

Home automation using Wireless Sensor Network

Sommario

Obiettivi
Scelte Progettuali
Progettazione: Hardware e Software
Sviluppi Futuri
Documentazione

Obiettivi

Scopo di questo progetto consiste nel configurare una rete zigbee, in particolare modo sfruttando una soluzione compatibile con Xbee ZB Series 2 (con Wired Whip Antenna), in modo da soddisfare le seguenti necessità:

- economicità;
- consumo energetico;
- out-of-the-box;
- non invasività;
- espandibilità;
- interfacciabilità;
- open-source;

realizzando così una condizione realistica della gestione di un impianto di home automation, cosa che non è possibile ottenere con l'aggiunta di un microcontrollore per ogni nodo della rete.

L'home automation è quel settore dell'IT (Information Technology) che si occupa di mettere a disposizione la tecnologia per migliorare la qualità di tutte quelle operazioni che compiamo tutti i giorni nell'ambito casalingo, "automatizzandole"; con particolare attenzione rivolta sulla gestione automatizzata di scenari di luci e interruttori.



Scelte Progettuali

Il progetto è stato articolato in quattro fasi:

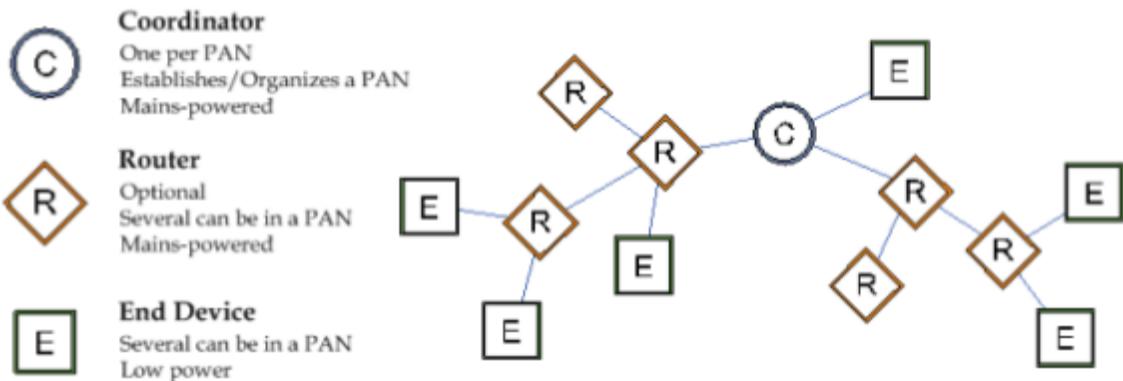
1. scelta dell'hardware;
2. sviluppo della piattaforma hardware dei nodi e del coordinatore;
3. sviluppo della piattaforma software e sistema di gestione;
4. testing finale.

Progettazione WSN

La tecnologia delle wireless sensor networks (WSN) consente di creare infrastrutture pervasive di monitoraggio e controllo senza la necessità di cablaggi, ottenendo di conseguenza un'elevata flessibilità di installazione e un significativo abbattimento dei costi di realizzazione e manutenzione. Uno dei vari campi cui le WSN trovano applicazione è l'ambito della home automation.

I nodi della rete possono essere di tre tipi:

- Coordinator: è il dispositivo che crea e configura la rete, cercando un canale radio adatto e definendo tra i parametri operativi il PAN ID, cioè il numero a 16bit usato dai membri della rete per riferirsi ad essa. Una volta avviata la rete si mette in modalità coordinatore, permettendo o meno a router ed end devices di fare una join alla propria rete. Tra le altre funzioni, agisce come deposito per le chiavi di sicurezza "trust center", ed occasionalmente effettua attività di routing. Lo ZigBee Coordinator è unico per ogni rete, ne costituisce la radice e può fare da ponte tra più reti diverse. È chiaramente un Full Function Device, e dato che non può andare in modalità sleep per ottemperare ai suoi servizi, deve essere alimentato da rete elettrica.
- Router: permette l'inoltro dei messaggi aumentando così le distanze di copertura della rete e ha il potere di accettare o meno richieste di join alla rete da parte di altri devices. Ovviamente prima si dovrà unire a una rete già esistente, chiedendo l'autorizzazione al coordinatore o a un altro router. L'inoltro dei messaggi avviene attraverso il mantenimento di tabelle di routing dinamiche e sempre aggiornate, in cui il router può memorizzare fino a 20 percorsi diversi. Per ogni rete può esserci più di un router, che però essendo un Full Function Device deve essere alimentato da rete elettrica, un router in sleep mode serve a poco.
- End Device: è un Reduced Function Device, perché le sue funzionalità si riducono al dialogare col coordinatore o col router (non può trasmettere o ricevere dati da altri End Device). Richiede poca memoria, ha bassi consumi va a batterie e ha costi inferiori rispetto agli altri due tipi di dispositivi. Al contrario del Coordinatore e del Router, la loro presenza in una rete ZigBee non è essenziale si possono benissimo realizzare reti con solo un coordinatore e uno o più router.

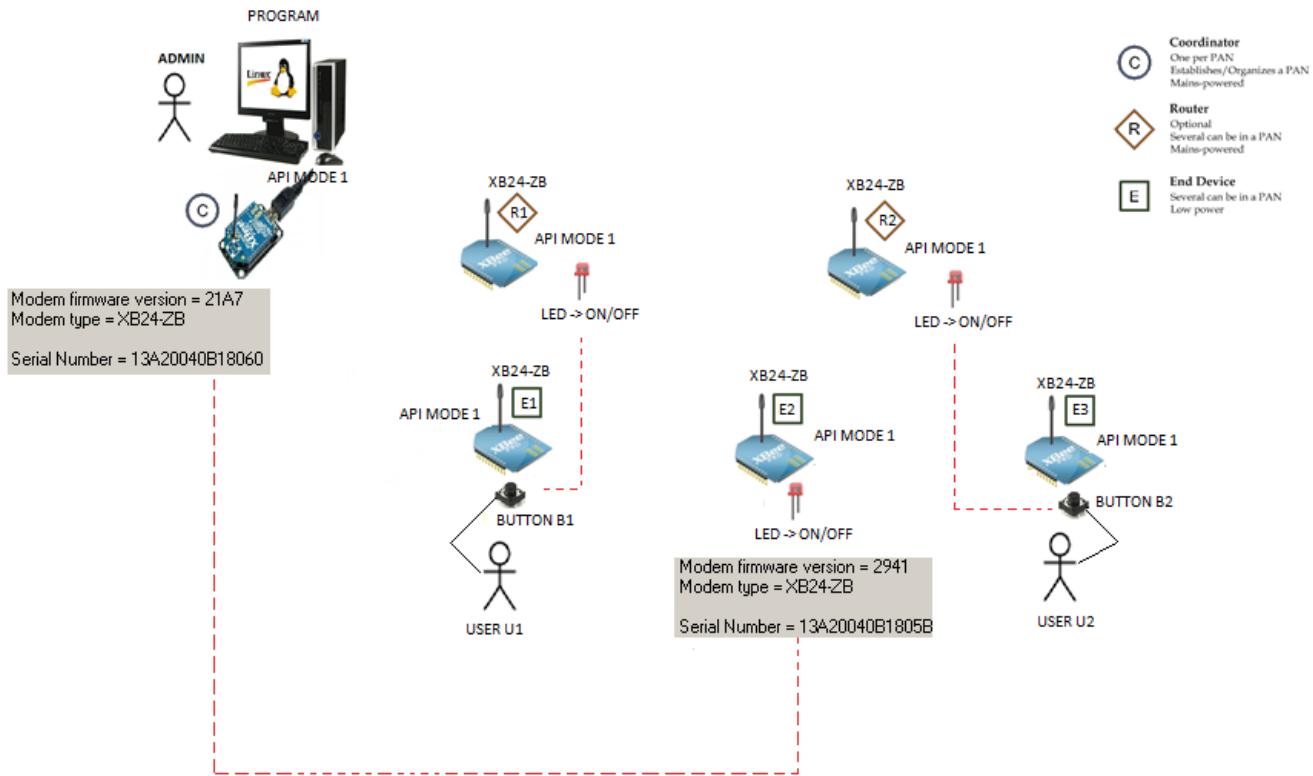


Scelta dell'Hardware

Da un'analisi di mercato valutando pro e contro abbiamo optato per i moduli della [Digi International](#) modello [XB24-Z7WIT-004](#):

Platform	XBee® ZB	XBee-PRO® ZB	Programmable XBee-PRO® ZB
Performance			
RF Data Rate		250 Kbps	
Indoor/Urban Range	133 ft (40 m)	300 ft (90 m)	
Outdoor/RF Line-of-Sight Range	400 ft (120 m)	2 miles (3200 m) / Int'l 5000 ft (1500 m)	
Transmit Power	1.25 mW (+1 dBm) / 2 mW (+3 dBm) boost mode	63 mW (+18 dBm) / Int'l 10 mW (+10 dBm)	
Receiver Sensitivity (1% PER)	-96 dBm in boost mode		-102 dBm
Features			
Adjustable Power		Yes	
I/O Interface	3.3V CMOS UART, ADC, DIO		3.3V CMOS UART, SPI, I2C, PWM, DIO, ADC
Configuration Method		API or AT commands, local or over-the-air	
Frequency Band		2.4 GHz	
Interference Immunity		DSSS (Direct Sequence Spread Spectrum)	
Serial Data Rate		1200 bps - 1 Mbps	
ADC Inputs		(4) 10-bit ADC inputs	
Digital I/O		10	
Antenna Options	Chip, Wire Whip, U.FL, RPSMA	PCB Embedded Antenna, Wire Whip, U.FL, RPSMA	
Operating Temperature		-40° C to +85° C, 0-95% humidity non-condensing	
Programmability			
Memory	N/A		32 KB Flash / 2 KB RAM
CPU/Clock Speed	N/A		HCS08 / Up to 50.33 MHz
Networking & Security			
Encryption		128-bit AES	
Reliable Packet Delivery		Retries/Acknowledgments	
IDs and Channels	PAN ID, 64-bit IEEE MAC, 16 channels	PAN ID, 64-bit IEEE MAC, 15 channels	
Power Requirements			
Supply Voltage	2.1 - 3.6VDC		2.7 - 3.6VDC
Transmit Current	35 mA / 45 mA boost mode @ 3.3VDC	205 mA	220 mA
Receive Current	38 mA / 40 mA boost mode @ 3.3VDC	47 mA	62 mA
Power-Down Current	<1 uA @ 25° C	3.5 uA @ 25° C	4 uA @ 25° C
Regulatory Approvals			
FCC, IC (North America)		Yes	
ETSI (Europe)		Yes	
C-TICK (Australia)		Yes	
TELEC (Japan)	Yes		Yes (int'l unit only)

In generale il progetto può essere schematizzato nel seguente modo:



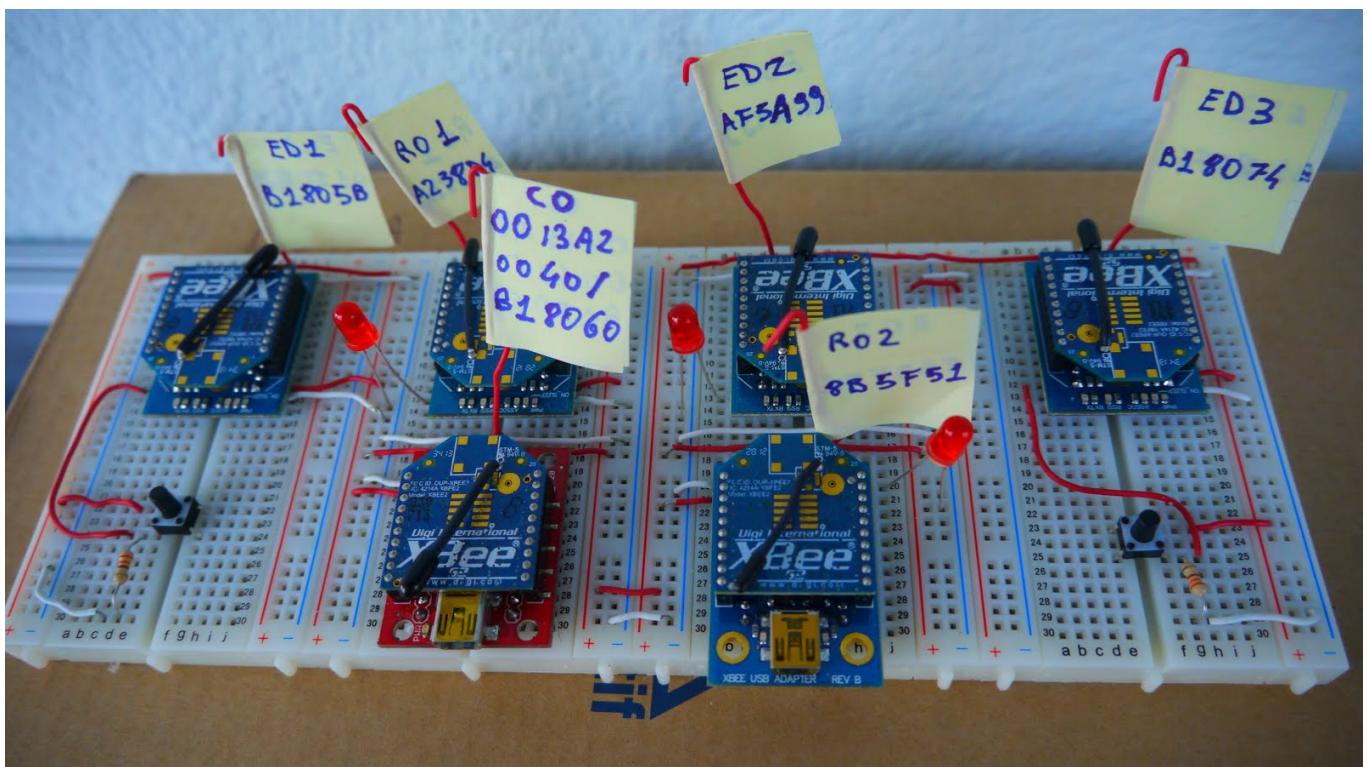
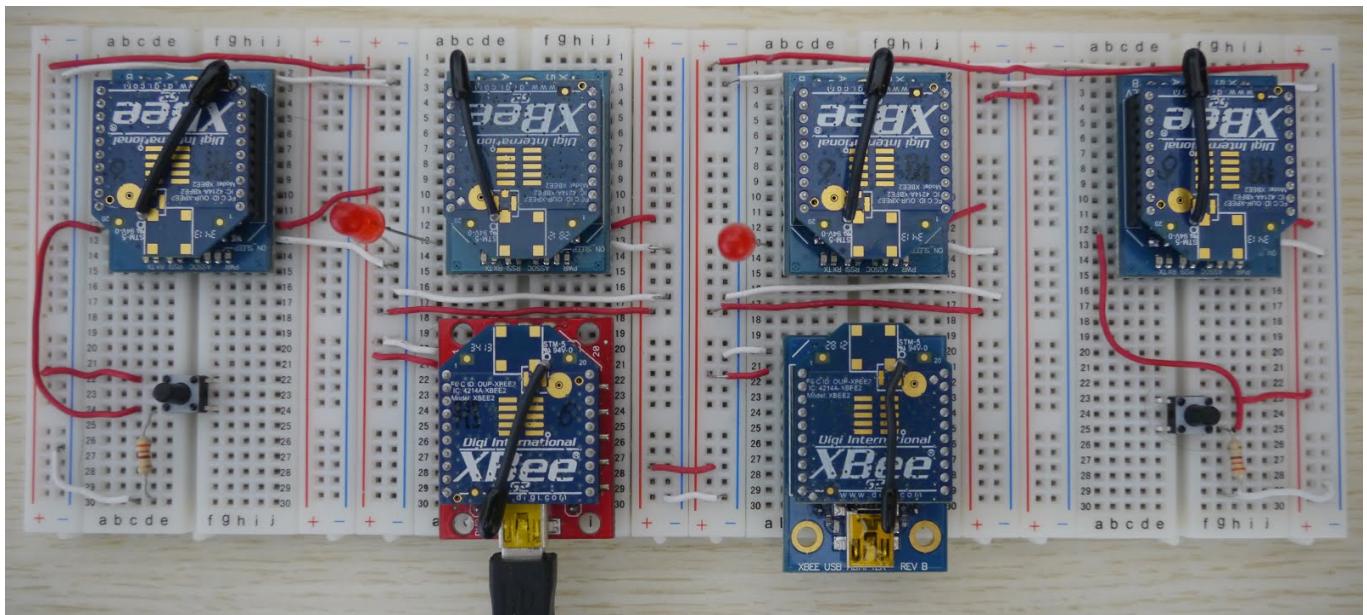
Hardware

