[RF Communication Release] 2010

-By Robosoft Systems

Features

- 2.4 GHz Carrier Frequency
- RS232 UART interface with variable baud rate
- Input supply voltage: 5V to 12V
- 255 possible Channels frequencies (0 to 255)
- Programmable Device Address (255 per channel)
- 2 run mode: Single Byte Transfer Mode and Packet Mode
- Variable Packet length (0 to 40 bytes)
- Standard configuration baud rate of 9600
- User friendly GUI for setting up RF Module (through configuration mode)
- Compact Size, Out of Box: Plug and Play
- On Board EEPROM for saving settings

Jumper Setting

 CONFIG MODE

Closed : Configuration mode

Open : Run mode

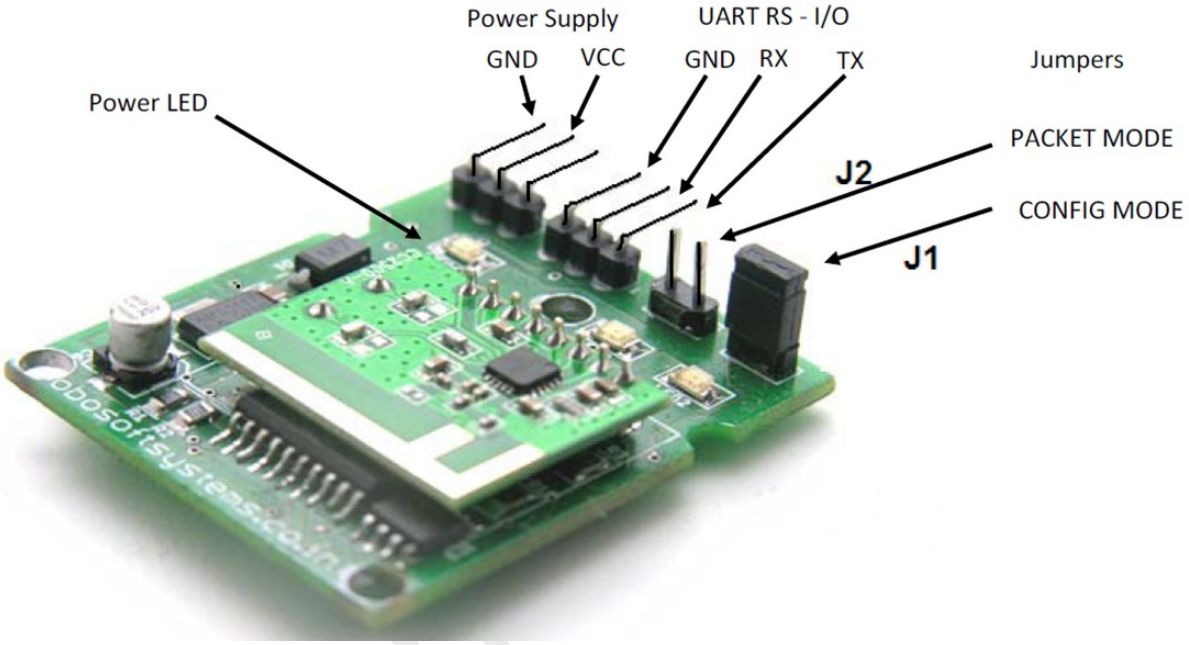
 PACKET MODE

Closed : Variable Packet Length (with device address selection)*

Open : Single Byte Transfer (Broadcast) (80msec delay between 2 char)

**Note: To switch between modes, you have to power on reset module*

Pin Configuration



Jumper Setting Priority

Jumper J1 → Configuration Mode Jumper → Higher Priority

Jumper J2 → Packet Mode/Single Character Mode Jumper → Lower Priority

When both the jumpers are connected, by default, the J1 Jumper (Configuration Mode) Will have higher priority. The Module will still be running in packet mode, but as J1 jumper is connected, configuration mode will be active & the module is ready to receive configuration settings.

