# MICROPROCESADORES

## PRÁCTICA 3

## 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 2020 – 2021

# ÍNDICE



## 1. INTRODUCCIÓN

En este enunciado se describen los trabajos a realizar durante esta tercera práctica. La práctica se divide en tres semanas, detallándose en las siguientes secciones el trabajo que debe realizarse en cada una de ellas. Adicionalmente, se describen también los trabajos *previos* que deben realizarse con anterioridad a cada una de las sesiones de la práctica.

El objetivo 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). Se añadirá finalmente un sensor telemétrico ultrasónico para medir distancias, que será controlado haciendo uso de varios de los recursos estudiados en la práctica.

Una vez conocidos los mecanismos básicos de gestión de eventos (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.

**Adjunto al enunciado encontrará en *Moodle* un fichero comprimido que, al descomprimirlo (dentro de la carpeta MICR), genera una estructura de carpetas 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.**

**Antes del fin de cada sesión de la práctica deberá comprimir la carpeta MICR\P3 (que incluye los resultados de su trabajo) y subirla a *Moodle*. Elimine antes todas las carpetas ~build y ~listings de todos los proyectos de *Keil µVision 5* y también todos los ejecutables de ejemplo (.bin) que se han suministrado. Estos entregables deberán ser subidos a *Moodle* por todos los integrantes de cada puesto de laboratorio. Si no se subiese el entregable o se hiciese después del final de la correspondiente sesión de laboratorio, se**

**entenderá la entrega como no realizada y se calificará con 0 puntos.**

### 1.1. 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;
- un sensor telemétrico ultrasónico del tipo *HC-SR04*.

Las dos primeras deberán ser adquiridas por cada estudiante de laboratorio con anterioridad a la primera sesión de esta práctica. El sensor telemétrico será necesario previo a la tercera sesión. Estos elementos se pueden adquirir muy fácilmente en *Internet* o en tiendas especializadas.

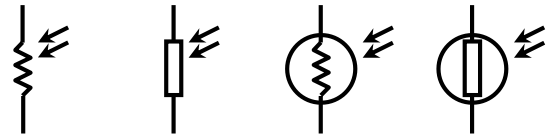
## 2. PRIMERA SESIÓN

### 2.1. Trabajos previos a la primera sesión presencial

#### 2.1.1. 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.

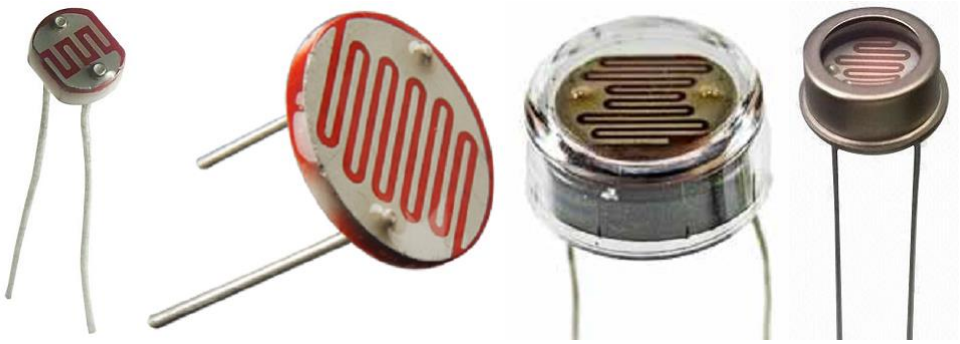


FIGURA 1: aspecto de una LDR.

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 $\gamma$ 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 ecua-

ción (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.

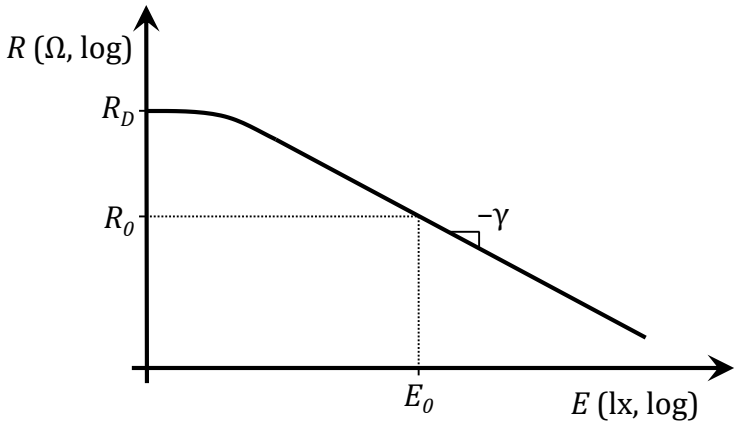



FIGURA 2: gráfica de $R$ en función de $E$ para una LDR.

### 2.1.2. 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 cada uno de los dos microcontroladores:

- para el microcontrolador NXP (se incluye además masa GND y las ***salidas*** de alimentación de +3.3V y +5V que puede emplear para $V_{DD-LDR}$):

+3V3 +5V

| VOUT | VU | IF- | IF+ | RD- | RD+ | TD- | TD+ | D- | D+ | p30 | p29 | p28 | p27 | p26 | p25 | p24 | p23 | p22 | p21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

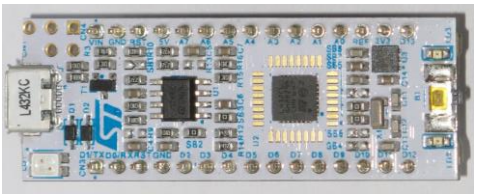
| GND | VIN | VB | nR | p5 | p6 | p7 | p8 | p9 | p10 | p11 | p12 | p13 | p14 | p15 | p16 | p17 | p18 | p19 | p20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

GND LIT

- y para el microcontrolador STM (se incluye además masa GND y las ***salidas*** de alimentación de +3.3V y +5V que puede emplear para $V_{DD-LDR}$):

GND +5V +3V3

| VIN | GND | NRST | +5V | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | AREF | +3V3 | D13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



| D1 | D0 | NRST | GND | D2 | D3 | D4 | D5 | D6 | | | D9 | D10 | D11 | D12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

LIT

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.

