SISTEMAS BASADOS EN MICROPROCESADOR **Primera convocatoria 2023-2024**

Apellidos:.....SOLUCIÓN.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

1.- [3 Puntos] Preguntas cortas:

a) [15%] Escriba un trozo de código que configure el `SMCLK` a una frecuencia de 3'5 MHz, el `MCLK` a 7Mhz y deje sin alterar el `ACLK`.

b) [15%] Indique cómo leería el valor del `TA0R` si b.1) El timer está parado; b.2) El timer está corriendo con fuente `ACLK`; b.3) ídem con fuente `SMCLK`.

c) [30%] Haga una tarea cooperativa (sólo la tarea y las definiciones que le afecten) que conmute el led1 del Launchpad (P1.0) cuando se pulse el sw1 (P1.1) sabiendo que el tiempo de rebote del mismo es de 5ms. No use interrupciones. Sólo la tarea cooperativa funcionando por sondeo. Dispone del módulo `st` desarrollado en prácticas.

d) [40%] Analice el siguiente código e indique qué hace. Señale si sigue o no el convenio de llamada de C. En caso afirmativo, muestre el prototipo. En caso negativo, enumere los parámetros de entrada/salida, ubicación y tipo. Calcule el número de ciclos de ejecución, así como el tiempo total para f_{MCLK}=1MHz. Ensamble manualmente el programa y dé en hexadecimal el código máquina.

```
;                               Emulación       CiclosCódigo máquina
;-------------------------------------------------------------------------------
npi         mov.w  r12, r14     ;_________________|___|_____________________________
npi1        mov.b  @r12+, r15   ;_________________|___|_____________________________
            tst.b  r15          ;_________________|___|_____________________________
            jz     npi2         ;_________________|___|_____________________________
            cmp.b  r13, r15     ;_________________|___|_____________________________
            jne    npi1         ;_________________|___|_____________________________
            dec.w  r12          ;_________________|___|_____________________________
            jmp    npi3         ;_________________|___|_____________________________
npi2        clr.w  r12          ;_________________|___|_____________________________
npi3        sub.w  r14, r12     ;_________________|___|_____________________________
            ret
```

SOLUCIÓN

a) Clock System:

```
            mov.b  #CSKEY_H, &CSCTL0_H          ;Desbloquear CS
            ;Seleccionar DCO como fuente de MCLK y SMCLK
            and.w  #SELA2|SELA1|SELA0, &CSCTL2  ;Borrar SELM y SELS. Respetar SELA
            bis.w  #SELM_DCOCLK|SELS_DCOCLK, &CSCTL2:DCO fuente de MCLK y SMCLK
            ;Divisor de MCLK a 1 y de SMCLK a 2
            and.w  #DIVA2|DIVA1|DIVA0, &CSCTL3  ;Borrar DIVM y DIVS. Respetar DIVA
            bis.w  #DIVM__1|DIVS__2, &CSCTL3    :DIVM=1 y DIVS=2
            ;DCO a 7MHz
            mov.w  #DCOFSEL_5, &CSCTL1          ;DCO a 7MHz
            clr.b  &CSCTL0_H                    ;Bloquear CS
```

b) Lectura del TA0R

b.1) Timer parado. La lectura es segura. Sólo hay que acceder al registro:

```
            mov.w  &TA0R, r12
```

