Instituto Universitario Aeronáutico

Especialización en Sistemas Embebidos

Entornos Inalámbricos

Trabajos Prácticos

Alumno:
Luis Alberto Guanuco
Santiago Nicolás Nolasco
Sebastian Agüero
Franco Bocalon

Docentes:
Víctor Frison
Julio Echevarría
José Ducloux

Septiembre 2016

Índice

1. Introducción

Se realizarán ejercitaciones sobre plataformas XBee con el objetivo de entender los *Entornos Inalámbricos*. Se trabajará con los modos *AT* y *API* a fin de comprar las ventajas y desventajas de estos. Estos ejemplos prácticos servirán, además, alcanzar la resolución del *Trabajo Final*.

1.1. Módulos XBee S2C

Para el establecimiento de una red basado en los estándares 802.15.4 se plantea como requisito la disponibilidad de módulos que implementen dicho estándar. En nuestro caso se utilizarán los módulos *XBee S2C*[1]. Las principales características de estos dispositivos son:

- Velocidades de datos
  - RF 250 Kbps
  - Comunicación serial hasta 1 Mbps
- Alcances
  - *Indoor* 60 metros
  - *Outdoor* 1200 metros
- Potencia de transmisión 3.1 mW (+5 dBm). En modo *Boost* 6.3 mW (+8 dBm)
- Sensibilidad de recepción -100 dBm. EN modo *Boost* -102 dBm
- Banda de frecuencia 2.4 GHz (16 canales)
- 15 puertos digitales I/O (4 entradas analógicas)
- Alimentación 2.1V a 3.6V. Consumos:
  - En la transmisión 33 mA @ 3.3 VDC. En modo *Boost*45 mA
  - En la recepción 28 mA @ 3.3 VDC. En modo *Boost* 31 mA

Los puntos anteriores solo describen las principales características de los módulos XBee 2SC. Para conocer más en detalle se puede acceder a la hoja de datos disponible en el sitio web del fabricante [4].

1.2. Plataformas adicionales

Se utilizará *hardware* adicional que permita establecer la comunicación de los módulos XBee con una computadora. Sí bien el Laboratorio de Sistemas embebidos pone a disposición las plataformas XBoard[2] se necesita adaptar la comunicación serial del módulo XBee con la la computadora mediante una puerto USB. Para esto se desarrolló la placa *XBee-MCP Adapter* (Figura 1). Sobre esta placa se montarán dos módulos, una

[1] http://www.digi.com/support/productdetail?pid=4838
[2] http://www.cika.com/soporte/Information/Digi-RF/XBee/

es el módulo XBee y el otro es la placa *MCP2200 Breakout Module*[3]. Este último proporciona una comunicación serial USB-UART. Además se agregaron dos LEDs conectados a los puertos `AD1/DIO1` y `AD2/DIO2`. Opcionalmente se puede conectar un potenciómetro al conector `K1` que se encuentra mapeado al puerto analógico de la XBee, `AD0/DIO0`.

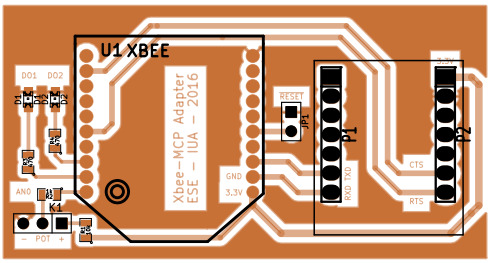


Figura 1: Placa XBee-MCP Adapter.

La Xboard, anteriormente mencionada, es una plataforma desarrollada por la empresa Cika Electrónica SRL. La XBoard permite interconectar los módulos XBee con dispositivos externos [5]. En la Figura 2 se presenta una vista superior de la XBoard. Se puede ver que esta placa tiene un conector hembra con las señales de comunicación serial, niveles de tensión y otras más. Para conectar la XBoard a una computadora se utiliza la placa MCP2200 Breakout Module. Al no requerir componentes externos, se utilizan simplemente cables que realicen la interconexión de las señales de comunicación serial.

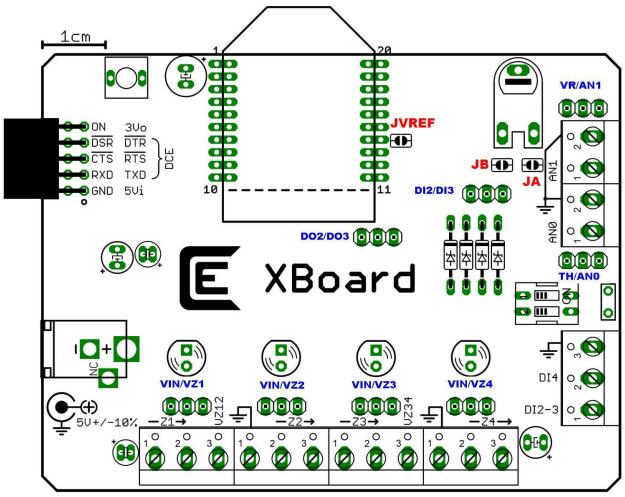


Figura 2: Placa XBoard.

Todas las características y configuraciones de las diferentes plataformas comerciales se encuentran en la bibliografía referenciada al final de este documento.

Se desarrollarán tres tipos de ejercitaciones. La primera sobre los alcances de RF que proporcionan estos dispositivos. La segunda parte sobre el manejo de los módulos en modo AT. Y por último se trabajará en modo API con los módulos sobre una red ZigBee

[3] http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=adm00393

completa. Se documentarán todos las implementaciones y por último se presentará una conclusión sobre lo hecho.

2. Cálculo de enlace y modelos de propagación

El diseño de un radioenlace implica toda una serie de cálculos que pueden resultar sencillos o complicados, dependiendo de las caractéristicas del sistema y del tipo de problema al que nos enfrentemos.

Es por ello que podemos dividir la propagación de la señal de acuerdo al entorno donde esta viaje:

- Espacios abiertos
- Entornos cerrados

2.1. Distancia máxima en espacios abiertos

La comunicación "outdoor" en este problema se asume en un entorno de propagación libre, donde no existen pérdidas atmosféricas, de polarización y de desadaptación de impedancias, es decir, operamos en regiones descubiertas.

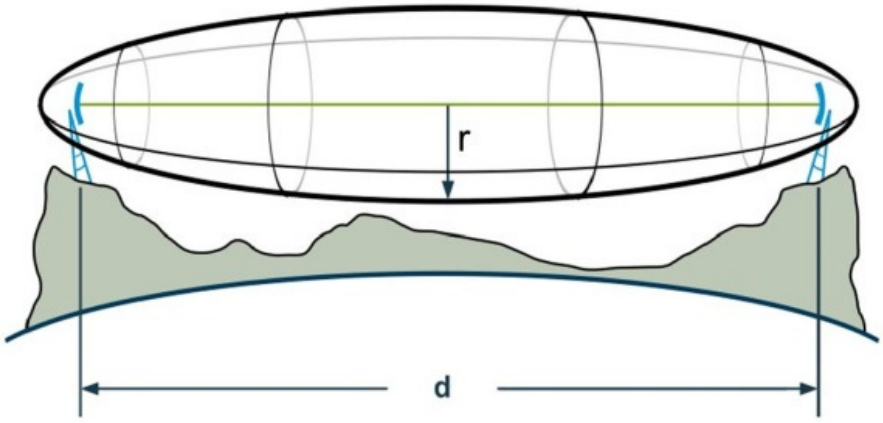


Figura 3: Primer zona de Fresnel

Modo de funcionamiento	**Normal**	**Boost**
Potencia de Tx	+5dBm(3.1mW)	+8dBm(6.3mW)
Sensibilidad de Rx	-100dBm	-102dBm

Cuadro 1: Technical review Xbee S2C

Para los cálculos siguientes se tomarán los datos de funcionamiento en modo "Normal".

Partiendo de la ecuación de Friis[1, Pág. 48]

$$Pr = \frac{PtGtGr\lambda^2}{(4\pi d)^2}$$

Siendo Pr (Potencia recibida), Pt(Potencia transmitida), Gt(Ganancia de antena tx), Gr(Ganancia de antena rx), λ(longitud de onda) y d(distancia radial entre antenas).

Podemos despejar la atenuación de espacio libre, también conocida como pérdida de trayectoria[2, Pág. 32].

$$P_{Loss}[dB] = 94{,}4dB + 20\log_{10} d[Km] + 20\log_{10} f[GHz] - Gt[dBi] - Gr[dBi]$$

Y de esta despejar la máxima distancia(adicionamos -5dB de atenuación, por margen)

$$d[Km] = 10^{\frac{P_{Loss}+Gt+Gr-20\log_{10} f-92{,}4dB-5dB}{20}}$$

$P_{Loss} = Gt[dBm] - Gr[dBm] = +5dBm - (-100dBm) = 105dBm$ $f = 2{,}4[GHz]$ Las ganancias Gr[dBi] y Gr[dBi] se toman como valor cero por ser antenas omnidireccionales.

$$d[Km] = 10^{\frac{105dBm-20\log_{10} 2{,}4-92{,}4dB-5dB}{20}} = 0{,}999[Km] = 999m$$

Con la máxima distancia d del radioenlace, calculamos la altura r de las antenas. Para ello nos valemos de las fórmulas del primer elipsoide de Fresnel, esto es determinando la zona libre de obstaculos[2, Pág. 40].

$$r = F1(m) = 17{,}32\sqrt{\frac{(d[Km]/2)^2}{d[Km]f[GHz]}} = 17{,}32\sqrt{\frac{(0{,}99Km/2)^2}{0{,}99Km2{,}4GHz}} = 7{,}28m$$

Por lo tanto la altura de las antenas, es decir para establecer una comunicación punto a punto a distancia máxima $d = 999m$ entre 2 XBee es de $r = 7{,}28m$ en la zona libre de obstaculos.Además este es un valor muy cercano al valor$d = 1200m$ dado por la hoka de datos de la XBee.

2.2. Distancia máxima en entornos cerrados

La propagación "indoor" difiere respectos a los sistemas "outdoor". Para asegurar una eficiente comunicación interior, la ITU a llevado una serie de propuestas para el caso de comunicaciones punto a punto. Debido a que en una comunicación en entornos cerrados esta muy influenciada por la geometría del lugar y los objetos en ella. Tanto estos objetos y la construcción de la misma, ocasionan pérdidas por reflexiones, dispersión y absorción de las señales RF.

