

Sistemas Electrónicos Para Automatización

PRÁCTICA 2: *Clasificador de frutas automático.*

27/10/2021

Francisco Javier Román Cortés 4º GIERM

Introducción

*****IMPORTANTE: RECORDAR QUE HAY QUE SALIR DEL MODO DEBUG ANTES DE PROBAR LOS PROGRAMAS PARA QUE NO DE FALLOS EL SYSCTLSLEEP()*****

En esta práctica se van a tratar varios aspectos de un problema concreto para tratar ciertos objetivos como son:

- Realizar un control mediante PWM con variación de parámetros
- Realizar un pequeño monitor por puerto serie
- Diseñar un sistema con todos los elementos de un SEPA

Para ello lo que se propone es diseñar una especie de clasificador de frutas, al que, por ejemplo, en función de cierto sensor, desplace cada tipo de fruta hacia una zona, llevando además un registro del tipo y cantidad de frutas de cada tipo que se ha ido detectando, así como el instante de tiempo concreto en que se detectó la fruta.

Material necesario

- Connected Launchpad de Texas Instruments con el microcontrolador TM4C1294NCPDT
- Manual de la librería DriverLib y datasheet del microcontrolador
- Placa TIVA-EVE-L293
- Servomotor de 5V
- Programa TeraTerm, para trabajar con el puerto serie

Fundamento teórico

Principalmente hay que conocer una serie de periféricos que se adicionan a los que ya se trataron en la primera práctica, en este caso se incidirá sobre los timers y sus interrupciones, los moduladores PWM y finalmente el puerto serie asíncrono (más conocido como UART). Además, se incidirá bastante sobre la gestión del modo de bajo consumo del sistema.

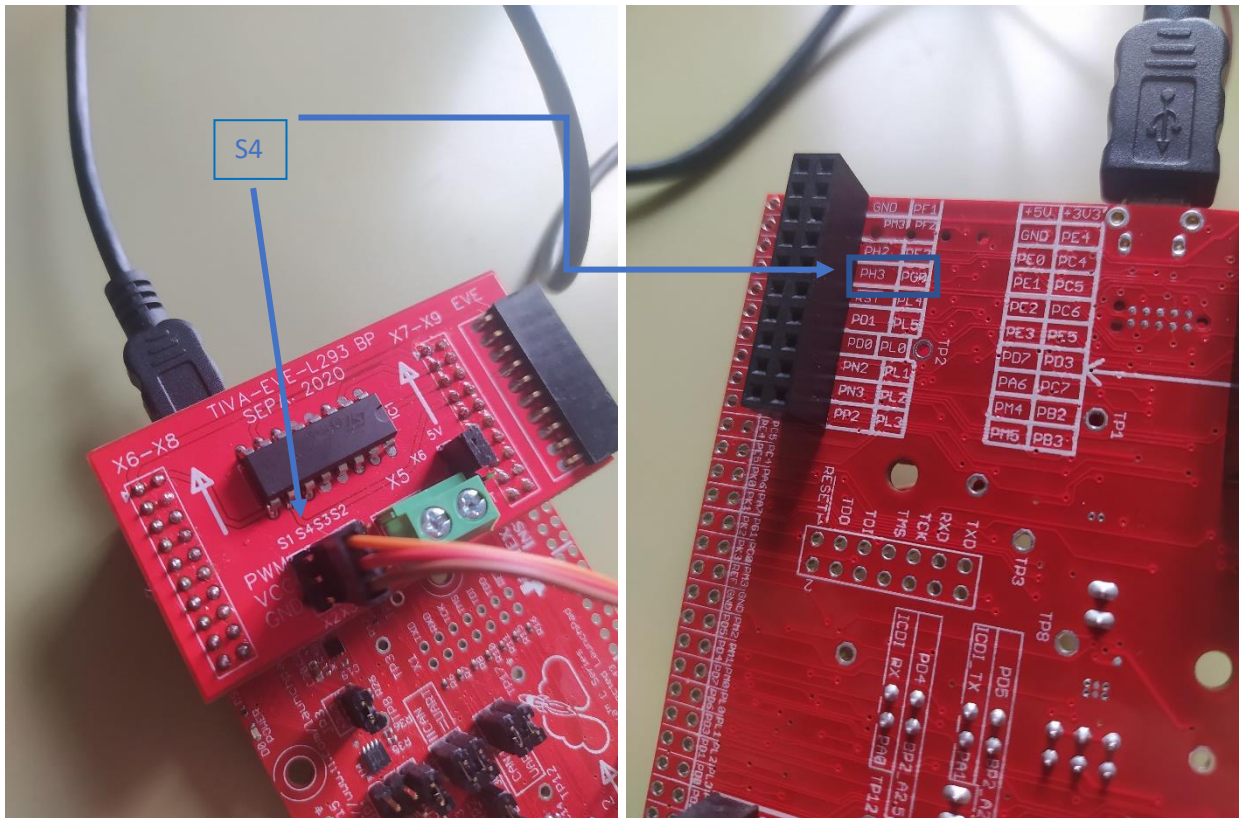
Realización de la práctica

1. Primer Ejercicio: Manejo de un Servomotor.

Se busca en este primer ejercicio realizar un programa que controle un servomotor conectado al pin PG0. El funcionamiento del programa será el siguiente:

- El servo comenzará en su posición central.
- Si se pulsa el botón 1, se moverá de la posición central a su posición en el extremo izquierdo, es decir, debe girar en sentido contrario a las agujas del reloj, y tras llegar a esa posición volverá a la posición inicial. (Se ha dejado 1s de espera antes de volver al centro).
- Si se pulsa el botón 2, se moverá de la posición central a su posición en el extremo derecho, es decir, debe girar en el sentido de las agujas del reloj, y tras llegar a esa posición volverá a la posición inicial. (Se ha dejado 1s de espera antes de volver al centro).
- Para conectar el servo al pin PG0 se parte del siguiente esquema y se conectará al pin S4 de la placa TIVA-EVE-L293, que estará conectado a dicho pin PG0 de la Launchpad,

como se ve en las siguientes imágenes. Además, habrá que conectar correctamente los conectores del servo a las señales PWM, VCC, GND.