In questa fase viene la rete WSN da noi progettata viene assemblata elettronicamente, configurata e testata tramite XCTU un software della MaxStream.

Per la realizzazione di tale rete è stato utilizzato il seguente hardware:

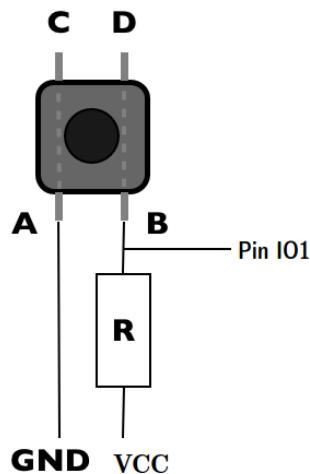
- n.6 XBee XB24-Z7WIT-004 2mW Wire Whip Antenna - from Digi Series 2 (ZB)
- n.1 XBee USB Adapter Board
- n.5 XBee Adapter Board
- n.2 Pulsanti
- n.2 Resistenze da 330 ohm
- n.3 Led
- n.1 breadboard
- n.1 Cavo Usb-mini Usb
- n.1. Pc

Da un punto di vista elettronico, per problemi legati all'alimentazione i vari nodi sono stati cablati nel modo seguente, ottenendo così l'alimentazione tutti dal coordinatore:

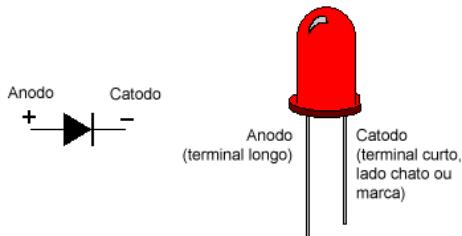


In linea di massima premendo uno dei bottoni collegati rispettivamente nell'End Device 1 e 3, a secondo lo stato che si intende inviare al bottone cambia lo stato di 1 o dei 3 led.

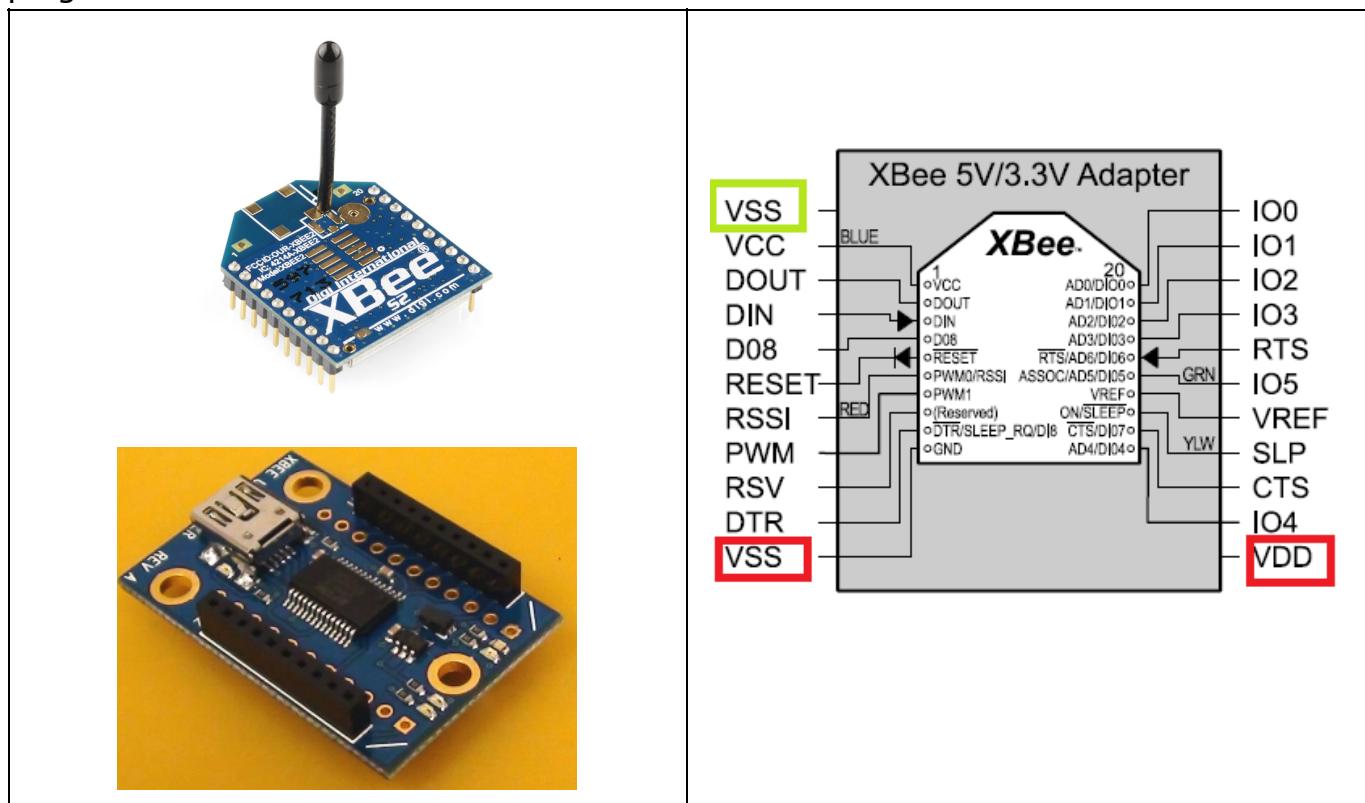
Nei pulsanti per una scelta mnemonica sono stati così collegati tutti nel Pin IO1 dell'End Device 1 e 3 :



Invece, il catodo dei led per le stessa dinamica del pulsante sono stati collegati nel Pin IO0 del Router 1, End Device 2, Router 2 e l'Anodo a +VCC.



I nodi XBee sono formati dal XBee Adapter Board e dal modulo XBee ZB, a differenza del coordinatore che è presente XBee USB Adapter Board, che serve per collegarlo al pc e programmare i vari moduli.



Per effettuare il cablaggio occorre conoscere il data sheet dei piedini del modulo XBee ZB:

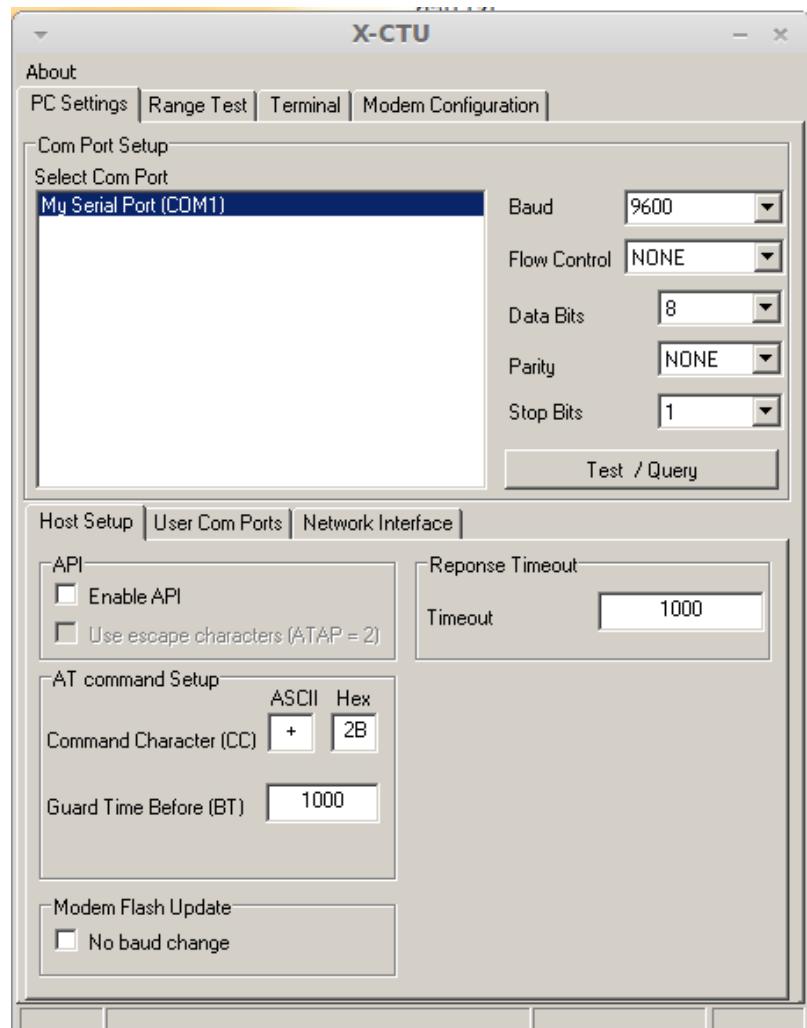
Pin Assignments for the XBee/XBee-PRO Modules

(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / CONFIG	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital I/O 12
5	RESET	Both	Open-Collector with pull-up	Module Reset (reset pulse must be at least 200 ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / Digital IO
7	DIO11	Both	Input	Digital I/O 11
8	[reserved]	-	Disabled	Do not connect
9	DTR / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital I/O 4
12	CTS / DIO7	Both	Output	Clear-to-Send Flow Control or Digital I/O 7. CTS, if enabled, is an output.
13	ON / SLEEP	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	Input	-	Not used for EM250. Used for programmable secondary processor. For compatibility with other XBEE modules, we recommend connecting this pin voltage reference if Analog sampling is desired. Otherwise, connect to GND.
15	Associate / DIO5	Both	Output	Associated Indicator, Digital I/O 5
16	RTS / DIO6	Both	Input	Request-to-Send Flow Control, Digital I/O 6. RTS, if enabled, is an input.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Both	Disabled	Analog Input 0, Digital IO 0, or Commissioning Button

Effettuati i collegamenti bisogna configurare opportunamente i moduli del coordinator, router e end device sfruttando XBee USB Adapter Board.

Per fare ciò, si è fatto uso del software fornito da Maxstream, chiamato X-CTU. Purtroppo, utilizzando il Sistema Operativo GNU-Linux, il software non è multipiattaforma, perciò si è fatto ricorso a [Wine](#) (Wine Is Not an Emulator), tramite una guida presente nella documentazione, che non ha il compito di emulare nessuna architettura o sistema operativo, bensì fornisce al programma in esecuzione il supporto necessario per l'utilizzo di specifiche funzioni di Windows. L'interfaccia grafica di X-Ctu si compone essenzialmente di quattro schermate: PC Settings, Range Test, Terminal e Modem Configuration.