2.2.1. Cálculo Planta alta

Para el cálculo de máxima distancia, como lo indica la figura. Nos valdremos de la fórmula de "path loss"[3, Pág.4]

$$L_{Loss}[dB] = L_{do}[dB] + N\log_{10} d/do + Lf_n[dB]$$

Donde N(Coef. de pérdida), f(Frecuencia en Mhz),d(Distancia entre base y terminal), L_{do}(Pérdida a do), L_f(Atenuación través del piso), $d0$(distancia de ref=1m) y n(nro de pisos entre terminal y base). En nuestro caso particular al igual que en el caso anterior, calculamos la máxima atenuación: $L_{Loss} = Gt - Gr = +5dBm - (-100dBm) = 105dBm$ $N = 28$ Por dato de tabla [3, Pág 4-5](Espacio residencial a 2.4GHz). $L_{do} =$

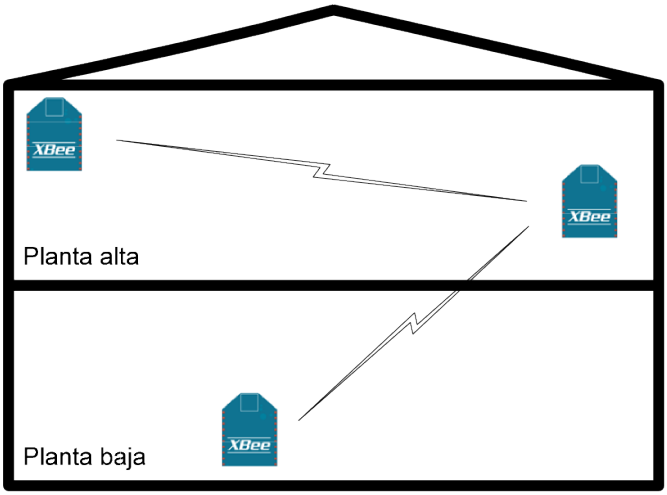


Figura 4: Comunicación indoor

$20\log_{10} f[MHz] - 28 = 39{,}6dB$ $Lf_n = 0$ Por ser un mismo piso. Despejando d, incluyendo una atenuación adicional de 10dB por la cantidad de objetos que puedan influir en la comunicación.

$$d[m] = 10^{\frac{L_{Loss}[dB] - L_{do}[dB] - L_f - 10dB[dB]}{N}} d[m] = 10^{\frac{105dBm - 39{,}6dB - 0dB - 10dB}{28}} = 95{,}18m$$

Entonces en un piso la distancia máxima de transmisión es $d = 95{,}18m$

2.2.2. Cálculo de comunicación entre Planta alta y baja

En este caso la máxima distancia d, será influenciada por la atenuación del piso, como separaci"on de los dos ambientes. Por lo tanto $L_f[dB] = 5$ dado por el cuadro, adicionamos una atenuación de 10dB . La distancia máxima será.

$$d[m] = 10^{\frac{L_{Loss}[dB] - L_{do}[dB] - L_f[dB]}{N}} d[m] = 10^{\frac{105dBm - 39{,}6dB - 5dB - 10dB}{28}} = 63{,}09m$$

Se puede notar aquí, que el valor de d distancia máxima es reducido por esta atenuación. Siendo el valor de $d = 63{,}09m$ para comunicaciones entre dos pisos. Lo que es comparable al $d = 60m$ dado por la hoja de datos.

3. Ejercitación en modo AT

El set de comandos AT, también conocidos como set de comandos Hayes, fue originalmente desarrollado para usar con los modem de Hayes en la década de 1980. Muchos modems modernos todavía utilizan este estándar. EL término *comando AT* viene de usar los caracteres ASCII para notificar al *host* que un le sigue un comando.

En el caso de los módulos XBee desarrollados por Digi, implementa un set de comandos propieratrios para interactuar con los módulos de Digi a través de una comunicación serial. Basado en el set de comandos AT, Digi usa los caracteres `AT` antes de cada comando enviado al modem. Un ejemplo de un comando utilizado por el radio Digi XBee

podría ser `ATCH`. Este comando es usado para leer o setear el canal que un radio XBeee es configurado[9].

3.1. Configuración de puertos GPIO

La configuración de los diferentes canales de propósitos generales se puede realizar utilizando el software XCTU en modo gráfico pero el objetivo es realizarlo mediante el set de comandos AT.

3.1.1. Comando ATIS

El comando `ATIS` fuerza la lectura de todos los canales habilitados. Analógicos como digitales. En nuestro caso tenemos la siguiente salida a la petición.

```
1 +++
2 OK
3 ATIS
4 01
5 0006
6 01
7 0000
8 0219
```

La respuesta que entrega el módulo comienza en la línea **4** hasta la **8**. El primer byte `01` es la cantidad de muestra recibidas. En la línea **5** se tiene configuración de los canales digitales y la línea **6** canales analógicos. Para entender se debe analizar a nivel de bits cada una de las respuestas. Recuerde que los valores representados están en hexadecimal. En el primer se tiene $0006_{HEX} = 000000000000110_{BIN}$ por lo los canales `DIO1` y `DIO2` se encuentran habilitados y configurados como salidas digitales. Mientras que para los canales analógicos tenemos $01_{HEX} = 0001_{BIN}$ solo el puerto `AD0` habilitado. Para terminar el análisis de la respuesta al comando `ATIS`, las líneas **7** y **8** son el estado de los canales digitales y analógicos respectivamente. Como se mencionó anteriormente, puede visualizarse la configuración de los canales de I/O en forma gráfica desde el software XCTU. En la Figura 5 se puede ver la misma información que proporciona el comando `ATIS`.

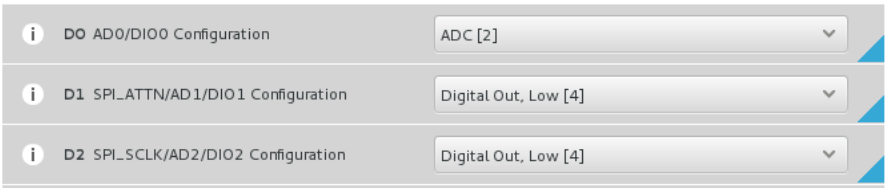


Figura 5: Configuración de los GPIO en modo gráfico.

Para dar comienzo con la ejercitación pedida por los docentes, se deshabilitarán todos los canales de forma tal que el módulo quede sin ningún GPIO en uso. Sí la configuración es tal como la descrita anteriormente, se deben aplicar los comandos siguientes.

```
+++OK
ATD00
OK
ATD10
OK
ATD20
OK
ATAC
OK
ATWR
OK
ATIS
ERROR
```

Para deshabilitar un canal digital/analógico se debe simplemente pasar como último argumento el valor `0`. Esto se aplica a los canales `DIO0`, `DIO1` y `DIO2`. Los comandos que le siguen son para aplicar los cambios y liberar los *buffers* (`ATAC`) y escribir la memoria no-volatil del módulo (`ATWR`). Al finalizar los cambios se envía el comando `ATIS` y se tiene como respuesta `ERROR` esto no se debe a un problema de comunicación sino que al no existir ningún canal habilitado, no puede ofrecer información alguna.

A continuación se presentan diferentes configuraciones sobre el módulo XBee montado en la placa XBoard.

3.1.2. Configurar canales analógicos

Código 1: Canales analógicos.

```
1  +++OK
2  ATD02
3  OK
4  ATD12
5  OK
6  ATAC
7  OK
8  ATWR
9  OK
10 ATIS
11 01
12 0000
13 03
14 0268
15 0208
```

En el código 1 se configuran los canales `DIO0` y `DIO1` como analógicos. El último parámetro define este comportamiento. Luego de guardar la configuración se envía el comando `ATIS` para tener la configuración final de todos los canales. Aquí se puede ver en la línea **13** no se tiene ningún canal digital. y en las líneas **15** y **16** se ven los valores de los ADC correspondientes a los canales `DIO0` y `DIO1`.

3.1.3. Configurar canales digitales

En el caso del código 2 se configuran los canales `DIO2` y `DIO5` como salida. La elección de estos canales se debe al circuito implementado por la placa XBoard. El parámetro adicional en los comandos **3** y **5** es el estado digital que se le asignará. El número `4` aplica un valor *bajo* mientras que el valor `5` define un estado en *alto*. Al igual que en el caso anterior, el comando `ATIS` muestra los estados de los puertos habilitados, en este caso no tenemos entradas analógicas por lo tanto en la línea **14** tenemos 0.

El código 3 muestra la configuración de dos canales digitales de entrada. La diferencia con el caso anterior es el parámetro asignado. En las líneas **3** y **5** el parámetro es `3` que configura los canales `DIO2` y `DIO4` como entradas digitales.

Código 2: Dos salidas en alto.

```
+++OK
ATD25
OK
ATD55
OK
ATAC
OK
ATWR
OK
ATIS
01
0024
00
0024
```

Código 3: Dos entradas

```
+++OK
ATD23
OK
ATD43
OK
ATAC
OK
ATWR
OK
ATIS
01
0014
00
0014
```

3.1.4. Configuración combinada de la XBoard

Código 4: GPIO de la placa XBoard.

```
+++OK
ATD02
OK
ATD12
OK
ATD25
OK
ATD55
OK
ATD33
OK
ATD43
OK
ATAC
OK
ATWR
OK
ATIS
01
003C
02
003C
0264
```

Finalmente configuraremos la placa XBoard con todos los modos anteriormente vistos. Los comandos aplicados desde la línea **2** hasta la **12** ya fueron descritos. Como respuesta al comando `ATIS` se puede ver todos los canales digitales habilitados (`003C`). Mientras que se tiene dos canales analógicos `02` y sus respectivos valores en las líneas **22** y **23**.

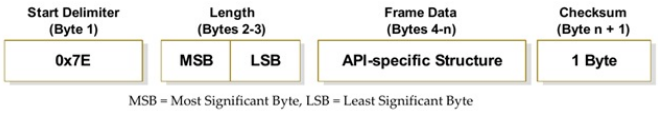
4. Ejercitación en modo API

4.1. Introducción al modo API

En modo API el nivel de programación es mayor, pero provee de mayor "flexibilidad" al momento de realizar el envío y recepción de datos. A diferencia del modo transparente, en este modo el módulo XBee espera por una secuencia específica de bytes que le indican que tipo de operación debería de realizar. Cada uno de estas secuencias se le conoce como "frame" o también paquetes. Y dependiendo del contenido de cada paquete se llamará a una función u otra de la API.

De igual manera, el módulo XBee responde con secuencias específicas dependiendo si lo que está reportando es un cambio en el estado del módem, o el estado de la transmisión si este esta recibiendo un paquete de datos desde un nodo remoto.

Cada paquete presenta la siguiente estructura: Donde cada parte corresponde a:



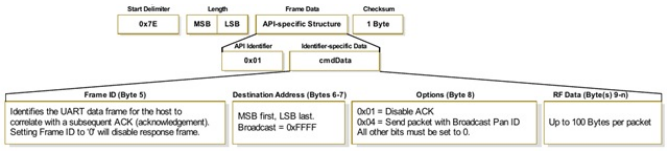
1. Byte 1: El encabezado que siempre tiene un valor de 0x7E, esto indica al módulo que está por comenzar la transmisión de un paquete.

2. Bytes 2, 3: Esta sección indica el tamaño del paquete, el tamaño se calcula contando los bits entre el byte menos significativo del tamaño y el byte de la suma de verificación (checksum).