Tras ver cómo se realiza la conexión del servo, se nos proponen una serie de consejos antes de proceder de la programación:

- Programar la lectura de los botones por interrupción.
- En el bucle infinito, se nos anima a poner el sistema en el modo de bajo consumo.
- Al saltar la rutina de interrupción (“despierta al micro”), realizar el movimiento requerido. Este movimiento deberá ser realizado en el bucle principal.
- Tras realizar el movimiento, deberá retornar al modo de bajo consumo.

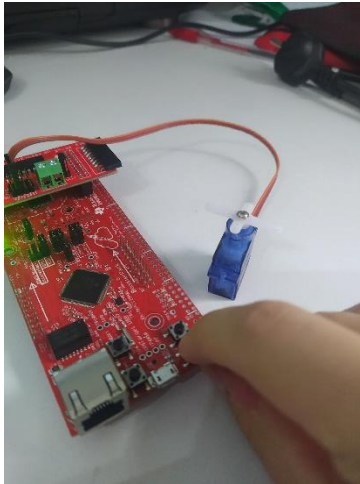
Para usar las características de bajo consumo se usarán las siguientes funciones de `sysctl.h`, que no explicaré aquí, simplemente citarlas:

- **extern void SysCtlPeripheralSleepEnable(uint32_t ui32Peripheral)**
- **extern void SysCtlPeripheralSleepDisable(uint32_t ui32Peripheral)**
- **extern void SysCtlPeripheralClockGating(bool bEnable)**

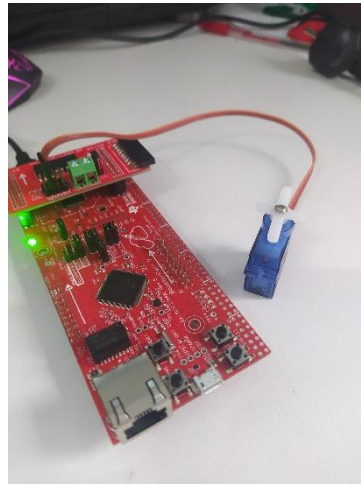
Con estas funciones podemos configurar el comportamiento de los periféricos al estar en modo de bajo consumo, y simplemente para entrar en modo de bajo consumo deberemos acudir a la siguiente función:

- **extern void SysCtlSleep(void)**

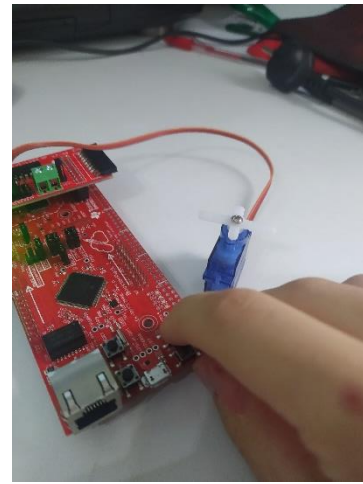
Tras haber comentado todos los aspectos a tener en cuenta, presentaré unas imágenes de las posiciones del servo en cada una de las posiciones que tendrá para ilustrar el funcionamiento antes de comentar el código:



-Posición Izquierda

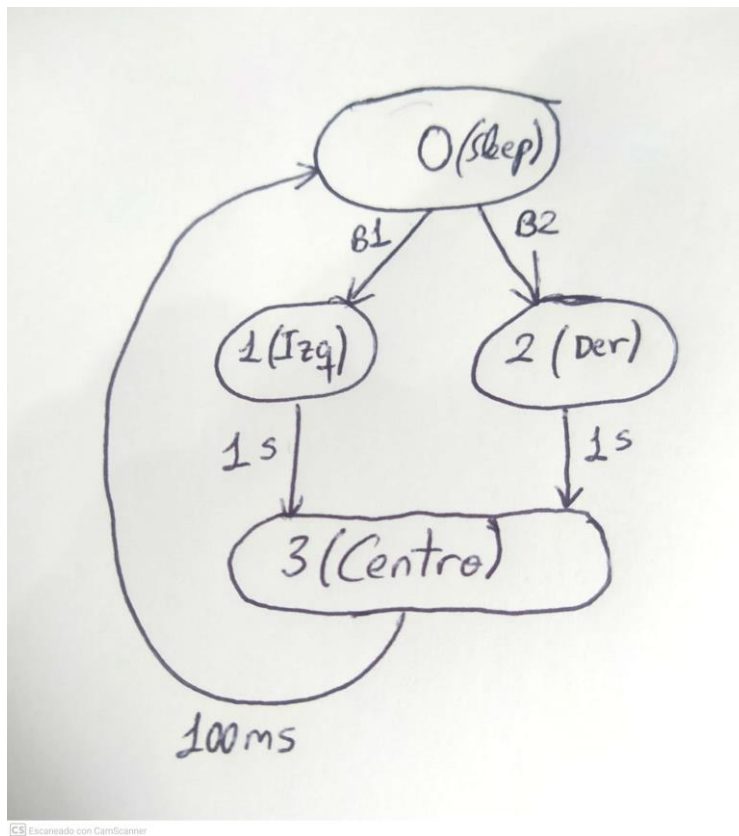


-Posición Central



-Posición Derecha

Básicamente lo que se ha propuesto para resolver el problema es basarse en la siguiente máquina de estados, que servirá también para el segundo ejercicio modificando ciertos detalles para la UART:



Dejo esta captura para orientación visual, veo más interesante explicarlo detalladamente sobre los programas posteriormente.

