MICROPROCESADORES

PRÁCTICA 2

INTERRUPCIONES, TIMERS Y I/O ANALÓGICA

ADC Y PWM

TITULACIONES DE GRADO DE LA
ETSIS DE TELECOMUNICACIÓN

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

PRIMAVERA 2022 – 2023

ÍNDICE

1. Planificación

En este enunciado se describen los trabajos a realizar durante esta segunda práctica. La práctica dispone de dos sesiones presenciales de laboratorio. Debe tener en cuenta que las sesiones presenciales de laboratorio son, más bien, sesiones de tutoría colectiva en las que el docente resolverá las dudas que durante la realización (previa a la sesión) de la práctica le hayan podido surgir. El grueso del trabajo práctico del laboratorio se espera que sea anterior a las sesiones presenciales de laboratorio y que usted organice ese trabajo de la forma que le sea más conveniente. Debe presupuestar un trabajo previo a cada sesión de laboratorio de al menos cuatro horas.

2. Objetivos

El objetivo fundamental de esta práctica es emplear el mecanismo de interrupciones (mediante líneas de solicitud de interrupción, IRQ, y *timers*) para la generación de eventos. Adicionalmente, se añadirá al circuito del laboratorio una fotorresistencia que permitirá explorar el manejo de un par de interfaces analógicas del microcontrolador (ADC y PWM).

Una vez conocidos los mecanismos básicos de gestión de eventos independientes (en la práctica anterior) y la generación de los mismos mediante interrupciones (en esta) estará preparado para (en la práctica siguiente) aplicar la gestión de *eventos interdependientes* mediante el uso de *máquinas de estados finitos (autómatas) controladas por eventos*, una potente técnica empleada asiduamente en el campo de los sistemas empotrados.

3. Estructura de carpetas

Adjunto al enunciado encontrará en *Moodle* un fichero comprimido que, al descomprimirlo (dentro de la carpeta MICR), genera una estructura de carpetas —conforme a lo dicho en la primera práctica— en la que debe ir trabajando y guardando todos los resultados de la práctica.

En diferentes partes del enunciado de la práctica se hace mención a carpetas y ficheros incluidos en esta estructura de directorios.

4. Trabajos previos a la primera sesión presencial

4.1. Introducción

En la primera parte de la práctica se explorarán los *timers* como recurso para la generación de interrupciones. Recordará de la anterior práctica que, en ella, se empleaba una librería (*sw_tick_serial*) que generaba una serie de eventos periódicos, que empleó para la gestión de la multiplexación del *display* y para realizar cuentas. **A partir de este momento no se permitirá más el uso de la librería *sw_tick_serial*** durante las prácticas y toda la generación de eventos deberá realizarse mediante los recursos que provee la librería *mbed*, recursos de más bajo nivel que los de *sw_tick_serial*, pero más potentes y flexibles —en posteriores asignaturas de la titulación de Electrónica de Comunicaciones se explorarán mecanismos más básicos aún para este tipo de tareas, en particular empleando la librería *cmsis* (suministrada por Arm) y la HAL (*Hardware Abstraction Layer*, suministrada por el fabricante del chip), que ya requieren un estudio detallado de los periféricos concretos del microcontrolador y de sus interfaces para poder usarse, pero permiten total control sobre los periféricos—.

En esta sesión también se hará uso de otros objetos de la librería *mbed* que permiten el uso de los convertidores AD y de los generadores PWM de que disponga el microcontrolador.

4.2. Ejercicio 0 - Montaje del circuito

4.2.1. MATERIALES NECESARIOS PARA LA PRÁCTICA

Adicionalmente a los componentes empleados hasta el momento, para la realización de esta práctica se requiere:

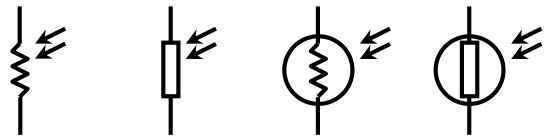
- una fotorresistencia o LDR (*Light Dependent Resistor*), su tipo concreto no es demasiado importante, emplee cualquiera que le sea sencillo localizar;
- un resistor fijo cuyo valor concreto deberá determinar más adelante.

En los siguientes apartados se describe con algo más de detalle lo qué es una LDR y cómo será su conexión

4.2.2. FOTORRESISTENCIAS

Una fotorresistencia o LDR, a veces también llamada *célula fotoconductora*, en un componente semiconductor cuya resistencia eléctrica es función del *flujo luminoso incidente* (*iluminancia*) sobre ella. Los tipos más usuales (aquellos con una respuesta espectral similar a la del ojo) se construyen empleando, como material semiconductor, sulfuro de cadmio (CdS) o seleniuro de cadmio (CdSe). Para los casos en los que se desea que la fotorresistencia sea sensible a otras longitudes de onda suelen emplearse otros materiales, como sulfuro de plomo (PbS) o antimoniuro de indio (InSb) para el infrarrojo medio.

