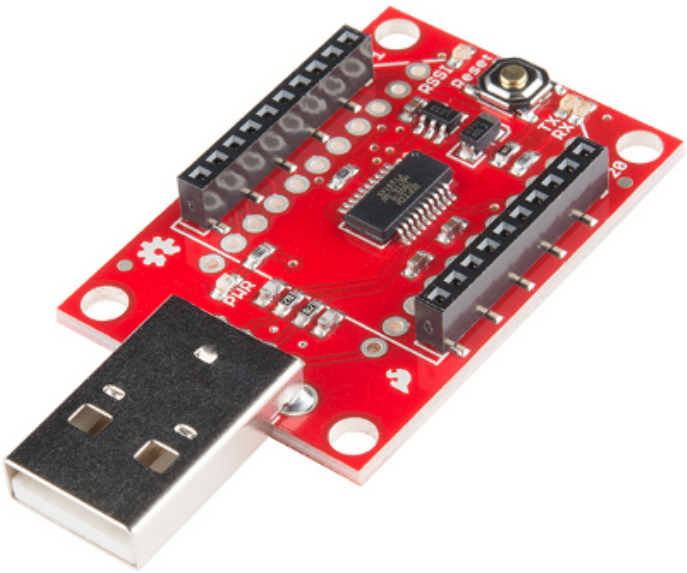
Mise en oeuvre des modules xbee1

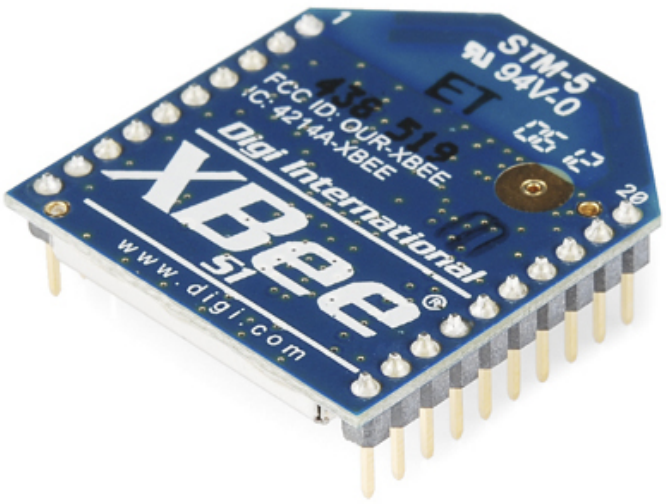
Thierry Vaira <tvaira@free.fr>

22/03/2016

Table des matières

Mise en oeuvre des modules xbee1





Introduction

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 pour les réseaux à dimension personnelle (*Wireless Personal Area Networks* : WPAN).

Lire ZigBee.

Installation libxbee

Lien : libxbee

```
$ unzip libxbee-v3.libxbee-76d63381f85f.zip
$ cd libxbee-v3.libxbee-76d63381f85f/
$ make configure
$ make all
$ sudo make install
```

Tests

Prise en charge

Avant :

```
$ lsusb > lsusb-avant.txt
$ lsmod > lsmod-avant.txt
$ dmesg > dmesg-avant.txt
```

Brancher maintenant le module Xbee USB.

Après :

```
$ lsusb > lsusb-apres.txt
$ lsmod > lsmod-apres.txt
$ dmesg > dmesg-apres.txt
```

Analyse :

```
$ diff lsusb-avant.txt lsusb-apres.txt
Bus 003 Device 091: ID 0403:6015 Future Technology Devices International, Ltd

$ diff dmesg-avant.txt dmesg-apres.txt
usb 3-9.4.2: new full-speed USB device number 91 using xhci_hcd
usb 3-9.4.2: New USB device found, idVendor=0403, idProduct=6015
usb 3-9.4.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 3-9.4.2: Product: FT231X USB UART
usb 3-9.4.2: Manufacturer: FTDI
usb 3-9.4.2: SerialNumber: DA011HYD
ftdi_sio 3-9.4.2:1.0: FTDI USB Serial Device converter detected
usb 3-9.4.2: Detected FT-X
usb 3-9.4.2: Number of endpoints 2
usb 3-9.4.2: Endpoint 1 MaxPacketSize 64
usb 3-9.4.2: Endpoint 2 MaxPacketSize 64
usb 3-9.4.2: Setting MaxPacketSize 64
usb 3-9.4.2: FTDI USB Serial Device converter now attached to ttyUSB0

$ diff lsmod-avant.txt lsmod-apres.txt
ftdi_sio               47922  0
usbserial              37161  1 ftdi_sio

$ modinfo ftdi_sio
...

$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 sept. 18 16:24 /dev/ttyUSB0
```


Par défaut, le système applique la politique des droits d'accès `rw-rw----` pour ce type de périphérique USB. Ici, le périphérique sera accessible en lecture/écriture par l'utilisateur propriétaire `root` et les membres du groupe `dialout`. Tous les autres (*other*) utilisateurs auront aucun accès.