A continuación, voy a adjuntar el código del primer ejercicio de manera íntegra como se solicita, aunque con los comentarios extensos puede quedar poco legible dentro de la memoria, recomiendo verlo mejor abierto en CCS.

```
#include <stdint.h>
#include <stdbool.h>

#include "driverlib2.h"
/*****
*Primer ejercicio de la Segunda Práctica de SEPA, en este
*ejercicio concreto se busca realizar un programa que maneje
*un servo, de manera que si se pulsa (1 pulsación, no dejar pulsado)
*el B1, el servo recorrerá desde el punto central hasta el extremo
*izquierdo (sentido contrario a agujas del reloj) y volverá al punto
central.
*Si por su lado, se pulsa B2 (1 pulsación, no dejar pulsado),
*recorrerá desde el punto central hasta el extremo derecho
*(sentido de las agujas del reloj) y volverá al punto central.
*Hacer hincapié en:
*-Lectura de botones por interrupción
*-En bucle infinito, poner sistema en modo bajo consumo
*-Al saltar la interrupción, realizar el movimiento requerido (Movimiento
DEBE realizarse en el bucle principal)
*-Tras ejecutar movimiento, volver al modo de bajo consumo
*****/

#define MSEC 40000 //Valor para 1ms con SysCtlDelay()

#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON  !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)) //Macros para los
botones
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON  !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))

volatile int Max_pos = 4400; // Posición máxima del servo
volatile int Min_pos = 1000; // Posición mínima del servo
volatile int media = 2700; // Posición central del servo

int RELOJ, PeriodoPWM; // Variables para el reloj y para el período del
PWM
volatile int pos_100, pos_1000; // Variables para el período del PWM
volatile int posicion; // Variable para la posición del servo
int estado = 0; // Variable para recorrer la máquina de estados
int boton1 = 0; // Variable para conocer si se ha pulsado boton1
int boton2 = 0; // Variable para conocer si se ha pulsado boton2

/*****
* Rutina de interrupción del puerto J. (botones)
* Leo los pines, y en cada caso hago la función necesaria,
* (dar valores a variables de control para movimientos del servo)
* Establezco 20ms de debouncing en cada caso
* Y debo borrar el flag de interrupción al final.
*****/
void rutina_interrupcion(void)
{
    TimerDisable(TIMER0_BASE, TIMER_A); //Se desactiva el timer0
para que no cuente hasta que toque
    TimerDisable(TIMER2_BASE, TIMER_A); //Se desactiva el timer2
para que no cuente hasta que toque
```

```

    if(B1_ON)                // Si se pulsa el boton1
    {
        SysCtlDelay(10*MSEC); //Debouncing B1...
        while(B1_ON);
        SysCtlDelay(10*MSEC);
        boton1 = 1;          // se activa la variable que indica que se ha
pulsado B1
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0); //Se borra el flag de
la interrupción correspondiente
    }
    if(B2_ON)                // Si se pulsa el boton2
    {
        SysCtlDelay(10*MSEC); // Debouncing B2...
        while(B2_ON);
        SysCtlDelay(10*MSEC);
        boton2 = 1;          // se activa la variable que indica que se ha
pulsado B2
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1); //Se borra el flag de
la interrupción correspondiente
    }
}

void IntTimer0(void);        //Prototipos para las rutinas de interrupción de
los Timers
void IntTimer2(void);

int main(void)
{
    //Fijar velocidad a 120MHz
    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG); //Se habilita puerto G
(PIN PWM)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ); //Se habilita puerto J
(BOTONES)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); //Se habilita generador
PWM 0

    /***** CONFIGURACION PWM *****/
    PWMClockSet(PWM0_BASE, PWM_SYSClk_DIV_64); // al PWM le llega un
reloj de 1.875MHz
    GPIOPinConfigure(GPIO_PG0_M0PWM4); //Configurar el pin0 del
puerto G (PG0) a PWM (tiene 2 pasos esta config)
    GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);
    PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC); //Configurar el pwm0, contador descendente y sin
sincronización (actualización automática)
    PeriodoPWM=37499; // 50Hz a 1.875MHz; 50
Hz es lo que usan de pwm los servos "baratillos"; Se obtiene como:
(1875000/50)-1 = 37500-1 = 37499
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PeriodoPWM); //frec:50Hz
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, media); //Inicialmente,
1ms // Para que el servo se inicialice en la posición central
    PWMGenEnable(PWM0_BASE, PWM_GEN_2); //Habilita el
generador 2 (donde está PWM4)

```



```

    PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true);           //Habilita la
    salida 4

/*****
*****/

/*****          BOTONES
*****/

    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);
//J0 y J1: entradas

    GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1, GPIO_STRENGTH_2MA, GPIO
    _PIN_TYPE_STD_WPU); //Pullup en J0 y J1
    GPIOIntTypeSet(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1, GPIO_FALLING_EDGE);
// Definir tipo int: flanco bajada
    GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);
//Habilitar pines de interrupción J0, J1
    GPIOIntRegister(GPIO_PORTJ_BASE, rutina_interrupcion);
//Registrar (definir) la rutina de interrupción del puerto J (de los botones)
    IntEnable(INT_GPIOJ);
//Habilitar interrupción del pto J
    IntMasterEnable();
//Habilitar globalmente las ints

/*****
*****/

/*****          Configuración TIMER0 (1s entre
estados 1-3 y 2-3) *****/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);           //Habilita T0
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);        //T0 oneshot y
    conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1);            //Se carga el
    timer0 con 1s
    TimerIntRegister(TIMER0_BASE, TIMER_A, IntTimer0);      // Cada vez que
    cuenta 1s que entre en la interrupción (al ser oneshot, habrá que recargarlo)
    IntEnable(INT_TIMER0A);                                  //Habilitar las
    interrupciones globales de los timers
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);        // Habilitar las
    interrupciones de timeout

/*****
*****/

/*****          Configuración TIMER2 (100ms para
transición 3-0) *****/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);           //Habilita T2
    TimerClockSourceSet(TIMER2_BASE, TIMER_CLOCK_SYSTEM);   //T2 a 120MHz
    TimerConfigure(TIMER2_BASE, TIMER_CFG_ONE_SHOT);        //T2 oneshot y
    conjunto (32b)
    TimerLoadSet(TIMER2_BASE, TIMER_A, (RELOJ/10)-1);      //Se carga el
    timer2 con 100ms
    TimerIntRegister(TIMER2_BASE, TIMER_A, IntTimer2);      // Cada vez que
    cuenta 100ms que entre en la interrupción (al ser oneshot, habrá que
    recargarlo)
    IntEnable(INT_TIMER2A);                                  //Habilitar las
    interrupciones globales de los timers

