Ejemplos de Aplicación para la placa de desarrollo TrackMe 3G

La placa TrackMe 3G es una poderosa herramienta de desarrollo para sistemas de rastreos, telemetría y control a distancia a través de la red celular. Esta placa incluye el modulo SIM5320 que incorpora tecnología 3G y GPS en el mismo encapsulado. Mediante diferentes ejemplos de aplicaciones, que van desde enviar un mensaje SMS hasta sincronizar datos en un servidor, aprenderemos las diferentes funcionalidades de la placa.

Los ejemplos están realizados en MPLABX con el compilador XC8 de Microchip, los cuales están disponibles desde la página www.microchip.com. El Microcontrolador que utiliza la placa es el PIC18F26J50. También haremos uso de las librerías de periféricos de la trackMe 3G que provee mcelectronics, que está escrito para el compilador XC8 y que contiene funciones para el manejo del modulo SIM5320 y los periféricos del micro como la USART y el ADC.

Ejemplo 1:

Al presionar el botón de pánico envía un SMS con el valor del acelerómetro, la temperatura y el nivel de luz.

main.c

```
/** INCLUDES *******************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;

MAIN_RETURN main(void)
{
    char mensaje[256];
    char telefono[16]="1562823174";

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    while(1)
    {
        SYSTEM_Tasks();

        if(BUTTON_ActivatePanic())
        {
            LED_On(LED_USB_DEVICE_STATE);
            //Se oprimio el boton de panico, armar el mensaje a enviar
            sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
            sprintf(mensaje,"%sTemp=%1.2f;",mensaje,ADCtrackme.temp);
            sprintf(mensaje,"%sAcelX=%1.2f;",mensaje,ADCtrackme.Xout);
            sprintf(mensaje,"%sAcelY=%1.2f;",mensaje,ADCtrackme.Yout);
            sprintf(mensaje,"%sAcelZ=%1.2f;",mensaje,ADCtrackme.Zout);
            sprintf(mensaje,"%sLDR=%i;",mensaje,ADCtrackme.ldr);
```

```
            //send SMS
            GSM_SendSMS(telefono,mensaje);

        }

    }//end while

}//end main
```

Explicación:

El código anterior corresponde al archivo main.c del proyecto, en el cual se monitorea el estado del pulsador y si encontramos que se oprime, se arma el mensaje de texto que se envía por el modulo SIM. El código comienza por incluir los dos archivos necesarios para realizar cualquier proyecto con la TrackMe, el archivo *"system.h"* y *"sim5320.h"*.

La declaración de las funciones que manejan el modulo SIM5320 están incluidas en el archivo de cabecera que tiene su mismo nombre, aquí se declaran las funciones que vamos a utilizar para resolver el ejercicio. Las funciones *SIM_On();* y *SIM_Init();* se utilizan para encender el modulo SIM y realizar la sincronización y la configuración inicial. Los parámetros que inicialmente se configuran en el modulo GSM es el manejo de los mensajes SMS en modo "texto" (se deshabilita el modo PDU) y ante la llegada de un nuevo mensaje SMS, este se envía inmediatamente en la interfaz USART con el micro, evitando así que el mensaje sea almacenado en memoria y consecuentemente se evita el llenado de la misma. La última función que se utiliza de la librería sim5320.h es *GSM_SendSMS();* que es la encargada de enviar el mensaje y recibe como parámetros el numero celular del destinatario y el mensaje a enviar. Ejecutando estas tres funciones provistas por la librería se puede enviar un mensaje SMS desde la placa TrackMe.

Ahora bien, con todo esto que hablamos, nos olvidamos de un detalle importante y es que antes de realizar cualquier manejo con el modulo SIM debemos inicializar todos los periféricos del Microcontrolador, es por ello que el otro archivo que se incluye en el código fuente es el de *"system.h".* Este ejemplo de la placa TrackMe requiere de la utilización del modulo USART para comunicarse con el SIM5320 y del modulo ADC para calcular los valores del acelerómetro, la temperatura y el nivel de Luz y es la función *SYSTEM_Initialize();* la encargada de configurar estos módulos y los pines GPIO correspondientes, para que ya tengamos el sistema configurado y podamos empezar la ejecución del programa. La código fuente de esta función de inicialización la veremos cuando realicemos el análisis del archivo *"system.c".* También allí veremos la utilidad de la función *SYSTEM_Tasks();* que es la encargada de contener las tareas que el sistema debe ejecutar periódicamente. Este ejemplo cuenta con una única tarea llamada ADC_tasks(); cuyo código fuente esta declarado en la librería adc_trackme.h y su funcionalidad es la de calcular los valores de los periféricos analógicos de la placa, como la temperatura, el valor de los tres ejes del acelerómetro y el nivel de LUZ en el LDR. Los resultados de estas conversiones se almacenan en una estructura global que provee la librería del ADC y que se debe declarar de manera global en el código fuente del archivo main.c de la siguiente manera.

```
extern sADCResult ADCtrackme;
```

Como es una variable externa se antepone el prefijo “extern” en la declaración de la variable *ADCtrackme* que es del tipo *sADCResult*, un tipo de dato definido en la librería del ADC. Esta variable ADCtrackme almacena en todo momento los valores de la conversión de la temperatura, los ejes del acelerómetro y el nivel de Luz en todo momento, por lo que si necesitamos saber el valor de la temperatura en cualquier instante, simplemente accedemos al valor almacenado en la variable ADCtrackme.temp definido en el tipo de dato sADCResult. Esta es una consecuencia de ejecutar periódicamente la tarea del sistema ADC, que en todo momento tendremos actualizado los valores de la conversión de los periféricos analógicos de la placa y resulta muy útil cuando queremos, por ejemplo, monitorear el valor de temperatura o de algún eje del acelerómetro y enviar un mensaje de alerta cuando este exceda de un determinado valor de umbral.

