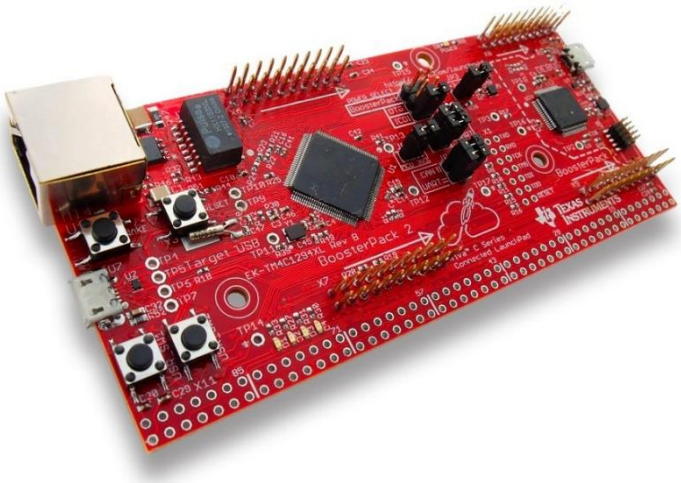

SISTEMAS ELECTRÓNICOS PARA AUTOMATIZACIÓN

Práctica 2

Álvaro Calvo Matos
Damián Jesús Pérez Morales

Índice

1. INTRODUCCIÓN

En la segunda práctica de la asignatura, se pretende controlar un servomotor y un control paso a paso con el uso de las interrupciones de timers, modos de bajo consumo y el empleo de la UART. Para ello, se realizarán tres ejercicios:

- Primer ejercicio: se hace mover un servomotor hacia arriba o hacia abajo según el botón que se pulse y, tras un segundo, moverse hacia la posición intermedia. Para el funcionamiento del servomotor, se ha recurrido al uso de timers para hacer el movimiento del servomotor; y se ha recurrido, además, al empleo de las funciones de bajo consumo para evitar cómputo innecesario.

- Segundo ejercicio: es similar al primero, pero permite mostrar por un terminal (UART) el número de piezas que se han almacenado de cada tipo: A y B, que se supone que llegan gracias a los botones implementados en el primer ejercicio de esta práctica.

- Tercer ejercicio: se simula un segundero con un motor paso a paso.

2. EJERCICIO 1

En el ejercicio 1, como se ha comentado en la introducción de la memoria de la práctica, se pretende mover un servomotor en tres diferentes posiciones: máxima, mínima y media. Cuando se pulsa un botón, el servo se mueve hacia la posición máxima o mínima (según cuál se pulse) y, al cabo de un segundo, vuelve a su posición de retorno. Se emplea el uso de timers e interrupciones para hacer funcionar el servomotor. El código está comentado línea por línea:

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

#include "driverlib2.h"

/**********************************
 Manejo del servomotor mediante pwm, interrupciones y modos de bajo consumo.
 El micro se encuentra en modo bajo consumo, solo con el pwm y
 los botones habilitados, y cuando estos ultimos producen una interrupcion,
 se activa el  timer para que cuente un segundo y luego vuelve a dormirse.

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

#define MSEC 40000
#define MaxEst 10


#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0))
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))


volatile int Max_pos = 4200; //3750
volatile int Min_pos = 1300; //1875
volatile int Mid_pos;
volatile int pos;

int RELOJ, PeriodoPWM;
int t;

// Rutina de interrupcion para el timer 0 (temporizamos un milisegundo)
void IntTimer0(void)
{
    t++;
    // Si ha pasado un segundo
    if(t >= 1000)
    {
        t = 0;
        // Volvemos a la posicion de reposo
        pos = Mid_pos;
        // Desabilitamos el timer 0 hasta que se vuelva a necesitar
        SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER0);  // Timer 0
    }
```

