Application Note

Wireless RS232

Table of Contents

1. Introduction 3
2. Wireless RS232 Hardware 4
2.1. Wireless RS232 principle 4
2.2. Set - up 5
2.3. Connections 6
3. Wireless RS232 Software 6
3.1. Source files 6
3.1.1. Initialisation.c 7
3.1.2. DFLLDRIVER.c 7
3.1.3. XE1203driver.c 8
3.1.4. RS232CommHandler.c 9
3.1.5. Main.c 10
3.2. How to download the code into the wireless RS232 link 10
4. UART and HyperTerminal configuration 11
5. How to use the wireless RS232 link 13
6. Limitations and further improvements 14
7. References 14

1. INTRODUCTION

This document describes the low-level design of a wireless RS232 link using a combination of 2 Semtech products: the XE1203 transceiver and the XE88LC06A low power microcontroller.

It can be seen as a base for full implementation of a wireless RS232 link. The modem functionalities are completely transparent, but the data rate, the frequency and the UART parameters can be defined by the user.

This design is based on the Semtech technical note “TN8000.18, XE8000 driving XE1200 transceiver standard API definition” and uses the BitJockey™ RF Receiver-Transmitter interface.

It is assumed that the user is familiar with Windows, has at least some familiarity with the CoolRISC 816 family and the C programming language.

The hardware set-up for the wireless RS232 is presented and the software implementation is described in detail. The last section shows the actual wireless RS232 capacities, its limitations and how the design could be upgraded to add more flexibility to the system.

2. HARDWARE DESCRIPTION

2.1. PRINCIPLE

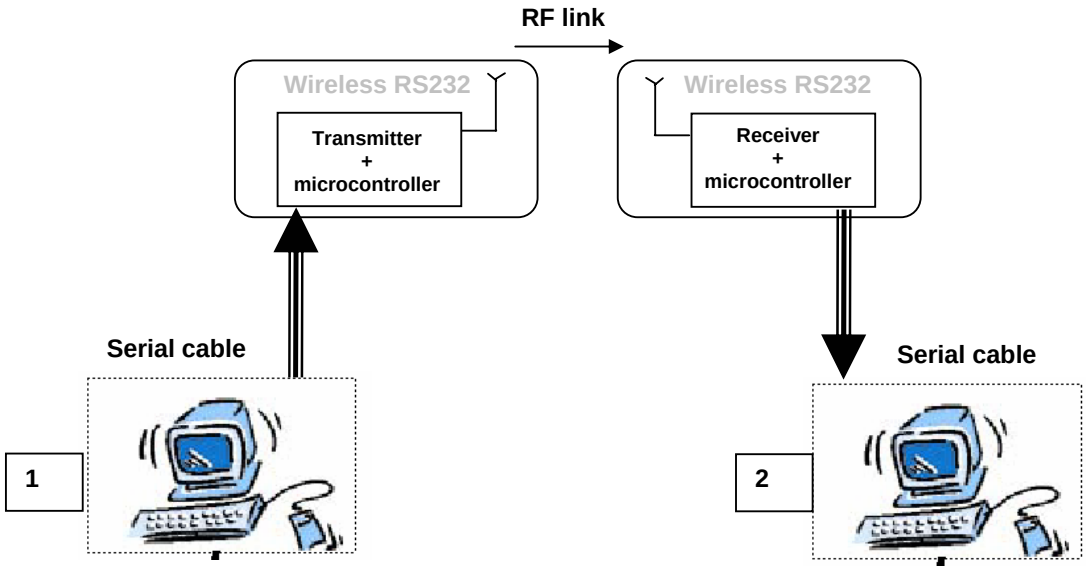


Figure 1: Wireless RS232 principle

PC number 1 transmits the packets to the microcontroller via a RS232 cable. The microcontroller carries out packet formatting and RF header insertion. The transmitter sends these packets over the RF link. When the data are received correctly on the other side, the microcontroller performs payload extraction and finally the packets are transmitted to the attached PC number 2.

The user can write the characters to be sent over the wireless RS232 link on the PC via the HyperTerminal software. He will then be able to read the characters he has send on the HyperTerminal of the remote PC.

2.2. SET - UP

The wireless RS232 set-up includes a complete receiver and transmitter system. Each system consists of a XE8000MP and a XE8000EV108 to program the XE88LC06A microcontroller, a XM1203, a serial cable and a PC to transmit or receive the data.