Entonces, ya vimos como desde el main.c se inicializan los periféricos del Microcontrolador desde la función *SYSTEM_Initialize();* y lo mismo para el modulo SIM, también como se realizan las tareas del sistema de manera periódica y poder tener en todo momento los valores de la conversión del ADC, lo que resta ver es como se realiza el monitoreo del pulsador y como se arma el mensaje que se enviara por SMS. Dentro del bucle principal del código, luego de que se realizan las tareas del sistema, existe un bloque condicional if que se ejecutara según el resultado de la función *BUTTON_ActivatePanic();* que pertenece a la librería “button.h” y nos entrega un valor booleano según si se presiono o no el botón de pánico de la placa, si la función devuelve TRUE, es porque el botón fue presionado. Lo próximo que se ejecuta luego de entrar en el bloque condicional es el armado del mensaje SMS que se envía como respuesta al oprimir el botón, esto se realiza con la función sprintf, que es una función de la librería estándar *“stdio.h”* en la cual se almacenara en la cadena de string que se ingresa como primer parámetro los caracteres que se añaden a continuación. Una vez armado el mensaje se lo envía como parámetro de la función *GSM_SendSMS();* junto con el número telefónico y esta se encargara de enviarla al modulo SIM para que sea enviado como SMS.

El siguiente archivo que analizaremos a continuación es system.c

system.c

```
#include <system.h>

#include "sim5320.h"
//#include <system_config.h>

/** CONFIGURATION Bits **********************************************/
#pragma config WDTEN = OFF          //WDT disabled (enabled by SWDTEN bit)
#pragma config PLLDIV = 3           //Divide by 3 (12 MHz oscillator input)
#pragma config STVREN = ON          //stack overflow/underflow reset enabled
#pragma config XINST = OFF          //Extended instruction set disabled
#pragma config CPUDIV = OSC1        //No CPU system clock divide
#pragma config CP0 = OFF            //Program memory is not code-protected
#pragma config OSC = HSPLL          //HS oscillator, PLL enabled, HSPLL used
by USB
#pragma config FCMEN = OFF          //Fail-Safe Clock Monitor disabled
#pragma config IESO = OFF           //Two-Speed Start-up disabled
#pragma config WDTPS = 32768        //1:32768
#pragma config DSWDTOSC = INTOSCREF //DSWDT uses INTOSC/INTRC as clock
#pragma config RTCOSC = T1OSCREF    //RTCC uses T1OSC/T1CKI as clock
#pragma config DSBOREN = OFF        //Zero-Power BOR disabled in Deep Sleep
#pragma config DSWDTEN = OFF        //Disabled
#pragma config DSWDTPS = 8192       //1:8,192 (8.5 seconds)
#pragma config IOL1WAY = OFF        //IOLOCK bit can be set and cleared
#pragma config MSSP7B_EN = MSK7     //7 Bit address masking
#pragma config WPFP = PAGE_1        //Write Protect Program Flash Page 0
#pragma config WPEND = PAGE_0       //Start protection at page 0
#pragma config WPCFG = OFF          //Write/Erase last page protect Disabled
#pragma config WPDIS = OFF          //WPFP[5:0], WPEND, and WPCFG bits ignored
#pragma config T1DIG = ON           //Sec Osc clock source may be selected
#pragma config LPT1OSC = OFF        //high power Timer1 mode

extern sSimResponse simResponse;            //Estructura global que contiene
respuestas del SIM5320
char intBuffer[256];                        //Buffer para recibir los
caracteres por USART

void SYSTEM_Initialize( SYSTEM_STATE state )
{
    switch(state)
    {
        case SYSTEM_STATE_START:
        {
             {
                 unsigned int pll_startup_counter = 600;
                 OSCTUNEbits.PLLEN = 1;
                 while(pll_startup_counter--);
             }
            //Inicializacion de los pine de la placa
            // Init PPS UART2
            PPSUnLock();
            PPSInput(PPS_RX2DT2,PPS_RP0);
            PPSOutput(PPS_RP1,PPS_TX2CK2);
            PPSLock();

            LED_Enable(LED_USB_DEVICE_STATE);
            LED_Off(LED_USB_DEVICE_STATE);
            BUTTON_Enable(BUTTON_TRACKME_PANIC);
            GSM_POWER_TRIS = PIN_OUTPUT;   //RC1 output
```

```
            GSM_POWER_LAT = GSM_OFF;      //Valor por defecto del pin
            USART_Initialize();
            ADC_Init();
            //Habilitamos interrupciones de perifericos
            INTCONbits.PEIE = 1; //Activamos interrupcion de perifericos
            INTCONbits.GIE = 1;  //Activamos las interrupciones globales
            break;
        }
    }
}


void SYSTEM_Tasks(void)
{
    //Tareas del sistema

//-----Tarea del ADC
    ADC_task();
//-----Fin Tarea ADC
}

#if defined(__XC8)
void interrupt SYS_InterruptHigh(void)
{
    static int indice = 0;
    char c;
    //Codigo para interrupciones de alta prioridad
    if(PIR3bits.RC2IF)      //Interrupcion por USART
    {
      if (RCSTA2bits.OERR)
      {
            RCSTA2bits.CREN = 0;
            c = ReadSIM_USART();
            RCSTA2bits.CREN = 1;
      }
        else
        {
            c = ReadSIM_USART();
        }
        if(simResponse.newResponse){
            //Reset de contadores para procesamiento de nueva respuesta
            simResponse.newResponse = FALSE;
            simResponse.countCRLF = 0;
            simResponse.enterResponse = FALSE;
            indice = 0;
        }
        if( c == 0x0A || c == 0x0D){
            simResponse.countCRLF++;
            if(simResponse.countCRLF >= CRLF_COUNT_PER_RESPONSE){
                simResponse.countCRLF = 0;
                simResponse.enterResponse = TRUE;
                intBuffer[indice] = '\0';
                //Copiamos respuesta en el buffer de respuesta de la SIM
                strcpy(simResponse.bufferRes,intBuffer);
                //simResponse.bufferRes[indice] = '\0';
                indice = 0;
            }
        }
        else{
```

```
            intBuffer[indice] = c;
            indice++;
        }
    }
}
#endif
```

Análisis:

Dentro del archivo system.c vamos a incluir la declaración de los valores de la palabra de configuración del Microcontrolador. Esta palabra de configuración contiene varios parámetros que indican como actuara el Microcontrolador, entre ellos se puede configurar el tipo de oscilador interno o externo, la habilitación del PLL interno, el Watch Dog Timer o la protección de la memoria. La configuración aconsejada para la placa trackme es la de utilizar un oscilador Externo, con el PLL habilitado, lo que incrementa por cuatro la frecuencia del oscilador externo y finalmente se aplica un divisor de frecuencia por dos, lo que baja a la mitad la frecuencia de salida del PLL. Como la placa tiene un cristal externo de 12MHz, la frecuencia con que se alimenta el clock del sistema es de 48MHz. Como se dijo anteriormente, es importante que la configuración del oscilador no se cambie, ya que no funcionaran los bloques de código que dependen del tiempo del sistema como la USART y los retardos.