```
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
}

void rutina_interrupcion(void)
{
    // Habilitamos el timer durante el bajo consumo para temporizar un
segundo a partir de este instante
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);  // Timer 0
    if(B1_ON){  //Si se pulsa B1 -> Max_pos
        SysCtlDelay(10*MSEC);
        // Configuramos el pwm para ir hacia la posicion maxima
        pos = Max_pos;
        t = 0;
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0);
    }
    else if(B2_ON){ //Si se pulsa B2 -> Min_pos
        SysCtlDelay(10*MSEC);
        // Configuramos el pwm para ir hacia la posicion minima
        pos = Min_pos;
        t = 0;
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1);
    }
}

int main(void)
{
    // Configuracion del reloj
      RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

      // Habilitamos los puertos que vamos a necesitar
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    // Configuracion del TIMER
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);       //Habilita T0

    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);    //T0 periodico y
conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A, 120000 - 1);  //Para cada ms

    TimerIntRegister(TIMER0_BASE,TIMER_A,IntTimer0);
    IntEnable(INT_TIMER0A); //Habilitar las interrupciones globales de los
timers
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    // Habilitar las
interrupciones de timeout
    IntMasterEnable();  //Habilitacion global de interrupciones
    TimerEnable(TIMER0_BASE, TIMER_A);  //Habilitar Timer0, 1A

    // Configuracion del modo bajo consumo
    // Perifericos que deben permanecer encendidos
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);    // Modulador PWM
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOK);   // PWM6
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);   // Botones

    // El Timer 0 permanece apagado la mayoria del tiempo
    // Habilitaremos el Timer 0 cuando nos convenga
```

```
    SysCtlPeripheralSleepDisable(SYSCTL_PERIPH_TIMER0);  // Timer 0
    SysCtlPeripheralClockGating(true);                   //Habilitar el
apagado selectivo de periféricos

    // Configuracion de las interrupciones
    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
    IntEnable(INT_GPIOJ);                                   //Habilitar
interrupción del pto J
    IntMasterEnable();                                      //Habilitar
globalmente las ints

      PWMClockSet(PWM0_BASE,PWM_SYSCLK_DIV_64);    // al PWM le llega un
reloj de 1.875MHz

      GPIOPinConfigure(GPIO_PK4_M0PWM6);                //Configurar el pin
a PWM
    GPIOPinTypePWM(GPIO_PORTK_BASE, GPIO_PIN_4);

    // Ponemos resistencias de pull-up en los pulsadores
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);

GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPU);

    //Configurar el pwm0, contador descendente y sin sincronización
(actualización automática)
    PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);

      PeriodoPWM=37499; // 50Hz  a 1.875MHz

      // Inicializamos la variable del tiempo
      t = 0;

      // Calculamos la posicion media
      Mid_pos = (Max_pos + Min_pos)>>1;

      // Llevamos inicialmente el servo a la posicion central
      pos = Mid_pos;

    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, PeriodoPWM); //frec:50Hz
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, Mid_pos);      //Inicialmente, 1ms

    PWMGenEnable(PWM0_BASE, PWM_GEN_3);             //Habilita el generador 0

    PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT , true);      //Habilita la
salida 1

    while(1){
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, pos); // Ejecutamos el
movimiento
        SysCtlSleep();                                // Entramos en el modo
bajo consumo
    }
      return 0;
```

3. EJERCICIO 2

En el ejercicio 2, se pretende hacer lo mismo que en el apartado uno, pero con la diferencia de que ahora los botones simulan la entrada de piezas de tipo A o de tipo B. Cada vez que entra una de estas piezas, se muestra por un terminal el tipo de pieza que ha entrado, indicando el momento en el que entra cada una de ellas; y, además, se cuentan las piezas almacenadas de cada tipo. El código está comentado línea por línea:

```
 #include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>
#include "driverlib2.h"

// #include "HAL_I2C.h"
#include "utils/uartstdio.h"

/**********************************
 Manejo del servomotor mediante pwm, interrupciones y modos de bajo consumo.
 El micro se encuentra en modo bajo consumo, solo con el pwm del motor,
 los botones y el timer 0 habilitados.

