.

All SIM300 commands are prefixed with **AT+** and are terminated by a **Carriage Return** (or <CR> in short). The ASCII code of CR is 0x0D (decimal 13). Anything you write to SIM300 will be echoed back from sim300's tx line. That means if you write a command which is 7 bytes long (including the trailing CR) then immediately you will have same 7 bytes in the MCU's UART incoming buffer. If you don't get the echo back then it means something is wrong !

So the first function we develop is **SIM300Cmd(const char *cmd)** which does the following job :-

(**NOTE:** All sim300 related function implementations are kept in file ***sim300.c***, and prototypes and constants are kept in ***sim300.h***)

- Writes the command given by parameter ***cmd***.
- Appends **CR** after the command.

- Waits for the echo, if echo arrives before timeout it returns **SIM300_OK**(constant defined in ***sim300.h***). If we have waited too long and echo didn't arrive then it returns **SIM300_TIMEOUT**.

Implementation of SIM300Cmd()

```
int8_t SIM300Cmd(const char *cmd)
{
   UWriteString(cmd);    //Send Command
   UWriteData(0x0D); //CR

   uint8_t len=strlen(cmd);

   len++;   //Add 1 for trailing CR added to all
commands

   uint16_t i=0;

   //Wait for echo
   while(i<10*len)
   {
      if(UDataAvailable()<len)
      {
         i++;

         _delay_ms(10);

         continue;
      }
      else
      {
         //We got an echo
         //Now check it
         UReadBuffer(sim300_buffer,len);  //Read
serial Data
```

```
            return SIM300_OK;

        }
    }

    return SIM300_TIMEOUT;

}
```

Commands are usually followed by a response. The form of the response is like this

<CR><LF><response><CR><LF>

LF is Line Feed whose ASCII Code is 0x0A (10 in decimal)

So after sending a command we need to wait for a response, three things can happen while waiting for a response :

- No response is received after waiting for a long time, reason can be that the SIM300 is not connected properly with the MCU.
- Response is received but not as expected, reason can be faulty serial line or incorrent baud rate setting or MCU is running at some other frequency than expected.
- Correct response is received.

For example, command **Get Network Registration** is executed like this :-

Command String: "**AT+CREG?**"

Response:

<CR><LF>**+CREG: <n>,<stat>**<CR><LF>

<CR><LF>**OK**<CR><LF>

So you can see the correct response is 20 bytes. So after sending command "**AT+CREG?**" we wait until we have received 20 bytes of data or certain amount of time has elapsed. The second condition is implemented to avoid the risk of hanging up if the sim300 module malfunctions. That means we do not keep waiting forever for response we simply throw error if SIM300 is taking too long to respond (this condition is called ***timeout***)

If correct response is received we analyses variable <***stat***> to get the current network registration.

depending on current network registration status the value of ***stat*** can be

- 0 - not registered, SIM300 is not currently searching a new operator to register to
- 1 registered, home network
- 2 not registered, but SIM300 is currently searching a new operator to register to
- 3 registration denied
- 4 unknown
- 5 registered, roaming

Implementation of SIM300GetNetStat() Function

```
int8_t SIM300GetNetStat()
{
    //Send Command
    SIM300Cmd("AT+CREG?");

    //Now wait for response
    uint16_t i=0;

    //correct response is 20 byte long
    //So wait until we have got 20 bytes
```

```
    //in buffer.
    while(i<10)
    {
        if(UDataAvailable()<20)
        {
            i++;

            _delay_ms(10);

            continue;
        }
        else
        {
            //We got a response that is 20 bytes
long
            //Now check it
            UReadBuffer(sim300_buffer,20);   //Read
serial Data

            if(sim300_buffer[11]=='1')
                return SIM300_NW_REGISTERED_HOME;
            else if(sim300_buffer[11]=='2')
                return SIM300_NW_SEARCHING;
            else if(sim300_buffer[11]=='5')
                return SIM300_NW_REGISTED_ROAMING;
            else
                return SIM300_NW_ERROR;
        }
    }

    //We waited so long but got no response
    //So tell caller that we timed out

    return SIM300_TIMEOUT;
```

```
}
```

Similarly we have implemented the following functions :-

- int8_t SIM300IsSIMInserted()

In another type of response we don't exactly know the size of response like we knew for the above command. Example is the Get Service Provider Name command where the provider name's length cannot be known in advance. It can be **Airtel** or **Reliance** or **TATA Docomo**. To handle that situation we make use of the fact that all response are followed by a **CR LF** pair. So we simple buffer in all characters until we encounter a **CR**, that indicates end of response.

To simplify handling of such commands we have made a function called **SIM300WaitForResponse(uint16_t timeout)**

This function waits for a response from SIM300 module (end of response is indicated by a **CR**), it returns the size if response received, while the actual response is copied to the global variable ***sim300_buffer[]***.

If no response is received before timeout it returns 0. Timeout in millisecond can be specified by the parameter ***timeout***. It does not count the trailing LF or the last <CR><LF>OK<CR><LF>, they remain in the UART fifo queue. So before returning we call ***UFlushBuffer()*** to remove those from the queue.

Implementation of function SIM300WaitForResponse(uint16_t timeout)

```
int8_t SIM300WaitForResponse(uint16_t timeout)
{
   uint8_t i=0;
```

```
    uint16_t n=0;

    while(1)
    {
       while (UDataAvailable()==0 &&
n<timeout){n++; _delay_ms(1);}

       if(n==timeout)
          return 0;
       else
       {
          sim300_buffer[i]=UReadData();

          if(sim300_buffer[i]==0x0D && i!=0)
             return i+1;
          else
             i++;
       }
    }
}
```

Implementation of function SIM300GetProviderName(char *name)

The function does the following :-

1. Flush the USART buffer to get rid of any leftover response from the last command or errors.
2. It send the command **"AT+CSPN?"** using **SIM300Cmd("AT+CSPN?");** function call.
3. Then it waits for a response using the function **SIM300WaitForResponse**()
4. If we receive a response of non zero length we parse it to extract string describing the service provider name.
5. If **SIM300WaitForResponse**() returns zero that means no valid response is received within specified time out period. In this case we also return SIM300_TIMEOUT.

