MICROPROCESADORES

PRÁCTICA 3

AUTÓMATAS CONTROLADOS POR EVENTOS

DISEÑO FINAL

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 tercera y última práctica. La práctica dispone de cuatro 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 planifique ese trabajo de la forma que le sea más conveniente.

La práctica, a su vez, se organiza en dos partes diferenciadas:

- Una primera en la que se afrontarán problemas similares mediante dos técnicas diferentes, por un lado la gestión de eventos independientes que ya conoce de la práctica anterior y por otro mediante la aplicación de la gestión de eventos interdependientes mediante el empleo de máquinas de estados finitos (FSM) controladas por eventos. Esta parte de la práctica ocupa hasta la primera sesión.
- Una segunda en que se emplea la técnica de máquinas de estados finitos controladas por eventos para la resolución de una aplicación completa que hace uso de varias FSM trabajando de forma cooperativa. Esta parte de la práctica ocupa las tres sesiones restantes.

2. OBJETIVOS

El objetivo de esta práctica es que aplique adecuadamente el diseño de sistemas reactivos basado en máquinas de estados finitos controladas por eventos y que interactúan entre ellas. En una primera etapa de la práctica se le pedirá que resuelva un mismo problema empleando las técnicas de gestión de eventos independientes, primeramente, y las máquinas de estados finitos controladas por eventos, después. El problema al que se enfrentará es la gestión de un sensor telemétrico ultrasónico cuyos detalles se describirán más adelante. Para terminar esta primera etapa de la práctica se le pedirá que añada algún detalle a la funcionalidad del autómata (particularidades del diseño que serían de

más compleja implementación empleando la técnica de gestión de eventos independientes, pero que son fáciles de implementar mediante una FSM).

En la segunda parte de la práctica (3 últimas sesiones) se enfrentará al diseño de una aplicación completa que hace uso de máquinas de estados finitos controladas por eventos, en un escenario en el que varias de estas máquinas trabajan conjuntamente, intercambiándose mensajes, para resolver el problema. Se le pedirá que vaya construyendo dicha aplicación de forma incremental (una FSM cada vez) añadiendo a lo visto en la primera parte de la práctica:

1. un autómata para gestionar un pulsador;
2. otra FSM para manejar la multiplexación del *display* y su brillo;
3. una última máquina de estados finitos para gestionar completamente la aplicación.

Estas etapas están diseñadas de modo que cada una puedan probarse también de forma incremental.

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 esta primera parte de la práctica montará en su placa un sensor telemétrico ultrasónico y, empleando dos enfoques diferentes, lo controlará por *software* para realizar medidas de distancia.

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:

- un sensor telemétrico ultrasónico del tipo *HC-SR04*. En *Moodle* encontrará, en los archivos de esta práctica, la *datasheet* del mismo.

4.2.2. 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 1.

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 Δt 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 y para una temperatura ambiente $T_A = 25$ °C dicha velocidad es $c = 346.13$ m/s. A esta temperatura, si la distancia d se mide en cm y el retardo Δt en µs, entonces es:

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

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

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

con lo que, en definitiva:

$$d(\text{cm}) \approx \frac{\Delta t(\mu\text{s})}{58\ \mu\text{s/cm}} \tag{1}$$

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—.

