MICROPROCESADORES

PRÁCTICA 3

AUTÓMATAS CONTROLADOS POR EVENTOS

DISEÑO FINAL

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

Debera 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 mostrara el mensaje «– –». La medida de distancia se realizara 10 veces por segundo y con una anchura en el pulso de *trigger* de 1 ms.

En este ejercicio debera emplear la tecnica de gestion de eventos independientes (no FSM). Se recomienda utilizar:

- un `Ticker` para poner la senal *trigger* a nivel alto 10 veces por segundo;
- un `Timeout` para bajar a cero *trigger* 1 ms despues;
- 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 empleara para la multiplexacion del *display*.

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

3.2. Ejercicio 2 - control del sensor (FSM)

Ahora debera resolver el mismo problema que en el apartado anterior, pero haciendo uso de una maquina de estados finitos. Encontrara en la carpeta MICR\P3\S1\E2 un proyecto de *Keil µVision 5* que debera usar como base. En este proyecto falta por completar codigo 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). Tambien es necesario que anada su propia funcion `to_7seg()` (en el fichero to_7seg.cpp). El diagrama de estados que debe implementar para este automata (en el fichero range_finder.cpp) esta dado en la figura 5.

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

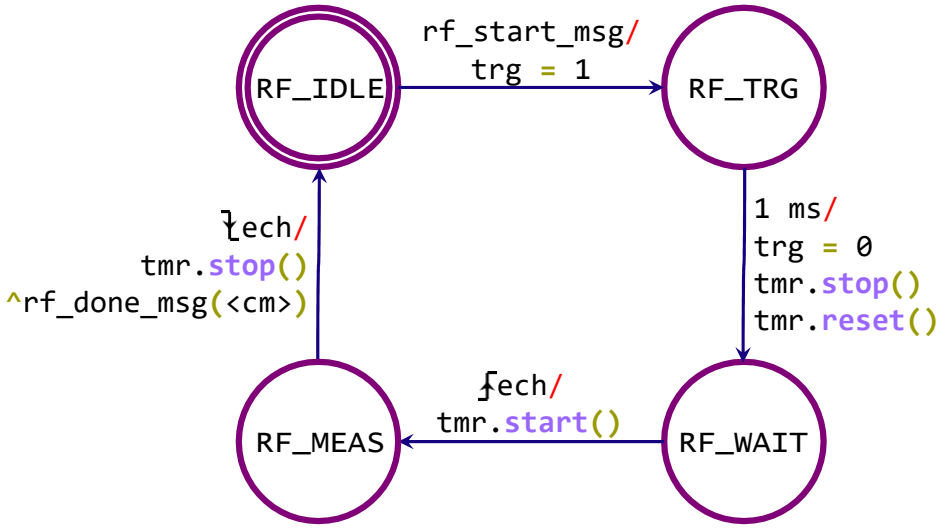

FIGURA 5: diagrama de estados inicial de la FSM del sensor *HC-SR04*.

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

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

4.1. Ejercicio 3 - control avanzado del sensor

Modifique el diagrama de estados de la figura 5 para que, ademas, si el sensor se desconecta o no responde, el parametro del mensaje gb_rf_done_msg tome el valor –1. Para detectar tal condicion espere un maximo de 50 ms a los flancos de subida y bajada de la senal ECH (estados RF_WAIT y RF_MEAS), si pasado ese intervalo de tiempo no se recibe el correspondiente flanco, se asume que el sensor esta desconectado o falla, volviendose 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 ademas el

fichero main.cpp para que, cuando se reciba el mensaje `gb_rf_done_msg` con parametro –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 algun momento para comprobar si la deteccion del error funciona adecuadamente y si el sistema se recupera del fallo.

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

4. Trabajos previos a la segunda sesión presencial

4.1. Ejercicio 4 – control de un pulsador sin rebotes (FSM)

En este ejercicio debera escribir un programa capaz de gestionar correctamente el pulsador central por medio de una maquina de estados finitos. Encontrara en la carpeta MICR\P3\S2\E4 un proyecto de *Keil µVision 5* que debera usar como base. En este proyecto falta por completar codigo 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). Tambien es necesario que anada su propia funcion `to_7seg()` (en el fichero to_7seg.cpp) y su implementacion de la maquina de estados finitos del sensor telemetrico realizada en la sesion anterior (en el fichero range_finder.cpp).