3. Bytes 4-n: Esta sección contiene la estructura específica a la API, de los contenidos de esta parte del paquete dependerá la función de la API que se llame.

4. Byte n+1: La suma de verificación de todos los bytes contenidos en el paquete. Esto se utiliza para verificar la integridad de los datos recibidos, si la suma no coincide el paquete es descartado.

El contenido del paquete de datos varía en base al comando que se quiera transmitir, por ejemplo si quisiéramos transmitir un dato a otro XBee utilizando una dirección de 16 bits deberíamos utilizar la siguiente estructura:



Configuración Frame ID a '0' desactivará trama de respuesta. Donde la estructura tiene la siguiente forma:

1. Byte 4:Identificador de la API, este código identifica el tipo de comando que queremos ejecutar, para este caso el código 0x01 corresponde a Transmit Request: 16-Bit Address" o solicitud de transmisión a una dirección de 16 bits.

2. Byte 5:Corresponde a un número secuencial del paquete. Para cada envío se recibe un acuse de recibo" (ACK) este número nos servirá luego para verificar que un paquete que enviamos fue recibido por el destinatario.

3. Bytes 6, 7:Corresponden a la dirección del XBee destinatario, se coloca primero el byte más significativo y luego el menos significativo.

4. Byte 8:Corresponde a las opciones de envío, si establecemos el bit 0x01 en 1 no se esperará un ACK, mientras que si habilitamos el bit 0x04 se enviará el mensaje con broadcast para todas las redes de área personal (PAN).

5. Byte 9-n:Datos a enviar (máximo 100 bytes).

4.2. El modo API 1 y API 2

Según la documentación del XBee existen dos modos API, la diferencia entre el modo 1 y el modo 2 es que en el Modo 2 las secuencias de bytes van "escapadas".

¿Qué significa esto? Resulta que el módulo XBee intentará leer el inicio de un paquete siempre encuentre el byte 0x7E, ¿pero qué pasa si este valor se está enviando en algún lugar dentro del paquete? Por ejemplo en el tamaño, o en la misma secuencia de datos.

Pues lo que ocurrirá será que el XBee va a descartar el paquete que estaba procesando y comenzará a procesar el nuevo de manera errónea. Para solventar este problema se incluyó el Modo API 2, en este modo antes de enviar un dato verificamos si el byte posee cualquiera de los siguientes valores:

- 0x7E: Inicio del paquete.
- 0x7D: Carácter de escape.
- 0x11: Señalizador XON.
- 0x13: Señalizador XOFF.

Para escapar la secuencia tenemos que enviar el carácter de escape más el dato que queremos enviar haciendo una operación XOR con el valor 0x20. Por ejemplo, la secuencia del siguiente paquete:

0x7E 0x00 0x02 0x23 0x11 0xCB

Quedaría escapado de la siguiente manera:

0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB

4.3. Modo API

Para habilitar el modo API debemos enviar la siguiente secuencia de comandos a nuestra XBee:

4.3.1. Modo API 1:

- Enviamos "+++" y esperamos confirmación "OK".
- Enviamos "ATAP1" y esperamos confirmación "OK"
- Enviamos "ATCN" y esperamos confirmación "OK"

4.3.2. Modo API 2:

- Enviamos "+++" y esperamos confirmación "OK".
- Enviamos "ATAP2" y esperamos confirmación "OK"
- Enviamos "ATCN" y esperamos confirmación "OK"

Una vez el XBee se encuentre en modo API, ignorará todos los datos que no muestren la secuencia correspondiente a una estructura de la API. Esto resulta muy útil ya que podemos imprimir sobre la misma línea Serie donde está conectado el XBee mensajes para "depurar" el funcionamiento y el XBee simplemente los ignorará.

Tomen en consideración que los datos recibidos del XBee también presentarán la estructura de la API así que leer los datos requerirá realizar el proceso detallado anteriormente pero a la inversa.

4.3.3. Ejemplo de mensaje

Si quisiéramos enviar el texto "Hello World!" en un broadcast, deberíamos enviar la siguiente secuencia de bytes al XBee:

```
1  0x7E: Encabezado
2  0x00:
3  0x11: Tama\~no en Bytes del paquete (17)
4  0x01: Identificación del comando de la API (0x01 para enviar
       a dirección de 16 bits)
5  0x01: Identificación del paquete (cualquier número entre 0
      x01 y 0xFF)
6  0xFF: Byte más significativo dirección destino
7  0xFF: Byte menos significativo dirección destino
8  0x00: Opciones, ‘‘0’’ utiliza opciones por defecto.
9  0x48: ’H’
10 0x65: ’e’
11 0x6C: ’l’
12 0x6C: ’l’
13 0x6F: ’o’
14 0x20: ’ ’
15 0x57: ’W’
16 0x6F: ’o’
17 0x72: ’r’
18 0x6C: ’l’
19 0x64: ’d’
20 0x21: ’!’
21 0xC3: Suma de verificación
```

4.3.4. La suma de verificación

El último byte, la suma de verificación, se utiliza para garantizar que el paquete se haya transmitido de manera íntegra, lo calculamos restando a 0xFF la suma truncada

a 8 bits de todos los bytes entre el tamaño del paquete y el checksum, para el ejemplo anterior la fórmula del checksum sería:

0xFF- 0xFF &(0x01+ 0xFF+ 0xFF+ 0x00+0x48+ 0x65+ 0x6C+ 0x6C+0x6F + 0x20 + 0x57+ 0x6F+0x72 +0x6C + 0x64+0x21) = 0xC3

Utilizando el software XCTU y comandos AT en modo API, configurar dos entradas como canales de conversión AD. Utilizar el comando ATIS para leer las entradas. Describir el significado de la información devuelta por el modulo luego del envió de este comando. Al igual que en el modo AT, las configuraciones de los puertos son simila-

Fig. 7. Manejo del módulo XBee ZB en modo API.

res, con la salvedad de que debemos enviar el comando AT dentro de una trama API especialmente diseñada para enviar comandos AT.

El software XCTU cuenta con, entre otras herramientas, con un frame generator y un frame interpreter especialmente diseñados para poder crear e interpretar tramas API.

Estas dos utilidades se muestran a continuación

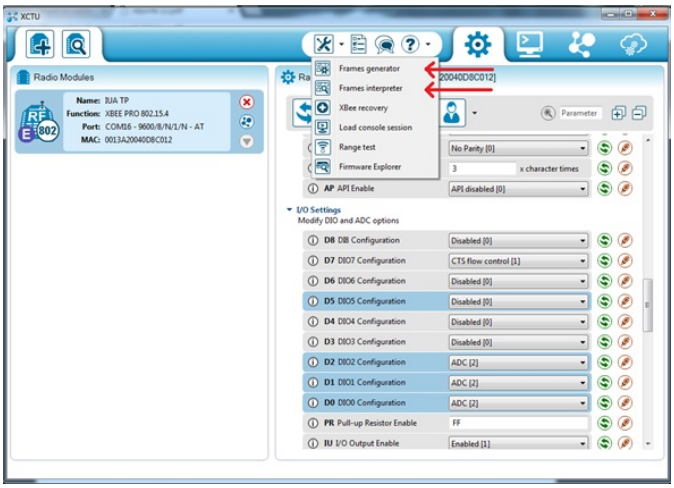


Figura 6: Ubicación del frame generador e interprete dentro de XCTU

A continuación veremos cómo utilizar el frame generator: Dentro del marco de generación, podemos encontrar varias opciones:

- Tipo de protocolo: 802.15. Wi-Fi, XTend(digimesh), Xted(Legacy), ZigBee, DigiMesh, Point to multipoint, ZNet 2.5,Smart Energy, XLR.
- Modo: API1-Modo API no escapado, API2-Modo API escapado.
- Tipo de frame: Dependiendo de lo que vayamos a transmitir, 2 bytes con opciones desde 0x00h a 0xFFh.

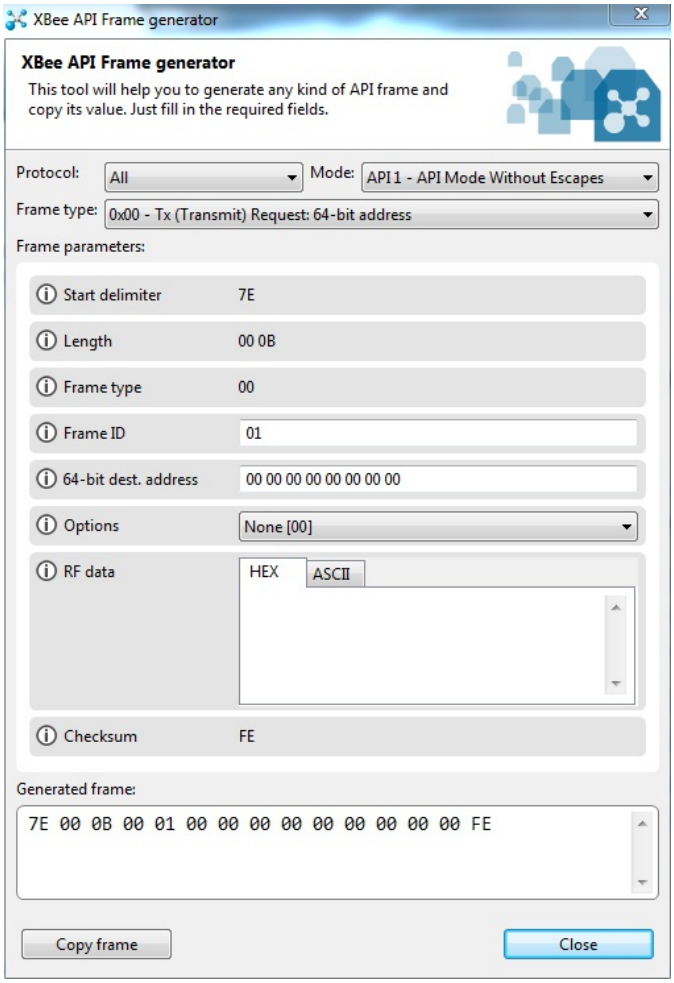


Figura 7: "Frame generator" de XCTU

Además, tenemos una lista de todos los bytes que componen la trama API, explicando cada uno de ellos, como la sig. Al colocar todos los parámetros pertinentes para crear la trama API, así como el payload que vamos a enviar, ya sea este un comando AT o caracteres ASCII, podremos ver el frame generado en la parte inferior del frame generator y podremos copiarlo al portapapeles con el botón "copy frame" Este frame que copiamos, es el que vamos a utilizar en la consola para poder enviar informacion. Acto seguido presionamos el botón "+" verde y veremos la siguiente pantalla. Dentro de esta pantalla podemos ver que tenemos acceso al Frame generator, con lo cual como antes vimos, crearemos nuestra trama API, en este caso creamos la trama API de comandos AT, para enviar el comando "ATD00", que nos sirve para deshabilitar el puerto D0 por ejemplo.

