SISTEMAS BASADOS EN MICROPROCESADOR
PRÁCTICA 3. E/S DIGITAL

Grado en Ingeniería Electrónica Industrial (3er curso)
Curso 2026/2027

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
```

Si se quieren usar etiquetas para ubicar el número del puerto y del bit de cada dispositivo, es necesario definir algunas constantes que puede encontrar en el fichero `msp430ports.h` disponible en la web de la práctica. Deberá incluir dicho fichero en cualquier módulo que quiera utilizarlas. Note que para especificar el número del puerto siempre debe especificar el registro `PxIN`:

```
                .cdecls "msp430ports.h" ;Cargar constantes de puertos

LED1_BIT        .equ    BIT0            ;El led 1 está en el bit 0...
LED1_PORT       .equ    P1IN            ;   ...del puerto 1

                bic.b   #LED1_BIT, &LED1_PORT+PDIR;P1.0 entrada
                bis.b   #LED1_BIT, &LED1_PORT+PREN;P1.0 resistencia...
                bis.b   #LED1_BIT, &LED1_PORT+POUT;     ...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 `P3.1`. 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

Seleccione el proyecto en el *Project Explorer* y cree un nuevo fichero para albergar el módulo pulsando sobre *File/New/File from template...*. Llame al fichero `pt.asm` y seleccione "*Ensamblador*" en "*Template*" (modelo)[1]. El nuevo fichero se abre automáticamente en el editor. Estudie el contenido:

- La directiva `.cdecls` permite incluir un fichero de cabecera de C y lo traduce a ensamblador. En nuestro caso carga las definiciones del MSP430.
- La directiva `.global` recoge las etiquetas que van a traspasar la frontera del módulo. Funciona en ambos sentidos: importación y exportación. Es decir, etiquetas externas al módulo que se van a acceder desde aquí y etiquetas definidas en el módulo que se quieren usar fuera. En nuestro caso nombra las subrutinas que se van a definir en este módulo. Si en el futuro añade subrutinas a este módulo, tendrá que añadir aquí sus etiquetas si van a ser accedidas desde otro módulo.

Todo módulo debe tener un fichero de cabecera que recoja las declaraciones de tipos y los prototipos del mismo, para poder incluirlo en aquéllos que lo utilicen. Seleccione el proyecto en el *Project Explorer* y cree un nuevo fichero para albergar el módulo pulsando sobre *File/New/Header File*. Llame al fichero `pt.h` y seleccione "*Default C header template*" en "*Template*".

Recuerde que por cada servicio (subrutina, función) que añada a un módulo y que quiera que sea accesible desde el exterior, deberá añadir su nombre en la directiva `.global` y su prototipo en el fichero de cabecera.

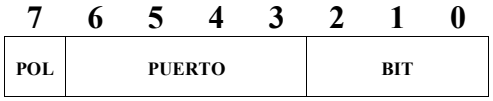
1. Si el modelo no aparece en la lista, añádalo a partir del fichero `TemplateEnsamblador.xml` (que puede encontrar en la web de la asignatura) pulsando en "*Configure...*" y después en "*Import...*". Seleccione el fichero descargado y pulse "*Abrir*".

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 uint8_t 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

El tipo `uint8_t` está definido en `stdint.h`.

3.2.3 Configuración de un puerto `ptConfigura`

Este servicio crea un nuevo puerto y lo configura.

```
puerto_t ptConfigura (int puerto, int bit, uint8_t 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):



  - IRQ, bit 7. Habilitación de interrupciones (sólo puertos 1 a 4): **`PT_NOIRQ`** o `PT_IRQ`.
  - SF, bit 6. Selector de flanco activo para interrupciones (sólo puertos 1 a 4): **`PT_SUBIDA`** o `PT_BAJADA`.
  - POL, bit 5. Polaridad del puerto: **`PT_ACTIVOALTA`** o `PT_ACTIVOBAJA`.
  - VINI, bit 4. Valor inicial del puerto: **`PT_OFF`** o `PT_ON`.
  - FUNCIÓN, bits 3 y 2. Función del puerto: **`PT_ESDIG`**, `PT_FUNC1`, `PT_FUNC2`, `PT_FUNC3`.
  - TIPO, bits 1 y 0. Tipo de puerto: **`PT_ENTRADA`**, `PT_ENTRADA_PULLUP`, `PT_ENTRADA_PULLDOWN` o `PT_SALIDA`.

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.

Tenga en cuenta 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.
- Ignore SF e IRQ si el puerto no está en el rango [1, 4].

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 MD_IRQ    BIT7
```

```
#define MD_SF     BIT6
#define MD_POL    BIT5
#define MD_INI    BIT4
#define MD_FUNC1  BIT3
#define MD_FUNC0  BIT2
#define MD_TIPO1  BIT1
#define MD_TIPO0  BIT0
#define MD_FUNC   BIT2|BIT3
#define MD_TIPO   BIT0|BIT1

#define PT_NOIRQ            (0<<7)
#define PT_IRQ              (1<<7)
#define PT_SUBIDA           (0<<6)
#define PT_BAJADA           (1<<6)
#define PT_ACTIVOALTA       (0<<5)
#define PT_ACTIVOBAJA       (1<<5)
#define PT_OFF              (0<<4)
#define PT_ON               (1<<4)
#define PT_ESDIG            (0<<2)
#define PT_FUNC1            (1<<2)
#define PT_FUNC2            (2<<2)
#define PT_FUNC3            (3<<2)
#define PT_ENTRADA          (0<<0)
#define PT_ENTRADA_PULLUP   (1<<0)
#define PT_ENTRADA_PULLDOWN (2<<0)
#define PT_SALIDA           (3<<0)
```