Pour modifier cette situation, vous avez les possibilités suivantes :

– changer les droits manuellement (à chaque fois !) : `$ sudo chmod 666 /dev/ttyUSB0`
– ajouter l'utilisateur demandeur (ici toto) dans le groupe `dialout` : `$ sudo adduser toto dialout`

Pour une gestion automatique, il faudra passer par `udev` qui est maintenant le service qui prend en charge le répertoire `/dev`.

```
$ sudo vim /etc/udev/rules.d/51-ttyusb.rules
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6015", MODE="0666"
```

Commandes AT

On communique avec le périphérique par commandes AT sur le port série virtuel. Il est possible d'utiliser la commande `screen` (`Ctrl-a k` pour sortir), `picocom` (`Ctrl-a Ctrl-x` pour sortir) ou avec le logiciel `cutecom`.

Par défaut le module Xbee a les paramètres suivants : 9600/8/1/N.

Il faut tout d'abord passer le module en mode commande en lui envoyant la séquence +++ (No line end) puis on peut lui envoyer des commandes AT (CR line end). Il quitte le mode commande automatiquement au bout de 10 s.

```
+++
OK

ATVL
XBee 802.15.4 BETA V 10EC, Build: Feb 22 2011 17:05:54
Hardware Version: W43\0x00\0x00   MC13213
Software Compatibility: 01
XCVR_ID: 6800
MAC FFDNB V1061 Build: Jul 23 2007 14:47:59
PHY XBEE4 V1061 Build: Feb 22 2011 16:41:52
MAX BOOTLOADER V 0B
```

Configuration

Pour communiquer les deux modules Xbee doivent avoir le même PAN ID (ATID) et le même CHANNEL (ATCH).

```
ATID=3332
OK

ATID
3332

ATCH
C
```

– 16 bits : On peut leur affecter une adresse MY sur 16 bits (ATMY).

XBee n°1 (PC) :

```
ATMY=5000
OK

ATMY
5000
```

XBee n°2 (RPI) :

```
ATMY=5001
OK

ATMY
5001
```


– 64 bits : Les modules ont une adresse 64 bits dans les registres SH et SL. Pour communiquer en 64 bits, il faut leur mettre une adresse 16 bits égale à 0xFFFF. La commande ATND permet de découvrir les autres modules sur le réseau.

XBee n°1 (PC) :

```
ATSH
13A200

ATSL
40C17415

ATND
13A200
40C1725A
43
```

XBee n°2 (RPI) :

```
ATSH
13A200

ATSL
40C1725A
```

Pour finir, il faut enregistrer les modifications dans la mémoire flash du module (commande ATWR) et sortir du mode de configuration (ATCN) :

```
ATWR
OK

ATCN
OK
```

Il faut définir le mode de fonctionnement du module en utilisant la commande ATAP :

– AP = 0 (default) : mode “Transparent Operation” (UART Serial line replacement), les 2 modes API sont désactivés.
– AP = 1 : mode “API Operation”
– AP = 2 : mode “API Operation” (les caractères transmis sont “échappés” par protection)

```
ATAP
1

ATAP=0
OK
```

En mode AP=1, les données seront reçues directement alors qu’en mode AP=0 on réceptionnera les trames. Le programme du module Xbee n°2 (RPI) envoie périodiquement une valeur entière qu’il incrémente à chaque tour de boucle :

```
for (i = 0; i < 500; i++)
{
    xbee_conTx(con, NULL, "%d\r\n", i);
    usleep(10000); /* 10ms */
}
```

On reçoit dans cutecom :

– avec AP = 0 (mode “Transparent Operation”) :

```
1
2
3
...
```

– avec AP = 1 (mode “API Operation”) :