Enviamos el frame con el botón verde "Send selected frame".

En la parte derecha de la consola podremos ver los detalles de cada frame como "AT Command" o "AT Command response" donde sabremos si el comando tuvo efecto sobre el puerto en cuestión.

Podemos copiar esa respuesta en el portapapeles con el botón "copy packet information" y lo vemos como sigue:

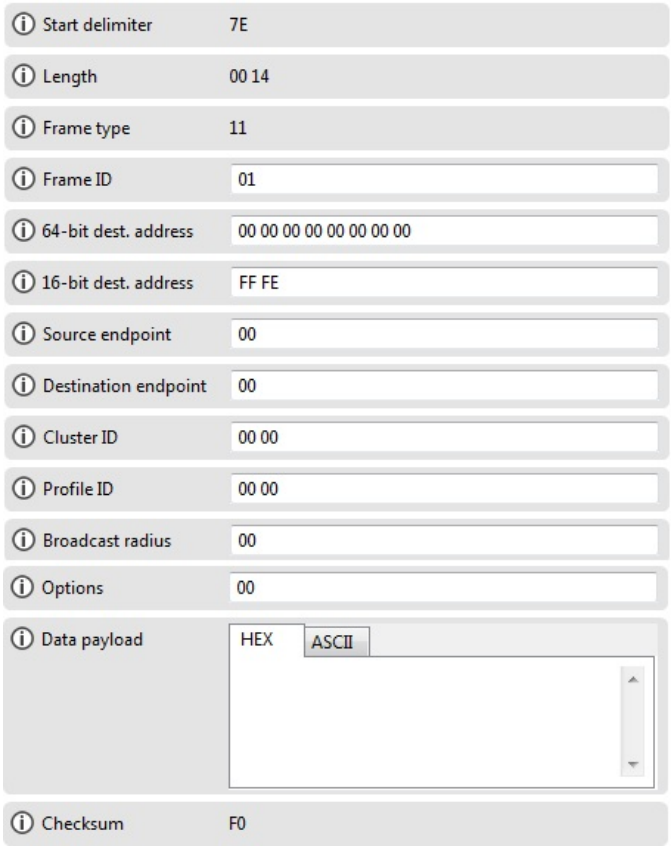


Figura 8: Trama API dentro del marco generador.

```
AT Command Response (API 1)
7E 00 05 88 01 44 30 00 02
Start delimiter:
7E
Length:
00 05 (5)
Frame type:
88 (ATCommandResponse)
Frame ID:
01 (1)
AT Command:
44 30 (D0)
Status:
00 (Status OK)
Checksum:
02
```

En el ejemplo anterior vimos como desactivar una entrada del GPIO, ahora como se solicitó en el práctico mostraremos la secuencia para configurar dos entradas como canales de conversión AD. Y luego leeremos la información con el comando ATIS. Vamos a generar una lista de tramas para enviar desde la consola API del XCTU y de ese modo

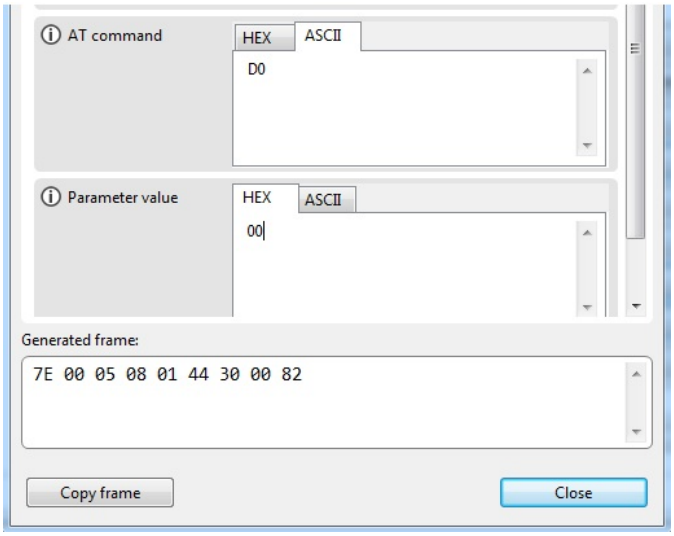


Figura 9: Frame generado con el Frame Generator.

poder configurar los puertos como se pidió. Los comandos que vamos a enviar en tramas API serán los siguientes:

```
ATD02: Configuramos la entrada D0 como puerto analógico.
Trama API: 7E 00 05 08 01 44 30 02 80
ATD12: Configuramos la entrada D1 como puerto analógico.
Trama API: 7E 00 05 08 01 44 31 02 7F
ATWR: Guardamos los cambios en la memoria.
Trama API: 7E 00 04 08 01 57 52 4D
ATAC: Aplica los cambios al módulo.
Trama API: 7E 00 04 08 01 41 43 72
```

En la esquina inferior derecha veremos un botón verde “Start sequence”, este botón envía los 4 frames directamente y guarda los cambios ocupando los comandos anteriormente mencionados.

Ahora vamos a configurar dos entradas digitales. Los comandos que vamos a enviar en tramas API serán los siguientes:

```
ATD03: Configuramos la entrada D0 como puerto analógico.
Trama API: 7E 00 05 08 01 44 30 03 7F
ATD13: Configuramos la entrada D1 como puerto analógico.
Trama API: 7E 00 05 08 01 44 31 03 7E
ATWR: Guardamos los cambios en la memoria.
Trama API: 7E 00 04 08 01 57 52 4D
ATAC: Aplica los cambios al módulo.
Trama API: 7E 00 04 08 01 41 43 72
```

Ahora vamos a configurar dos salidas digitales. Los comandos que vamos a enviar en tramas API serán los siguientes:

```
ATD04: Configuramos la entrada D0 como puerto analógico.
Trama API: 7E 00 05 08 01 44 30 04 7E
ATD14: Configuramos la entrada D1 como puerto analógico.
```

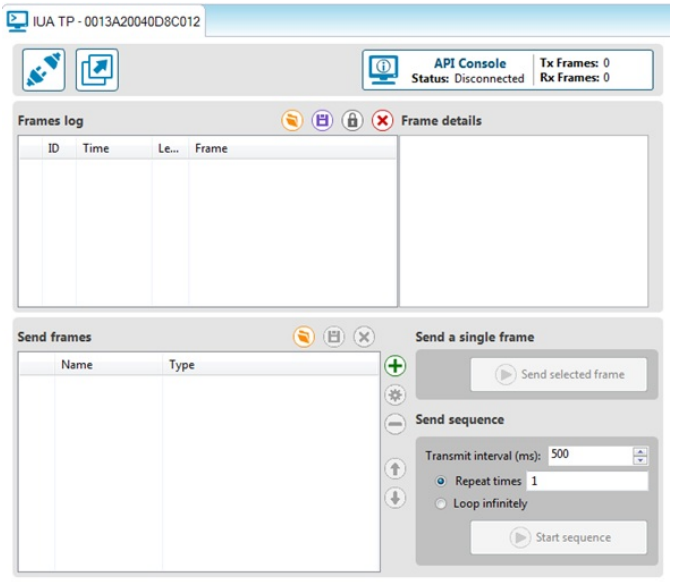


Figura 10: Consola de comandos API.

```
Trama API: 7E 00 05 08 01 44 31 04 7D
ATWR: Guardamos los cambios en la memoria.
Trama API: 7E 00 04 08 01 57 52 4D
ATAC: Aplica los cambios al módulo.
Trama API: 7E 00 04 08 01 41 43 72
```

Empleo del comando ATIS.

Para probar este comando vamos a configurar las dos primeras entradas analógicas y las dos siguientes digitales. Los comandos que vamos a enviar en tramas API serán los siguientes:

```
ATD02: Configuramos la entrada D0 como puerto analógico.
Trama API: 7E 00 05 08 01 44 30 32 50
ATD12: Configuramos la entrada D1 como puerto analógico.
Trama API: 7E 00 05 08 01 44 31 32 4F
ATD23: Configuramos la entrada D1 como puerto analógico.
Trama API: 7E 00 05 08 01 44 32 33 4D
ATD33: Configuramos la entrada D1 como puerto analógico.
Trama API: 7E 00 05 08 01 44 33 33 4C
ATWR: Guardamos los cambios en la memoria.
Trama API: 7E 00 04 08 01 57 52 4D
ATAC: Aplica los cambios al módulo.
Trama API: 7E 00 04 08 01 41 43 72
ATIS: se utiliza para forzar una lectura de todas las líneas
    ADC/DIO habilitadas
Trama API: 7E 00 04 08 12 49 53 49
```

Luego del envió de la secuencia de tramas para configurar el GPIO enviamos la trama con el comando ATIS y la misma nos devuelve lo siguiente:

```
RX (Receive) Packet 16-bit Address IO (API 1)
```

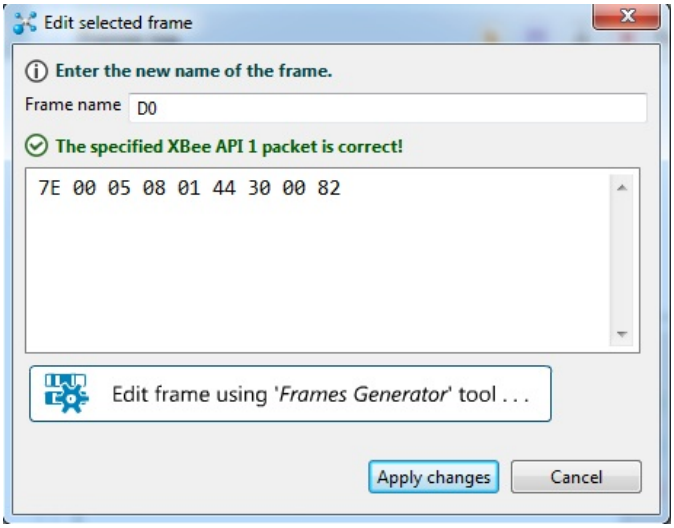


Figura 11: Agregando una nueva trama API a la lista de comandos

```
7E 00 0E 83 00 00 00 00 01 06 0C 00 0C 03 FF 03 FF 59
Start delimiter:          7E
Length:                   00 0E (14)
Frame type:  83 (RX(Receive)Packet16-bitAddressIO)
16-bit source address:    00 00
RSSI:                     00
Options:                  00
Number of samples:        01
Digital channel mask:     00 0C
Analog channel mask:      06 00
DIO2/AD2 digital value:   High
DIO3/AD3 digital value:   High
DIO0/AD0 analog value:    03 FF (1023)
DIO1/AD1 analog value:    03 FF (1023)
Checksum:                 59
```

El último Frame enviado, nos devuelve una respuesta que es el estado de las líneas activas, en este caso el estado de las dos entradas analógicas en 03FF (1023) y las dos entradas digitales en High.