```

```

    TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);           // Habilitar las
    interrupciones de timeout

/*****
*****/

/***** CONFIGURACION MODO BAJO CONSUMO *****/
*****/

    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);          // Dejar
    despierto el Pto J durante el Sleep
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOG);          // Dejar
    despierto el Pto G durante el Sleep
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);            // Dejar
    despierto el PWM durante el Sleep
    SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER0);         // Dormir el
    timer 0 durante el Sleep
    SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER2);         // Dormir el
    timer 2 durante el Sleep
    SysCtlPeripheralClockGating(true);                          // Habilitar
    el apagado selectivo de periféricos

/*****
*****/

    while(1){

        switch(estado){

            case 0:           // Estado 0:
                SysCtlSleep();                          // Entramos en
                modo bajo consumo
                if(boton1 == 1) estado = 1;              // Si se ha
                pulsado el botón 1, pasar al estado 1
                if(boton2 == 1) estado = 2;              // Si se ha
                pulsado el botón 2, pasar al estado 2
                break;

            case 1:           // Estado 1:
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, Max_pos); // Se pone el
                servo en la posición izquierda
                TimerEnable(TIMER0_BASE, TIMER_A);        // Habilitar
                Timer0 (cuenta 1s)
                boton1 = 0;                                // Se resetea
                la variable del boton1 para prepararla para la próxima pulsación
                break;

            case 2:           // Estado 2:
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, Min_pos); // Se pone el
                servo en la posición derecha
                TimerEnable(TIMER0_BASE, TIMER_A);        // Habilitar
                Timer0 (cuenta 1s)
                boton2 = 0;                                // Se resetea
                la variable del boton2 para prepararla para la próxima pulsación
                break;

            case 3:           // Estado 3:
                PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, media); // Se pone el
                servo en la posición central
                TimerEnable(TIMER2_BASE, TIMER_A);        // Habilitar
                Timer2 (cuenta 100ms)

```



```

        break;
    }
}

void IntTimer0(void) //Rutina Interrupción Timer0:
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag Interrupción
    Timer0
    if(estado == 1) //Si estamos en el
    estado1:
    {
        estado = 3; //Pasamos al estado 3
        TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1); //Se recarga el
        timer0 con 1s (era oneshot)
        TimerDisable(TIMER0_BASE, TIMER_A); //Se desactiva el
        timer0 para que no cuente hasta que toque
    }

    if(estado == 2) //Si estamos en el
    estado2:
    {
        estado = 3; //Pasamos al estado 3
        TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1); //Se recarga el
        timer0 con 1s (era oneshot)
        TimerDisable(TIMER0_BASE, TIMER_A); //Se desactiva el
        timer0 para que no cuente hasta que toque
    }
}

void IntTimer2(void) //Rutina
Interrupción Timer2:
{
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
    Interrupción Timer2
    if(estado == 3) //Si estamos en
    el estado3:
    {
        TimerLoadSet(TIMER2_BASE, TIMER_A, (RELOJ/10)-1); //Se carga el
        timer2 con 100ms (era oneshot)
        TimerDisable(TIMER2_BASE, TIMER_A); //Se desactiva el
        timer2 para que no cuente hasta que toque
        estado = 0; //Pasamos al
        estado 0
    }
}

//FIN

```

Ahora, aunque está comentado cada parte del código con bastante detalle, comentaré en esta memoria resumidamente los aspectos fundamentales del programa de este primer ejercicio.

Siguiendo el orden “cronológico” del código, al principio se realizan los includes de las librerías necesarias, para posteriormente hacer defines de ciertas macros de interés para el programa. A continuación, se declaran las variables para trabajar con la posición del servo (Max_pos, Min_pos y media), así como variables para reloj, período del PWM, posiciones... Finalmente se declaran una variable para recorrer la anterior máquina de estados planteada, así como dos variables para gestionar si se ha pulsado alguno de los botones.

Definimos la rutina de interrupción del puerto J, de manera que entraremos en ella si se pulsa cualquiera de los botones asociados a esos pines del puerto J. Si se ha pulsado el botón 1 se activa la variable boton1 y lo mismo para boton2 si se ha pulsado el botón 2.

Se definen los prototipos para las dos funciones de interrupción de los dos timers a usar.

Dentro del main, realizamos primero una “zona” de configuraciones, donde vamos activando periféricos, así como configurando PWM, botones y timers. No veo muy interesante comentar cada paso de la configuración, pero sí comentar que el Timer0 se configura one_shot y se carga con 1s, siendo que es el que contará 1s entre los estados 1 ó 2 y el 3. Mientras que el Timer2 también se configura one_shot pero se carga con 100ms, y este es el que cuenta los 100ms de la transición del estado 3 al 0. Mencionar que ambos al ser oneshot deberán recargarse como veremos posteriormente.

Finalmente se configura el comportamiento de los distintos periféricos durante el modo de bajo consumo.

Entramos ahora en el while(1), donde tenemos la máquina de estados. En ella tenemos estados del 0 al 3, como veíamos en la captura anterior. Primero entramos en el estado 0, donde entramos en modo de bajo consumo. Entonces si se pulsa alguno de los botones, se despertará el micro y según cual se haya pulsado pasaremos al estado 1 ó 2.

En el estado 1, en el cual entraremos si se ha pulsado el botón 1, llevamos el servo a la posición izquierda (giro antihorario) y se activa el timer0, que contaba 1s, además reseteamos la variable del boton1. De esta manera cuando el timer0 cuente 1s, cambiará al estado 3, como veremos al explicar la rutina de interrupción del timer0.

En el estado 2, en el cual entraremos si se ha pulsado el botón 2, llevamos el servo a la posición derecha (giro horario) y se activa el timer0, que contaba 1s, además reseteamos la variable del boton1. De esta manera cuando el timer0 cuente 1s, cambiará al estado 3, como veremos al explicar la rutina de interrupción del timer0.

Finalmente, en el estado 3, del que entramos tras 1s de espera después de los estados 1 ó 2, se lleva el servo a la posición central y se activa el timer2, que contaba 100ms, de manera que cuando el timer2 cuente esos 100ms, volverá al estado0 para entrar de nuevo en el modo de bajo consumo.

Por último, tenemos las dos rutinas de interrupción de los timers (fuera del main). En la rutina del timer0, primero borramos el flag de interrupción y tenemos dos posibilidades:

- Si estábamos en el estado 1, se pasa al estado 3, se recarga el timer con 1s y se desactiva dicho timer para que no cuente hasta que vuelva a los estados 1 ó 2 con su TimerEnable().

- Si estábamos en el estado 2, se pasa al estado 3, se recarga el timer con 1s y se desactiva dicho timer para que no cuente hasta que vuelva a los estados 1 ó 2 con su TimerEnable().

