

Xbee module configuration from a μcontroller

Soulier Baptiste

Polytech Clermont Ferrand

2012-2013





The purpose of this application note is to explain how to configure the main options of a Xbee RF module with a basic μ controller.

Table of Contents

Introduction	2
I. The Xbee modules	3
I.I. General Overview	3
I.II. Main Configuration's Options	4
I.III. Configuration's Stream	5
II. Xbee configuration from a μController	7
II.I. Connection Xbee $\Leftrightarrow \mu$ Controller	7
II.II. Designing a Configuration function	3
II.III. Peripheral functions needed	9
II.IV. Example of a C Configuration function10)
Conclusion	3

Introduction

Nowadays, a lot of applications require wireless transmissions. Different technologies exist, and, according to the needs of the application (data rate, transmit range, transmit power, sensitivity, consumption...), some are more appropriate than others. You can find, for example, Wi-Fi, Infrared (IR), Bluetooth...

One of this wireless ways to transfer data is the Xbee modules. Using the Zigbee protocol, based on the IEEE 802.15.4 Standard for Wireless Personal Area Networks WPANs, these Xbee modules are low-power digital radios designed by DIGI.

The purpose of this application note is to explain how, from a classic micro controller, to configure the main options of these Xbee modules in order to create a simple wireless communication.

I. <u>The Xbee modules</u>

This part of the application note presents the module, and the general options to configure in order to established a basic wireless communication.

I.I. General Overview

The Xbee modules sold by DIGI (<u>www.digi.com</u>) are small radios operating within the ISM 2.4 GHz frequency band and meeting IEEE 802.15.4 standards.



Figure 1: Xbee Module (S1 &S1 pro)

Easily integrable with its reduced size, the Xbee modules offer the following properties (Xbee1 and Xbee1 pro):

- Transmit range: 30m up to 750m
- Transmit power: 1mW and 63mW
- Receiver sensitivity : -92dBm to -100dBm
- Transmit peak current (@3.3V): 45mA and 250mA
- Reception current (@3.3V): 50mA and 55mA
- Power down current: $<10\mu A$
- RF data rate 250 kbps
- Retries and Acknowledgements
- Source/Destination Addressing
- Unicast & Broadcast Communications
- Point-to-point, point-to-multipoint and peer-to-peer topologies supported
- Coordinator/End Device operations

You can find more Xbee specifications at <u>www.digi.com</u> .

For low data rate transmissions with a reasonable consumption, the Xbee seems to be a good compromise. Plus, it is rather simple to understand it way of functioning and to get it communicating.

I.II. Main Configuration's Options

There is no particular need to configure the Xbee module to communicate, but, according to the application developed and the integration environment of the module, you might have to change some parameters (like the serial data rate for example).

Here are the main parameters you might have to configure on a Xbee module:

- Serial communication parameters:
 - Data rate (1.2kbps to 250kbps)
 - Number of Stop bits
 - o Parity
 - Data length
- Sleep mode
 - Enabled/Disabled
 - o Cyclic/Pin
 - Time before sleep
- RF Power Level
- Get Serial Number (module address)
- Destination address
- Source address
- Channel used
- PAN ID
- MAC mode
- Packetization timeout
- Pull-up resistor options
- Pins configurations
- ...

Find the complete datasheet at: <u>http://ftp1.digi.com/support/documentation/90000982_J.pdf</u>

The "classic" way to configure a Xbee module is to use the Digi's X-CTU Software and a serial connection to a PC. Digi stocks RS-232 and USB boards to facilitate interfacing with a PC.

X-CTU software: http://ftp1.digi.com/support/driver/40002636_A.zip



Figure 2: Xbee development Kit

🖳 X-CTU			23
About			
PC Settings Range Test Terminal Modem Configu	uration		
Com Port Setup			
Select Com Port	Baud	250000	-
	Flow Control		-
	Data Bits	8	•
	Parity	NONE	•
	Stop Bits	1	•
	Te	st / Query	
Host Setup User Com Ports Network Interface			
API	nse Timeout		_ []
Enable API	Timeout 1000		
Use escape characters (ATAP = 2)	· ·		
AT command Setup			
Command Character (CC) + 2B			
Guard Time Before (BT) 1000			
─ Modem Flash Update			
□ No baud change			

Figure 3: X-CTU, Configuration and Communication software

With the X-CTU software, you're able to configure the serial connection (between the module and the PC) and you're able to communicate with the connected module (to configure some of its parameters for example) and with distant module via the connected module.

I.III. Configuration's Stream

In order to set up the module, the AT Command Mode is used. The AT Commands allow to turn the module into it configuration mode and to easily configure each parameter available.

From the X-CTU terminal on a PC or from a μ controller, the principle of this Command mode is the same, as described in the example below:



Figure 4: Example of Configuration's Stream

This first example shows the command stream to, in this case, configure the serial baud rate at 9600.