Los símbolos normalmente empleados para representar las LDR son:



El aspecto de una LDR puede verse en la figura 1. Los dos primeros ejemplos muestran el tipo más básico, en el que el material fotosensible (naranja) se encuentra depositado sobre un disco cerámico (blanco), sobre el que se disponen unas metalizaciones (gris) que forman los contactos y que se conectan a los terminales del componente. El cuerpo del componente se protege con una resina epoxídica transparente. El tercer caso muestra una LDR con encapsulado plástico y el último es una LDR con encapsulado metálico.

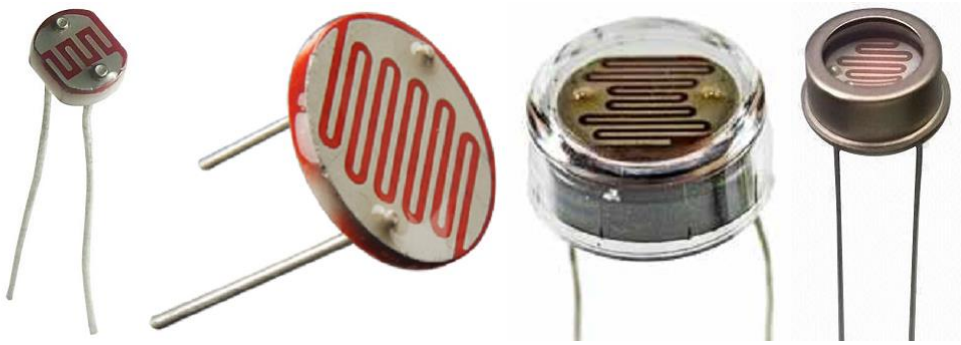


FIGURA 1: aspecto de una LDR.

Como ya se ha dicho, la resistencia R de una LDR es función de la iluminancia E sobre ella. La unidad del SI empleada para medir la iluminancia es el *Lux* (lx). La relación entre la iluminancia y la resistencia en una LDR viene dada por la expresión:

$$R = R_0 \left(\frac{E}{E_0}\right)^{-\gamma} \tag{1}$$

donde E_0 y R_0 son un par iluminancia-resistencia conocido y el parámetro γ suele tomar valores entre 0.5 y 1.0. Como se desprende de la ecuación (1), la resistencia de una LDR disminuye al aumentar la iluminancia y viceversa. Aunque para iluminaciones muy bajas (oscuridad) la ecuación (1) arroja valores muy elevados para la resistencia, dicha ecuación deja de ser válida en tales condiciones debido a la existencia de una resistencia parásita R_D (*Dark Resistance*, resistencia en oscuridad) entre los terminales del componente, de modo que la resistencia real del componente nunca excede el valor R_D.

Al existir una relación potencial —según la ecuación (1)— entre la iluminancia y la resistencia, si dichas magnitudes se representan en una gráfica logarítmica, la expresión (1) da lugar a una recta con pendiente $-\gamma$, como se ve en la figura 2, en la que se han incluido también los valores E_0 y R_0 y el efecto de R_D, que hace que para iluminancias bajas la resistencia no aumente indefinidamente. En la *datasheet* de su LDR encontrará gráficas similares a esta, aunque es posible que no incluyan los valores de E_0 y R_0 y que tampoco muestren la zona en la que el efecto de R_D es apreciable.

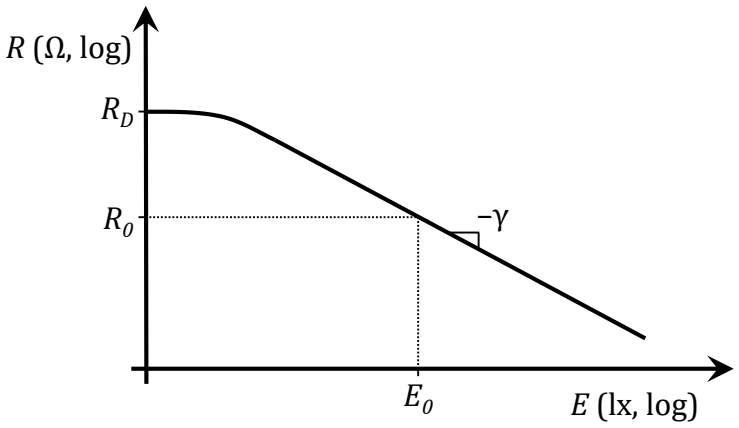
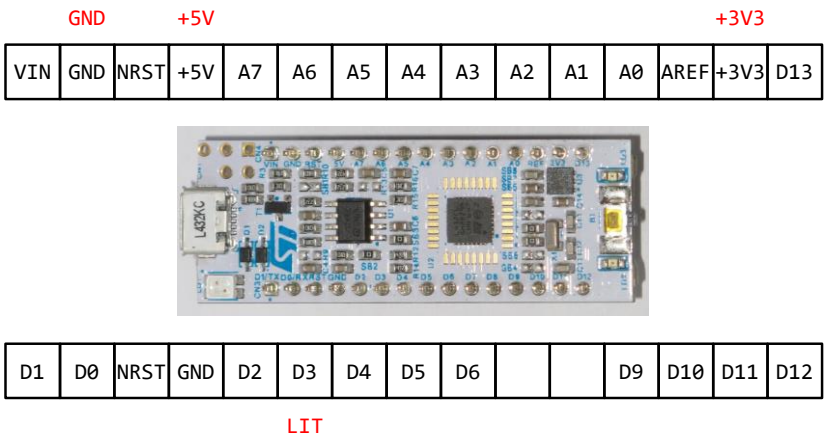


FIGURA 2: gráfica de R en función de E para una LDR.