Configuring the RF Module

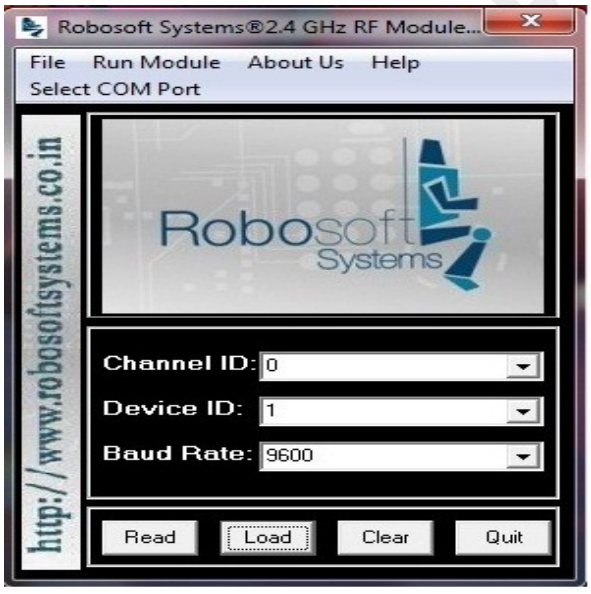
For entering Configuration,

- Power down the RF Module.
- Place the jumper J1, on the RF module
- Re-plug/ power-up the RF module

Note:- These steps are to be followed while switching between Run Mode & Configuration Mode.

Test settings are as below:

RF Module No. 1: 1) Device id – 1 , 2) Channel id – 0 , 3) baud rate – 9600

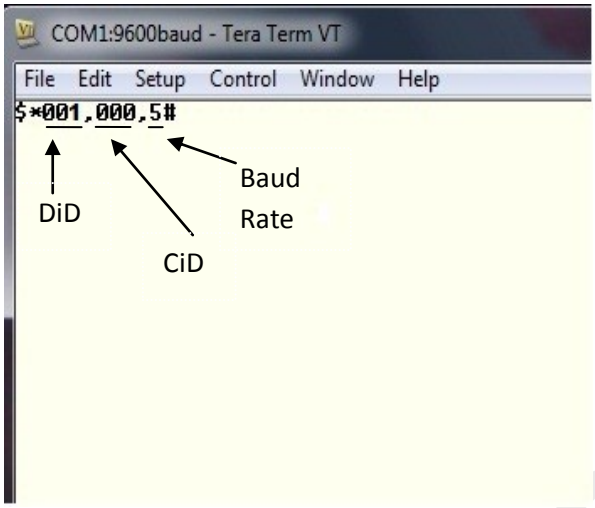


RF Module No. 2: 1) Device id – 2 , 2) Channel id – 0 , 3) baud rate – 9600

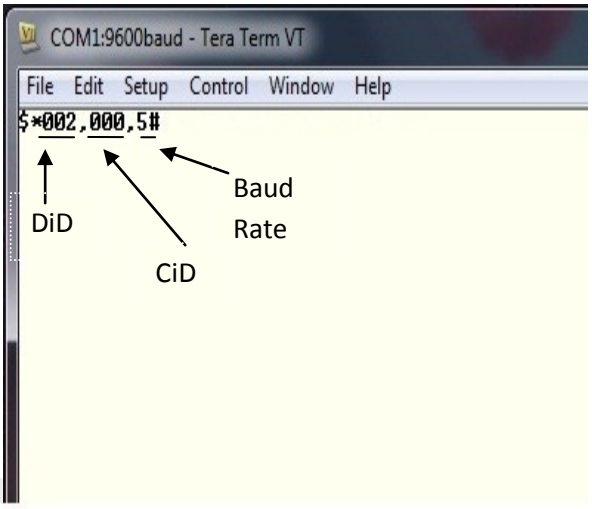
Initial Checking for successful switch into configuration mode & confirming the settings for DiD, CiD & baud rate:

After having placed the Jumper J1, one can check whether the module has entered Configuration Mode by Issuing " Shift + $ " on any Terminal Software . Following are the screenshots for the same check for both the above configured modules.