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

#define MSEC 40000
#define MaxEst 10


#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0))
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))


volatile int Max_pos = 4200; //3750
volatile int Min_pos = 1300; //1875
volatile int Mid_pos;
volatile int pos;

// Variables para la monitorizacion del sistema
volatile int horas,minutos,segundos,milisegundos; //Tiempo
volatile int A,B;                                 //Piezas
char string[120];                                 //Comunicacion
volatile int aux;       //Variable auxiliar para la comunicacion

int RELOJ, PeriodoPWM;
int t;

// Rutina de interrupcion para el timer 0 (temporizamos un milisegundo)
void IntTimer0(void)
{
    // Variables que contabilizan el tiempo del sistema
    // Actualizamos segundos, minutos y horas segun corresponda
```

```
    milisegundos ++;
    if (milisegundos >= 1000)
    {
        milisegundos = 0;
        segundos ++;
        if (segundos == 60)
        {
            segundos = 0;
            minutos ++;
            if (minutos == 60)
            {
                minutos = 0;
                horas ++;
            }
        }
    }

    // Variables que controlan la vuelta del motor a la posicion central
    t++;
    // Si ha pasado un segundo
    if(t >= 1000)
    {
        t = 0;
        // Volvemos a la posicion de reposo
        pos = Mid_pos;
        // Desabilitamos el timer 0 hasta que se vuelva a necesitar
    }
    // Limpiamos la interrupcion del timer
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
}

void rutina_interrupcion(void)
{
    //Variable que indica al sistema que debe comunicar algo por UART
    aux = 0;

    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);  // Timer 0
    if(B1_ON){  //Si se pulsa B1 -> Max_pos
        SysCtlDelay(10*MSEC);
        // Configuramos el pwm para ir hacia la posicion maxima
        pos = Max_pos;
        t = 0;
        A ++;
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_0);
    }
    else if(B2_ON){ //Si se pulsa B2 -> Min_pos
        SysCtlDelay(10*MSEC);
        // Configuramos el pwm para ir hacia la posicion minima
        pos = Min_pos;
        t = 0;
        B ++;
        //sprintf(string,"%d piezas tipo A / %d piezas tipo B\n\n",A, B);
        //UARTprintf(string);
        GPIOIntClear(GPIO_PORTJ_BASE, GPIO_PIN_1);
    }
}

int main(void)
{
```

```
    // Configuracion del reloj
    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

    // Habilitamos los puertos que vamos a necesitar
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    // Configuracion del TIMER
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);       //Habilita T0

    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);    //T0 periodico y
conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A, 120000 - 1);  //Para cada ms

    TimerIntRegister(TIMER0_BASE,TIMER_A,IntTimer0);
    IntEnable(INT_TIMER0A); //Habilitar las interrupciones globales de los
timers
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    // Habilitar las
interrupciones de timeout
    IntMasterEnable();  //Habilitacion global de interrupciones
    TimerEnable(TIMER0_BASE, TIMER_A);  //Habilitar Timer0, 1A

    // Configuracion del modo bajo consumo
    // Perifericos que deben permanecer encendidos
    // Como ahora hay que monitorizar el tiempo en el que llega cada pieza,
el timer 0 debe permanecer encendido
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);    // Modulador PWM
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOK);   // PWM6
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);   // Botones
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);  // Timer 0

    SysCtlPeripheralClockGating(true);                   //Habilitar el
apagado selectivo de periféricos

    // Configuracion de las interrupciones
    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
    IntEnable(INT_GPIOJ);                                   //Habilitar
interrupción del pto J
    IntMasterEnable();                                      //Habilitar
globalmente las ints