b.2) Fuente ACLK. Como el MCLK deriva otro reloj (DCO), la lectura no es segura. Para garantizar que la lectura es correcta se pueden usar dos procedimientos:

```
            ;Leer repetidamente TA0R hasta que dos lecturas consecutivas sean iguales
            mov.w  &TA0R, r12                   ;Primera lectura del TA0R
LeeTA0R     mov.w  &TA0R, r13                   ;Segunda lectura del TA0R
            cmp.w  r12, r13                     ;Son iguales?
```

```
            jeq    FinTA0R                      ;...sí. Salir. TA0R en R12 y R13
            mov.w  r13, r12                     ;...no. La última lectura a R12
            jmp    LeeTA0R                      ;Repetir segunda lectura
FinTA0R     ...                                 ;Usar TA0R

            ;Usar un CCR libre y forzar captura usando las contantes de entrada
            mov.w  &CM_1|CCIS_2|CAP, &TA0CCTL1  ;Captura canal 2 (GND) flanco subida
            mov.w  &CM_1|CCIS_3|CAP, &TA0CCTL1  ;Captura canal 3 (Vcc) flanco subida
            ...                                 ;El TA0R está en el TA0CCR1
```

b.3) Fuente SMCLK. Como el MCLK y SMCLK derivan del mismo reloj (DCO), la lectura es segura:

```
            mov.w  &TA0R, r12
```

c) Multitarea:

El proceso consiste en hacer una tarea con frecuencia 1/5ms=200Hz que escanee el SW1. Para eliminar los rebotes tendrá 3 estados. EPUL que indica que la tecla no está pulsada, EPUL, tecla pulsada y ERBT de esperando rebote. Se empieza en ENOPUL. Se permanece en el estado mientras no se pulse la tecla. Una vez que se hace, se pasa al estado ERBT. En dicho estado se vuelve a leer la tecla (5ms después) y se va al estado EPUL conmutando previamente el LED1 si la tecla sigue pulsada, y a ENOPUL si no lo estaba, ignorando la pulsación. En EPUL simplemente se espera a que se suelte la tecla para pasar a ENOPUL. En este estado no se hace la gestión de rebotes ya que no hay acción asociada a la liberación de la misma.

```
            .cdecls C,LIST,"msp430ports.h"
;------------------------------
; Datos de configuración
;------------------------------
; *** Puertos de E/S ***
LED1BIT     .equ   BIT0
LED1PORT    .equ   P1IN
SW1BIT      .equ   BIT1
SW1PORT     .equ   P1IN

;Frecuencia de los distintos procesos. En Hz
FTECLA      .equ   200           ;Frecuencia de escaneo de la tecla
PERIODO     .equ   FST/FTECLA    ;Tiempo entre ejecuciones (TICs)

;------------------------------
; Variables
;------------------------------
            .bss   teclaPE, 4    ;Próxima Ejecución del prceso de Parpadeo
            .bss   Estado, 1     ;Estado del proceso
;------------------------------
; Codificación de estados
;------------------------------
EINI        .equ   0*2           ;Estado inicial. Ficticio. Sólo para inicializar
ENOPUL      .equ   1*2           ;Estado tecla no pulsada
EPUL        .equ   2*2           ;Estado tecla pulsada
ERBT        .equ   3*2           ;Estado rebote
EULTIMO     .equ   ERBT          ;Último estado. Sólo a efectos de comprobación
;-------------------------------------------------------------------------------
; Inicializa                                                            v1.0
;
; Inicializar proceso
;-------------------------------------------------------------------------------
Inicializa clr.w  &teclaPE       ;Ejecutar desde el principio
            clr.w  &teclaPE+2
            mov.b  #EINI, &Estado
            ret

;-------------------------------------------------------------------------------
; Proceso Tecla                                                         v1.0
;-------------------------------------------------------------------------------
Tecla       mov.w  &teclaPE, r12 ;R15:R14=Instante de próxima ejecución
```

```
            mov.w  &teclaPE+2, r13
            call   #stComp       ;Comparar con SystemTimer
            tst.w  r12           ;Toca?
            jlo    teclaFin      ;...no. Salir
            add.w  #PERIODO, &teclaPE;...sí.Actualizar instante próxima ejecución
            adc.w  &teclaPE+2

            ;Proceso útil de la tarea. Procesar máquina de estados
            mov.b  &Estado, r14
            cmp.b  #EULTIMO+1, r14
            jhs    RbtError
            add.w  r14, pc
            jmp    RbtIni        ;Inicialización
            jmp    RbtNoPul      ;No pulsada
            jmp    RbtPul        ;Pulsada
            jmp    RbtRebote     ;Esperando rebote

            ;Estado Error. Desactivar tarea
RbtError    bic.b  #LED1BIT, &POUT+LED1PORT;LED1 apagado
            mov.w  #-1, &teclaPE ;Próxima ejecución... en el infinito
            mov.w  #-1, &teclaPE+1
            jmp    teclaFin
            ;Estado inicialización. LED salida apagadpo, SW1 entrada resist. pullup
RbtIni      bic.b  #LED1BIT, &POUT+LED1PORT;LED1 apagado
            bis.b  #LED1BIT, &PDIR+LED1PORT;LED1 como salida
            ;bic.b #SW1BIT, &PDIR+SW1PORT;SW1 como entrada
            bis.b  #SW1BIT, &PREN+SW1PORT;Resistencia ...
            bis.b  #SW1BIT, &POUT+SW1PORT;... de pullup
            mov.b  #ENOPUL, &Estado;Pasar a esperar pulsación de tecla
            ;jmp   teclaFin      ;Descomentar para esperar un ciclo
            ;Estado tecla no pulsada. Esperar a que se pulse
RbtNoPul    bit.b  #SW1BIT, &PIN+SW1PORT;Tecla pulsada?
            jz     teclaFin      ;...no. Salir
            mov.b  #ERBT, &Estado;Cambiar de estado a esperar tiempo de rebote
            jmp    teclaFin
            ;Estado tecla rebote. Comprobar si sigue pulsada
RbtRebote   bit.b  #SW1BIT, &PIN+SW1PORT;Tecla pulsada?
            jz     RbtANoPul     ;...no. A estado no pul
            xor.b  #LED1BIT, &POUT+LED1PORT;...sí. Conmutar led1
            mov.b  #EPUL, &Estado;Cambiar de estado a pulsada
            jmp    teclaFin
            ;Estado tecla pulsada. Esperar a que se suelte
RbtPul      bit.b  #SW1BIT, &PIN+SW1PORT;Tecla pulsada?
            jnz    teclaFin      ;...sí. Salir
                                 ;...no. Cambiar a no pulsada
RbtANoPul   mov.b  #ENOPUL, &Estado;Cambiar de estado a no pulsada
            ;jmp   teclaFin

teclaFin    ret
```

