

Apellidos:.....**SOLUCIÓN**.....

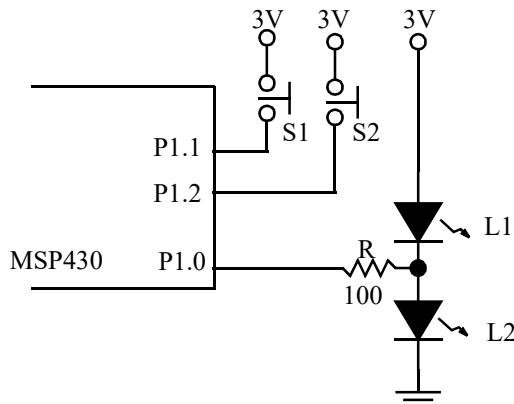
P1	P2

Nombre:.....

**Duración 2:00 horas**

- 1.- (2526C2 5/10) Considere el circuito de la figura. La tensión directa de los leds es  $V_f=2V$ , por lo que no es posible encender los dos leds simultáneamente con una tensión de alimentación de 3V. Explique y haga un programa en ensamblador del MSP430 que gestione **por multitarea cooperativa** y en el modo de menor consumo el encendido/apagado de los leds en función de los pulsadores. Inicialmente ambos leds están apagados, pero el led activo es el L1 y el modo de pulsación. Cada vez que se pulsa S2, se cambia el led activo. En modo de pulsación, cada vez que se pulsa S1 se enciende el led activo y se apaga cuando se suelta. En modo de conmutación, cada pulsación de S1 hace que se invierta el estado del led activo. Se cambia de modo cuando se pulsa S2 estando S1 pulsado.

Nota: Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados y la pila inicializada. Dispone de la librería `st.asm` y la función `cmp32`.



### SOLUCIÓN

Dado que los pulsadores no tienen resistencia externa, hay que usar las internas. En este caso debe ser de *pull-down*.

Los leds se controlan con una única línea y se dispone de 3 estados: en modo entrada los dos leds están apagados; en modo salida se enciende L1 cuando se escribe un 0 y L2 si se escribe un 1.

Como hay que actuar en los flancos, será necesario detectarlos y, dado que no se dispone de interrupciones, se codificará la tarea como una sencilla máquina de estados que guardará el valor anterior de la tecla. Al tratarse de dos teclas, o bien se codifica una tarea con 4 estados, o bien 2 tareas con 2 estados. La lectura de las teclas se hará con la periodicidad suficiente y sin necesidad de gestionar los tiempos, puesto que son tareas interactivas. Se escoge 100Hz como frecuencia del SystemTimer.

```

.cdecls C,LIST,"msp430ports.h"
.cdecls C,LIST,"st.h"
; -----
; Datos de configuración
; -----
; *** Puertos de E/S ***
LEDSBIT    .equ  BIT0
LEDSPORT   .equ  P1IN
S1BIT      .equ  BIT1
S1PORT     .equ  P1IN
S2BIT      .equ  BIT2
S2PORT     .equ  P1IN

;Frecuencia de los distintos procesos. En Hz
FTECLA     .equ  100           ;Frecuencia de escaneo de la tecla