    PWMClockSet(PWM0_BASE,PWM_SYSCLK_DIV_64);// al PWM le llega un reloj de
1.875MHz

    GPIOPinConfigure(GPIO_PK4_M0PWM6);              //Configurar el pin a PWM
    GPIOPinTypePWM(GPIO_PORTK_BASE, GPIO_PIN_4);

    // Ponemos resistencias de pull-up en los pulsadores
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);

GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPU);
```

```
    //Configurar el pwm0, contador descendente y sin sincronización
(actualización automática)
    PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);


    // Configuracion de la uart para la monitorizacion
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTStdioConfig(0, 115200, RELOJ);

    PeriodoPWM=37499; // 50Hz  a 1.875MHz

    // Inicializamos la variable del tiempo
    t = 0;


    // Inicializamos las variables de monitorizacion
    horas = 0;
    minutos = 0;
    segundos = 0;
    milisegundos = 0;
    A = 0;
    B = 0;
    aux = 0;

    // Calculamos la posicion media
    Mid_pos = (Max_pos + Min_pos)>>1;

    // Llevamos inicialmente el servo a la posicion central
    pos = Mid_pos;

    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, PeriodoPWM); //frec:50Hz
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, Mid_pos);      //Inicialmente, 1ms

    PWMGenEnable(PWM0_BASE, PWM_GEN_3);             //Habilita el generador 0

    PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT , true);      //Habilita la
salida 1

    while(1){
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, pos); // Ejecutamos el
movimiento
        // Si al salir del modo bajo consumo hay algo nuevo que transmitir
por la uart, lo transmitimos
        if(pos == Max_pos && aux == 0)
        {
            sprintf(string,"[%d:%d:%d] Pieza tipo A detectada. \n %d piezas
tipo A / %d piezas tipo B \n\n",horas, minutos, segundos, A, B);
            UARTprintf(string);
            aux = 1;
        }
        if(pos == Min_pos && aux == 0)
        {
```

```
            sprintf(string,"[%d:%d:%d] Pieza tipo B detectada. \n %d piezas
tipo A / %d piezas tipo B \n\n",horas, minutos, segundos, A, B);
            UARTprintf(string);
            aux = 1;
        }
        SysCtlSleep();                              // Entramos en el modo
bajo consumo
    }
    return 0;
}
```

4. EJERCICIO 3

En el ejercicio 3, se emplea un motor paso a paso para la implementación de un segundero y es importante destacar que el motor paso a paso tiene 514 pasos/vuelta, por lo que equivale a 8’53 pasos/vuelta, por lo que se ha hecho que se alterne para cada segundo dar 8 pasos o 9 pasos por segundo. Al llegar a los segundos múltiplos de 15, se aumenta esta variable para que encaje a la perfección con los ángulos de 90º, 180º, 270º y 360º. En el código está comentado el procedimiento:

```
//#define PART_TM4C1294NCPDT

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/tm4c1294ncpdt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/tm4c1294ncpdt.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"


/**********************************
 * Manejo de un motor paso a paso conectado en el boosterpack 1 (pines PK4, PK5, PM0, PG1)
 * donde se simula un segundero de un reloj. Puesto que los pasos del motor usado no son
 * divisibles entre 60, se hacen los ajustes necesarios al estilo de los años bisiestos,
 * es decir, sumando pasos extra para cubrir los decimales cada vez que pasamos por los cuartos
 * de minuto. Ademas se alterna entre 8 y pasos cada segundo.
 **********************************/

#define MSEC 40000
#define MaxEst 10

// Puertos necesarios (agrupados para facilitar el codigo)
uint32_t Puerto[]={
        GPIO_PORTK_BASE,
        GPIO_PORTK_BASE,
        GPIO_PORTM_BASE,
        GPIO_PORTG_BASE,

};
// Pines usados (agrupados para facilitar el codigo)
uint32_t Pin[]={
        GPIO_PIN_4,
        GPIO_PIN_5,
```

```
        GPIO_PIN_0,
        GPIO_PIN_1,
        };