### 2.1.3. 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 es correcto. Para ello, una vez montado el divisor resistivo de la LDR se volcará el fichero ejecutable .bin adjunto a la práctica (carpeta MICR\Previo_S1). 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.

**Al inicio de la primera sesión el docente comprobará que el montaje de todo lo anterior se ha realizado y que la aplicación de prueba (que ya debe venir cargada en la placa) funciona correcta-**

**mente. También le preguntará cómo se han obtenido la topología del divisor resistivo y los valores de $V_{DD-LDR}$ y $R_{LIT}$, y también qué valores obtenidos de la *datasheet* (o medidos) de la LDR han sido relevantes en estos cálculos.**

Terminan aquí los trabajos previos a realizar antes de la primera sesión presencial de esta práctica.

## 2.2. Trabajos durante la primera sesión presencial

### 2.2.1. INTRODUCCIÓN

En esta sesión 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.

Un efecto de no emplear la librería *sw_tick_serial* es que la configuración de las comunicaciones serie para el uso de `printf()` es diferente a lo expuesto en la práctica anterior. En particular, el *baud rate* pasa de ser 115 200 a 9 600 sin la librería. Téngalo en cuenta si desea usar `printf()` para la depuración.

### 2.2.2. 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.

Como ya se ha dicho, 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. Sí puede usar, en cambio, la librería *to_7seg* que creó en la anterior práctica para la conversión a 7 segmentos. No dude en reutilizar el código que considere adecuado de anteriores prácticas.

Trabaje sobre la carpeta `MICR\P3\S1\E1` suministrada, que incluye un proyecto de *µVision 5* vacío, pero configurado para trabajar con su microcontrolador.

**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, pudiendo ocasionar problemas si trabaja con la versión gratuita de la herramienta *Keil µVision 5*, como sabe limitada a un tamaño máximo de ejecutable de 32 KiB. **Se espera que de ahora en adelante nunca se empleen números en punto flotante** (es perfectamente posible realizar todas las prácticas sin usarlos).

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.**

### 2.2.3. 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\P3\S1\E2` y trabaje sobre esta copia.

### 2.2.4. 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\P3\S1\E3` y trabaje sobre esta copia.

**Deberá enseñar al docente su programa funcionando.**

### 2.2.5. 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 5 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.

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\P3\S1\E4` y trabaje sobre esta copia.

### 2.2.6. 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 un ciclo de trabajo para un `PwmOut` que dé lugar a una anchura de pulso inferior a 1 µs y situaciones similares). Copie el proyecto del apartado anterior sobre la carpeta `MICR\P3\S1\E5` y trabaje sobre esta copia.

**Deberá enseñar al docente su programa funcionando. También le mostrará las señales DSL y DSR en el osciloscopio, para comprobar sus formas de onda.**

Terminan aquí las tareas a realizar durante la primera sesión presencial de esta práctica.

# 3. Segunda Sesión

## 3.1. Trabajos previos a la segunda sesión presencial

### 3.1.1. Prueba de un control de los pulsadores

Abra el proyecto de *Keil µVision 5* que se adjunta (carpeta MICR\P3\Previo_S2\1_Bouncing) 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.**

### 3.1.2. 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.

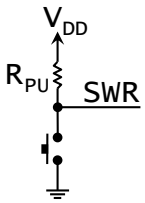



FIGURA 3: circuito de interfaz para un pulsador.

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.

