SISTEMAS BASADOS EN MICROPROCESADOR
PRÁCTICA 6. TECLADO MATRICIAL

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

1 INTRODUCCIÓN

El BOOST-IR es un módulo de la serie *BoosterPack* de Texas Instruments para complementar su serie de *Launchpads*. Consta de, entre otros elementos, un teclado matricial de membrana de 16 teclas. Un teclado de este tipo está construido con dos láminas plásticas delgadas con pistas conductoras grabadas en la cara interior, y una tercera membrana de separación. La membrana interna se agujerea en las zonas donde se quieren poner las teclas y se produce una burbuja de aire para que no haya contacto entre las pistas conductoras y para producir una pequeña resistencia a la pulsación. Cuando se hace presión sobre las burbujas, las pistas conductoras se tocan, produciendo un contacto eléctrico. Estos teclados también suelen llamarse teclados de burbujas.

Estos módulos se conectan al *Launchpad* mediante un par de conectores. Puede encontrar toda la información sobre las señales presentes en los conectores, así como los elementos de los *boosterpacks* en los correspondientes manuales disponibles en la web de la asignatura.

1.1 Objetivos

- Gestionar un teclado matricial.
- Establecer un mecanismo por interrupciones para eliminar los rebotes.

2 ESTUDIO TEÓRICO

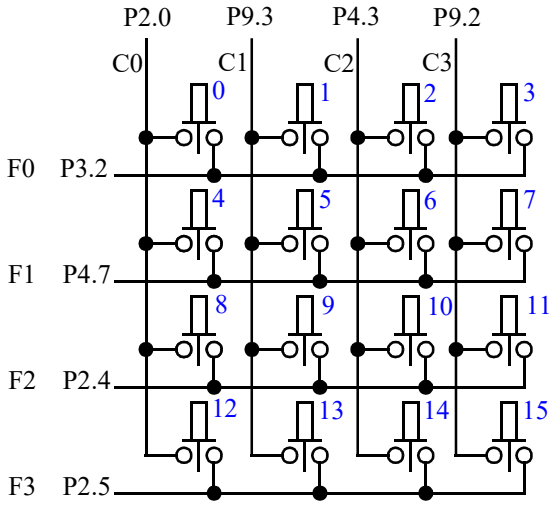
Busque y estudie en la documentación de la web de la asignatura los manuales del *Launchpad* y *Boosterpack*. Preste especial atención a las señales presentes en los conectores de expansión del *Launchpad* y a qué líneas se conecta el teclado.

2.1 Modelo hardware del teclado

El teclado consta de 16 interruptores distribuidos en forma de matriz de 4x4. Desafortunadamente, los diseñadores del *Launchpad*/*Boosterpack* no colocaron las distintas señales de control en posiciones consecutivas de un puerto, lo que va a suponer un esfuerzo extra de programación.

Cuando se pulsa una tecla se produce una conexión entre la fila y la columna correspondiente. La ventaja de este tipo de montaje es el reducido número de puertos para

controlar un elevado número de teclas[1]. El problema es que la lectura requiere un proceso más complejo que llamaremos *barrido* o *scan* de teclado. En la figura se muestra la disposición de las teclas en filas y columnas y los puertos en los que se conectan al MPS430. Cada tecla tiene asociado un número T que se puede calcular como T = F * 4 + C, siendo F y C las fila y columna respectivamente que se han cortocircuitado.



2.2 Proceso de lectura de una tecla

El proceso de lectura requiere un punto de partida que se conseguirá en la fase de inicialización:

- Configurar las filas como entradas[2] con resistencia de *pullup*.
- Activar las columnas (configurarlas en modo salida con valor a 0).

El proceso de barrido en sí hará lo siguiente:

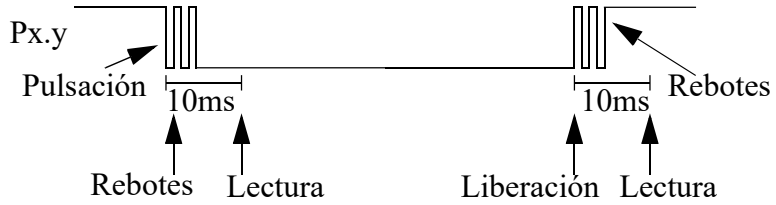
- Desactivar todas las columnas (ponerlas en modo entrada).
- Activar la columna 0 poniéndola como salida (con valor 0).
- Si ninguna de las teclas de la columna está pulsada, en los puertos de las filas se leerán 1's gracias a los *pullups*. En cambio, si alguna está pulsada, la fila correspondiente estará a 0 gracias al contacto de la tecla.
- Para leer el resto de las columnas se desactivará la columna actual y se activará la siguiente, repitiendo el proceso.
- Una vez leídas las 4 columnas, se activarán todas antes de salir para permitir un proceso por interrupciones.