// Secuencia de activacion de los pines para avanzar un paso (antihorario)
int Step[4][4]={0,1,0,0,
                0,0,1,0,
                0,0,0,1,
                1,0,0,0
};

#define B1_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0)
#define B1_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_0))
#define B2_OFF GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1)
#define B2_ON !(GPIOPinRead(GPIO_PORTJ_BASE,GPIO_PIN_1))



int RELOJ;

// Variables para controlar el movimiento y la posicion del motor
volatile int secuencia = 0;
int paso = 0;

volatile int t_5ms = 0;
volatile int mov = 0;
volatile int aux = 0;
volatile int segundos = 0;
volatile int descuadre = 0;

#define FREC 200 //Frecuencia en hercios del tren de pulsos: 5ms


// Rutina de interrupcion del timer (cada 5 ms)
void IntTimer(void)
{
    int i;

    t_5ms ++;

    // Cuando pasa un segundo actualizamos las variables y damos los pasos
necesarios
    if(t_5ms == 200)
    {
        mov = 1;
        t_5ms = 0;
        segundos ++;
        if(segundos == 60)
            segundos = 0;
        //Comprobamos si toca dar un paso extra para compensar los decimales
        if(!(segundos%15))
            descuadre = 1;
    }

    if(mov == 1 || descuadre == 1){
        // Damos los pasos al reves para que sean en sentido horario:
secuencia --;
        secuencia --;
        if(secuencia==-1) secuencia=3;
```

```
        // Damos el paso siguiente
        for(i = 0; i < 4; i ++)
            GPIOPinWrite(Puerto[i],Pin[i],Pin[i]*Step[secuencia][i]);

        // Contabilizamos el paso dado
        paso ++;
        // Si el paso recien dado era para descuadrar, no debemos
contabilizarlo
        if(descuadre == 1)
        {
            descuadre = 0;
            paso --;
        }
        // Dejamos de movernos si hemos dado los pasos necesarios para marcar
el segundo
        if(paso == (8 + aux))
        {
            paso = 0;
            // Variable que alterna entre 0 y 1 para dar un paso mas de forma
alternada cada segundo
            aux ^= 1;
            mov = 0;
        }
    }
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
}



int main(void)
{
      RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);


    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);       //Habilita T0
    TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM);   //T0 a 120MHz
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);    //T0 periodico y
conjunto (32b)
    TimerLoadSet(TIMER0_BASE, TIMER_A, 600000 - 1);     // Configuramos las
interrupciones cada 5 ms
    TimerIntRegister(TIMER0_BASE,TIMER_A,IntTimer);

    IntEnable(INT_TIMER0A); //Habilitar las interrupciones globales de los
timers

    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);    // Habilitar las
interrupciones de timeout
    IntMasterEnable();  //Habilitacion global de interrupciones

    TimerEnable(TIMER0_BASE, TIMER_A);  //Habilitar Timer0, 1, 2A y 2B
```

```
    GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0|GPIO_PIN_1);

GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPU);

    GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_4|GPIO_PIN_5);
    GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_1);
    GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_0);

    // Configuracion del modo bajo consumo
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);  // Timer 0

    SysCtlPeripheralClockGating(true);                   //Habilitar el
apagado selectivo de periféricos



      while(1){
          // Estamos siempre en bajo consumo menos cuando se produce una
interrupcion
          SysCtlSleep();
      }
      return 0;
}
```

5. COMENTARIOS ACERCA DE LA PRÁCTICA

En cuanto a la realización de la práctica, se han tenido varios contratiempos que nos ha demorado. En concreto, en el segundo ejercicio porque en el terminal no se mostraba por completo la cadena de caracteres que se imponen; y, en el tercer ejercicio, se trató de implementar un algoritmo que cuadre los segundos correspondientes a los múltiplos de 15 y, por ello, el servo no paraba de moverse. Se tardó bastante poco en corregir ambos errores.