5. Enlaces entre módulos XBee

Ya se mostraron en las secciones anteriores las diferentes formas de manipular los módulos XBee. En esta sección se desarrollarán pruebas de enlace de los módulos. Las características de radio frecuencia no son objeto de estudio, se busca evaluar las virtudes en la red que implementa el estándar 802.15.4. Se utilizará la topología más sencilla del alcance de las redes Zigbee. Se implementará un *Coordinador* y los demás serán configurados como *Dispositivos finales*. En la Figura 21 se presenta una red de módulos XBee en las que se puede ver las diferentes roles que pueden asumir estos dispositivos. A parte de los dos modos anteriormente nombrados se puede configurar en modo *router*

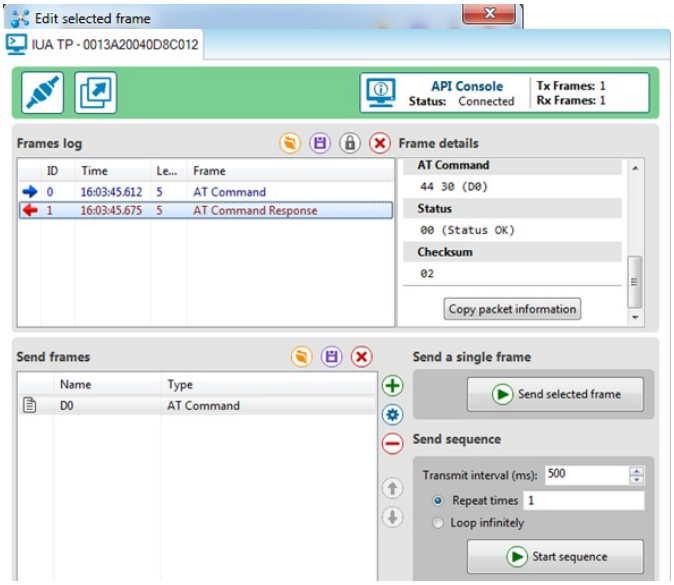


Figura 12: Envió del frame generado para deshabilitar el puerto D0

que permite extender la red.

5.1. Asignación de Identificador y Función de los módulos

Antes de comenzar con la configuración de la red, se definirá a uno de los módulos como coordinador y los demás como dispositivos finales. Además proporcionaremos un nombre para identificarlos a cada uno ellos. En la línea **4** del código 5 se habilita el modo coordinador y luego se asigna el nombre `COORDINADOR` al módulo (línea **8**). Como en los casos anteriores, seimpre se debe aplicar los cambios (`ATAC`) y finalmente escribir en la memoria no-volatil (`ATWR`).

Código 5: Obtención del *seral number*.

```
1  +++OK
2  ATCE
3  0
4  ATCE1
5  OK
6  ATCE
7  1
8  ATNICOORDINADOR
9  OK
10 ATNI
11 COORDINADOR
12 ATAC
13 OK
14 ATWR
15 OK
```

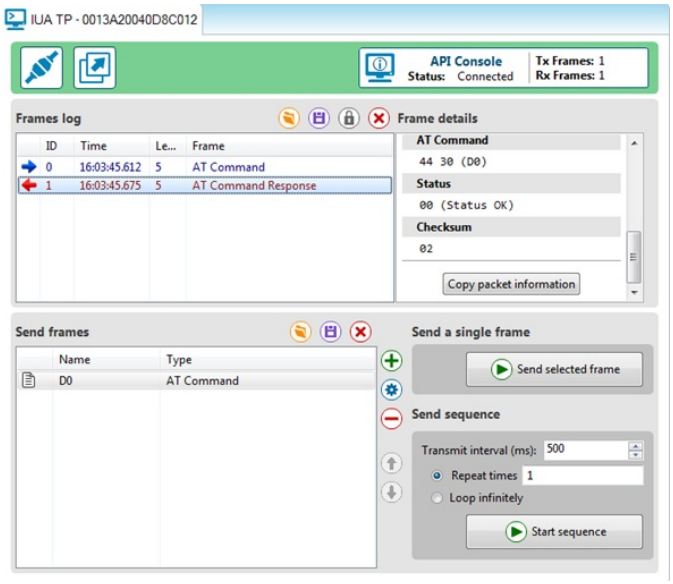


Figura 13: Entradas del GPIO antes de aplicar las tramas API.

5.2. Configuración de la red de área personal

Las redes ZigBee se las llaman PANs (*Personal Area Networks*). Cada red es definida con un único identificador PAN, por lo que todos los dispositivos que quieran pertenecer a la red deben tener el mismo `ID PAN`.

ZigBee soporta tanto PAN ID de 16 a 64 bits. Ambos PAN IDs son utilizados como único identificador de la red. Los dispositivos que quieran integrar la misma red deberán compartir el mismo PAN IDs.

En el código 6 se asigna el PAN ID `260816` al módulo coodinador y será el mismo que utilizarán los demás dispositivos para conectarse a la red ZigBee. En la línea **2** se solicita el actual PAN ID y el módulo responde `0`. Luego se aplica el identificador `260816`, se utilizan solo 24 de los los 64 bits disponibles.

Código 6: Obtención del *serial number*.

```
1  +++OK
2  ATID
3  0
4  ATID260816
5  OK
6  ATID
7  260816
8  ATAC
9  OK
10 ATWR
11 OK
```

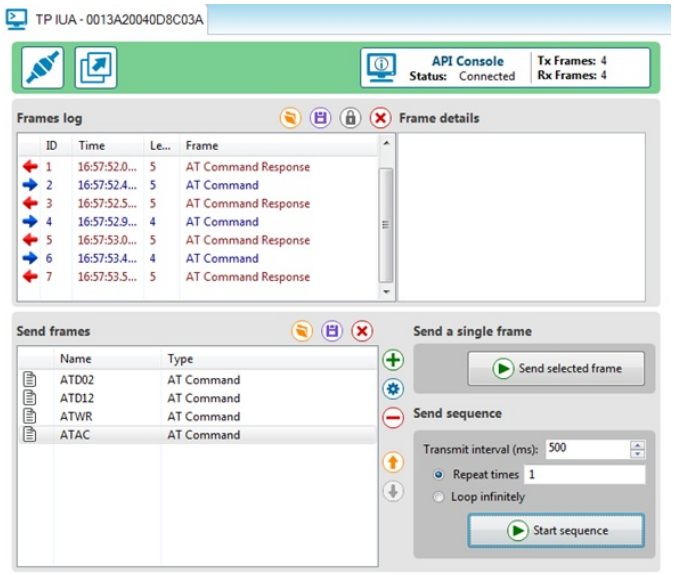


Figura 14: Envió de la secuencia de tramas para configurar el GPIO.

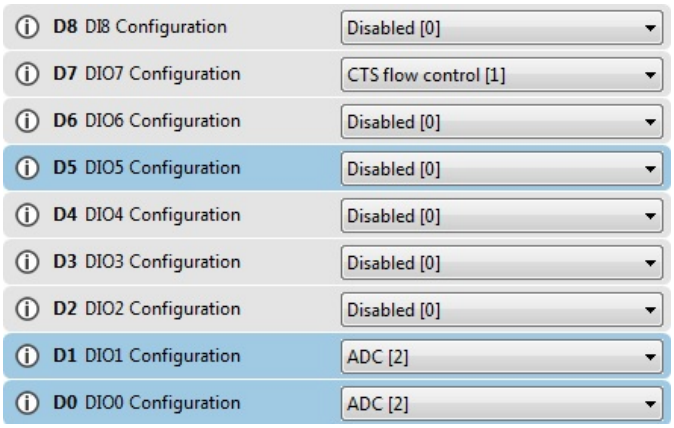


Figura 15: Estado del GPIO luego de la configuración.

5.3. Número de serie (identificador) los módulos

Cada módulo XBee tiene una número de serie de 64 bits (*extended address*) grabado en el *hardware* por el fabricante[4]. Mediante los comandos AT se puede obtener esta dirección en dos partes. Los comandos `ATSH` y `ATSL` devuelven la parte alta y baja respectivamente. Estos valores se utilizan para la configuración de la red Zigbee (junto a los valores de los demás módulos). En el código 7 se muestra cómo obtener el número de serie del dispositivo al que se encuentra conectado el terminal serial.

Código 7: Obtener el *serial number* del módulo XBee

```
1 +++OK
2 ATSH
3 13A200
```

[4]Recuerde que también los XBee tienen un direccionamiento de 16 bits. Para mayor información ver la bibliografía.

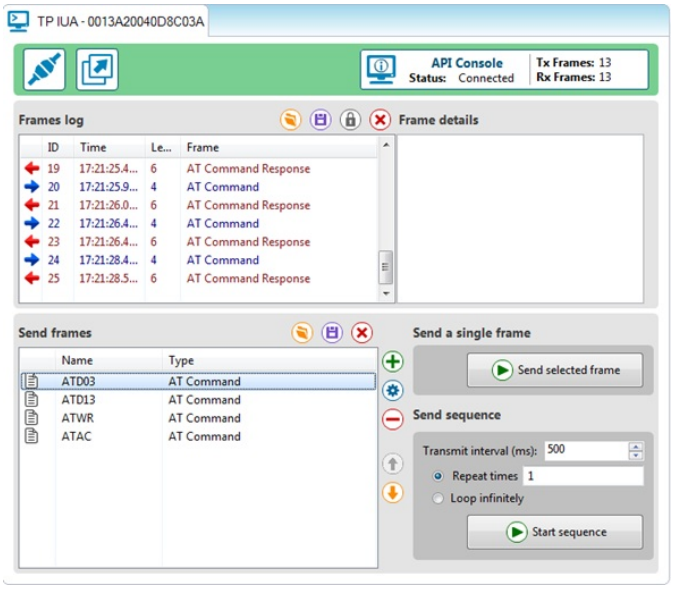


Figura 16: Envió de la secuencia de tramas para configurar el GPIO.