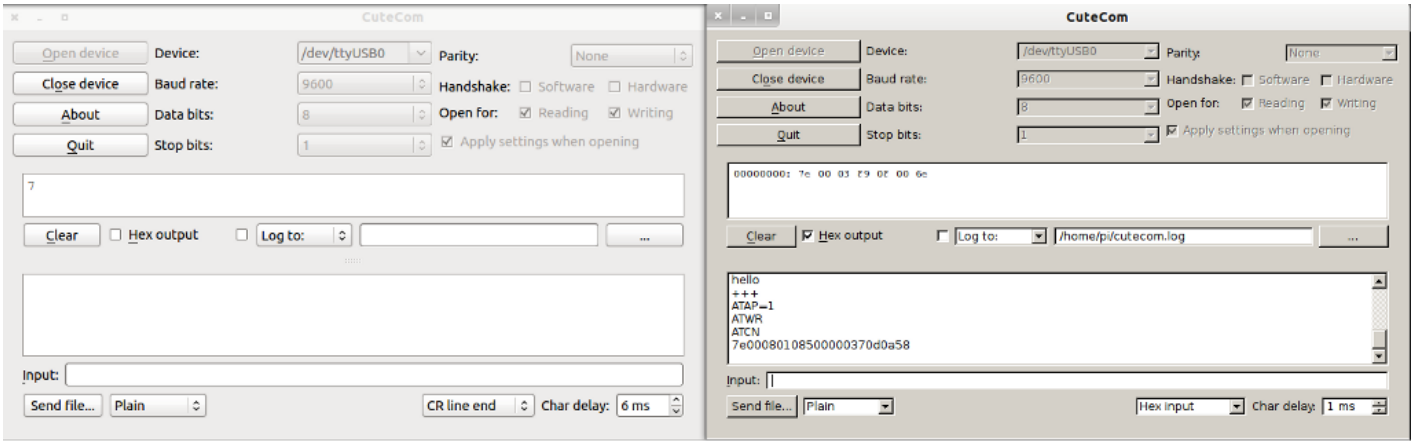
```
...
7e 00 08 01 08 50 00 00 37 0d 0a 58
...
```


Le protocole est expliqué dans la documentation xbee.pdf.

On peut maintenant faire un test de communication :

– on se place en mode AP = 1 sur le module de la Raspberry Pi
– on se place en mode AP = 0 sur le module côté PC

On démarre 2 cutecom et on envoie la trame hexadécimale “7e00080108500000370d0a58” à partir de la Raspberry Pi (adresse 5001) vers le PC (adresse 5000) :



Tests en Mode Transparent (AP = 0)

Pour communiquer les deux modules Xbee doivent avoir le même PAN ID (ATID) et le même CHANNEL (ATCH).

```
ATID=3332
OK

ATID
3332

ATCH
C
```

On leur affecte une adresse MY sur 16 bits (ATMY). On définit ensuite le mode de fonctionnement (ATAP) du module en AP = 0 : mode “Transparent Operation” (UART Serial line replacement).

On les fera communiquer en *broadcast* (l’adresse destination se configure avec les registres DH et DL). Pour une adresse de *broadcast* (diffusion générale), il faut mettre 0xFFFF dans le registre DL (ATDH et ATDL). Comme on utilise des adresses sur 16 bits, il faudra mettre le registre DH à 0.

Pour finir, il faut enregistrer les modifications dans la mémoire flash du module (ATWR) et sortir du mode de configuration (ATCN).

– XBee n°1 :

```
ATMY=5000
OK

ATMY
5000

ATAP=0
OK

ATDH=0
OK

ATDL=FFFF
OK
```



```
ATWR
OK

ATCN
OK
```

– XBee n°2 :

```
ATMY=5001
OK

ATMY
5001

ATAP=0
OK

ATDH=0
OK

ATDL=FFFF
OK

ATWR
OK

ATCN
OK
```

Dans le mode AP=0, on émet et on reçoit directement les données sur le port série. Utiliser `cutecom` (ou un logiciel équivalent) pour tester la communications entre les 2 modules.

Comme le module XBee est vu comme un "port série virtuel", on peut alors le programmer simplement comme un port série sous Qt [PDF].

