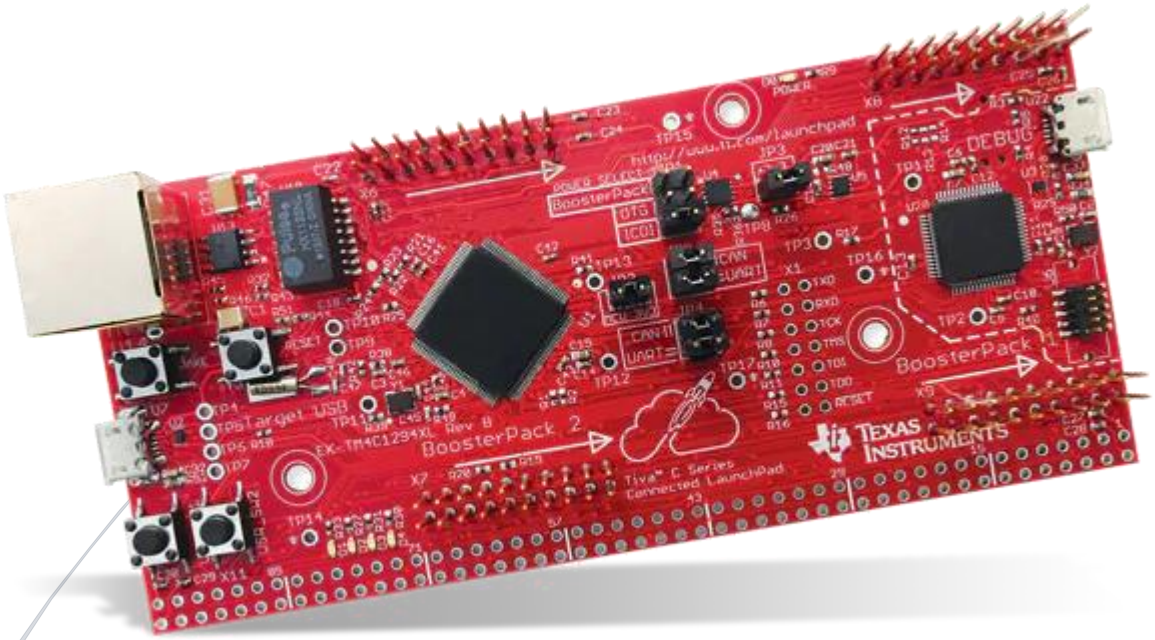
4ºGIERM

Práctica 2 SEPA

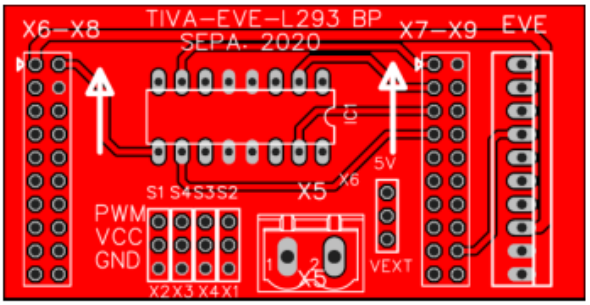
Clasificador de frutas automático



Daniel Peinado Ramírez, Francisco Javier Román Escorza
SISTEMAS ELECTRÓNICOS PARA LA AUTOMATIZACIÓN

1. Primer Ejercicio: Manejo de un Servomotor.

Partiendo del ejemplo disponible en Enseñanza virtual, se realizará un programa que maneje un servo conectado al pin PG0. El servo empezará estando en el estado central. Al pulsar el botón 1, hará el recorrido desde el punto central hasta el extremo izquierdo (giro en sentido contrario a las agujas del reloj), esperará 1s y volverá al punto inicial. De manera análoga, al pulsar el otro botón se realizará el giro contrario. Para ello es necesario conocer el diagrama de la placa de conexión TIVA-EVE-L293, que se adjunta como anexo a la práctica.



Conector	Pin X7
S1	1
S2	2
S3	3
S4	4

El sistema se construirá como un sistema síncrono con periodo 50ms. Para ello se programará un timer que marque el final de dicho periodo, y el bucle principal deberá durar siempre mucho menos que dicho periodo.

- ***En el bucle infinito, poner el sistema en modo de bajo consumo, a la espera que la interrupción del timer despierte al sistema.***
- ***Cuando la rutina de interrupción salte (y saque al sistema del modo de bajo consumo), se comprobará si hay algún botón apretado. En caso necesario, ejecutar***
- ***el movimiento requerido.***
- ***El movimiento NO SE DEBE realizar en la rutina de interrupción sino en el bucle principal.***
- ***Si se debe esperar un tiempo, se usará un nuevo estado en que se esperará que el tiempo llegue a ese valor (incrementándose a cada vuelta en función del periodo del timer del sistema).***

En el primer ejercicio procedemos a configurar los botones B1 y B2 de la misma forma que en la práctica anterior. En la función principal habilitamos el TIMER0, que es el que hemos utilizado, así como el conector S4 al que va conectado el servomotor.