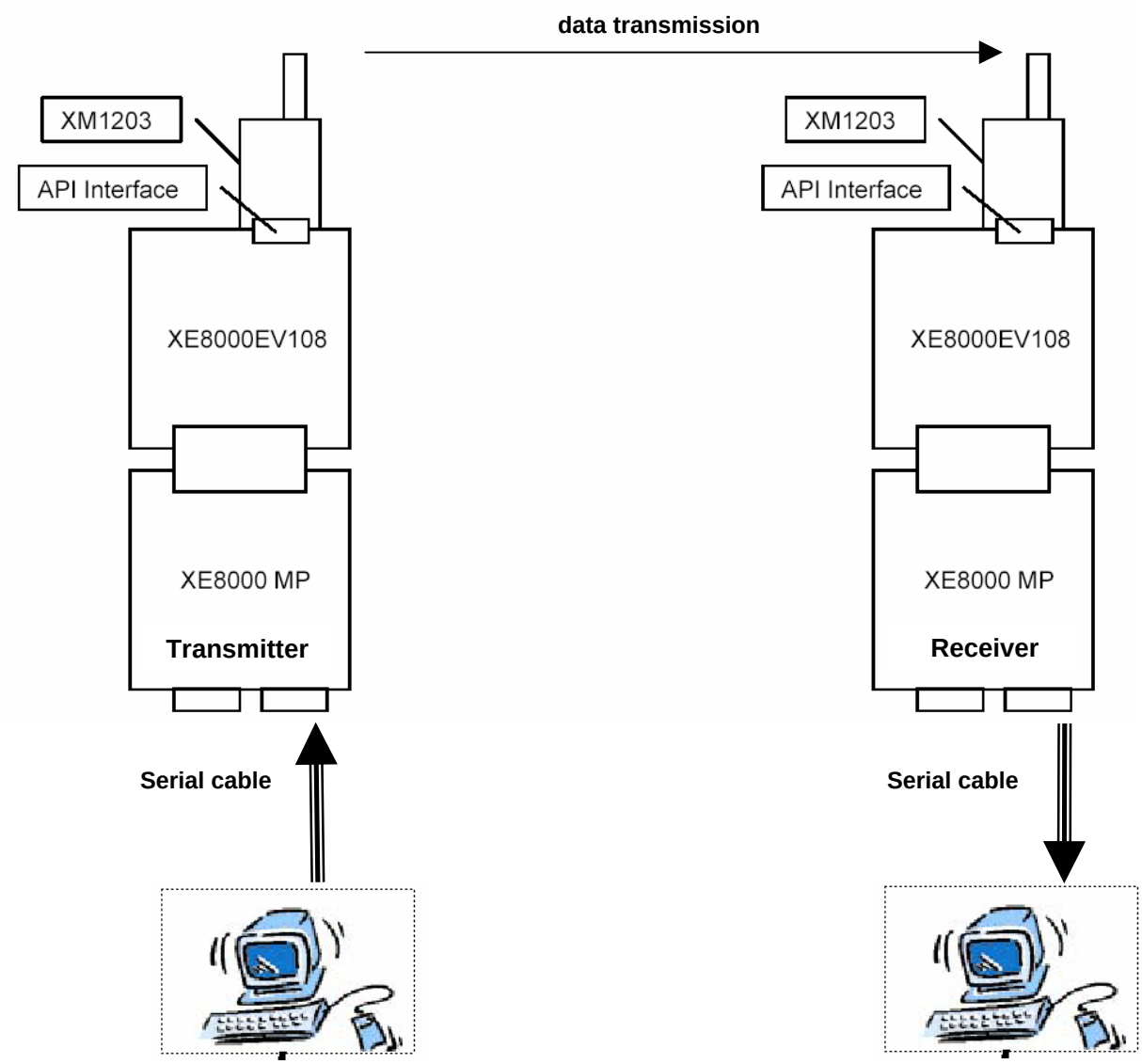


Figure 2: Wireless RS232 set-up

In this application, the BitJockey™ interface is enabled. The BitJockey™ is a PCM Bit Stream Encoder-Decoder which simplifies the handling of the low level data for wireless data system, using techniques similar to that of a UART interface for wired transmission systems. In a normal microcontroller, the bits and low level coding of the RF protocol are handled by software. This is both time consuming and code inefficient since the processor has to handle each bit as it is received or as it is about to be transmitted.