```

```

; *** Frecuencia del System Timer ***
FTA      .equ  32768           ;Frecuencia del reloj del TA. Hz
FST      .equ  100            ;Frecuencia del SystemTimer. Hz

;-----
; Constantes calculadas
;-----
CCR0     .equ  FTA/FST-1      ;Valor a programar en CCR0 para stIni
PERIODO  .equ  FST/FTECLA   ;Tiempo entre ejecuciones (TICs)

;-----
; Variables
;-----
        .bss  Banderas, 1    ;Bit 0: Modo (0:pulsación, 1:conmutación)
                           ;Bit 1: Estado de pulsación de S1 (0: no pulsado)
                           ;Bit 2: Estado de pulsación de S2 (0: no pulsado)
MODO     .equ  BIT0          ;Modo de las teclas
TECLA1   .equ  BIT1          ;Estado de la tecla 1
TECLA2   .equ  BIT2          ;Estado de la tecla 2

;-----
; main                                         v1.0
;-----
main     ;Inicializar SystemTimer
        mov.w #CCR0, r12
        call #stIni

        ;Inicializar procesos
        call #Inicializa

superbucle call #TeclaS1      ;Tarea que gestiona S1
            call #TeclaS2      ;Tarea que gestiona S2
            bis.w #LPM3+GIE, sr ;Entrar en bajo consumo
            jmp superbucle
            .intvec RESET_VECTOR, main
            .text

;-----
; Inicializa                                         v1.0
;
; Inicializar proceso
;-----
Inicializa clr.b &Banderas
            ;bit.b #LEDSBIT, &LEDSPORT|POUT      ;Ll activo
            ;bit.b #LEDSBIT, &LEDSPORT|PDIR      ;Pero apagado
            ;bic.b #S1BIT, &S1PORT|PDIR
            ;bic.b #S2BIT, &S2PORT|PDIR
            bis.b #S1BIT, &S1PORT|PREN        ;Habilitar resistencia
            bic.b #S1BIT, &S1PORT|POUT        ;...de pulldown
            bis.b #S2BIT, &S2PORT|PREN        ;Habilitar resistencia
            bic.b #S2BIT, &S2PORT|POUT        ;...de pulldown
            ret

;-----
; Proceso de tecla S1                                         v1.0
;-----
TeclaS1  bit.b #TECLA1, &Banderas      ;Estado anterior tecla. Pulsada?
            jnz S1Pul                    ;...si. Ver si flanco de bajada
            ;La tecla estaba suelta. Sigue suelta?
S1NoPul  bit.b #S1BIT, &S1PORT|PIN      ;...no. Pulsada ahora?
            jz  TeclaS1Fin               ;...no. Salir
            ;Se acaba de pulsar S1. Comutar led activo
            xor.b #TECLA1, &Banderas      ;...si. Tecla pulsada. Guardar estado
            xor.b #LEDSBIT, &LEDSPORT|PDIR  ;Comutar led
            jmp  TeclaS1Fin              ;Salir
            ;La tecla estaba pulsada. Sigue pulsada?
S1Pul    bit.b #S1BIT, &S1PORT|PIN      ;Pulsada ahora?

```

```

        jnz    TeclaS1Fin           ;...si. Salir
        ;Se acaba de soltar S1. Comutar led activo si modo conmutación
        xor.b #TECLA1, &Banderas      ;...no. Tecla suelta. Guardar estado
        bit.b #MODO, &Banderas        ;Modo conmutación?
        jnz    TeclaS1Fin           ;...no. Salir
        xor.b #S1BIT, &S1PORT|PDIR    ;...si. Comutar led

TeclaS1Fin ret

;----- v1.0
; Proceso de tecla S2
;----- v1.0

TeclaS2    bit.b #TECLA2, &Banderas      ;Estado anterior tecla. Pulsada?
        jnz    S2Pul                 ;...si. Ver si flanco de bajada
        ;La tecla estaba suelta. Sigue suelta?
S2NoPul   bit.b #S2BIT, &S2PORT|PIN      ;...no. Pulsada ahora?
        jz     TeclaS2Fin           ;...no. Salir
        ;Se acaba de pulsar S2. Cambiar led activo y ver si cambiar modo
        xor.b #TECLA2, &Banderas      ;...si. Tecla pulsada. Guardar estado
        xor.b #LEDSBIT, &LEDSPORT|POUT  ;Cambiar led activo
        bit.b #S1BIT, &S1PORT|PIN      ;Pulsada S1 también?
        jz     TeclaS2Fin           ;...no. Salir
        xor.b #MODO, &Banderas        ;...si. Comutar modo
        jmp    TeclaS2Fin           ;Salir
        ;La tecla estaba pulsada. Sigue pulsada?
S2Pul     bit.b #S2BIT, &S2PORT|PIN      ;Pulsada ahora?
        jnz    TeclaS2Fin           ;...si. Salir
        xor.b #TECLA2, &Banderas      ;...no. Tecla suelta. Guardar estado

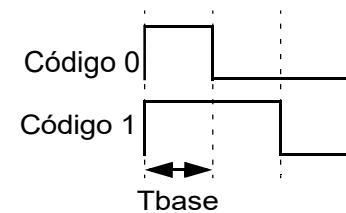
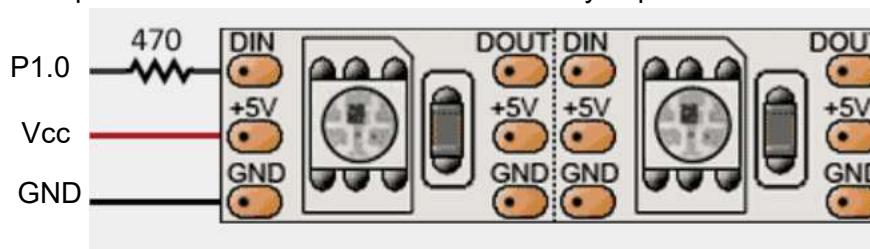
TeclaS2Fin ret

