Cika Electrónica

Nota de Aplicación: CAN-088
Título: **Utilización del modo API en módulos XBee 802.15.4**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	09/06/08	
1	24/11/08	ampliación y modificaciones para soportar otros XBee manteniendo la independencia

En esta Nota de Aplicación veremos la forma de utilizar el modo API en módulos XBee (o XBee-PRO) 802.15.4; en particular incluimos una implementación open-source en C.

Introducción

Comentamos en varios Comentarios Técnicos la existencia de un modo de comunicación denominado API. En este modo de trabajo, el módulo XBee sigue respondiendo a la secuencia de escape y el envío de comandos AT, pero la información de y hacia otros módulos respeta un framing particular, el cual permite simplificar la operatoria con varios módulos remotos, dado que es posible identificar el origen y seleccionar el destino de la información dentro del mismo paquete de datos.

Breve descripción del framing API

El framing y los distintos tipos de tramas se hallan descriptas en el manual del usuario de los módulos, por lo que nos centraremos aquí en los conceptos fundamentales.

El modo API se configura mediante el parámetro *AP*. Existen dos modos de trabajo, uno "normal" (*AP*=1) en el que la trama comienza con el caracter 0x7E, y otro "con escape" (*AP*=2) en el cual se garantiza que el caracter 0x7E sólo aparece en el stream de datos para indicar el inicio de una trama, de igual modo que en SDLC/HDLC. En este modo, si alguno de los datos a transmitir contiene el valor 0x7E, éste resulta reemplazado por una secuencia de escape. Lo mismo se aplica a caracteres de control de flujo XON/XOFF (0x11 y 0x13) y el caracter utilizado para indicar la secuencia de escape, 0x7D.

En resumen, en el modo "normal", la presencia de un caracter 0x7E indica el inicio de una trama sólo si es el primer caracter que se recibe luego de un silencio, mientras que en el modo "con escape" siempre indica el inicio de una trama y debemos detectar las secuencias de escape y reemplazarlas por los caracteres originales.

Básicamente, una trama comienza con el caracter de inicio de trama, 0x7E, contiene dos bytes que indican la longitud (LEN), un byte adicional que indica el tipo de trama de que se trata (ID), una serie de bytes dependientes del tipo de trama en cuestión (DATA), y un checksum que permite verificar que lo que se recibió es correcto. El tipo de trama (ID) nos indica de qué se trata, por ejemplo los datos enviados desde un remoto (recibidos por su puerto serie) corresponden al tipo 0x81 si la dirección reportada es de 16-bits y 0x80 si es de 64-bits. Los datos correspondientes a mediciones de un remoto (I/O data) corresponden a los tipos 0x83 y 0x82 para direcciones de 16 y 64-bits respectivamente.

<0x7E><LEN: 2 bytes><INFO: len bytes><CHECKSUM>
INFO: <ID><DATA>

Las tramas incorporan además información adicional como por ejemplo intensidad de señal, opciones (por ejemplo si la trama fue un broadcast):

ID=0x81: <ADDR: 8 bytes><RSSI><OPTIONS><DATA>

Breve descripción de la implementación propuesta

Si bien el framing es simple, hemos desarrollado un sencillo esquema en C que nos permite utilizar el modo API para poder identificar remotos y enviarles mensajes en un sistema con muchos módulos. La

implementación sugerida ha sido desarrollada y probada bajo Linux, pero resulta fácil de portar a cualquier microcontrolador.

El esquema propuesto se maneja mediante dos juegos de tres funciones que permiten enviar y recibir los datos valiéndose de una estructura de tipo *APIstruct*, la cual contiene internamente otras estructuras que a su vez contienen el buffer para los datos y dan el formato de la trama. Las funciones son:

- *void API_buildframe(APIstruct *api,unsigned char datalen)*; da formato a una trama
- *int API_checkframe(APIstruct *api)*; valida la trama recibida, devuelve la longitud de los datos
- *void API_initsendingframe(APIstruct *api)*; comienza el proceso de envío de una trama
- *int API_sendframe(APIstruct *api, unsigned char escape)*; realiza el proceso de envío y genera las secuencias de escape (si 'escape' es distinto de cero). Devuelve *API_DONE* al finalizar y *API_AGAIN* mientras es necesario llamarla nuevamente.
- *void API_initgettingframe(APIstruct *API)*; comienza el proceso de recepción de una trama
- *int API_getframe(APIstruct *api, unsigned char escape)*; realiza el proceso de recepción y recupera las secuencias de escape (si 'escape' es distinto de cero). Devuelve la longitud de la trama (valor mayor que cero) al finalizar y *API_AGAIN* mientras es necesario llamarla nuevamente. Si se utiliza el modo con escape, la sincronización de inicio de trama se realiza mediante el caracter 0x7E. Si no se lo utiliza, debe sincronizarse de forma externa detectando no utilización y llamando a *API_initgettingframe()*. Esta función llama automáticamente a *API_checkframe()* retornando *API_DONE* sólo en el caso de recibir una trama válida (borra el buffer y comienza a colectar de nuevo si falla la detección).

Como seguramente pudimos inferir, el parámetro *escape* indica si se utiliza o no este modo de operación, y debe coincidir con lo configurado en el módulo del que recibamos y queramos mandar datos. Por ejemplo, si el parámetro *AP* en el módulo tiene el valor 1, no utilizamos el modo con escape y deberemos asignar el valor 0 al parámetro *escape* al llamar a estas funciones.

Dos funciones adicionales son llamadas por las anteriores, y son las que se ocupan de acceder al hardware para interactuar con la UART:

- *extern int TX(unsigned char)*; devuelve -1 si no puede transmitir (no hay lugar en el buffer), 0 si pudo
- *extern int RX(void)*; devuelve el caracter recibido o -1 si no hay ninguno

La función *TX()* puede o no implementar un buffer, pero *RX()* debe hacerlo para permitir recuperar la cantidad de caracteres que puedan recibirse entre llamadas a *API_getframe()* y/o entre el retorno de ésta y la subsiguiente llamada a *API_initgettingframe()* para obtener la trama siguiente. Obviamente, éstas son las funciones que deberemos portar al hardware que deseemos utilizar.

En cuanto a lo que deseamos transmitir o recibir, existe una estructura para cada tipo de trama API. La estructura *APIstruct* incluye a la estructura *APIframe*, la cual implementa el formato de la trama. Posee un campo de tipo *APIinfo* en el cual viaja la parte útil de la trama y contiene el buffer que llenaremos con nuestros datos a transmitir y del que extraeremos los datos recibidos: *APIstructname.frame.info.data*

```
// Should accomodate 100 bytes data packet + header overhead
#define APIBUFFERSZ     128
#define API_MAXFRAMESZ  APIBUFFERSZ
//account for start, len, and checksum
#define API_OVERHEAD    4
//account for id
#define API_DATAOVH     (API_OVERHEAD+1)

typedef struct {
        unsigned char id;
        unsigned char data[API_MAXFRAMESZ-API_DATAOVH];
} APIinfo;

typedef struct {
        unsigned char start;
        unsigned char len[2];
        APIinfo info;
} APIframe;

typedef struct {
```

```
        unsigned char *ptr;
        unsigned char cnt;
        unsigned char len;
        APIframe frame;
} APIstruct;
```

Estas estructuras se hallan descriptas en *apiframe.h*, que se incluye en el archivo adjunto. Las demás estructuras que soportan a los diversos tipos de frames se hallan descriptas en *apicommon.h* y *api15_4.h*, también incluidas en el archivo adjunto.

Ejemplo de utilización

Vemos aquí un simple ejemplo de como aplicar estas funciones a un caso de lectura de tramas de información. Inicializamos el proceso de recepción, llamamos a la función que recibe hasta tanto ésta nos indique que dispone de una trama lista. A continuación copiamos la trama a un buffer para permitir seguir recibiendo tramas y apuntamos a los datos. El campo *info* de la estructura *APIinfo* nos permite determinar qué tipo de trama es, y realizando un cast al tipo correspondiente podemos obtener fácilmente la información del campo data:

```
#include "apiframe.h"
#include "apicommon.h"
#include "api15_4.h"
#define escape 1

main ()
{

int ret;
static APIstruct apirx;
static APIdata rxdata;
APIinfo *info;
APIrecv16 *r16;
APIrecv64 *r64;


        API_initgettingframe(&apirx);                       // comienza recepción
        while(1){
                while((ret=API_getframe(&apirx,escape))<=0); // recibe trama
                rxdata.len=ret;                             // obtiene longitud de datos
                memcpy(&rxdata.info,&apirx.frame.info,ret);  // copia, libera buffer
                API_initgettingframe(&apirx);               // recibe trama siguiente (luego)
                info=&rxdata.info;                          // apunta a INFO
                switch(info->id){                           // tipo de trama
                case API_APIRECV64:                         // DATA, dirección de 64-bits
                        r64=(APIrecv64 *)&info->data;       // apunta a DATA
                        /*
                                ADDR: r64->addr,8
                                RSSI: r64->rssi
                                OPT:  r64->options
                                DATA: r64->data
                        */
                        break;
                case API_APIRECV16:                         // DATA, dirección de 16-bits
                        r16=(APIrecv16 *)&info->data;       // apunta a DATA
                        break;
                }
        }
}
```

