Práctica 5. Comunicaciones serie

(Duración: 2 sesiones)

Introducción

En esta práctica nos familiarizaremos con las comunicaciones serie usando un módulo Bluetooth para comunicarnos con un ordenador o con un teléfono móvil mediante la UART del PIC32MX230F064Dya sea por *polling* (sesión I) o usando interrupciones y colas (sesión II).

Bluetooth Low Energy (BLE) es una tecnología de comunicación inalámbrica optimizada para dispositivos de bajo consumo, como sensores y dispositivos portátiles. La estructura básica de comunicación en BLE se basa en servicios y características. Un servicio es un conjunto lógico de funcionalidades relacionadas, identificado por un UUID, y agrupa varias características, que son las unidades mínimas de datos intercambiables. Cada característica también posee un UUID y puede ser leída, escrita o enviada mediante notificaciones.

Por ejemplo, un servicio de "batería" podría tener una característica que indique el nivel de carga. BLE funciona bajo un modelo cliente-servidor: el dispositivo servidor (como un sensor) expone sus servicios y características, mientras que el cliente (como un smartphone) se conecta, descubre esos servicios y realiza operaciones sobre sus características. Estas operaciones incluyen Read (leer datos), Write (escribir datos), Notify (el servidor envía actualizaciones automáticamente) e Indicate (similar a Notify, pero requiere confirmación del cliente). Esta estructura modular y eficiente permite una comunicación flexible, segura y de bajo consumo energético, ideal para aplicaciones en el Internet de las Cosas (IoT).

Objetivos

- Configurar la UART del microcontrolador para enviar y recibir caracteres, en modo *polling*.
- Conectar el módulo HM–10 para comunicarse por Bluetooth LE con un móvil (IOS o Android) o con un PC por medio de la UART.
- Configurar la UART del microcontrolador para enviar y recibir caracteres usando interrupciones.
- Usar colas para comunicar las interrupciones con el programa principal.
- Escribir un intérprete de comandos sencillo.

Crear un proyecto

En primer lugar ha de crear un proyecto para esta práctica, siguiendo los pasos indicados en el apartado 2 de la primera práctica. No olvide trabajar en la carpeta `D:▸Micros▸Grupo_XX`, creada en la práctica 1.

Al igual que en la práctica 3, es necesario configurar la tarjeta para que funcione con el oscilador de cuarzo externo. Como recordará basta con incluir en el proyecto el archivo `Pic32Ini.c` disponible en Moodle.

Sesión I: UART por polling

Trabajo previo

Antes de ir al laboratorio ha de:

- Leer detenidamente la práctica, anotando las dudas que le surjan para preguntárselas al profesor.
- Escribir los programas solicitados en las secciones 2 y 3.
- Descargar alguna de las formas o varias indicadas en la sección de Terminales del siguiente repositorio Github para probarla durante el laboratorio.

1. Prueba del módulo HM–10 y del programa de comunicaciones

En la práctica se va a usar el módulo HM–10 que nos permite comunicarnos fácilmente con un microcontrolador mediante Bluetooth. El HM–10 funciona como puente entre un ordenador o un móvil al que se conecta mediante Bluetooth y un microcontrolador al que se conecta mediante un interfaz serie asíncrono. Por defecto el interfaz serie funciona a 9600 baudios en formato 8N1 y con niveles de tensión de 0 a 5 V.

Conecte en primer lugar el módulo HM–10 a la placa tal como se muestra en la figura 1. Sí, ya se que en la figura se muestra un Arduino, pero recuerde que el PicTranier32 es compatible con Arduino.

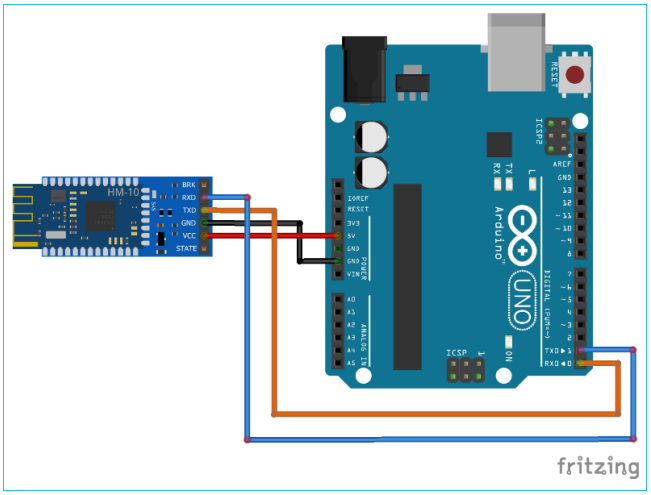

Figura 1. Conexión del HM–10a la placa..

A continuación ha de configurar el PC o su móvil Android/IOS para poder comunicarse con el HM–10. Recuerde que las opciones para los terminales para BLE se encuentran en el siguiente repositorio Github para probarla durante el laboratorio. Existen otras para módulos no BLE más antiguas que no sirven.