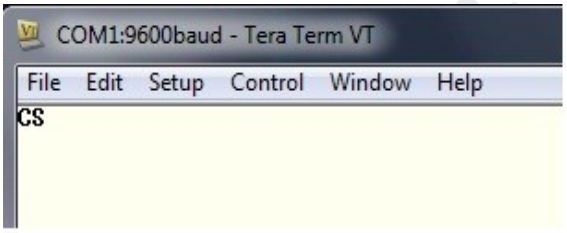
-By Robosoft Systems



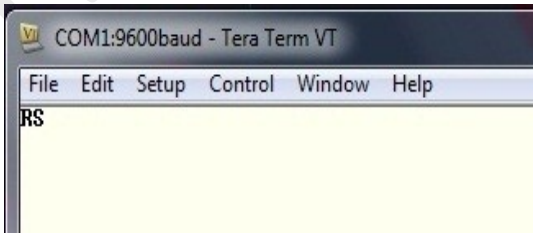
RF Module No. 1



RF Module No. 2



Following response is observed every time when the Module is in Configuration mode and is reset i.e. Power is replugged. Screen-shot for the same is as below.



Remove the Jumper J1 and reset the module, The module has entered Run Mode correctly or not can be verified by following response on Hyper terminal. This response is observed every time when the Module is Run mode and is reset i.e. Power is replugged. Screen-shot for the same is as below.

*__Note:-__ While serial connection properly setup, and power to the module is resetted,

1st Character indicates whether the module is in **C**onfiguration mode or **R**un mode.

2nd Character indicates whether the module is in **S**ingle byte or **P**acket data transfer mode.

Single Character Mode Activation

Before entering the single character mode, jumper J1 needs to be removed while using the modules in packet Mode. Both, Jumper J1 and J2 are not needed while using Single Character mode.

***Note:-** In single character mode, all modules with the same Channel id will receive the data sent by any of the modules on that channel irrespective of device id. This is not true for Packet mode.

After Single Character Mode is activation, open terminal. Create a connection with following settings:

1) Port – Com1, 2) Baud rate – 9600, 3) Flow Control – None, 4) Data – 8bit,

5) Parity – None, 6) Stop Bit – 1bit

Test Case 1: PC to PC Wireless communication in Single Character Mode using CC2500 based RF modules.

Initial Settings for Test Case 1: Enter configuration mode using jumper J1 only.

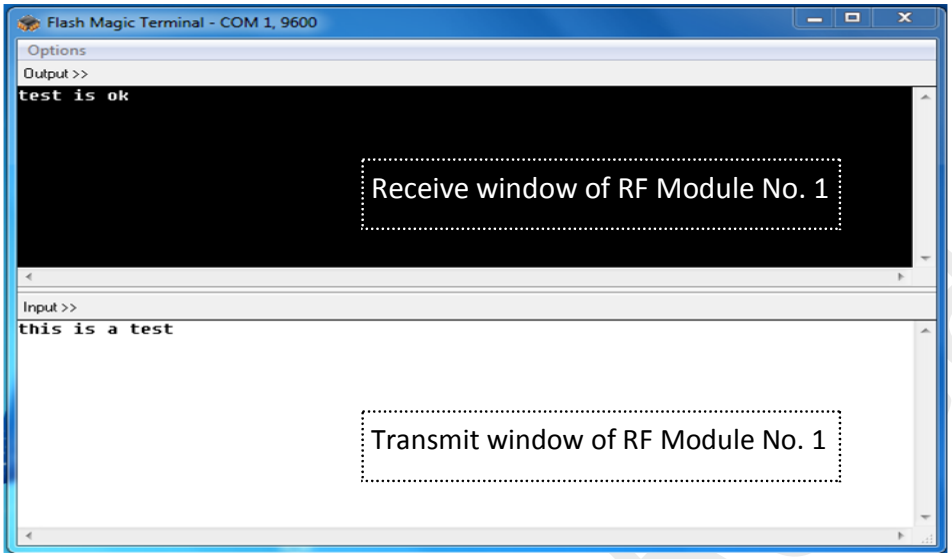
Configuration mode :

Follow the steps over page 3, 4 & 5 to configure the module.