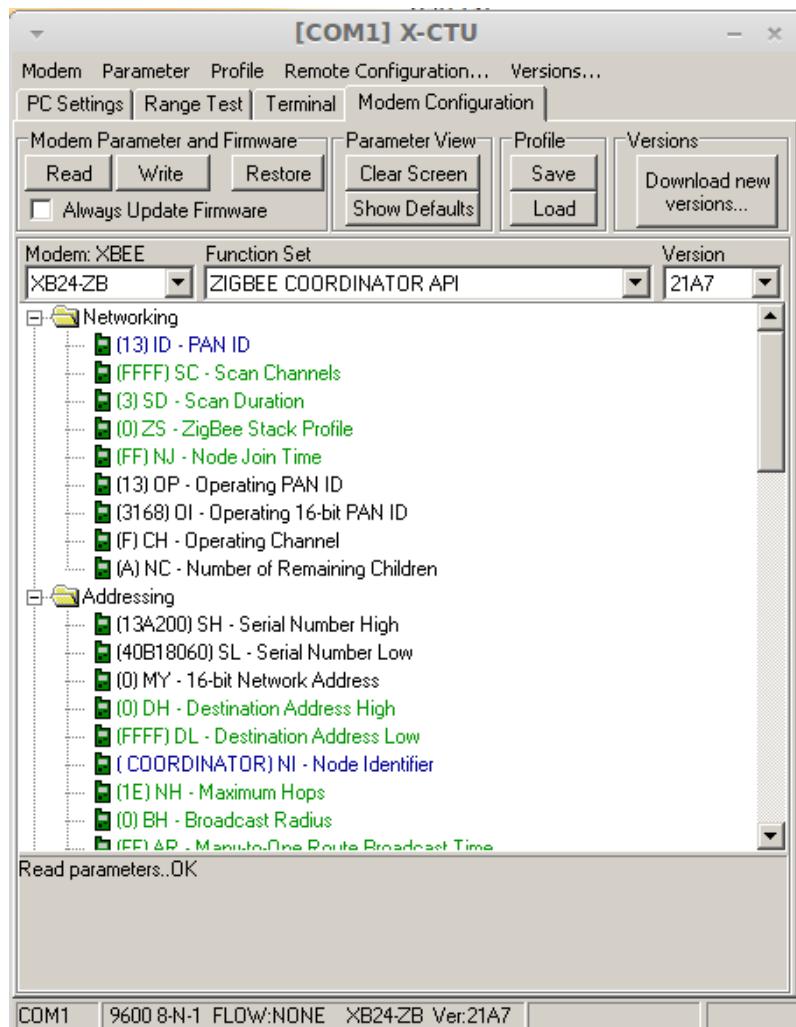
La schermata “PC Settings” consente di rendersi conto di quale sia la porta COM associata al modulo, di effettuare un test di comunicazione per verificarne il corretto funzionamento avendo cura di impostare correttamente il baud rate del modulo e il formato del pacchetto seriale di dati: per default il baud rate è di 9600, i bit di dati sono otto, senza bit di parità e con un singolo bit di stop. La schermata di “Range Test” è utile per un primo test sul corretto funzionamento dei moduli e, più in generale, per valutare la massima distanza possibile di comunicazione. Per effettuare il test, è opportuno che su uno dei due moduli si inserisca un componente hardware aggiuntivo, il “loopback adapter”, che fa sì che i dati ricevuti dal modulo su cui viene adattato vengano immediatamente ritrasmessi verso la sorgente di informazione. Il test consiste nell’inviare ripetutamente dati e permette di valutare tempi e correttezza nella risposta del modulo ricevente. La schermata “Terminal” è utile per la comunicazione di dati tra moduli, in quanto consente di tenere sotto controllo l’invio di dati tramite il comando “Assemble Packet” o semplicemente scrivendo nella finestra. È importante conoscere il valore del parametro RO proprio del modulo, che indica quanti caratteri in ingresso è necessario inviare al modulo prima che inizi la trasmissione. Infine, la schermata “Modem Configuration” fornisce una via alternativa alla “Command Mode” per la modifica dei parametri propri del nodo. Sarà sufficiente cliccare su “Read” per valutare tutti i parametri attuali, in verde vengono evidenziati quelli uguali ai valori di default per tale modulo. Dopo le eventuali modifiche, sarà necessario cliccare su “Write” per renderle permanenti. La funzione

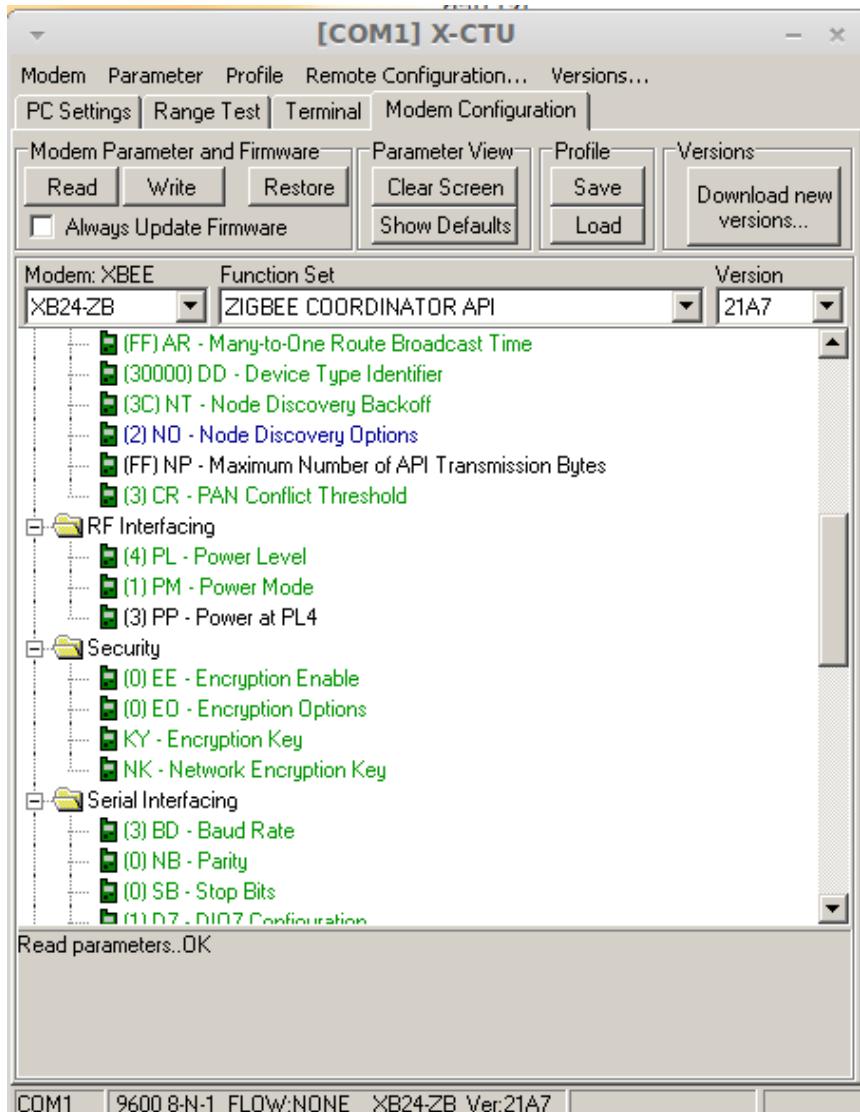
"Restore" è invece utile per riportare i valori di tutti i parametri a quelli di default. Inoltre in questa schermata è possibile uploadare i firmware che si desiderano.

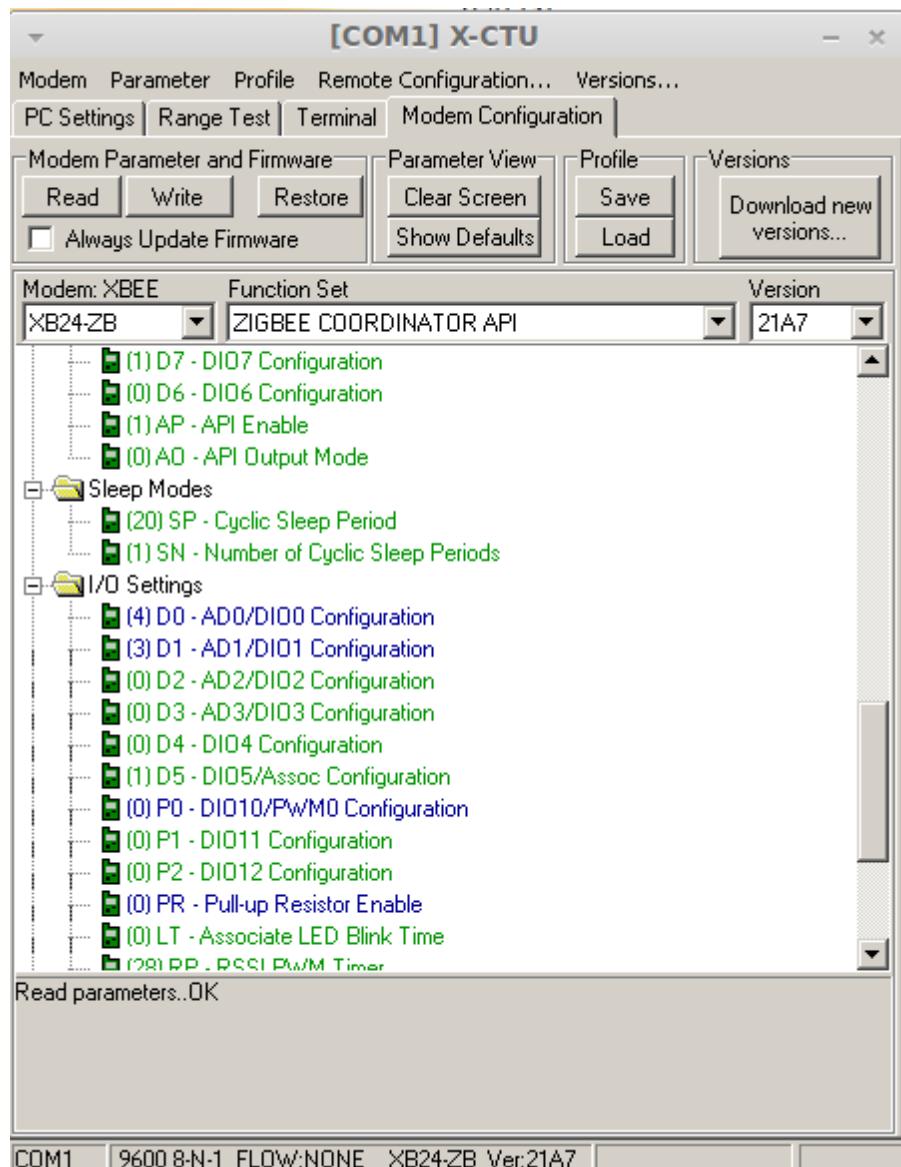
Configuriamo il Coordinator, settando i seguenti parametri, nell'XCTU:

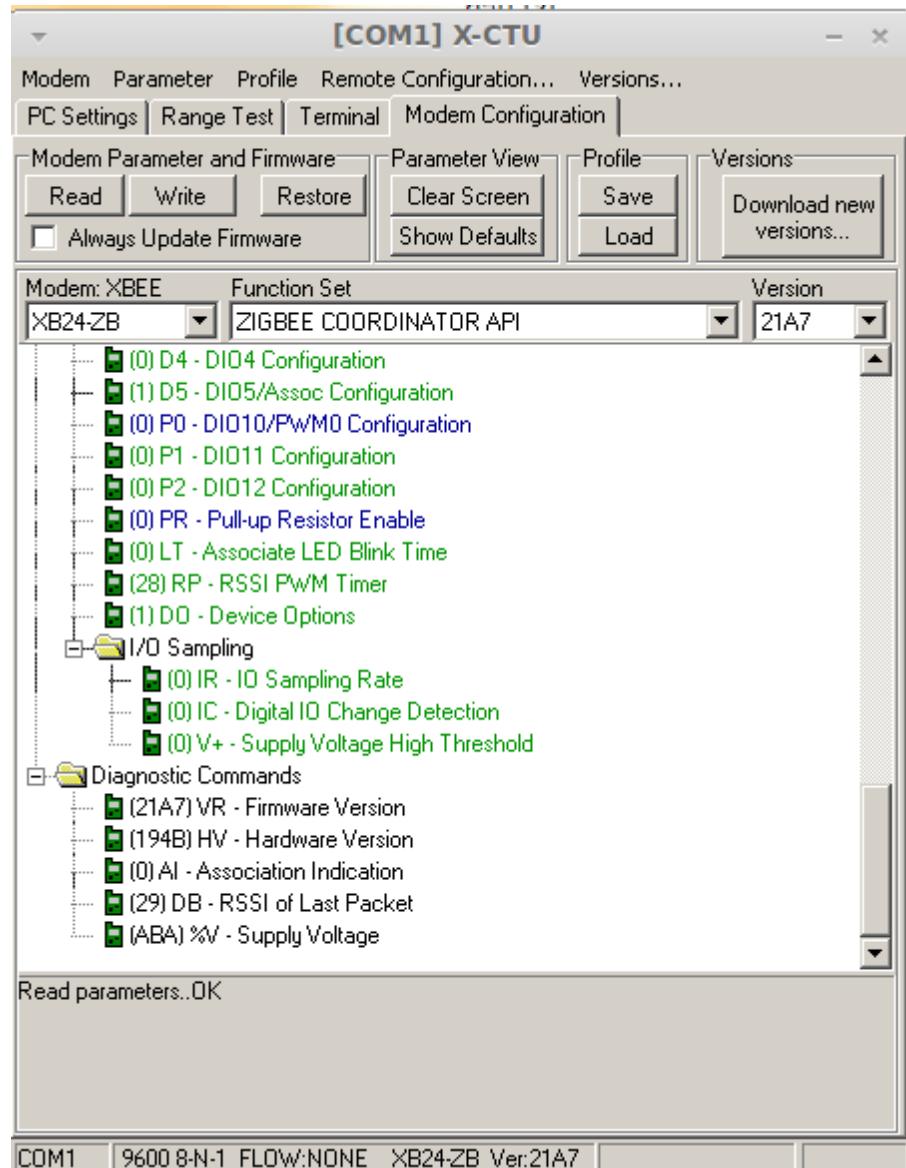
COORDINATOR API=1:

- Type: XB24-ZB version 21A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A20040B18060
- PAN ID: 13
- DH: 0
- DL: FFFF (indica broadcast)
- NI: COORDINATOR