Para enviar información debemos armar la trama y llamar a la función que la envía:

```
static APIstruct api;
APIsend64 *s64;

    api.frame.info.id=API_APISEND64;                        // tx data
    s64=(APIsend64 *)&api.frame.info.data;                  // apunta a data
        /*
        ID:   s64->frameid                                  // id para reconocer la respuesta
        ADDR: s64->addr,8
        OPT:  s64->options
```

```
       DATA: s64->data                                  // datos a enviar
       datalen                                          // longitud de los datos en DATA
    */
    API_buildframe(&api,sizeof(APIsend64)+datalen-1);   // arma trama
    API_initsendingframe(&api);                         // inicia envío
    while(API_sendframe(&api,0)==API_AGAIN);            // envía
```

Contenido del archivo adjunto

El archivo adjunto incluye los siguientes archivos:

- *apiframe.h*: definiciones de las estructuras del framing y funciones involucradas
- *apiframe.c*: código de dichas funciones
- *apicommon.h*: definiciones de las estructuras para frames comunes
- *api15_4.h*: definiciones de las estructuras para frames particulares del XBee 802.15.4
- *monlib15_4.h*: declaraciones de funciones adicionales para extracción de información de estos frames con la intención de presentarla de forma amigable.
- *monlib15_4.c*: código de dichas funciones, el foco está puesto en la claridad y simpleza, no en la performance.
- *decoder15_4.c*: ejemplo de utilización, la función *RX()* extrae caracteres de *stdin* y funciona como un decodificador ingresándose la trama por *stdin* (copy/paste).
- *monitor15_4.c*: ejemplo de utilización, decodifica y muestra en pantalla las tramas que recibe por el puerto serie. La lectura propiamente dicha se realiza mediante una función adicional incluida en *serial.c*, sólo para Linux. El ejemplo no implementa sincronismo por inactividad en el modo normal (sin escape), de modo que si se interrumpe el flujo normal debe reiniciarse el programa sin información por el puerto serie para resincronizar.

Dichos archivos compilan bajo *gcc* y no requieren más que las dependencias entre ellos para linkear; por ejemplo:

```
$ gcc -c apiframe.c
$ gcc -c monlib15_4.c
$ gcc -o decoder15_4 decoder15_4.c apiframe.o monlib15_4.o
$ gcc -c serial.c
$ gcc -o monitor15_4 monitor15_4.c apiframe.o monlib15_4.o serial.o
$ ./decoder15_4 0
7E 00 14 82 00 13 A2 00 40 52 85 80 30 00 01 06 1C 00 18 02 30 01 F7 9C

RX: 82 00 13 A2 00 40 52 85 80 30 00 01 06 1C 00 18 02 30 01 F7
I/O Data Receive, 64-bit address: ADDR: 00 13 A2 00 40 52 85 80 RSSI: 30 OPT: NONE (00)
1 samples, active channels: 06 1c
        I/O: 018
        AN0: 230
        AN1: 1F7

$ ./monitor15_4 0 /dev/modem 9600

RX: 82 00 13 A2 00 40 01 30 69 43 00 01 06 1C 00 18 02 3F 01 FE
I/O Data Receive, 64-bit address: ADDR: 00 13 A2 00 40 01 30 69 RSSI: 43 OPT: NONE (00)
1 samples, active channels: 06 1c
        I/O: 018
        AN0: 23F
        AN1: 1FE
```