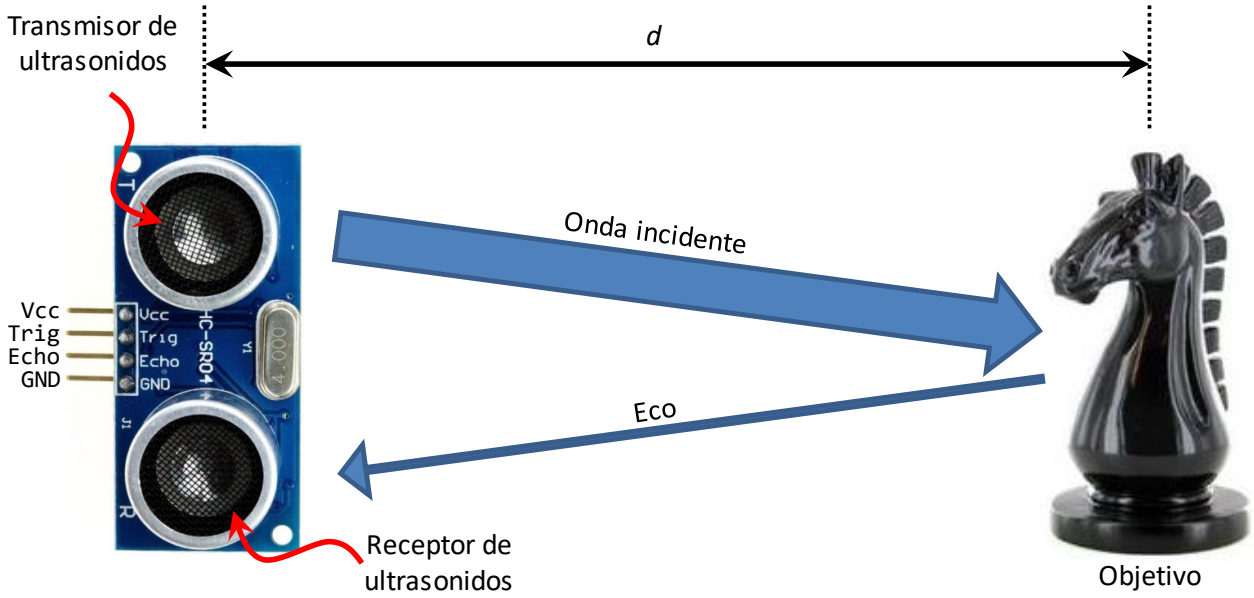


FIGURA 1: principio de funcionamiento de un sensor telemétrico ultrasónico.

Para poder realizar la medida del retardo Δt, 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 2, que muestra la señal de excitación del transmisor ultrasónico del sensor.

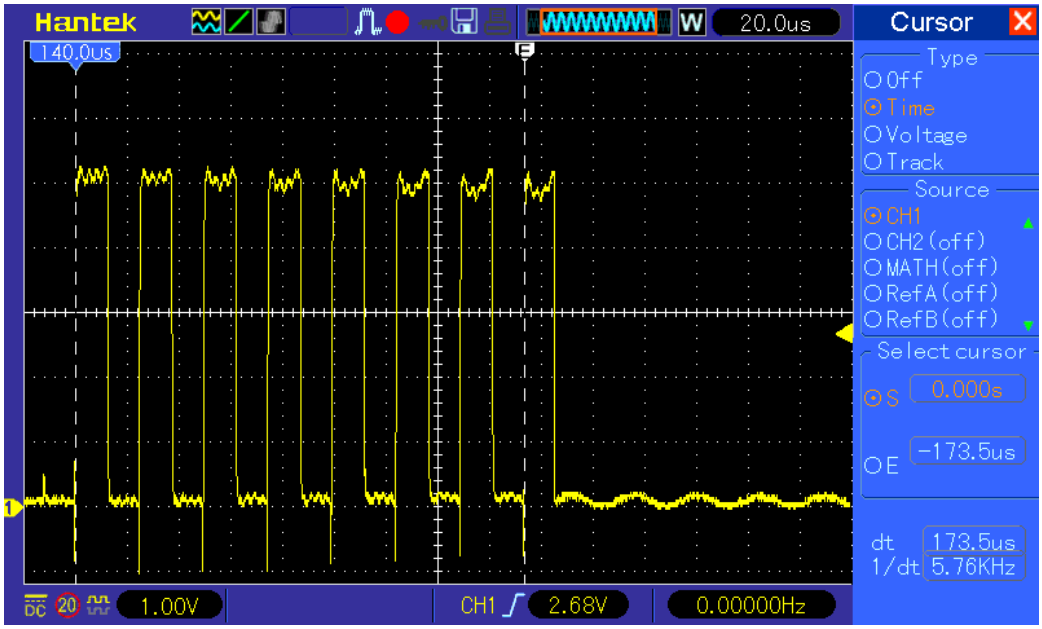


FIGURA 2: salva transmitida por el transmisor de ultrasonidos del sensor *HC-SR04*.

El sensor espera entonces a recibir el eco proveniente del objetivo. La señal de eco —que puede verse en la figura 3, junto con la salva—, es procesada por el sensor *HC-SR04* para obtener el retardo Δt.

Todo el procesado necesario para, a partir de esas señales, obtener el retardo Δt 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.