Programmation en C/C++

Mode Transparent (AP = 0)

On place le module en mode AP=0. On reçoit directement les données sur le port série :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#include <string.h>

int main (int argc, char** argv)
{
    setbuf(stdout, NULL);

    int xbeefd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);
    if (xbeefd == -1)
    {
        std::cerr << "Erreur ouverture port !" << std::endl;
        return -1;
    }

    unsigned char buffer[128];

    while(1)
    {
        usleep(100000); /* 100ms */
```



```
        // remet le buffer à 0
        memset(buffer, 0, 128);

        int n = read(xbeefd, buffer, 128);

        if( n > 0)
        {
            printf("Octets lus : %d\n", n);
            fflush(stdout);

            printf("Trame reçue : ");
            for (int i = 0; i < n; i++)
                printf("0x%02X ", buffer[i]);
            printf("\n");
        }
    }

    close(xbeefd);

    return 0;
}
```

On obtient :

```
Octets lus : 2
Trame reçue : 0x00 0x0A
Octets lus : 9
Trame reçue : 0x30 0x0D 0x0A 0x31 0x0D 0x0A 0x32 0x0D 0x0A
Octets lus : 9
Trame reçue : 0x33 0x0D 0x0A 0x34 0x0D 0x0A 0x35 0x0D 0x0A
Octets lus : 6
Trame reçue : 0x36 0x0D 0x0A 0x37 0x0D 0x0A
Octets lus : 10
Trame reçue : 0x38 0x0D 0x0A 0x39 0x0D 0x0A 0x31 0x30 0x0D 0x0A
...
```

Mode “API Operation” (AP = 1) avec les appels systemcalls

On place le module en mode AP=1.

On envoie par exemple la commande ATCH :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#include <string.h>

int main (int argc, char** argv)
{
    int xbeefd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);
    if (xbeefd == -1)
    {
        std::cerr << "Erreur ouverture port !" << std::endl;
        return -1;
    }

    unsigned char buffer[128];

    // met le buffer à 0
    memset(buffer, 0, 128);

    // Commande ATID
    // voir protocole page 60 (xbee.pdf)
```



```
    buffer[0] = 0x7e;  // frame start
    buffer[1] = 0x00;  // MSB frame size
    buffer[2] = 0x04;  // LSB frame size
    buffer[3] = 0x08;  // Frame type (0x08 = Local AT)
    buffer[4] = 0x52;  // Frame ID (used for ACK)
    buffer[5] = 'C';   // First char of command
    buffer[6] = 'H';   // Second char of command

    // Calcul du checksum (page 59)
    unsigned char checksum = 0;
    for (int i = 3; i < 7; i++)
        checksum += buffer[i];

    buffer[7] = 0xFF - checksum;

    printf("Trame envoyée : ");
    for (int i = 0; i < 8; i++)
        printf("0x%02X ", buffer[i]);
    printf("\n");

    write(xbeefd, buffer, 8);

    sleep(1);

    // remet le buffer à 0
    memset(buffer, 0, 128);

    int n = read(xbeefd, buffer, 128);

    printf("Octets lus : %d\n", n);
    fflush(stdout);

    printf("Trame reçue : ");
    for (int i = 0; i < n; i++)
        printf("0x%02X ", buffer[i]);
    printf("\n");

    close(xbeefd);

    return 0;
}
```

On obtient pour la commande ATCH (C) :

```
Trame envoyée : 0x7E 0x00 0x04 0x08 0x52 0x43 0x48 0x1A
Octets lus : 10
Trame reçue : 0x7E 0x00 0x06 0x88 0x52 0x43 0x48 0x00 0x0C 0x8E
```

Et avec la commande ATMY (5000) :

```
Trame envoyée : 0x7E 0x00 0x04 0x08 0x52 0x4D 0x59 0xFF
Octets lus : 11
Trame reçue : 0x7E 0x00 0x07 0x88 0x52 0x4D 0x59 0x00 0x50 0x00 0x2F
```

En utilisant la documentation, on peut par exemple écrire la fonction envoyer() suivante :

```
int PortXbee::envoyer(char myid[2], char *donnees, int nb)
{
    int retour = -1;
    unsigned char checksum = 0;
    int i, pos, n;

    if(ouvert == true)
    {
        memset(trame, 0, LG_MAX_TRAME);
```



```
        // page 63
        trame[0] = 0x7E;
        trame[1] = ((nb+5) >> 8) & 0xFF;
        trame[2] = ((nb+5)     ) & 0xFF;
        trame[3] = 0x01;
        trame[4] = 0x08; // ACK
        trame[5] = myid[0];
        trame[6] = myid[1];
        trame[7] = 0x00; // no options
        for (i = 0; i < nb; i++)
        {
            trame[8+i] = donnees[i];
        }
        n = 8+i;
        // page 59
        checksum = 0;
        for (pos = 3; pos < n; pos++)
        {
            checksum += trame[pos];
        }
        trame[n++] = 0xFF - checksum;

        // cf. man 2 write
        retour = write(xbeefd, trame, n);

        #ifdef DEBUG_PORTXBEE
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d/%d) : ", n, retour);
        fprintf(stderr, "trame : ");
        int i;
        for(i=0;i<n;i++)
        {
            fprintf(stderr, "0x%02X ", *(trame+i));
        }
        fprintf(stderr, "\n");
        //fprintf(stderr, "%s\n", trame);
        #endif
        if (retour == -1)
        {
            perror("write");
        }
    }
    else
    {
        #ifdef DEBUG_PORTXBEE
        //debug : affichage
        fprintf(stderr, "-> envoyer (%d) : ERREUR port !\n", nb);
        #endif
        retour = nb;
    }

    return retour;
}
```