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`

```
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) y el puerto está en modo E/S digital. 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.

Ejemplo:

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

3.2.5 Escritura en un puerto `ptEscribe`

```
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) y el puerto está en modo E/S digital. 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.

Ejemplo:

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

3.2.6 Lectura del flanco activo de interrupción `ptLeeFlanco`

```
int ptLeeFlanco (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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale devolviendo 0.

Devuelve un 0 en caso de error o si está configurado flanco de subida y un número distinto de 0 si el flanco es de bajada.

3.2.7 Establecer el flanco activo de interrupción `ptEscFlanco`

```
void ptEscFlanco (puerto_t pt, int flancobajada);
```

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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale sin hacer nada.
- Si `flanco` es 0, configura flanco de subida. En caso contrario, flanco de bajada.

3.2.8 Lectura del estado de la habilitación de interrupción `ptLeeHabIRQ`

```
int ptLeeHabIRQ (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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale devolviendo 0.

Devuelve un 0 en caso de error o si la interrupción no está habilitada y un número distinto de 0 en caso contrario.

3.2.9 Establecer el estado de la habilitación de interrupción `ptHabIRQ`

```
void ptEscHabIRQ (puerto_t pt, int siono);
```

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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale sin hacer nada.
- `siono` es el estado deseado de la habilitación de interrupciones. Si es 0, deshabilita las interrupciones del puerto y las habilita en caso contrario.

3.2.10 Lectura de la bandera de interrupción `ptLeeIRQ`

```
int ptLeeIRQ (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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale devolviendo 0.

Devuelve un 0 en caso de error o si no se ha producido un flanco activo en el puerto y un número distinto de 0 en caso contrario.

3.2.11 Borrar la bandera de interrupción `ptBorraIRQ`

```
void ptBorraIRQ (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), el puerto está en modo E/S digital y que está en el rango [1, 4]. En caso contrario, sale sin hacer nada.

3.2.12 Declarando una función de tratamiento de interrupción en C

Es posible definir una subrutina de servicio de interrupción en C incluyendo el modificador "`__interrupt`" antes del proptotipo[2]. La subrutina no debe aceptar parámetros ni devolver nada. El compilador de C genera código para que se vuelva con `RETI` en lugar de con `RET` e intenta no usar ningún registro, o apilar los que use. Si dentro de la función se llama a otra, se apilan todos los registros para evitar alteraciones.

Antes de la definición de la ISR hay que usar una directiva `#pragma` para indicar el vector de interrupción en el que se colocará:

```
#pragma vector=PORT1_VECTOR
__interrupt void P1ISR (void)
{
  //Código de la ISR aquí
}
```

No olvide habilitar las interrupciones del procesador en el programa principal después de configurarlo todo. Puede usar la macro:

```
_EINT ();
```

3.3 Programa principal en C

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

- Al entrar se mostrará el mensaje "Hola" centrado en pantalla.
- El led 1 se encenderá mientras se presione el pulsador 1 y apagado en caso contrario. Haga una programación por sondeo.
- El led 2 se encenderá mientras se presione el pulsador 2 y apagado en caso contrario. Haga una programación por interrupciones.
- Salga del programa cuando se pulsen las dos teclas a la vez, mostrando el mensaje "Adios".

2. Hay dos guiones bajos antes de la palabra reservada.