4.2.3. CIRCUITO DE APLICACIÓN DE LA LDR

Para conectar la LDR al microcontrolador se propone que emplee un divisor resistivo formado por la LDR y un resistor fijo R_{LIT} cuyo valor debe elegir. Este divisor se conectará a masa y a una tensión V_{DD-LDR} que también debe elegir. La tensión a la salida del divisor resistivo V_{LIT} será función de los valores de V_{DD-LDR}, R_{LIT} y R, y al ser esta última función de la iluminancia, también será V_{LIT} función de E.

La salida del divisor resistivo (señal LIT) se conectará al pin indicado a continuación para el microcontrolador STM (se incluye además masa GND y las ***salidas*** de alimentación de +3.3V y +5V que puede emplear para V_{DD-LDR}):



Los requisitos que se imponen para este circuito son:

- Para iluminaciones muy elevadas (superiores a 10 000 lx, sol directo o iluminando con alguna fuente luminosa potente y muy cercana) la tensión V_{LIT} será superior al 90 % de la tensión de alimentación del microcontrolador V_{DD}, que para estas placas es de 3.3 V.
- Para iluminaciones muy bajas (inferiores a 50 lx, habitación con baja luz y tapando la LDR) la tensión V_{LIT} será inferior al 10 % de la tensión de alimentación del microcontrolador V_{DD}.
- Para una iluminación intermedia (entre 250 y 500 lx, habitación iluminada con luz artificial que permita trabajar cómodamente) la tensión V_{LIT} estará comprendida entre el 40 % y el 60 % de la tensión de alimentación del microcontrolador V_{DD}.
- La tensión V_{LIT} nunca será superior a V_{DD} ni negativa.

Deberá montar el anterior divisor resistivo sobre la placa de prototipado, de manera que verifique todos los requisitos anteriores.

4.2.4. VERIFICACIÓN DEL MONTAJE

En este apartado se cargará una aplicación de ejemplo, ya compilada, para simplemente verificar que el conexionado del circuito de la LDR y que el diseño del divisor resistivo son correctos. Para ello, una vez montado el divisor resistivo de la LDR, se volcará el fichero ejecutable .bin adjunto a la práctica (carpeta MICR\P2\S1\E0). Si el cableado y los valores de los componentes son correctos:

- En el *display* de 7 segmentos se mostrará un número del 0 al 99 que indica el porcentaje de V_{DD} al que corresponde V_{LIT}. Esta lectura se refrescará 3 veces por segundo.
- El LED izquierdo se encenderá si el número anterior es menor o igual a 10. El LED medio lo hará si el número anterior es igual o superior a 90. Sin embargo, si el número representado en el *display* está comprendido entre 40 y 60 (ambos inclusive) se encenderán los dos LED.
- El brillo del *display* de 7 segmentos será, aproximadamente, proporcional al número mostrado en él.
- El LED derecho parpadeará a una frecuencia de 1 Hz. La duración del encendido será proporcional al número presentado en el *display*. Si ese número es 0, el LED permanecerá apagado; si es 99, estará encendido constantemente. Para valores intermedios la

duración del encendido variará linealmente entre ambos extremos.

Verifique que las indicaciones en el *display* de 7 segmentos se corresponden con los requisitos de medida impuestos al divisor resistivo que se enumeraron en el apartado 4.2.3*. Si no fuera así, esto sería indicador de que el diseño del divisor no es correcto, en cuyo caso debería rediseñarlo.

4.3. 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 multiplexación será de 250 Hz.

Trabaje sobre la carpeta `MICR\P2\S1\E1` suministrada, que incluye un proyecto de *µVision 5* vacío, 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 deberá emplear los recursos que ya conoce de la librería *mbed* para la generación del evento periódico que controla la multiplexación. No dude, eso sí, en reutilizar el código que considere adecuado de la práctica anterior., particularmente la función `to_7seg()`, ya sea en su versión en C/C++ o en lenguaje de ensamble.

Evite el uso de números en punto flotante. Su uso dará lugar a la inclusión, durante el *linkado*, de las correspondientes librerías para manejo de este tipo de datos, lo que provocará un mayor tamaño de los ejecutables resultantes. **De ahora en adelante tampoco se permite el empleo de números en punto flotante** (es perfectamente posible realizar todas las prácticas sin usarlos). Tenga en cuenta que varios métodos de la librería *mbed* hacen uso de `float`, por lo que no podrá emplearlos. Estos métodos son (para las clases, `PwmOut`, `AnalogIn` y

* En particular: en la oscuridad debe encenderse el LED izquierdo; ante fuerte iluminación lo hará el medio; para una iluminación intermedia —habitación iluminada con luz artificial que permita trabajar cómodamente— lo harán ambos; y para el resto de casos se apagarán los dos.

AnalogOut): `read()`, `write()`, `period()`, `pulsewidth()`, `operator=()` y `operator float()`.

Tampoco se permite el uso de las funciones `wait_us()` y `wait_ns()`, como se indicó 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.

4.4. 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.