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

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

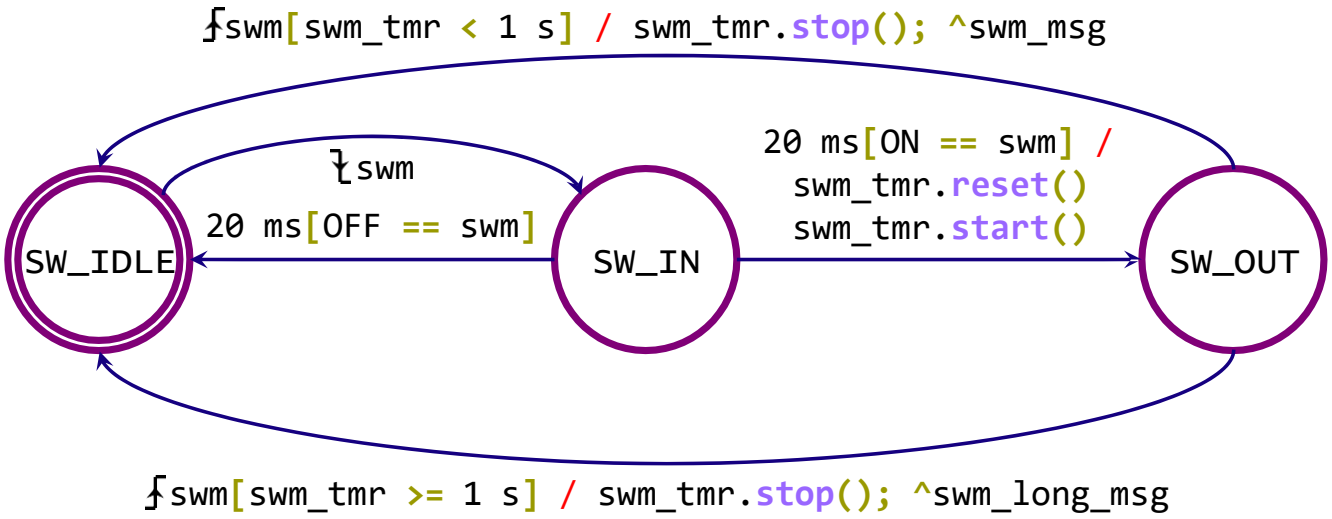

FIGURA 6: 1ª version del diagrama de estados de la FSM del pulsador.

1. se incluyan las funcionalidades de la maquina de estados finitos del pulsador central, primero inicializandola y despues ejecutandola en cada iteracion del bucle infinito;
2. se modifique el codigo para que, con cada pulsacion corta, se haga una unica medida de distancia utilizando la maquina de estados finitos que se implemento en el ejercicio anterior;
3. se modifique el codigo para que, con cada pulsacion larga, se inicie o se termine una secuencia de medidas automaticas de distancia. Para conseguirlo se utilizara una variable booleana que indicara si esta en curso una secuencia de medidas automaticas («en curso» o «no en curso»). Cuando el estado cambie a «en curso», se iniciara el objeto `Ticker` empleado en el ejercicio anterior para conseguir la medida automatica cada 100 ms y las pulsaciones cortas no afectaran al sistema. Cuando la variable cambie a «no en curso» se dejaran de hacer medidas automaticas cada 100 ms, unicamente respondiendo el sistema a las pulsaciones cortas para hacer la medida de distancia.

Verifique el funcionamiento final sobre la placa, comprobando que la aplicacion mide distancias de forma automatica o manual, dependiendo de las acciones sobre el pulsador central.

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

En este ejercicio debera 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 pulsacion, sino cuando el pulsador lleve pulsado 1 s. Al realizar esta modificacion notara que ya no sera 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 codigo del ejercicio anterior. Solamente debe modificar el codigo del fichero switch.cpp.

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

Esta version del control del pulsador sera la que se use en el resto de esta practica.

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

5. Trabajos previos a la tercera sesión presencial

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

En este ejercicio debera escribir un programa capaz de gestionar la multiplexacion de los *displays* de 7 segmentos por medio de una maquina de estados finitos. Encontrara en la carpeta MICR\P3\S3\E6 un unico fichero display.h, que debera utilizar para complementar el proyecto de *Keil* desarrollado en la sesion anterior. Este fichero contiene los prototipos de las funciones y las declaraciones de los mensajes utilizados por la maquina de estados finitos del control de la multiplexacion, siendo necesaria la creacion de un fichero display.cpp que contenga la implementacion de dicha maquina.

En primer lugar, debera generar el fichero display.cpp (acorde al fichero display.h suministrado) y completarlo en su totalidad para implementar una maquina 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 esten encendidos, sera posible variar tanto el valor que representan como su brillo;
3. para variar el valor que representan se utilizara el mensaje `gb_display_update_msg` (que informa de que hay una variacion en el valor representado) junto con su parametro `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* mas 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 utilizara el mensaje `gb_display_brigthness_msg` (que informa sobre una variacion en el brillo deseado) junto con su parametro `g_display_brigthness` (que contiene el valor de brillo al cual cambian los *displays*, codificandose este valor como un porcentaje de brillo desde 0 —casi apagados— hasta 99 —completamente encendidos—);
5. los parametros `g_display_segs` y `g_display_brigthness` son tambien leídos y tenidos en cuenta al procesar el mensaje `gb_display_on_msg`;
6. se tomaran las medidas necesarias para asegurar la inexistencia de sombras;
7. cuando (por lo que respecta a este automata) sea posible dormir al procesador, se debera enviar el mensaje `gb_display_can_sleep`.