```

### CRITERIO DE CORRECCIÓN

- Constantes: 10%
- Principal: 10%
- Proceso S1: 40%
- Proceso S2: 40%

- 2.- (2526C2 5/10) El WS2812B es un módulo RGB inteligente que integra lógica de comunicaciones, tres controladores PWM, un led rojo, uno verde y otro azul dentro de cada píxel, permitiendo iluminar individualmente con hasta 256 niveles de brillo cada led y dando  $2^{24}=16,777.216$  colores. Los píxeles se conectan en cascada usualmente en tiras o paneles, usando un solo pin de datos; funcionan a 5V y son populares para proyectos de iluminación personalizados debido a su flexibilidad y capacidad de crear efectos visuales complejos.



La comunicación se hace por una única línea de datos. Se envían 24 bits por píxel, un byte por color, empezando por el msb en el orden RGB (rojo, verde, azul). Cada píxel se queda con los primeros 24 bits que le llegan y retransmite los demás, permitiendo que la información llegue al resto de leds de la tira. La transmisión de un 0 consiste en poner la línea en alto durante  $T_{base}$ , y en bajo durante  $2 \cdot T_{base}$ . La transmisión de un 1 mantiene la línea en alto durante  $2 \cdot T_{base}$ , y en bajo  $T_{base}$  (ver figura). Además, un código de reset consiste en dejar la línea a 0 durante al menos 50us. Hace que cada píxel actualice el color según la información recibida y que espere una nueva recepción.  $T_{base}=500\text{ns}$ .

Realice un programa en ensamblador del MSP430 que configure el puerto P1.0 en su función primaria (TA0.1) para controlar una tira de 64 píxeles WS2812B con la ayuda del TA0 y por intracciones. El programa mantendrá un vector llamado Leds con la información de color de los píxeles y los enviará por la línea cada 20ms (es decir, a 50 fps o frames per second).

Con objeto de que la CPU no se vea sobrecargada en exceso, aumente la frecuencia de la misma a 16MHz. Minimice también el número de interrupciones del TA0 configurándolo para que genere los códigos a enviar por la línea usando el modo PWM (sólo una IRQ por bit en lugar de 2). Al final de la secuencia de bits deberá generar un código de `reset` para actualizar los leds y dejarlos listos para el siguiente ciclo.

Minimice el consumo del sistema (optimice el código, use los modos de bajo consumo,...). Considere el perro guardián desactivado, los puertos desbloqueados y la pila inicializada.

## SOLUCIÓN

Al inicio se programa el DCO en 16MHz y se baja el divisor de MCLK DIVM=1 para que la CPU vaya a 16MHz. No se toca DIVS, con lo que el SMCLK sube a 2MHz, para que su periodo sea  $1/2\text{MHz}=500\text{ns}$ , coincidiendo con el Tbase.

El puerto se configura en modo salida con función primaria.

Se configurará el TA0 con fuente SMCLK/1 en modo UP (para que funcione en modo PWM= con CCR0=2. De esta forma, el TA0 hará 3 ciclos antes de resetearse. Se jugará con el valor de CCR1 para generar las distintas señales del protocolo: CCR1=0 para generar el ciclo de reset; CCR1=1 para generar el bit 0 y CCR1=2 para generar el bit 1. Se activarán las interrupciones de CCR0 para actualizar el ciclo de trabajo del ciclo siguiente. Se termina el programa principal habilitando las interrupciones y entrando el modo LPM1 ya que se necesita SMCLK activo. Todo lo demás, se hará por interrupciones.