4.5. 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 tensión analógica presente en la señal LIT respecto de la tensión de alimentación del microcontrolador. Cuando la tensión en LIT sea baja (próxima a masa) se mostrarán valores bajos, próximos a 0. Si la tensión en LIT es próxima a la tensión de alimentación se mostrarán valores altos, próximos a 99. Entre esos extremos la variación será lineal. El valor representado se refrescará a una frecuencia de 3 Hz. Con

el fin de evitar el uso de números en punto flotante el método `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.

4.6. Ejercicio 4 - LED

Modifique el programa anterior para que, además, el LED derecho luzca intermitentemente a una frecuencia de 100 Hz. La duración del encendido será proporcional al número presentado en el *display*: si ese número es 0, el LED permanecerá encendido solamente durante 100 µs; si es 99, estará encendido constantemente. Para valores intermedios la duración del encendido variará linealmente entre ambos extremos. Recuerde que no deben emplearse números en punto flotante. Puesto que la frecuencia de intermitencia del LED es demasiado elevada como para que el ojo pueda apreciarla, lo que se percibirá será que el brillo del LED varia de forma más o menos proporcional al dato mostrado en el *display*.

Tenga en cuenta que el pin del microcontrolador al que está conectado el LED derecho no tiene capacidad PWM. Por ello, para resolver este apartado, deberá 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 librería *mbed* con parámetros 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.

4.7. 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*, también 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 podrá emplear la clase `PwmOut` de *mbed* para gestionar dichos pines de forma más cómoda a como lo hizo con `Ticker` y `Timeout` en el apartado anterior. La frecuencia del PWM de estas señales será

de 25 kHz. Tome las adecuadas precauciones para no llamar a las funciones de la librería *mbed* con parámetros 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 sesión presencial de esta práctica.

5. TRABAJOS PREVIOS A LA SEGUNDA SESIÓN PRESENCIAL

5.1. 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 código para comprender su funcionamiento. La aplicación pretende mostrar una cuenta ascendente del 0 al 99, incrementándose la cuenta con cada pulsación del pulsador derecho.

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

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

Al pulsar, muchas veces la cuenta se incrementa en más de una unidad. A veces la cuenta también se incrementa al soltar el pulsador.

5.1.1. REBOTES EN LOS PULSADORES

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

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 señal 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.

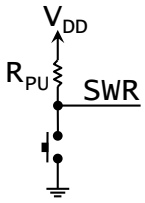


FIGURA 3: circuito de interfaz para un pulsador.

Sin embargo la realidad no es esa. Un pulsador es un componente mecánico en el que, al activarse, un par de conductores deben moverse hasta entrar en contacto mecánico para así cerrar el circuito. Dichos conductores, cuando entran en contacto, rebotan y se separan, de modo que el circuito se cierra durante un breve tiempo para volver a abrirse debido al rebote. Al mantener aplicada la presión del dedo los conductores vuelven a entrar en contacto y el circuito vuelve a cerrarse, pero puede producirse un nuevo rebote que los separe. Este ciclo se repite varias veces, de manera que la forma de onda en la señal SWR es como la vista en la figura 4, en la que se aprecia cómo, tras un primer contacto inicial (en $t = 2$ ms), los contactos se cierran y abren en varias ocasiones durante —en este caso— más de 5 ms. Un programa que base su funcionamiento (como el del apartado anterior) en la ocurrencia de flancos de bajada en la señal del pulsador funcionará de forma muy deficiente, ya que, como se ve, una sola actuación sobre el pulsador ha generado —de nuevo, en este caso— un mínimo de 6 flancos de bajada.

Aún más, dadas las características mecánicas de este tipo de componentes, los rebotes pueden ocurrir no solo al pulsar, sino también al soltar, como puede verse en la figura 5, que muestra la forma de onda en el circuito interfaz de pulsador al soltar el mismo. Como se aprecia, aunque es una operación de apertura de los contactos, se producen rebotes que generan flancos tanto de subida como de bajada en la señal. Debido a estos rebotes el programa del apartado anterior podría incrementar erróneamente la cuenta también al soltar el pulsador.