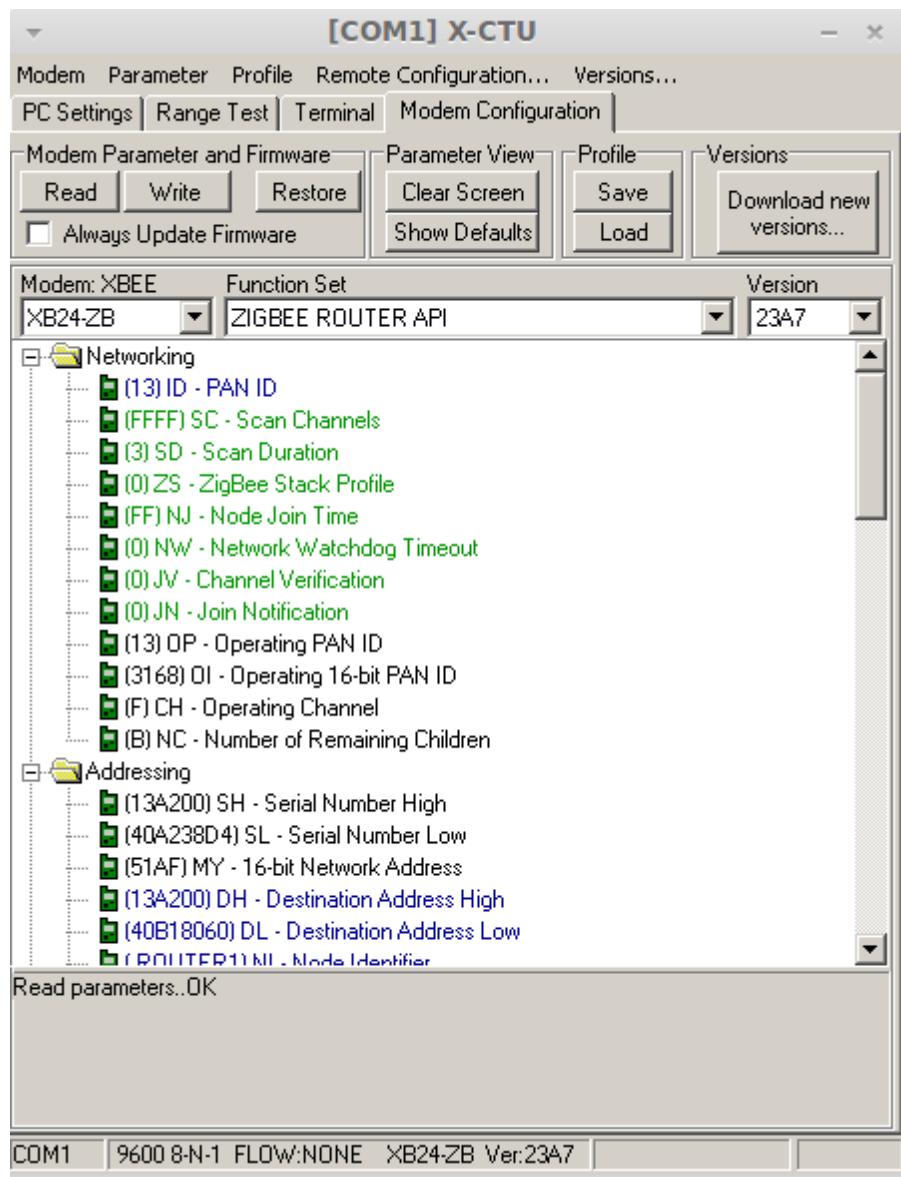


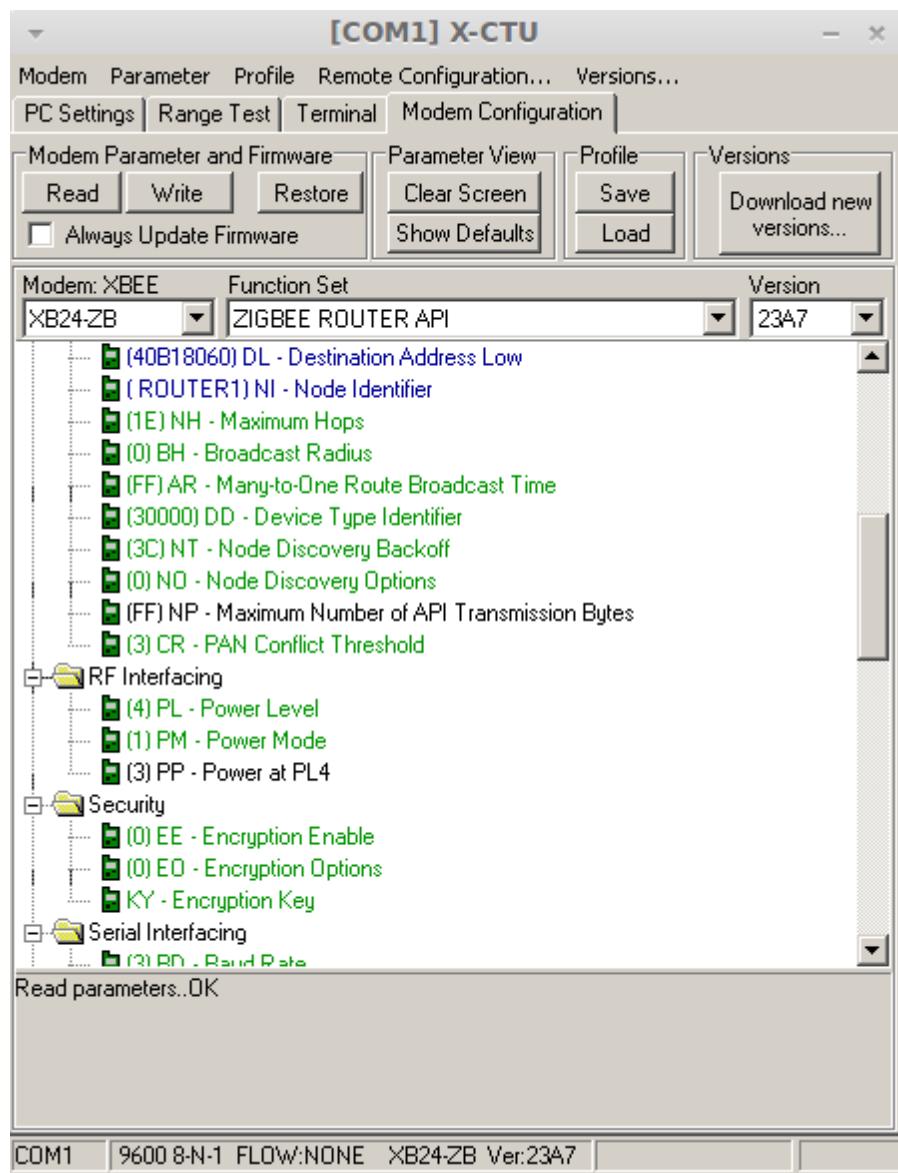


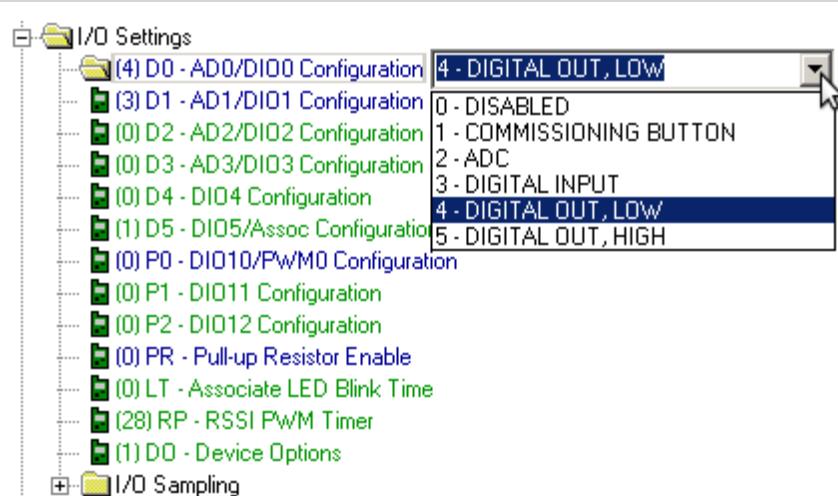
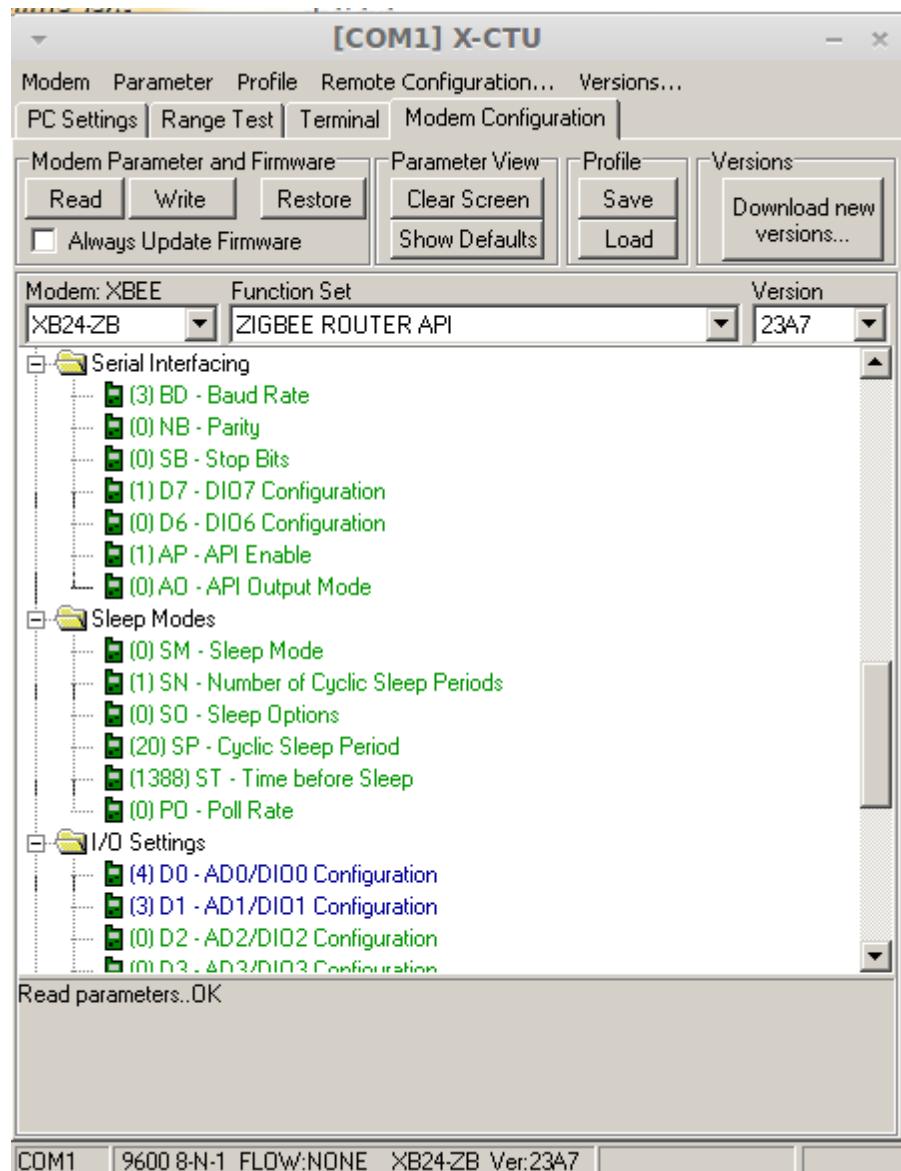
Configuriamo il Router 1 settando i seguenti parametri, nell'XCTU:

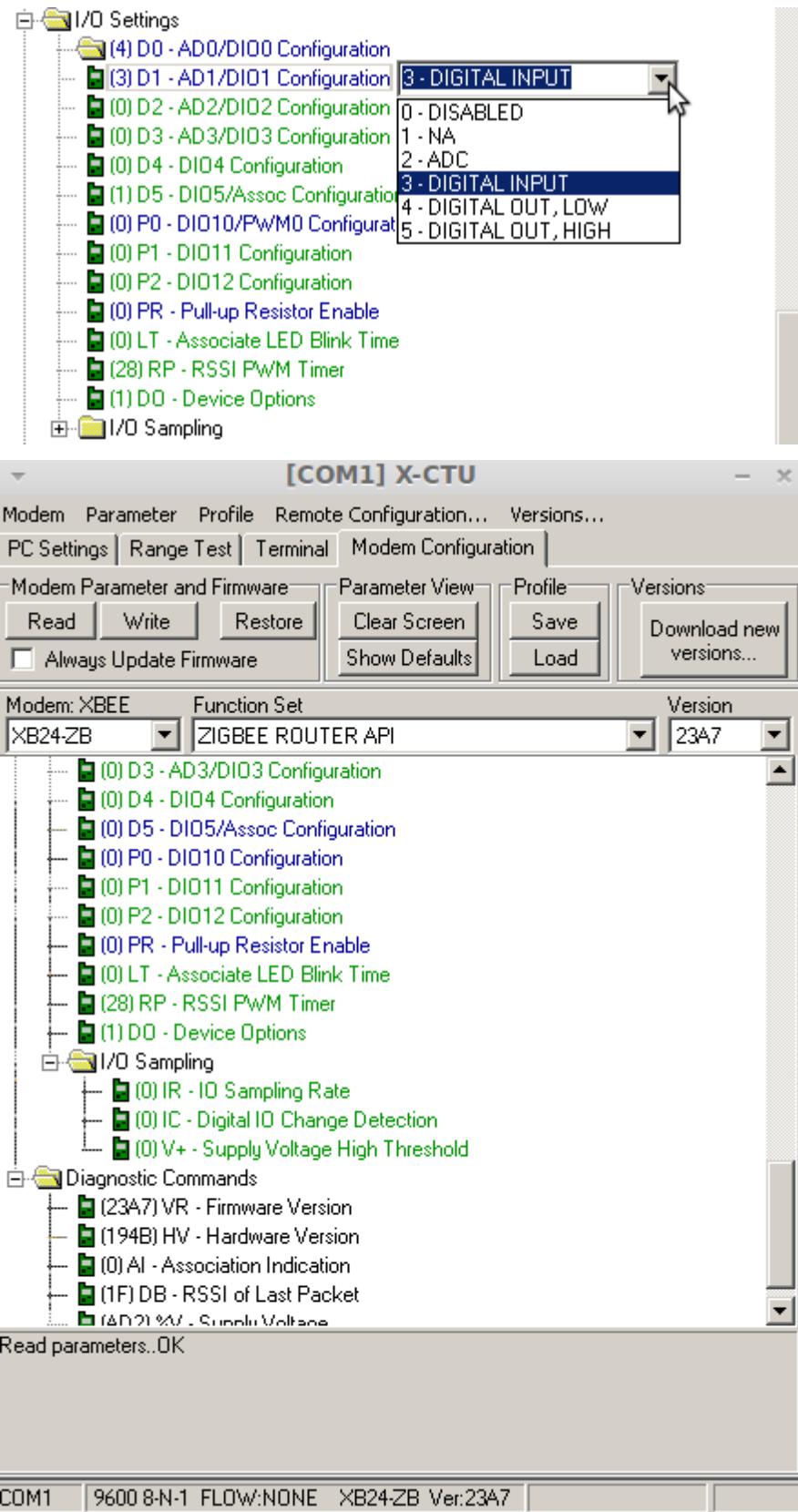
ZIGBEE ROUTER 1 API=1:

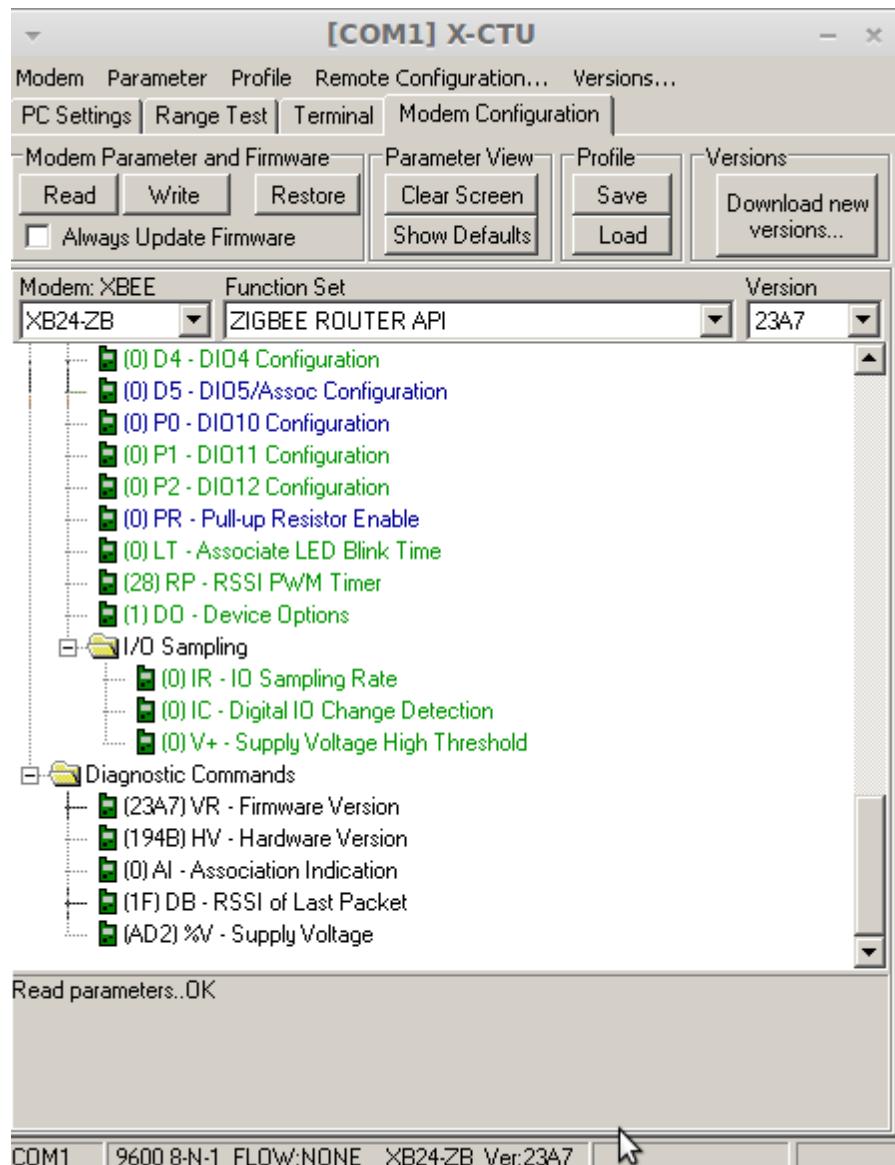
- Type: XB24-ZB version 23A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A20040A238D4
- PAN ID: 13
- DH: 0013A200 (del coordinator)
- DL: 40B18060 (del coordinator)
- NI: ROUTER1
- D0: 4 - DIGITAL OUT, LOW (per il led)
- D1: 3 - DIGITAL INPUT (per il pulsante)







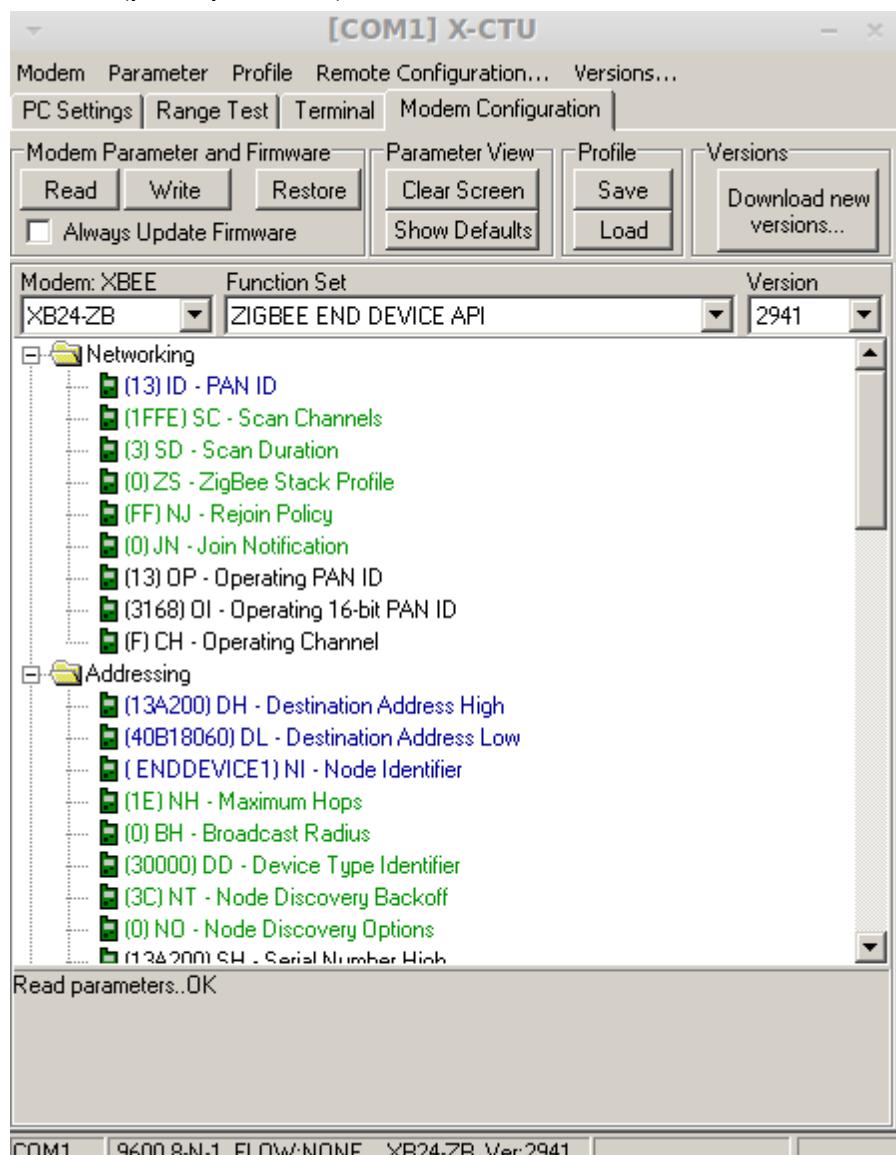


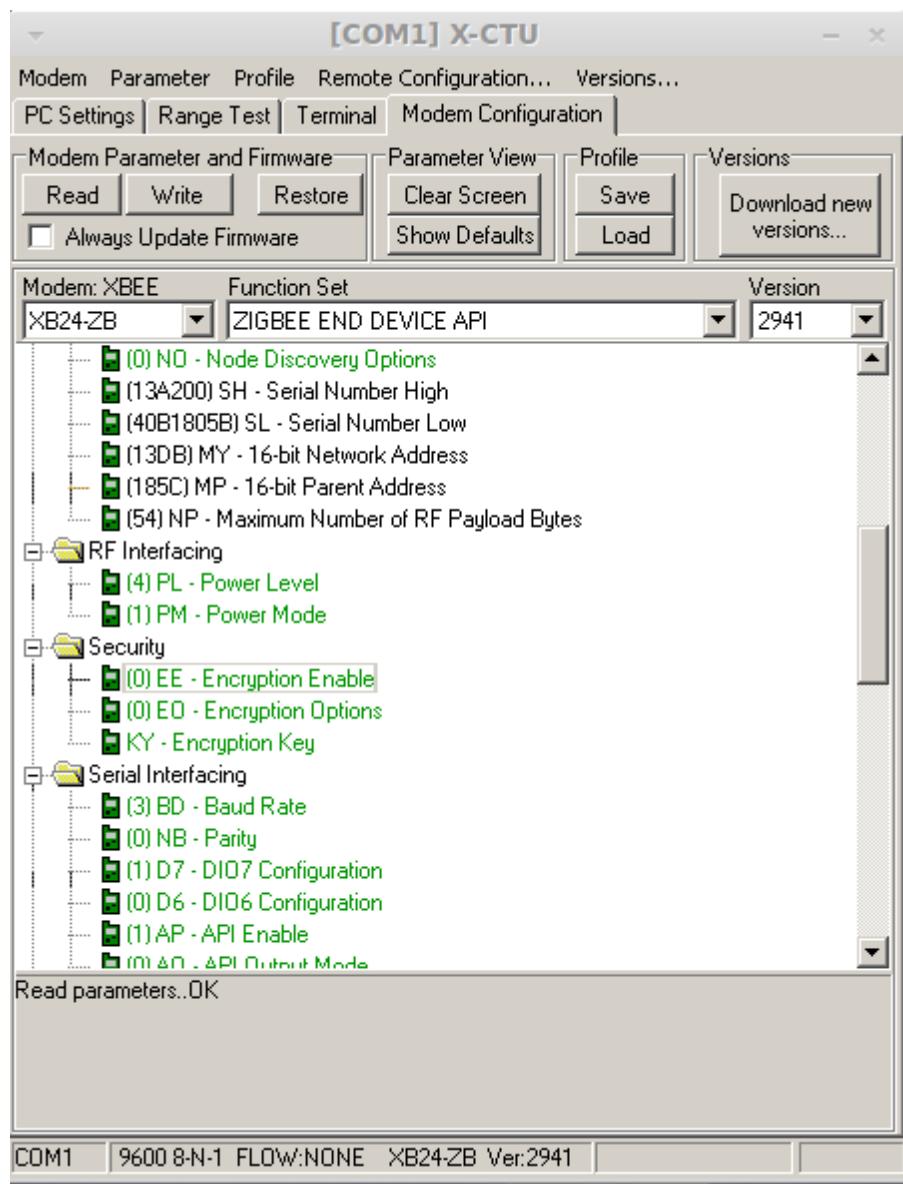


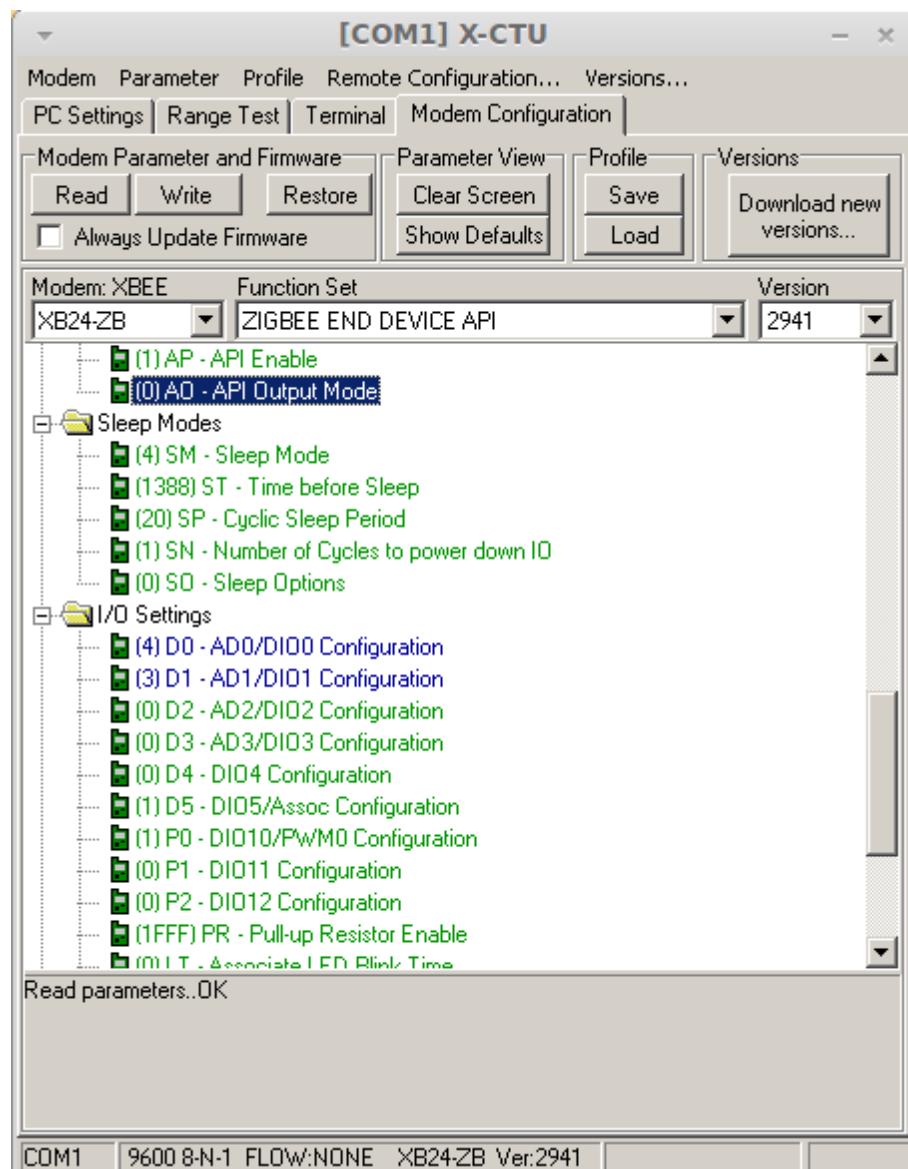
Configuriamo il End Device 1 settando i seguenti parametri, nell'XCTU:

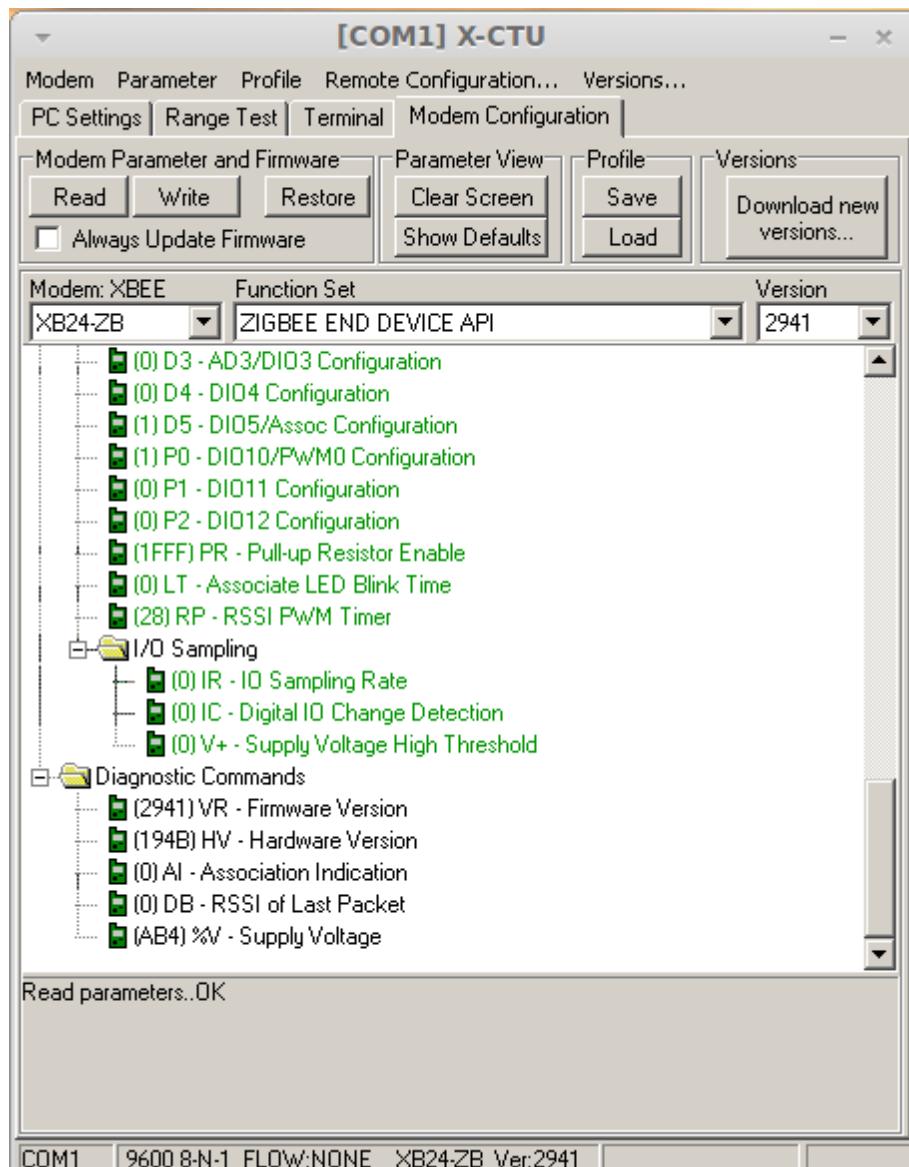
ZIGBEE END DEVICE 1 API=1:

- Type: XB24-ZB version 23A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A20040B1805B
- PAN ID: 13
- DH: 0013A200 (del coordinator)
- DL: 40B18060 (del coordinator)
- NI: ENDDEVICE1
- D0: 4 - DIGITAL OUT, LOW (per il led)
- D1: 3 - DIGITAL INPUT (per il pulsante)









Allo stesso modo si configurano tutti gli altri nodi, configuriamo l'Router 2, End Device 2 e 3 settando i seguenti parametri, nell'XCTU:

ZIGBEE ROUTER 2 API=1:

- Type: XB24-ZB version 23A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A200408B5F51
- PAN ID: 13
- DH: 0013A200 (del coordinator)
- DL: 40B18060 (del coordinator)
- NI: ROUTER2
- D0: 4 - DIGITAL OUT, LOW (per il led)
- D1: 3 - DIGITAL INPUT (per il pulsante)

ZIGBEE END DEVICE 2 API=1:

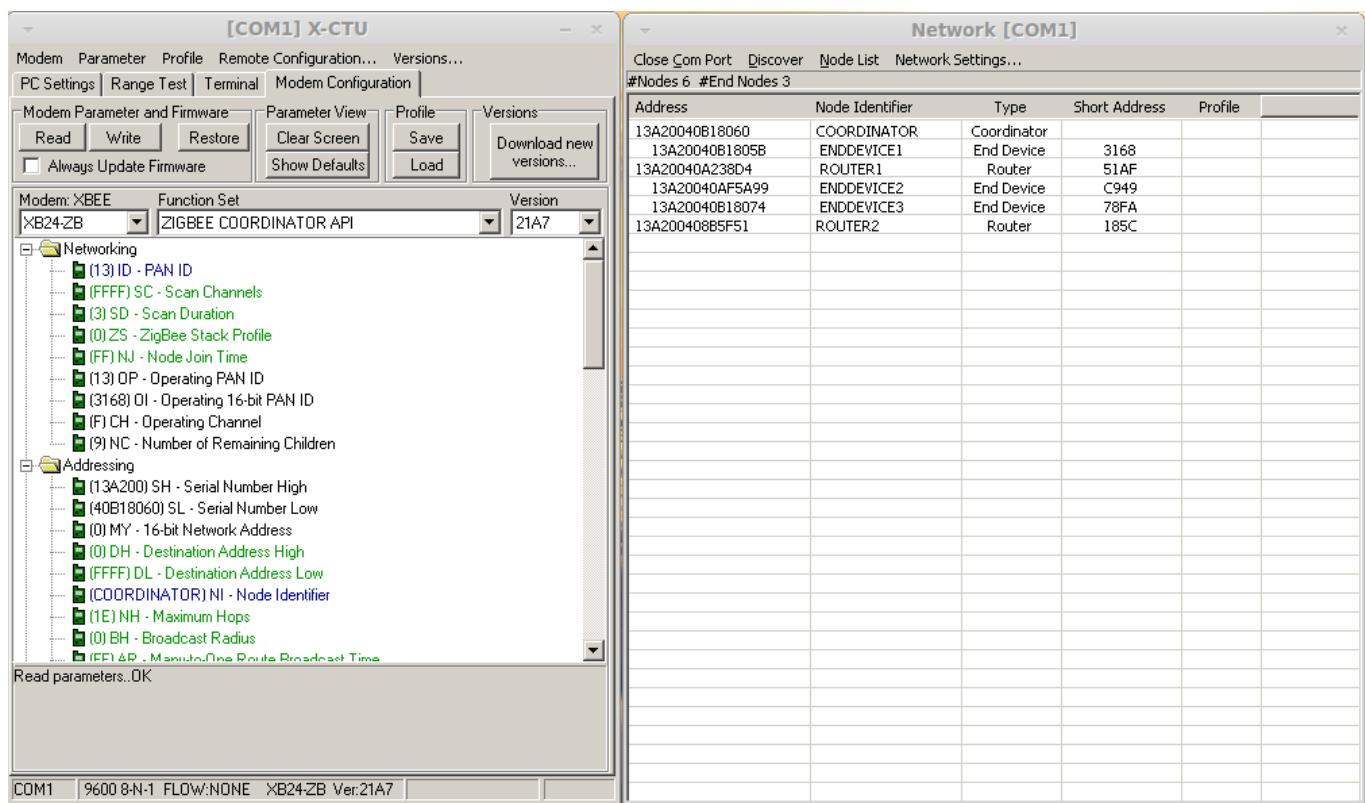
- Type: XB24-ZB version 23A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A20040AF5A99

- PAN ID: 13
- DH: 0013A200 (del coordinatore)
- DL: 40B18060 (del coordinatore)
- NI: ENDDEVICE2
- D0: 4 - DIGITAL OUT, LOW (per il led)
- D1: 3 - DIGITAL INPUT (per il pulsante)

ZIGBEE END DEVICE 3 API=1:

- Type: XB24-ZB version 29A7 (XB24-Z7WIT-004 revK)
- Serial Number/Address: 0013A20040B18074
- PAN ID: 13
- DH: 0013A200 (del coordinatore)
- DL: 40B18060 (del coordinatore)
- NI: ENDDEVICE3
- D0: 4 - DIGITAL OUT, LOW (per il led)
- D1: 3 - DIGITAL INPUT (per il pulsante)

Infine, con la funzione Remote Configuration dell'XCTU dal Coordinator è stato possibile testare la presenza di tutti i nodi presenti nella rete:



Software

In questa ultima fase è stato possibile creare il sistema di gestione software che andrà ad automatizzare la rete, con un minimo di interfaccia user friendly per l'utente .

Tale sistema è stato scritto in C per questioni prestazionali rispetto a un linguaggio ad alto livello, sfruttando il compilatore [GCC](#) (GNU Compiler Collection) gcc version 4.8.1 (Ubuntu/Linaro 4.8.1-10ubuntu9) e le librerie [Libxbee v3](#) sotto [GNU Lesser General Public License \(GPL\)](#) per interfacciarsi con i moduli XBee ZB di [Digi](#) (inizialmente, a posto di tale libreria si era pensato a leggere brutalmente dalla seriale, elaborando le frame in arrivo, ma da ulteriori ricerche per una questione di complessività e velocità di gestione si è preferito gestire il tutto con l'utilizzo di tale libreria).

Per verificare se è presente il compilatore gcc, da terminale digitiamo:

```
$ gcc --version
gcc (Ubuntu/Linaro 4.8.1-10ubuntu9) 4.8.1
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

altrimenti, installiamo da super user, digitando da terminale:

```
$ apt-get install gcc-4.8
```

Successivamente, scarichiamo la libreria con git, digitando da terminale:

```
$ git clone https://github.com/attie/libxbee3
```

Adesso nella nostra directory di lavoro sarà presente una cartella libxbee.libxbee-v3, dentro questa directory c'è tutta la libreria, con:

- la documentazione nella cartella:/libxbee.libxbee-v3/html ;
- i manuali dei moduli XBee ZB nella cartella:/libxbee.libxbee-v3/manual ;
- gli esempi nella cartella sample:/libxbee.libxbee-v3/sample ;

Inoltre è presente un file di configurazione importante se si vuole tenere traccia dei log del programma, il file/libxbee.libxbee-v3/config.mk .

Aprendo tale file con un qualsiasi editor, è possibile vedere che per default non tiene traccia dei log. Per abilitarli occorre decommentare tale righe e salvare il file:

```
OPTIONS+=          XBEE_LOG_NO_COLOR
OPTIONS+=          XBEE_LOG_LEVEL=100
OPTIONS+=          XBEE_LOG_RX
OPTIONS+=          XBEE_LOG_TX
```

Successivamente, affinchè i log siano attivi, occorre compilare la libreria, apriamo un terminale e all'interno della cartella ./libxbee.libxbee-v3 digitiamo:

```
make configure  
make all  
sudo make install
```

Per disabilitare i log commentiamo le seguenti righe nel file config.mk, compilando nuovamente la libreria:

```
#OPTIONS+= XBEE_LOG_NO_COLOR  
#OPTIONS+= XBEE_LOG_LEVEL=100  
#OPTIONS+= XBEE_LOG_RX  
#OPTIONS+= XBEE_LOG_TX
```

Per compilare il nostro programma in c da terminale digitiamo:

```
gcc -o six_xb24zb_automation six_xb24zb_automation.c -lxbee -pthread
```

Prima di eseguire il nostro programma verifichiamo da terminale, dove il sistema operativo carica il device della USB, visto che per default il programma in c legge i dati in /dev/ttyUSB0:

```
$ dmesg | tail  
[16193.966834] usb 1-1.2: Product: FT232R USB UART  
[16193.966838] usb 1-1.2: Manufacturer: FTDI  
[16193.966842] usb 1-1.2: SerialNumber: A702LCSO  
[16193.969870] ftdi_sio 1-1.2:1.0: FTDI USB Serial Device converter detected  
[16193.969997] usb 1-1.2: Detected FT232RL  
[16193.970009] usb 1-1.2: Number of endpoints 2  
[16193.970017] usb 1-1.2: Endpoint 1 MaxPacketSize 64  
[16193.970023] usb 1-1.2: Endpoint 2 MaxPacketSize 64  
[16193.970029] usb 1-1.2: Setting MaxPacketSize 64  
[16193.970860] usb 1-1.2: FTDI USB Serial Device converter now attached to ttyUSB0
```

Una volta compilato e se tutto andato a buon fine eseguiamo il nostro programma in c, digitando da terminale:

