MICROPROCESADORES

PRÁCTICA 2

INTERRUPCIONES, TIMERS Y I/O ANALÓGICA

ADC Y PWM

1.1. Ejercicio 1 - control de la multiplexación

En este apartado se pide que desarrolle un programa para su microcontrolador que muestre en el *display* de 7 segmentos el mensaje «On». La frecuencia de multiplexacion sera de 250 Hz.

Trabaje sobre la carpeta MICR\P2\S1\E1 suministrada, que incluye un proyecto de *µVision 5* vací´o, pero configurado para trabajar con su microcontrolador.

Como ya se ha dicho, **de ahora en adelante no se permite el uso de la librería *sw_tick_serial***, de modo que debera emplear los recursos que ya conoce de la librerí´a *mbed* para la generacion del evento periodico que controla la multiplexacion. No dude, eso sí´, en reutilizar el codigo que considere adecuado de la practica anterior., particularmente la funcion `to_7seg()`, ya sea en su version en C/C++ o en lenguaje de ensamble.

Evite el uso de numeros en punto flotante. Su uso dara lugar a la inclusion, durante el *linkado*, de las correspondientes librerí´as para manejo de este tipo de datos, lo que provocara un mayor tamano de los ejecutables resultantes, pudiendo ocasionar problemas si trabaja con la version gratuita de la herramienta *Keil µVision 5*, como sabe limitada a un tamano maximo de ejecutable de 32 KiB. **De ahora en adelante tampoco se permite el empleo de números en punto flotante** (es perfectamente posible realizar todas las practicas sin usarlos). Tenga en cuenta que varios metodos de la librerí´a *mbed* hacen uso de `float`, por

---

*En particular: en la oscuridad debe encenderse el LED izquierdo; ante fuerte iluminacion lo hara el medio; para una iluminacion intermedia —habitacion iluminada con luz artificial que permita trabajar comodamente— lo haran ambos; y para el resto de casos se apagaran los dos.

lo que no podra emplearlos. Estos metodos son (para las clases `Timer`, `Ticker`, `Timeout`, `PwmOut`, `AnalogIn` y `AnalogOut`): `read()`, `write()`, `attach()`, `period()`, `pulsewidth()`, `operator=()` y `operator float()`.

Tampoco se permite el uso de las funciones `wait()`, `wait_ms()` y `wait_us()`, como se indico en las clases de teoría.

Tenga en cuenta que, cuando no existan eventos pendientes de procesado, es deseable que el procesador duerma para así minimizar el consumo de energía. Considere, por tanto, el uso de la función `__WFI()` de la librería *mbed* (en realidad de la librería *cmsis*, incluida por *mbed*) en tales circunstancias. **Se espera que, a partir de este momento, siempre se duerma al procesador cuando no haya eventos pendientes.**

Finalmente, **se recuerda que el código en C/C++ que escriba debe respetar la guía de estilo**:

[BarrC18] Barr, Michael. *Embedded C Coding Standard*. CreateSpace Independent Publishing Platform, 2018. Disponible *online* en https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard.

1.2. Ejercicio 2 - cuenta

Modifique el programa del apartado anterior para que el *display* muestre una cuenta ascendente, desde el 0 hasta el 99 (y vuelta a empezar), incrementando la cuenta a una frecuencia de 3 Hz.

Copie el proyecto del apartado anterior sobre la carpeta `MICR\P2\S1\E2` y trabaje sobre esta copia.

1.3. Ejercicio 3 - lectura de la LDR

Modifique el programa del apartado anterior para que el *display* muestre un valor, desde 0 hasta 99, que indique el porcentaje que representa la tension analogica presente en la senal LIT respecto de la tension de alimentacion del microcontrolador. Cuando la tension en LIT sea baja (proxima a masa) se mostraran valores bajos, proximos a 0. Si la tension en LIT es proxima a la tension de alimentacion se mostraran valores altos, proximos a 99. Entre esos extremos la variacion sera

lineal. El valor representado se refrescara a una frecuencia de 3 Hz. Con el fin de evitar el uso de numeros en punto flotante el metodo `read_u16()` de la clase `AnalogIn` puede serle de utilidad.

Copie el proyecto del apartado anterior sobre la carpeta `MICR\P2\S1\E3` y trabaje sobre esta copia.

1.4. Ejercicio 4 - LED

Modifique el programa anterior para que, ademas, el LED derecho luzca intermitentemente a una frecuencia de 100 Hz. La duracion del encendido sera proporcional al numero presentado en el *display*: si ese numero es 0, el LED permanecera encendido solamente durante 100 µs; si es 99, estara encendido constantemente. Para valores intermedios la duracion del encendido variara linealmente entre ambos extremos. Recuerde que no deben emplearse numeros en punto flotante. Puesto que la frecuencia de intermitencia del LED es demasiado elevada como para que el ojo pueda apreciarla, lo que se percibira sera que el brillo del LED varia de forma mas o menos proporcional al dato mostrado en el *display*.