Une utilisation possible :

```
char destid[2] = {0x50, 0x00};
char donnees[128];

//...

while(1)
{
    sprintf(donnees, "%d\r\n", i);
    portXbee.envoyer(destid, donnees, strlen(donnees));
    i++;
```



```
}
```

Mode “API Operation” (AP = 1) avec la librairie libxbee

On place le module en mode AP=1.

En utilisant la librairie libxbee, on peut reprendre un des exemples fournis (en mode “Local AT”) :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <xbee.h>

const char commandes[7][3] = { { "ID" },
                               { "MY" },
                               { "SH" },
                               { "SL" },
                               { "VR" },
                               { "CH" },
                               { "NI" } };

int main(void)
{
    struct xbee *xbee;
    struct xbee_con *con;
    unsigned char txRet;
    xbee_err ret;

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("Erreur : %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

    if ((ret = xbee_conNew(xbee, &con, "Local AT", NULL)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "Erreur xbee_conNew() : %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    for(int n=0;n<7;n++)
    {
      ret = xbee_conTx(con, &txRet, commandes[n]);
      if (ret == XBEE_ETX)
      {
         printf("Erreur : %s... (0x%02X)\n", xbee_errorToStr(ret), txRet);
      }
      else if (ret)
      {
         printf("Erreur : %s (%d)\n", xbee_errorToStr(ret), ret);
      }
      else
      {
         struct xbee_pkt *pkt;
         if ((ret = xbee_conRx(con, &pkt, NULL)) != XBEE_ENONE)
         {
            printf("Erreur xbee_conRx() : %s\n", xbee_errorToStr(ret));
         }
         else
         {
            int i;
            printf("Commande AT%s\n", commandes[n]);
            printf("Réponse : (%d octets)\n", pkt->dataLen);
            for (i = 0; i < pkt->dataLen; i++)
            {
```



```
                printf("%3d : 0x%02X - %c\n", i, pkt->data[i], (((pkt->data[i] >= ' ') && (pkt->data[i] <= '~'))?
    pkt->data[i]:'.'));
            }
            printf("\n");
        }
      }
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "Erreur xbee_conEnd() : %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

Remarque : ajouter les bibliothèques -lxbee -lpthread -lm -lrt à la fabrication

On obtient :

```
Commande ATID
Réponse : (2 octets)
  0: 0x33 - 3
  1: 0x32 - 2

Commande ATMY
Réponse : (2 octets)
  0: 0x50 - P
  1: 0x00 - .

Commande ATSH
Réponse : (4 octets)
  0: 0x00 - .
  1: 0x13 - .
  2: 0xA2 - .
  3: 0x00 - .

Commande ATSL
Réponse : (4 octets)
  0: 0x40 - @
  1: 0xC1 - .
  2: 0x74 - t
  3: 0x15 - .

Commande ATVR
Réponse : (2 octets)
  0: 0x10 - .
  1: 0xEC - .

Commande ATCH
Réponse : (1 octets)
  0: 0x0C - .

Commande ATNI
Réponse : (1 octets)
  0: 0x20 -
```

Il est même possible d'exécuter des commandes AT distantes (en mode “Remote AT”) :

```
...
struct xbee_conAddress address;

memset(&address, 0, sizeof(address));
/* 64 bits */
```



```
/*address.addr64_enabled = 1;
address.addr64[0] = 0x00;
address.addr64[1] = 0x13;
address.addr64[2] = 0xA2;
address.addr64[3] = 0x00;
address.addr64[4] = 0x40;
address.addr64[5] = 0xC1;
address.addr64[6] = 0x72;
address.addr64[7] = 0x5A;*/

/* 16 bits */
address.addr16_enabled = 1;
address.addr16[0] = 0x50;
address.addr16[1] = 0x01;

if ((ret = xbee_conNew(xbee, &con, "Remote AT", &address)) != XBEE_ENONE)
{
    printf("Erreur xbee_conNew() : %d (%s)", ret, xbee_errorToStr(ret));
    return ret;
}
...
```