En cuanto a la rutina de interrupción del Timer2, primero borramos de nuevo el flag de la interrupción y en este caso, por asegurar comprobamos que veníamos del estado 3, aunque sólo se utiliza este timer en ese estado, y en ese caso, recargamos el timer2 con 100ms, desactivamos el timer2 para que no vuelva a contar hasta el TimerEnable() del estado 3 y pasamos al estado 0 de nuevo, donde como dijimos antes, se vuelve al modo de bajo consumo.

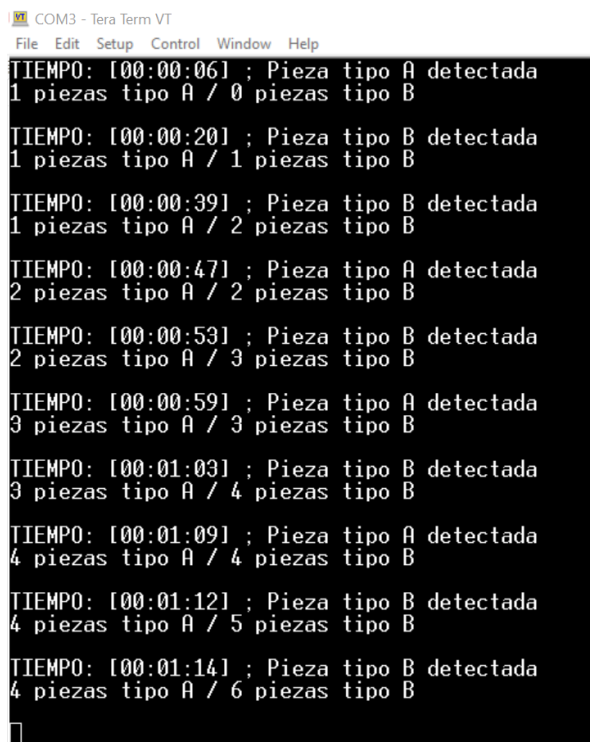
2. Segundo Ejercicio: Monitorización del proceso.

Se busca realizar una ampliación del apartado anterior, donde cada uno de los botones emula un sensor que detecta si llega una pieza tipo A ó tipo B y en función de esa información el servo deberá moverse en una dirección o en la otra. Lo que buscamos en este apartado, es que mantengamos un registro de las detecciones de los diferentes tipos de pieza, es decir, se busca mostrar el instante de tiempo en que se produjo una detección y qué tipo de pieza fue detectada, así como llevar un conteo del total de piezas de cada tipo detectadas durante todo el proceso de trabajo.

Se propone que el mensaje por consola sea similar al siguiente formato:

```
>> [hh:mm:ss] Pieza tipo A detectada.  
>> XX piezas tipo A / YY piezas tipo B
```

Por ilustrarlo, aunque se verá en el video, adjunto una imagen del registro recogido durante unas pruebas con el programa de este apartado:



```
COM3 - Tera Term VT
File Edit Setup Control Window Help
TIEMPO: [00:00:06] ; Pieza tipo A detectada
1 piezas tipo A / 0 piezas tipo B
TIEMPO: [00:00:20] ; Pieza tipo B detectada
1 piezas tipo A / 1 piezas tipo B
TIEMPO: [00:00:39] ; Pieza tipo B detectada
1 piezas tipo A / 2 piezas tipo B
TIEMPO: [00:00:47] ; Pieza tipo A detectada
2 piezas tipo A / 2 piezas tipo B
TIEMPO: [00:00:53] ; Pieza tipo B detectada
2 piezas tipo A / 3 piezas tipo B
TIEMPO: [00:00:59] ; Pieza tipo A detectada
3 piezas tipo A / 3 piezas tipo B
TIEMPO: [00:01:03] ; Pieza tipo B detectada
3 piezas tipo A / 4 piezas tipo B
TIEMPO: [00:01:09] ; Pieza tipo A detectada
4 piezas tipo A / 4 piezas tipo B
TIEMPO: [00:01:12] ; Pieza tipo B detectada
4 piezas tipo A / 5 piezas tipo B
TIEMPO: [00:01:14] ; Pieza tipo B detectada
4 piezas tipo A / 6 piezas tipo B
□
```

Donde cada una de esas entradas de dos líneas equivale a una detección de los sensores (botón 1 ó botón 2).

Tras comentar brevemente las novedades a implementar sobre el sistema del apartado 1, adjunto de nuevo íntegramente el código de este apartado:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "driverlib2.h"
/*****
*Segundo ejercicio de la Segunda Práctica de SEPA, en este
*ejercicio concreto se busca realizar un programa que maneje
*un servo, de manera que si se pulsa (1 pulsación, no dejar pulsado)
*el B1, el servo recorrerá desde el punto central hasta el extremo
*izquierdo (sentido contrario a agujas del reloj) y volverá al punto
central.
*Si por su lado, se pulsa B2 (1 pulsación, no dejar pulsado),
*recorrerá desde el punto central hasta el extremo derecho
*(sentido de las agujas del reloj) y volverá al punto central.
*Hacer hincapié en:
*-Lectura de botones por interrupción
*-En bucle infinito, poner sistema en modo bajo consumo
*-Al saltar la interrupción, realizar el movimiento requerido (Movimiento
DEBE realizarse en el bucle principal)
*-Tras ejecutar movimiento, volver al modo de bajo consumo
*
*Como ampliación de este segundo apartado además se comunicará por puerto
serie el instante y el tipo de pieza
*detectado, así como un conteo de las piezas de cada tipo totales detectadas
*****/

//*****
**
// The error routine that is called if the driver library encounters an
error.
//*****
**
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

#define MSEC 40000 //Valor para 1ms con SysCtlDelay()

#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)) //Macros para
los botones
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))

volatile int Max_pos = 4400; // Posición máxima del servo
volatile int Min_pos = 1000; // Posición mínima del servo
volatile int media = 2700; // Posición central del servo

int RELOJ, PeriodoPWM; // Variables para el reloj y para el período del
PWM
volatile int pos_100; // Variables para el período del PWM
volatile int posicion; // Variable para la posición del servo
int estado = 0; // Variable para recorrer la máquina de estados
```