d) Análisis:

```
npi         mov.w  r12, r14      ;Preservar parámetro en R14. Hacer recorrido con R12
npi1        mov.b  @r12+, r15    ;Leer un carácter de cadena
            tst.b  r15           ;Fin de cadena?
            jz     npi2          ;...sí, salir
            cmp.b  r13, r15      :...no. Es el carácter buscado?
            jne    npi1          ;...no, siguiente carácter
            dec.w  r12           ;...sí, devolver dirección del carácter
            jmp    npi3
npi2        clr.w  r12           ;Devolver NULL por no encontrado
npi3        sub.w  r14, r12      ;R12=Índice del carácter buscado. Negativo si no
            ret
```

La subrutina recibe un puntero a byte en R12, que se copia a R14 para evitar que se pierda. Después se recorre un vector leyendo los elementos en R15. En caso de que se encuentre un elemento de valor 0, se va a la etiqueta `npi2`. Esto hace suponer que R12 es un puntero a una cadena de caracteres de tipo C. Mientras no se encuentre el final de la cadena, el carácter leído

es comparado con R13. Se hace un bucle que recorre la cadena hasta que se encuentra el final (yendo a `npi2`) o se encuentre el carácter buscado R13 (se va a `npi3`). Si el carácter se encuentra, se calcula la dirección del mismo en R12. Si no, dicha dirección es 0. Se sale restando la dirección de la cadena que se almacenó en R14 a R12, dejando así en R12 el índice a la posición del carácter buscado si se encontró, o un número negativo si no estaba.