El proceso es similar en todas ellas:

1. Hay que escanear (SCAN) para que aparezcan los dispositivos cercanos y buscar aquel que se llame BT05-XX, siendo XX el de nuestro dispositivo.

2. Una vez encontrado habrá que seleccionarlo y conectarse a él. En este momento el LED del HM–10 pasará de estar parpadeando a estar fijo para indicarnos que se ha establecido la conexión.
3. Desde el terminal podemos teclear mensajes en algún cuadro de texto y éstos se enviarán al HM–10 (y este lo enviará al microcontrolador por la UART). De la misma manera, lo que se envie desde el microcontrolador, se recibe por el HM–10 y se imprimirá en la terminal.

Tanto si usamos el móvil como si usamos el PC, es conveniente probar que el enlace está bien. Para ello cortocircuite o junte los terminales RXD y TXD del HM–10 (por ejemplo con la protoboard) para formar un bucle de forma que lo que se reciba en el HM–10 se vuelva a enviar de vuelta. De ese modo, lo que teclee en el terminal aparecerá en la pantalla. Una vez verificado el funcionamiento, vuelva a dejar conectado el módulo a la tarjeta como se muestra en la figura 1.

2. Recepción de caracteres mediante la UART1

En primer lugar es necesario hacer un programa que reciba un carácter desde la UART1 y muestre los cuatro bits menos significativos de dicho carácter en los LEDs de la placa. Tenga en cuenta para ello que:

- El HM–10 envía datos a 9600 baudios y en formato 8N1.
- El pin de recepción de la UART1, `U1RX`, se conectará al pin RB13.
- La recepción se gestionará por *polling*.
- El receptor se mantendrá siempre habilitado (bit `URXEN` a 1).

3. Transmisión de caracteres mediante la UART1

Ahora es necesario escribir un programa que envíe una cadena de caracteres (por ejemplo `"Hola mundo"` para variar) por la UART1 cada vez que se pulse el pulsador de la placa. Para ello tenga en cuenta que:

- Al igual que antes, el HM–10 recibe datos a 9600 baudios y en formato 8N1.
- El pin de transmisión de la UART1, `U1TX`, se conectará al pin RB7.
- La transmisión se gestionará por *polling*.
- El transmisor se habilitará cuando se vaya a enviar la cadena y se inhabilitará cuando se termine de enviar.

4. Transmisión y recepción de caracteres

Por último realice un programa que una los dos anteriores: cuando se reciba un carácter por la UART1 se copiarán los 4 bits menos significativos a los LEDS de la placa y cuando se pulse el pulsador se enviará una cadena por la UART1.

5. Eco

Realice un programa que envíe por la UART1 los caracteres que reciba para hacer un eco por software, en lugar de por hardware como hizo al principio de la práctica para probar el módulo.

6. Usando MIT APP INVENTOR o la terminal de python, elabore algo propio para el control de leds de la placa (opcional)

Si le queda tiempo, puede probar a modificar las terminales y añadir gráficos o botones pare enviar ordenes y modificar leds en la placa o enviar desde el micro valores que reflejen el estado de un led.

7. Eco por interrupciones (opcional)

Si le queda tiempo, haga este apartado pero con la uart con interrupciones simples de un carácter para empezar a practicar de cara a la siguiente sesión.

Sesión II: UART por interrupciones

Trabajo previo

Antes de ir al laboratorio ha de:

- Leer detenidamente la práctica, anotando las dudas que le surjan para preguntárselas al profesor.
- Escribir los programas solicitados en las secciones 8 (el módulo descrito usando estructuras) y 9.1 ("Traiga")

8. Módulo para la UART1 con colas e interrupciones

En primer lugar ha de crear un módulo, denominado `UART1` en el que se incluirán todas las funciones para el manejo de la UART usando colas e interrupciones. Dicho módulo ha de incluir:

- Dos colas, una para el transmisor y otra para el receptor. Para que el código esté más claro, cada una de estas colas ha de definirse como una estructura de datos junto con sus índices.
- Una función para inicializar el módulo con el prototipo:

  ```
  void InicializarUART1(int baudios);
  ```

  La función será similar a la expuesta en clase pero añadiendo la posibilidad de elegir la velocidad de la línea serie. Tenga en cuenta que para velocidades superiores a 38 400 baudios hay que usar el divisor por 4 en lugar del divisor por 16, pues de lo contrario se cometerá demasiado error.
- Una función para introducir cadenas de caracteres en la cola:

  ```
  void putsUART(char s[]);
  ```

- Una función para extraer un carácter de la cola.

  ```
  char getcUART(void);
  ```

- La rutina de atención a la interrupción.

Recuerde que todas estas funciones han sido vistas en clase. Tan sólo tendrá que modificar la función de inicialización para incluir la selección de la velocidad deseada.

Para probar este módulo se creará un programa que haga el eco de los caracteres recibidos por la UART. Para ello, puede usar la función `putsUART` y enviar una cadena ("X") o vector de char con un sólo carácter y el segundo el carácter de fin de cadena (`c[2]; c[0]='X'; c[1]='\0';`). Otra opción, es crear una función, denominada por ejemplo `putcUART`, que introduzca un sólo carácter en la cola y active las interrupciones para que se envíe.