```

int boton1 = 0;           // Variable para conocer si se ha pulsado boton1
int boton2 = 0;           // Variable para conocer si se ha pulsado boton2
char Pieza;               // Variable para gestionar si la pieza detectada
era tipo A ó B
int s = 0, m = 0, h = 0;   // Variables para almacenar horas,
minutos, segundos
int deteccionA = 0, deteccionB = 0; // Variables para indicar cuándo se ha
producido una detección tanto tipo A como B
int conta = 0, contB = 0;   // Variables para contar y acumular el
número de detecciones de cada tipo
int flag_ints1 = 0, flag_ints2 = 0; // Variable para gestionar
que interrupciones despiertan al micro

/*****
* Rutina de interrupción del puerto J. (botones)
* Leo los pines, y en cada caso hago la función necesaria,
* (dar valores a variables de control para movimientos del servo)
* Establezco 20ms de debouncing en cada caso
* Y debo borrar el flag de interrupción al final.
*****/

void rutina_interrupcion(void)
{
    TimerDisable(TIMER0_BASE, TIMER_A); //Se desactiva el timer0
para que no cuente hasta que toque
    TimerDisable(TIMER2_BASE, TIMER_A); //Se desactiva el timer2
para que no cuente hasta que toque

    if(B1_ON) // Si se pulsa el boton1
    {
        SysCtlDelay(10*MSEC); // Debouncing B1...
        while(B1_ON);
        SysCtlDelay(10*MSEC);
        boton1 = 1; // se activa la variable que indica que se ha
pulsado B1
        flag_ints1 = 1; // Para que se despierte del modo bajo
consumo
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0); //Se borra el flag de la
interrupción correspondiente
    }
    if(B2_ON) // Si se pulsa el boton2
    {
        SysCtlDelay(10*MSEC); // Debouncing B2...
        while(B2_ON);
        SysCtlDelay(10*MSEC);
        boton2 = 1; // se activa la variable que indica que se ha
pulsado B2
        flag_ints1 = 1; // Para que se despierte del modo bajo
consumo
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1); //Se borra el flag de la
interrupción correspondiente
    }
}

void IntTimer0(void);
void IntTimer1(void); //Prototipos para las rutinas de interrupción de
los Timers
void IntTimer2(void);

```

```

int main(void)
{
    //Fijar velocidad a 120MHz
    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);    //Se habilita puerto G
(PIN PWM)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);    //Se habilita puerto J
(BOTONES)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);    //Se habilita generador
PWM 0

    /***** CONFIGURACION PWM *****/
    /*****
    PWMClockSet(PWM0_BASE,PWM_SYSCLOCK_DIV_64);    // al PWM le llega un
reloj de 1.875MHz
    GPIOPinConfigure(GPIO_PG0_M0PWM4);            //Configurar el pin0 del
puerto G (PG0) a PWM (tiene 2 pasos esta config)
    GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);
    PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);    //Configurar el pwm0, contador descendente y sin
sincronización (actualización automática)
    PeriodoPWM=37499;                                // 50Hz a 1.875MHz; 50
Hz es lo que usan de pwm los servos "baratillos"; Se obtiene como:
(1875000/50)-1 = 37500-1 = 37499
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PeriodoPWM);    //frec:50Hz
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, media);    //Inicialmente,
1ms // Para que el servo se inicialice en la posición central
    PWMGenEnable(PWM0_BASE, PWM_GEN_2);            //Habilita el
generador 2 (donde está PWM4)
    PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true);    //Habilita la
salida 4

    /*****
    /*****

    /***** BOTONES *****/
    /*****
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);
    //J0 y J1: entradas

    GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPU); //Pullup en J0 y J1
    GPIOIntTypeSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_FALLING_EDGE);
    // Definir tipo int: flanco bajada
    GPIOIntEnable(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);
    //Habilitar pines de interrupción J0, J1
    GPIOIntRegister(GPIO_PORTJ_BASE, rutina_interrupcion);
    //Registrar (definir) la rutina de interrupción del puerto J (de los botones)
    IntEnable(INT_GPIOJ);
    //Habilitar interrupción del pto J
    IntMasterEnable();
    //Habilitar globalmente las ints

    /*****
    /*****

```



```

/***** Configuración TIMER0 (1s entre
estados 1-3 y 2-3) *****/
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);           //Habilita T0
TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);        //T0 oneshot y
conjunto (32b)
TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1);           //Se carga el
timer0 con 1s
TimerIntRegister(TIMER0_BASE, TIMER_A, IntTimer0);      // Cada vez que
cuente 1s que entre en la interrupción (al ser oneshot, habrá que recargarlo)
IntEnable(INT_TIMER0A);                                 //Habilitar las
interrupciones globales de los timers
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);        // Habilitar las
interrupciones de timeout

/*****

/***** Configuración TIMER2 (100ms para
transición 3-0) *****/
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);           //Habilita T2
TimerClockSourceSet(TIMER2_BASE, TIMER_CLOCK_SYSTEM);   //T2 a 120MHz
TimerConfigure(TIMER2_BASE, TIMER_CFG_ONE_SHOT);        //T2 oneshot y
conjunto (32b)
TimerLoadSet(TIMER2_BASE, TIMER_A, (RELOJ/10)-1);      //Se carga el
timer2 con 100ms
TimerIntRegister(TIMER2_BASE, TIMER_A, IntTimer2);      // Cada vez que
cuente 100ms que entre en la interrupción (al ser oneshot, habrá que
recargarlo)
IntEnable(INT_TIMER2A);                                 //Habilitar las
interrupciones globales de los timers
TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);        // Habilitar las
interrupciones de timeout

/*****

/***** Configuración TIMER1 (periódico a 1s
para el "reloj de detecciones") *****/
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);           //Habilita T1
TimerClockSourceSet(TIMER1_BASE, TIMER_CLOCK_SYSTEM);   //T1 a 120MHz
TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);        //T1 periódico y
conjunto (32b)
TimerLoadSet(TIMER1_BASE, TIMER_A, RELOJ-1);           //Se carga el
timer1 con 1s
TimerIntRegister(TIMER1_BASE, TIMER_A, IntTimer1);      //Cada vez que
cuente 1s que entre en la interrupción
IntEnable(INT_TIMER1A);                                 //Habilitar las
interrupciones globales de los timers
TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);        //Habilitar las
interrupciones de timeout
TimerEnable(TIMER1_BASE, TIMER_A);                     //Habilitar
Timer0 (no hace falta recargarlo, se activa al principio y que cuente
"infinito")

/*****

```

```

/***** CONFIGURACION UART *****/
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);           //Habilitar
Puerto A
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);           //Habilitar
UART0
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
//Configuración pines A0 y A1 como UART0
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
UARTStdioConfig(0, 115200, RELOJ);
//Configuración UART para stdio, ajustar baudios en TeraTerm a 115200!!