```
./six_xb24zb_automation
```

```
## PROGRAMMA: six_xb24zb_automation.c ##  
/*  
six_xb24zb_automation - a program C made during a project of technologies of  
control systems in University of Catania with six Digi's XBee wireless modules  
XB24-Z7WIT-004 revK running in API mode.  
This program creates a condition of a realistic management of a home automation  
system with buttons and lights.
```

Copyright (C) 2014 onwards Nicola Didomenico (nicola.didomenico@gmail.com)

Copyright (C) 2014 onwards Riccardo Merenda (riccardomerenda@gmail.com)

Contributors:

Attie Grande (attie@attie.co.uk)

Eng. Giuseppe Avon (giuseppe.avon@hibas.it)

Prof. Robert Faludi (robert.faludi@nyu.edu)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public Licens along with this program. If not, see <<http://www.gnu.org/licenses/>>.

*/

```
#include <stdio.h>
#include <stdlib.h>
#include <xbee.h>
#include <pthread.h>

#define XBEE_IO_IN_DIGITAL 0x03
#define XBEE_IO_OUT_LOW    0x04
#define XBEE_IO_OUT_HIGH   0x05
#define green "\033[01;32m"

pthread_mutex_t led_mutex;

struct xbee *xbee;
xbee_err ret;
int led1_status, led2_status, led3_status;
int b1Behaviour=0, b2Behaviour=0;

//ZIGBEE END DEVICE 1 API=1
struct xbee_con *edcon_b1805b, *cbcon_b1805b;
struct xbee_conAddress ed_addr_b1805b = { .addr64 = {
0x00,0x13,0xA2,0x00,0x40,0xB1,0x80,0x5B }, .addr64_enabled = 1 }; //used

// ZIGBEE ROUTER 1 API=1
struct xbee_con *rocon_a238d4;
struct xbee_conAddress ro_addr_a238d4 = { .addr64 = {
0x00,0x13,0xA2,0x00,0x40,0xA2,0x38,0xD4 }, .addr64_enabled = 1 }; //not used

// END DEVICE 2 API=1
struct xbee_con *edcon_af5a99;
struct xbee_conAddress ed_addr_af5a99 = { .addr64 = {
0x00,0x13,0xA2,0x00,0x40,0xAF,0x5A,0x99 }, .addr64_enabled = 1 }; //not used
// END DEVICE 3 API=1
struct xbee_con *edcon_b18074, *cbcon_b18074;
```

```

struct xbee_conAddress ed_addr_b18074 = { .addr64 = {
0x00,0x13,0xA2,0x00,0x40,0xB1,0x80,0x74 }, .addr64_enabled = 1 }; //used

// ZIGBEE ROUTER 2 API=1
struct xbee_con *rocon_8b5f51;
struct xbee_conAddress ro_addr_8b5f51 = { .addr64 = {
0x00,0x13,0xA2,0x00,0x40,0x8B,0x5F,0x51 }, .addr64_enabled = 1 }; //not used

void clear_screen() {
printf("\x1B[2J\x1B[1;1H");
}

void general_menu() {
printf("%s##### GENERAL MENU ##### \n",green);
printf("0. List Network \n");
printf("1. Toggle Led 1\n");
printf("2. Toggle Led 2 \n");
printf("3. Toggle Led 3 \n");
printf("4. Button 1 Behaviour\n");
printf("5. Button 2 Behaviour\n");
printf("6. Clear all \n");
printf("7. Exit \n");
printf("##### \n");
}

void list_menu(){
printf("Led 1, Led 2, Led3, Button Behaviour \n");
printf("0 , 0 , 0 , 0\n");
printf("0 , 0 , 1 , 1\n");
printf("0 , 1 , 0 , 2\n");
printf("0 , 1 , 1 , 3\n");
printf("1 , 0 , 0 , 4\n");
printf("1 , 0 , 1 , 5\n");
printf("1 , 1 , 0 , 6\n");
printf("1 , 1 , 1 , 7\n");
}

void list_network_static(){
if (ed_addr_b1805b.addr64_enabled) {
    printf(" End Device 1, 64-bit address: 0x%02X%02X%02X%02X 0x%02X%02X%02X%02X,
attached button ed1 in DIO1\n",ed_addr_b1805b.addr64[0], ed_addr_b1805b.addr64[1],
ed_addr_b1805b.addr64[2],ed_addr_b1805b.addr64[3],ed_addr_b1805b.addr64[4],
ed_addr_b1805b.addr64[5], ed_addr_b1805b.addr64[6], ed_addr_b1805b.addr64[7]);
}
if (ro_addr_a238d4.addr64_enabled) {
    printf(" Router 1,       64-bit address: 0x%02X%02X%02X%02X 0x%02X%02X%02X%02X,
attached led ro1 in DIO0\n",ro_addr_a238d4.addr64[0], ro_addr_a238d4.addr64[1],
ro_addr_a238d4.addr64[2],ro_addr_a238d4.addr64[3],ro_addr_a238d4.addr64[4],
ro_addr_a238d4.addr64[5], ro_addr_a238d4.addr64[6], ro_addr_a238d4.addr64[7]);
}
if (ro_addr_8b5f51.addr64_enabled) {
    printf(" Router 2,       64-bit address: 0x%02X%02X%02X%02X 0x%02X%02X%02X%02X,
attached led ro2 in DIO0\n",ro_addr_8b5f51.addr64[0], ro_addr_8b5f51.addr64[1],

```

```

ro_addr_8b5f51.addr64[2],ro_addr_8b5f51.addr64[3],ro_addr_8b5f51.addr64[4],
ro_addr_8b5f51.addr64[5], ro_addr_8b5f51.addr64[6], ro_addr_8b5f51.addr64[7]);
}

if (ed_addr_af5a99.addr64_enabled) {
    printf(" End Device 2, 64-bit address: 0x%02X%02X%02X%02X 0x%02X%02X%02X,
attached led ed2 in DIO0\n",ed_addr_af5a99.addr64[0], ed_addr_af5a99.addr64[1],
ed_addr_af5a99.addr64[2],ed_addr_af5a99.addr64[3],ed_addr_af5a99.addr64[4],
ed_addr_af5a99.addr64[5], ed_addr_af5a99.addr64[6], ed_addr_af5a99.addr64[7]);
}

if (ed_addr_b18074.addr64_enabled) {
    printf(" End Device 3, 64-bit address: 0x%02X%02X%02X%02X 0x%02X%02X%02X,
attached button ed3 in DIO1\n",ed_addr_b18074.addr64[0], ed_addr_b18074.addr64[1],
ed_addr_b18074.addr64[2],ed_addr_b18074.addr64[3],ed_addr_b18074.addr64[4],
ed_addr_b18074.addr64[5], ed_addr_b18074.addr64[6], ed_addr_b18074.addr64[7]);
}

void die(const char *msg, xbee_err ret) {
    fprintf(stderr, "%s returned %d (%s)\n", msg, ret, xbee_errorToStr(ret));
exit(1);
}

// Call Back End Device 1
void callback_btn_b1805b(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt
**pkt, void **data) {
    if (!data) return;
    struct xbee_con *con_xbee = *data;
    int val;
    if (xbee_pktDigitalGet(*pkt, 1, 0, &val) == XBEE_ENONE) {
        if (!val) {
            pthread_mutex_lock (&led_mutex);
            switch (b1Behaviour){
                case 0: {
                    //NO LED AFFECTED
                    break;
                }
                case 1: {
                    led3_status=!led3_status;
                    break;
                }
                case 2: {
                    led2_status=!led2_status;
                    break;
                }
                case 3: {
                    led2_status=!led2_status;
                    led3_status=!led3_status;
                    break;
                }
                case 4: {
                    led1_status=!led1_status;
                    break;
                }
            }
        }
    }
}

```

```

        case 5: {
            led1_status=!led1_status;
            led3_status=!led3_status;
            break;
        }
        case 6: {
            led1_status=!led1_status;
            led2_status=!led2_status;
            break;
        }
        case 7: {
            led1_status=!led1_status;
            led2_status=!led2_status;
            led3_status=!led3_status;
            break;
        }
    }
    pthread_mutex_unlock (&led_mutex);

    if (led1_status+4 == 4 ) printf("Router 1 - LED OFF \n");
    else printf("Router 1 - LED ON \n");
    xbee_conTx(rocon_a238d4, NULL, "D0%c", led1_status+4);

    if (led2_status+4 == 4 ) printf("End Device 2 - LED OFF \n");
    else printf("End Device 2 - LED ON \n");
    xbee_conTx(edcon_af5a99, NULL, "D0%c", led2_status+4);

    if (led3_status+4 == 4 ) printf("Router 2 - LED OFF \n");
    else printf("Router 2 - LED ON \n");
    xbee_conTx(rocon_8b5f51, NULL, "D0%c", led3_status+4);
}
}

// Call Back End Device 3
void callback_btn_b18074(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt
**pkt, void **data) {
    if (!data) return;
    struct xbee_con *con_xbee = *data;
    int val;
    if (xbee_pktDigitalGet(*pkt, 1, 0, &val) == XBEE_ENONE) {
        if (!val) {
            pthread_mutex_lock (&led_mutex);
            switch (b2Behaviour){
                case 0: {
                    //NO LED AFFECTED
                    break;
                }
                case 1: {
                    led3_status=!led3_status;
                    break;
                }
                case 2: {

```

```

                led2_status=!led2_status;
            break;
        }
    case 3: {
        led2_status=!led2_status;
        led3_status=!led3_status;
    break;
}
case 4: {
    led1_status=!led1_status;
break;
}
case 5: {
    led1_status=!led1_status;
    led3_status=!led3_status;
break;
}
case 6: {
    led1_status=!led1_status;
    led2_status=!led2_status;
break;
}
case 7: {
    led1_status=!led1_status;
    led2_status=!led2_status;
    led3_status=!led3_status;
break;
}
}
pthread_mutex_unlock (&led_mutex);
if (led1_status+4 == 4 ) printf("Router 1 - LED OFF \n");
else printf("Router 1 - LED ON \n");
xbee_conTx(rocon_a238d4, NULL, "D0%c", led1_status+4);

if (led2_status+4 == 4 ) printf("End Device 2 - LED OFF \n");
else printf("End Device 2 - LED ON \n");
xbee_conTx(edcon_af5a99, NULL, "D0%c", led2_status+4);

if (led3_status+4 == 4 ) printf("Router 2 - LED OFF \n");
else printf("Router 2 - LED ON \n");
xbee_conTx(rocon_8b5f51, NULL, "D0%c", led3_status+4);
}

}

int setup(char * device){
// Set serial xbee
if ((ret = xbee_setup(&xbee, "xbeeZB", device, 9600)) != XBEE_ENONE)
die("xbee_setup()", ret);

// Led 1
}

```

```

if ((ret = xbee_conNew(xbee, &rocon_a238d4, "Remote AT", &ro_addr_a238d4)) != XBEE_ENONE) die("xbee_conNew(rocon_a238d4)", ret);
if ((ret = xbee_conTx(rocon_a238d4, NULL, "D0%c", XBEE_IO_OUT_LOW)) != XBEE_ENONE) die("xbee_conTx(rocon_a238d4, D1)", ret);

// Led 2
if ((ret = xbee_conNew(xbee, &edcon_af5a99, "Remote AT", &ed_addr_af5a99)) != XBEE_ENONE) die("xbee_conNew(edcon_af5a99)", ret);
if ((ret = xbee_conTx(edcon_af5a99, NULL, "D0%c", XBEE_IO_OUT_LOW)) != XBEE_ENONE) die("xbee_conTx(edcon_af5a99, D1)", ret);

// Led 3
if ((ret = xbee_conNew(xbee, &rocon_8b5f51, "Remote AT", &ro_addr_8b5f51)) != XBEE_ENONE) die("xbee_conNew(rocon_8b5f51)", ret);
if ((ret = xbee_conTx(rocon_8b5f51, NULL, "D0%c", XBEE_IO_OUT_LOW)) != XBEE_ENONE) die("xbee_conTx(rocon_8b5f51, D1)", ret);

// Button End Device 1
if ((ret = xbee_conNew(xbee, &edcon_b1805b, "Remote AT", &ed_addr_b1805b)) != XBEE_ENONE) die("xbee_conNew(edcon_b1805b)", ret);
if ((ret = xbee_conTx(edcon_b1805b, NULL, "D1%c", XBEE_IO_IN_DIGITAL)) != XBEE_ENONE) die("xbee_conTx(edcon_b1805b, D1)", ret);
// Set IC Button End Device 1
if ((ret = xbee_conTx(edcon_b1805b, NULL, "IR%c", 0)) != XBEE_ENONE) die("xbee_conTx(edcon_b1805b, IR)", ret);
if ((ret = xbee_conTx(edcon_b1805b, NULL, "IC%c", 0x02)) != XBEE_ENONE) die("xbee_conTx(edcon_b1805b, IR)", ret); // 0x02 is the mask for D1
// Call Back edcon_b1805b
if ((ret = xbee_conNew(xbee, &cbcon_b1805b, "I/O", &ed_addr_b1805b)) != XBEE_ENONE) die("xbee_conNew(cbcon_b1805b)", ret);
if ((ret = xbee_conCallbackSet(cbcon_b1805b, callback_btn_b1805b, NULL)) != XBEE_ENONE) die("xbee_conCallbackSet(cbcon_b1805b)", ret);