The connections between the XE88LC06A and the XM1203 (API Bus) will be detailed in the next section.

2.3. CONNECTIONS

The code has been implemented on the Semtech XE88LC06A microcontroller.

Connections between the XM1203 and the XE88LC06A BitJockey™:

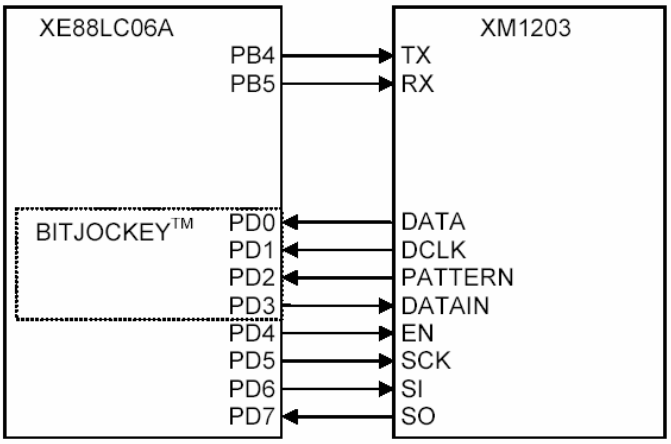


Figure 3: XM1203 to XE88LC06A connections

NB: Port D shown in Figure 3 is actually Port C on the XE8000 evaluation board.

3. WIRELESS RS232 SOFTWARE

The code has been written in C language. There are 2 files per driver, a ".c" file and a ".h" file, ie XE1203driver.c and XE1203driver.h, the ".c" file contains the function implementation and the ".h" file contains the function prototype, #define directives and macros.

3.1. SOURCE FILES

The main program calls the following sub-functions summarized in the table below:

file	description
initialization.c	XE88LC06A microcontroller initialisation
dflldriver.c	RC oscillator frequency calibration
xe1203 driver.c	XM1203 initialisation
	operation control (Tx, Rx)
RS232commhandler.c	UART handling functions and initialisation

3.1.1. Initialisation.c

The file “Initialisation.c” contains all the functions used to initialize the XE88LC06A.
By default the initialization sets the RC at 2457600 Hz using the crystal as external reference for the DFLL (Digital Frequency Locked Loop)

InitMicro: Initializes the MicroController peripherals, mainly the A, B and D ports, the counters and the RC oscillator to 2457600 Hz.

InitPortA: By definition, Port A is an input digital port. This function sets Port A in pull up and debounce mode
InitPortB : This function sets Port B as an output with no open drain outputs and no pull-up. The chosen mode is digital.
InitPortD: Port D is set as an output.
InitCounters : The counters A,B,C and D are initialized to zero, the RC oscillator and the external crystal oscillator are enabled. It waits for cold crystal and calls the “DFLLRun” function.
InitRCoscillator **:** This function initializes the RC oscillator (DFLL)
InitUART: This is the function where the UART is initialized. The input parameters of this function are the baud rate, the length of the data, the parity and the RC factor. These variables are defined by the user.

The UART is fixed to a baud rate of 19200, a data length of 8 bits, an odd parity and the RC factor is set to 2 times the default RC frequency. The UART parameters are user-definable.

3.1.2. DFLLDRIVER.c

The file “DFLLDRIVER.c” calibrates the frequency of the RC oscillator until the desired frequency is found. It is called by the “Initialisation.c” details are in the technical note TN8000.09 [1]

3.1.3. XE1203driver.c

The file XE1203Driver.c contains the appropriate functions to drive the XM1203:

The baud rate, the frequency or any of the XM1203 parameters can be modified in the configuration "RegistersCfg". In this table, most of the variables used to control the XM1203 can be found. However, note that the registers "DataOut1" and "DataOut2" are not writable and have not been included in the "RegisterCfg" variable.

The XE1203driver.c functions are fully explained in the technical note TN8000.18 [3].

A quick summary of these functions is given below:

Configuration functions:

- InitRFChip : This function sends the values contained in the "RegisterCfg" variable to the transceiver
  It also initializes the baud rate and pattern detection variable.
- SetRFMode : This function defines the transceiver operating mode (Sleep, Receiver, Transmitter)
- SetModeRFIF: This function initializes the BitJockey™ registers to set a predefined mode as described in the XE88LC06A datasheet [4]
- WriteRegister : This function writes to the XM1203F register via dedicated programming pins, following the protocol given in the transceiver datasheet.
- ReadRegister : This function reads from registers via dedicated programming pins, following the protocol given in the transceiver datasheet.