Dado que el puntero entra por R12, el carácter buscado por R13, el resultado sale por R12 y no se alteran los registros R4-R10, podemos afirmar que la subrutina sigue el convenio de llamada de C, cuyo prototipo sería:

```
int npi (char *s, char c);
```

donde s es un puntero a una cadena de caracteres de tipo C y c es un carácter a buscar en al cadena.

Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
npi         mov.w  r12, r14     ;                 | 1 |
npi1        mov.b  @r12+, r15   ;                 | 2 |\
            tst.b  r15          ;cmp.b #0, r15    | 1 | |
            jz     npi2         ;                 | 2 | | N iteraciones
            cmp.b  r13, r15     ;                 | 1 | |
            jne    npi1         ;                 | 2 |/
            dec.w  r12          ;sub.w #1, r12    | 1 |
            jmp    npi3         ;                 | 2 |
npi2        clr.w  r12          ;mov.w #0, r12    | 1 |
npi3        sub.w  r14, r12     ;                 | 1 |
            ret                 ;mov.w @r1+,r0    | 3 |
```

Se ha comentado el código para expresar las instrucciones emuladas y el número de ciclos de cada instrucción. El bucle `npi1` se ejecuta N veces. N depende de la longitud de la cadena y de la probabilidad de que el carácter buscado esté dentro y su posición. El tiempo del bucle es de 8 en cada iteración. Si se sale por carácter encontrado, hay que sumar 1+7 ciclos por las instrucciones de antes y después del bucle. Si no se encuentra, habría que sumar 1+10 ciclos a las iteraciones previas y siendo N=L (L es la longitud de la cadena). El número total de ciclos sería 1+8*L+10 ciclos si el carácter no está y 1+8*N+7 si el carácter sí está. N dependerá de muchos factores, pero es siempre menor que L. Si suponemos una cadena de 10 caracteres, los datos serían 91 ciclos si el carácter no está y 48 ciclos si el carácter sí está y suponemos el caso medio, que sería que se encuentra en el carácter 5. Para un reloj de 1MHz, los tiempos son 91 y 48 μs respectivamente.

Ensamblado:

Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos), las de tipo II (las que tienen 1 operando) y las de salto relativo. En éstas sólo hay que codificar la condición y el desplazamiento, que se calcula como el número de palabras desde el origen (instrucción posterior al salto), al destino y que se anota en los comentarios. Cada instrucción ocupa una palabra, dando un total de 11 palabras.

```
;                               Emulación           Código máquina
;                                                   OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                   000100 OpC BDf -Rd-(tipo II)
;                                                   001 Cnd Desplazami (salto)
;-------------------------------------------------------------------------------
npi         mov.w  r12, r14     ;                   0100 1100 0000 1110 = 4C0E
npi1        mov.b  @r12+, r15   ;                   0100 1100 0111 1111 = 4C7F
            tst.b  r15          ;cmp.b #0, r15      1001 0011 0100 1111 = 934F
            jz     npi2         ;+4                 001 001 0000000100  = 2404
            cmp.b  r13, r15     ;                   1001 1101 0100 1111 = 9D4F
            jne    npi1         ;-5                 001 000 1111111011  = 23FB
            dec.w  r12          ;sub.w #1, r12      1000 0011 0001 1100 = 831C
            jmp    npi3         ;+1                 001 111 0000000001  = 3C01
npi2        clr.w  r12          ;mov.w #0, r12      0100 0011 0000 1100 = 430C
npi3        sub.w  r14, r12     ;                   1000 1110 0000 1100 = 8E0C
```

```
ret                 ;mov.w @r1+,r0      0100 0001 0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

- Análisis: 40%
- Tiempo de ejecución: 20%
- Codificación: 40% (1 punto por instrucción)