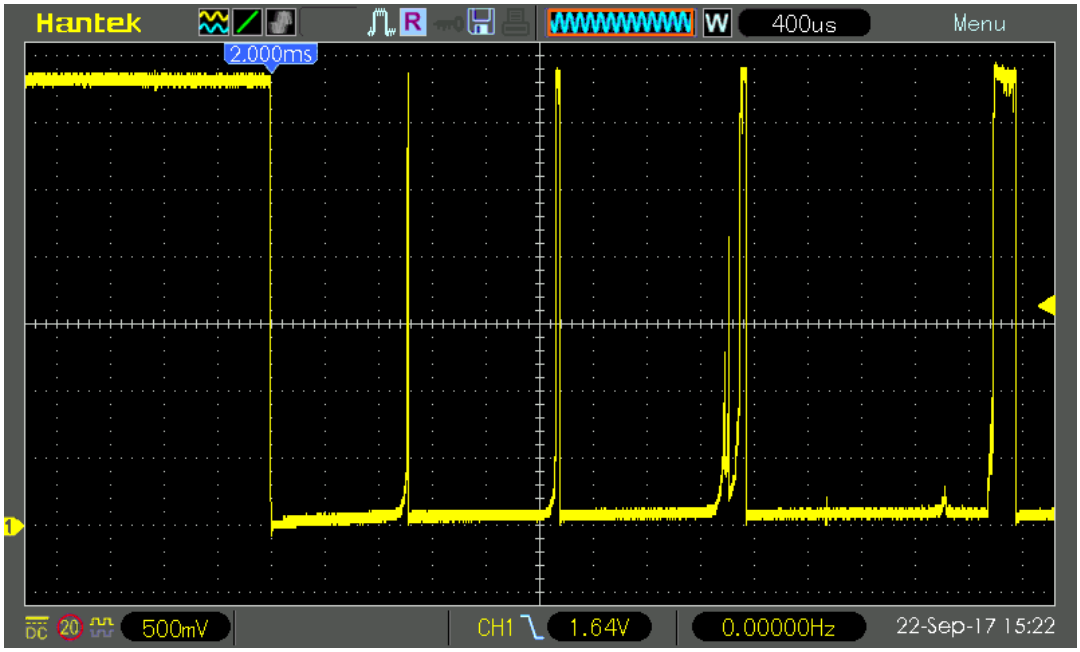


FIGURA 4: forma de onda real a la salida del circuito interfaz de pulsador de la figura 3 tras pulsar.

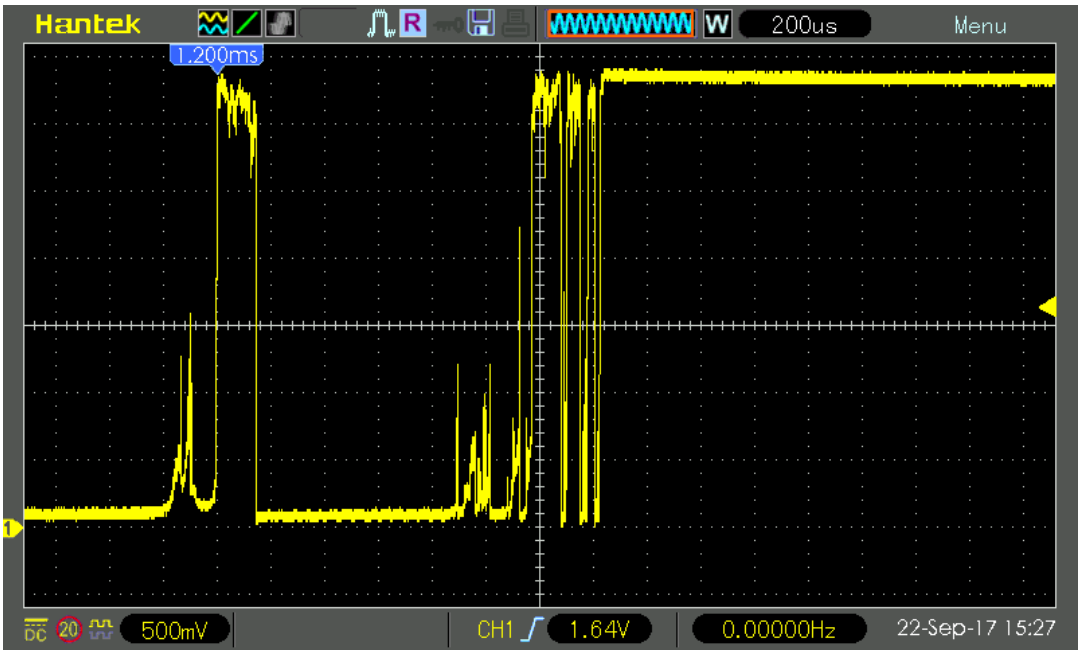


FIGURA 5: forma de onda real a la salida del circuito interfaz de pulsador de la figura 3 tras soltar.