Communication functions:

- SendRfFrame : This function encapsulates the given user buffer in a RF frame and sends it to the transceiver.

The RF frame format is described below:

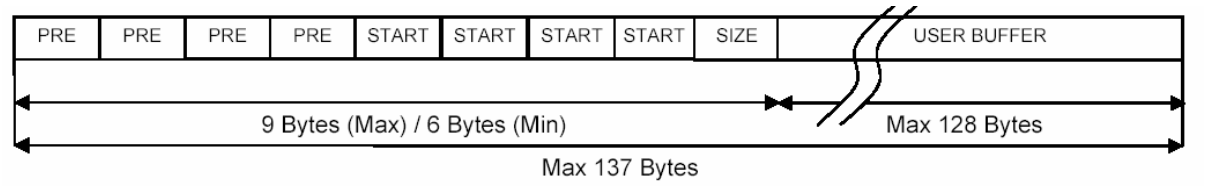


Figure 4: RF frame format

- ReceiveRfFrame : This function receives a RF frame from the transceiver and returns the user buffer and the user buffer size.

3.1.4. RS232CommHandler.c

Description:

The file RS232CommHandler.c interfaces between the microcontroller and the PC. It regroups all the PC protocol and UART handling functions. The packet format and the various commands will be described in this section.

The HyperTerminal 1 will be related to PC1, which is the one transmitting the information to the second PC, and HyperTerminal 2 is connected to PC2 receiving the information.

User commands on HyperTerminal 1

start of packet	command byte	user buffer		end of packet
*	CmdReadRegister : R	address to read the value		carriage return
*	CmdWriteRegister : W	address to write the value	value to write	carriage return
*	CmdSetRFMode : M	RF mode (sleep, stby, rx,tx)		carriage return
*	CmdSendRfFrame : _	bytes to send over the RF modem		carriage return

Figure 5: HyperTerminal 1 commands

HyperTerminal 1 response

start byte	acknowledgement / non-acknowledgement	
*	acknowledge : a	acknowledgement
*	unknown command: u	not a valid command byte (cf previous table)
*	packet error : p	no start of packet and/or no end of packet
*	size error : s	packet is not of the correct size

Figure 6: HyperTerminal 1 ack/nack

The HyperTerminal 1 will set the wireless RS232 link in transmit mode only if a valid “SendRfFrame” command is sent by the user which means that there are start and end of packet commands and the packet doesn’t exceed 52 characters.
In addition to the wireless RS232 link functionalities, 3 other commands are available such as read or write in a register and set the XM1203 in a given mode (sleep, standby, receive or transmit mode).

HyperTerminal 2

The HyperTerminal 2 is attached to the receiving system - that system is constantly in receiving mode. Whenever a frame with a valid format is detected by the microcontroller, the data is sent to the UART and then appears on the HyperTerminal 2.

Protocol function and UART handling functions:

- Decode packet: This function decodes the packet sent by the PC. If the start byte has been detected, there is an identification of the command amongst read / write register, set mode and send RF frame (wireless RS232 link).
If the command doesn’t correspond to any of the commands listed above, there is a packet error response.

- OnTxInit: This function starts the packet transmission to the PC. It initializes the Tx buffer pointer and sends the first byte (Start byte).

- OnTx: This function initializes the transmission buffer each time a transmission event is generated.

- OnRx: This function handles the receiver event and executes the different commands sent by the PC. If the packet is of the correct size and a valid command has been identified, the microcontroller sends back either an acknowledgement when the requests have been completed, or a packet error or a non-acknowledgement if the size is not correct or the command doesn't fit in the list of microcontroller commands.

- ReceiveData: This function receives the data via the RF link and sends them to the UART.

- OnComm: This function handles the events generated by the Rx and Tx IRQ handler. The main purpose of this function is to test which event needs to be handled. When the event is a Rx Signal, it calls the "OnRx" function and "OnTx" in case the event is a Tx Signal.

- OnCommInit: This function initializes the communications.

- Handle_Irq_uartTx: This function handles the interruption from the Uart Tx. If the Tx buffer can be used, send the Tx Buffer content to the Tx UART, else free the Tx buffer to the main program.

