Práctica 7. Comunicaciones serie II

(Duración: 1 sesión)

Introducción

En esta práctica nos familiarizaremos con las comunicaciones serie usando colas e interrupciones.

Objetivos

Al concluir la práctica, el alumno deberá ser capaz de:

- 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.

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

1. 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.

Por otro lado asegúrese de que los *jumpers* `JP1` y `JP3` están conectados, ya que en caso contrario no funcionarán ni el pulsador ni los LEDs.

2. Módulo para gestionar 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 con un sólo carácter o bien crear una función, denominada por ejemplo `putcUART` que introduzca un sólo carácter en la cola y active las interrupciones para que se envíe.

3. Bluetooth bit whacker

Una vez comprobado que el módulo de la UART funciona correctamente, se escribirá un programa para poder controlar los pines de la tarjeta desde el PC o el móvil. Para ello se usarán los siguientes comandos, adaptados de la tarjeta USB bit whacker:[1]

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

  `PD,<puerto>,<pin>,<dirección>\n`.

  En donde:

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

  El comando retornará `"OK\n"` si los parámetros son correctos o `"Error\n"` si el comando es erróneo (por ejemplo si se usa la letra D para el puerto).

  Por ejemplo `PD,B,5,1` pondrá el pin `RB5` como entrada.

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

  `PI,<puerto>,<pin>\n`.

  En donde:

  - `<puerto>` es la letra A, B o C según el puerto al que pertenezca el pin que queremos leer.

[1] Para más información consulte http://www.schmalzhaus.com/UBW/index.html.

- `<pin>` es el número del pin que queremos leer, expresado en hexadecimal (0 a F).

El comando retornará `"PI,1\n"` si el pin está a 1, `"PI,0\n"` si el pin está a 0 o `"Error\n"` si el comando es erróneo (por ejemplo si se usa la letra D para el puerto).

Por ejemplo `PI,B,5` devolverá el valor del pin `RB5`.

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

  `PO,<puerto>,<pin>,<valor>\n`.

  En donde:

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

  El comando retornará `"OK\n"` si los parámetros son correctos o `"Error\n"` si el comando es erróneo (por ejemplo si se usa la letra D para el puerto).

  Por ejemplo `PO,C,0,1` pondrá `RB0` a 1.

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 el `\n` se analizará la cadena para interpretar el comando y ejecutarlo, devolviendo con `putsUART` el resultado de dicho comando.