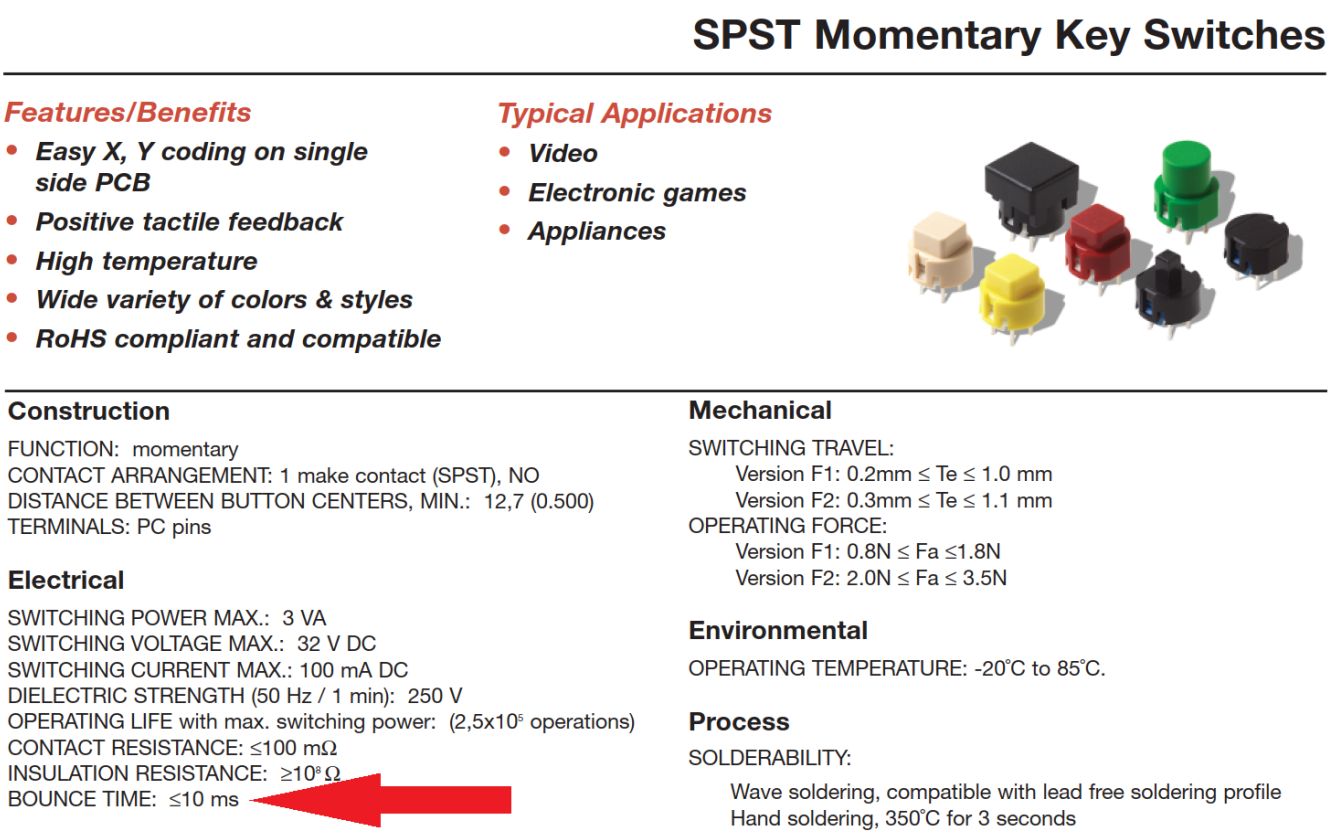
Este fenómeno no es exclusivo de los pulsadores táctiles, está presente, en general, siempre que haya conductores en movimiento que deban entrar en contacto, como ocurre en relés, contactores, termostatos, presostatos, durante la inserción o retirada de conectores, etc.

Es posible diseñar pulsadores en los que estos efectos se minimicen (aunque no se eliminan totalmente). Pero tales pulsadores suelen ser caros, por lo que la *eliminación del efecto de los rebotes en los contactos* (en inglés *debouncing*) suele realizarse: bien por *hardware* (complicando la circuitería de la interfaz, añadiendo normalmente *latches* o condensadores); o, más comúnmente, mediante *software*. La eliminación del efecto de los rebotes mediante *software* es el objeto de la segunda sesión de esta práctica.

5.1.2. EJEMPLOS DE ESTRATEGIAS DE *DEBOUNCING*

En este apartado se presentarán un par de ejemplos de estrategias para la gestión de los rebotes en los pulsadores, aunque ninguna de ellas será suficientemente adecuada (de hecho una será inaceptable en la práctica, como se verá más adelante), por lo que se le pedirá que proponga nuevas estrategias para dicha gestión.

Todas estas estrategias se basan en que los rebotes en los contactos son un fenómeno de carácter eminentemente transitorio, con lo que pasado un tiempo desde la actuación sobre el pulsador (ya sea al pulsar o al soltar), los rebotes desaparecen. De hecho los fabricantes de pulsadores (y de otros componentes basados en contactos móviles) caracterizan el llamado *tiempo de rebotes* (*bounce time*), que fija una cota máxima a la duración de los efectos de los rebotes tras la actuación sobre el pulsador. En la figura 6 puede verse un ejemplo de *datasheet* con la caracterización del *bounce time*. Los pulsadores de mayor calidad presentan tiempos de rebotes bajos, por debajo de 3 ms o incluso 1 ms, mientras que los más mediocres en este sentido pueden tenerlos superiores a 10 ms e incluso 30 ms. Debe tenerse en cuenta, también, que este tiempo puede incrementarse si el pulsador no está convenientemente montado. En particular, en las *protoboards*, en las que el componente no está soldado sino simplemente insertado, el *bounce time* real del pulsador puede ser bastante mayor que el caracterizado por el fabricante en condiciones de montaje ideales.