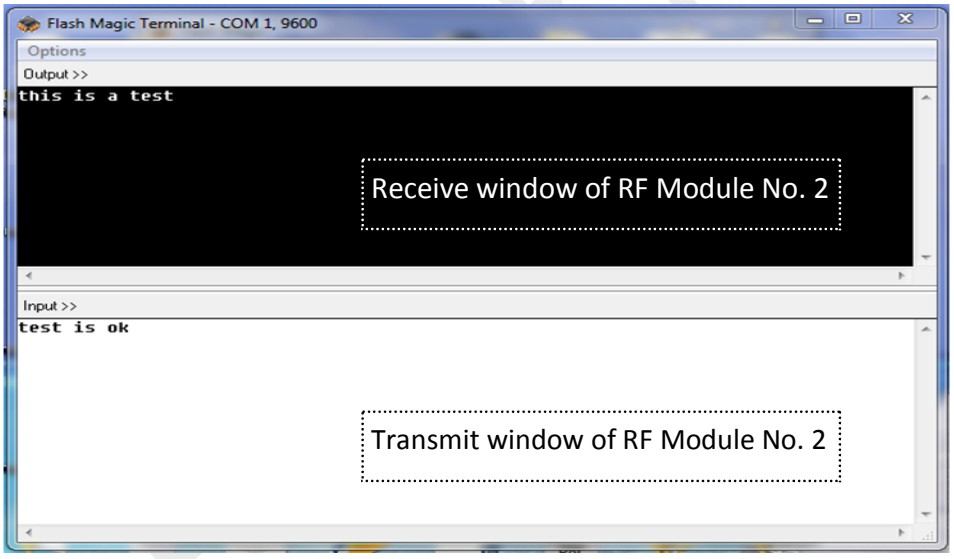
RF Module No. 1: 1) Device id – 1 , 2) Channel id – 0 , 3) baud rate – 9600

RF Module No. 2: 1) Device id – 2 , 2) Channel id – 0 , 3) baud rate – 9600

Testing Communication between Modules: Once the 2 modules have been connected to 2 different PC's and are powered on with the terminal initialized on both of them, the connection may be tested. Any data transmitted from one PC will appear on the other PC, character by character. Check figure below for input and output on two different PC's.



RF Module No. 1



RF Module No. 2

Test Case 2: BOT to BOT Wireless communication in Single Character Mode using CC2500 based RF modules

The example shows the implementation of data transfer using the modules. The following code will allow LED1 of BOT1 to be synchronized with LED1 of BOT2.

Initial Settings for Test Case 2: Enter configuration mode using jumper J1 only.

Configuration mode :

Follow the steps over page 3, 4 & 5 to configure the module.

RF Module No. 1: 1) Device id – 1 , 2) Channel id – 0 , 3) baud rate – 9600

RF Module No. 2: 1) Device id – 2 , 2) Channel id – 0 , 3) baud rate – 9600

Single Character Mode Activation: The jumper J2 needs to be removed while using the modules in packet Mode. Jumper J1 and J2 are not needed while using Single Character mode.

- **Below is the code required to be programmed in both the bots:**

The code part labeled as bot1 will be the master. For the master the bot2 code must be commented. The code part labeled as bot2 is for slave. For the slave the bot1 code must be commented.