Luego de la palabra de configuración, se define en el código la función de inicialización del sistema, SYSTEM_Initialize(); el cual comienza con un retardo para dejar estabilizar la frecuencia entregada por el PLL, luego configura los pines remapeables RP0 y RP1 para que actúen como los pines de Recepción y Trasmisión de la USART2 respectivamente, estos serán los encargados de comunicarse con el modulo SIM. Luego se configuran los pines GPIO para el LED de la placa, el botón de pánico y el pin de POWER_KEY del modulo SIM. A continuación las funciones USART_Initialize() y ADC_Init() inicializan el modulo USART2 y el conversor ADC, estas funciones se definen, como se dijo anteriormente, en las librería usart_trackme.h y adc_trackme.h respectivamente. Por último se habilitan las interrupciones de la USART2 para las recepciones, que mas adelante veremos la función de interrupción que se encargara de almacenar los datos recibidos por el modulo SIM.

La siguiente función definida en system.c es la que contiene las tareas que deben ejecutarse periódicamente en el sistema. Hasta el momento, la única función declarada en SYSTEM_Tasks() es la tarea del ADC, ADC_task(), la cual realiza periódicamente la conversión de los canales analógicos de la placa para mantener los resultados de la conversión actualizados dentro de la estructura ADCtrackme. Es importante destacar que las tareas del sistema se deben ejecutar periódicamente, por lo que no es aconsejable que se realicen tareas que se queden esperando la respuesta de un evento de manera indefinida, ya que esta función no se ejecutaría.

El archivo termina con la declaración de la rutina de interrupción, que en este caso solo tendremos una única fuente que genera las interrupciones y es la recepción de un carácter por la USART2, que es la interfaz conectada al modulo SIM. Basicamente esta rutina toma el

carácter recibido por la USART2 y lo almacena en un buffer interno y una vez que detecta que se ha recibido una respuesta completa, lo almacena en el buffer de una estructura global llamado simResponse, que por el momento no se utiliza, pero veremos que es de mucha utilidad en los próximos ejemplos, ya que uno de los parámetros que contiene esta estructura es la información de GPS y los parámetros de recepción de un mensaje de texto, si se habilita la recepción del mismo, pero esto lo veremos en los siguientes ejemplos.

Repasando lo que se vio hasta el momento, el primer ejemplo se resuelve mediante el código mostrado en los archivos main.c y system.c junto con las funciones que nos entregan las librerías de periféricos de la placa TrackMe. En los próximos ejemplos el archivo system.c no se modificara demasiado, por lo que en los próximos análisis solo nos concentraremos en los cambios introducidos con respecto al que acabamos de cotejar.

Ejemplo 2:

Al presionar el botón de pánico envía un SMS con las coordenadas en un link de google maps.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;           //Estrtucra que almacena los
resultados del ADC.
extern sSimResponse simResponse;        //Estructura global que contiene
respuestas del SIM5320

MAIN_RETURN main(void)
{
    char mensaje[256];
    char telefono[16]="1562823174";

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();
    //Se inicia una sesion GPS
    delay_s(1);
    SIM_WriteCommand(CGPS, OPEN_GPS_SESSION);
    while(1)
    {
        SYSTEM_Tasks();
        //La funcion BUTTON_ActivatePanic devuelve TRUE si se oprime el boton
de
        //panico de la placa TrackMe
        if(BUTTON_ActivatePanic())
        {
            LED_On(LED_USB_DEVICE_STATE);
            //Se oprimio el boton de panico, armar el mensaje a enviar
            sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
            //La funcion GPS_UpdateData actualiza los datos de lat y lon
            //en la estructura simResponse.gpsInfo
            GPS_UpdateData();
            //Armamos el Link a Google Maps
            sprintf(mensaje,"%shttp://maps.google.com/maps?q=%s,%s",mensaje,
simResponse.gpsInfo.latitude, simResponse.gpsInfo.longitude);

            sprintf(mensaje,"%s\r\nVelocidad: %1.2f",mensaje,
simResponse.gpsInfo.speed);