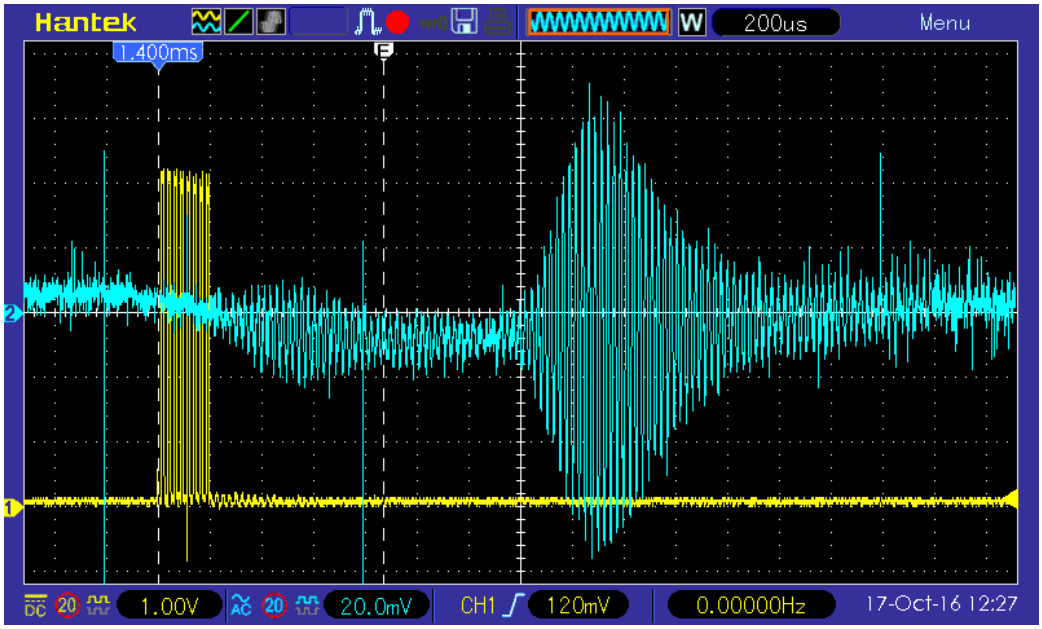


FIGURA 3: salva transmitida por el transmisor de ultrasonidos (en amarillo) y eco (en azul) recibido por el receptor de ultrasonidos del sensor *HC-SR04*. La distancia al objetivo era de unos 24 cm.

4.2.3. INTERFAZ ELÉCTRICA Y MONTAJE

La interfaz eléctrica del sensor telemétrico ultrasónico *HC-SR04* está formada, tal como se ve en la figura 1, 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 Δt. 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 el microcontrolador del laboratorio trabaja con una tensión de alimentación de 3.3 V, los pulsos

generados por él 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 Δt, entrega por esta salida un pulso a nivel alto cuya duración es igual a Δt. A través de este valor de Δt, mediante la expresión (1), se puede determinar la distancia al objetivo d. La salida *ECHO* trabaja con niveles TTL, pero el microcontrolador del laboratorio tolera, e interpreta correctamente, dichos niveles TTL aunque se alimente a 3.3 V, tal como se describe en la *datasheet* y el *user manual* del mismo.

En la figura 4 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 (1).

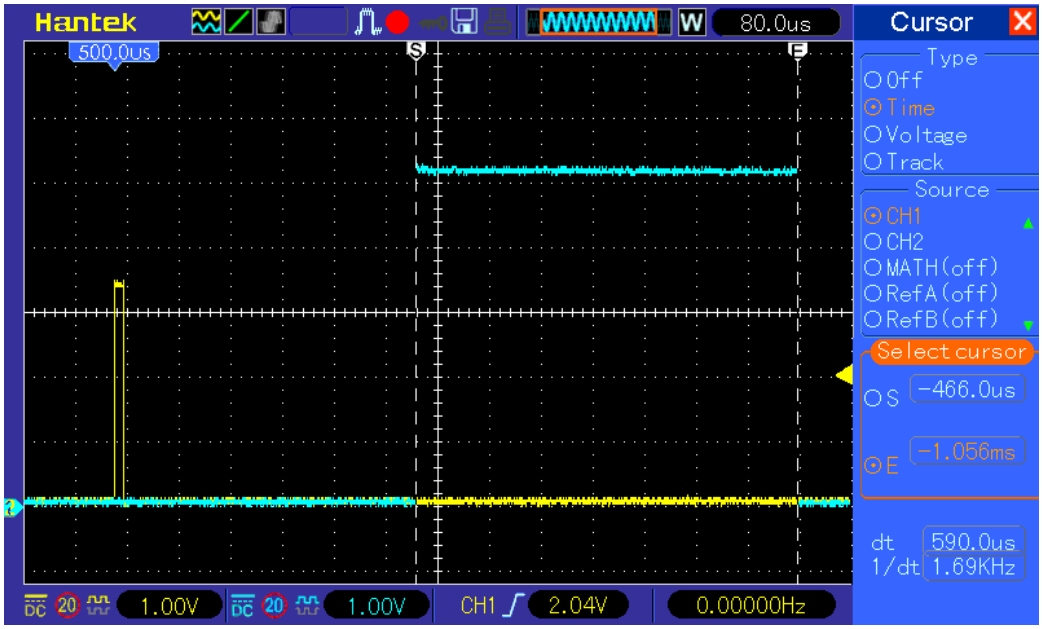
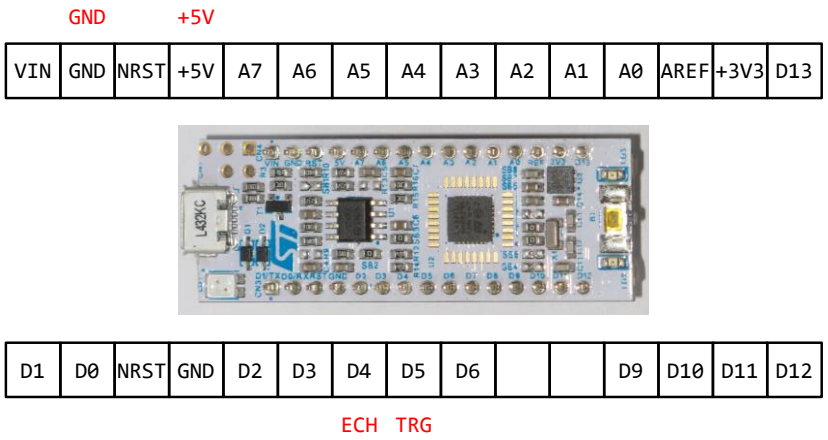