In the below code, the Bot2 section (slave bot) is shown as commented. Make sure to use proper commenting for each case, in order for the code to work correctly.

```
#include <avr/io.h>              //      header file defining  I/o operation

#define F_CPU 8000000            //      CPU @ 8MHZ

#define USART_BAUDRATE 9600      //      SERIAL Communication @ 9600 baud

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)    // baud rate formula

#include <util/delay.h>                  //      header file defining  delay

int main (void)          //      main code begins

{

        DDRB = 0xff;  //      PORT B as output

        DDRD = 0xfc;  //      PORT D PIN 3-7 as output. PIN 2 is input for switch

        UCSRB |= (1 << RXEN) | (1 << TXEN);          //      Enable Rx & Tx of USART

        UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);       // 8bit, 1 Stop bit, no parity

        UBRRL = BAUD_PRESCALE;   //      setting baud rate @ 9600 baud

        UBRRH = (BAUD_PRESCALE >> 8);

        unsigned char a=1;

        unsigned char b;        // Character Variable for Rx packet.

        while(1)        // loop forever
```

```
{
/*
//BOT1 code
        //Tx Code
        if (a == '0')
                a='1';
        else
                a='0';
        UDR = a;        //      Add next byte to be transmitted to the UDR buffer
        while ((UCSRA & (1 << UDRE)) == 0); // Loop until UDR buffer is ready to receive
                                            // next  byte to be transmitted.
        while ((UCSRA & (1 << RXC)) == 0); // Loop until UDR buffer is receiving RECEIVE data.
        b = UDR;        //      Read byte by byte from the Received Data Packet into the variable
        if (b == '1') //check for Actual data from received Packet to be =  (ASCII - '1')
        {
                PORTB = 0x01;           // led1 on led2 off if true if a[2] == 0x52
        }
        else
        PORTB = 0x00;
```

```
        _delay_ms(1000);
*/
/*
//BOT2 code.
        //Rx Code
        while ((UCSRA & (1 << RXC)) == 0); // Loop until UDR buffer is receiving RECEIVE data.
        b = UDR;        //      Read byte by byte from the Received Data Packet into the variable
        if (b == '1') //check for Actual data from received Packet to be = 0x52 (ASCII - R)
        {
                PORTB = 0x01;           // led1 on led2 off if true if a[2] == 0x52
        }
        else
        PORTB = 0x00;
        if (a == '0')
        a='1';
        else
        a='0';
        UDR = a;        //      Add next byte to be transmitted to the UDR buffer
```

```
        while((UCSRA & (1 << UDRE)) == 0); // Loop until UDR buffer is ready to receive
                                           // next  byte to be transmitted.
        _delay_ms(1000);
        */
        }       // While loop closed.
    return(0);          // syntax never reached
}
```

Note 1:- Mount the RF modules, one each on the mounting space provided on the two bots. Switch on the two bots.

Note 2:- To trace the communication simultaneously on Terminal, open terminal on separate Computers one for each bot. Create a connection with following settings:

1) Port – Com1, 2) Baud rate – 9600, 3) Flow Control – None, 4) Data – 8bit,

5) Parity – None, 6) Stop Bit – 1bit

- **Make use of the 3 wire serial interface provided on the board besides the MCU to watch the signals.**
- **You should get a stream of toggling 0's and 1's in terminal.**

Packet Mode Activation: The jumper J2 needs to be placed while using the modules in packet Mode. Jumper J1 is not needed while using in packet mode. Make sure that you note down the device id.

After the packet mode is activated, open Hyper terminal. Create a connection with following settings:

1) Port – Com1, 2) Baud rate – 9600, 3) Flow Control – None, 4) Data – 8bit,

5) Parity – None, 6) Stop Bit – 1bit

Test Case 3: PC to PC Wireless communication in Packet Mode using CC2500 based RF modules

Packet Structure for Transmit Data in Packet Mode

Start Character	Packet Length	Device ID	Actual Data

The format of the transmitted packet is as shown above. Its typical value are given in the below column.

FRAME DEFINATION	VALUES	TYPE
Start Character	#	char
Packet Length	0x02 – 0x40	char
Device ID	0x00 – 0xff	char
Actual Data	Actual Data Size = packet length - 1	Array of char

***Note:-** In case of packet mode transmission, the packet send by a device will be received by all the devices on the same channel id, but only the one with the same Device id (matching to packet) will accept the packet. The other devices will discard the received packet.