Para gestionar las distintas fases del protocolo se usará la variable `EstadoTx=[ETX, ERESET, ECUADRO]`. Inicialmente `EstadoTx=ETX`, `BufferTx=Led[0]`, `CBit=8` y `CLed=0`.

Cuando `EstadoTx=ETX`, se procede a transmitir la trama almacenada en `Leds` que tiene un tamaño  $3*\text{NUMPIXELS}$  bytes. La variable `CLed=[0, 3*NUMPIXELS-1]` se usará para indicar cuál es el byte que se está transmitiendo. Dicho byte se guarda temporalmente en `BufferTx` para ir serializándolo. La variable `CBit=[1, 8]` indica cuántos bits quedan por transmitir. Cuando se trasmiten todos los bits de la trama, se pasa al estado `ERESTET`.

Cuando `EstadoTx=ERESTET`, se genera un silencio en la línea de 50us contando `NRESET` ciclos de la señal PWM con la ayuda de la variable `CReset=[1, NRESET]`. Una vez que se termina el ciclo de reset, se pasa a esperar que se cumplan los ciclos necesarios hasta completar 20ms, contados desde que se inició la transmisión. Cuando esto ocurra, pasamos al estado `ECUADRO`.

Cada vez que se pasa por la IRQ del CCR0 se decrementa la variable `CPWM=[1, NCUADRO]` para contar el número de ciclos de PWM que van pasando. `NCUADRO` se ha calculado para que sea igual a 20ms. Terminados los 20ms se pasa al estado `ETX` para repetir la transmisión de la trama.

```
.cdecls C,LIST,"msp430ports.h"
; Configuración de puertos -----
WSBIT      .equ  BIT0
WSPORT     .equ  P1IN
; Constantes de configuración -----
NPIXELS   .equ  64          ;Número de píxeles
FMCLK      .equ  16000000   ;Frecuencia del MCLK. En Hz
TBASE      .equ  500         ;Tiempo base WS2812. En ns
TRESET     .equ  50000       ;Tiempo del pulso de reset. En ns
TCUADRO    .equ  20000000   ;Tiempo del cuadro. En ns
RESPWM     .equ  3           ;Resolución de PWM
; Constantes calculadas -----
FSMCLK     .equ  1000000000/TBASE;Frecuencia del SMCLK. En Hz
DIVSMCLK   .equ  FMCLK/FSMCLK ;Divisor del SMCLK
TBIT       .equ  RESPWM*TBASE ;Tiempo bit WS2812. En ns
DIVTA      .equ  1           ;Divisor de reloj para TA0
FTA        .equ  FSMCLK/DIVTA ;Frecuencia del TA0
FPWM       .equ  FTA/RESPWM  ;Frecuencia del PWM
CCR0       .equ  RESPWM-1    ;Valor del CCR0 para conseguir FPWM
```

```

NRESET      .equ  TRESET/TBIT+1 ;Número de ciclos PWM para un cuadro (por exceso)
NPWMCUADRO .equ  TCUADRO/TBIT+1;Número de ciclos PWM para un cuadro (por exceso)
; Variables -----
.bss  Leds,3*NPIXELS;Buffer de colores de pixels
.bss  CPWM, 2, 2      ;Contador de ciclos PWM
.bss  CLed, 2          ;Contador de colores actual [0..3*NPIXELS-1]
.bss  CBit, 1          ;Contador de bit en el color actual
.bss  CReset, 1         ;Contador de ciclos de PWM en reset
.bss  BufTx, 1          ;Buffer de Tx
.bss  EstadoTx, 1       ;Estado de Tx. 0:Tx trama; 1:Tx reset
ETX      .equ  0          ;Transmitiendo trama
ERESET    .equ  1          ;Transmitiendo reset
ECUADRO   .equ  2          ;Esperando tiempo entre cuadros
;-----
; void main (void);                                     v1.0
;-----
main      ;DCO a 16MHz. MCLK=DCO/1 y SMCLK=DCO/8.
        mov.b #CSKEY_H, &CSCTL0_H           ;Desbloquear CS
        mov.w #DCOFSEL_4, &CSCTL1           ;DCO=16MHz
        bic.w #DIVM1|DIVM0, &CSCTL3         ;DIVM=1, MCLK=16MHz
        clr.b &CSCTL0_H                   ;Bloquear CS