Dentro de la máquina de estados definimos 4 posibilidades: reposo, ángulo mínimo, ángulo máximo y espera. Si pulsamos B1 el servo se mueve hacia el ángulo mínimo y vuelve a la posición de reposo y si pulsamos B2 hacemos lo mismo, pero hacia el ángulo máximo. Cada una de estas posiciones son calculadas en la función giro, que tomando de partida los valores máximo y mínimo que le llegan al PWM, devuelve un porcentaje de ángulo barrido donde 0 corresponde al ángulo mínimo, 50 a la posición de reposo y 100 al ángulo máximo.

En el caso de que tras el debug del programa existan problemas de funcionamiento, se ha de usar SysCtlSleepFake() en vez de SysCtlSleep, para dormir al micro justo al inicio de la máquina de estados para no consumir recursos en estado de reposo.

- ***Código:***

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

#include "driverlib2.h"


#define MSEC 40000// Multiplicador para que los delays estén en ms
//Definición de los pines de los botones
#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))
//Definicion estados, que utilizaremos dentro del while 1
#define reposo 0
#define ang_min 1
#define ang_max 2
#define espera 3
//Definición de SLEEP para poder debugear el programa
#define SLEEP SysCtlSleep()
//#define SLEEP SysCtlSleepFake()
//Valores máximo y mínimo que llegarán PWM, y serán utilizados en la
función giro
volatile int Max_pos = 4700;//4200; //3750
volatile int Min_pos = 1000; //1875

int RELOJ, PeriodoPWM, Flag_ints;
//Función que coloca el servo en el ángulo deseado, introduciendo como
parametro el porcentaje del rango posible.
void giro (int pos)
{
    int posicion=Min_pos+((Max_pos-Min_pos)*pos)/100;
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, posicion);   //Inicialmente, 1ms
}
//Función del SLEEP fake, utilizar en caso de que dé problemas al usar el
debugger
void SysCtlSleepFake(void)
{
 while(!Flag_ints);
 Flag_ints=0;
}

int contador; //Variable que utlizamos en la rutina de interrupción del
timer para contar las veces que se activa
//Interrupción del Timer0
void IntTimer0(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    Flag_ints=1;
    contador ++;
}

int main(void)
{   //Inicialización de la variable estado para la máquina de estados
    int estado = reposo;
```

```
RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

    //Activación de los pines necesarios
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    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,GP
IO_PIN_TYPE_STD_WPU); //Pullup en J0 y J1

    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, RELOJ/20 -1);
    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

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

    GPIOPinConfigure(GPIO_PG0_M0PWM4);          //Configurar los pines a
PWM
    GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);

    //Configurar el PWM4, contador descendente y sin sincronización
(actualización automática)
    PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);

    PeriodoPWM=37499; // 50Hz  a 1.875MHz
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PeriodoPWM); //frec:50Hz
    giro(50); //Posicion inicial del servo

    PWMGenEnable(PWM0_BASE, PWM_GEN_2);     //Habilita el generador 0
    PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT , true);    //Habilita la
salida 1
    //Activación de los periféricos necesarios durante el modo SLEEP
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);

    while(1){

        SLEEP;
        //Máquina de estados
```

```
      switch(estado)
        {   case reposo: //En el caso de que estemos en la posición inicial
de reposo
            if(B1_ON) estado = ang_min; //Pulsando el botón 1 el servo se
mueve a la izquierda
            else if (B2_ON) estado = ang_max; //Pulsando el botón 2 el
servo se mueve a la derecha
            contador = 0;
            break;

        case ang_min: //En el caso de que estemos en la posición izquierda,
de ángulo mínimo
            giro (0); //Llamada a la función giro
            estado = espera;
            contador = 0;
            break;
        case ang_max: //En el caso de que estemos en la posición derecha,
de ángulo máximo
            giro (100);//Llamada a la función giro
            estado = espera;
            contador = 0;
            break;
        case espera: //Tras hacia la izquierda o hacia la derecha y esperar
un tiempo, volvemos al reposo
            if (contador >= 20)
            {
                estado = reposo;
                giro(50);
            }
        }