On obtient alors :

```
...
Command: ATMY
Response is 2 bytes long:
  0: 0x50 - P
  1: 0x01 - .

Command: ATSH
Response is 4 bytes long:
  0: 0x00 - .
  1: 0x13 - .
  2: 0xA2 - .
  3: 0x00 - .

Command: ATSL
Response is 4 bytes long:
  0: 0x40 - @
  1: 0xC1 - .
  2: 0x72 - r
  3: 0x5A - Z
...
```

Programmation en C/C++ (Mode “API Operation” AP = 1)

Les exemples ci-dessous ont été écrits à partir de ceux fournis pas la libxbee.

Remarque : ajouter les bibliothèques -lxbee -lpthread -lm -lrt à la fabrication

Emission (16 bits)

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>

#include <xbee.h>

int main(void)
{
    int i;
    struct xbee *xbee;
```



```
    struct xbee_con *con;
    struct xbee_conAddress address;
    xbee_err ret;

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

    memset(&address, 0, sizeof(address));
    address.addr16_enabled = 1;
    address.addr16[0] = 0x50;
    address.addr16[1] = 0x00;
    if ((ret = xbee_conNew(xbee, &con, "16-bit Data", &address)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    for (i = 0; i < 500; i++)
    {
        printf("id: %02X%02X tx: [%d]\n", address.addr16[0], address.addr16[1], i);

        xbee_conTx(con, NULL, "%d\r\n", i);

        /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
        usleep(10000); /* 10ms */
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

Réception (16 bits)

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>

#include <xbee.h>

int main(void)
{
    int i;
    struct xbee *xbee;
    struct xbee_con *con;
    struct xbee_conAddress address;
    struct xbee_pkt *pkt;
    xbee_err ret;

    setbuf(stdout, NULL);

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
```



```
    }

    memset(&address, 0, sizeof(address));
    address.addr16_enabled = 1;
    address.addr16[0] = 0x50;
    address.addr16[1] = 0x01;
    if ((ret = xbee_conNew(xbee, &con, "16-bit Data", &address)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    for (i = 0; i < 500; i++)
    {
        printf(".");

        if ((ret = xbee_conRx(con, &pkt, NULL)) == XBEE_ENONE)
        {
            printf("Reçu : (%d octets)\n", pkt->dataLen);
            if(pkt->address.addr16_enabled)
            {
                printf("id: %02X%02X rx: [%s] datalen: [%d] datas: ", pkt->address.addr16[0], pkt->address.
    addr16[1], pkt->data, pkt->dataLen);
                for(int i = 0; i < pkt->dataLen; i++)
                {
                    printf("0x%02X ", pkt->data[i]);
                }
                printf("\n");
            }
            else
                printf("rx: [%s]\n", pkt->data);
        }

       /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
       usleep(100000); /* 100ms */
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

Réception par callback (16 bits)

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>

#include <xbee.h>

void myCB(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data)
{
    //printf(" callback ! ");
    if ((*pkt)->dataLen > 0)
    {
        if ((*pkt)->data[0] == '@')
        {
            xbee_conCallbackSet(con, NULL, NULL);
```

```
            printf("*** DISABLED CALLBACK... ***\n");
        }
        if((*pkt)->address.addr16_enabled)
        {
            printf("id: %02X%02X rx: [%s] datalen: [%d] datas: ", (*pkt)->address.addr16[0], (*pkt)->address.
    addr16[1], (*pkt)->data, (*pkt)->dataLen);
            for(int i = 0; i < (*pkt)->dataLen; i++)
            {
                printf("0x%02X ", (*pkt)->data[i]);
            }
            printf("\n");
        }
        else
            printf("rx: [%s]\n", (*pkt)->data);
    }
}

int main(void)
{
    int i;
    struct xbee *xbee;
    struct xbee_con *con;
    struct xbee_conAddress address;
    xbee_err ret;

    setbuf(stdout, NULL);

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

    memset(&address, 0, sizeof(address));
    address.addr16_enabled = 1;
    address.addr16[0] = 0x50;
    address.addr16[1] = 0x01;

    if ((ret = xbee_conNew(xbee, &con, "16-bit Data", &address)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    if ((ret = xbee_conCallbackSet(con, myCB, NULL)) != XBEE_ENONE)
    {
        printf("xbee_conCallbackSet() returned: %d\n", ret);
        return ret;
    }

    for (i = 0; i < 500; i++)
    {
       printf(".");