SPST Momentary Key Switches

Features/Benefits

- ***Easy X, Y coding on single side PCB***
- ***Positive tactile feedback***
- ***High temperature***
- ***Wide variety of colors & styles***
- ***RoHS compliant and compatible***

Typical Applications

- ***Video***
- ***Electronic games***
- ***Appliances***

Construction

FUNCTION: momentary
CONTACT ARRANGEMENT: 1 make contact (SPST), NO
DISTANCE BETWEEN BUTTON CENTERS, MIN.: 12,7 (0.500)
TERMINALS: PC pins

Electrical

SWITCHING POWER MAX.: 3 VA
SWITCHING VOLTAGE MAX.: 32 V DC
SWITCHING CURRENT MAX.: 100 mA DC
DIELECTRIC STRENGTH (50 Hz / 1 min): 250 V
OPERATING LIFE with max. switching power: (2,5x10^5 operations)
CONTACT RESISTANCE: ≤100 mΩ
INSULATION RESISTANCE: ≥10^8 Ω
BOUNCE TIME: ≤10 ms

Mechanical

SWITCHING TRAVEL:
Version F1: 0.2mm ≤ Te ≤ 1.0 mm
Version F2: 0.3mm ≤ Te ≤ 1.1 mm
OPERATING FORCE:
Version F1: 0.8N ≤ Fa ≤1.8N
Version F2: 2.0N ≤ Fa ≤ 3.5N

Environmental

OPERATING TEMPERATURE: -20°C to 85°C.

Process

SOLDERABILITY:
Wave soldering, compatible with lead free soldering profile
Hand soldering, 350°C for 3 seconds

FIGURA 6: extracto de la *datasheet* de unos pulsadores. Se destaca el valor del *bounce time* para los mismos.

5.1.2.1. *Muestreo periódico del estado del pulsador*

Esta técnica se basa es muestrear periódicamente el estado del pulsador y cuando se detecta que durante un número de muestreos *consecutivos* su estado es «pulsado», se determina que la pulsación se ha producido. Igualmente, cuando habiendo sido pulsado anteriormente se detecta que tras una serie de muestreos *consecutivos* su estado es «no pulsado», se determina que se ha terminado la pulsación.

Empleando la librería *mbed* el muestreo periódico del estado del pulsador puede efectuarse muy fácilmente mediante un objeto `Ticker` (para la interrupción periódica) y otro objeto `DigitalIn` para leer el estado del pin asociado al pulsador. En el ejemplo que se mostrará a continuación se supone que el `Ticker` se registra, desde `main()`, de modo que la interrupción asociada se ejecute cada 1 ms. A su vez, para determinar si el pulsador ha sido pulsado o no, se requerirá que *cinco*

lecturas seguidas arrojen el mismo resultado para su estado. Un extracto del código del ejemplo es (se ha eliminado todo lo relativo al sueño y a la multiplexación del *display*):

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

// switch management
static Ticker         g_swr_tick;
static bool volatile  gb_swr_evnt;

static void swr_isr (void) {
  gb_swr_evnt = true;
}

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

  g_swr.mode(PullUp);
  g_swr_tick.attach(swr_isr, 1ms);

  for (;;) {
    if (gb_swr_evnt) {
      gb_swr_evnt = false;           // here every 1 ms
      if (b_swr_state != !g_swr) {  // swr changing?
        if (++swr_cnt >= 5) {
          b_swr_state = !g_swr;
          if (b_swr_state) {
            cnt += ((cnt >= 99) ? -cnt : 1);
          }
        }
      } else {
        swr_cnt = 0;
      }
    }
  }
}
```

El objeto `bool b_swr_state` indica si, tras la gestión de los rebotes, el pulsador está pulsado (`true`) o abierto (`false`). El objeto `uint8_t swr_cnt` lleva la cuenta de las veces, consecutivas, que se detecta que el

pulsador está siendo actuado (ya sea abriéndose o cerrándose). Finalmente el objeto `bool volatile gb_swr_evnt` se activa cuando el programa detecta que el pulsador está siendo pulsado (flanco de bajada en `swr`). Recuerde también que la señal `swr` es activa a nivel bajo, de modo que al actuar sobre el pulsador su valor pasa a ser 0 (por ello el `!g_swr` en la condición del `if` marcado con `// swr changing?`). En la figura 7 se muestra un ejemplo de ejecución de este código, mostrando la señal SWR en el pin y los valores de los objetos recién descritos. En esta figura las líneas verticales de trazos indican los instantes del tiempo en los que la ISR del objeto `Ticker` es llamada (cada 1 ms).

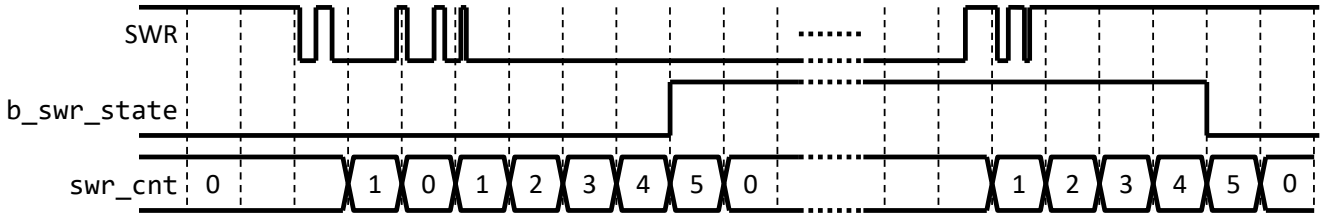


FIGURA 7: ejemplo de funcionamiento para al código basado en `Ticker`.

Analice, compile y pruebe sobre la placa este código que encontrará dentro de la carpeta MICR\P2\S2\E6_Debounce_ticker. Si el funcionamiento no es correcto anote en el siguiente espacio las deficiencias que observe.

Funciona correctamente.

La anterior estrategia es usada frecuentemente para el *debouncing* de pulsadores. Presenta, sin embargo, el problema de que el procesador está sometido a interrupciones periódicas que hacen que sea permanentemente despertado y consuma energía, lo que puede hacer que esta estrategia no sea adecuada para sistemas alimentados a baterías. **En cualquier caso, para este laboratorio no se permitirá el empleo de estrategias de este tipo (`Ticker`) para la gestión de los rebotes.**

5.1.2.2. Medida de tiempos desde el anterior flanco

Esta segunda estrategia se basa en dar como «válido» un determinado flanco en SWR (ya sea de subida o bajada) solo si el flanco anterior a éste ocurrió 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 tendrá en cuenta el primero de ellos. Como en la anterior estrategia, un objeto (`bool b_swr_state`) indicará si, tras la gestión de los rebotes, el pulsador está pulsado (`true`) o abierto (`false`). Igualmente, otros dos objetos (`bool volatile gb_swr_fall_evnt` y `bool volatile gb_swr_rise_evnt`) se activarán cuando el programa detecta que el pulsador está siendo actuado de alguna manera. Para medir el tiempo desde el anterior flanco se empleará un objeto `Timer`. Los flancos se gestionan mediante un objeto de la clase `InterruptIn`, cuyos métodos `fall()` y `rise()` —llamados desde `main()`— registran sendas ISR para la gestión de los flancos de bajada o subida. El código de ejemplo es (de nuevo un extracto, se ha eliminado todo lo relativo al sueño y a la multiplexación 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.elapsed_time() > 10ms)) {
        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.elapsed_time() > 10ms)) {
        b_swr_state = false;
      }
      g_swr_tmr.reset();
    }
  }
}
```