; Inicializar variables
        mov.w #NPWMCUADRO, &CPWM
        mov.b #ETX, &EstadoTx
        clr.w &CLed
        mov.b #8, &CBit
        mov.b &Leds, &BufTx
;TA0. Generación de la señal PWM que controla el servo
;Puertos. P1.0 salida función primaria
        bis.b #WSBIT, &WSPORT+PDIR
        bis.b #WSBIT, &WSPORT+PSEL0
;CCR0. Fija frecuencia de PWM
        mov.w #CCRO, &TA0CCR0
        mov.w #CCIE, &TA0CCTL0           ;IRQ tras cada bit
;CCR1. PWM en modo Reset/Set (PWM Positivo). Inicialmente Tx reset
        mov.w #OUTMOD_0, &TA0CCTL1         ;Salida a 0 por defecto
        mov.w #0, &TA0CCR1                ;DC=0 para primer bit fake
        mov.w #OUTMOD_7, &TA0CCTL1         ;PWM positivo
;TA0, en modo UP con fuente SMCLK y divisor 1
        mov.w #TASSEL__SMCLK|ID__1|MC__UP|TACLR, &TA0CTL

        bis.w #GIE|LPM1, sr ;Entrar en bajo consumo

;-----
; TA0ISR                                         v1.0
;
; Subrutina de servicio de la IRQ de CCR0 de TA0. Genera la salida a los pixeles
;-----
TA00ISR    cmp.b  #ETX, &EstadoTx      ;Transmitiendo trama?
        jeq    TxTrama          ;...sí. Ir a sección de transmisión
        cmp.b  #ECUADRO, &EstadoTx    ;...no. Esperando siguiente cuadro?
        jeq    TA00ISRCdro      ;...sí. Contarlo
TxReset    dec.w   &CReset           ;...no. Contabilizar ciclo PWM en reset
        jnz    TA00ISRCdro      ;Quedan ciclos? Sí, salir
        mov.b  #ECUADRO, &EstadoTx    ;...no. Esperar hasta nuevo ciclo de tx
        jmp    TA00ISRCdro      ;Salir
TxTrama   push.w r12            ;Salvar registros
        mov.w  #1, r12           ;R12=1 (DC de valor lógico 0 por defecto)
        rla.b  &BufTx            ;C=Siguiente bit a Tx
        adc.w  r12              ;Actualizar ciclo trabajo PWM
        mov.w  r12, &TA0CCR1      ;Preparado para siguiente bit
        dec.w   &CBit             ;Contabilizar bit. Byte terminado?
        jnz    TA00ISRFTx      ;...no. Salir
        mov.b  #8, &CBit           ;...sí. Quedan 8 bits del siguiente color
        mov.w  &CLed, r12          ;R12=contador de color
        inc.w   r12              ;Siguiente color
        cmp.w  #3*NPIXELS, r12    ;Quedan?
        jlo    TA00ISRResF      ;...sí. Salir

```

```

        ;Transmitir pulso de reset
TA00ISRRes  mov.b  #ERESET, &EstadoTx      ;Transmitir pulso de reset
              mov.b  #NRESET, &CReset        ;Inicializar contador de reset
              clr.w  &TA0CCR1           ;DC=0 para trama de reset
              clr.w  r12                ;Inicializar contador de colores
TA00ISRResF mov.b  Leds(r12), &BufTX        ;Siguiente color al buffer
              mov.w  r12, &CLed          ;Contador de led a memoria
TA00ISRFTx  pop.w  r12                ;Recuperar registros
TA00ISRCdro dec.w  &CPWM            ;Un ciclo PWM menos para repetición
              jnz   TA00ISRFIn3        ;Quedan? Sí, salir
              mov.b  #ETX, &EstadoTx      ;....no. Transmitir trama de nuevo
              mov.w  #NPWMCUADRO, &CPWM        ;Inicializar contador de PWM
              mov.b  &Leds, &BufTx          ;Inicializar buffer de Tx
TA00ISRFIn3 reti

.intvec TIMER0_A0_VECTOR, TA00ISR
.intvec RESET_VECTOR, main

```

## CRITERIO DE CORRECCIÓN

- Main: 30%
- ISR: 70%