In the same way we have implemented the following functions :-

- uint8_t SIM300GetProviderName(char *name)
- int8_t SIM300GetIMEI(char *emei)
- int8_t SIM300GetManufacturer(char *man_id)
- int8_t SIM300GetModel(char *model)

```
uint8_t SIM300GetProviderName(char *name)
{
    UFlushBuffer();

    //Send Command
    SIM300Cmd("AT+CSPN?");

    uint8_t len=SIM300WaitForResponse(1000);

    if(len==0)
        return SIM300_TIMEOUT;

    char *start,*end;
    start=strchr(sim300_buffer,'"');
    start++;
    end=strchr(start,'"');

    *end='\0';

    strcpy(name,start);

    return strlen(name);
}
```

SIM300 and ATmega32 Hardware Setup

To run the basic demo showing communication with SIM300 using AVR ATmega32 we need the following circuit :-

- ATmega32 Core circuit, including the reset register, ISP header, 16MHz crystal oscillator.
- Power Supply circuit to supply 5v to the ATmega32 and the **LCD Module**.
- A 16x2 Alphanumeric LCD Module to show programs output.
- **SIM300 Module**.

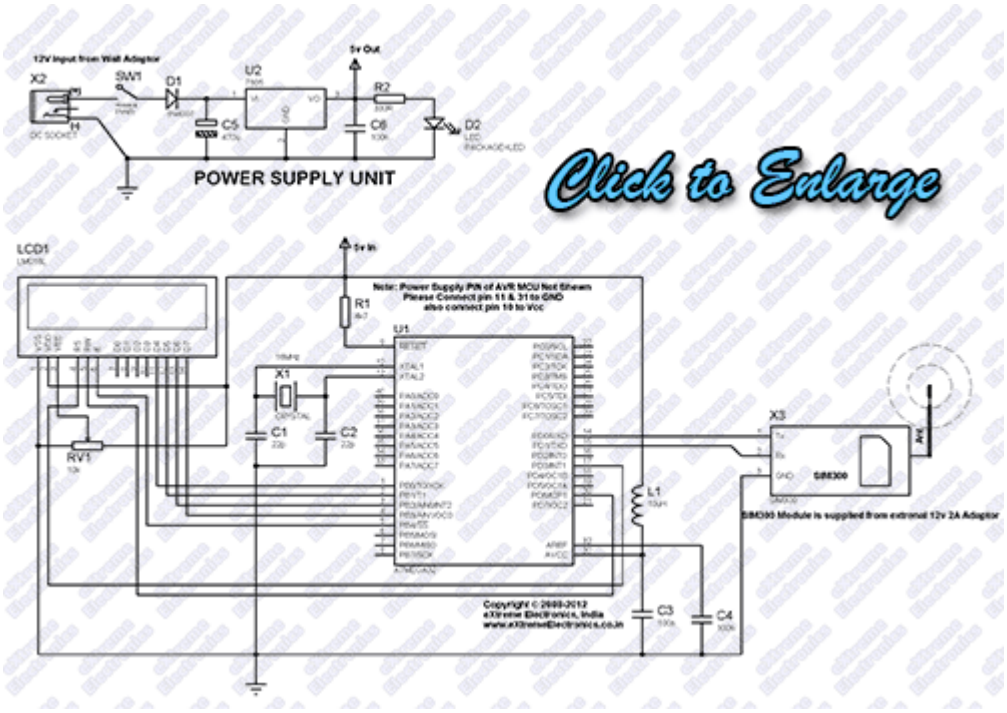


Fig. SIM300 and ATmega32 Schematic

We have made the prototype using xBoard development board because it has ATmega32 core circuit, 5v power supply circuit and the LCD module.