// Button End Device 3
if ((ret = xbee_conNew(xbee, &edcon_b18074, "Remote AT", &ed_addr_b18074)) != XBEE_ENONE) die("xbee_conNew(edcon_b18074)", ret);
if ((ret = xbee_conTx(edcon_b18074, NULL, "D1%c", XBEE_IO_IN_DIGITAL)) != XBEE_ENONE) die("xbee_conTx(edcon_b18074, D1)", ret);
// Set IC Button End Device 3
if ((ret = xbee_conTx(edcon_b18074, NULL, "IR%c", 0)) != XBEE_ENONE) die("xbee_conTx(edcon_b18074, IR)", ret);
if ((ret = xbee_conTx(edcon_b18074, NULL, "IC%c", 0x02)) != XBEE_ENONE) die("xbee_conTx(edcon_b18074, IR)", ret); // 0x02 is the mask for D1
// Call Back edcon_b18074
if ((ret = xbee_conNew(xbee, &cbcon_b18074, "I/O", &ed_addr_b18074)) != XBEE_ENONE) die("xbee_conNew(cbcon_b18074)", ret);
if ((ret = xbee_conCallbackSet(cbcon_b18074, callback_btn_b18074, NULL)) != XBEE_ENONE) die("xbee_conCallbackSet(cbcon_b18074)", ret);
return 0;
}

void sync(){
xbee_conTx(rocon_a238d4, NULL, "D0%c", led1_status+4);

```

```

xbee_conTx(edcon_af5a99, NULL, "D0%c", led2_status+4);
xbee_conTx(rocon_8b5f51, NULL, "D0%c", led3_status+4);
}

int main(int argc, char *argv[]) {
pthread_mutex_init (&led_mutex, NULL);
setup("/dev/ttyUSB0");
int nselectg, nselects;
int get_b1Behaviour, get_b2Behaviour;
unsigned int status_pin;
sync();
do {
    clear_screen();
    general_menu ();
    printf("select operation: ");
    scanf("%d",&nselectg);
    switch (nselectg) {
        case 0: {
            list_network_static();
            usleep(5000000);
            break;
        }
        case 1: {
            led1_status=!led1_status;
            if (led1_status+4 == 4 ) printf("Router 1 - LED OFF \n");
            else printf("Router 1 - LED ON \n");
            usleep(1000000);
            break;
        }
        case 2: {
            led2_status=!led2_status;
            if (led2_status+4 == 4 ) printf("End Device 2 - LED OFF \n");
            else printf("End Device 2 - LED ON \n");
            usleep(1000000);
            break;
        }
        case 3: {
            led3_status=!led3_status;
            if (led3_status+4 == 4 ) printf("Router 2 - LED OFF \n");
            else printf("Router 2 - LED ON \n");
            usleep(1000000);
            break;
        }
        case 4: {
            list_menu();
            printf("Get into Button 1 Behaviour: ");
            scanf("%d",&get_b1Behaviour);
            if ((get_b1Behaviour >=0) && (get_b1Behaviour <=7)) {
                b1Behaviour=get_b1Behaviour;
            }
            else printf("error digit\n");
            break;
        }
    }
}

```

```

        case 5: {
            list_menu();
            printf("Get into Button 2 Behaviour: ");
            scanf("%d",&get_b2Behaviour);
            if ((get_b2Behaviour >=0) && (get_b2Behaviour <=7)) {
                b2Behaviour=get_b2Behaviour;
            }
            else printf("error digit\n");
            break;
        }
        case 6: {
            clear_screen();
            general_menu ();
            break;
        }
    }
    sync();
    //usleep(1000000); /* 1 second sleep */
} while (nselectg!=7);
xbee_shutdown(xbee);
return 0;
}

```

```
struct xbee_con *edcon_b1805b, *cbcon_b1805b;
```

```
- struct xbee_conAddress ed_addr_b1805b = { .addr64 ={  
0x00,0x13,0xA2,0x00,0x40,0xB1,0x80,0x5B }, .addr64_enabled = 1 };
```

La struttura xbee_conAddress contiene dettagli sull'indirizzo di rete di un nodo remoto. E' consigliabile inizializzare tale struttura a 0 invocando la funzione memset() prima di riempirla e quindi invocare xbee_conNew(3). I dati sono ordinati con i bit più significativi (Most Significant Bit) dell'indirizzo, nel primo byte dell'array

```
xbee_err xbee_setup(struct xbee **retXbee, const char *mode, ...):
```

```
- xbee_setup(&xbee, "xbeeZB", device, 9600)
```

```
struct xbee *xbee;
```

```
xbee_err ret;
```

La funzione xbee_setup() avvia un'istanza di libxbee.

retXbee è un puntatore restituito all'istanza di libxbee.

mode specifica quale modo dovrebbe essere avviato.

```
xbee_err xbee_conNew(struct xbee *xbee, struct xbee_con **retCon, const char *type,  
struct xbee_conAddress *address):
```

```
- xbee_conNew(xbee, &edcon_b1805b, "Remote AT", &ed_addr_b1805b)
```

xbee_conNew() creerà una nuova connessione attraverso l'istanza xbee messa a disposizione dalla libreria libxbee. Una lista di tipi di connessione disponibili può essere ottenuta usando xbee_conGetTypes(3). L'indirizzo indica con quale dispositivo remoto si desidera comunicare - non è sempre appropriato usarlo (ad

esempio con una connessione "Local AT"), in tal caso NULL dovrebbe essere passato per sicurezza.

xbee_conGetTypes() permette di ricavare una lista di tipi di connessione che sono messi a disposizione dal modo dell'istanza di libxbee. Una volta che l'array restituito è stato utilizzato, si dovrebbe invocare free() per liberare il puntatore ed evitare spreco di memoria. L'ultimo elemento dell'array viene impostato a NULL per marcare la fine.

- xbee_conTx(rocon_a238d4, NULL, "D1%c", XBEE_IO_OUT_LOW)

Quando si invoca la funzione xbee_conTx, si usa uno stile di formattazione di stringhe e parametri come nella printf(). Analogamente, xbee_convTx() fa uso della vprintf(). Quando si usano queste due funzioni, la lunghezza del messaggio è determinata dal valore restituito da vprintf(). Questo significa che usando gli specificatori di conversione "%c" dentro il formato stringa, è possibile ottenere il i byte del valore zero - 0x00 - prima della fine del messaggio. Non usare "\\\" in quanto verrà concluso in anticipo il formato stringa.

xbee_err xbee_conCallbackSet (struct xbee_con *con, xbee_t_conCallback newCallback, xbee_t_conCallback *oldCallback):

- xbee_conCallbackSet(cbcon_b1805b, callback_btn_b1805b, NULL)

xbee_conCallbackSet() consente di associare una funzione di callback con una connessione, durante il recupero della callback corrente.

con deve avere una connessione valida, restituita da xbee_conNew().

newCallback deve essere o NULL (per disabilitare le callbacks) o non-NUL per abilitarle con l'indirizzo della funzione data.

oldCallback può essere NULL per indicare che non si desidera recuperare la callback precedente assegnata.

- xbee_pktDigitalGet(*pkt, 1, 0, &val)

Queste funzioni forniscono l'accesso ai dati che sono stati analizzati dal body del pacchetto di libxbee. In generale, i dati grezzi sono ancora accessibili attraverso il pacchetto, ma così si crea un'interfaccia più user friendly.

xbee_pktDataGet() cercherà i dataItems del pacchetto per una chiave nominata. L'ID e l'indice sono usati per dare un layout bidimensionale alla chiave. In ogni caso, se l'ID ha valore -1, viene ignorato, e solo la prima colonna verrà usata. Ciò permette a dati simili - ad esempio campioni analogici - di essere memorizzati usando una singola chiave, con differenti numeri di canale (ID). Analogamente, xbee_pktDigitalGet() è quasi sinonimo con xbee_pktDataGet().

- xbee_shutdown(xbee)

Terminerà tutte le connessioni rimanenti e libererà tutti i dati associati con l'istanza di libxbee.

Appena avviato il programma è presente un menu a scelta:

```
##### GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: □
```

Digitando 0 si ha la lista dei nodi statici, con i relativi indirizzi, pulsanti e led attaccati:

```
##### GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: 0
End Device 1, 64-bit address: 0x0013A200 0x40B1805B, attached button ed1 in DIO1
Router 1, 64-bit address: 0x0013A200 0x40A238D4, attached led r01 in DIO0
Router 2, 64-bit address: 0x0013A200 0x408B5F51, attached led r02 in DIO0
End Device 2, 64-bit address: 0x0013A200 0x40AF5A99, attached led ed2 in DIO0
End Device 3, 64-bit address: 0x0013A200 0x40B18074, attached button ed3 in DIO1
□
```

Digitando 1, 2 o 3 cambia lo stato del led attaccato rispettivamente al Router 1, End Device 2 e Router 2.

Consideriamo solo il caso 1, visto che è la stessa cosa per tutti i led, si parte da una condizione di tutti i led spenti.

Digitando 1 il led attaccato al Router 1 si accende:

```
##### GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: 1
Router 1 - LED ON
```

Adesso, se digitiamo nuovamente 1 il led attaccato al Router 1 si spegne.

```
#####
# GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: 1
Router 1 - LED OFF
█
```

Digitando 4, abbiamo una tabella delle operazioni da effettuare al pulsante attaccato all'End Device 1. Successivamente, digitiamo 7 in modo che il pulsante memorizza lo stato inviato.

```
#####
# GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: 4
Led 1, Led 2, Led3, Button Behaviour
0 , 0 , 0 , 0
0 , 0 , 1 , 1
0 , 1 , 0 , 2
0 , 1 , 1 , 3
1 , 0 , 0 , 4
1 , 0 , 1 , 5
1 , 1 , 0 , 6
1 , 1 , 1 , 7
Get into Button 1 Behaviour: 7█
```

Quando premiamo il pulsante abbiamo, che i 3 led si accendono:

```
#####
# GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: Router 1 - LED ON
End Device 2 - LED ON
Router 2 - LED ON
█
```

Se ripremiamo il pulsante i 3 led si spengono:

```
##### GENERAL MENU #####
0. List Network
1. Toggle Led 1
2. Toggle Led 2
3. Toggle Led 3
4. Button 1 Behaviour
5. Button 2 Behaviour
6. Clear all
7. Exit
#####
select operation: Router 1 - LED ON
End Device 2 - LED ON
Router 2 - LED ON
Router 1 - LED OFF
End Device 2 - LED OFF
Router 2 - LED OFF
```

Riassumendo, il coordinatore, si preoccupa di gestire tutte le richieste che gli arrivano. Se arriva una richiesta da terminale di cambiare lo stato di un led, manderà al nodo selezionato un pacchetto di spegnimento o accensione quando si farà il toggle del led. Differenti sono i discorsi della gestione del pulsante. Tale gestione prevede l'utilizzo del comando IC (IO Digital Change Detection, vedi tabella sotto) associato al nodo dove è collegato il pulsante. Infatti, a ogni cambio di stato del pulsante, il nodo invia un pacchetto al coordinatore. Quest'ultimo esamina il pacchetto e invia un pacchetto di spegnimento o accensione al nodo associato al pulsante per spegnere o accendere il led. Questo è stato un punto difficile nella progettazione di tale software.

I/O Commands

I/O Commands

AT Command	Name and Description	Node Type ¹	Parameter Range	Default
IR	IO Sample Rate. Set/Read the IO sample rate to enable periodic sampling. For periodic sampling to be enabled, IR must be set to a non-zero value, and at least one module pin must have analog or digital IO functionality enabled (see D0-D8, P0-P2 commands). The sample rate is measured in milliseconds.	CRE	0, 0x32:0xFFFF (ms)	0
IC	IO Digital Change Detection. Set/Read the digital IO pins to monitor for changes in the IO state. IC works with the individual pin configuration commands (D0-D8, P0-P2). If a pin is enabled as a digital input/output, the IC command can be used to force an immediate IO sample transmission when the DIO state changes. IC is a bitmask that can be used to enable or disable edge detection on individual channels. Unused bits should be set to 0. Bit (IO pin): 0 (DIO0) 4 (DIO4) 8 (DIO8) 1 (DIO1) 5 (DIO5) 9 (DIO9) 2 (DIO2) 6 (DIO6) 10 (DIO10) 3 (DIO3) 7 (DIO7) 11 (DIO11)	CRE	: 0 - 0xFFFF	0
P0	PWM0 Configuration. Select/Read function for PWM0.	CRE	0 = Disabled 1 = RSSI PWM 3 - Digital input, monitored 4 - Digital output, default low 5 - Digital output, default high	1
P1	DIO11 Configuration. Configure options for the DIO11 line of the RF module.	CRE	0 - Unmonitored digital input 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0

P2	DIO12 Configuration. Configure options for the DIO12 line of the RF module.	CRE	0 - Unmonitored digital input 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0
P3	DIO13 Configuration. Set/Read function for DIO13. This command is not yet supported.	CRE	0, 3-5 0 – Disabled 3 – Digital input 4 – Digital output, low 5 – Digital output, high	
D0	AD0/DIO0 Configuration. Select/Read function for AD0/DIO0.	CRE	1- Commissioning button enabled 2 - Analog input, single ended 3 - Digital input 4 - Digital output, low 5 - Digital output, high	1
D1	AD1/DIO1 Configuration. Select/Read function for AD1/DIO1.	CRE	0, 2-5 0 – Disabled 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D2	AD2/DIO2 Configuration. Select/Read function for AD2/DIO2.	CRE	0, 2-5 0 – Disabled 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0

Test Finale

Il test finale è stato importante, evidenziando malfunzionamenti e errori non emersi durante la fase di progettazione nel codice. Portando così alla soluzioni di tali problemi non emersi.

Sviluppi Futuri

In tale progetto è stato implementato il controllo di una WSN, lo sviluppo successivo è il monitoraggio con la lettura di qualche sensore di temperatura (es. [MCP9700](#)) o una fotoresistenza (es. [GL5528](#)).

Documentazione

- [Easy Bee - Guida alla scelta e alla comprensione dei moduli XBee](#)
- [Building Wireless Sensor Networks with ZigBee, XBee, Arduino, and Processing By Robert FaludiBy](#)
- [XBee® & XBee-PRO® ZB ZigBee® PRO RF Modules](#)
- [XBee manual: XB24-Z7WIT-004](#)
- [What is API \(Application Programming Interface\) Mode and how does it work?](#)
- [Digital and Analog Sampling Using XBee Radios](#)
- [Digi XBee Examples & Guides](#)
- [Digi X-CTU User's Guide](#)

- [Digi X-CTU Download Page](#)
- [Wikispaces XBee](#)
- [Parallax USB Driver Installer](#)
- [Getting X-CTU in Wine to detect your serial ports](#)
- [Libxbee](#)
- [Github Libxbee](#)
- [Doc Libxbee](#)
- [Example Libxbee](#)
- [GNU Compiler Collection](#)
- [GNU Lesser General Public License \(GPL\)](#)