- ➤ To enter in the command mode, the characters "+++" must be sent to the module with a Carriage Return (CR).
- Once these characters have been sent, a guard time (GT, default 1s), while no characters should be sent, must be respect.
- The module answers to a correct command by sending back the word "OK" with a Carriage Return.
- ➤ In the previous example, the baud rate command is used: "BD" => "ATBD" with the parameter 03 which refers to the defined value of 9600 bauds (page 33/68 datasheet).
- The configuration is then saved to non-volatile memory using the command "ATWR" (Write) so that configuration persists through subsequent power-up or reset.
- > Finally, to exit the Command Mode, the characters "ATCN" are sent to the module.

The AT commands available must always be prefixed by the characters "AT" and formatted as defined:



Figure 5: Syntax of AT commands

The various AT commands available are described in the datasheet: <u>http://ftp1.digi.com/support/documentation/90000982_J.pdf</u>

NOTE: Failure to enter AT Command Mode is most commonly due to baud rate mismatch. To avoid this problem, ensure that the 'Baud' setting on the "PC Settings" tab or on the μ Controller's UART matches with the interface data rate of the RF module. By default, the BD parameter = 3 (9600 bps).

ΙΙ. <u>Xbee configuration from a μController</u>

According to the application developed, and particularly when the Xbee modules are to be embedded into autonomous systems, the module's configuration cannot always be proceeded manually by connecting each module on a development kit and by configuring the parameters from the X-CTU software.

In that case, one solution is to program the configuration of the module directly into the micro controller used to transmit the wireless data.

II.I. Connection Xbee ⇔ µController

Of course, a such micro controller must have a serial interface to be connected to the module. The minimum PINs required for the serial communication are DIN (Data In) and DOUT (Data Out). Obviously, the Xbee module needs to be alimented (3.3V) with the PINs VDD and GND.



Figure 6: Basic connection µController-Xbee

As shown in the Figure 6, the serial communication is asynchronous (no clock's signal shared). This means that the serial interface should be an UART (Universal Asynchronous Receiver Transmitter).

The pin UART_TX of the μ controller should be connected to the pin DI and the UART_RX should be connected to the Xbee's pin DOUT.

Considering that the Xbee module, now connected to the μ controller, is supposed to be in it default state with it start parameters, the configuration program can be written.

II.II. Designing a Configuration function

With the X-CTU software and it graphical interface, the configuration is pretty easy, the commands are sent in the terminal and the module's response are posted and shown in this same terminal.

But, from a μ controller, the configuration must be operated blindly considering that we have no screen to print the data exchanged with the module. This is why one solution consists in developing a configuration function as a Finite State Machine (FSM).

This FSM should first enter in the command mode, send the various commands successively, save the configuration to the non-volatile memory and finally, quit the command mode. If any step can't be proceeded (Fail), the FSM goes directly to the final state and quits the command mode. This first description is very basic, we'll see later that, in order to facilitate the debug of the program, the FSM should in fact be more complete. For example, a failure at one specific state of the FSM should be reported with the number of the missed state.



Figure 7: Principle of the FSM Configuration Function

With this principle, the configuration is realized at the function's call and is completely autonomous. The user just has to schedule once the configuration, prepare the command lines to send, choose the right parameters and the order of the commands.

II.III. Peripheral functions needed

To use a such configuration function, we have to be able to communicate properly with the Xbee module. That means we need peripherals functions:

> Configure and Initialize the UART

- Initial Serial baud rate : 9600
- Data length : 8 bits
- Bit Stop : 1 bit
- Parity : None
- UART FIFO : enabled and reset
- Enable UART transmit and reception
- Pin Configuration
 - Port used
 - Pins used
 - Pin's Function : UART (Tx & Rx)
 - Pull-up resistor...

Send a buffer (command) to the Xbee

- o UART's port used
- Pointer on the buffer to send
- Size of the buffer
- Transmit mode

Receive a buffer (response) from the Xbee

- UART's port used
- Pointer on the reception buffer
- Size expected
- Reception mode (blocking/ none blocking)

> Wait a defined time (to avoid guard time violations)

- Use a Timer which generates an Interruption each 1ms (for example) and increments a counter variable (SysTickCnt++)
- Create a delay function
 - Void delay (int tempo) //wait 'tempo' ms

II.IV. Example of a C Configuration function

This part presents an example of configuration function developed and used in a specific application which requires to change the serial data baud to 250kbps and to enable the sleep mode on Pin request.

- > First, the UART is initialized with 9600 bps, 8bits data length, 1 bit stop, no parity.
- > Then the configuration function (Xbee_config) is called.
- ➢ Finally, the return code of the function is analyzed to determine if the process succeeded or failed.
 - \circ If the process succeeded, the UART configuration is changed (baud rate 250kbps) to keep the same serial configuration as the Xbee module (now at 250kbps).
 - If the process failed, an error code is generated and signaled to the user.

At this moment, and if the configuration succeeded, the module is configured and can be used to communicate with another wireless system.