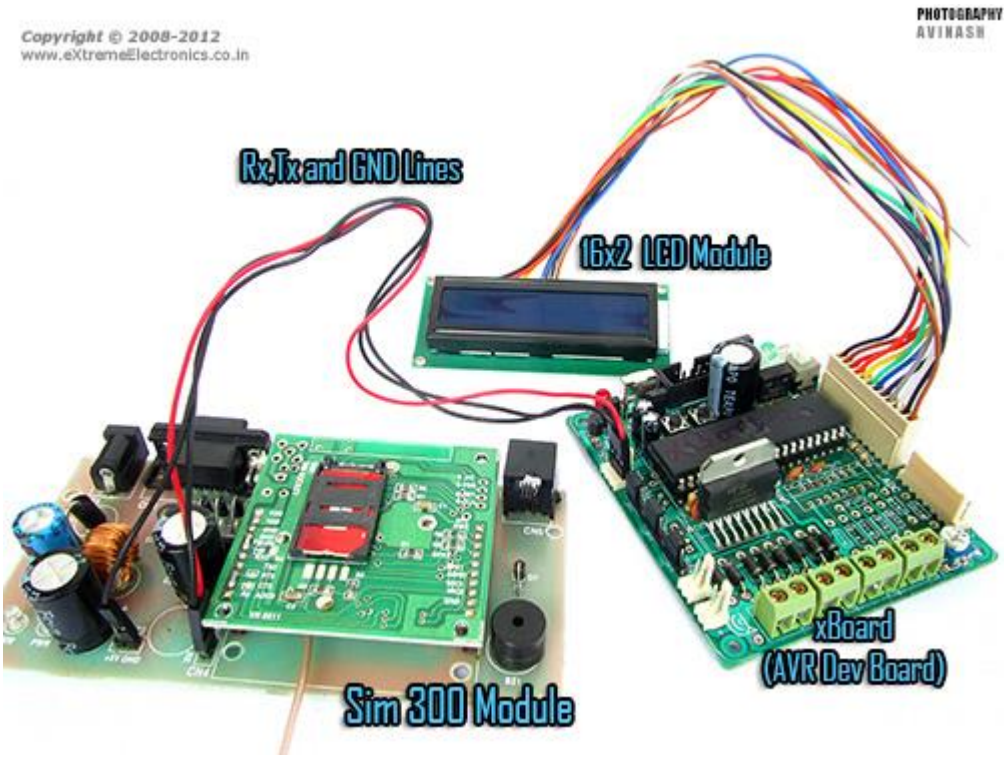


Fig. SIM300 and ATmega32 Connection

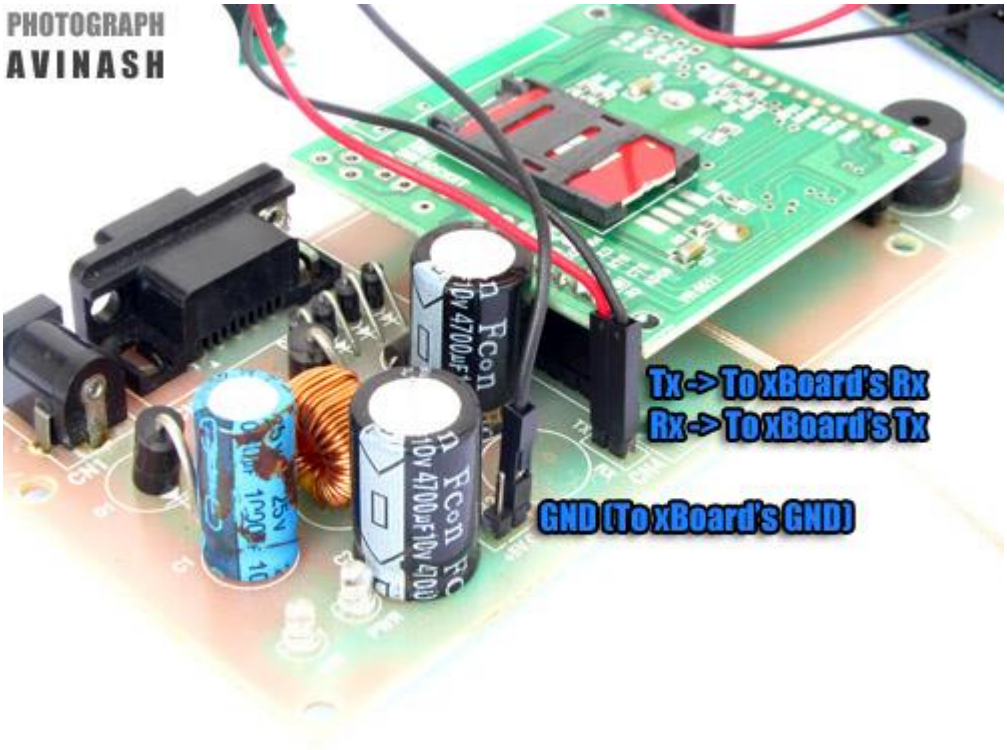


Fig. SIM300's PINs

Fig. xBoard's USART PINs

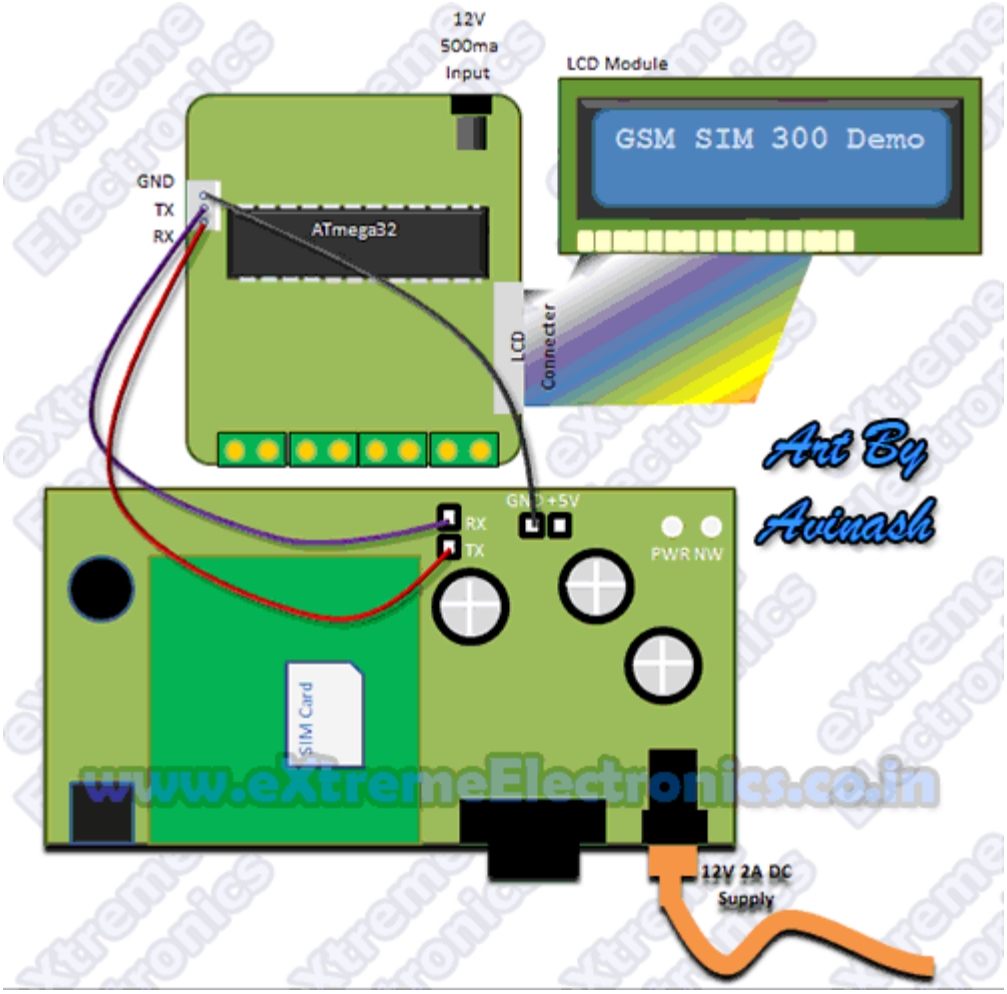


Fig. GSM Module connected with ATmega32

NOTE

The board shown below is the new version of **SIM300 Modem**, it can also be used to make this project. New version is much smaller and low cost.

Fig. SIM300 Module New Version

Demo Source Code for AVR and SIM300 Interface

The demo source code is written in C language and compiled using free avr-gcc compiler using the latest Atmel Studio 6 IDE. The project is split into the following modules.

- LCD Library
  - Files ***lcd.c, lcd.h***, ***myutils.h, custom_char.h***
  - Job is to control standard 16x2 LCD Module.
  - More information on LCD library.
- USART Library
  - Files ***usart.c,usart.h***

    - Job is to control the USART hardware of AVR MCU. Includes functions to initialize the USART, Send/Receive chars, Send/Receive Strings.
- SIM300 Library
    - Files ***sim300.c, sim300.h***

To build the project you need to be have working knowledge of the Atmel Studio 6 IDE. Please refer to the following tutorial.

- Working with latest Atmel Studio 6 IDE.