2.3 Los rebotes

La mayoría de los teclados mecánicos presentan un efecto no deseado: los rebotes. Cuando se pulsa, la tecla realiza una conexión eléctrica entre dos láminas metálicas. Es posible que haya rebotes en dichas láminas tanto en la fase de cierre como en la fase de apertura, produciendo la sensación de varias pulsaciones. Para evitarlo hay procedimientos *hardware*

1. Se requieren tantas líneas como 2 veces la raíz cuadrada del número de teclas. Para 16 teclas, 8 líneas (4 filas y 4 columnas). Para un teclado tipo PC de más de 100 teclas, se necesitan sólo 22 líneas.
2. En general los papeles de filas y columnas son perfectamente intercambiables. En nuestro Launchpad no es así dado que los puertos P1 a P4 disponen de interrupciones, pero no los P5 a P10. Las filas se encuentran dentro del rango de puertos con interrupciones, no así las columnas.

y *software*. En nuestro caso, vamos a usar una procedimiento *software* consistente en esperar un tiempo después de detectar la pulsación de la tecla para realizar el barrido de teclado. Dicho tiempo es específico para cada tecla. Para nuestro teclado el valor adecuado para dicho tiempo es de 10 ms.



Es importante notar que, debido a estos rebotes, es posible que la liberación de una tecla se interprete como una pulsación si se ha configurado la detección de flancos de bajada. De la misma forma, una pulsación podría interpretarse como una liberación en la que se están detectando los flancos de subida.

El procedimiento de lectura será detectar el flanco de bajada de la línea de lectura. En la subrutina de servicio de la interrupción se programará un temporizador con el tiempo del rebote y se deshabilitarán ulteriores interrupciones de esta línea. Cuando se produzca la interrupción del temporizador, se hará un barrido de teclado y se cambiará el flanco activo, para detectar la liberación de la tecla. Si el rebote ha sido en la fase de liberación de la tecla debería detectarse que no hay ninguna tecla pulsada. Esta última ISR también deshabilitará la IRQ del temporizador y habilitará de nuevo la del flanco de bajada de la línea para detectar futuras pulsaciones.

3 ESTUDIO PRÁCTICO

3.1 Preparando el proyecto

Copie el proyecto de la práctica anterior y llámelo `P6.1`. Se va a reaprovechar la mayor parte del código, incluido el código C.

Borre la función `AnimaLcd` y todas las constantes relacionadas, ya que no se necesitarán.

3.2 Módulo de gestión del teclado `teclado.asm`

Añada el nuevo módulo `teclado.asm`. y su fichero de cabecera `teclado.h`. En él se van a incluir funciones para manejar el teclado del *boosterpack*. Las variables y servicios de este módulo son:

3.2.1 Buffer de teclado `Tecla`

Las teclas van a ser leídas por interrupciones y se guardan en un *buffer* de teclado a la espera de que sean leídas por el programa principal. En nuestro caso, el *buffer* sólo se guardará una tecla:

```
.bss    Tecla, 1
```

Esta variable sólo será accesible internamente, por lo que no se publicará con la directiva `.global`.

3.2.2 Inicialización del teclado `kbIni`

Inicializa los puertos del teclado, poniendo las señales de las columnas como salidas y con valor 0 y las de las líneas como entradas con resistencia de *pullup* e interrupciones activas en el flanco de bajada[3]. También inicializa el *buffer* de teclado a vacío (`Tecla=NOTECLA=-1`):

```
void kbIni (void);
```

3.2.3 Subrutina de barrido `kbScan`

Realiza un barrido de teclado conforme a lo indicado en *2.2 Proceso de lectura de una tecla*:

```
char kbScan (void);
```

Devuelve el código ASCII de la tecla pulsada. Internamente, la subrutina calcula el código de la tecla pulsada (un número entre 0 y 15) a partir de los números de fila y de columna. Con dicho código se direccionará una tabla de conversión a ASCII (`TabTeclas`). Si no hay ninguna tecla pulsada se devuelve el carácter ETX (código ASCII 3) o si se han pulsado varias a las vez, devuelve `NOTECLA`. Antes de salir deja todas las columnas activas.

Los códigos ASCII del teclado serán[4]:

‘1’	‘2’	‘3’	‘C’
‘4’	‘5’	‘6’	‘D’
‘7’	‘8’	‘9’	‘E’
‘A’	‘0’	‘B’	‘F’

Se recomienda usar lo aprendido en la práctica de E/S digital para hacer el proceso de barrido más sencillo y salvar el desorden de las filas y columnas.