2.- (23/24 Conv1) **[3 Puntos]** La subrutina *bcd2bin* usa el convenio de llamada de C y atiende al siguiente prototipo:

```
long int bcd2bin (unsigned long int bcd);
```

bcd2bin convierte el número que se pasa en formato BCD empaquetado de 32 bits (un dígito binario cada cuatro bits), a binario natural (32 bits). En caso de error devuelve -1. Ejemplos de entradas y su salida:

- BCD: 0000 0000 0000 0000 0000 0000 1001 0101 (95 en BCD)
  Bin: 0000 0000 0000 0000 0000 0000 0101 1111 (95 en binario)
- BCD: 0000 0000 0000 0000 0000 0000 1011 0000
  Bin: 1111 1111 1111 1111 1111 1111 1111 1111 (Error).

a) Explique con palabras el algoritmo que va a emplear.

b) Escriba la subrutina en ensamblador.

SOLUCIÓN

a) El siguiente algoritmo procesa los dígitos BCD en orden de peso decreciente y los va acumulando en el par R_{13}:R_{12}. Si el BCD es $x_7x_6x_5x_4x_3x_2x_1x_0$ se implementa la fórmula:

$$n = (((((((0 \cdot 10 + x_7) \cdot 10 + x_6) \cdot 10 + x_5) \cdot 10 + x_4) \cdot 10 + x_3) \cdot 10 + x_2) \cdot 10 + x_1) \cdot 10 + x_0$$

Es decir, se entra en un bucle que, comenzando por el BCD más significativo, se multiplica el acumulado por 10 y se suma un nuevo dígito.

La multiplicación por 10 se hace con un sencillo algoritmo a base de desplazamientos y sumas.

Para extraer los dígitos BCD se colocan en el par R_{15}:R_{14} y se van rotando hacia la izquierda de 4 en 4 bits para colocar cada dígito BCD en los 4 lsb. Cada dígito se extrae con una operación AND.

b)

```
;-------------------------------------------------------------------------
; long int bcd2bin (unsigned long int bcd)                          v1.1
;
; Convierte el número que se pasa en formato bcd empaquetado a binario natural.
; En caso de error (algún dígito BCD ilegal), devuelve -1.
; -----Control de versiones-------
; v1.1 Optimizada la rotación de 4 bits a la izquierda de R15:R14
;-------------------------------------------------------------------------
bcd2bin     ;Salvar registros
            push   r9
            push   r10

            mov.w  r12, r14
            mov.w  r13, r15      ;R15:R14 = bcd
            clr.w  r12
            clr.w  r13           :R13:R12 = acumulador = 0

            mov.w  #8, r9        ;R9 contador de dígitos BCD
            jmp    bcd2binPV     ;Saltar primera multiplicación por 10

            ;Acumulador *= 10
bcd2binBuc2 rla.w  r12
            rlc.w  r13           ;R13:R12 = 2X
            mov.w  r12, r10
            mov.w  r13, r11      ;R11:R10 = acumulador = 2X
            rla.w  r10
            rlc.w  r11           ;R11:R10 = 4X
            rla.w  r10
```

```
            rlc.w  r11          ;R11:R10 = 8X
            add.w  r10, r12
            addc.w r11, r13     ;R13:R12 = 8X + 2X = 10X

            ;Obtener dígitos BCD por la izquierda y colocarlos en los 4LSB
bcd2binPV   mov.b  #4, r11      ;R11 contador de bits
bcd2binBuc1 bis.w  #BITF, r15   ;Copiar msb en carry
            rlc.w  r14          ;Desplazar R14 a la izq 1 bit metiendo C por dcha
            rlc.w  r15          ;Desplazar R15 a la izq 1 bit metiendo C por dcha
            dec.b  r11          ;Otra vuelta más
            jnz    bcd2binBuc1

            ;Poner a 0 todo menos los 4 LSB
            mov.w  r14, r11     ;R11 = bcd rotado (sólo palabra baja)
            and.w  #0xF, r11    ;Dejar sólo los 4 LSB
            cmp.b  #10, r11     ;Es mayor que 9?
            jhs    bcd2binErr   ;...sí, salir con error

            ;Acumular dígito. Acumulador = 10*acumulador + dígito
            add.w  r11, r12     ;Sumar dígito
            adc.w  r13          ;Sumar posible acarreo

            ;Procesar otro dígito
            dec.w  r9
            jnz    bcd2binBuc2

            jmp    bcd2binFin   ;Terminado. Salir

bcd2binErr  ;Se ha encontrado un dígito BCD erróneo. Salir con -1
            mov.w  #-1, r12
            mov.w  #-1, r13     R13:R12 = -1 (Error)

            ;Recuperar registros
bcd2binFin  pop    r10
            pop    r9
            ret
```