- Handle_Irq_UartRx: This function handles the interruption from the Uart Rx. When the Rx buffer can be used, the data in the Rx UART are transferred into the Rx buffer.

3.1.5. Main.c

In the main project function (located in main.c) we need to call the functions InitRFChip and InitUART.

Once the transceiver and the UART are initialized the function enters an infinite loop where the application is tested to determine whether it is a transmission or a reception, and then calls the corresponding function that will implement the transmission or reception function.

The main function is similar to that described below:

```
int main (void){
   InitMicro();

_Monitor_Init();
_Monitor_SoftBreak;

   InitUART(19200, 1, 0, 0, 4);
   CommInit();
   InitRFChip();

//Main Loop
   while(1){

      if (Transmission)
      OnComm();
      else
      ReceiveData();
      }

   return 0;
```

3.2. HOW TO DOWNLOAD THE CODE INTO THE WIRELESS RS232 LINK

The source code for the application has been detailed above. The project can now be compiled with RIDE and no errors should be generated. If needed, the technical note TN8000.16 "coding with RIDE, Quick start" [5] explains how to compile and download a code.
To test the application, set the variable "volatile _U8 Transmission" to true in the main.c file, compile the project and download the code to the transmitter system.
Change the variable "volatile _U8 Transmission" from true to false in the main.c file, compile the project and download the code into the receiver system.

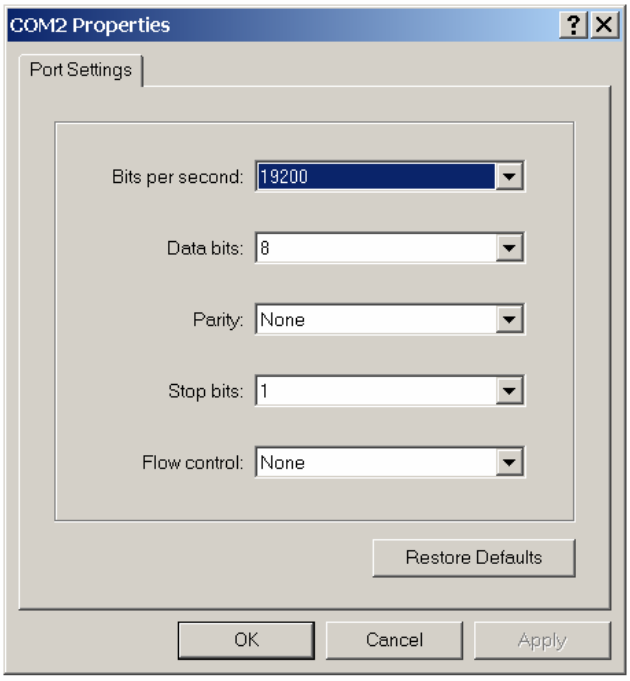
4. UART AND HYPERTERMINAL CONFIGURATION

The UART configuration is set in the file initialization.c.

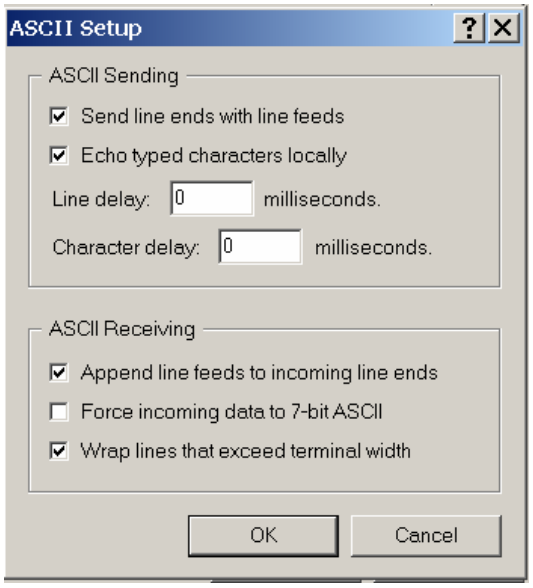
InitUART (baud rate :19200,data length:1,parity : 0,parity enable : 0,rc factor : 4)

- the baud rate is set to 19200 bits/s
- the data length is 8 bits
- the parity is disabled
- the rc factor is 4 times the default rc frequency