       /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
       usleep(100000); /* 100ms */
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);
```



```
    return 0;
}
```

Emission (64 bits)

Remarque : il faut mettre 0xFFFF dans l'adresse 16 bits pour passer en 64 bits (ATMY=FFFF)

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>

#include <xbee.h>

/*
 * For 64-bits :
 * ATMY=FFFF
 */

int main(void)
{
    int i;
    void *d;
    struct xbee *xbee;
    struct xbee_con *con;
    struct xbee_conAddress address;
    struct xbee_conSettings settings;
    xbee_err ret;

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

    memset(&address, 0, sizeof(address));
    address.addr64_enabled = 1;
    address.addr64[0] = 0x00;
    address.addr64[1] = 0x13;
    address.addr64[2] = 0xA2;
    address.addr64[3] = 0x00;
    address.addr64[4] = 0x40;
    address.addr64[5] = 0xC1;
    address.addr64[6] = 0x74;
    address.addr64[7] = 0x15;
    if ((ret = xbee_conNew(xbee, &con, "64-bit Data", &address)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    /* getting an ACK for a broadcast message is kinda pointless... */
    xbee_conSettings(con, NULL, &settings);
    settings.disableAck = 1;
    xbee_conSettings(con, &settings, NULL);

    for (i = 0; i < 1000; i++)
    {
        xbee_conTx(con, NULL, "Hello %s", i);

        /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
        usleep(10000); /* 10ms */
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
```



```
    {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

Réception par callback (64 bits)

Remarque : il faut mettre 0xFFFF dans l'adresse 16 bits pour passer en 64 bits (ATMY=FFFF)

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <unistd.h>

#include <xbee.h>

/*
 * For 64-bits :
 * ATMY=FFFF
 */

void myCB(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data)
{
    if ((*pkt)->dataLen > 0)
    {
        if ((*pkt)->data[0] == '@')
        {
            xbee_conCallbackSet(con, NULL, NULL);
            printf("*** DISABLED CALLBACK... ***\n");
        }
        if((*pkt)->address.addr64_enabled)
            printf("id: %02X%02X%02X%02X%02X%02X%02X%02X rx: [%s]\n",
                    (*pkt)->address.addr64[0], (*pkt)->address.addr64[1], (*pkt)->address.addr64[2],    (*pkt)->
    address.addr64[3],
                    (*pkt)->address.addr64[4],  (*pkt)->address.addr64[5],  (*pkt)->address.addr64[6],  (*pkt)->
    address.addr64[7],
                    (*pkt)->data);
        else
            printf("rx: [%s]\n", (*pkt)->data);
    }
}

int main(void)
{
    int i;
    struct xbee *xbee;
    struct xbee_con *con;
    struct xbee_conAddress address;
    xbee_err ret;

    if ((ret = xbee_setup(&xbee, "xbee1", "/dev/ttyUSB0", 9600)) != XBEE_ENONE)
    {
        printf("ret: %d (%s)\n", ret, xbee_errorToStr(ret));
        return ret;
    }

    memset(&address, 0, sizeof(address));
    address.addr64_enabled = 1;
    address.addr64[0] = 0x00;
    address.addr64[1] = 0x13;
```



```
    address.addr64[2] = 0xA2;
    address.addr64[3] = 0x00;
    address.addr64[4] = 0x40;
    address.addr64[5] = 0xC1;
    address.addr64[6] = 0x72;
    address.addr64[7] = 0x5A;

    if ((ret = xbee_conNew(xbee, &con, "64-bit Data", &address)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conNew() returned: %d (%s)", ret, xbee_errorToStr(ret));
        return ret;
    }

    if ((ret = xbee_conCallbackSet(con, myCB, NULL)) != XBEE_ENONE)
    {
        printf("xbee_conCallbackSet() returned: %d\n", ret);
        return ret;
    }

    for (i = 0; i < 500; i++)
    {
        printf("id: %02X%02X%02X%02X%02X%02X%02X%02X%02X tx: [%d]\n",
                address.addr64[0], address.addr64[1], address.addr64[2], address.addr64[3],
                address.addr64[4], address.addr64[5], address.addr64[6], address.addr64[7], i);

        xbee_conTx(con, NULL, "%d\r\n", i);

        /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
        usleep(10000); /* 10ms */
    }

    if ((ret = xbee_conEnd(con)) != XBEE_ENONE)
    {
        xbee_log(xbee, -1, "xbee_conEnd() returned: %d", ret);
        return ret;
    }

    xbee_shutdown(xbee);

    return 0;
}
```

C++

On utilise la classe XBee fournie. Si on veut utiliser les callbacks, il faut dériver la classe ConCallback et définir la méthode xbee_conCallback().