            //Envia mensaje
            GSM_SendSMS(telefono,mensaje);
        }

    }//end while
}//end main
```

Análisis:

El análisis de este ejemplo es muy similar al anterior, comenzamos inicializando los periféricos del Microcontrolador y luego encendemos e inicializamos el modulo SIM, pero a continuación enviamos un comando para abrir una sesión GPS mediante la función SIM_WriteCommand(); que acepta como parámetros el comando a enviar y el valor del mismo. El comando AT enviado es CGPS y el valor debe ser "1" para abrir una sesión GPS. Una vez enviado el comando, se entra en el bucle infinito que se encargara de actualizara las tareas del sistema y de monitorear el estado del pulsador. Cuando se detecta que se ha oprimido el botón de pánico, se comienza construir el mensaje SMS que se enviara por el modulo. Para poder armar un link de GoogleMaps es necesario obtener nuestras coordenadas actuales y esto lo realiza la función *GPS_UpdateData();* la cual se encarga de enviar el comando "AT+CGPSINFO" y procesara la respuesta del modulo para obtener las coordenadas de la posición actual (longitud y latitud) y las guardara en la estructura *simResponse.gpsInfo* para poder utilizarlas en la construcción del link de GoogleMaps. La estructura *simResponse* se declara de manera global en la librería del sim5320.h, por lo que cualquier archivo fuente puede tener acceso a ella si en su encabezado se la declara como *extern*, al igual que la estructura ADCtrackme que vimos en el ejemplo anterior.

Una vez que la función GPS_UpdateData termina de ejecutarse, nos devolverá la latitud en la variable *simResponse.gpsInfo.latitud* y la longitud en *simResponse.gpsInfo.longitud*. Entonces ya podemos armar el mensaje SMS que se entregara al modulo SIM, donde deberemos colocar el link con las coordenadas y se debe armar de la siguiente manera:

http://maps.google.com/maps?q=latitud,longitud

Utilizando nuevamente la función **sprintf** armamos el mensaje y llamamos a la rutina *GSM_SendSMS();* para enviarlo. Las coordenadas que devuelve el modulo GPS del SIM5320 estan en el formato Grados-Minutos Decimales o DMM, que es un formato soportado por GoogleMaps.

Por último, solo queda decir que el archivo *system.c* es el mismo que en el ejercicio anterior que ya fue analizado en su momento.

Ejemplo 3:

Cuando la placa TrackMe 3G recibe un SMS con la palabra "ACEL" envía a ese mismo teléfono el valor del acelerómetro

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;            //Estructura global que contiene
respuestas del SIM5320

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    while(1)
    {
        SYSTEM_Tasks();

        //Monitoreo la llegada de un Mensaje
        if(simResponse.smsRead.newSMS)
        {
            //LLegada de un nuevo mensaje
            simResponse.smsRead.newSMS = FALSE;     //Borro el flag
            //Leer el Mensaje y ejecutar las acciones correspondientes
            if(!stricmp(simResponse.smsRead.messenger,"ACEL"))
            {
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                sprintf(mensaje,"%sAcelX=%1.2f;",mensaje,ADCtrackme.Xout);
                sprintf(mensaje,"%sAcelY=%1.2f;",mensaje,ADCtrackme.Yout);
                sprintf(mensaje,"%sAcelZ=%1.2f;",mensaje,ADCtrackme.Zout);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
        }

    }//end while
}//end main
```

Análisis

main.c

Comenzamos el análisis del archivo fuente main.c donde se encuentra la ejecución principal del programa que espera recibir un mensaje de texto con un comando específico para enviar el estado del acelerómetro. Nuevamente volveremos a utilizar las estructuras de datos que nos provee la librería del ADC y del modulo SIM, ADCtrackme y simResponse, por lo que las declaramos en la zona superior del código como variables globales y con el prefijo **extern**. Al

comenzar la rutina principal, inicializamos los periféricos que utilizaremos del Microcontrolador, la USART, el ADC y los pines GPIO, utilizando la función *SYSTEM_Initialize()* que es igual al de los ejercicios anteriores y procedemos a encender el modulo e inicializarlo, con las funciones *SIM_On()* y *SIM_Init()*.

Una vez finalizado el proceso de inicialización, se entra en un bucle infinito que ejecuta periódicamente las tareas del sistema, con la llamada *SYSTEM_Tasks()*. En este ejemplo, a diferencia de los anteriores, no solo tendremos la tarea del ADC que continuamente se encuentra calculando la conversión de los periféricos analógicos de la placa para almacenarlo en la estructura ADCtrackme, si no que además se agrega una nueva tarea del sistema para el modulo SIM, llamado *SIM_Tasks()*. Esta tarea se debe ejecutar si se requiere utilizar la funcionalidad de recepción de mensajes SMS, ya que esta tarea monitorea si se produjo la llegada de un mensaje, lo cual es indicado con el estado booleano de la variable **simResponse.smsRead.newSMS**, y desglosa el comando recibido desde el modulo SIM para separar por un lado el numero telefónico de quien envió el mensaje y por el otro el mensaje recibido, ambos valores los almacena en la estructura global **simResponse** del modulo SIM y pueden encontrarse dichas variables en **simResponse.smsRead.phoneSMS** y **simResponse.smsRead.messenger** respectivamente. Es muy importante destacar que la tarea *SIM_Tasks()* no funcionaria si no se define la interrupción por la USART que recibe las respuestas del modulo SIM, ya que en ella existe una porción de código que detecta el comando de un mensaje entrante y le avisa a la tarea *SIM_Tasks*() que se está recibiendo un mensaje. En ese momento la tarea *SIM_Tasks*() espera la recepción completa del mensaje y una vez recibido todo, comienza a separar los campos del mensaje, para almacenarlo en la estructura y activar el flag de recepción de Mensaje. Más adelante veremos qué es lo que se debe agregar a la rutina de Interrupción para indicar que se está recibiendo un SMS desde el modulo SIM.

Volviendo al bucle infinito que se genera en la rutina principal del programa, lo que se hace a continuación de actualizar las tareas del sistema es monitorear el estado del flag **simResponse.smsRead.newSMS**, el cual se pone en **TRUE** si se detecta la llegada de un SMS (Recordemos que la tarea *SIM_Tasks()* es la encargada de actualizar este flag). Si este flag esta en TRUE, entonces tendremos acceso a los datos de la estructura que contiene el mensaje recibido y el numero telefónico del emisor de ese mensaje, y siempre recordar de colocar este flag en FALSE para esperar el siguiente mensaje.

Entonces, una vez detectado la llegada del mensaje se realiza la comparación del mensaje contenido en el SMS con el comando que ejecutara la acción correspondiente, en este caso el comando es "ACEL" y la acción a realizar es la de generar un SMS con el valor de los ejes del acelerómetro, y para armarlo haremos uso nuevamente de la funcion **sprintf** y de la estructura del ADC que nos devuelve el valor actualizado de los ejes en las variables **ADCtrackme.Xout**, **ADCtrackme.Yout** y **ADCtrackme.Zout**. Por último solo queda enviar el mensaje a través de la

función *GSM_SendSMS()* y utilizando como destinatario el numero telefónico que se obtuvo de los campos del SMS recibido y que se almacena en la variable **simResponse.smsRead.phoneSMS**.