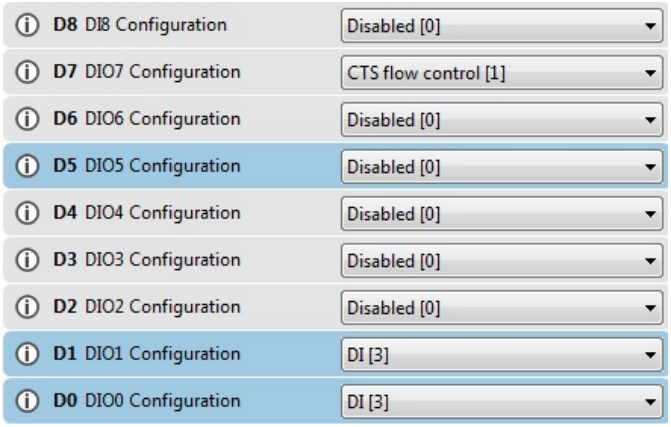


Figura 17: Estado del GPIO luego de la configuración.

```
4 ATSL
5 41257816
```

5.4. Configuración de direccionamiento de los dispositivos en una red ZigBee

En la sección anterior se mostró como obtener las identificaciones de cada módulo. Esta información se utiliza para mapear la red a implementar. Se recuerda que la topología utilizada en estos prácticos, se tendrá un coordinador y varios dispositivos finales que proporcionarán información al primero. En este caso el coordinador configura su dirección de destino al único (por el momento) dispositivo conectado, código 8. Se cargan la parte alta (`ATDH13A200`) y la baja (`ATDL4125768A`) con los valores propios del dispositivo final al que se conectará el coordinador. En forma recíproca se debe cargar el identificador del coordinador como dirección de destino del único dispositivo final

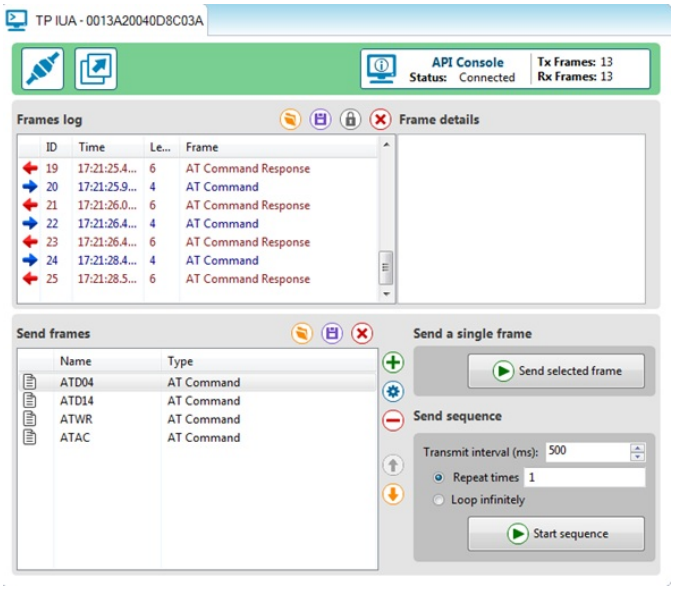


Figura 18: Envió de la secuencia de tramas para configurar el GPIO.

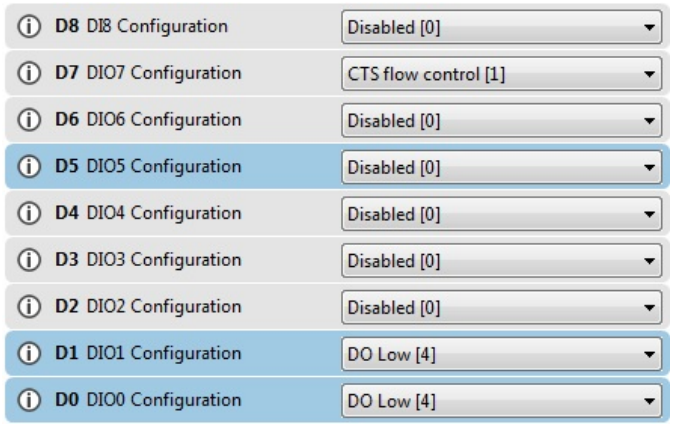


Figura 19: Estado del GPIO luego de la configuración.

conectado a la red. Esto último se muestra en el código 9. Para mejor entendimiento de los códigos de configuración se solicita antes a los módulos responder el nombre que identifica a los módulos y sus respectivos números de serie. Al final se guardan todos los cambios realizados.

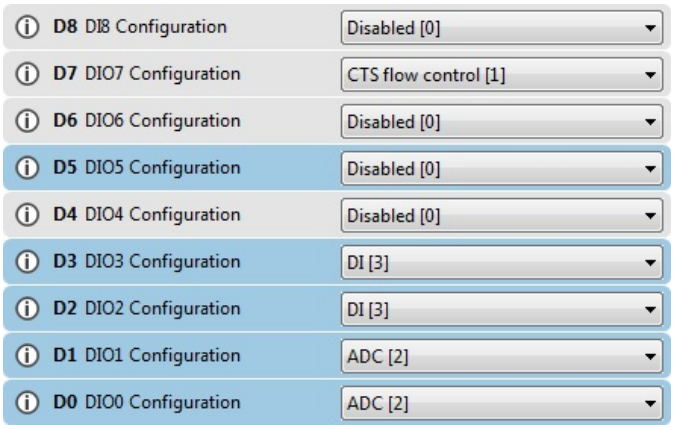


Figura 20: Estado del GPIO luego de la configuración.

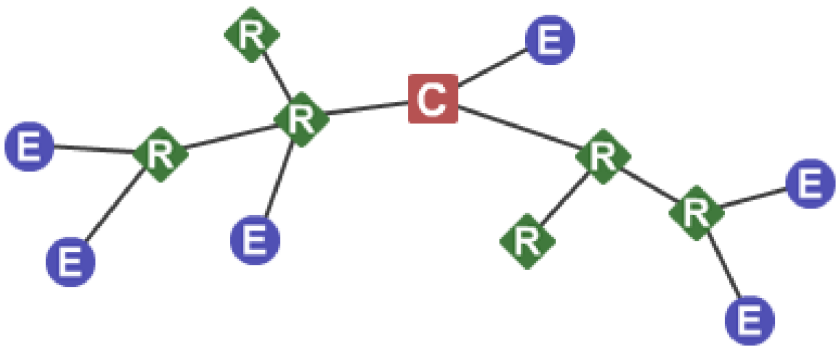


Figura 21: Red de módulos XBee con sus diferentes configuraciones.

Código 8: Destino del coordinador.

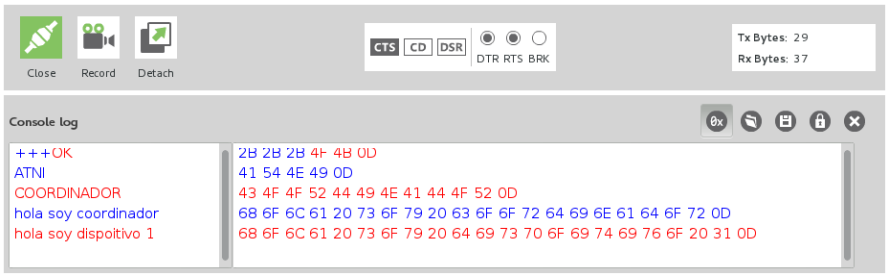
```
+++OK
ATNI
COORDINADOR
ATSH
13A200
ATSL
41257816
ATDH13A200
OK
ATDL4125768A
OK
ATAC
OK
ATWR
OK
```

Código 9: Destino del coordinador.

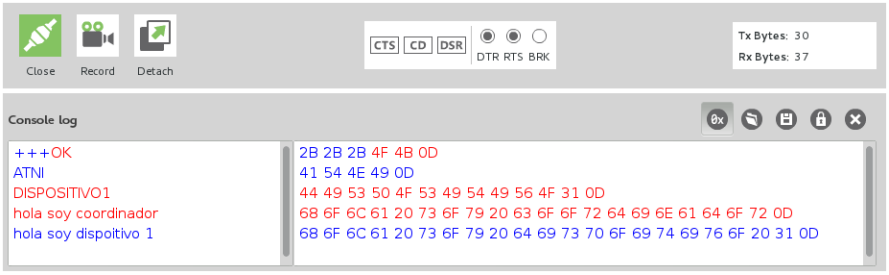
```
+++OK
ATNI
DISPOSITIVO1
ATSH
13A200
ATSL
4125768A
ATDH13A200
OK
ATDL41257816
OK
ATAC
OK
ATWR
OK
```

5.5. Comunicación entre computadoras mediante módulos XBee

Con los módulos configurados anteriormente para establecer una comunicación punto a punto, se probará el enlace entre los módulos comunicando ambos módulos entre sí mediante un terminal de computadora. Es decir, se utilizarán dos computadoras conectando sus respectivos puertos seriales (RS232) a cada módulo XBee ya configurados. A continuación se presentan las capturas de pantalla de dos terminales. En la Figura 22a se muestran la consola de XCTU del coordinador. Mientras que en la Figura 22b se ven los mensajes recibidos desde el terminal del dispositivo conectado a la red ZigBee (`DISPOSITIVO 1`).



(a) Consola con el módulo coordinador.



(b) Consola con el módulo conectado a la red.

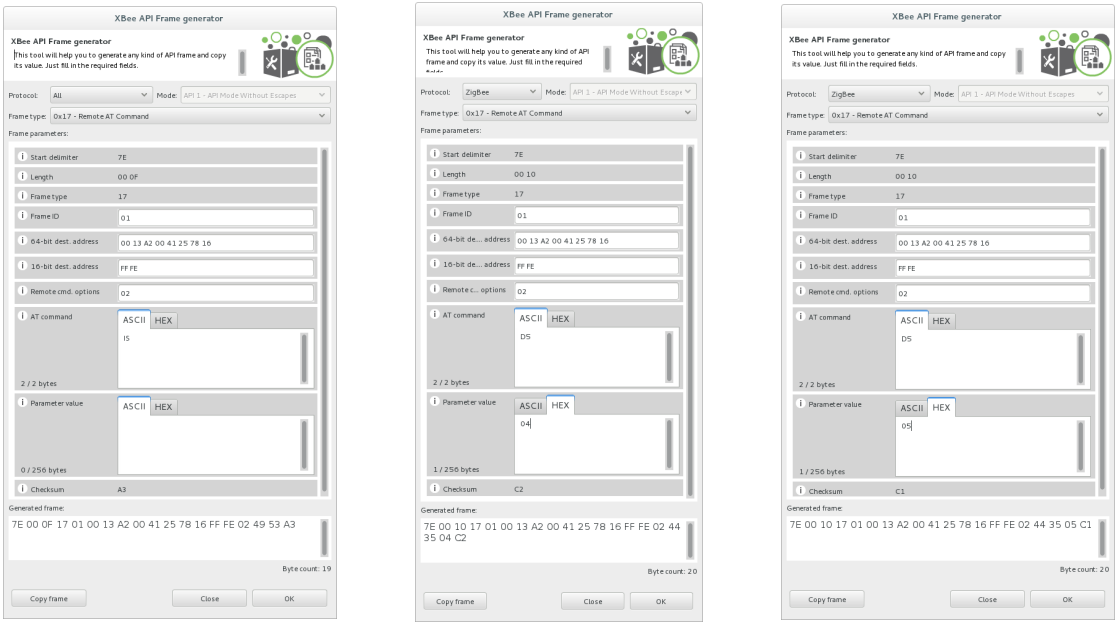
Figura 22: Captura de pantalla de las consolas utilizando XCTU.

6. Control remoto de los módulos XBee

En la Sección 4 se realizaron pruebas de los módulos en modo API. Este modo introduce complejidad pues debe generarse una trama específica pero de esta forma se logra mayores prestaciones por parte de los módulos.

En el siguiente ejemplo se describirá el mecanismo por el cual se configura el comportamiento de un módulo XBee en forma remota desde otro. Los módulos debe estar ya configurados como se describió anteriormente. Se utilizará el *Generador de Frames*, aplicativo del software XCTU.