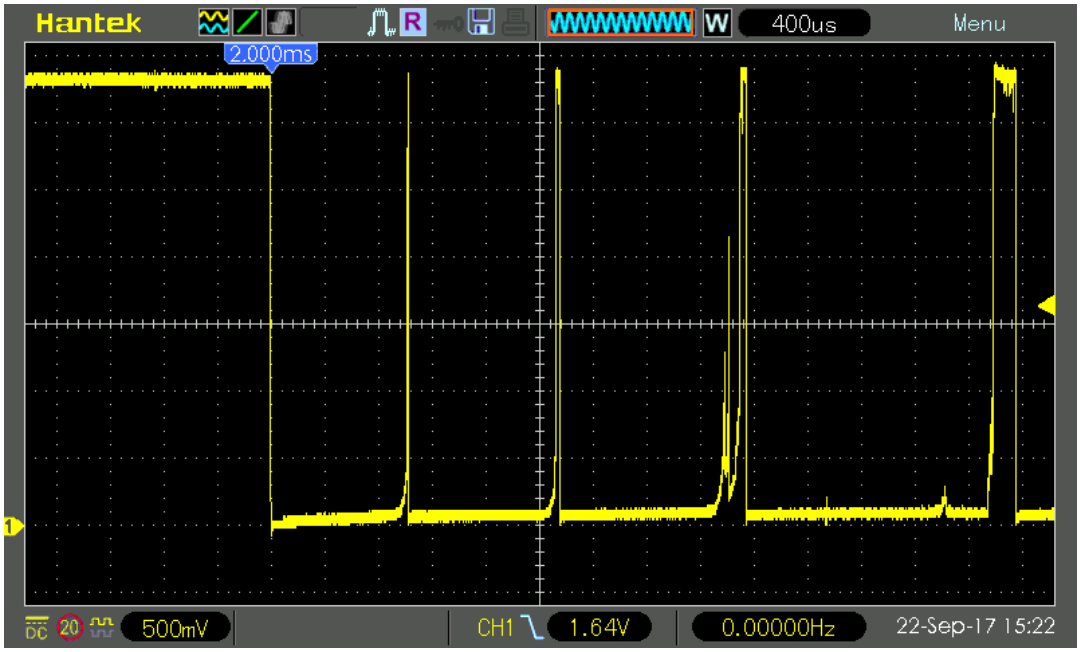



FIGURA 4: forma de onda real a la salida del circuito interfaz de pulsador de la figura 3 tras pulsar.

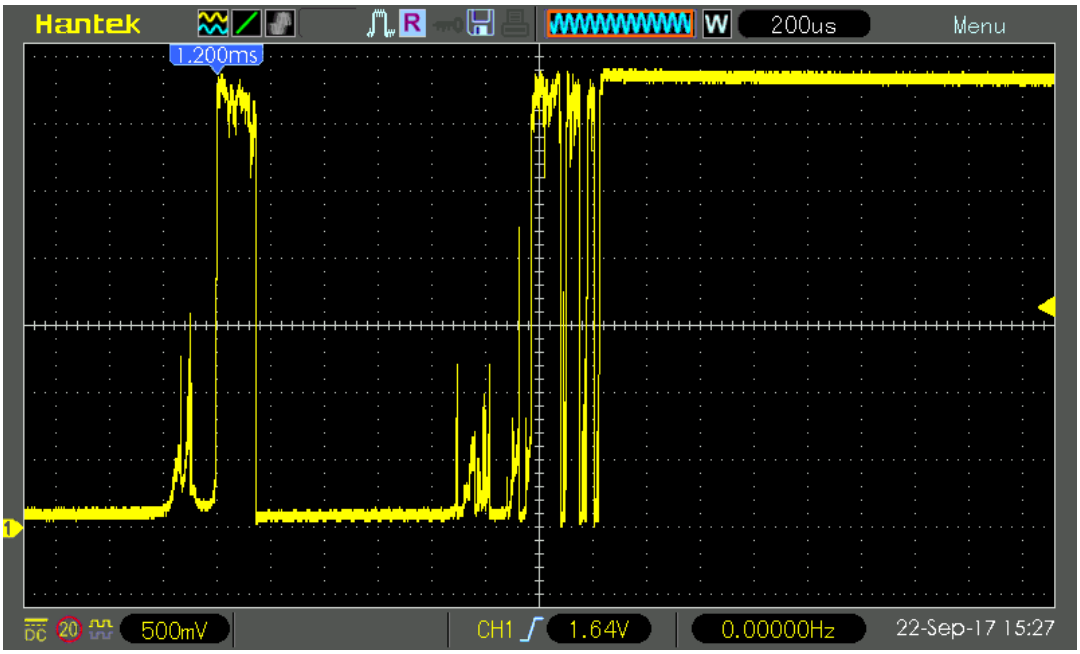



FIGURA 5: forma de onda real a la salida del circuito interfaz de pulsador de la figura 3 tras soltar.

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.

### 3.1.3. 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 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: ($2{,}5 \times 10^5$ operations)
CONTACT RESISTANCE: ≤100 mΩ
INSULATION RESISTANCE: $\geq 10^8$ 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.

#### 3.1.3.1. *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 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_us(swr_isr, 1000);  // 1 ms, 1000 Hz

  for (;;) {
    if (gb_swr_evnt) {
      gb_swr_evnt = false;           // here every 1 ms
      if (b_swr_state != !g_swr) {  // swr changing?
        if (swr_cnt++ > 3) {  // this means 5 times, not 3
          b_swr_state = !b_swr_state;
          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).

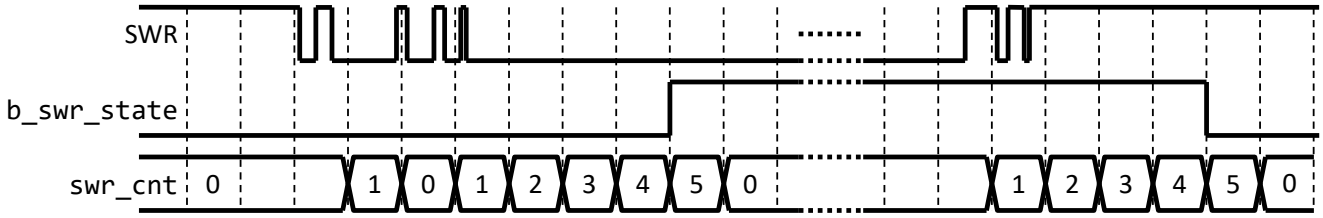



FIGURA 7: ejemplo de funcionamiento para al código basado en `Ticker`.

Analice, compile y pruebe sobre la placa este código que encontrará dentro de la carpeta MICR\P3\Previo_S2\2_Debounce_ticker.