system.c

```
void SYSTEM_Tasks(void)
{
    //Tareas del sistema

//-----Tarea del ADC
    ADC_task();
//-----Fin Tarea ADC
//-----Tarea Modulo SIM
    SIM_Tasks();
//-----Fin Tarea SIM
}

#if defined(__XC8)
void interrupt SYS_InterruptHigh(void)
{
    static int indice = 0;
    char c;
    //Codigo para interrupciones de alta prioridad
    if(PIR3bits.RC2IF)      //Interrupcion por USART
    {
      if (RCSTA2bits.OERR)  // in case of overrun error
      {
            RCSTA2bits.CREN = 0;  // reset the port
            c = ReadSIM_USART();
            RCSTA2bits.CREN = 1;  // and keep going.
      }
         else
        {
            c = ReadSIM_USART();
        }
        if(simResponse.newResponse){
            //Reset de contadores para procesamiento de nueva respuesta
            simResponse.newResponse = FALSE;
            simResponse.countCRLF = 0;
            simResponse.enterResponse = FALSE;
            indice = 0;
        }

        intBuffer[indice++] = c;
        intBuffer[indice] = '\0';   //Cierro la cadena de String
        //indice++;

        if( c == 0x0A || c == 0x0D){
            simResponse.countCRLF++;
            if(simResponse.countCRLF >= simResponse.maxCRLF){
                simResponse.countCRLF = 0;
                simResponse.enterResponse = TRUE;
                intBuffer[indice] = '\0';
                //Copiamos respuesta en el buffer de respuesta de la SIM
                strcpy(simResponse.bufferRes,intBuffer);
                //simResponse.bufferRes[indice] = '\0';
                indice = 0;
            }
```

```
        }
        if( c == 'T'){
            //observo si los valores anteriores fueron +CMT, pues esto quiere
            // decir que se esta recibiendo un mensaje.
            if(indice > 3){
                if((intBuffer[indice-2]=='M')&&(intBuffer[indice-
3]=='C')&&(intBuffer[indice-4]=='+')){
                    //Se tiene que asegurar que el mensaje se va a procesar
                    simResponse.smsRead.receiveSMS = TRUE;
                    simResponse.maxCRLF = simResponse.countCRLF + 4;
                }
            }
        }
    }
}
#endif
```

system.c

El código fuente de system.c va a diferir ligeramente de los ejercicios anteriores, ya que se agregara una nueva tarea en la función *SYSTEM_Tasks()* y se agrega una porción de código en la interrupción de la recepción de un carácter de la USART para detectar el comando de un nuevo SMS desde el modulo SIM.

Entonces, comenzando con la función *SYSTEM_Tasks(),* se debe agregar a continuación de la tarea *ADC_task()*, la tarea *SIM_Tasks()*, para que se ejecute periódicamente. Esta tarea trabaja de manera conjunta con la rutina de interrupción de la USART, ya que esta última le avisa cuando se recibe el comando de un nuevo mensaje entrante. En ese caso, la tarea *SIM_Tasks()* cambiara su estado para esperar a recibir el mensaje completo y luego procesarlo.

En la rutina de interrupción se agrega un segmento de código que busca entre los caracteres recibidos la recepción de la letra "T", si la encuentra observa los valores anteriores para identificar si lo que se está recibiendo es el comando "+CMT". Si es así, se modifica el valor de la variable **simResponse.maxCRLF** para asegurarse de que se va a recibir el comando de la SIM completo. El comando "+CMT" del modulo SIM tiene el siguiente formato: <CR><LF>+CMT: "Telefono","parametros SMS"<CR><LF>Mensaje<CR><LF>, esto luego es tomado por la tarea SIM_Tasks, el cual desglosa los campos y lo almacena en la estructura **simResponse.smsRead**.

Ejemplo 4:

Cuando la placa TrackMe 3G recibe un SMS con la palabra "TEMP" envía a ese mismo teléfono el valor de la temperatura.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    while(1)
    {
        SYSTEM_Tasks();

        if(simResponse.smsRead.newSMS)
        {
            //LLegada de un nuevo mensaje
            simResponse.smsRead.newSMS = FALSE;     //Borro el flag
            //Leer el Mensaje y ejecutar las acciones correspondientes
            if(!stricmp(simResponse.smsRead.messenger,"TEMP"))
            {
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                sprintf(mensaje,"%sTemp=%1.2f",mensaje,ADCtrackme.temp);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
        }

    }//end while

}//end main
```

Análisis

En los siguientes ejemplos, en donde se recibe un mensaje de texto SMS para realizar alguna acción, la estructura de los programas es muy similar, se comienza inicializando todos los periféricos del Microcontrolador y encendiendo el módulo SIM y en el bucle principal se ejecutan las tareas del sistema, que en este caso son del ADC y del modulo SIM, y luego se monitorea la llegada de un nuevo mensaje con el comando a realizar para luego ejecutar o no la acción. Todo este análisis ya se realizo en el ejemplo anterior, ahora nos concentraremos en el monitoreo de la llegada de un mensaje, para lo cual utilizamos la variable booleana **simResponse.smsRead.newSMS**, que será TRUE si la tarea *SIM_Tasks()* completa el procesamiento de un nuevo mensaje. Una vez que se termina de recibir el mensaje, se ejecuta el bloque condicional, que lo primero que realiza es borrar el flag de recepcion de mensaje y se

compara el texto del mensaje con el comando que se espera recibir. Notar que la comparación se realiza con la función **stricmp** y no con **strcmp**, y la diferencia en los nombres de estas funciones se encuentra en esa letra "i" en el medio del nombre de la función que utilizamos y que significa que la comparación se realiza en el modo insensitive case, es decir, que no se tiene en cuenta las mayúsculas y minúsculas en la comparación. Una vez que se detecta el comando dentro del mensaje recibido se ejecuta la acción, que en este caso es devolver un SMS con el valor de la temperatura. Otra vez utilizamos la función **sprintf** para armar el mensaje y utilizamos el valor de la temperatura que se almacena en la variable **ADCtrackme.temp**. Por último se envía el mensaje utilizando la función *GSM_SendSMS().*

Ejemplo 5:

Cuando la placa TrackMe 3G recibe un SMS con la palabra "LUZ" envía a ese mismo teléfono el valor de la intensidad lumínica.

main.c

```
/** INCLUDES *******************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    while(1)
    {
        SYSTEM_Tasks();

        if(simResponse.smsRead.newSMS)
        {
            //LLegada de un nuevo mensaje
            simResponse.smsRead.newSMS = FALSE;     //Borro el flag
            //Leer el Mensaje y ejecutar las acciones correspondientes
            if(!stricmp(simResponse.smsRead.messenger,"LUZ"))
            {
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                sprintf(mensaje,"%sLDR=%i\r\n",mensaje,ADCtrackme.ldr);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
        }