Steps to configure the AS6 Project

- Create a new AS6 Project with name ***"Sim300Demo"***.
- Using **solution explorer** create a folder named ***"lib"*** in current folder.
- Inside the ***"lib"*** folder create the following sub folders ***"lcd", "usart" and "sim300"***.
- copy followings files to the ***lcd*** folder (using Windows File Manager) ***lcd.c, lcd.h***, ***myutils.h, custom_char.h***
- copy followings files to the ***usart*** folder (using Windows File Manager) ***usart.c,usart.h***
- copy followings files to the ***sim300*** folder (using Windows File Manager ***sim300.c, sim300.h***
- Add the files ***lcd.c, lcd.h***, ***myutils.h, custom_char.h*** to the project using solution explorer.
- Add the files***usart.c,usart.h*** to the project using solution explorer.
- Add the files ***sim300.c, sim300.h*** to the project using solution explorer.
- Define a symbol F_CPU=16000000 using AS6's
- in the main application file Sim300Demo.c copy paste the following demo program.
- Build the project to get the executable hex file.
- Burn this hex file to the xBoard using USB AVR Programmer.
- If you are using a new ATMega32 MCU from the market set the LOW FUSE as **0xFF** and HIGH FUSE are **0xC9**.

Fig.: Setting the Fuse bytes.

```
/*
 * Sim300Demo.c
 *
 * Created: 10-07-2012 PM 12:23:08
 *  Author: Avinash
 */


#include <avr/io.h>
#include <util/delay.h>

#include "lib/lcd/lcd.h"
#include "lib/sim300/sim300.h"


void Halt();
int main(void)
```

```
{
   //Initialize LCD Module
   LCDInit(LS_NONE);

   //Intro Message
   LCDWriteString("SIM300 Demo !");
   LCDWriteStringXY(0,1,"By Avinash Gupta");

   _delay_ms(1000);

   LCDClear();


   //Initialize SIM300 module
   LCDWriteString("Initializing ...");
   int8_t r= SIM300Init();

   _delay_ms(1000);

   //Check the status of initialization
   switch(r)
   {
      case SIM300_OK:
         LCDWriteStringXY(0,1,"OK !");
         break;
      case SIM300_TIMEOUT:
         LCDWriteStringXY(0,1,"No response");
         Halt();
      case SIM300_INVALID_RESPONSE:
         LCDWriteStringXY(0,1,"Inv response");
         Halt();
      case SIM300_FAIL:
         LCDWriteStringXY(0,1,"Fail");
         Halt();
      default:
```

```
        LCDWriteStringXY(0,1,"Unknown Error");
        Halt();
}

_delay_ms(1000);

//IMEI No display
LCDClear();

char imei[16];

r=SIM300GetIMEI(imei);

if(r==SIM300_TIMEOUT)
{
   LCDWriteString("Comm Error !");
   Halt();
}

LCDWriteString("Device IMEI:");
LCDWriteStringXY(0,1,imei);

_delay_ms(1000);

//Manufacturer ID
LCDClear();

char man_id[48];

r=SIM300GetManufacturer(man_id);

if(r==SIM300_TIMEOUT)
{
   LCDWriteString("Comm Error !");
   Halt();
```

```
}

LCDWriteString("Manufacturer:");
LCDWriteStringXY(0,1,man_id);

_delay_ms(1000);

//Manufacturer ID
LCDClear();

char model[48];

r=SIM300GetModel(model);

if(r==SIM300_TIMEOUT)
{
   LCDWriteString("Comm Error !");
   Halt();
}

LCDWriteString("Model:");
LCDWriteStringXY(0,1,model);

_delay_ms(1000);



//Check Sim Card Presence
LCDClear();
LCDWriteString("Checking SIMCard");

_delay_ms(1000);

r=SIM300IsSIMInserted();
```

```
if (r==SIM300_SIM_NOT_PRESENT)
{
   //Sim card is NOT present
   LCDWriteStringXY(0,1,"No SIM Card !");

   Halt();
}
else if(r==SIM300_TIMEOUT)
{
   //Communication Error
   LCDWriteStringXY(0,1,"Comm Error !");

   Halt();
}
else if(r==SIM300_SIM_PRESENT)
{
   //Sim card present
   LCDWriteStringXY(0,1,"SIM Card Present");

   _delay_ms(1000);
}

//Network search
LCDClear();
LCDWriteStringXY(0,0,"SearchingNetwork");

uint8_t     nw_found=0;
uint16_t tries=0;
uint8_t     x=0;

while(!nw_found)
{
   r=SIM300GetNetStat();

   if(r==SIM300_NW_SEARCHING)
```

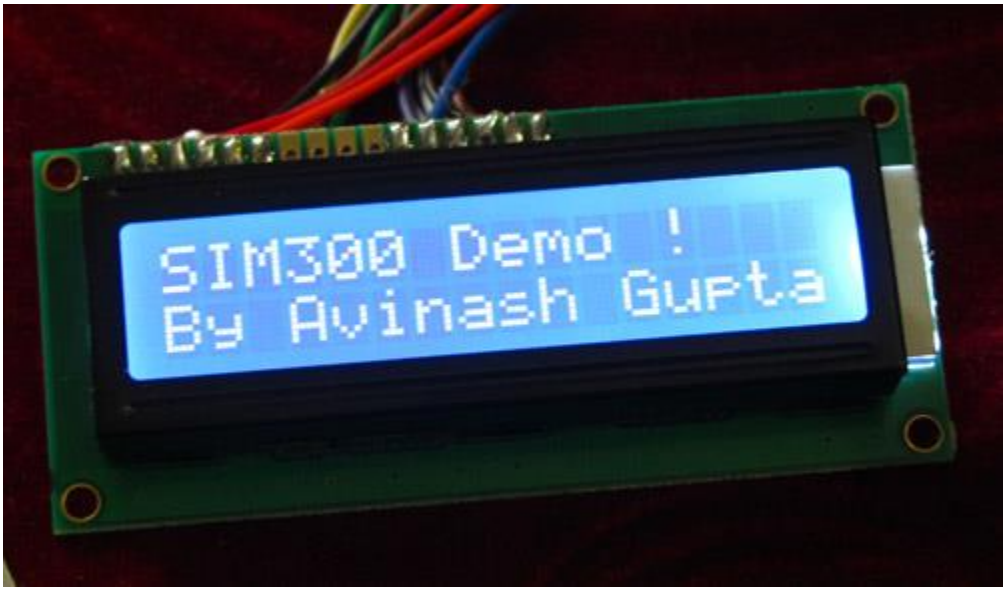
```
      {

LCDWriteStringXY(0,1,"%0%0%0%0%0%0%0%0%0%0%0%0%0
%0%0%0");
         LCDWriteStringXY(x,1,"%1");
         LCDGotoXY(17,1);

         x++;

         if(x==16) x=0;

         _delay_ms(50);

         tries++;

         if(tries==600)
            break;
      }
      else
         break;

   }
   LCDClear();

   if(r==SIM300_NW_REGISTERED_HOME)
   {
      LCDWriteString("Network Found");
   }
   else
   {
      LCDWriteString("Cant Connt to NW!");
      Halt();
   }

   _delay_ms(1000);
```

```
    LCDClear();

    //Show Provider Name
    char pname[32];
    r=SIM300GetProviderName(pname);

    if(r==0)
    {
       LCDWriteString("Comm Error !");
       Halt();
    }

    LCDWriteString(pname);

    _delay_ms(1000);

    Halt();
}

void Halt()
{
    while(1);
}
```

What the Demo Program does?