Encontrará este código dentro de la carpeta MICR\P2\S2\E6_Debounce_timer. Analícelo, compílelo y pruébelo sobre la placa. Si el funcionamiento no es correcto anote en el siguiente espacio las deficiencias que observe.

Algunas pulsaciones no las cuenta.

Debe considerarse que esta estrategia presenta, en su implementación con la librería *mbed* y sobre procesadores basados en arquitecturas ARM *Cortex-M* una deficiencia que se describe a continuación.

Si los rebotes generan flancos muy rápidamente, como se ve en la figura 8 (4 flancos en ~8 µs), la librería *mbed* puede no procesar todas las interrupciones generadas. En particular, es posible que algunas inte-

rrupciones no invoquen a su ISR o que *las ISR no sean llamadas en el mismo orden en el que se generaron las interrupciones* (podría llamarse dos veces consecutivas a la ISR de `fall()`, sin una llamada intermedia a la ISR de `rise()`, llamada que podría producirse con posterioridad a las dos llamadas a la otra ISR, por ejemplo). Con otras librerías (*cmsis*) es posible una gestión más precisa de este caso. Aún más, la arquitectura de los procesadores ARM *Cortex-M* no es la más adecuada para estos casos, siendo las arquitecturas ARM *Cortex-R* mucho más apropiadas para una gestión 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 rápidos al pulsar.

5.1.3. PROPUESTAS PARA LA GESTIÓN DE LOS REBOTES

A la vista de todo lo anterior, proponga aquí un par de estrategias (distintas a las anteriores) para la gestión 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 código, solamente describa las ideas en las que se basará. 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**.

5.2. Ejercicio 7 - gestión de un pulsador

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

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

- Con cada pulsación 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 rápidamente, 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*, deberán 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`.

5.3. Ejercicio 8 - gestión de varios pulsadores

Modifique el programa anterior para que:

- La cuenta se incremente con cada pulsación del pulsador derecho.
- La cuenta se decremente con cada pulsación del pulsador izquierdo.
- Si la cuenta vale n y se pulsa el pulsador central, pasará 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, simultáneamente, se actúe sobre los demás y similares.

5.4. Ejercicio 9 - pulsadores y LDR

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

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

6. SUBIDA DE RESULTADOS A MOODLE

Elimine todas las carpetas de nombre ~build y ~listings de las carpetas de los ejercicios E1 a E9. Elimine igualmente todos los ficheros con extensión .bin que se entregaron con la práctica. Comprima entonces la carpeta P2 completa (en formato .7z) y súbala a *Moodle* en el enlace correspondiente. Al emplear el programa *7–Zip* para la compresión emplee la opción Añadir al archivo... y, en la ventana que aparece, seleccione en Nivel de Compresión la opción Ultra. Emplee siempre este mecanismo para generar los ficheros comprimidos que subirá a *Moodle*, en caso contrario el fichero comprimido generado podría tener un tamaño excesivo y ser rechazado por *Moodle*.

Los resultados de las prácticas no puntúan en la calificación del laboratorio, pero es necesario demostrar mediante estos entregables la realización completa de las prácticas para poder aprobar el laboratorio y la asignatura. Por ello estos entregables son pruebas de evaluación y están sujetos a la normativa vigente en la UPM en cuanto al fraude académico se refiere. No suba entregables que no haya realizado usted personalmente, que no sean originales y tampoco comparta sus resultados con otras personas.

Terminan aquí las tareas a realizar para esta práctica.