**Si el funcionamiento no es correcto anote en el siguiente espacio las deficiencias que observe.**

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**

**este laboratorio no se permitirá el empleo de estrategias de este tipo (`Ticker`) para la gestión de los rebotes.**

#### *3.1.3.2. 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.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();
    }
  }
}
```

Encontrará este código dentro de la carpeta MICR\P3\Previo_S2\3_Debounce_timer. Analícelo, compílelo y pruébelo 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 implementación con la librería *mbed* y sobre procesadores basados en arquitecturas ARM *Cortex-M* un par de deficiencias:

- Los objetos `Timer` de la librería *mbed* solo permiten la medida de tiempos hasta, aproximadamente, 35 minutos. Aunque poco probable, podría ser posible que, tras 35 minutos sin actuar sobre el pulsador, una pulsación sobre él pasase inadvertida.
- 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 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* (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.

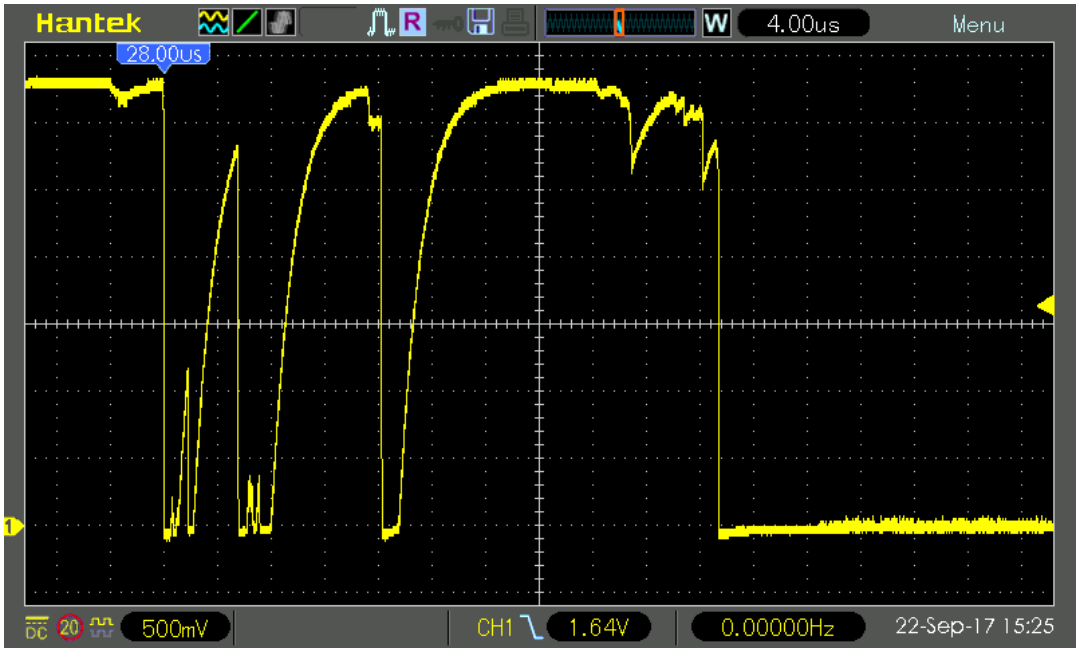



FIGURA 8: rebotes rápidos al pulsar.

### 3.1.4. 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**.

Terminan aquí los trabajos previos a realizar antes de la segunda sesión presencial de esta práctica.

## 3.2. Trabajos durante la segunda sesión presencial

### 3.2.1. Ejercicio 6 - 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\P3\S2\E6 copiando en ella el ejercicio de la carpeta MICR\P3\S1\E2.

**Deberá enseñar al docente su programa funcionando.**

### 3.2.2. Ejercicio 7 - 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\P3\S2\E7 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 (especialmente si ha decidido emplear autómatas para resolver este apartado).** 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.

#### 3.2.3. EJERCICIO 8 - 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.

Trabaje sobre la carpeta MICR\P3\S2\E8 copiando en ella el proyecto del ejercicio anterior.

Terminan aquí las tareas a realizar durante la segunda sesión presencial de esta práctica.

**Deberá enseñar al docente su programa funcionando.**

## 4. TERCERA SESIÓN

### 4.1. Trabajos previos a la tercera sesión presencial

#### 4.1.1. SENSOR TELEMÉTRICO ULTRASÓNICO

Un sensor telemétrico se emplea para la medida de distancias. En el caso de los sensores telemétricos ultrasónicos el mecanismo empleado para la medida es el envío y la recepción de una señal acústica de ultrasonidos. En este laboratorio se empleará un sensor telemétrico ultrasónico del tipo *HC-SR04*.