CRITERIO DE CORRECCIÓN

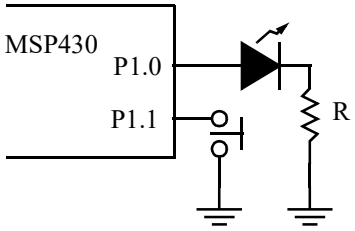
3.- **[4 Puntos]** Considere el circuito de la figura. Se trata de un sistema de luces con encendido/apagado y control de intensidad en una sola tecla con rebotes. Cada vez que se pulsa momentaneamente el *switch*, se conmuta el estado del led. Si se actúa sobre el *switch* al menos un cuarto de segundo, se entra en el modo de cambio de intensidad hasta que es liberado. La misma varía entre el 25% y el 100% a un ritmo del 100%/2s. Cada vez que se llega a los límites, la variación de brillo cambia de sentido. En el momento de la liberación del *switch* se paraliza el cambio de intensidad de la luz y se memoriza para posteriores encendidos/apagados. Suponga inicialmente el led apagado, con un brillo memorizado del 100%.

Haga un programa de mínimo consumo en ensamblador del MSP430 que gestione con el TA0 y por interrupciones este comportamiento.

Notas:

- Use una frecuencia de 128 Hz y 32 niveles de brillo.
- El pulsador tiene un tiempo de rebote de 10ms.
- Si la pulsación es menor al tiempo de rebote, se ignorará la misma. Si el tiempo de pulsación es inferior a 0’25s, se considerará una pulsación momentánea y se procederá a conmutar el estado de la luz. Si es mayor, se entra en modo de cambio de intensidad (si el led estaba apagado, se enciende para proceder al ajuste de la intensidad).
- Considere el perro guardián desactivado, los puertos desbloqueados, pila inicializada y LFXT en marcha con cristal de 32768Hz.
- La función primaria de P1.0 es TA0.1 (CCI1A para capturas y Out1 para salidas).

- La función primaria de P1.1 es TA0.2 (CCI2A para capturas y Out2 para salidas).



SOLUCIÓN

Inicialmente se configurará P1.1 como entrada con resistencia de *pull-up*. La tecla pulsada se detectará con un valor bajo (lógica negativa). El control del led del puerto 1.0, que se configurará como salida, es con lógica positiva (1 enciende, 0 apaga).

El puerto P1.0 se configurará en su función primaria para que sea controlado por TA0 directamente como una señal PWM. La detección de los flancos de la tecla puede hacerse dejando el pin en modo E/S digital o en su función primaria y se controlaría por el TA0 configurando el CCR2 en modo captura. En este caso se ha optado por la primera opción.

Dado que $f_{TA} = 2^N \cdot f_{PWM}$, sustituyendo N=5 (32 niveles de brillo) y f_{PWM}=128 Hz, obtenemos f_{TA}=4096 Hz. Tomando ACLK como entrada de TA0, tenemos que usar un divisor de 32768/4096=8.

