SISTEMAS BASADOS EN MICROPROCESADOR
PRÁCTICA 5. E/S DIGITAL

Grado en Ingeniería Electrónica Industrial (3er curso)
Curso 2023/2024

1 INTRODUCCIÓN

La programación de sistemas empotrados se hace mayoritariamente en C, dejando el ensamblador para programar las tareas más sensibles en tiempo, memoria y más cercanas al *hardware*. En esta práctica se va a desarrollar una librería de funciones (subrutinas) para usar los puertos de E/S digital usables desde C.

1.1 Objetivos

- Desarrollar una librería para manejar los puertos de E/S digital.

2 ESTUDIO TEÓRICO

Busque y estudie en la documentación de examen de la página web de la asignatura la documentación de los puertos de E/S digital (`EntradaSalida.pdf`). En ella se recogen los registros de control de los puertos del FR6989. Estudie también la documentación del *Launchpad* (sección de teoría de la web) y anote a continuación los puertos de E/S donde están ubicados los leds y pulsadores de usuario, así como la polaridad de los mismos (activos en alta o en baja):

- Led1:
- Led2:
- Pulsador1:
- Pulsador2:

2.1 Trabajando con referencias variables de puerto

Es fácil trabajar con puertos conocidos a priori. Por ejemplo, configurar el `P1.0` como entrada con resistencia de *pullup*:

```
bic.b   #BIT0, &P1DIR   ;P1.0 entrada
bis.b   #BIT0, &P1REN   ;P1.0 resistencia...
bis.b   #BIT0, &P1OUT   ;     ...de pullup
```

Sin embargo, si se quiere hacer lo mismo con el puerto cuyo número se guarda en R12 y el bit en R13, la cosa se complica. Para facilitarlo, estudie el contenido de los ficheros `msp430ports.asm` y `msp430ports.h` disponibles en la web de la práctica. En ellos se definen unas tablas de puertos y de bits que pueden utilizarse para obtener las direcciones de puerto y las constantes necesarias. El siguiente código hace lo mismo que el anterior, pero usando referencias variables del puerto y bit:

```
;Supongamos que R12=puerto y R13=bit
;Hacer R11 = Dirección base del puerto r12
mov.w   r12, r11        ;R11=puerto
rla.w   r11             ;R11=2*puerto (offset de tabla)
mov.w   TabPuertos(r11), r11 ;R11=Dir base del puerto
;Hacer R15 = Máscara para trabajar con el bit R13
mov.b   TabBits(r13), r15
;Ahora ya se puede configurar el puerto como entrada y Rpullup
bic.b   r15, PDIR(r11)  ;Configurar entrada
bis.b   r15, PREN(r11)  ;Configurar resistencia...
bis.b   r15, POUT(r11)  ;     ...de pullup
```

3 ESTUDIO PRÁCTICO

3.1 Preparando el proyecto

Copie el proyecto de la práctica anterior y llámelo `P5.1`. Para ello, selecciónelo en el Explorador de Proyectos y pulse `Control-C` (copiar) y `Control-V` (pegar). Se van a reaprovechar los módulos creados en ella, así como parte del programa principal en C.

3.2 Módulo de gestión de los puertos de E/S digital `pt.asm`

3.2.1 Creación del módulo

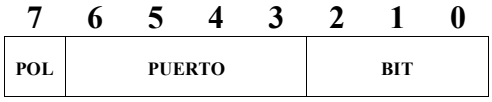
Cree los ficheros `pt.asm` y `pt.h` conforme a lo aprendido en prácticas anteriores.

3.2.2 Definición del tipo `puerto_t`

Definamos un nuevo tipo para gestionar las variables de tipo puerto. Escriba la siguiente definición en `pt.h` en algún punto antes de la línea "`#endif`":

```
typedef unsigned char puerto_t;
```

Se trata de un objeto de 8 bits que codifica información básica del puerto de entrada/salida, como el número de puerto, el bit y su polaridad. La codificación de este tipo es la siguiente:



PUERTO: Número entre 0 y 10
POL: 0 para señales activas en alta
1 para señales activas en baja
BIT: Número de bit del puerto

3.2.3 Configuración de un puerto `ptConfigura`

Este servicio crea un nuevo puerto y lo configura.

```
puerto_t ptConfigura (int puerto, int bit, int modo);
```

Los parámetros son:

- `puerto` es el número del puerto a configurar. Debe ser un número entre 0 y 10 donde el puerto 0 se refiere al puerto J, 1 al puerto 1, etc.
- `bit` es el número de bit del puerto. Se tomarán los 3 lsb de este parámetro ignorando el resto.
- `modo` es un campo de bits en el que se codifican distintas configuraciones del puerto (las opciones por defecto se muestran en negrita):

  - POL, bit 5. Polaridad del puerto: **`PT_ACTIVOALTA`** o `PT_ACTIVOBAJA`.
  - VINI, bit 4. Valor inicial del puerto: **`PT_OFF`** o `PT_ON`.
  - TIPO, bits 3 y 2. Tipo de puerto: **`PT_ENTRADA`**, `PT_ENTRADA_PULLUP`, `PT_ENTRADA_PULLDOWN` o `PT_SALIDA`.
  - FUNCIÓN, bits 1 y 0. Función del puerto: **`PT_ESDIG`**, `PT_FUNC1`, `PT_FUNC2`, `PT_FUNC3`.