    }//end while

}//end main
```

Análisis

El código de este ejemplo es igual al anterior, lo único diferente es la acción a realizar una vez que se recibe el mensaje, que en este caso debe enviar el valor de la conversión del LDR. El valor de esta variable analógica se almacena en ADCtrakme.ldr, y se lo utiliza para construir el mensaje que responde al comando “LUZ”. Nuevamente utilizamos *GSM_SendSMS()* para enviar el mensaje de texto.

Ejemplo 6:

Cuando la placa TrackMe 3G recibe un SMS con la palabra "GPS" envía a ese mismo teléfono las coordenadas en un link de Google Maps.

main.c

```
/** INCLUDES *******************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();
    //Se inicia una sesion GPS
    delay_s(1);
    SIM_WriteCommand(CGPS, OPEN_GPS_SESSION);
    while(1)
    {
        SYSTEM_Tasks();

        if(simResponse.smsRead.newSMS)
        {
            //LLegada de un nuevo mensaje
            simResponse.smsRead.newSMS = FALSE;     //Borro el flag
            //Leer el Mensaje y ejecutar las acciones correspondientes
            if(!stricmp(simResponse.smsRead.messenger,"GPS"))
            {
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                GPS_UpdateData();

sprintf(mensaje,"%shttp://maps.google.com/maps?q=%s,%s",mensaje,
simResponse.gpsInfo.latitude, simResponse.gpsInfo.longitude);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
        }

    }//end while

}//end main
```

Análisis

Comenzamos la inicialización de los periféricos del Microcontrolador y encendemos el modulo de la misma manera que lo hicimos en los ejercicios anteriores y se debe agregar además el inicio de sesión GPS, que lo realiza la instrucción *SIM_WriteCommand(CGPS, OPEN_GPS_SESSION).* Luego tenemos que esperar la recepción del mensaje que envié el comando “GPS” y una vez que lo recibimos, se arma el mensaje de respuesta con el enlace a

Google Maps y para obtener las coordenadas actuales utilizamos la función *GPS_UpdateData();* que llenara las variables de coordenadas que se alojan dentro de la estructura simResponse.

Ejemplo 7:

Cuando la placa TrackMe 3G recibe un SMS con la palabra "ACTIVAR" enciende el LED conectado al PIC y con “DESACTIVAR” lo apaga.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    while(1)
    {
        SYSTEM_Tasks();

        if(simResponse.smsRead.newSMS)
        {
            //LLegada de un nuevo mensaje
            simResponse.smsRead.newSMS = FALSE;     //Borro el flag
            //Leer el Mensaje y ejecutar las acciones correspondientes
            if(!stricmp(simResponse.smsRead.messenger,"ACTIVAR"))
            {
                LED_On(LED_USB_DEVICE_STATE);
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                sprintf(mensaje,"%sLED ON\r\n",mensaje);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
            if(!stricmp(simResponse.smsRead.messenger,"DESACTIVAR"))
            {
                LED_Off(LED_USB_DEVICE_STATE);
                sprintf(mensaje,"SMS de la placa TrackMe.\r\n");
                sprintf(mensaje,"%sLED OFF\r\n",mensaje);
                GSM_SendSMS(simResponse.smsRead.phoneSMS,mensaje);
            }
        }

    }//end while

}//end main
```

Análisis

En este caso los mensajes recibidos deben realizar acciones concretas sobre algún periférico de la placa, como el LED conectado al pin RB2, que la librería de la placa TrackMe lo declara con el nombre **LED_USB_DEVICE_STATE** y utilizamos las funciones del archivo “leds.h” para manejar este LED. Entonces en el caso de recibir el mensaje con el comando “ACTIVAR” se encenderá el

LED de la placa con la función *LED_On(LED_USB_DEVICE_STATE)* y enviaremos un mensaje de respuesta indicando de que la operación se realizo satisfactoriamente. Del mismo modo se procede al recibir un mensaje con el comando “DESACTIVAR” el cual apagara el LED mediante la ejecución de la función *LED_Off(LED_USB_DEVICE_STATE)* y envía el mensaje de respuesta ante la acción solicitada indicando el estado del LED.

Ejemplo 8:

Guardar cada 10 segundos o un tiempo que se pueda modificar, la trama NMEA completa en un pendrive conectado al FTDI de la placa.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;
extern int32_t sysUpTime;          //tiempo en segundos del inicio el sistema

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];
    int  intervaloNMEA;
    int  lastNMEArequest;

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();
    //Se inicia una sesion GPS
    delay_s(1);
    SIM_WriteCommand(CGPS, OPEN_GPS_SESSION);

    //Intervalo de peticiones de tramas NMEA en segundos
    intervaloNMEA = 10;
    lastNMEArequest = sysUpTime;
    while(1)
    {
        SYSTEM_Tasks();

        if(sysUpTime > (lastNMEArequest + intervaloNMEA))
        {
            lastNMEArequest = sysUpTime;
            //Envio comando GPS para obtener trama NMEA
            putrsSIM_USART("AT+CGPSINFOCFG=255,31\r");
            //Espero a que se reciba la respuesta
            SIM_ReadResponse(CRLF_COUNT_GPS_NMEA);
            //sumamos un 6 para apuntar al primer caracter de la trama NMEA
            putsFTDI_UART(simResponse.bufferRes+6);

            //Envio comando GPS para detener trama NMEA
            SIM_WriteCommand(CGPSINFOCFG, NMEA_GPS_STOP);
        }