FIGURA 4: señales *TRIG* (en amarillo) y *ECHO* (azul) que ilustran el funcionamiento de la interfaz del sensor *HC-SR04*. El objetivo se encontraba a unos 10 cm de distancia del sensor.

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 si se conectase el pin *ECHO* del sensor (una salida) a otro pin de salida del microcontrolador podría ocasionarse 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.

El sensor se conectará al microcontrolador 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 a continuación:



4.2.4. 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\S1\E0). 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 1) la información mostrada en el *display*.

4.3. Ejercicio 1 - control del sensor (eventos independientes)

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.

En este ejercicio deberá emplear la técnica de gestión de eventos independientes (no FSM). Se recomienda utilizar:

- un `Ticker` para poner la señal *trigger* a nivel alto 10 veces por segundo;
- un `Timeout` para bajar a cero *trigger* 1 ms después;
- un objeto `InterruptIn` para detectar los flancos de subida y bajada en *echo*;
- un objeto `Timer` para medir la anchura del pulso en *echo*;
- un segundo `Ticker` se empleará para la multiplexación del *display*.

Trabaje sobre la carpeta MICR\P3\S1\E1, copiando en ella algún proyecto de ejercicios anteriores, el que considere más adecuado. Verifique el funcionamiento final sobre la placa.

4.4. Ejercicio 2 - control del sensor (FSM)

Ahora deberá resolver el mismo problema que en el apartado anterior, pero haciendo uso de una máquina de estados finitos. Encontrará en la carpeta MICR\P3\S1\E2 un proyecto de *Keil µVision 5* que deberá usar como base. En este proyecto falta por completar código en los ficheros main.cpp y range_finder.cpp (busque los comentarios que enmarcan las zonas que debe completar, hay una en cada uno de los ficheros indicados). También es necesario que añada su propia función `to_7seg()` (en el fichero to_7seg.cpp). El diagrama de estados que debe implementar para este autómata (en el fichero range_finder.cpp) está dado en la figura 5.

Aparte de los fragmentos de código indicados, no debe modificar ninguna otra parte del código.