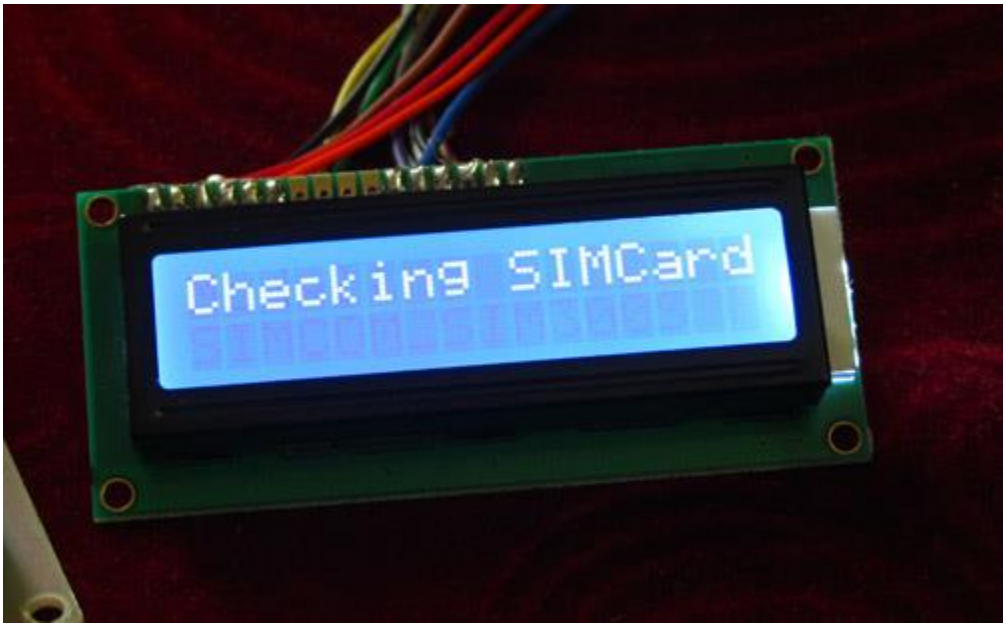
- Initializes the LCD Module and SIM300 module.
- Check if the SIM300 module is present on the USART and responding properly.
- Displays the IMEI No of the SIM300 Module.
- Displays Manufacturer ID
- Then checks for the SIM card presence.
- Searches for a GSM Network and establishes a connection. The sim card need to be valid in order to connect to the GSM network.

- Shows the Network Provider's name like **Airtel** or **Vodafone**.

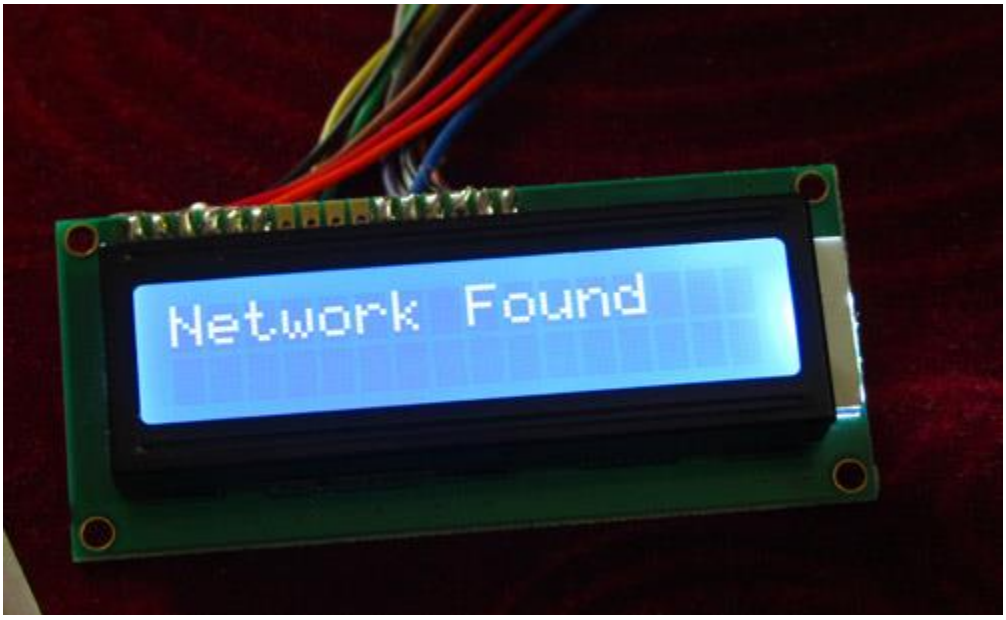



Device IMEI:


Manufacturer:
SIMCOM_Ltd


Checking SIMCard


Checking SIMCard
SIM Card Present



Troubleshooting

- NO Display on LCD
  - Make sure AVR Studio Project is set up for clock frequency of **16MHz** (16000000Hz)
  - Adjust the Contrast Adj Pot.
  - Press reset few times.
  - Power On/Off few times.
  - Connect the LCD only as shown on schematic above.

- On SIM300 Initialization phase if you get error "***No Response***"
  - Check Rx,Tx and GND line connection between SIM300 and the xBoard.
  - Make sure crystal frequency is 16MHz only.
  - Fuse bits are set as described above.
- Compiler Errors
  1. Many people these days has jumped to embedded programming without a solid concept of computer science and programming. They don't know the basics of compiler and lack experience. To learn basic of compilers and their working PC/MAC/Linux(I mean a desktop or laptop) are great platform. But embedded system is **not good** for learning about compilers and programming basics. **It is for those who already have these skills and just want to apply it.**
  2. Make sure all files belonging to the LCD Library are "added" to the "Project".
  3. avr-gcc is installed. (The Windows Binary Distribution is called **WinAVR**)
  4. The AVR Studio project Type is AVR GCC.
  5. Basics of Installing and using AVR Studio with avr-gcc is described in this **tutorial**
- General Tips for newbies
  - Use ready made **development boards** and **programmers**.
  - Try to follow the AVR Tutorial Series from the very beginning. (Remember the list spans four pages, page 1 is most recent addition thus most advance)

Downloads for GSM Module Demo.