Remarque : ajouter les bibliothèques -lxbeep -lxbee -lpthread -lm -lrt à la fabrication

```
#ifndef XBEE_PI_H
#define XBEE_PI_H

#include <iostream>
#include <string.h>
#include <unistd.h>

#include <xbeep.h>

using namespace std;
using namespace libxbee;

class ConnexionXbee : public ConCallback
{
    public:
        explicit ConnexionXbee(XBee &parent, string type, struct xbee_conAddress *address = NULL);
```



```
        ~ConnexionXbee();
        void xbee_conCallback(libxbee::Pkt **paquet);
        void recevoir(Pkt **paquet);
        std::string myData;

    private:
        string donnees;
};

#endif //XBEE_PI_H
```

```
#include "xbee-pi.h"

ConnexionXbee::ConnexionXbee(XBee &parent, string type, struct xbee_conAddress *address) : ConCallback(parent,
    type, address)
{
}

ConnexionXbee::~ConnexionXbee()
{
}

void ConnexionXbee::xbee_conCallback(libxbee::Pkt **paquet)
{
    recevoir(paquet);
}

void ConnexionXbee::recevoir(Pkt **paquet)
{
    cout << "recevoir() !\n";

    int i;
    for (i = 0; i < (*paquet)->size(); i++)
    {
        cout << (**paquet)[i];
    }
    cout << "\n";

    /* if you want to keep the packet, then you MUST do the following:
          libxbee::Pkt *myhandle = *paquet;
          *paquet = NULL;
       and then later, you MUST delete the packet to free up the memory:
          delete myhandle;
       if you do not want to keep the packet, then just leave everything as-is, and it will be free'd for you */
}
```

Et le programme principal :

```
#include <iostream>
#include <string.h>
#include <unistd.h>

#include <xbeep.h>
#include "xbee-pi.h"

using namespace std;
using namespace libxbee;

void listerModes()
{
    try
    {
        std::list<std::string> modes = getModes();
        std::list<std::string>::iterator i;

        cout << "Modes xbee disponibles : ";
```



```
        for (i = modes.begin(); i != modes.end(); i++)
        {
            cout << "  " << *i;
        }
        cout << "\n";
    }
    catch (xbee_err ret)
    {
        cerr << "Erreur récupération modes xbee !\n";
    }
}

void listerConnexions(XBee &xbee)
{
    try
    {
        std::list<std::string> types = xbee.getConTypes();
        std::list<std::string>::iterator i;

        cout << "Connexions disponibles : ";
        for (i = types.begin(); i != types.end(); i++)
        {
            cout << "  " << *i;
        }
        cout << "\n";
    }
    catch (xbee_err ret)
    {
        cerr << "Erreur récupération connexions xbee !\n";
    }
}

int main(int argc, char *argv[])
{
    int i;

    setbuf(stdout, NULL);

    //listerModes();

    try
    {
        XBee xbee("xbee1", "/dev/ttyUSB0", 9600);
        cout << "Mode sélectionné : " << xbee.mode() << "\n";

        //listerConnexions(xbee);

        struct xbee_conAddress adresse;
        memset(&adresse, 0, sizeof(adresse));

        /*adresse.addr64_enabled = 1;
        adresse.addr64[0] = 0x00;
        adresse.addr64[1] = 0x13;
        adresse.addr64[2] = 0xA2;
        adresse.addr64[3] = 0x00;
        adresse.addr64[4] = 0x40;
        adresse.addr64[5] = 0xC1;
        adresse.addr64[6] = 0x72;
        adresse.addr64[7] = 0x5A;
        ConnexionXbee connexionXbee(xbee, "64-bit Data", &adresse);*/

        adresse.addr16_enabled = 1;
        adresse.addr16[0] = 0x50;
        adresse.addr16[1] = 0x01;
        ConnexionXbee connexionXbee(xbee, "16-bit Data", &adresse);
```



```
        for (i = 0; i < 500; i++)
        {
            printf(".");

            /* XBee Series 1 modules don't use meshing, so you can broadcast much faster than Series 2 */
            usleep(100000); /* 100ms */
        }
    }
    catch (xbee_err err)
    {
        cerr << "Erreur " << err << " !\n";
    }

    return 0;
}
```

Codes sources

Code source : src-xbee-usb.zip