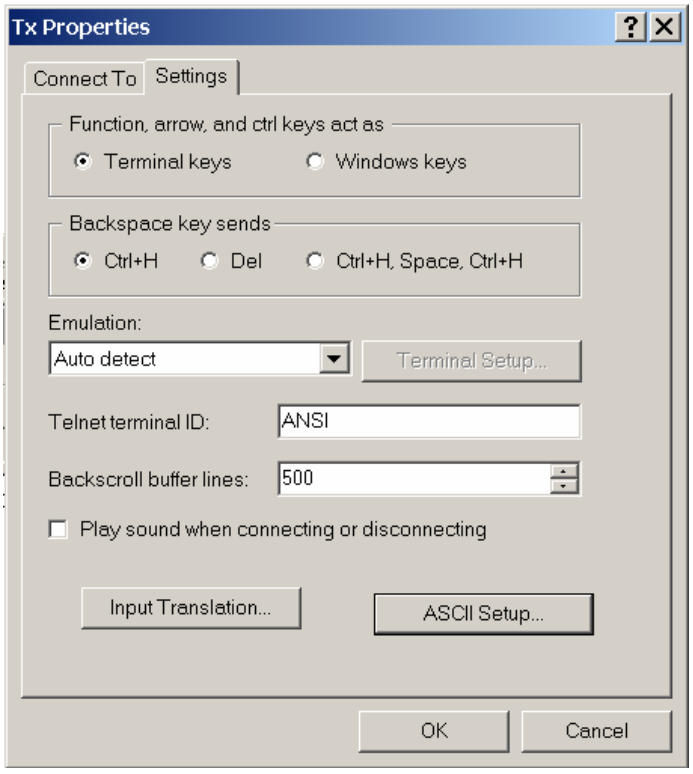
The 2 ComPort settings in the HyperTerminal software are shown below:



To have a user-friendly version of the HyperTerminal, tick the following boxes in the ASCII set-up.



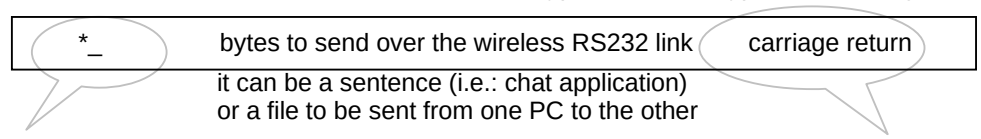
The ASCII set-up can be found under : file > properties > settings:



When the 2 HyperTerminals have been configured correctly and the code has been downloaded in the 2 units, the wireless RS232 link is ready to start.