El sensor telemétrico ultrasónico *HC-SR04* basa su funcionamiento, al igual que un sistema RADAR, en el envío de una señal hacia un objetivo y la reflexión de esa señal, en forma de eco, de nuevo hacia el sensor, tal como se ilustra en la figura 9.

El tiempo transcurrido desde la emisión de la señal hasta la recepción del eco es proporcional a la distancia total recorrida por la señal, que es el doble de la distancia entre el sensor y el objetivo, siendo la constante de proporcionalidad la velocidad de la señal en el medio, de modo que:

$$d = \frac{c \cdot \Delta t}{2}$$

donde $\Delta t$ es el tiempo transcurrido desde que se emite la señal hasta que se recibe el eco; $d$ es la distancia entre sensor y objetivo y $c$ es la velocidad del sonido en el medio (para el caso de sistemas RADAR o LIDAR, $c$ sería la velocidad de la luz en el medio).

La velocidad del sonido en el aire es función de la temperatura del mismo, para una temperatura ambiente $T_A = 25\ °C$, dicha velocidad es $c = 346.13\ \text{m/s}$. A esta temperatura, si la distancia $d$ se mide en cm y el retardo $\Delta t$ en µs, entonces es:

$$d(\text{cm}) = \frac{346.13\ \text{m/s} \cdot \frac{100\ \text{cm}}{\text{m}} \cdot \frac{\text{s}}{10^6\ \mu\text{s}} \cdot \Delta t(\mu\text{s})}{2} =$$

$$= 0.0173065\ \text{cm}/\mu\text{s} \cdot \Delta t(\mu\text{s}) =$$

$$= \frac{\Delta t(\mu\text{s})}{57.7818\ldots\ \mu\text{s/cm}} \approx \frac{\Delta t(\mu\text{s})}{58\ \mu\text{s/cm}}$$

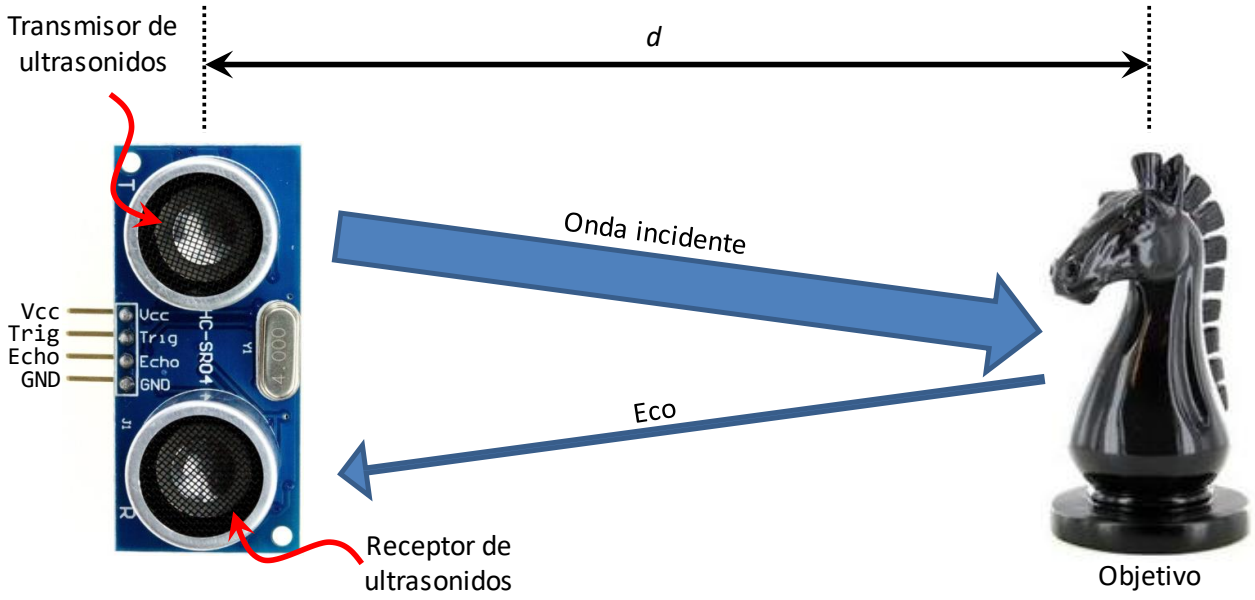



FIGURA 9: principio de funcionamiento de un sensor telemétrico ultrasónico.

con lo que, en definitiva:

$$d(\text{cm}) \approx \frac{\Delta t(\mu\text{s})}{58\ \mu\text{s/cm}} \quad (2)$$

De este modo, una vez medido —en µs— el retardo desde la emisión de la señal hasta la recepción del eco, sólo es necesario dividir el resultado de esa medida entre 58, determinando así la distancia al objetivo —en cm—.

Para poder realizar la medida del retardo $\Delta t$, el sensor *HC-SR04* no emplea una onda incidente continuada indefinidamente en el tiempo, sino que genera una *salva* de 8 pulsos de una frecuencia de 40 kHz (fuera, por tanto, del rango auditivo normal, de 20 Hz a 20 kHz), tal como se aprecia en la figura 10, que muestra la señal de excitación del transmisor ultrasónico del sensor.

El sensor espera entonces a recibir el eco proveniente del objetivo. La señal de eco —que puede verse en la figura 11, junto con la salva—, es procesada por el sensor *HC-SR04* para obtener el retardo $\Delta t$.