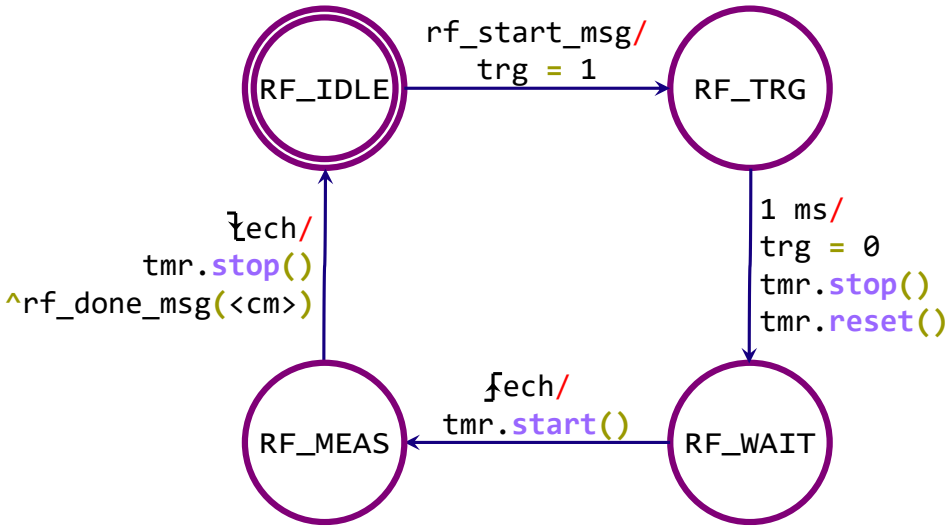


FIGURA 5: diagrama de estados inicial de la FSM del sensor *HC-SR04*.

Lea, como ayuda, los comentarios de los ficheros main.cpp, rangefinder.h y range_finder.cpp.

Verifique el funcionamiento final sobre la placa. Pruebe además a desconectar el sensor durante el funcionamiento (simulando una mala conexión o un fallo del sensor) y compruebe si el sistema es capaz de recuperarse al restablecerse la conexión.

4.1. Ejercicio 3 - control avanzado del sensor

Modifique el diagrama de estados de la figura 5 para que, además, si el sensor se desconecta o no responde, el parámetro del mensaje gb_rf_done_msg tome el valor −1. Para detectar tal condición espere un máximo de 50 ms a los flancos de subida y bajada de la señal ECH (estados RF_WAIT y RF_MEAS), si pasado ese intervalo de tiempo no se recibe el correspondiente flanco, se asume que el sensor está desconectado o falla, volviéndose al inicio e informando del error tal como se ha indicado.

Posteriormente (y trabajando sobre la carpeta MICR\P3\S1\E3) incorpore estos cambios al fichero range_finder.cpp. Modifique además el

fichero main.cpp para que, cuando se reciba el mensaje `gb_rf_done_msg` con parámetro `-1`, se muestre en el *display* el mensaje «Er».

Verifique el funcionamiento final sobre la placa, no olvide desconectar y volver a conectar el sensor en algún momento para comprobar si la detección del error funciona adecuadamente y si el sistema se recupera del fallo.

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 4 – control de un pulsador sin rebotes (FSM)

En este ejercicio deberá escribir un programa capaz de gestionar correctamente el pulsador central por medio de una máquina de estados finitos. Encontrará en la carpeta MICR\P3\S2\E4 un proyecto de *Keil µVision 5* que deberá usar como base. En este proyecto falta por completar código en los ficheros main.cpp y switch.cpp (busque los comentarios que enmarcan las zonas que debe completar, hay varias en cada uno de los ficheros indicados). También es necesario que añada su propia función `to_7seg()` (en el fichero to_7seg.cpp) y su implementación de la máquina de estados finitos del sensor telemétrico realizada en la sesión anterior (en el fichero range_finder.cpp).

En primer lugar, deberá completar el fichero switch.cpp (busque los comentarios que enmarcan las zonas que debe completar) para implementar una máquina de estados finitos que responda al diagrama de estados de la figura 6. Este autómata envía el mensaje `swm_long_msg` si detecta una pulsación de duración igual o mayor a 1 s en el pulsador central, y el mensaje `swm_msg` si la pulsación es más breve, eliminando en ambos casos el efecto de los rebotes. Estos mensajes son enviados cuando se suelta el pulsador.

En segundo lugar, deberá completar el fichero main.cpp para que:

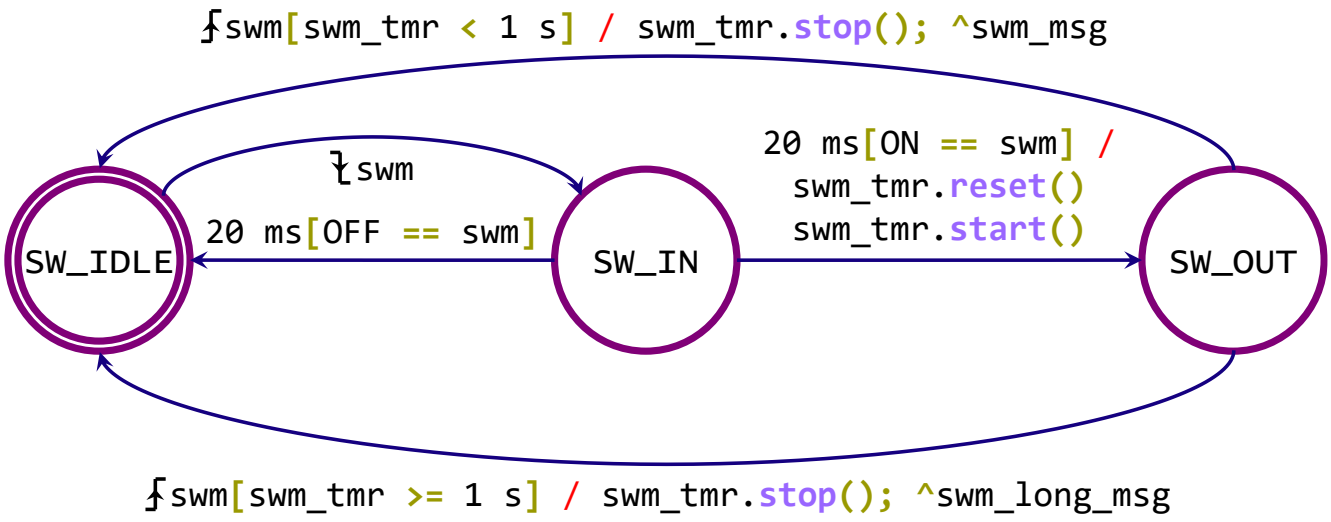


FIGURA 6: 1ª versión del diagrama de estados de la FSM del pulsador.

1. se incluyan las funcionalidades de la máquina de estados finitos del pulsador central, primero inicializándola y después ejecutándola en cada iteración del bucle infinito;
2. se modifique el código para que, con cada pulsación corta, se haga una única medida de distancia utilizando la máquina de estados finitos que se implementó en el ejercicio anterior;
3. se modifique el código para que, con cada pulsación larga, se inicie o se termine una secuencia de medidas automáticas de distancia. Para conseguirlo se utilizará una variable booleana que indicará si está en curso una secuencia de medidas automáticas («en curso» o «no en curso»). Cuando el estado cambie a «en curso», se iniciará el objeto `Ticker` empleado en el ejercicio anterior para conseguir la medida automática cada 100 ms y las pulsaciones cortas no afectarán al sistema. Cuando la variable cambie a «no en curso» se dejarán de hacer medidas automáticas cada 100 ms, únicamente respondiendo el sistema a las pulsaciones cortas para hacer la medida de distancia.

Verifique el funcionamiento final sobre la placa, comprobando que la aplicación mide distancias de forma automática o manual, dependiendo de las acciones sobre el pulsador central. En este ejercicio debiera hacer

un uso adecuado de los recursos de depuración que brinda la herramienta *Keil µVision 5*. En particular puede depurar la parte del control del pulsador sin necesidad de añadir la parte de medidas automáticas. Empleando *breakpoints* puede comprobar si la FSM del pulsador genera los mensajes adecuados en respuesta a pulsaciones de diversas duraciones.

5.1. Ejercicio 5 – 2ª versión del control del pulsador

En este ejercicio deberá modificar el diagrama de estados de la FSM del pulsador, de modo que ahora el mensaje `swm_long_msg` no se genere al terminar la pulsación, sino cuando el pulsador lleve pulsado 1 s. Al realizar esta modificación notará que ya no será necesario el objeto de la clase `Timer` que antes se empleaba para medir el tiempo. Trabaje sobre la carpeta MICR\P3\S2\E5, copiando sobre ella el código del ejercicio anterior. Solamente debe modificar el código del fichero switch.cpp.

Verifique el funcionamiento final sobre la placa, comprobando que ahora las pulsaciones largas producen su efecto no al soltar el botón, sino cuando transcurre 1 s desde el inicio de la pulsación.

Esta versión del control del pulsador será la que se use en el resto de esta práctica.

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

6. TRABAJOS PREVIOS A LA TERCERA SESIÓN PRESENCIAL

6.1. Ejercicio 6 – control de la multiplexación (FSM)

En este ejercicio deberá escribir un programa capaz de gestionar la multiplexación de los *displays* de 7 segmentos por medio de una máquina de estados finitos. Encontrará en la carpeta MICR\P3\S3\E6 un único fichero display.h que deberá utilizar para complementar el proyecto de

Keil desarrollado en la sesión anterior. Este fichero contiene los prototipos de las funciones y las declaraciones de los mensajes utilizados por la máquina de estados finitos del control de la multiplexación, siendo necesaria la creación de un fichero display.cpp que contenga la implementación de dicha máquina.