Se da la causalidad de que el tiempo de pulsación corta (250ms) es múltiplo del periodo de la señal PWM (7'8ms), lo que nos permite medir de una manera cómoda el mismo. Bastaría contar el número de ciclos (32) de la señal PWM desde la pulsación de la tecla en la ISR del CCR0. Esto nos permite usar el modo UP que simplifica mucho la gestión de la señal PWM y hace que el rango de ciclos de trabajo sea completo. Sin embargo, la medida del tiempo de rebote tendría que hacerse aproximada a un ciclo de la señal PWM (7'8ms) en lugar de los 1'28 ciclos que sería lo más correcto (10ms). De necesitar una medida más precisa de tiempos, sería necesario dejar el timer en modo CONT, lo que complica la generación de señales PWM, pero aumentaría la resolución de la medida de tiempos de los 7'8ms (1/128) a 0'244ms (1/4096).

El control del tiempo que lleva la tecla pulsada se hace con la variable `CiclosPul`, que se inicializa a -1 para indicar que el contador está desactivado. En la ISR de P1.1, cuando se detecta la pulsación de la tecla, se pone `CiclosPul=0` iniciando la cuenta de ciclos en la ISR del CCR0. Aquí se incrementa `CiclosPul` siempre que no sea un número negativo y no se haya llegado al número de ciclos de la pulsación corta. En caso de igualar los ciclos de la pulsación corta, se entra en modo dinámico de brillo. Si no, se sale sin más. Si la luz estaba apagada al terminar el tiempo de la pulsación corta, se enciende la luz antes de proceder al cambio de brillo.

En la ISR de P1.1, cuando se libera la tecla se mira cuánto vale `CiclosPul`. Si es 0 o 1, se considera que no ha pasado el tiempo de rebote y se ignora la pulsación. Si es menor de 32, se considera una pulsación corta y se conmuta el estado de la luz. Nótese que esto tiene el efecto de conmutar la luz cuando se libera la tecla y no cuando se pulsa. En cambio, si es mayor o igual, se finaliza el modo dinámico que se activó en la ISR del CCR0 haciendo `CiclosPul=-1`. Aunque que la variable es de tamaño byte y tiene signo, como deja de incrementarse una vez se llega a los 32 ciclos, no hay problemas de desbordamiento si se tiene el pulsador mucho tiempo activado.

El control del brillo se hará con la ayuda de tres variables.

- `Brillo`, que recogerá el valor actual de brillo aunque la luz esté apagada y que será un número entre 8 (25%) y 31 (100%).
- `BrilloFlags` que recoge un puñado de banderas: el bit 0 es `BRDIR`, que indicará la dirección de cambio (0 para bajar brillo y 1 para subirlo); y el bit 1 es `BRON` que indica si la luz está encendida (1) o apagada (0).
- La pendiente de subida o bajada es de 100%/2s, lo que significa variar 32 tonos de brillo en 2 segundos. Para una señal PWM de 128Hz (128 ciclos en un segundo), significa variar 32 veces en 2*128 ciclos, o 1 vez cada 8 ciclos. Se usará la variable `BrilloCont` como un contador de ciclos de la señal PWM. Inicialmente se cargará con 8 y, cada vez

que se entre en la ISR (si `CiclosPul>=32`), se decrementará. Al llegar a 0, se incrementará/decrementará el brillo actual en función de `BRDIR` y se volverá a reponer `BrilloCont` a 8. Aquí también se controla que el brillo nunca suba del 100% ni baje del 25% y que se cambie la dirección cuando se llegue a alguno de los topes.