Note:- The device id is at the receiver end is matched, and if found matching to its own, the packet is accepted. As a result, the user will find the following packet at the 3-wire serial interface end.

Packet Structure for Receive Data in Packet Mode

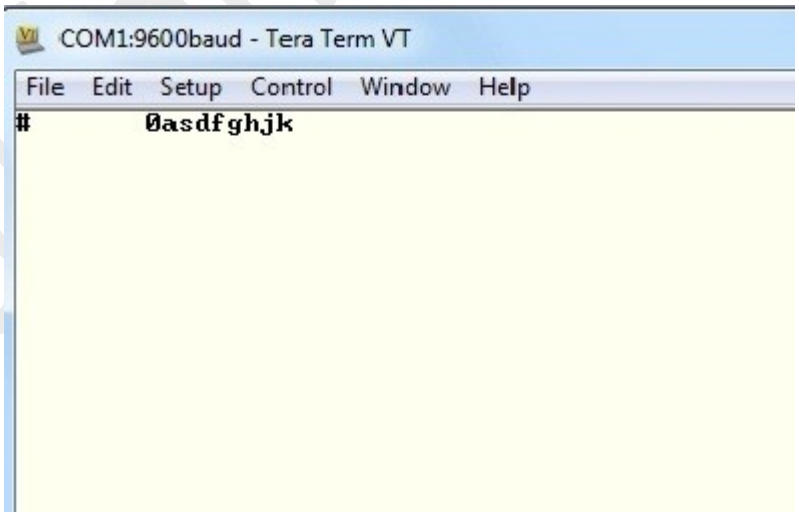
Start Character	Packet Length	Actual Data

The format of the received packet is as shown above. Its typical value are given in the below column.

FRAME DEFINATION	VALUES	TYPE
Start Character	#	char
Packet Length	0x02 – 0x40	char
Actual Data	Actual Data Size = packet length - 1	Array of char

Below is the screen shot of a example packet transmission

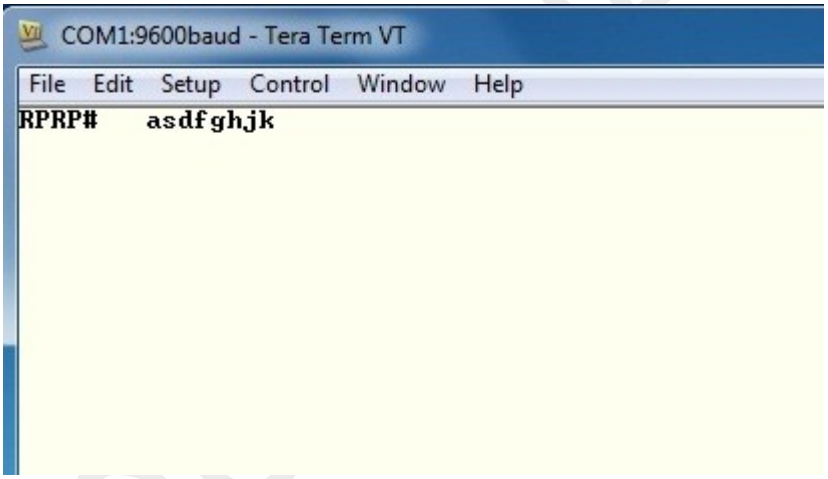
Tx packet:



FRAME DEFINATION	ASCII VALUES to be send on hyper Terminal	Hex Equivalent
Start Character	#	23
Packet Length	Tab	9
Device ID	0	30
Actual Data	asdfghjk	asdfghjk

Below is the screen shot of a example packet reception

Rx packet:



#	Tab	asdfghjk

FRAME DEFINATION	ASCII VALUES received on hyper Terminal	Hex Equivalent
Start Character	#	#
Packet Length	Tab	Tab
Actual Data	asdfghjk	asdfghjk

Test Case 2: BOT to BOT Wireless communication in Packet Mode using CC2500 based Robosoft RF modules.

Initial Settings for Test Case 2: Enter configuration mode using jumper J1 only.