La función comprueba las precondiciones, configura el puerto conforme a lo solicitado y devuelve el objeto de tipo `puerto_t` recién configurado o -1 en caso de error y cuando se ha configurado una función distinta de `PT_ESDIG`.

Tenga en cuenta en su código las siguientes consideraciones:

- Ignore el valor inicial de la salida (VINI) cuando el puerto no esté configurado como tal.
- Ignore también los bits 4 (VINI) y 5 (POL) si el puerto no está en su función de E/S digital.

Incluya el código en `pt.asm`. No olvide incluir el nombre de la subrutina en la directiva `.global`, y el prototipo de esta función en `pt.h`. Incluya también las siguientes definiciones en `pt.h`:

```
#define PT_ESDIG            (0)
#define PT_FUNC1            (1)
#define PT_FUNC2            (2)
#define PT_FUNC3            (3)
#define PT_ENTRADA          (0<<2)
#define PT_ENTRADA_PULLUP   (1<<2)
#define PT_ENTRADA_PULLDOWN (2<<2)
#define PT_SALIDA           (3<<2)
#define PT_OFF              (0<<4)
#define PT_ON               (1<<4)
#define PT_ACTIVOALTA       (0<<5)
#define PT_ACTIVOBAJA       (1<<5)
```

Ejemplo de uso de la función:

```
puerto_t led1, sw1;

/*Configurar el led1 situado en el P1.0 como puerto de salida, con polaridad
positiva y apagado*/
led1 = ptConfigura (1, 0, PT_SALIDA|PT_ACTIVOALTA|PT_OFF);
//Teniendo en cuenta las opciones por defecto, sería equivalente a escribir
led1 = ptConfigura (1, 0, PT_SALIDA);
/*Configurar el switch1 situado en el P1.1 como puerto de entrada con
resistencia de pullup y polaridad inversa (cuando se pulsa, se lee 0)*/
sw1 = ptConfigura (1, 1, PT_ENTRADA_PULLUP|PT_ACTIVOBAJA);
```

3.2.4 Lectura de un puerto `ptLee`

Este servicio lee un puerto.

```
int ptLee (puerto_t pt);
```

Los parámetros son:

- `pt` es una variable de tipo `puerto_t`. Se debe comprobar que el puerto ha sido inicializado correctamente (el número de puerto debe ser válido). En caso contrario, sale sin hacer nada.

Devuelve un 0 si el puerto está desactivado y un número distinto de 0 en caso contrario. La lectura se hace teniendo en cuenta la polaridad configurada. Así, si el puerto es activo en baja y tiene un 1 en su entrada, devolverá un 0.

No olvide poner el prototipo en `pt.h` e incluir `ptLee` en `.global`.

Ejemplo:

```
if (ptLee (sw1))
  //Poner aquí el código si se ha pulsado la tecla sw1
```

3.2.5 Escritura de un puerto `ptEscribe`

Este servicio escribe en un puerto.

```
void ptEscribe (puerto_t pt, int valor);
```

Los parámetros son:

- `pt` es una variable de tipo `puerto_t`. Se debe comprobar que el puerto ha sido inicializado correctamente (el número de puerto debe ser válido). En caso contrario, sale sin hacer nada.
- si `valor` es 0, se desactivará el puerto. En caso contrario se activará. Para ello se tiene en cuenta la polaridad configurada. Así, si el puerto es activo en baja y se escribe un 0, el puerto se pondrá a 1.

No olvide poner el prototipo en `pt.h` e incluir `ptEscribe` en `.global`.

Ejemplo:

```
ptEscribe (led1, 123);              //Enciende el led1
```

3.3 Programa principal en C

Para poder usar el módulo anterior desde C, debe incluir el fichero de cabecera correspondiente:

```
#include "pt.h"
```

Las dobles comillas le indican al compilador que debe buscar los ficheros en el directorio del proyecto. A diferencia de

```
#include <msp430.h>
```

que le indica que el fichero está en el directorio *include* del sistema.

Complete el código para que configure los puertos de los leds y pulsadores del *Launchpad* y gestione la siguiente lógica por sondeo:

- El led 1 se encenderá mientras se tenga el pulsador 1 y apagado en caso contrario.
- El led 2 cambiará su estado cada vez que se pulse el pulsador 2.