En primer lugar, deberá generar el fichero display.cpp (acorde al fichero display.h suministrado) y completarlo en su totalidad para implementar una máquina de estados finitos que cumpla lo siguiente:

1. el *display* deben pasar a encendido por medio del mensaje `gb_display_on_msg` y a apagado por medio del mensaje `gb_display_off_msg`;
2. mientras estén encendidos, será posible variar tanto el valor que representan como su brillo;
3. para variar el valor que representan se utilizará el mensaje `gb_display_update_msg` (que informa de que hay una variación en el valor representado) junto con su parámetro `g_display_segs` (que contiene el valor al cual cambian los *displays*, este valor se codifica como una variable de 16 bits en la cual los 7 LSB de su *byte* más significativo contienen los segmentos a encender en el *display* izquierdo y los 7 LSB de su *byte* menos significativo contienen los segmentos a encender en el *display* derecho);
4. para variar el brillo se utilizará el mensaje `gb_display_brigthness_msg` (que informa sobre una variación en el brillo deseado) junto con su parámetro `g_display_brigthness` (que contiene el valor de brillo al cual cambian los *displays*, codificándose este valor como un porcentaje de brillo desde 0 —casi apagados— hasta 99 —completamente encendidos—);
5. los parámetros `g_display_segs` y `g_display_brigthness` son también leídos y tenidos en cuenta al procesar el mensaje `gb_display_on_msg`;
6. se tomarán las medidas necesarias para asegurar la inexistencia de sombras;
7. cuando (por lo que respecta a este autómata) sea posible dormir al procesador, se deberá activar el *booleano* `gb_display_can_sleep`.

Tal y como se ha especificado en el punto 3 de esta descripción (variable `g_display_segs`), deberá trabajar con variables en la que de-

terminados grupos de bits tienen un cierto significado. Para poder leer o modificar algunos bits de una variable (en C/C++) le serán de utilidad:

- los operadores de desplazamiento `>>` y `<<`, que permiten desplazar un operando entero un cierto número de bits a derecha o izquierda, respectivamente;
- los operadores `&` y `|` bit a bit, que realizan las operaciones lógicas *and* y *or*, respectivamente, bit a bit, de un par de operandos enteros.

Con estos operadores podrá separar una variable de 16 bits en sus correspondientes *bytes* alto y bajo y también realizar la operación inversa, a partir de dos *bytes* formar una palabra de 16 bits. El siguiente código ilustra una posible forma de realizar estas operaciones, aunque en este caso se trata de un dato de 32 bits que desea separarse en, o formarse a partir de, dos «medios datos», alto y bajo, de 16 bits cada uno.

```
uint32_t  high_low = 0x0123CDEFU;
uint16_t  high;
uint16_t  low;

// separar high_low (32 bit) en dos partes de 16 bit cada una
high = high_low >> 16;          // high = 0x0123U
low = high_low & 0xFFFFU;       // low = 0xCDEFU

// juntar dos partes de 16 bit cada una en high_low (32 bit)
high = 0x4567U;
low = 0x89ABU;
high_low = (high << 16) | low;  // high_low = 0x456789ABU
```

En segundo lugar, deberá modificar el fichero main.cpp del que dispone para que:

1. se incluyan las funcionalidades de la máquina de estados finitos de la multiplexación, primero inicializándola, y después ejecutándola en cada iteración del *super-loop*;
2. se modifique el código para que la multiplexación se realice por medio de la máquina de estados finitos recién construida y no por medio de la gestión de eventos anterior, pero manteniendo la misma funcionalidad;

3. el brillo de los *displays*, hasta ahora constante, deberá ser proporcional a la distancia que se muestra, siendo mínimo cuando el sensor mida objetos cercanos y máximo cuando mida objetos lejanos (si la distancia es mayor a 99 cm o se detecta un error en el sensor, el brillo debe ser máximo);
4. se actualizará el código correspondiente a la gestión del sueño.

Dibuje el diagrama de estados de esta FSM y, posteriormente, impleméntela en C++.

Verifique el funcionamiento final sobre la placa, comprobando que la aplicación cumple con la misma funcionalidad que en la sesión anterior además de que ahora el brillo en los *displays* depende de la distancia medida.

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

7. Trabajos previos a la cuarta sesión presencial

7.1. Ejercicio 7 – diseño final (FSM)

En este ejercicio deberá escribir un programa capaz de gestionar una máquina de estados finitos que hará uso de las FSM implementadas en las sesiones anteriores. Encontrará en la carpeta MICR\P3\S4\E7 una carpeta vacía donde deberá incluir la cabecera e implementación de las máquinas de estados finitos del sensor telemétrico, el pulsador central y el *display* (no debe modificarlas). Para ello, utilice como base el proyecto de la sesión anterior y añada unos archivos control.h y control.cpp, donde desarrollará la máquina de estados finitos de este diseño final. Tenga en cuenta que el fichero main.cpp contiene la implementación de la anterior sesión, por lo que tendrá también que modificarlo para que la funcionalidad sea la del diseño final.