Tal y como se ha especificado en el punto 3 de esta descripcion (variable `g_display_segs`), debera trabajar con variables en la que determinados grupos de bits tienen un cierto significado. Para poder leer o modificar algunos bits de una variable (en C/C++) le seran de utilidad:

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

Con estos operadores podra separar una variable de 16 bits en sus correspondientes *bytes* alto y bajo y tambien realizar la operacion inversa, a partir de dos *bytes* formar una palabra de 16 bits. El siguiente codigo 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, debera modificar el fichero main.cpp del que dispone para que:

1. se incluyan las funcionalidades de la maquina de estados finitos de la multiplexacion, primero inicializandola, y despues ejecutandola en cada iteracion del bucle infinito;
2. se modifique el codigo para que la multiplexacion se realice por medio de la maquina de estados finita recien construida y no por medio de la gestion de eventos anterior, pero manteniendo la misma funcionalidad;
3. el brillo de los *displays*, hasta ahora constante, debera ser proporcional a la distancia que se muestra, siendo mí´nimo cuando el sensor mida objetos cercanos y maximo cuando mida objetos lejanos (si la distancia es mayor a 99 cm o se detecta un error en el sensor, el brillo debe ser maximo);
4. se actualizara el codigo correspondiente a la gestion del sueno.

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

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

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

7. TRABAJOS PREVIOS A LA CUARTA SESIÓN PRESENCIAL

7.1. Ejercicio 7 – diseño final (FSM)

En este ejercicio debera escribir un programa capaz de gestionar una maquina de estados finitos que hara uso de las FSM implementadas en las sesiones anteriores. Encontrara en la carpeta MICR\P3\S3\E7 una carpeta vacía donde debera incluir la cabecera e implementacion de las maquinas de estados finitos del sensor telemetrico, el pulsador central y el *display*. Para ello, utilice como base el proyecto de la sesion anterior y anada unos archivos control.h y control.cpp, donde desarrollara la maquina de estados finitos del diseno final. Tenga en cuenta que el fichero main.cpp contiene la implementacion de la anterior sesion, por lo que tendra tambien que modificarlo para que la funcionalidad sea la del diseno final.

La maquina de estados finitos del diseno final debe cumplir con la siguiente funcionalidad:

1. el sistema debe comenzar con el *display* apagado, pasando a encendido por medio de una pulsacion larga (mayor a un segundo) en el pulsador central;
2. ahora, sucesivas pulsaciones cortas iran 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», activandose el funcionamiento asociado a cada uno de estos modos mediante una pulsacion larga;
3. una vez activado cualquier modo, se podra salir de el (volviendo al mismo ítem del menu de seleccion de modos del punto 2) mediante una nueva pulsacion larga;

4. las pulsaciones cortas no tendran efecto alguno dentro de cualquier modo (excepto por lo dicho en el punto 10 mas adelante);
5. al activar el modo «Li» se empezara una secuencia automatica 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 empezara una secuencia automatica de medidas de distancia (en cm) por medio del sensor telemetrico cada 100 ms, representando el valor capturado (desde 0 hasta 99, mas «– –» y «Er», como en el ejercicio anterior) en el *display*;
7. al activar el modo «LE» el LED central parpadeara´ a una frecuencia de 10 Hz, permaneciendo encendido unicamente los primeros 50 ms del ciclo y en el *display* se debe mostrar «– –»;
8. al activar el modo «OF» el sistema se apagara, volviendo al punto 1;
9. en cualquier momento, siempre que el *display* este encendido, el sistema debe encender durante 50 ms el LED izquierdo al reconocer una pulsacion larga;
10. en cualquier momento, siempre que el *display* este encendido, el sistema debe encender durante 50 ms el LED derecho al reconocer una pulsacion corta;
11. cuando se muestre en el *display* un valor numerico, su brillo dependera de ese valor, de modo que si es 0, el brillo sera mí´nimo (pero no apagado) y cuando el valor indicado sea maximo (99), el brillo tambien lo sera;
12. si en el *display* se muestra cualquier mensaje distinto a un valor numerico, el brillo sera maximo;
13. informara a main.cpp, mediante el correspondiente mensaje, de cuando este en disposicion de dormir al procesador.

Encontrara 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.

Una vez terminada la codificacion de la aplicacion completa, verifique el funcionamiento final sobre la placa comprobando que se satisfacen todos los puntos de la funcionalidad recien descrita.

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