3.2.4 ISR de tecla pulsada `kbISR`

Subrutina de atención a la interrupción de fila. Conecte los vectores de todas las filas a la misma subrutina, de forma que se ejecute cuando se pulse cualquier tecla. Sus tareas son:

- Borrar los posibles flags.
- Deshabilitar las interrupciones de las filas.
- Programar el `Timer2_A1` para que produzca una interrupción después de 10ms[5].

3.2.5 ISR de tiempo de rebote `kbRebote`

Subrutina de atención a la interrupción secundaria del `TimerA2` (`CCR1`). Sus tareas son:

- Borrar el flag de esta interrupción y deshabilitarla.
- Hacer un barrido de teclado (`kbScan`) y guardar el código ASCII de la tecla pulsada o ETX si se ha liberado en el *buffer* de teclado (`Tecla`). Tenga en cuenta que no se puede alterar ningún registro en una ISR, por lo que deberá guardar todos aquellos que se modifiquen incluidos los que pueda modificar `kbScan`.

3. En una primera fase no se activarán las interrupciones, por lo que deberá comentar las líneas correspondientes.
4. Es fácil cambiar la tabla de asignación de códigos ASCII para adaptar el módulo a cada aplicación.
5. No olvide borrar el flag antes de activar la interrupción.

- Cambiar el flanco activo de las interrupciones de teclado.
- Habilitar las interrupciones de los puertos de las líneas[5].

3.2.6 Subrutina de lectura de tecla `kbGetc`

Devuelve la tecla almacenada en el *buffer* de teclado (variable interna `Tecla`) y vacía dicho *buffer* escribiendo `NOTECLA` en el mismo:

```
char kbGetc (void);
```

3.3 Programa principal en C

Además, de la animación de los leds de usuario del *Launchpad* y de los segmentos de batería como en la práctica anterior, se van a pintar las teclas que pulse el usuario en la pantalla. Lo haremos en varias fases, para asegurar que todo funciona. Cree una nueva tarea cooperativa llamada `InterfazUsuario` que pinte en pantalla las teclas leídas. Al ser una tarea interactiva, irá a la velocidad del `SystemTimer`, por lo que no es necesario usar una variable `ProximaEjecucion` ni comprobar si tiene que ejecutarse o no. Simplemente lee la tecla y actúa en consecuencia:

3.3.1 Fase 1. Leyendo teclas por sondeo

- Comente las líneas de `kbIni` que activan las interrupciones en las líneas del teclado.
- `InterfazUsuario` llamará a `kbScan` para leer la tecla y se imprimirá en pantalla con `lcdPintaIzq`. Pinte un guión ('-') cuando se detecte una liberación de tecla.

Si todo va bien, debería llenar la pantalla con el código ASCII de la tecla que se pulse en cada momento. Si no, refine `kbScan` y `kbIni`. Cuando todo funcione, vaya a la fase siguiente.

3.3.2 Fase 2. Leyendo teclas por interrupciones

Es el momento de activar las interrupciones:

- Descomente las líneas de `kbIni` que activan las interrupciones en las líneas del teclado.
- Cambie la llamada a `kbScan` por `kbGetc`.
- Cuando se pulse una tecla, se entrará en `kbISR`. Ésta debe hacer lo descrito en *3.2.4 ISR de tecla pulsada `kbISR`*.
- Una vez que pase el tiempo de rebote, se entrará en `kbRebote`. Ésta debe hacer lo descrito en *3.2.5 ISR de tiempo de rebote `kbRebote`*.

Si todo va bien, cada vez que se pulse una tecla, debería aparecer en la pantalla (sólo una vez por pulsación). Cada vez que se suelte, deberá pintarse un guión. Si no, refine `kbISR` y `kbRebote`.

3.3.3 Fase 3. Reduciendo el consumo

La principal ventaja del sistema basado en interrupciones es que libera al procesador de tener que comprobar si hay teclas pendientes. Sin esa tarea, es posible que la CPU no tenga nada que hacer si no se ha pulsado una tecla[6]. Así que lo más adecuado sería entrar en algún modo

6. Posible, pero no obligatorio. Si está usando multitarea cooperativa es posible que otros procesos sí tengan cosas que hacer. En la práctica 5 ya se programó la IRQ periódica para que despertara el micro con cada tic de reloj.

de bajo consumo a la espera de alguna actividad en el teclado. Dado que se requiere que el `ACLK` esté activo para alimentar el `TimerA2`, no podrá ir más allá del `LPM3`.

Modifique `kbRebote` para que el micro se despierte tras la lectura de la tecla (no es necesario hacerlo tras `kbISR`) y pueda volver al programa principal para que la gestione.

Note que con esta programación el superbucle entrará en LPM3 después de hacer una pasada de todas las tareas y saldrá del mismo después de una pulsación de tecla y después de un tic de reloj, permitiendo la lectura y procesado de la tecla y la ejecución de las tareas cooperativas según convenga.