La máquina de estados finitos del diseño final debe cumplir con la siguiente funcionalidad:

1. el sistema debe comenzar con el *display* apagado, pasando a encendido por medio de una pulsación larga (mayor a un segundo) en el pulsador central;
2. ahora, sucesivas pulsaciones cortas irán mostrando diferentes posibles modos de funcionamiento en el *display*, estos modos son «Li» (que debe ser el que aparezca tras el encendido), «di», «LE» y «OF», activándose el funcionamiento asociado a cada uno de estos modos mediante una pulsación larga;
3. una vez activado cualquier modo, se podrá salir de él (volviendo al mismo ítem del menú de selección de modos del punto 2) mediante una nueva pulsación larga;
4. las pulsaciones cortas no tendrán efecto alguno dentro de cualquier modo (excepto por lo dicho en el punto 10 más adelante);
5. al activar el modo «Li» se empezará una secuencia automática de medidas de luz por medio de la fotorresistencia cada 100 ms, representando el valor porcentual capturado (desde 0 hasta 99) en el *display*;
6. al activar el modo «di» se empezará una secuencia automática de medidas de distancia (en cm) por medio del sensor telemétrico cada 100 ms, representando el valor capturado (desde 0 hasta 99, más «– –» y «Er», como en el ejercicio anterior) en el *display*;
7. al activar el modo «LE» el LED central parpadeará a una frecuencia de 10 Hz, permaneciendo encendido únicamente los primeros 50 ms del ciclo y en el *display* se debe mostrar «– –»;
8. al activar el modo «OF» el sistema se apagará, volviendo al punto 1;
9. en cualquier momento, siempre que el *display* esté encendido, el sistema debe encender durante 50 ms el LED izquierdo al reconocer una pulsación larga;
10. en cualquier momento, siempre que el *display* esté encendido, el sistema debe encender durante 50 ms el LED derecho al reconocer una pulsación corta;
11. cuando se muestre en el *display* un valor numérico, su brillo dependerá de ese valor, de modo que si es 0, el brillo será mínimo (pero no apagado) y cuando el valor indicado sea máximo (99), el brillo también lo será;
12. si en el *display* se muestra cualquier mensaje distinto a un valor numérico, el brillo será máximo;

13. informará a main.cpp, mediante el correspondiente *booleano*, de cuando esté en disposición de dormir al procesador.

Además de todo lo anterior, el sistema informará por el puerto serie (230400-8N2) de una serie de circunstancias:

1. cuando se active alguno de los modos «Li», «di» o «LE», se enviará por el puerto el mensaje «=> Li», «=> di» o «=> LE» respectivamente;
2. cuando se salga de alguno de los modos «Li», «di» o «LE», se enviará por el puerto el mensaje «<= Li», «<= di» o «<= LE» respectivamente;
3. cuando en los modos «Li» o «di» se realice una medida, se enviará el contenido del *display* precedido de 4 espacios;
4. cuando se encienda el sistema se enviará el mensaje «On»;
5. cuando se apague el sistema se enviará el mensaje «Off».

Encontrará en la carpeta MICR\P3\S4\E7 un fichero .bin que realiza toda esta funcionalidad y que puede serle de utilidad para entender el funcionamiento esperado del sistema.

Como ya se ha dicho, el código correspondiente a los anteriores apartados (FSM de sensor telemétrico, pulsador y *display*) no debe modificarse en forma alguna. Solo debe modificar los ficheros: control.h y control.cpp (que deben incluir la funcionalidad descrita en este apartado); main.cpp (para inicializar y llamar a la nueva FSM de control); y también hardware.h y hardware.cpp (para crear e inicializar el puerto serie).

Una vez terminada la codificación de la aplicación completa, verifique el funcionamiento final sobre la placa comprobando que se satisfacen todos los puntos de la funcionalidad recién descrita.

8. Subida de resultados a Moodle

Elimine todas las carpetas de nombre `~build` y `~listings` de las carpetas de los ejercicios E1 a E7. Elimine igualmente todos los ficheros con extensión .bin que se entregaron con la práctica. Comprima entonces la carpeta P3 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 antes de la cuarta sesión presencial de esta práctica.