Configuration mode :

RF Module No. 1: 1) Device id – 1 , 2) Channel id – 0 , 3) baudrate – 9600

RF Module No. 2: 1) Device id – 2 , 2) Channel id – 0 , 3) baudrate – 9600

Packet Mode Activation: The jumper J2 needs to be placed while using the modules in packet Mode. Jumper J1 is not needed while using in packet mode. Make sure that you note down the device id.

Below is the code required to be programmed in both the bots:

The code part labeled as bot1 will be the master. For the master the bot2 code must be commented. The code part labeled as bot2 is for slave. For the slave the bot1 code must be commented.

In the below code, the Bot2 section (slave bot) is shown as commented. Make sure to use proper commenting for each case, in order for the code to work correctly.

```
/* code for packet mode communication */

#include <avr/io.h>                 //      header file defining  I/o operation

#define F_CPU 8000000               //      CPU @ 8MHZ

#define USART_BAUDRATE 9600         //      SERIAL Communication @ 9600 baud

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)    // baudrate formula

#include <util/delay.h>>            //      header file defining  delay

int main(void) >            //      main code begins

{

   DDRB = 0xff;             //      PORT D as output

   DDRD = 0xfc;             //      PORT D PIN 2-7 as output.

        UCSRB |= (1 << RXEN) | (1 << TXEN);         //      Enable Rx & Tx of USART

        UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);       // 8bit, 1 Stop bit, no parity

        UBRRL = BAUD_PRESCALE;  //      setting baud rate @ 9600 baud

        UBRRH = (BAUD_PRESCALE >> 8);
```

```
while(1)        // loop forever
 {
        //bot1 code
        unsigned char s=0;    //        for use as a counter
        unsigned char a[] = {0x23, 0x02, 0x30, 0x52, 0x00};        // Tx packet in the forrmat
       of //char array a[] = {0x23, 0x02, 0x30, 0x52, 0x00}, where 0x00 is not the part of ac-
       tual //tx packet but is rather used to indicate within this code the end of packet.
        while (a[s] != '\0')     // Loop to send tx packet through USART while scanning for '\0'
        {
        while ((UCSRA & (1 << UDRE)) == 0);          // Loop until UDR buffer is ready to re-
ceive // next  byte to be transmitted.
        UDR = a[s];    //        Add next byte to be transmitted to the UDR buffer
        s++;    //        Increment the counter
        _delay_ms(50);        //        delay of .02ms
        }
        _delay_ms(2000);    //        delay of 2 sec
        s=0;    //        reset counter
        a[3] = 0x53;    //modify char array to a[] = {0x23, 0x02, 0x30, 0x53, 0x00},
```

```
while (a[s] != '\0')      // Loop to send tx packet through USART while scanning for '\0'
        {       // While loop opened.
        while ((UCSRA & (1 << UDRE)) == 0);          // Loop until UDR buffer is ready to re-
ceive // next  byte to be transmitted.
        UDR = a[s];    //       Add next byte to be transmitted to the UDR buffer
        s++;    //       Increment the counter
        _delay_ms(50);         //       delay of .02ms
        }
        _delay_ms(2000);     //       delay of 2 sec
/*
   //bot2 code
        unsigned char a[3];   // Char Array for Rx packet.
        unsigned char s;      //       for use as a counter
        while ((UCSRA & (1 << RXC)) == 0);  // Loop until UDR buffer is receiving RECEIVE data.
        for (s=0;s<3;s++)     //       loop for storing the received Rx packet as char array
        {
        while ((UCSRA & (1 << RXC)) == 0);  // Loop until UDR buffer is receiving RECEIVE data.
```

```
        a[s] = UDR;     //        Read byte by byte from the Received Data Packet into the array
        }
        if(a[2] == 0x52) //check for Actual data from received Packet to be = 0x52 (ASCII – R)
        {
        PORTB = 0x01;          // led1 on led2 off if true if a[2] == 0x52
        }
        else
        {
        PORTB = 0x02;          // led1 on led2 off if true if a[2] == 0x52
        }
*/
        }       // While loop closed.
    return(0);          // syntax never reached
}
```


Note 1:- Mount the RF modules, one each on the mounting space provided on the two bots. Switch on the two bots.