9. Bluetooth bit whacker

Una vez comprobado que el módulo de la UART funciona correctamente, se escribirá un programa para poder **controlar los pines de los puertos B y C** de la tarjeta desde el PC o el móvil. Para ello, habra que mandar unos comandos o ordenes al micro codificados de alguna forma.

9.1. Funciones útiles

De cara a poder realizar el siguiente apartado, adelante alguna de las funciones y traiga estudiadas las distintas opciones para enviar la respuesta:

- Configurar el terminal para que envié un carácter de terminación `'\n'` o cualquier otro al final. Cuando se reciba dicho carácter se asume que ha llegado la orden completa que se ha ido guardando en una cadena o cola. Para realizar el programa, en el bucle de *scan* se llamará a la función `getcUART` y se irán copiando los caracteres que se vayan recibiendo a una cadena. Cuando se reciba la terminación, se analizará la cadena con `ProcesaOrden(orden)` para interpretar el comando y ejecutarlo, devolviendo con `putsUART` el resultado de dicho comando.
- Los números vendrán en hexadecimal `'0'->'9'`o `'A'-> 'F'`. Habrá que transformarlo para que sea un número en una función (int toInt(char c)). Para ello, recuerde la tabla ASCII. **Traiga esta función hecha.**
- Las letras pueden venir en mayúsculas o minúsculas, mejor unificarlas a mayúsculas antes del procesado. `'a'->'z'`o `'A'-> 'Z'`. Si importa `#include <ctype.h>` podrá usar `toupper(c)` para transformar el carácter c en ese mismo carácter pero en mayúscula.
- **Traiga un prototipo de la función `ProcesaOrden(orden)` para la instrucción PI** del apartado siguiente. El resto se completará en clase.
- Escribir strings en c puede resultar tedioso. Aquí van varias formas en función de si queremos hacerlo o no sobre una variable o si en un string queremos poner algún número.

```
Funciones útiles para enviar:
#include <stdio.h>
#include <string.h>
a) char respuesta[TAM];
   putsUART("Error\n");
b) strcpy(respuesta, "Error\n");
   putsUART(respuesta);
c) sprintf(respuesta, "PI, %d\n", 1); /%d para int %f para float
   putsUART(respuesta);
d) char respuesta[]="Hola Mundo\n";
e) char grados_str[3];
   char dist_str[3];
   //Pasar número a string
   itoa(valor_str, 1, 10);
   //Vacia la cadena
   memset(respuesta, 0, 10);
   //Concatena
   strcat(respuesta, "PI,");
   strcat(respuesta, valor_str);
   strcat(respuesta, "\n");
```

9.2. Programa para manejo de los puertos B y C

Implemente las siguientes instrucciones solo para los puertos B y C.

Instrucciones:

- **PD** (de *Pin Direction*). Selecciona la dirección de un pin. El formato es:

  `PD,<puerto>,<pin>,<dirección>\n`. El comando retornará `"OK\n"` si los parámetros son correctos.

- **PI** (de *Pin Input*). Devuelve el estado de un pin. El formato es:

  `PI,<puerto>,<pin>\n`. El comando retornará `"PI,1\n"` si el pin está a 1, `"PI,0\n"` si está a 0.

- **PO** (de *Pin Output*). Cambia el estado de un pin. El formato es:

  `PO,<puerto>,<pin>,<valor>\n`. El comando retornará `"OK\n"` si los parámetros son correctos.

  En donde:

  - `<puerto>` es la letra B o C según el puerto al que pertenezca el pin que queramos.
  - `<pin>` es el número del pin que queramos, expresado en hexadecimal (0 a F).
  - `<dirección>` es 0 para salida o 1 para entrada.
  - `<valor>` es 1 o 0 según el valor que queramos poner en el pin.

Casos de Error:

- En los casos donde la Instrucción no se reconozca (no sea D, I o O) mande un `"Instrucción Desconocida\n"`.
- En los casos donde el puerto no se reconozca (no sea B o C) mande un `"Puerto no soportado\n"`.
- En cualquier otro caso de error mandará `"Error\n"` (e.g. no hay `','` o si se manda un numero incorrecto...).

Ejemplos:

- `PD,B,5,1` pondrá el pin `RB5` como entrada.
- `PD,C,0,0` pondrá el pin `RC0` como salida.
- `PA,C,0,0` devolverá mensaje de error de instrucción.
- `PD,A,0,0` devolverá mensaje de error de puerto.
- `PD,C,0` devolverá mensaje de error.
- `P,A,0,0` devolverá mensaje de error.
- `PO,C,0,1` pondrá `RC0` a 1, apaga led.
- `PO,C,0,1` pondrá `RC0` a 0, enciende led.
- `PD,C,0,1` pondrá el pin `RC0` como entrada y debería apagar el led si estaba encendido.
- `PI,B,5` lee el valor del pulsador en `RB5`, 1 o si pulsado 0.