Todo el procesado necesario para, a partir de esas señales, obtener el retardo $\Delta t$ es realizado por el sensor *HC-SR04* de forma transparente al usuario. Para él, la interfaz eléctrica del sensor muestra un funcionamiento muy simple que se describe en el siguiente apartado.

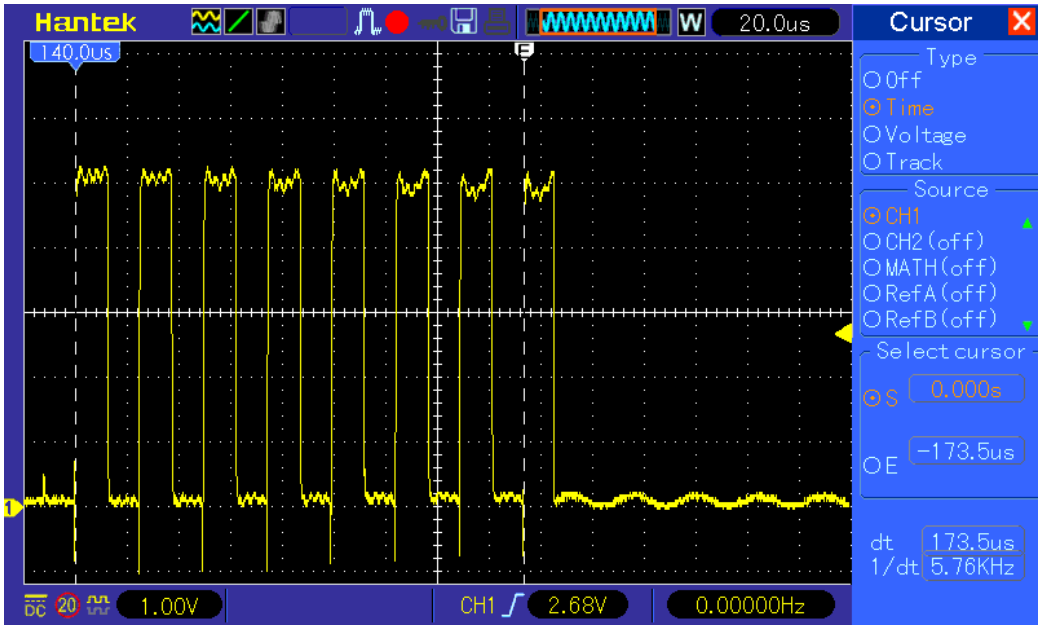



FIGURA 10: salva transmitida por el transmisor de ultrasonidos del sensor *HC-SR04*.

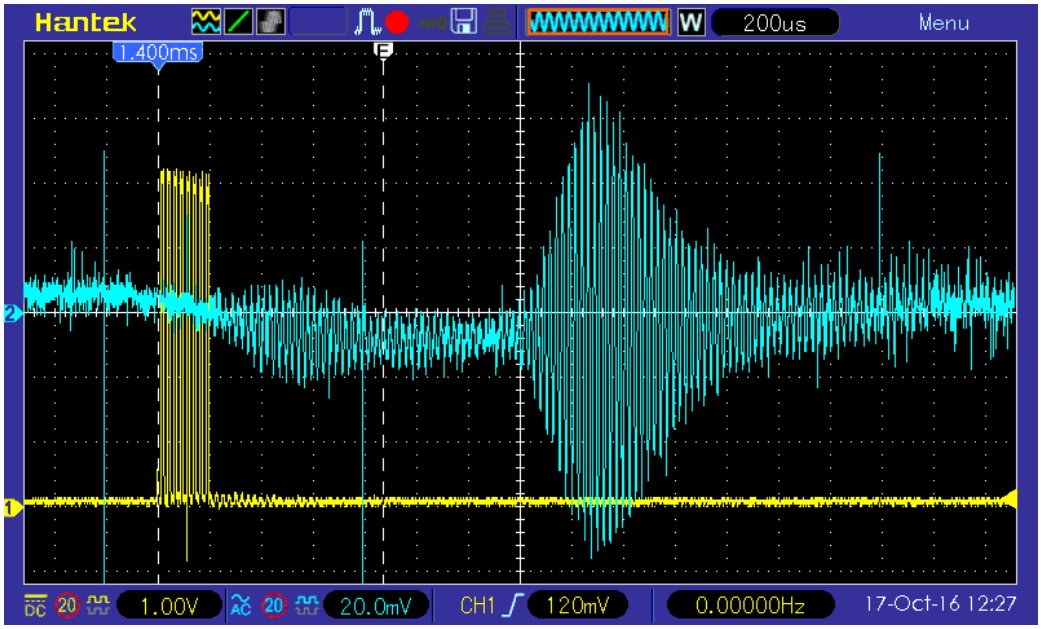



FIGURA 11: salva transmitida por el transmisor de ultrasonidos (en amarillo) y eco (en azul) recibido por el receptor de ultrasonidos del sensor *HC-SR04*. La distancia al objetivo era de unos 24 cm.

### 4.1.2. INTERFAZ ELÉCTRICA Y MONTAJE

La interfaz eléctrica del sensor telemétrico ultrasónico *HC-SR04* está formada, tal como se ve en la figura 9, por cuatro terminales: *VCC*, *GND*, *TRIG* y *ECHO*. Sus funciones son:

*VCC*: tensión de alimentación. Su valor nominal, según se recoge en la *datasheet* del sensor, es de 5 $V_{DC}$;

*GND*: masa;

*TRIG*: *entrada* de disparo (*trigger*). Cada vez que por esta entrada se aplica un pulso a nivel alto de duración superior a 10 µs, el sensor *HC-SR04* genera una salva e inicia todas las operaciones necesarias para procesar el eco recibido y determinar el tiempo de retardo $\Delta t$. La separación entre dos pulsos consecutivos de *TRIG* debe ser, según la *datasheet*, de al menos 60 ms. Esta entrada acepta niveles lógicos TTL, de modo que, aunque los microcontroladores del laboratorio trabajan con una tensión de alimentación de 3.3 V, los pul-

sos generados por ellos son interpretados correctamente por el sensor *HC-SR04*;

*ECHO*: salida de eco. Una vez que el sensor ha procesado el eco y determinado el tiempo de retardo $\Delta t$, entrega por esta salida un pulso a nivel alto cuya duración es igual a $\Delta t$. A través de este valor de $\Delta t$, mediante la expresión (2), se puede determinar la distancia al objetivo $d$. La salida *ECHO* trabaja con niveles TTL, pero los microcontroladores del laboratorio toleran, e interpretan correctamente, dichos niveles TTL aunque se alimenten a 3.3 V, tal como se describe en las *datasheet* y el *user manual* de los mismos.

En la figura 12 se muestra el funcionamiento de esta interfaz. En este caso, el objetivo se encontraba a una distancia aproximada de 10 cm y la anchura del pulso en *ECHO*, como se ve en la figura, es de 590 µs, en buena concordancia con la expresión (2).