- Atmel Studio 5 Project for SIM300 Demo.
- Atmel Studio 6 Project for SIM300 Demo.
- HEX File Ready to Burn on ATmega32 running at 16MHz.
- SIM300 Complete Command Set.

Sending an Receiving SMS using SIM300 GSM Module

Posted by Avinash on August 2nd, 2012 01:09 PM. Under AVR Tutorials, Code Snippets

Hi friends in this part we will have a look at the functions related to text messages. **By the end of this article you will have a clear idea of how to wait for a text message, read the message, send a new text message and deleted a received message.** We have already discussed the basics of SIM300 GSM Module interface with AVR MCU in our previous tutorial you can refer that article to know about the schematic and basic communication code.

- GSM Module SIM300 Interface with AVR

After reading the above article you will know how we have connected the SIM300, AVR ATmega32 and LCD Module to make a basic test rig. Also covered in the article is the detail about the communication method between SIM300 and AVR which is called asynchronous serial communication, done using AVR's USART peripheral. The article shows you how to use the AVR USART library to send and receive data to/from the GSM Module. We have also discussed the command response method used for the interfacing with the module. Working code is provided that shows you how to implement the command response based communication over USART with the SIM300 GSM Module. Those techniques are used through out this article so we recommend you to go through the article and do the demo project given there.

Waiting for a text message (SMS)

When a text message (SMS) arrives on SIM300 GSM Module it sends a **unsolicited response** <CR><LF>+CMTI: <mem>,<n><CR><LF>

I have already told in previous article that <CR> refers to a control character whose ASCII is 0D (Hex) and <LF> with ASCII code of 0A(Hex). The new thing you will learn about is **unsolicited response**. Earlier I told commands are followed by response. But the response discussed above is not followed by any command, it can come at any moment. So it is called an **unsolicited response**.

value of ***mem*** is the storage location where the sms was stored. Usually its value is ***SM***, which stands for SIM memory.

the value of ***n*** is the sms slot on which the incoming message was stored. Depending on the size of storage in your SIM card their may me 20 or so slots in your card. When a message is received it is stored in the lowest numbered empty slot. For example you first received 4 messages then deleted the 1st message, then 5th message will get stored in slot 1.

Code Example showing how Wait for a message.

```
int8_t   SIM300WaitForMsg(uint8_t *id)
{
   //Wait for a unsolicited response for 250ms
   uint8_t len=SIM300WaitForResponse(250);

   if(len==0)
      return SIM300_TIMEOUT;

   sim300_buffer[len-1]='\0';

   //Check if the response is +CMTI (Incoming
msg indicator)

if(strncasecmp(sim300_buffer+2,"+CMTI:",6)==0)
   {
      char str_id[4];
```

```
        char *start;

        start=strchr(sim300_buffer,',');
        start++;

        strcpy(str_id,start);

        *id=atoi(str_id);

        return SIM300_OK;
    }
    else
        return SIM300_FAIL;
}
```

Code Walkthrough

1. We wait for a response from SIM300 with a time-out of 250 millisecond. That means if nothing comes for 250 millisecond we give up!
2. If we get a response, SIM300WaitForResponse() returns its length till trailing <CR>. So suppose we got <CR><LF>+CMTI: SM,1<CR><LF> then ***len*** will be 14.
3. Next line `sim300_buffer[len-1]='\0';` puts an NULL character at position len - 1 that is 13(points to last <CR>). So the response actually becomes <CR><LF>+CMTI: SM,1
4. Now we want to check if the first 6 characters are ***+CMTI:*** or not, we check first 6 characters only because the <n> following +CMTI is variable. Remember it is the slot number where the message is stored! Also while comparing we want to ignore the case, that means +CMTI or +cMtI are same. This type of string comparison is easily done using the standard C library function ***strncasecmp()***. Well if you have ever attended a C training session in school you must have remembered ***strcmp()*** , so you can look ***strncasecmp()***.So you can see only an **n** and **case** is added in the

name of the function. **n** in the name indicate you don't have to compare the whole string, you can check the first **n** characters. While the **case** in the name indicate a case in-sensitive match. Also you notice we don't pass ***sim300_buffer*** (which holds the response) directly to the comparison function, but we pass ***sim300_buffer + 2***, this removes the leading <CR><LF> in the response string.

5. If the response matches the next step is to extract the value of <n>, i.e. the message slot id. As you can see the slot id is after the first **comma(,)** in the response. So we search for the position of first comma in the response. This is done using ***strchr()*** standard library function. Now start points to a string which is ",1".
6. The we do start++, this makes start points to a string which is "1", but remember it is still a string. So we use standard library function ***atoi()*** which converts a string to a integer. Which we store in ****id***. Remember the parameter ***id*** is pass by reference type, if you don't know what does that means go and revise your C book!
7. Finally we return ***SIM300_OK*** which is a constant defined in ***sim300.h*** indicating a success to caller.

Reading the Content of Text Message

The command that is used to read a text message from any slot is **AT+CMGR=<n>** where <n> is an integer value indicating the sms slot to read. As I have already discussed that their are several slots to hold incoming messages.

The response is like this

+CMGR: "STATUS","OA",,"SCTS"<CR><LF>Message Body<CR><LF><CR><LF>OK<CR><LF>

where STATUS indicate the status of message it could be **REC UNREAD** or **REC READ**

OA is the Originating Address that means the mobile number of the sender.

SCTS is the Service Center Time Stamp.

This is a simple example so we discard the first line data i.e. STATUS, OA and SCTS. We only read the Message Body.

One of the most interesting thing to note is that three things can happen while attempting to read a message ! They are listed below.

1. Successful read in that case the response is like that discussed above.
2. Empty slot ! That means an attempt has been make to read a slot that does not have any message stored in it. In this case the response is <CR><LF>OK<CR><LF>
3. SIM card not ready! In this case **+CMS ERROR: 517** is returned.