/***** CONFIGURACION MODO BAJO CONSUMO *****/
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);       //Dejar
despierto el Pto J durante el Sleep
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOG);       //Dejar
despierto el Pto G durante el Sleep
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);        //Dejar
despierto el PWM durante el Sleep
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER1);      //Dejar
despierto el timer 1 durante el Sleep
SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER0);    //Dormir el
timer 0 durante el Sleep
SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER2);    //Dormir el
timer 2 durante el Sleep
SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_GPIOA);     //Dormir el
puerto A durante el Sleep
SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_UART0);     //Dormir la
UART0 durante el Sleep
SysCtlPeripheralClockGating(true);                     //Habilitar
el apagado selectivo de periféricos

while(1){
    switch(estado){
        case 0:                                           //Estado 0:
            while(flag_ints1 != 1 && flag_ints2 == 0) SysCtlSleep();
            //Entramos en modo bajo consumo
            if(boton1 == 1) estado = 1;                  //Si se ha
            pulsado el botón 1, pasar al estado 1
            if(boton2 == 1) estado = 2;                  //Si se ha
            pulsado el botón 2, pasar al estado 2
            break;

        case 1:                                           //Estado 1:
            flag_ints1 = 0;
            PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, Max_pos); //Se pone el
            servo en la posición izquierda
            TimerEnable(TIMER0_BASE, TIMER_A);           //Habilitar
            Timer0 (cuenta 1s)
            boton1 = 0;                                   //Se resetea
            la variable del boton1 para prepararla para la próxima pulsación

```

```

        break;

        case 2: //Estado 2:
            flag_ints1 = 0;
            PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, Min_pos); //Se pone el
servo en la posición derecha
            TimerEnable(TIMER0_BASE, TIMER_A); //Habilitar
Timer0 (cuenta 1s)
            boton2 = 0; //Se resetea
la variable del boton2 para prepararla para la próxima pulsación
            break;

        case 3: //Estado 3:
            flag_ints1 = 0; //Reset del
flag de interrupciones antes de volver al estado 0 (donde se duerme)
            PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, media); //Se pone el
servo en la posición central
            TimerEnable(TIMER2_BASE, TIMER_A); //Habilitar
Timer2 (cuenta 100ms)
            break;
    }
}
}

void IntTimer0(void) //Rutina Interrupción
Timer0:
{
    flag_ints1 = 1; // Para que se
despierte del modo bajo consumo
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
Interrupción Timer0
    if(estado == 1) //Si estamos en el
estado1:
    {
        Pieza = 'A'; //Se ha clasificado
una pieza del tipo A, se guarda en su variable correspondiente
        deteccionA = 1; //Se activa la
variable de detección de tipo A
        contA++; //Se incrementa el
contador de piezas tipo A detectadas
        estado = 3; //Pasamos al estado 3
        TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1); //Se recarga el
timer0 con 1s (era oneshot)
        TimerDisable(TIMER0_BASE, TIMER_A); //Se desactiva el
timer0 para que no cuente hasta que toque
    }

    if(estado == 2) //Si estamos en el
estado2:
    {
        Pieza = 'B'; //Se ha clasificado
una pieza del tipo B, se guarda en su variable correspondiente
        deteccionB = 1; //Se activa la
variable de detección de tipo B
        contB++; //Se incrementa el
contador de piezas tipo B detectadas
        estado = 3; //Pasamos al estado 3
    }
}

```

```

        TimerLoadSet(TIMER0_BASE, TIMER_A, RELOJ-1);           //Se recarga el
timer0 con 1s (era oneshot)
        TimerDisable(TIMER0_BASE, TIMER_A);                   //Se desactiva el
timer0 para que no cuente hasta que toque
    }

    if(deteccionA == 1 || deteccionB == 1)
//Si se ha detectado una pieza A o una pieza B:
    {
        deteccionA = 0;
//Se resetea la variable de activación de detección A para la próxima
detección
        deteccionB = 0;
//Se resetea la variable de activación de detección B para la próxima
detección
        UARTprintf("TIEMPO: [%02d:%02d:%02d] ; Pieza tipo %c detectada
\n",h,m,s,Pieza); //Se muestra por puerto serie el instante y el tipo de
pieza que se detectó
        UARTprintf("%d piezas tipo A / %d piezas tipo B \n\n",contA,contB);
//Se muestra por puerto serie el conteo de piezas totales de cada tipo
detectadas
    }

}

void IntTimer1(void)                                           //Rutina Interrupción
Timer1:
{
    flag_ints2 = 0;                                           //Para que cuando el
micro se despierte cada segundo, se vuelva a dormir al momento
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //Borra flag Interrupción
Timer1
    s++;                                                       //Incrementa variables de
segundos en 1
    if(s == 60)                                               //Si se llega a 60
segundos:
    {
        m++;                                                 //Se incrementa minutos
en 1
        s = 0;                                               //Se resetea segundos a 0
    }
    if(m == 60)                                               //Si se llega a 60
minutos:
    {
        h++;                                                 //Se incrementa horas en
1
        m = 0;                                               //Se resetea minutos a 0
    }
    if(h == 100)                                              //Si horas llega a 100
(parece sensato contar más de 24h por si la máquina trabaja turnos muy
prolongados)
    {
        h = 0;                                               //Se resetea horas a 0
    }
}

void IntTimer2(void)                                           //Rutina
Interrupción Timer2:

```