Tenga en cuenta que el pin del microcontrolador al que esta conectado el LED derecho no tiene capacidad PWM. Por ello, para resolver este apartado, debera utilizar las clases `Ticker` y `Timeout` de *mbed*, de forma combinada para producir el resultado esperado. Tome las adecuadas precauciones para no llamar a las funciones de la librerí´a *mbed* con parametros inapropiados (por ejemplo, registrar un `Timeout` para un tiempo negativo o cero y situaciones similares). Copie el proyecto del apartado anterior sobre la carpeta `MICR\P2\S1\E4` y trabaje sobre esta copia.

1.5. Ejercicio 5 - control del brillo

Modifique el programa anterior para que, de la misma forma que el brillo del LED derecho depende del valor mostrado en el *display*, tambien lo haga el brillo del propio *display* de 7 segmentos.

En este caso los pines DSL y DSR sí disponen de capacidad PWM, por lo que podra emplear la clase `PwmOut` de *mbed* para gestionar dichos pines de forma mas comoda a como lo hizo con `Ticker` y `Timeout` en el

apartado anterior. La frecuencia del PWM de estas senales sera de 25 kHz. Tome las adecuadas precauciones para no llamar a las funciones de la librería *mbed* con parametros inapropiados (por ejemplo, fijar una anchura de pulsos para un `PwmOut` inferior a 1 µs y situaciones similares). Copie el proyecto del apartado anterior sobre la carpeta MICR\P2\S1\E5 y trabaje sobre esta copia.

Terminan aquí las tareas a realizar antes de la primera sesion presencial de esta practica.

2. TRABAJOS PREVIOS A LA SEGUNDA SESIÓN PRESENCIAL

2.1. Ejercicio 6 - prueba de controles de los pulsadores

Abra el proyecto de *Keil µVision 5* que se adjunta (carpeta MICR\P2\S2\E6) y analice el codigo para comprender su funcionamiento. La aplicacion pretende mostrar una cuenta ascendente del 0 al 99, incrementandose la cuenta con cada pulsacion del pulsador derecho.

Para ello se emplea un objeto `InterruptIn` de la librerí´a *mbed* que, cada vez que se detecte un flanco de bajada en la senal SWR (lo que debiera ocurrir cada vez que se presione el pulsador), pondra un *flag* de evento a `true`. El bucle principal incrementa el contador cada vez que dicho *flag* se activa.

Compile el programa, vuelquelo sobre la placa y verifique el funcionamiento. Si el funcionamiento no es correcto anote en el siguiente espacio las deficiencias que observe.

2.1.1. . REBOTES EN LOS PULSADORES

Las deficiencias que con seguridad ha encontrado en el funcionamiento del anterior programa se deben a un fenomeno conocido como *rebote de los contactos* (*contact bouncing* en ingles).

En un circuito de interfaz para un pulsador, como el de la figura 3, se espera que, cada vez que se presione el pulsador, la senal SWR se ponga a nivel bajo, o lo que es lo mismo, que cada vez que se active el pulsador se genere un flanco de bajada en SWR. En esto se basaba el funcionamiento del programa anterior.

5.1.2.1. *Medida de tiempos desde el anterior flanco*

Esta segunda estrategia se basa en dar como «valido» un determinado flanco en SWR (ya sea de subida o bajada) solo si el flanco anterior a éste ocurrio hace un tiempo mayor que un determinado valor (mayor que el tiempo de rebotes —10 ms en este ejemplo—). Así´, si se producen varios flancos muy seguidos (debidos a rebotes), solo se tendra en cuenta el primero de ellos. Como en la anterior estrategia, un objeto (`bool b_swr_state`) indicara si, tras la gestion de los rebotes, el pulsador esta pulsado (`true`) o abierto (`false`). Igualmente, otros dos objetos (`bool volatile gb_swr_fall_evnt` y `bool volatile gb_swr_rise_evnt`) se activaran cuando el programa detecta que el pulsador esta siendo actuado de alguna manera. Para medir el tiempo desde el anterior flanco se empleara un objeto `Timer`. Los flancos se gestionan mediante un objeto de la clase `InterruptIn`, cuyos metodos `fall()` y `rise()` —llamados desde `main()`— registran sendas ISR para la gestion de los flancos de bajada o subida. El codigo de ejemplo es (de nuevo un extracto, se ha eliminado todo lo relativo al sueno y a la multiplexacion del *display*):

```
// switch
static InterruptIn    g_swr(SWR_PIN);

// switch management
static Timer          g_swr_tmr;
static bool volatile gb_swr_fall_evnt;
static bool volatile  gb_swr_rise_evnt;

static void swr_fall_isr (void) {
  gb_swr_fall_evnt = true;
}

static void swr_rise_isr (void) {
  gb_swr_rise_evnt = true;
}

int main (void) {
  uint8_t cnt = 0;                          // 0 to 99
  bool    b_swr_state = false;
```

```
  g_swr.mode(PullUp);
  g_swr.fall(swr_fall_isr);
  g_swr.rise(swr_rise_isr);
  g_swr_tmr.start();

  for (;;) {
    if (gb_swr_fall_evnt) {
      gb_swr_fall_evnt = false;
      if ((!b_swr_state) && (g_swr_tmr.read_us() > 10000)) {
        b_swr_state = true;
        cnt += ((cnt >= 99) ? -cnt : 1);
      }
      g_swr_tmr.reset();
    }
    if (gb_swr_rise_evnt) {
      gb_swr_rise_evnt = false;
      if (b_swr_state && (g_swr_tmr.read_us() > 10000)) {
        b_swr_state = false;
      }
      g_swr_tmr.reset();
    }
  }
}
```

Encontrara este codigo dentro de la carpeta MICR\P2\S2\E6_Debounce_timer. Analí´celo, compí´lelo y pruebelo sobre la placa. Si el funcionamiento no es correcto anote en el siguiente espacio las deficiencias que observe.

