Práctica 3
Arquitecturas de Firmware

1. Máquina de estado - Diseño

Una supuesta máquina expendedora de tres productos normalmente está a la espera de que alguien acerque una tarjeta de la cual lee los datos para cobrar. Cuando alguien acerca la tarjeta espera que el usuario presione uno de los tres botones para seleccionar el producto dentro de un tiempo máximo de 30 segundos. Si el usuario no presiona ningún botón no se acredita el pago y la máquina vuelve a esperar la tarjeta. Por el contrario, si el usuario presiona un botón de la máquina dentro de esos 30 segundos se activará uno de los tres motores para expender el producto seleccionado, durante un tiempo de 10 segundos, luego ejecutará el cobro del producto y volverá a esperar una tarjeta.

Diseñe una máquina de estados para la expendedora considerando como entradas (eventos disparadores de transiciones):

- que el usuario acerque una tarjeta (se guarda en una estructura los datos)
- que se presione algún botón (se guarda en una variable el botón presionado)
- que haya expirado un tiempo de espera

Como salidas (acción que se ejecuta al transicionar de un estado al otro) considere:

- encendido/apagado de algún motor
- ejecución del cobro
- impresión en consola de cada transición que ocurra

2. Máquina de estado - Implementación

Implementar en el microcontrolador la FSM del Ej. 1, utilizando 4 pulsadores (3 para los botones de productos y uno para simular la tarjeta) y 3 leds para visualizar las salidas de los motores.

3. Superloop y UART bloqueante

Implemente un programa que en un bucle infinito lea un caracter numérico de la UART (bloqueando indefinidamente) utilizando la función HAL_UART_Receive() y Configure la comunicación en 9600-8-N-1.

Si es un carácter válido, haga destellar el LED la cantidad de veces indicada (pulsos de 50ms separados por 50ms). Implemente las esperas con HAL_Delay().

Cada vez que se prenda el LED transmita por la UART utilizando HAL_UART_Transmit():

```
“******************** ON ********************\r\n”
“******************** ON ********************\r\n”
“******************** ON ********************\r\n”
“******************** ON ********************\r\n”
```

y cuando se apague envíe:

```
“******************** OFF ********************\r\n”
“******************** OFF ********************\r\n”
“******************** OFF ********************\r\n”
“******************** OFF ********************\r\n”
```

Realice un diagrama de tiempos y verifique usando el analizador lógico.

5. Máquina de estados del cronómetro

A partir de la formalización vista para representar las FSM repita el ejercicio anterior, verificando si la implementación de las FSM es correcta o no. Analice si es posible lograr una implementación única de la FSM que sea compatible con las tres arquitecturas vistas.

Ejercicios de años anteriores:

Ejercicio 1

Crear el firmware de un controlador para un portón automático (como los de las cocheras) junto con un “simulador” de portón para poder probarlo. Es decir que se asemeja a realizar dos proyectos distintos, pero en el mismo microcontrolador, donde implementará dos módulos, cada uno controlado por una máquina de estados.

El sistema se compone de:

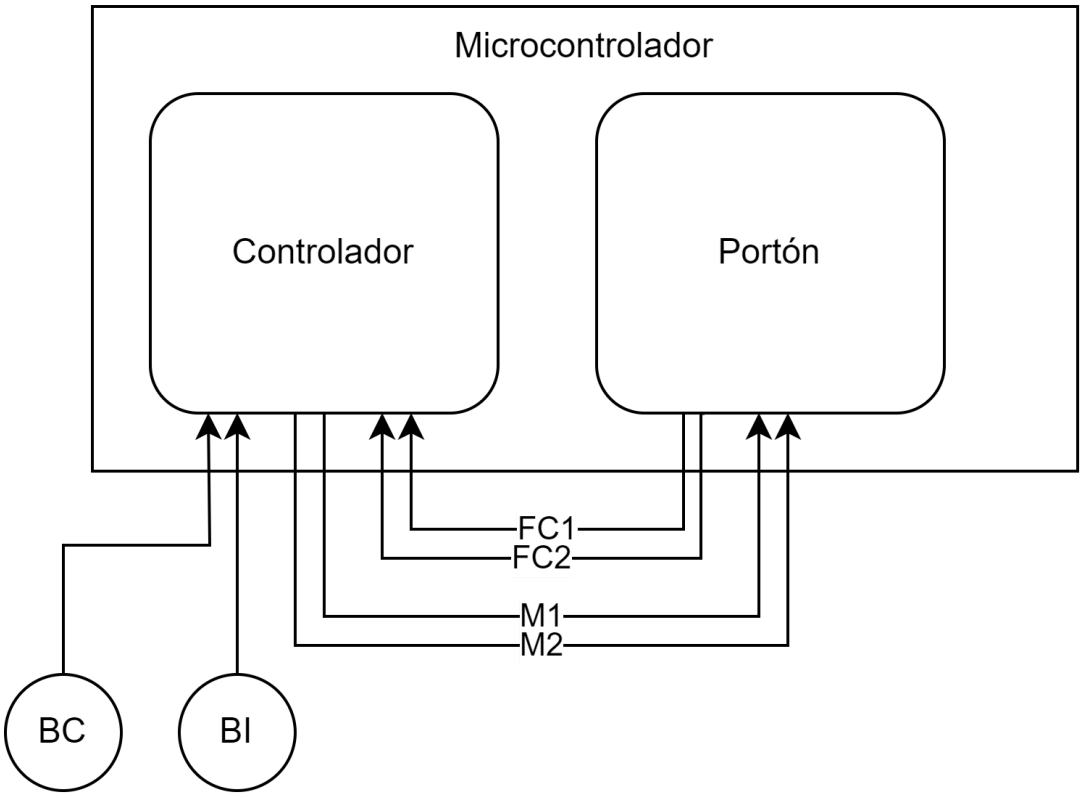
- Un módulo controlador, que a su vez posee cuatro entradas y dos salidas:
  - Un botón único para comandarlo (BC)
  - Un sensor de fin de carrera para la apertura (FC1)
  - Otro sensor de fin de carrera para el cierre (FC2)
  - Un haz infrarrojo que lo detiene si algún objeto lo intercepta (BI)
  - Dos salidas para mover el motor para un lado o para el otro (M1, M2). El motor se mueve para el lado correspondiente cuando se mantiene activa la línea (es decir, se activa por nivel, no por pulso).
- Un segundo módulo de portón, que posee dos entradas y dos salidas:
  - Dos entradas para emular el movimiento del motor para un lado o el otro (M1,M2)
  - Dos salidas que emulan la activación de los finales de carrera (FC1, FC2)

El funcionamiento de los módulos respeta los siguientes puntos:

- Para el controlador
  - Al encender el controlador, se miden los sensores de fin de carrera para determinar en qué posición está el portón. Si no se encuentra ninguno activado, al presionar el botón se cerrará. Si se encuentra alguno activado, el portón irá hasta el otro al pulsar el botón.

    - Se decidió que el movimiento se detiene tanto si se intercepta el haz durante la apertura como durante el cierre, y para continuar deberá dejar de interrumpirse el haz y volverse a presionar el botón de comando (y continuar en el sentido en que venía, por supuesto).
- Para el emulador de portón:
    - El portón tardará 4 segundos en ir desde un fin de carrera al otro (es decir, abrirse o cerrarse)
- Ambas máquinas de estados se ejecutan cada 10 ms

La interrelación de los módulos se representa en el siguiente diagrama, donde queda claro que no se vinculan en el código sino externamente, mediante conexiones entre sus entradas/salidas :



1. Implementar el módulo de portón en base a una máquina de estados utilizando pulsadores en M1, M2 y leds para FC1, FC2
   Considere, si lo desea, los estados:

   ABIERTO, CERRADO, ABRIENDO, CERRANDO, REPOSO_INTERMEDIO

2. Implementarlo como librería

3. Implementar módulo controlador en base a una máquina de estados conectado al módulo de portón

4. Implementarlo como librería

5. Describa el tipo de arquitectura/s de firmware utilizada/s (superloop, foreground/background, time-triggered) fundamentando.

6. Si no lo hizo, implementar la temporización o alguna de las entradas como interrupción, evitando problemas de concurrencia con las máquinas de estado y explicando el funcionamiento

Ejercicio 2

La estructura var_t representa una magnitud física utilizada por un sistema SCADA (control y monitoreo de datos) y permite almacenar un valor real, junto con un umbral alto de alarma, un umbral bajo de alarma, y una variable de estado que indica distintas condiciones asociadas a las alarmas

Una librería compuesta por por los archivo variable.h/.c permite crear variables de tipo var_t y manipularlas mediante una serie de funciones publicadas en el .h:

```
#ifndef _VARIABLE_H_
#define _VARIABLE_H_

typedef enum {VAR_NORMAL,
        VAR_HI,
        VAR_LO,
        VAR_WAITING_ACK} estado_t;

typedef struct{
        float valor;
        float umbral_alto;
        float umbral_bajo;
        estado_t estado;
}var_t;

/* inicializa el valor de la variable v pasada por referencia, los umbrales de alarma superior e inferior, e inicializa el estado interno*/
void var_init(var_t *v, float val_ini, float umbral_hi_ini, float umbral_lo_ini);

/* setea el valor de la variable v pasada por referencia*/
void var_set_val(var_t* v, float new_val);

/*reconoce la alarma y reinicia el estado de la variable (si la situación de alarma se extinguió)*/
void var_ack_alarm(var_t* v);
```

```
/*obtiene el estado interno de la variable*/
estado_t var_get_state(var_t* v);

#endif
```

.