    }
}
```

2. Segundo Ejercicio: Monitorización del proceso.

En este segundo apartado, realizaremos una monitorización del proceso anterior.

Suponiendo que cada uno de los botones es un sensor que detecta una pieza tipo A o tipo B, y que en función de eso está moviendo el servo para mandarla hacia un lado o hacia otro, se realizará una modificación en el programa anterior para que, cada vez que se pulse un botón, se mande un mensaje a la consola, informando de que se ha detectado una pieza (de tipo A o tipo B), y la hora (en HH:MM:SS desde que se arranca el proceso).

El segundo ejercicio es análogo al anterior, pero en este caso enviamos por puerto serie que tipo de pieza se ha detectado (si se pulsa B1 o B2) y el tiempo transcurrido en horas minutos y segundos desde que comenzó a correr el programa. Para hacer uso del puerto serie recurrimos a configurar los pines de la UART. Además, con la función imprimir escribimos lo que se nos pide en la pantalla del terminal, así como la conversión del tiempo transcurrido desde el compilado a horas, minutos y segundos. Esto lo conseguimos con la variable tiempo, que incrementa su valor en la interrupción del timer y se utiliza como base para calcular el tiempo.

En la máquina de estados simplemente añadimos 2 contadores para poder cuantificar cuantas veces se ha pulsado cada botón.

En el vídeo *"Practica_2.mp4"* adjuntado se muestra el funcionamiento del segundo ejercicio.

Código:

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

#include "driverlib2.h"


#define MSEC 40000 // Multiplicador para que los delays estén en ms
//Definición de los pines de los botones
#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))
//Definicion estados, que utilizaremos dentro del while 1
#define reposo 0
#define ang_min 1
#define ang_max 2
#define espera 3
//Definición de SLEEP para poder debugear el programa
#define SLEEP SysCtlSleep()
//#define SLEEP SysCtlSleepFake()
//Valores máximo y mínimo que llegarán PWM, y serán utilizados en la
función giro
volatile int Max_pos = 4700;//4200; //3750
volatile int Min_pos = 1000; //1875

int RELOJ, PeriodoPWM, Flag_ints;

uint32_t g_ui32SysClock;
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif
//Función que coloca el servo en el ángulo deseado, introduciendo como
parametro el porcentaje del rango posible.
void giro (int pos)
{
    int posicion=Min_pos+((Max_pos-Min_pos)*pos)/100;
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, posicion);   //Inicialmente, 1ms
}
//Función del SLEEP fake, utilizar en caso de que dé problemas al usar el
debugger
void SysCtlSleepFake(void)
{
 while(!Flag_ints);
 Flag_ints=0;
}
int contador; //Variable que utlizamos en la rutina de interrupción del
timer para contar las veces que se activa
```

```
unsigned int tiempo=0;
//Interrupción del timer: contador para el modo espera, y tiempo para saber el tiempo transcurrido desde la ejecución del programa
void IntTimer0(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    Flag_ints=1;
    contador ++;
    tiempo++;
}
// Variables para contar piezas A y B, así como de horas minutos y segundos
unsigned int contA = 0, contB = 0;
int seg, min, hora;
// Función que imprime los mensajes por UART y da un formato a las variables de tiempo
void imprimir (int pieza)
{
    seg = tiempo/20;
    hora = seg/3600;
    seg = seg%3600;
    min = seg/60;
    seg = seg%60;
    if (pieza == 0) UARTprintf("\n [%02d:%02d:%02d] Pieza tipo A detectada",hora, min, seg );
    else if (pieza == 1) UARTprintf("\n [%02d:%02d:%02d] Pieza tipo B detectada",hora, min, seg);
    UARTprintf("\n%02d piezas tipo A / %02d piezas tipo B", contA, contB);
}
int main(void)
 {   //Inicialización de la variable estado para la máquina de estados
    int estado = reposo;

    RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

    //Activación de los pines necesarios
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    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


    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, RELOJ/20 -1);
    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, 1, 2A y 2B

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

    GPIOPinConfigure(GPIO_PG0_M0PWM4);          //Configurar el pin a PWM
    GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_0);

    //Configurar el PWM4, contador descendente y sin sincronización
(actualización automática)
    PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);

    PeriodoPWM=37499; // 50Hz  a 1.875MHz
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PeriodoPWM); //frec:50Hz
    giro(50); //Posicion inicial del servo
    PWMGenEnable(PWM0_BASE, PWM_GEN_2);     //Habilita el generador 0
    PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT , true);    //Habilita la
salida 1
    //Activación de los periféricos necesarios durante el modo SLEEP
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOG);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER0);


    //Configuración del reloj para la UART
    g_ui32SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);
    //Inicialización
    //Configuración para el uso de la UART
    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, g_ui32SysClock);

    while(1){

        SLEEP;
        //Máquina de estados
      switch(estado)
        {
        case reposo: //En el caso de que estemos en la posición inicial de
reposo
            if(B1_ON) estado = ang_min;//Pulsando el botón 1 el servo se
mueve a la izquierda
            else if (B2_ON) estado = ang_max; //Pulsando el botón 2 el
servo se mueve a la derecha
            contador = 0;
            break;
```

```
case ang_min: //En el caso de que estemos en la posición izquierda, de
ángulo mínimo
            giro (0);
            //SysCtlDelay (1000*MSEC);
            estado = espera;
            contador = 0;
            contA++; //Aumentamos el contador de piezas A
            imprimir(0);//Llamada a la función
            break;
        case ang_max: //En el caso de que estemos en la posición derecha,
de ángulo máximo
            giro (100);
            //SysCtlDelay (1000*MSEC);
            estado = espera;
            contador = 0;
            contB++; //Aumentamos el contador de piezas B
            imprimir(1);//Llamada a la función
            break;
        case espera:
            if (contador >= 20)
            {
                estado = reposo;
                giro(50);
            }
        }


    }
}
```