Puesto que ambos extremos de la conexión (*HC-SR04* y microcontrolador) soportan niveles TTL, no es necesario introducir ninguna circuitería de adaptación de niveles lógicos entre ellos. Debe tenerse, sin embargo, cuidado a la hora de interconectarlos, ya que conectar el pin *ECHO* del sensor (una salida) a otro pin de salida del microcontrolador puede ocasionar la rotura de algún componente. Del mismo modo, téngase especial cuidado al conectar los pines *VCC* y *GND* del sensor a +5 V y masa respectivamente, hacerlo al revés puede ocasionar que el sensor se estropee de forma irreversible.

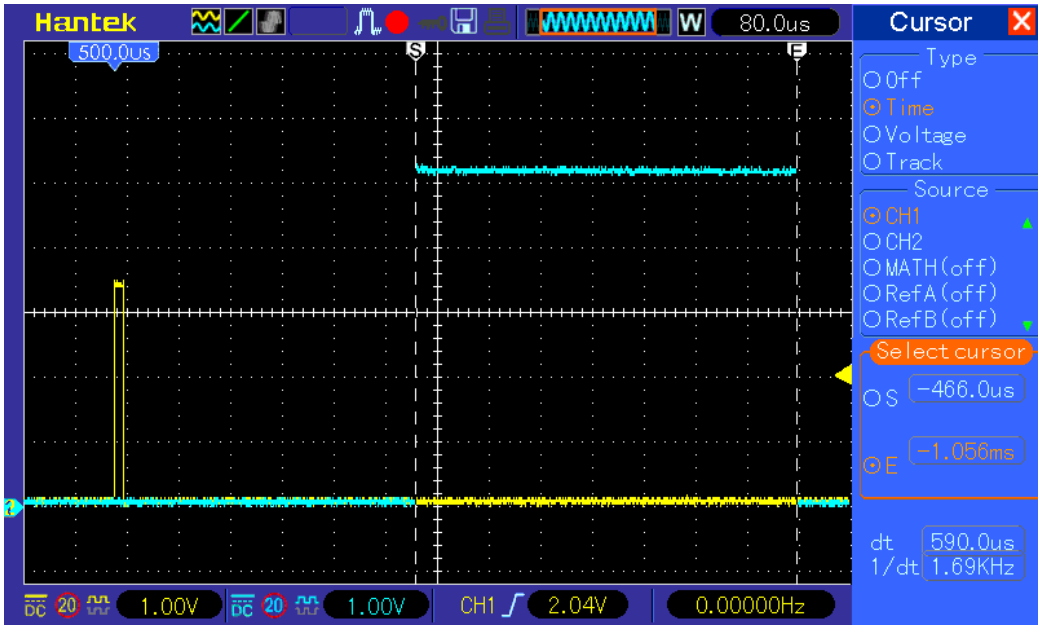
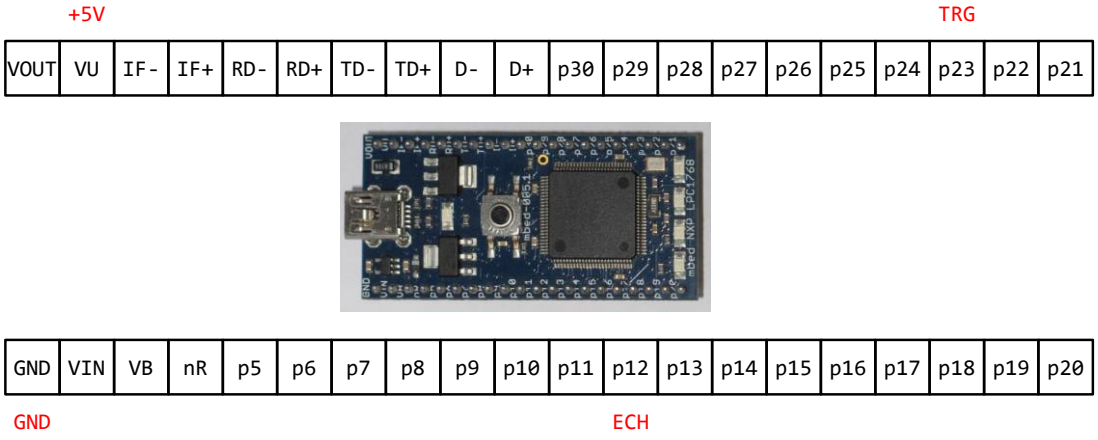


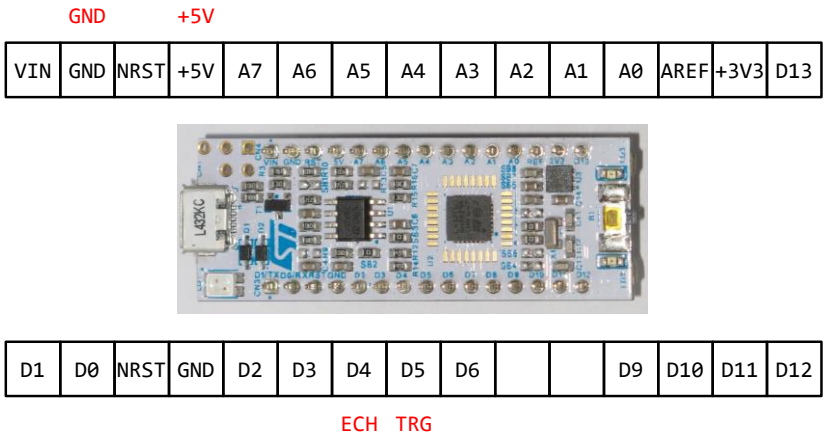

FIGURA 12: señales *TRIG* (en amarillo) y *ECHO* (azul) que ilustran el funcionamiento de la interfaz del sensor *HC-SR04*. El objetivo se encontraba a unos 10 cm de distancia del sensor.

El sensor se conectará a los microcontroladores a través de las señales TRG y ECH para los puertos TRIG y ECHO del sensor, respectivamente. Dichas señales se conectarán a los pines que se indican ahora:

- para el microcontrolador NXP:




- y para el microcontrolador STM:




### 4.1.3. VERIFICACIÓN DEL MONTAJE

En este apartado se cargará una aplicación de ejemplo, ya compilada, para simplemente verificar que el sensor *HC-SR04* está correctamente conectado. Para ello, una vez montado el sensor se volcará el fichero ejecutable .bin adjunto a la práctica (carpeta MICR\P3\Previo_S3). Si el cableado es correcto:

- En el *display* de 7 segmentos se mostrará un número del 0 al 99 que indica la distancia en cm desde el sensor hasta al objetivo. Esta lectura se refrescará 3 veces por segundo.
- Si la distancia al objetivo es mayor a 99 cm, en el *display* se mostrará el mensaje «– –».
- Si el sensor no responde (no se recibe pulso por ECHO), en el *display* se mostrará el mensaje «Er».
- Se transmitirá (con `printf()`, para ser visualizado con *Tera Term* empleando la misma configuración que en último ejercicio de la práctica 2) la información mostrada en el *display*.

**Al inicio de la tercera sesión de la práctica el docente comprobará que el montaje del sensor se ha realizado y que la aplicación de prueba (que ya debe venir cargada en la placa) funciona correctamente.**

Terminan aquí los trabajos previos a realizar antes de la tercera sesión presencial de esta práctica.