Debe considerarse que esta estrategia presenta, en su implementacion con la librerí´a *mbed* y sobre procesadores basados en arquitecturas ARM *Cortex-M* un par de deficiencias:

- Los objetos `Timer` de la librerí´a *mbed* solo permiten la medida de tiempos hasta, aproximadamente, 35 minutos. Aunque poco probable, podrí´a ser posible que, tras 35 minutos sin actuar sobre el pulsador, una pulsacion sobre el pasase inadvertida.

- Si los rebotes generan flancos muy rapidamente, como se ve en la figura 8 (4 flancos en ~8 µs), la librerí´a *mbed* puede no procesar todas la interrupciones generadas. En particular, es posible que algunas interrupciones no invoquen a su ISR o que *las ISR no sean llamadas en el mismo orden en el que se generaron las interrupciones* (podrí´a llamarse dos veces consecutivas a la ISR de `fall()`, sin una llamada intermedia a la ISR de `rise()`, llamada que podrí´a producirse con posterioridad a las dos llamadas a la otra ISR, por ejemplo). Con otras librerí´as (*cmsis*) es posible una gestion mas precisa de este caso. Aun mas, la arquitectura de los procesadores ARM *Cortex-M* no es la mas adecuada para estos casos, siendo las arquitecturas ARM *Cortex-R* mucho mas apropiadas para una gestion precisa y estricta de las interrupciones. Si ha observado deficiencias en el funcionamiento de esta estrategia seguramente se deban a estos efectos. Por todo ello esta estrategia puede ser inaceptable.

FIGURA 8: rebotes rapidos al pulsar.

21.2. PROPUESTAS PARA LA GESTION DE LOS REBOTES

A la vista de todo lo anterior, proponga aquí un par de estrategias (distintas a las anteriores) para la gestion de los rebotes del pulsador. El objetivo es que, cada vez que se pulse, se active un *flag*, eliminando el efecto de los rebotes. No es necesario que escriba el codigo, solamente describa las ideas en las que se basara. Se recomienda que explore el uso de objetos `Timeout`. **Aunque en las clases de teoría le hayan sido ya presentados los autómatas controlados por eventos, se aconseja que aborde este apartado sin recurrir a ellos**.

2.2. Ejercicio 7 - gestión de un pulsador

Implementando alguna de las estrategias propuestas por usted para la gestion de los rebotes en el pulsador, escriba un programa que muestre en el *display* de 7 segmentos un numero del 0 al 99, incrementandose la cuenta con cada pulsacion del pulsador derecho.

No debe percibirse el efecto de los rebotes en los pulsadores. Asegurese, por tanto, de que:

- Con cada pulsacion la cuenta se incremente en una sola unidad.
- La cuenta se incrementa al pulsar y no al soltar.
- Si se mantiene pulsado un tiempo largo, la cuenta solo se incremente una vez.
- Si se pulsa muy rapidamente, la cuenta se incrementa tantas veces como pulsaciones, sin perder ninguna.

Recuerde que, para el correcto funcionamiento de los pulsadores, y al no haber montado resistencias externas de *pull-up*, deberan activarse los *pull-ups* internos del microcontrolador para cada pulsador.

Trabaje sobre la carpeta MICR\P2\S2\E7 copiando en ella el ejercicio de la carpeta MICR\P2\S1\E2.

2.3. Ejercicio 8 - gestión de varios pulsadores

Modifique el programa anterior para que:

- La cuenta se incremente con cada pulsacion del pulsador derecho.
- La cuenta se decremente con cada pulsacion del pulsador izquierdo.
- Si la cuenta vale n y se pulsa el pulsador central, pasara a valer $99 - n$.

Trabaje sobre la carpeta MICR\P2\S2\E8 copiando en ella el programa del ejercicio anterior. **Tome precauciones para que la gestión de los rebotes de un pulsador no interfiera con la de los demás.** En este sentido debe verificar el funcionamiento de su sistema en situaciones tales como que un pulsador permanezca un largo tiempo pulsado y, simultaneamente, se actue sobre los demas y similares.

2.4. Ejercicio 9 - pulsadores y LDR

Modifique el programa del apartado anterior para que, ademas, el brillo del *display* de 7 segmentos sea proporcional a la tension V_{LIT} entregada por la LDR. El brillo del *display* se actualizara tres veces por segundo.

Trabaje sobre la carpeta MICR\P2\S2\E9 copiando en ella el proyecto del ejercicio anterior.

Terminan aquí las tareas a realizar para esta practica.