Our function handles all the three situations.

```
int8_t   SIM300ReadMsg(uint8_t i, char *msg)
{
   //Clear pending data in queue
   UFlushBuffer();

   //String for storing the command to be sent
   char cmd[16];

   //Build command string
   sprintf(cmd,"AT+CMGR=%d",i);

   //Send Command
   SIM300Cmd(cmd);

   uint8_t len=SIM300WaitForResponse(1000);
```

```
    if(len==0)
        return SIM300_TIMEOUT;

    sim300_buffer[len-1]='\0';

    //Check of SIM NOT Ready error
    if(strcasecmp(sim300_buffer+2,"+CMS ERROR:
517")==0)
    {
        //SIM NOT Ready
        return SIM300_SIM_NOT_READY;
    }

    //MSG Slot Empty
    if(strcasecmp(sim300_buffer+2,"OK")==0)
    {
        return SIM300_MSG_EMPTY;
    }

    //Now read the actual msg text
    len=SIM300WaitForResponse(1000);

    if(len==0)
        return SIM300_TIMEOUT;

    sim300_buffer[len-1]='\0';
    strcpy(msg,sim300_buffer+1);//+1 for removing
trailing LF of prev line

    return SIM300_OK;
}
```

Code Walkthrough

1. Clear pending data in the buffer.

2. A command string is built using the standard library function sprintf().
    1. `sprintf(cmd,"AT+CMGR=%d",i);`
    2. You must be knowing that the above code gives a string in which placeholder %d is replaced by the value of ***i***.
3. Command is sent to the SIM300 module.
4. Then we wait for the response.
5. Response is analyzed.
6. Finally messages is read and copied to the memory address pointed by ****msg*** using standard library function ***strcpy()***.

Sending a New Text Message

We will develop a function to easily send message to any mobile number. This function has the argument

- num (IN) - Phone number where to send the message ex "+919939XXXXXX"
- msg (IN) - Message Body ex "This a message body"
- msg_ref (OUT) - After successful send, the function stores a unique message reference in this variable.

The function returns an integer value indicating the result of operation which may be

- SIM300_TIMEOUT - When their is some problem in communication line or the GSM module is not responding or switched off.
- SIM300_FAIL - Message Sending Failed. A possible reason may be not enough balance in your prepaid account !
- SIM300_OK - Message Send Success!

```
int8_t   SIM300SendMsg(const char *num, const
char *msg,uint8_t *msg_ref)
{
```

```
    UFlushBuffer();

    char cmd[25];

    sprintf(cmd,"AT+CMGS= %s",num);

    cmd[8]=0x22; //"

    uint8_t n=strlen(cmd);

    cmd[n]=0x22; //"
    cmd[n+1]='\0';

    //Send Command
    SIM300Cmd(cmd);

    _delay_ms(100);

    UWriteString(msg);

    UWriteData(0x1A);

      //Wait for echo
    while(   UDataAvailable()<(strlen(msg)+5)
);

    //Remove Echo
    UReadBuffer(sim300_buffer,strlen(msg)+5);

    uint8_t len=SIM300WaitForResponse(6000);

    if(len==0)
       return SIM300_TIMEOUT;

    sim300_buffer[len-1]='\0';
```

```
    if(strncasecmp(sim300_buffer+2,"CMGS:",5)==0)
    {

        *msg_ref=atoi(sim300_buffer+8);

        UFlushBuffer();

        return SIM300_OK;
    }
    else
    {
        UFlushBuffer();
        return SIM300_FAIL;
    }
}
```

Code Walkthrough

1. The beginning of the function implementation is similar to those done above, first we flush the buffer and build command string. Command string must be like this :-
    - AT+CMGS=<DA>, Where DA is the destination address i.e. the mobile number like AT+CMGS="+919939XXXXXX"
    - This function `sprintf(cmd,"AT+CMGS= %s",num);` gives a string like this AT+CMGS= +919939XXXXXX
    - `cmd[8]=0x22; //"` this like replaces the space just before +91 with " so we have a string like this AT+CMGS="+919939XXXXXX
    - `cmd[n]=0x22; //"` this adds a " in the end also so the string becomes AT+CMGS="+919939XXXXXX", but caution it removes the '\0' the null character that must be present to mark the end of string

```
        in C. So the following statement adds a
        NULL character at the end.
    ◦ cmd[n+1]='\0';
```

2. Then we send the command string prepared above.\
3. We now write the message body to SIM300 module using the function `UWriteString(msg);`
4. `UWriteData(0x1A);` Is used to send the control character called EOF (End of File) to mark the end of message.
5. Since SIM300 echoes back everything we write to it, so we remove the echo by reading the data from the buffer.
6. Finally we wait for the response, read the response and compare it to find out whether we succeeded or not.

Deleting a Text Message

This function takes an integer input which should be slot number of the message you wish to delete. Function may return the following values.

- SIM300_TIMEOUT - When their is some problem in communication line or the GSM module is not responding or switched off.
- SIM300_FAIL - Message Deleting Failed. A possible reason may be an incorrect slot number id.
- SIM300_OK - Message Delete Success!

AT command of deleting a message is AT+CMGD=<n> where n is an slot of number of message you wish to delete. If delete is successful it returns OK. The function implementation is very simple compared to above functions. The steps are similar, that involves building a command string, sending command, waiting for response and verifying response.

```
int8_t   SIM300DeleteMsg(uint8_t i)
{
   UFlushBuffer();
```

```
    //String for storing the command to be sent
    char cmd[16];

    //Build command string
    sprintf(cmd,"AT+CMGD=%d",i);

    //Send Command
    SIM300Cmd(cmd);

    uint8_t len=SIM300WaitForResponse(1000);

    if(len==0)
        return SIM300_TIMEOUT;

    sim300_buffer[len-1]='\0';

    //Check if the response is OK
    if(strcasecmp(sim300_buffer+2,"OK")==0)
        return SIM300_OK;
    else
        return SIM300_FAIL;
}
```

SIM300 Message Send Receive Demo

To show all the above functions in working demo we have developed a small program. This program does all the basic routine task on boot up, that include the following :-

- Initialize the LCD Module and the SIM300 GSM Module.
- Print IMEI, Manufacturer name and model name.
- Then it checks SIM card presence and connects to network.
- When network connection succeeds its show name of the network. Eg. Airtel or Vodafone etc.
- Finally it sends a message with message body "Test".