El archivo variable.c contiene las definiciones de las funciones expuestas en el .h, junto con la definición de una función interna que ejecuta una FSM y una enumeración de eventos, que pueden ser VAR_VAL_EVT o VAR_ACK_EVT. Los estados son los definidos en el .h. La FSM cumple con las siguientes reglas:

a) Deberá dispararse un evento VAR_VAL_EVT cada vez que se cambia el valor principal de la variable con la función correspondiente. Si la variable está en estado VAR_NORMAL o VAR_WAITING_ACK se verificará si excede los límites pasando a los estados de alarma. Si el valor es menor al umbral bajo, la variable estará en estado VAR_LO, si es mayor al umbral alto estará en estado VAR_HI.
b) Si la variable está en alguno de los estados de alarma y vuelve a tener un valor en el rango normal pasará a VAR_WAITING_ACK.
c) Si se está en el estado VAR_WAITING_ACK y se dispara el evento VAR_ACK_EVT, se volverá al estado normal.
d) Si está en el estado de alarma VAR_LO (VAR_HI) y se actualiza el valor a uno mayor (menor) al umbral alto (bajo) se cambia al otro estado de alarma VAR_HI (VAR_LO).
e) En los estados de alarma se ignora el VAR_ACK_EVT.

1. Dibuje el esquema de la FSM mencionada, identificando correctamente cada elemento
2. implemente el archivo variable.c con los elementos mencionados anteriormente
3. Realice un programa de prueba en el cual llamen secuencialmente a las funciones correspondientes para hacer pasar la máquina por todos los estados, verificando el resultado correcto en una sesión de debug,
4. Implemente un sistema que utilice la librería creada y agregue la capacidad de imprimir por UART mensajes, setear valor de la variables con el ADC, y utilizar un pulsador para reconocer la alarma. Respete la siguiente funcionalidad con una estructura time triggered con tres tareas:
    - una tarea toma una muestra del ADC y setea el valor de la variable con la función correspondiente cada 100 ms,
    - otra, que se ejecuta cada 1 segundo, imprime mensajes por uart según el estado en el que esté la variable (dicho estado se obtiene con la función correspondiente):
        i. en estado normal: se verificará si el valor cambió en más de un 5 % respecto al anterior, solo en ese caso se imprimirá una línea con el valor de la variable

ii. en estado de alarma se imprimirá el valor actual y la leyenda “ALARMA ALTA” o “ALARMA BAJA” según corresponda
iii. en estado de espera del acknowledge se imprimirá el valor actual y la leyenda “RECONOCIMIENTO DE ALARMA PENDIENTE”

- otra lee el estado de un pulsador cada 20 ms, cuando se presiona envía el reconocimiento de alarma mediante la función correspondiente

Como ayuda proponemos el siguiente esquema:

```
var_t v;
 var_init(&v, 0, 3000, 1000);
…
 while (1)
 {
        if(HAL_GetTick()-lastTick1>=periodo1){
                lastTick1=HAL_GetTick();
               tarea1(&v);
        }
        if(HAL_GetTick()-lastTick2>=periodo2){
                lastTick2=HAL_GetTick();
                tarea2(&v);
        }
        if(HAL_GetTick()-lastTick3>=periodo3){
                lastTick3=HAL_GetTick();
                tarea3(&v);
        }
}
…
```

Ejercicio 3

Se desea crear una solución para una máquina expendedora que permite, con 2 pulsadores, seleccionar 1 de N opciones ciclando las opciones con la presión de un pulsador y seleccionando la opción actual con la presión del segundo pulsador. El ciclado de opciones se mostrará cambiando el led iluminado entre N=4. Cuando se selecciona la opción se activa el mecanismo expendedor lo cual se simulará haciendo titilar el led seleccionado durante 4 segundos a una tasa de 5 Hz. Implemente una máquina de estados para lograrlo, y córrala en un esquema event-driven o time-triggered (elija uno y aténgase a la estructura seleccionada). Diseñe la máquina de estados e implemente la solución.

Ejercicio 4

Cree un programa que permita medir el tiempo entre dos presiones de un pulsador y envíe el resultado por UART a la PC. Implemente una máquina de estados apropiada para evitar problemas por rebotes del pulsador considerando que el tiempo mínimo que se quiere medir es de 200 ms. El sistema tendrá un LED que titilará 1 vez por segundo mientras está esperando la presión del pulsador, y 5 veces por segundo cuando está contando el tiempo entre pulsaciones.