    }//end while

}//end main
```

Análisis

La placa TrackMe tiene la posibilidad de conectar un pendrive para realizar el almacenamiento de datos, logrando realizar un data logger de manera simple y eficaz y con gran capacidad de

almacenamiento. La interfaz de comunicación entre el Microcontrolador y el Pendrive está efectuada mediante un chip de la empresa FTDI, en el cual el Microcontrolador solamente debe enviar los caracteres a almacenar por la USART conectado al chip y este será el responsable de crear el archivo de texto que alojara los datos enviados, evitando tener que utilizar el manejo del sistema de archivo como FAT32, que es necesario para utilizar un pendrive. Así que las funciones del Microcontrolador para almacenar datos en un archivo ubicado en el Pendrive se limitan nada mas a enviar comandos por la USART donde está conectado el chip FTDI. La implementación por USART para la comunicación con el FTDI se debe realizar por Software y la librería de la placa TrackMe nos provee las funciones para realizar esta comunicación en el archivo "sw_uart_trackme.h".

Comenzando con el análisis del código, se inicializan los periféricos del PIC agregando la configuración del TIMER3 que será utilizado para contar los segundos desde que se inicia la placa y almacenara este valor en una variable global llamado *sysUpTime,* que nos será de utilidad para contar los intervalos de tiempo en que debemos escribir la trama NMEA. Creamos además dos variables que nos ayudaran a contar estos intervalos de tiempo en los cuales se escriben las tramas, una de las variables se llama *intervaloNMEA* que contiene el tiempo entra cada escritura de la trama NMEA en segundos y la variable *lastNMEArequest* que almacena el tiempo en que se realizo la última actualización de la trama NMEA en el pendrive. Entonces una vez que iniciamos la sesión GPS, comenzamos a ejecutar el bucle principal que actualiza las tereas del sistema y observa los valores de estas dos variables junto con el tiempo del sistema y determina si es momento de realizar la actualización o no. Si ya se cumplió el tiempo de actualización, se almacena el tiempo actual en la variable *lastNMEArequest* y se procede a ejecutar el comando "AT+ CGPSINFOCFG=255,31" a lo que el modulo SIM responde con la trama NMEA completa. La función *SIM_ReadResponse(CRLF_COUNT_GPS_NMEA)* leera la trama completa y la almacena en **simResponse.bufferRes** para que a continuación lo podamos enviar al FTDI mediante la rutina *putsFTDI_UART().* El último paso es el envió de un comando para detener la reproducción de la trama NMEA, evitando que el buffer de recepción se llene con las sucesivas respuestas.

system.c

```
void SYSTEM_Initialize( SYSTEM_STATE state )
{
    switch(state)
    {
        case SYSTEM_STATE_START:
        {
             {
                 unsigned int pll_startup_counter = 600;
                 OSCTUNEbits.PLLEN = 1;
                 while(pll_startup_counter--);
             }
            // Init PPS UART2
            PPSUnLock();
            PPSInput(PPS_RX2DT2,PPS_RP0);
            PPSOutput(PPS_RP1,PPS_TX2CK2);
```

```
            PPSLock();

            LED_Enable(LED_USB_DEVICE_STATE);
            LED_Off(LED_USB_DEVICE_STATE);
            BUTTON_Enable(BUTTON_TRACKME_PANIC);
            GSM_POWER_TRIS = PIN_OUTPUT;   //RC1 output
            GSM_POWER_LAT = GSM_OFF;   //Valor por defecto del pin
            USART_Initialize();
            ADC_Init();
            OpenFTDI_UART();

            //Inicializacion de la variable tiempo de sistema
            sysUpTime = 0;
            //Configuracion del Timer 3
            T3GCON = 0x00;
            T3CON = 0x32;  //Prescaler en 1:8
            //Habilitar Interrupciones del Timer 3
            PIE2bits.TMR3IE = 1;
            WriteTimer3(TIMER3_VALUE);
            //Encendemos el timer 3
            T3CONbits.TMR3ON = 1;

            //Habilitamos interrupciones de perifericos
            INTCONbits.PEIE = 1; //Activamos interrupcion de perifericos
            INTCONbits.GIE = 1;  //Activamos las interrupciones globales
            break;
        }
    }
}

#if defined(__XC8)
void interrupt SYS_InterruptHigh(void)
{
    static int indice = 0;
    static int counter = 0;
    char c;

    if(PIR2bits.TMR3IF)
    {
        WriteTimer3(TIMER3_VALUE);
        PIR2bits.TMR3IF = 0;       //Borro Flag
        counter++;
        if(counter == MAX_COUNTER)
        {
            counter = 0;
            sysUpTime++;    //paso 1 segundo, incremento variable sistema
        }
    }
    //Codigo para interrupciones de alta prioridad
    if(PIR3bits.RC2IF)      //Interrupcion por USART
    {
      if (RCSTA2bits.OERR)  // in case of overrun error
      {
            RCSTA2bits.CREN = 0;  // reset the port
            c = ReadSIM_USART();
            RCSTA2bits.CREN = 1;  // and keep going.
      }
          else
        {
```

```
            c = ReadSIM_USART();
        }
        if(simResponse.newResponse){
            //Reset de contadores para procesamiento de nueva respuesta
            simResponse.newResponse = FALSE;
            simResponse.countCRLF = 0;
            simResponse.enterResponse = FALSE;
            indice = 0;
        }

        intBuffer[indice++] = c;
        intBuffer[indice] = '\0';   //Cierro la cadena de String
        //indice++;

        if( c == 0x0A || c == 0x0D){
            simResponse.countCRLF++;
            if(simResponse.countCRLF >= simResponse.maxCRLF){
                simResponse.countCRLF = 0;
                simResponse.enterResponse = TRUE;
                intBuffer[indice] = '\0';
                //Copiamos respuesta en el buffer de respuesta de la SIM
                strcpy(simResponse.bufferRes,intBuffer);
                //simResponse.bufferRes[indice] = '\0';
                indice = 0;
            }
        }
        if( c == 'T'){
            //observo si los valores anteriores fueron +CMT, pues esto quiere
decir que
            //se esta recibiendo un mensaje.
            if(indice > 3){
                if((intBuffer[indice-2]=='M')&&(intBuffer[indice-
3]=='C')&&(intBuffer[indice-4]=='+')){
                    //Se tiene que asegurar que el mensaje se va a procesar
                    simResponse.smsRead.receiveSMS = TRUE;
                    simResponse.maxCRLF = simResponse.countCRLF + 4;
                }
            }
        }
    }
}
```

Análisis

En el archivo fuente del sistema se agregan la configuración del TIMER 3 utilizado para contar los segundos desde que se inicia el sistema y la rutina de interrupción que actualiza el conteo del temporizador.

Ejemplo 9:

Sincronizar el valor de la temperatura de la placa TrackMe 3G con los servidores de Ubidots.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>
#include <ubidots.h>

//Token del Dispositivo
#define UBI_TOKEN_AUTH_TRACKME          "0MiJaxYS6ikWwlnTkKjQC7cc0bR2gl"
//IDs de las variables
#define UBI_ID_TEMP                     "55528b5a762542380f69f870"

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;
extern int32_t sysUpTime;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    char data;
    int  intervaloUpData;
    int  lastUpDatarequest;

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    //Intervalo de peticiones de tramas NMEA en segundos
    intervaloUpData = 30;
    lastUpDatarequest = sysUpTime;
    //Configurar datos del APN
    delay_s(1);
    SIM_WriteCommand(CGSOCKCOUNT, APN_PERSONAL);
    SIM_WriteCommand(CSOCKAUTH, APN_USER_PASS_PERSONAL);

    while(1)
    {
        SYSTEM_Tasks();

        if(sysUpTime > (lastUpDatarequest + intervaloUpData))
        {
            lastUpDatarequest = sysUpTime;
            // Envio Temperatura
    UBI_SendData(UBI_TOKEN_AUTH_TRACKME, UBI_ID_TEMP, ADCtrackme.temp, FALSE);


        }

    }//end while

}//end main
```