## 4.2. Trabajos durante la tercera sesión presencial

### 4.2.1. Medidas sobre el sensor

En este apartado deberá verificarse experimentalmente el funcionamiento del sensor *HC-SR04*. Para ello se le aplicarán una serie de pulsos, a nivel alto, de entre 10 µs y 1 ms de duración y a una frecuencia de 10 Hz por su entrada *TRIG*. Con el osciloscopio se visualizarán simultáneamente las señales *TRIG* y *ECHO* del sensor, midiendo el ancho del pulso de la señal *ECHO* y verificando, para varias distancias al objetivo (10 cm, 30 cm y 1 m), que las medidas son compatibles con la expresión (2).

Para generar los pulsos en *TRIG* puede emplear la propia tarjeta del microcontrolador (PWM) o el generador de funciones del laboratorio.

**Deberá enseñar al docente sus medidas (incluyendo las capturas de pantalla del osciloscopio), de modo que se confirme el funcionamiento del sensor telemétrico.**

*Para el semestre de primavera del curso 2020-21 la tercera sesión no será presencial, por lo que no se requiere que realice ni enseñe estas medidas al docente.*

### 4.2.2. Ejercicio 9 - control del sensor

Deberá escribir un programa que muestre en el *display* de 7 segmentos la distancia, en cm, a la que se encuentra el objetivo. Si la distancia es superior a 99 cm en el *display* se mostrará el mensaje «– –». La medida de distancia se realizará 10 veces por segundo y con una anchura en el pulso de *trigger* de 1 ms.

Se recomienda emplear un conjunto de `Ticker` y `Timeout` para generar la señal de *trigger* del sensor y emplear objetos `InterruptIn` y `Timer` para determinar la duración del pulso de *echo*. Un segundo `Ticker` se empleará para la multiplexación del *display*.

Trabaje sobre la carpeta `MICR\P3\S3\E9`, copiando en ella algún proyecto de ejercicios anteriores, el que considere más adecuado.

**Existen disponibles varias clases para la librería *mbed*, aportadas por la comunidad, capaces de gestionar directamente el sensor**

***HC-SR04*. Para la realización de esta práctica no se permite el uso de ninguna de ellas. Es obligatorio emplear para la gestión de este sensor los recursos que la librería *mbed* ofrece en cuanto a gestión de interrupciones y *timers*.**

**Deberá enseñar al docente su programa funcionando.**

Terminan aquí las tareas a realizar durante la tercera sesión presencial de esta práctica.

### 4.2.3. SUBIDA DE RESULTADOS A MOODLE

**Elimine todas las carpetas ~build y ~listings de todos los proyectos de *Keil µVision 5* de esta práctica. Borre también todos los ejecutables de ejemplo (.bin) que se han suministrado y, solo entonces, comprima la carpeta MICR\P3 en formato .7z (nivel de compresión Ultra). Suba este archivo comprimido a *Moodle* en el enlace correspondiente (una entrega por cada estudiante, ambos miembros de la pareja deberán subir el mismo fichero).**

Terminan aquí las tareas a realizar para esta práctica.