The configuration FSM used in this application is pretty simple:



Figure 8: Configuration FSM used in the example application

The C source of this configuration function is the following one:

```
@brief Xbee module configuration
  @param[in] none
  @return int : (1:succes) (0:fail)
4
*
  @author : BAPTISTE SOULIER
                                            int config_Xbee(void)
£
int etape_config=0;
while(1)
{
   switch (etape_config)
   {
     case 0 : //Initial step : Entering commande mode
      Delay(1500);
      UART_Send(TEST_UART, commande_ON, sizeof(commande_ON), BLOCKING);
      Delay (1500); //Ensure no Guard Time Violation
      UART_Receive(TEST_UART, commande_RESP, 3, NONE_BLOCKING);
if (commande_RESP[0]==commande_OK[0] && commande_RESP[1]==commande_OK[1] && commande_RESP[2]==commande_OK[2])
      £
       etape_config = 1;
      }
      else
      {
       etape_config = 6;
      }
     break;
     case 1
             : //Serial Baud Rate Configuration
      UART_Send(TEST_UART, commande_BD, sizeof(commande_BD), BLOCKING);
      UART_Receive(TEST_UART, commande_RESP, 3, BLOCKING);
if (commande_RESP[0]==commande_OK[0] && commande_RESP[1]==commande_OK[1] && commande_RESP[2]==commande_OK[2])
      -{
       etape config = 2;
      }
      else
      £
       etape_config = 6;
      }
     break;
      Coase 2 : //Sleep Mode Configuration
UART_Send(TEST_UART, commande_SM, sizeof(commande_SM), BLOCKING);
     case 2
      UART_Receive(TEST_UART, commande_RESP, 3, BLOCKING);
if (commande_RESP[0]==commande_OK[0] && commande_RESP[1]==commande_OK[1] && commande_RESP[2]==commande_OK[2])
      {
       etape_config = 3;
      }
      else
      £
       etape_config = 6;
      }
     break;
     case 3 : //Saving full configuration
      UART_Send(TEST_UART, commande_WR, sizeof(commande_WR), BLOCKING);
UART_Receive(TEST_UART, commande_RESP, 3, BLOCKING);
if (commande_RESP[0]==commande_OK[0] && commande_RESP[1]==commande_OK[1] && commande_RESP[2]==commande_OK[2])
      {
       etape config = 4;
      }
      else
      -
       etape_config = 6;
      }
     break;
    Gase 4 : //exit commande mode
UART_Send(TEST_UART, commande_CN, sizeof(commande_CN), BLOCKING);
UART_Receive(TEST_UART, commande_RESP, 3, BLOCKING);
if (commande_RESP[0]==commande_OK[0] && commande_RESP[1]==commande_OK[1] && commande_RESP[2]==commande_OK[2])
      {
       etape_config = 5;
      }
      else
      £
       etape_config = 6;
      }
     break;
    case 5 : return 1; // Full configuration completed => return success
case 6 : return 0; //configuration failed => return fail
     default : return 0;
     }
  }
```

Figure 9: C source code of the example function

The prototypes of the sub functions used are:

```
int config_Xbee (void);
void Delay (unsigned long tick);
uint32_t UART_Send(UART_TypeDef *UARTx, uint8_t *txbuf,uint32_t buflen,TRANSFER_BLOCK_Type flag);
uint32_t UART_Receive(UART_TypeDef *UARTx, uint8_t *rxbuf,uint32_t buflen,TRANSFER_BLOCK_Type
flag);
```

Figure 10: Function's prototypes

The data used in this function are:

```
//uint8_t = unsigned char
uint8_t commande_ON[]={'+','+','+'}; //openning commande mode
uint8_t commande_BD[]={'A','T','B','D','3','D','0','9','0','\r'}; //250kbps serial interface
uint8_t commande_SM[]={'A','T','S','M','1','\r'}; //Sleep Mode (0:disable) (1:pin controled)
uint8_t commande_WR[]={'A','T','W','R','\r'}; //write configuration (saving)
uint8_t commande_CN[]={'A','T','C','N','\r'}; // exit commande mode
uint8_t commande_OK[]={'O','K','\r'}; // !\ Input : module response if commande well understood
uint8_t commande_RESP[3]; // buffer used to record the answer sent by the module to our commandes and used to
compare with commande_OK
```

Figure 11: Data declarations

The function fills its role.

This example illustrates a simple way to program the Xbee module configuration from a µcontroller.

Conclusion

Using a Xbee module in an embedded application is a pretty simple way to manage a wireless communication.

The most important step in the development of a such application is the configuration: the μ controller configuration, but also the Xbee module configuration.

To configure a Xbee module for an integrated utilization, it's hardly possible to use the development tools like the development kit and the X-CTU software. Then, it's not a big problem, now you've seen, in this application note, how to manage an Xbee configuration from a μ controller.

Keep in mind that the configuration's way proposed in this document is only a solution among others and that the best solution will still be the one that you understand best. The solution proposed here is also very basic and according to your needs deserves to be improved.

Contact: <u>baptiste.soulier@gmail.com</u>

Sources: <u>www.digi.com</u> Complete datasheet: <u>http://ftp1.digi.com/support/documentation/90000982_J.pdf</u>