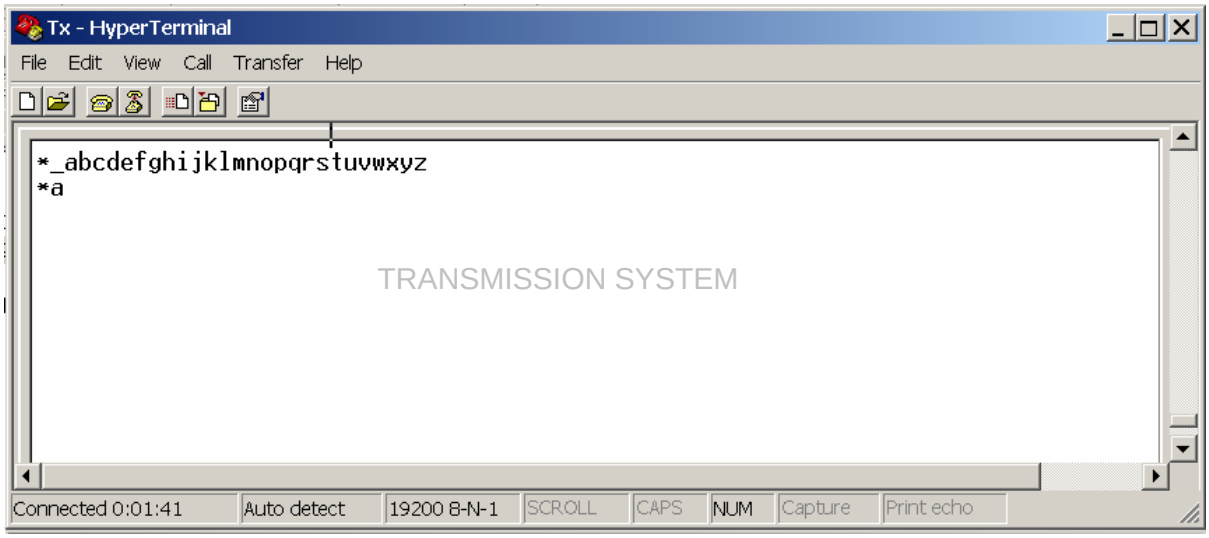
5. HOW TO USE THE WIRELESS RS232 LINK

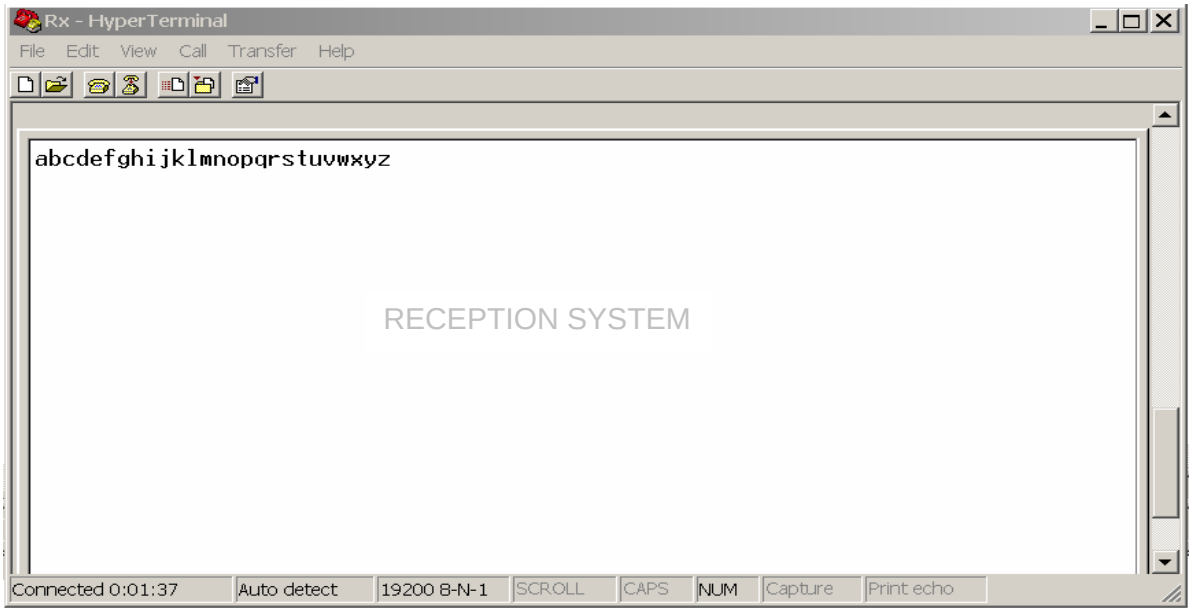
As explained in section **3.1.4**, to send a command on HyperTerminal1, type the following line:



The Hyperterminal1 will answer:
"*a" if the packet has been sent correctly,
"*u" if the "_" has not been typed correctly,
"*p" if there is no start or no end of packet
"*s" if the packet is not of the correct size

If "*a" appears on HyperTerminal1, the sentence written on HyperTerminal1 will be sent via the wireless RS232 link and can be read on the HyperTerminal2.





6. LIMITATIONS AND FURTHER IMPROVEMENTS

A wireless RS232 link including hardware and software has been presented in this application note. The software allows for a one-way communication, one system (microcontroller, RF transceiver plus PC) being the transmitter and the other one the receiver. But the hardware platform is ready for a bi-directional link.

For a chat application or for files transfer from one PC to another; both computers should be able to transmit and receive data. Also the data would have to be transformed into packets of the right size (maximum size being 52 characters so far). To add more flexibility onto the wireless RS232 link, protocol level design should be added on top of the design detailed in this application.

7. REFERENCES

[1] TN8000.09: D.F.L.L. Digital Frequency Locked Loop
http://www.xemics.com/xe8000/xe88lc06
[2] XE1203 datasheet
http://www.xemics.com/xe1200/xe1203/
[3] TN8000.18:XE8000 driving XE1200 transceivers standard API definitions
http://www.xemics.com/xe8000/xe88lc06
[4] XE88LC06A and XE88LC07A datasheet
http://www.xemics.com/xe8000/xe88lc06
[5] TN8000.16: Coding with RIDE quick start
http://www.xemics.com/xe8000/xe88lc06

Contact Information

Semtech Corporation
Wireless and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804