Nota de Aplicación: CAN-096
Título: **Utilización de XBee IP Services**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	16/09/11	

En esta Nota de Aplicación veremos la forma de utilizar los XBee IP Services (XBee Application Service y Serial Communication Service) para configurar y recibir información de módulos XBee Wi-Fi con cualquier host IP; en particular incluimos una implementación open-source en C para Linux.

Introducción

El XBee Wi-Fi opera sobre una red IP. A fin de soportar tanto el flujo de datos de su puerto serie como la configuración remota y los reportes de entradas y salidas, presenta dos servicios:

- XBee Application Service: por UDP en el puerto 0xBEE
- Serial Communication Service: por UDP ó TCP, en puerto configurado por el usuario

La comunicación entre módulos XBee Wi-Fi transcurre de forma transparente para el usuario, pero para poder interactuar con los módulos desde cualquier host IP, debemos conocer las características de estos servicios.

Serial Communication Service

Los datos que el módulo XBee recibe por su puerto serie en modo transparente, o dentro de una trama Transmit Request IPv4 en modo API, son enviados al socket determinado por los siguientes parámetros:
DL: dirección de destino
DE: port de destino
MA: protocolo
C0: port de origen
Los datos que el XBee recibe en el port configurado en el parámetro *C0*, son enviados por el puerto serie, en modo transparente o modo API (dentro de una trama Receive Packet IPv4), según como esté configurado el parámetro *AP*.

XBee Application Service

Las funcionalidades principales de este servicio son las de proveer configuración remota y reportar el estado de entradas y salidas. A tal fin, la comunicación se realiza por el puerto 0xBEE en UDP con el siguiente formato:

<ID><OPTIONS><DATA>

ID: identifica a la trama, los identificadores para estas aplicaciones son:
0x04: información de estado de I/O
0x02: comando remoto
0x82: respuesta a un comando remoto
OPTIONS: no utilizado para estas aplicaciones
DATA: el contenido útil en sí

Breve descripción de la implementación propuesta

Si bien el framing es simple, hemos desarrollado un sencillo esquema en C. La implementación sugerida ha sido desarrollada y probada bajo Linux, pero resulta fácil de portar a cualquier sistema en el que se conozca la forma de utilizar sockets.

En cuanto a lo que deseamos transmitir o recibir, existe una estructura para cada tipo de trama. La estructura *WFXAframe* implementa el formato de la trama y contiene el buffer que llenaremos con nuestros datos a transmitir y del que extraeremos los datos recibidos: *APIstructname.data*

```
// Should accomodate 1400 bytes data packet
#define WFXABUFFERSZ   1400
typedef struct {
       unsigned char id;
       unsigned char options;
       unsigned char data[WFXABUFFERSZ];
} WFXAframe;

typedef struct {
       unsigned char frameid;
       unsigned char options;
       unsigned char cmd[2];
       unsigned char param;
} WFXArmtcmd;

typedef struct {
       unsigned char frameid;
       unsigned char cmd[2];
       unsigned char status;
       unsigned char data;
} WFXArmtrsp;
```

Estas estructuras se hallan descriptas en *wfxa.h*, que se incluye en el archivo adjunto.

Ejemplo de utilización

En el caso del Serial Communications Service, la información se obtiene directamente del socket, por lo que no reviste mayor complejidad, más allá de la utilización de las funciones correspondientes que dependerán del sistema utilizado.

Respecto al XBee Application Service, vemos aquí un simple ejemplo de como aplicar estas funciones a un caso de recepción de mensajes. El campo *id* de la estructura *WFXAframe* nos permite determinar qué tipo de trama es, y realizando un cast al tipo correspondiente podemos obtener fácilmente la información del campo data. Recordemos que la información de quién envía la obtenemos del socket, y esto dependerá de nuestro sistema en particular. En este caso, el buffer es provisto por el sistema de sockets.

```
#include "wfxa.h"


main (int argc, char *argv[])
{
WFXAframe *wfxa;
WFXArmtcmd *rcmd;
WFXArmtrsp *rresp;


  while(1){
    if(mensaje){
          wfxa=(WFXAframe *)rbuf;
          switch(wfxa->id){
          case WFXA_RMTRSP:
                rresp=(WFXArmtrsp *)wfxa->data;
                /* Remote Command Response frame:
                 ID:   rresp->frameid
                 STS:  rresp->status
                 DATA: rresp->data
                */
                break;
          case WFXA_WIORECV:
                /* I/O Data Receive
                 wior: (APIwiorecv *)wfxa->data;
                */
```

```
                break;
            }
        }
    }
}
```

Para enviar información debemos armar la trama y pasársela a la función que envía información por un socket:

```
#include "wfxa.h"
#include "apicommon.h"


main (int argc, char *argv[])
{
WFXAframe wfxa;
WFXArmtcmd *rcmd;

    wfxa.id=WFXA_RMTCMD;
    wfxa.options=0;
    rcmd=(WFXArmtcmd *)wfxa.data;
    rcmd->frameid=0x5A;
    rcmd->options=RCO_APPLY;
    rcmd->cmd[0]='D';
    rcmd->cmd[1]='2';
    rcmd->param=5;
```

Contenido del archivo adjunto

El archivo adjunto incluye los siguientes archivos:

- *wfxa.h*: definiciones de las estructuras del framing
- *monitorwf_udp.c*: ejemplo de utilización, decodifica y muestra en pantalla las tramas que recibe por ambos servicios. El acceso a sockets es Berkeley, pero la espera propiamente dicha se realiza mediante una función disponible sólo para Linux (*poll()*).
- *d2onwf_udp.c*: ejemplo de utilización, coloca D2 en estado alto mediante el envío de un comando remoto
- *rsamplewf_udp.c*: ejemplo de utilización, pide una muestra a un módulo remoto

Dichos archivos compilan bajo *gcc* y requieren algunos archivos provistos en CAN-095 para presentación amigable para linkear; por ejemplo:

```
$ gcc -o monitorwf_udp monitorwf_udp.c monlibwf.o monlib.o
$ ./monitorwf_udp 9750

Got msg from 192.168.1.16 port 0xBEE
RX: 04 00 01 08 1C 03 00 18 02 J(4A) 02 1D
I/O Data Receive,1 samples, active digital channels: 08 1C, analog channels: 03
        I/O: 018  AN0: 24A  AN1: 21D
```