En la Figura 23 se capturan la generación de comandos enviados desde el módulo coordinador a uno de sus dispositivos conectados. Primero se releva el estado del módulo, esto con el comando `ATIS` (Fig. 23a). Aquí se ingresa la dirección destino del módulo al que se le enviará el comando. Luego de chequear el estado de los puertos se enviará un comando para asignar un estado bajo al puerto `DIO5` (Fig. 23b). De la misma forma, se envía un comando al puerto 5 con el cambio de estado `ATD55` (Fig. 23c).



(a) Comando `ATIS`. (b) Comando `ATD54`. (c) Comando `ATD55`.

Figura 23: Comandos remotos desde un módulo a otro.

Los comandos AT se encuentran concatenados con los caracteres propios de la trama definida por el API. Por lo tanto, la clave es las funcionalidades y prestaciones del *interface*. Se podría enviar los mismos comandos AT de la Sección 3 sin problemas. Pues la trama API identifica qué tipo de información se está enviando.

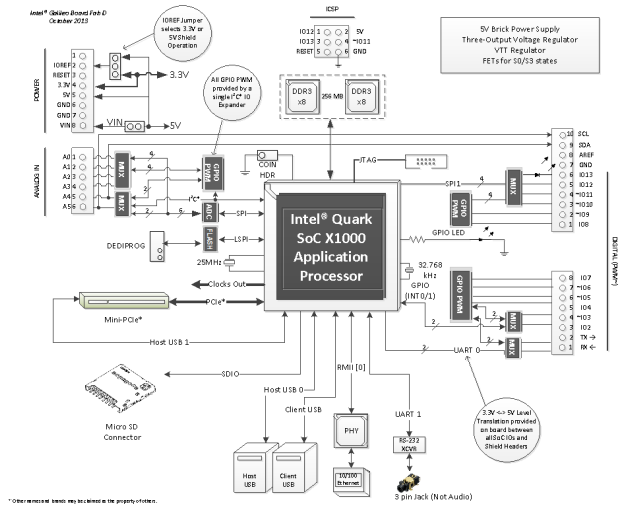
7. Librería `xbee-arduino`

Esta librería permite la comunicación con los módulos XBee en modo API. Tiene soporte tanto para los dispositivos *Series 1* (802.15.4) y *Series 2* (ZB Pro/ZNet). De esta forma la librería da soporte para enlaces TX/RX, comandos AT locales y remotos, modos I/O *samples* y *Modem Status*[7].

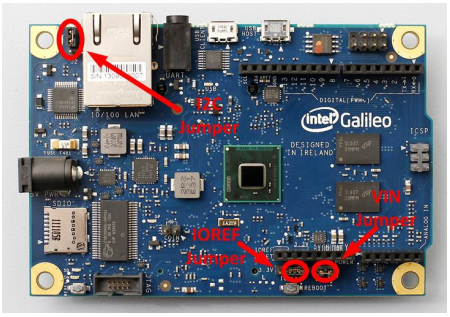
La plataforma arduino que se utilizará será la *Intel Galileo*, Figura 24. Esta plataforma está basada en el microcontrolador *Intel® Quark SoC X1000 Application Processor*. Las características de esta plataforma se detallan en la hoja de datos disponibles en la bibliografía citada[8]. Sí bien esta plataforma es una SBC[5] dispone de un proceso en el sistema operativo tal que permite utilizar el IDE Arduino. Para el usuario la programación es transparente e incluso la placa Galileo permite conectar los *shields* de Arduino. Utilizaremos estos conectores para comunicar nuestros módulos XBee con la Galileo.

Una vez instalada la librería desde los repositorios de Arduino se procede a iniciar la comunicación entre la plataforma Galileo y el módulo XBee.

[5]Siglas de *Single-Board Computer*(`https://en.wikipedia.org/wiki/Single-board_computer`).



(a) Diagrama en bloque.



(b) Pines de configuración.

Figura 24: Plataforma Intel® Galileo.

7.1. Comandos AT

La primera prueba será enviar comandos AT desde la Galileo hacía el módulo XBee conectado en los conectores disponibles. En el código 10 se muestra la estructura básica de cómo se debe utilizar la librería `xbee-arduino`.

Código 10: Ejemplo `AtCommand`

```
#include <XBee.h>
XBee xbee = XBee();
// serial high
uint8_t shCmd[] = {'S','H'};
// serial low
uint8_t slCmd[] = {'S','L'};

AtCommandRequest atRequest = AtCommandRequest(shCmd);
AtCommandResponse atResponse = AtCommandResponse();

void setup() {
  Serial1.begin(9600);
  xbee.begin(Serial1);
  // start soft serial
  Serial.begin(9600);

  // Startup delay to wait for XBee radio to initialize.
  // you may need to increase this value if you are not getting a
     response
  delay(5000);
}

void loop() {

  // get SH
  sendAtCommand();
```

```
26
27    // set command to SL
28    atRequest.setCommand(slCmd);
29    sendAtCommand();
30
31    // we're done.  Hit the Arduino reset button to start the sketch
          over
32    while (1) {};
33  }
```

En este documento no se explicará el paradigma de programación de Arduino, tampoco se entrará en detalle de los códigos utilizados. De todas formas se explicarán las códigos que hacen el uso de la librería `xbee-arduino`. En las líneas **8-9** se crean los objetos correspondientes a la recepción y respuesta de un comando AT. Se habilitarán dos puertos de la Galileo. El primero (línea **12**) corresponde al puerto mapeado al conector *shield* arduino. Este será asignado para que la librería `xbee-arduino` la utilice en su comunicación con el módulo. El puerto serie que está vinculado al cable USB conectado con la placa galileo será utilizado como *Monitor Serial* del IDE Arduino (línea **15**). Ya en el bucle `loop()` se envían los comandos y se interpreta consecutivamente la respuesta desde el módulo. En la Figura 25 se puede ver el intercambio de mensajes entre la Galileo y el módulo XBee.

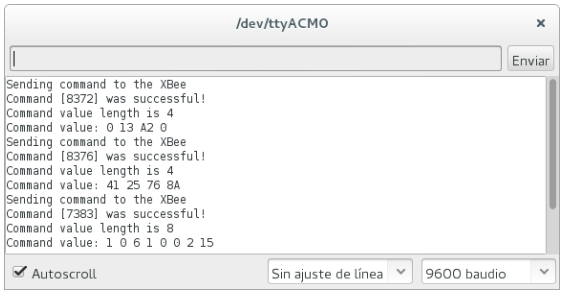


Figura 25: Monitor serial del IDE Arduino del ejemplo `AtCommand`.

7.2. Comunicación entre dos plataformas Galileo utilizando XBee.

Se plantea la posibilidad de utilizar dos plataformas Galileo e interactuar entre sí utilizando los módulos XBee como medio de comunicación. Se adaptará los ejemplos provistos por la librería. Primero se adaptará el ejemplo `Series2_Tx`.

7.2.1. Transmisión de trama

En el código 11 se tiene las principales líneas que implementa la librería. El arreglo `payload[]` almacena los datos a ser enviados. Por la inicialización del arreglo, se enviarán solo dos bytes (línea **6**). Luego se crean los objetivos necesarios para direccionar el mensaje, realizar el envío y chequear el estado de la transacción (líneas **9-11**). Se enviarán los valores devueltos por la función `millis()` y `micros()` (líneas **25-26**). El proceso cíclico comienza cargando los datos a ser enviado (línea **28**). Luego del envío se espera el paquete de respuesta. Esta respuesta proporciona el estado en que ha sido recibido el mensaje (línea **40**). Esto se repite cada 1 segundo (línea **55**).

Código 11: Ejemplo Series2_Tx

```
#include <XBee.h>

// create the XBee object
XBee xbee = XBee();

uint8_t payload[] = { 0, 0 };

// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200,0x41257816);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

//...

void setup() {
  pinMode(statusLed, OUTPUT);
  pinMode(errorLed, OUTPUT);

  Serial1.begin(9600);
  xbee.setSerial(Serial1);
}

void loop() {
  // break down 10-bit reading into two bytes and place in payload
  payload[0] = millis() & 0xff;
  payload[1] = micros() & 0xff;

  xbee.send(zbTx);

  // after sending a tx request, we expect a status response
  // wait up to half second for the status response
  if (xbee.readPacket(500)) {
    // got a response!

    // should be a znet tx status
    if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE) {
      xbee.getResponse().getZBTxStatusResponse(txStatus);

      // get the delivery status, the fifth byte
      if (txStatus.getDeliveryStatus() == SUCCESS) {
        // success.  time to celebrate
        flashLed(statusLed, 5, 50);
      } else {
        // the remote XBee did not receive our packet. is it
           powered on?
        flashLed(errorLed, 3, 500);
      }
    }
  } else if (xbee.getResponse().isError()) {
    //nss.print(‘‘Error reading packet.  Error code: ‘‘);
    //nss.println(xbee.getResponse().getErrorCode());
  } else {
```

```
52      // local XBee did not provide a timely TX Status Response --
            should not happen
53      flashLed(errorLed, 2, 50);
54    }
55    delay(1000);
56  }
```

7.2.2. Recepción de trama

Desde otra plataforma Galileo se adapta el ejemplo `Series2_Rx` de la librería `xbee-arduino`. El código 12 muestra lo básico. Las primeras líneas mantienen la estructura del ejemplo anterior. Los objetos que se crean hacen a la respuesta ante algún pedido desde otro módulo. Se utilizará el monitor del IDE Arduino para presentar los datos recibidos (línea **17**). El bucle se encuentra a la espera de un paquete (línea **26**). Una vez que el paquete llega (línea **28**) se procede a identificar los diferentes campos de bytes de la trama. Se utilizan indicadores leds que proporcionan los estados en que llega la trama. Luego se presentan los bytes enviados desde el otro módulo. Las líneas **46** y **48** imprimen en la consola del IDE arduino.

Código 12: Ejemplo `Series2_Rx`

```
1  #include <XBee.h>
2
3  XBee xbee = XBee();
4  XBeeResponse response = XBeeResponse();
5  // create reusable response objects for responses we expect to
       handle
6  ZBRxResponse rx = ZBRxResponse();
7  ModemStatusResponse msr = ModemStatusResponse();
8
9  int statusLed = 13;
10 int errorLed = 12;
11
12 void setup() {
13   pinMode(statusLed, OUTPUT);
14   pinMode(errorLed, OUTPUT);
15
16   // start serial
17   Serial.begin(9600);
18   Serial1.begin(9600);
19   xbee.begin(Serial1);
20
21 }
22
23 // continuously reads packets, looking for ZB Receive or Modem
       Status
24 void loop() {
25
26     xbee.readPacket();
27
28     if (xbee.getResponse().isAvailable()) {
29       // got something
30
```

```
31        if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
32          // got a zb rx packet
33
34          // now fill our zb rx class
35          xbee.getResponse().getZBRxResponse(rx);
36
37          if (rx.getOption() == ZB_PACKET_ACKNOWLEDGED) {
38              // the sender got an ACK
39              // ...
40          } else {
41              // we got it (obviously) but sender didn't get an ACK
42              // ...
43          }
44          // set dataLed PWM to value of the first byte in the data
45          Serial.print(‘‘Mensaje recibido (LOW): ‘‘);
46          Serial.println(rx.getData(0));
47          Serial.print(‘‘Mensaje recibido (HIGH): ‘‘);
48          Serial.println(rx.getData(1));
49        } else if (xbee.getResponse().getApiId() ==
             MODEM_STATUS_RESPONSE) {
50          // ...
51      } else if (xbee.getResponse().isError()) {
52        Serial.print(‘‘Error reading packet.  Error code: ‘‘);
53        Serial.println(xbee.getResponse().getErrorCode());
54      }
55  }
```

En la Figura 26 se capturó la consola del IDE Arduino y se pueden ver cómo llegan los mensajes desde el código 11.