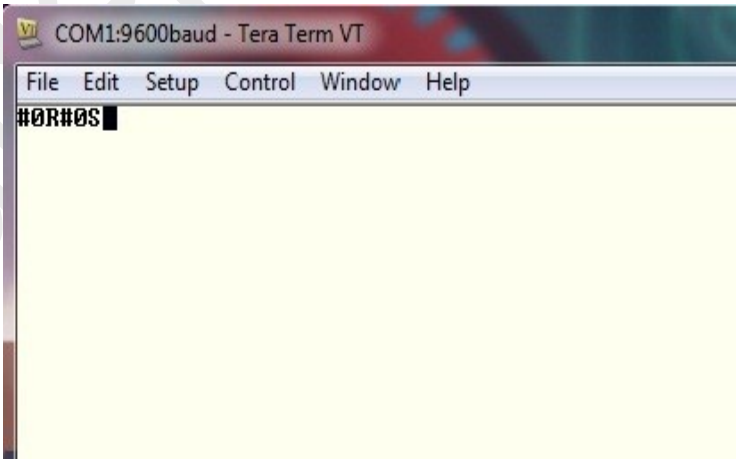
Note 2:- To trace the communication simultaneously on Terminal, open terminal on separate Computers one for each bot. Create a connection with following settings:

1) Port – Com1, 2) Baud rate – 9600, 3) Flow Control – None, 4) Data – 8bit,

5) Parity – None, 6) Stop Bit – 1bit

- **Make use of the 3 wire serial interface provided on the board besides the MCU to watch the signals.**
- **You should get a stream of toggling 0's and 1's in terminal.**

Below is the screen shot of a example packet transmission



Tx packet :

Packet to transmit data : R

#	STX	0	R

FRAME DEFINATION	ASCII VALUES	HEX Equivalent values to be send
Start Character	#	0x23
Packet Length	STX (Start of Text)	0x02
Device ID	0	0x30
Actual Data	R	0x52

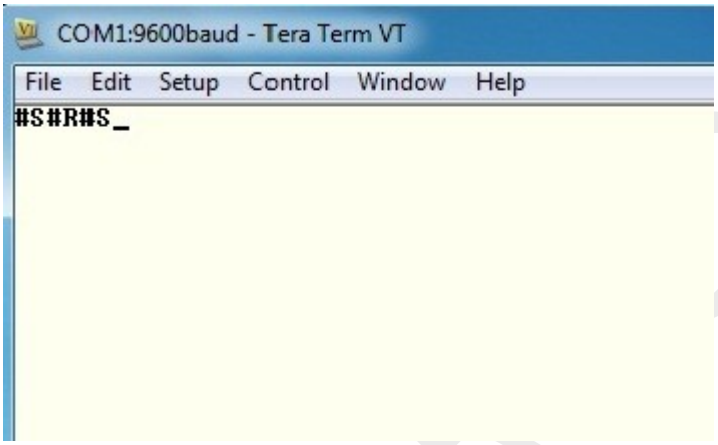
Packet to transmit data : S

#	STX	0	S

FRAME DEFINATION	ASCII VALUES	HEX Equivalent values to be send
Start Character	#	0x23
Packet Length	STX (Start of Text)	0x02
Device ID	0	0x30
Actual Data	R	0x52

Below is the screen shot of a example packet reception

Rx packet:



Received packet for data : R

#	STX	0x52

FRAME DEFINATION	ASCII VALUES	HEX Equivalent values re-
Start Character	#	0x23
Packet Length	STX (Start of Text)	0x02
Actual Data	R	0x52

Received packet for data : S

#	STX	0x53

FRAME DEFINATION	ASCII VALUES	HEX Equivalent values re-
Start Character	#	0x23
Packet Length	STX (Start of Text)	0x02
Actual Data	S	0x53