- Then it waits for a message, when a message arrives reads it and displays on LCD.
- Then the message is deleted.

```
/*************************************************
*******************************

A basic demo program showing sms functions.

                                          NOTICE
                                 --------
NO PART OF THIS WORK CAN BE COPIED, DISTRIBUTED
OR PUBLISHED WITHOUT A
WRITTEN PERMISSION FROM EXTREME ELECTRONICS
INDIA. THE LIBRARY, NOR ANY PART
OF IT CAN BE USED IN COMMERCIAL APPLICATIONS. IT
IS INTENDED TO BE USED FOR
HOBBY, LEARNING AND EDUCATIONAL PURPOSE ONLY. IF
YOU WANT TO USE THEM IN
COMMERCIAL APPLICATION PLEASE WRITE TO THE
AUTHOR.


WRITTEN BY:
AVINASH GUPTA
me@avinashgupta.com

*************************************************
*******************************/


#include <avr/io.h>
#include <util/delay.h>
```

```
#include "lib/lcd/lcd.h"
#include "lib/sim300/sim300.h"


void Halt();
int main(void)
{
   //Initialize LCD Module
   LCDInit(LS_NONE);

   //Intro Message
   LCDWriteString("SIM300 Demo !");
   LCDWriteStringXY(0,1,"By Avinash Gupta");

   _delay_ms(1000);

   LCDClear();


   //Initialize SIM300 module
   LCDWriteString("Initializing ...");
   int8_t r= SIM300Init();

   _delay_ms(1000);

   //Check the status of initialization
   switch(r)
   {
      case SIM300_OK:
         LCDWriteStringXY(0,1,"OK !");
         break;
      case SIM300_TIMEOUT:
         LCDWriteStringXY(0,1,"No response");
         Halt();
      case SIM300_INVALID_RESPONSE:
```

```
        LCDWriteStringXY(0,1,"Inv response");
        Halt();
    case SIM300_FAIL:
        LCDWriteStringXY(0,1,"Fail");
        Halt();
    default:
        LCDWriteStringXY(0,1,"Unknown Error");
        Halt();
}

_delay_ms(1000);

//IMEI No display
LCDClear();

char imei[16];

r=SIM300GetIMEI(imei);

if(r==SIM300_TIMEOUT)
{
    LCDWriteString("Comm Error !");
    Halt();
}

LCDWriteString("Device IMEI:");
LCDWriteStringXY(0,1,imei);

_delay_ms(1000);

//Manufacturer ID
LCDClear();

char man_id[48];
```

```
r=SIM300GetManufacturer(man_id);

if(r==SIM300_TIMEOUT)
{
   LCDWriteString("Comm Error !");
   Halt();
}

LCDWriteString("Manufacturer:");
LCDWriteStringXY(0,1,man_id);

_delay_ms(1000);

//Manufacturer ID
LCDClear();

char model[48];

r=SIM300GetModel(model);

if(r==SIM300_TIMEOUT)
{
   LCDWriteString("Comm Error !");
   Halt();
}

LCDWriteString("Model:");
LCDWriteStringXY(0,1,model);

_delay_ms(1000);



//Check Sim Card Presence
LCDClear();
```

```
LCDWriteString("Checking SIMCard");

_delay_ms(1000);

r=SIM300IsSIMInserted();

if (r==SIM300_SIM_NOT_PRESENT)
{
   //Sim card is NOT present
   LCDWriteStringXY(0,1,"No SIM Card !");

   Halt();
}
else if(r==SIM300_TIMEOUT)
{
   //Communication Error
   LCDWriteStringXY(0,1,"Comm Error !");

   Halt();
}
else if(r==SIM300_SIM_PRESENT)
{
   //Sim card present
   LCDWriteStringXY(0,1,"SIM Card Present");

   _delay_ms(1000);
}

//Network search
LCDClear();
LCDWriteStringXY(0,0,"SearchingNetwork");

uint8_t     nw_found=0;
uint16_t tries=0;
uint8_t     x=0;
```

```
    while(!nw_found)
    {
       r=SIM300GetNetStat();

       if(r==SIM300_NW_SEARCHING)
       {

 LCDWriteStringXY(0,1,"%0%0%0%0%0%0%0%0%0%0%0%0%0
 %0%0%0");
          LCDWriteStringXY(x,1,"%1");
          LCDGotoXY(17,1);

          x++;

          if(x==16) x=0;

          _delay_ms(50);

          tries++;

          if(tries==600)
             break;
       }
       else
          break;

    }
    LCDClear();

    if(r==SIM300_NW_REGISTERED_HOME)
    {
       LCDWriteString("Network Found");
    }
    else
```

```
    {
       LCDWriteString("Cant Connt to NW!");
       Halt();
    }

    _delay_ms(1000);

    LCDClear();

    //Show Provider Name
    char pname[32];
    r=SIM300GetProviderName(pname);

    if(r==0)
    {
       LCDWriteString("Comm Error !");
       Halt();
    }

    LCDWriteString(pname);

    _delay_ms(1000);

    //Send MSG
    LCDClear();
    LCDWriteString("Sending Msg");

    uint8_t ref;



r=SIM300SendMsg("+919939XXXXXX","Test",&ref);//C
hange phone number to some valid value!

    if(r==SIM300_OK)
```

```
    {
        LCDWriteStringXY(0,1,"Success");
        LCDWriteIntXY(9,1,ref,3);
    }
    else if(r==SIM300_TIMEOUT)
    {
        LCDWriteStringXY(0,1,"Time out !");
    }
    else
    {
        LCDWriteStringXY(0,1,"Fail  !");
    }

    _delay_ms(2000);

    //Wait for MSG
    uint8_t id;

    UFlushBuffer();

    while(1)
    {
        LCDClear();

        LCDWriteStringXY(0,0,"Waiting for msg");

        x=0;
        int8_t vx=1;

        while(SIM300WaitForMsg(&id)!=SIM300_OK)
        {

LCDWriteStringXY(0,1,"%0%0%0%0%0%0%0%0%0%0%0%0%0
%0%0%0");
            LCDWriteStringXY(x,1,"%1");
```

```
    LCDGotoXY(17,1);

    x+=vx;

    if(x==15 || x==0) vx=vx*-1;
}

LCDWriteStringXY(0,1,"MSG Received    ");

_delay_ms(1000);

//Now read and display msg
LCDClear();
char msg[300];

r=SIM300ReadMsg(id,msg);

if(r==SIM300_OK)
{

    LCDWriteStringXY(0,0,msg);

    _delay_ms(3000);

}
else
{
    LCDWriteString("Err Reading Msg !");

    _delay_ms(3000);

}

//Finally delete the msg
if (SIM300DeleteMsg(id)!=SIM300_OK)
```

```
        {
            LCDWriteString("Err Deleting Msg !");

            _delay_ms(3000);
        }

    }


    Halt();
}

void Halt()
{
    while(1);
}
```

Downloads

- Complete AVR Studio 5 Project.