Análisis

Para comunicarnos con los servidores de Ubidots, las librerías de la placa TrackMe implementa funciones para enviar los datos y sincronizarlos con el servidor, solo es necesario proveer de una conexión de datos para poder abrir un socket. Comenzamos el programa inicializando los periféricos y el modulo SIM, luego enviamos los comandos “AT+ CGSOCKCOUNT” y “AT+ CSOCKAUTH” para enviar los datos del APN. Un ejemplo para la configuración del APN de la empresa de Telefonía Claro es el siguiente:

AT+CGSOCKCONT=1,"IP"," internet.ctimovil.com.ar"

AT+CSOCKAUTH=1,2," ctigprs "," ctigprs "

Una vez configurado los datos del APN ya estamos en condiciones de realizar los llamados a la rutina que se comunicara con el servidor de Ubidots, que se realizara cada vez que se cumpla el intervalo de actualización de datos que esta dado por la variable *intervaloUpData.*

La rutina encargada de sincronizar los datos con Ubidots se llama *UBI_SendData()* que necesita como parámetros el **Token de Autenticación** del dispositivo y el **ID de la variable**, que son datos que se deben tomar al crear las variables de monitoreo en la página de Ubidots, además también se debe pasar como parámetro de esta función el valor de la variable a sincronizar y un booleano indicando si se desea enviar las coordenadas de la posición actual junto con el valor o no. En este caso se sincroniza el valor de la temperatura, del cual previamente obtuvimos el token de autenticación y el ID del mismo y lo pasamos como parámetro a través de los defines UBI_TOKEN_TRACKME y UBI_AUTH_ID_TEMP.

La función *UBI_SenData()* se encarga de abrir el socket para la comunicación de datos con el servidor y los envía mediante el método POST de HTTP, luego se recibe la respuesta y se cierra el socket para iniciar luego otro envió de datos.

Ejemplo 10:

Sincronizar el valor de la temperatura de la placa TrackMe 3G con los servidores de Ubidots junto con las coordenadas actuales de la placa.

main.c

```
/** INCLUDES *************************************************/
#include <system.h>
#include <sim5320.h>
#include <ubidots.h>

//Token del Dispositivo
#define UBI_TOKEN_AUTH_TRACKME          "0MiJaxYS6ikWwlnTkKjQC7cc0bR2gl"
//IDs de las variables
#define UBI_ID_TEMP                     "55528b5a762542380f69f870"

extern sADCResult ADCtrackme;
extern sSimResponse simResponse;
extern int32_t sysUpTime;

MAIN_RETURN main(void)
{
    char mensaje[SIZE_SMS];

    char data;
    int  intervaloUpData;
    int  lastUpDatarequest;

    SYSTEM_Initialize(SYSTEM_STATE_START);

    SIM_On();
    SIM_Init();

    //Intervalo de peticiones de tramas NMEA en segundos
    intervaloUpData = 10;
    lastUpDatarequest = sysUpTime;
    //Configurar datos del APN
    delay_s(1);
    SIM_WriteCommand(CGSOCKCOUNT, APN_PERSONAL);
    SIM_WriteCommand(CSOCKAUTH, APN_USER_PASS_PERSONAL);

    //Se inicia una sesion GPS
    delay_s(1);
    SIM_WriteCommand(CGPS, OPEN_GPS_SESSION);

    while(1)
    {
        SYSTEM_Tasks();

        if(sysUpTime > (lastUpDatarequest + intervaloUpData))
        {
            // Envio Temperatura
            if(UBI_SendData(UBI_TOKEN_AUTH_TRACKME, UBI_ID_TEMP,
ADCtrackme.temp, TRUE)==SIM_STATUS_OK)
            {
                lastUpDatarequest = sysUpTime;
            }
        }
```

```
    }//end while
}//end main
```

Análisis

Este ejemplo es igual al anterior, solo que además de enviar el dato de la temperatura se debe anexar las coordenadas de la posición actual de la placa y esto mismo lo realiza la función *UBI_SendData().* El último parámetro que se le pasa a esta función es una variable booleana que indica si queremos enviar los datos de la coordenada actual junto con la variable a sincronizar o no, si es TRUE, se envía las coordenadas.

Los coordenadas GPS que deben enviarse a los servidores de Ubidots deben ser entregados en grados decimales, pero la trama NMEA los entrega en formato de grados y minutos decimales, por lo que se debe realizar la conversión. En la librería del sim5320.h existe una función que realiza esta conversión y es utilizado para enviar las coordenadas actuales. Debido a que el compilador XC8 utiliza float a 24 bits como tamaño por defecto, es necesario entrar a las propiedades del proyecto, y en la sección XC8 linker y en la pestaña “Memory Model” seleccionar el tamaño de las variables double y float a 32 bits.

Si la función GPS_UpdateData() no puede obtener los valores de coordenadas, entonces no lo envía, hasta que el GPS pueda recibir datos, mientras solo sincroniza los valores de la temperatura.