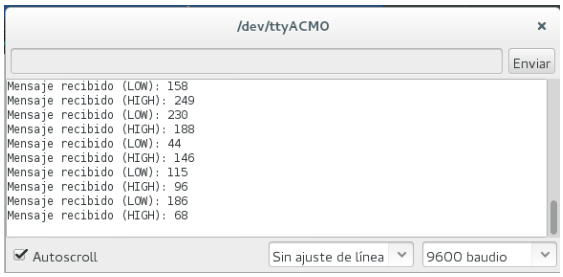


Figura 26: Monitor serial del IDE Arduino del ejemplo `Series2_Rx`.

7.3. Envío de datos periódicos (*Samples Mode*)

Los módulos XBee que se encuentren configurados como *end-devices* pueden configurar sus diferentes GPIO de forma tal que envíen información

Los módulos XBee ZB tienen la posibilidad de monitorear los canales analógicos y digitales habilitados. En esta configuración el módulo puede leer para uso local o transmitir la información relevada a otro módulo en forma remota. Para esto debe habilitarse el modo API[4].

En nuestro caso, se configura a uno de los dispositivos considerado *end-devices* de forma tal que envíe en forma periódica (cada 2 segundos) los estados de sus diferentes

puertos habilitados (analógicos y digitales). En la Figura 27 se muestra el parámetro que define esta funcionalidad de los XBee. Este módulo es parte de un sistema donde también hay un dispositivo central (coordinador). Este último recibirá los datos enviados desde los *end-devices*. Por esta razón los módulos no tienen la necesidad de configurar la dirección de destino pues todos los datos serán enviados al coordinador de la red.

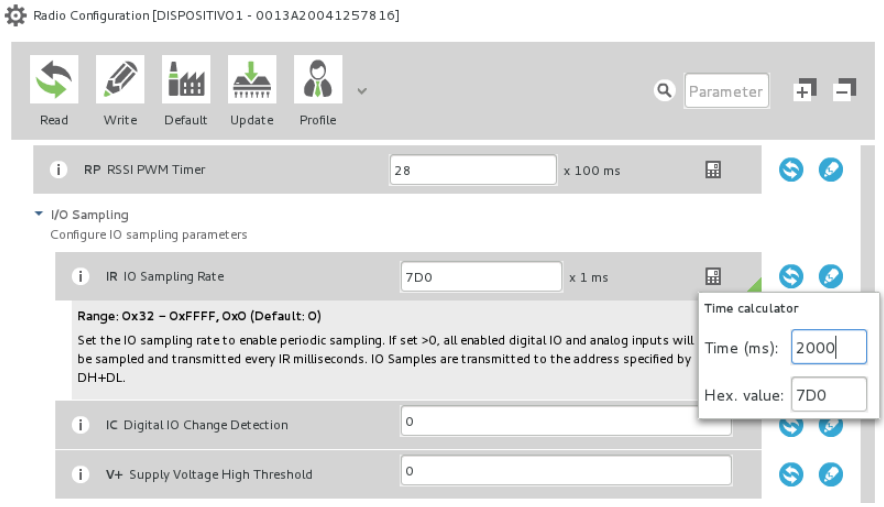


Figura 27: Configuración en *Samples Mode* utilizando la aplicación XCTU.

La librería `xbee-arduino` dispone de un ejemplo donde la plataforma arduino está conectado al coordinador de la red ZigBee. El código 13 muestran las principales líneas del ejemplo `Series2_IoSamples`. El ejemplo crea un objeto `ZBRxIoSampleResponse` al que se le vinculará con el paquete recibido por algún módulo *end-devices* (línea **24**). Los mensajes que recibe el coordinador se presentan en la consola IDE Arduino identificando a cada uno con su dirección (líneas **28-29**). Cada *frame* incluye la cantidad de entradas analógicas y digitales habilitadas por parte del *end-device* que envía el dato (líneas **41-56**).

Código 13: Ejemplo `Series2_IoSamples`

```
1  #include <XBee.h>
2
3  XBee xbee = XBee();
4
5  ZBRxIoSampleResponse ioSample = ZBRxIoSampleResponse();
6
7  XBeeAddress64 test = XBeeAddress64();
8
9  void setup() {
10   Serial1.begin(9600);
11   xbee.setSerial(Serial1);
12   // start soft serial
13   Serial.begin(9600);
14 }
15
16 void loop() {
17   //attempt to read a packet
18   xbee.readPacket();
```

```
19
20  if (xbee.getResponse().isAvailable()) {
21    // got something
22
23    if (xbee.getResponse().getApiId() == ZB_IO_SAMPLE_RESPONSE) {
24      xbee.getResponse().getZBRxIoSampleResponse(ioSample);
25
26      Serial.print(``Received I/O Sample from: '');
27
28      Serial.print(ioSample.getRemoteAddress64().getMsb(), HEX);
29      Serial.print(ioSample.getRemoteAddress64().getLsb(), HEX);
30      Serial.println(`"');
31
32      if (ioSample.containsAnalog()) {
33        Serial.println(``Sample contains analog data'');
34      }
35
36      if (ioSample.containsDigital()) {
37        Serial.println(``Sample contains digtal data'');
38      }
39
40      // read analog inputs
41      for (int i = 0; i <= 4; i++) {
42        if (ioSample.isAnalogEnabled(i)) {
43          Serial.print(``Analog (AI'');
44          Serial.print(i, DEC);
45          Serial.print(`') is '');
46          Serial.println(ioSample.getAnalog(i), DEC);
47        }
48      }
49
50      // check digital inputs
51      for (int i = 0; i <= 12; i++) {
52        if (ioSample.isDigitalEnabled(i)) {
53          Serial.print(``Digital (DI'');
54          Serial.print(i, DEC);
55          Serial.print(`') is '');
56          Serial.println(ioSample.isDigitalOn(i), DEC);
57        }
58      }
59
60      // method for printing the entire frame data
61      //for (int i = 0; i < xbee.getResponse().getFrameDataLength
            (); i++) {
62      //  Serial.print(``byte ['');
63      //  Serial.print(i, DEC);
64      //  Serial.print(`'] is '');
65      //  Serial.println(xbee.getResponse().getFrameData()[i], HEX
            );
66      //}
67    }
68    else {
69      Serial.print(``Expected I/O Sample, but got '');
70      Serial.print(xbee.getResponse().getApiId(), HEX);
```

```
71      }
72    } else if (xbee.getResponse().isError()) {
73      Serial.print(‘‘Error reading packet.  Error code: ‘‘);
74      Serial.println(xbee.getResponse().getErrorCode());
75    }
76  }
```

En la Figura 28 se presenta la consola con la información proporcionada por el código 13.

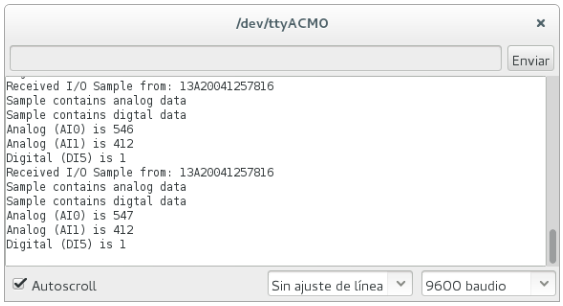


Figura 28: Monitor serial del IDE Arduino del ejemplo `Series2_IoSamples`.

8. Conclusiones

Los sistemas en los que requieren la implementación de pequeñas redes locales inalámbricas se caracterizan por la autonomía de los dispositivos, el alcance, topología y compromiso entre costo/beneficio que prestan los potenciales candidatos. En el módulo *Entornos Inalámbricos* se propone la utilización de los módulos XBee de Digi International. Estos módulos implementan el estándar IEEE 802.15.4. Sí bien existen otros módulos comerciales con menos llegada a un costo menor[6], estos no presentan la garantía y posibilidades en rendimiento que los módulos XBee. En las pruebas realizadas se comprobó la flexibilidad de su utilización. La posibilidad de utilizar un *Application Programming Interface* permite la interacción con otros sistemas embebidos de una manera mucho más amigable que los comandos AT. En esta línea, las plataformas Arduino son la forma más directa y sencilla de implementar los módulos XBee en las diferentes configuraciones disponibles. Como se pudo leer en las diferentes secciones, se comenzó de básico en interactuar con el módulo XBee en modo AT (*transparent mode*) al punto implementar el API y la automatización del envío de datos de los módulos XBee al coordinador (unidad central de procesamiento del sistema).

Como trabajos futuros podría considerarse la utilización de plataformas de menor costo que proporcione al usuario una API flexible. Además sería interesante la disponibilidad de mayores módulos de forma tal que se puedan asomar problemas en el protocolo utilizado. Quizá algún emulador de pequeñas redes (PAN) pueda ser deseable.

Referencias

[1] Andreas F. Molisch. *Wireless Communications*. John Wiley & Sons. 2011.

[6] Por ejemplo, SparkFun Transceiver Breakout: https://www.sparkfun.com/products/691

[2] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press. 2005.

[3] Recommendation ITU-R P.1238-8. *Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 300 MHz to 100 GHz*. P Series Radiowave propagation. 2015.

[4] Digi International, *XBee®/XBee-PRO ZigBee® RF Module – User Guide*. 2016.

[5] Cika Electrónica SRL., *XBoard [-W]*. http://www.cika.com/soporte/Information/Digi-RF/XBee/XBoard_doc.pdf.

[6] Digi International, *Knowledge Base – The AT Command Set*. http://knowledge.digi.com/articles/Knowledge_Base_Article/The-AT-Command-Set.

[7] andrewrapp, *Arduino library for communicating with XBee radios in API mode*, https://github.com/andrewrapp/xbee-arduino.

[8] Intel®, *Galileo Datasheet*. http://www.intel.com/newsroom/kits/quark/galileo/pdfs/Intel_Galileo_Datasheet.pdf.

[9] Digi International, *Knowledge Base – The AT Command Set*. http://knowledge.digi.com/articles/Knowledge_Base_Article/The-AT-Command-Set.