```

{
    flag_ints1 = 1; // Para que se
    despierte del modo bajo consumo
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT); //Borra flag
    Interrupción Timer2
    if(estado == 3) //Si estamos en
    el estado3:
    {
        TimerLoadSet(TIMER2_BASE, TIMER_A, (RELOJ/10)-1); //Se carga el
        timer2 con 100ms (era oneshot)
        TimerDisable(TIMER2_BASE, TIMER_A); //Se desactiva el
        timer2 para que no cuente hasta que toque
        estado = 0; //Pasamos al
        estado 0
    }
}
//FIN

```

Ahora, aunque está comentado cada parte del código con bastante detalle, comentaré en esta memoria resumidamente los aspectos fundamentales del programa de este segundo ejercicio.

Al igual que antes, también seguiré el orden “cronológico” del código (de principio a fin).

Primeramente, se incluyen las librerías, luego se prepara una rutina de error que salta en caso de que hubiese algún problema con las librerías.

A continuación, definimos una serie de macros de interés para el programa, así como las posiciones definidas para el servo. Se declaran una serie de variables que coinciden con las del primer ejercicio para estados, boton1, boton2... Como novedad, declaramos una variable tipo Char (Pieza) para en el mensaje mostrar que tipo de pieza se detectó, así como 3 variables (s,m,h) para contar en nuestro “reloj del sistema” las horas, minutos y segundos. Tenemos también dos variables para indicar que se ha producido una detección tanto tipo A como B, así como dos contadores para contar el total de detecciones de cada tipo. Finalmente, declaramos dos variables “flags”, para dormir ó no dormir el micro según las interrupciones que salten.

En la rutina de interrupción de los botones tenemos prácticamente lo mismo que antes, a diferencia de que activamos flag_ints1, lo cual sirve para despertar al micro y que no se quede en SysCtlSleep().

Se definen los prototipos de los timers, ahora tendremos el timer2 que cuenta 1s entre estados (one_shot), el timer2 que cuenta 100ms del estado 3 al 0 (one_shot) y uno nuevo, timer1, que contará periódicamente 1s, para actuar como reloj para el sistema de detecciones.

Entramos de nuevo en el main, donde habilitamos y configuramos los periféricos. Destacar en este caso que tenemos lo mismo de antes respecto a configuraciones, salvo que ahora tenemos también que configurar el timer1, que será periódico y cargado con 1s. También como novedad, es necesario configurar la UART y sus pines asociados. Para terminar las configuraciones, se define el comportamiento de los periféricos en el modo de bajo consumo, donde por destacar, dejamos dormidos Timer0, Timer2, y la UART durante el sleep, mientras que el resto de periféricos siguen activos durante el sleep. Con “activo” quiere decir que sus interrupciones pueden saltar estando en sleep.

Entramos ahora en el while(1), donde como antes tenemos estados de 0 a 3. En el estado 0, tenemos una comprobación de las flags, tal que si flag_ints1 es distinto de 1 y flag_ints2 es 0, entonces entraremos en bajo consumo. Está planteado así ya que flag_ints2 será puesto a 0 por parte del timer1, el que actúa como reloj para las detecciones, de manera que se duerma el micro instantáneamente después de despertarse en el caso de que no haya otra interrupción que imponga flag_ints1 a 1, siendo que si esto ocurre, es que ha habido una interrupción de los botones o de los timer0/timer2 y entonces saldremos de ese while() para ejecutar normalmente la máquina de estados. Al igual que antes, según que botón se pulse, pasaremos al estado 1 o al 2.

En el estado 1, reseteamos el flag_ints1, llevamos el servo a la posición izquierda y habilitamos el contador de 1s entre estados, así como se resetea también la variable del boton1.

En el estado 2, reseteamos el flag_ints1, llevamos el servo a la posición derecha y habilitamos el contador de 1s entre estados, así como se resetea también la variable del boton2.

En el estado 3, reseteamos el flag_ints1, llevamos el servo a la posición central y habilitamos el contador de 100ms entre estados 3-0, para volver al estado 0, donde de nuevo según los flags, nos dormiremos o no según vaya interesando en función de las interrupciones existentes.

El tema de los flags principalmente está planteado para que, si hay cualquier interrupción como la de los botones, o timer0/timer2, se despierte al micro y funcione. Lo interesante es, si no hay interrupciones de botones ni nada más, sólo tendremos las interrupciones cada segundo del timer1 que hace de reloj para las detecciones, en este caso interesa que se vuelva al SysCtlSleep (modo de bajo consumo) rápidamente para no consumir recursos con esas interrupciones periódicas cuando no es necesario.

Finalmente, salimos del main, para definir las funciones de interrupción de los 3 timers:

- **Timer0:**

Se activa flag_ints1, se borra el flag de la interrupción de timer0.

Si estaba en el estado 1:

- Se guarda en pieza el carácter 'A' para mostrarlo indicando que fue detección tipo A
- Se activa la variable para indicar detección tipo A
- Se incrementa el contador de detecciones tipo A
- Se pasa al estado 3
- Se recarga el Timer0 con 1s
- Se desactiva el Timer0 para que no cuente hasta que convenga de nuevo

Si estaba en el estado 2:

- Se guarda en pieza el carácter 'B' para mostrarlo indicando que fue detección tipo B
- Se activa la variable para indicar detección tipo B
- Se incrementa el contador de detecciones tipo B
- Se pasa al estado 3
- Se recarga el Timer0 con 1s
- Se desactiva el Timer0 para que no cuente hasta que convenga de nuevo

Por último, si hemos tenido una detección sea del tipo A o tipo B haremos lo siguiente:

- Se resetean ambas variables de indicar detección

- Sacamos por la UART un mensaje con el instante de tiempo en que se detectó, el tipo de pieza que fue detectada, y el recuento de detecciones totales de cada tipo de pieza.

- **Timer1:**

En esta rutina, ponemos flag_ints a 0, para que si sólo se presenta esta interrupción el micro vuelva a dormirse como ya explicamos antes. Borramos el flag de la interrupción del timer1 y:

- Incrementamos la variable que cuenta los segundos
- Si segundos llega a 60, se incrementa los minutos y se resetean los segundos
- Si minutos llega a 60, se incrementan las horas y se resetean los minutos
- Si horas llega a 100, se resetean horas.

- **Timer2:**

En esta rutina, Se activa flag_ints1, se borra el flag de la interrupción de timer2.

Si estábamos en estado 3:

- Se recarga el timer con 100ms
- Se desactiva el timer para que no cuente de nuevo hasta que convenga
- Se pasa al estado 0

Con esto, quedan comentados ambos programas en detalle tanto en los propios códigos como en esta memoria.

Antes de concluir la memoria, recomiendo acudir a los vídeos adjuntos para comprobar el funcionamiento de ambos programas.