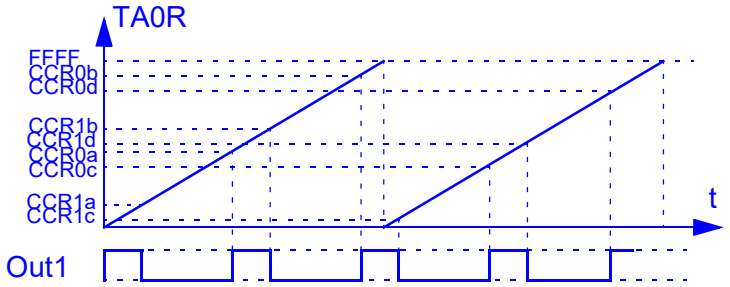
```
;Constantes de configuración
LED         .equ   BIT0
PUL         .equ   BIT1

FACLK       .equ   32768          ;Frecuencia de ACLK. En Hz
FPWM        .equ   128            ;Frecuencia del PWM. En Hz
RESPWM      .equ   32             ;Resolución del PWM. En unidades
TREBOTE     .equ   10             ;Tiempo de rebote. En ms
TCORTA      .equ   250            ;Tiempo de pulsación corta. En ms

;Constantes calculadas
FTA         .equ   FPWM*RESPWM    ;Frecuencia del TA0. En Hz
DIVTA       .equ   FACLK/FTA      ;Divisor del TA0
NREBOTE     .equ   TREBOTE*FPWM/1000;Tiempo de rebote (en ciclos PWM)
NCORTA      .equ   TCORTA*FPWM/1000;Tiempo de pulsación corta (en ciclos PWM)
VBRILLO     .equ   2*FPWM/RESPWM ;Velocidad de la pendiente de brillo
BRMIN       .equ   25*RESPWM/100 ;Brillo mínimo
BRMAX       .equ   RESPWM-1      ;Brillo máximo

;Variables
            .bss   Brillo,2      ;Brillo actual si estuviera encendida
            .bss   CiclosPul,1   ;Número de ciclos que lleva la tecla pulsada
            .bss   BrilloCont,1  ;Contador de ciclos con mismo brillo
            .bss   BrilloFlag,1  ;Conjunto de flags de brillo (ver más abajo)
BRDIR       .equ   0             ;0:Bajando brillo; 1:subiendo brillo
BRON        .equ   1             ;0:luz apagada; 1:luz encendida

;-------------------------------------------------------------------------------
;main                                                                      v1.0
;-------------------------------------------------------------------------------
main        ;Inicializar variables
            clr.b  &BrilloFlag   ;Luz apagada, bajando brillo
            mov.b  #BRMAX, &Brillo;Brillo al 100%
            mov.b  #VBRILLO, &BrilloCont;Inicializar contador de ciclos
            mov.b  #-1, &CiclosPul;Contador de ciclos desde pulsación desactivado

            ;P1.1 entrada pulldown sensible a flanco de bajada (IRQ)
            ;bic.b #PUL, &P1DIR  ;Entrada
            bis.b  #PUL, &P1REN  ;Resistencia...
            bic.b  #PUL, &P1OUT  ;...de pulldown
            bis.b  #PUL, &P1IES  ;Sensible a flanco de bajada
            bic.b  #PUL, &P1IFG  ;Borrar flag
            bis.b  #PUL, &P1IE   ;Habilitar IRQ

            ;P1.0 salida función 1 (TA0.1)
            bis.b  #LED, &P1DIR  ;Salida
            bis.b  #LED, &P1SEL0 ;Función 1

            ;TA0.1 en modo comparación con salida PWM RESET-SET. Brillo al 0%
            mov.w  #OUTMOD_7, &TA0CCTL1
            clr.w  &TA0CCR1

            ;TA0.0 fijando excursión del TA0. IRQ
            mov.w  #CCIE, &TA0CCTL0
            mov.w  #RESPWM-1, &TA0CCR0

            ;TA0 con ACLK/DIVTA modo UP
            mov.w  #DIVTA-1, &TA0EX0;Divisor extra
            mov.w  #TASSEL__ACLK|ID__1|MC__UP|TACLR, &TA0CTL
            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; P1ISR                                                                    v1.0
```

```
;
;ISR del P1. Gestiona las pulsaciones y liberaciones de la tecla. Para ello, cambia
;el flanco cada vez que se entra.
;Cuando se pulsa la tecla (flanco de bajada), pone CiclosPul a 0 para que la ISR del
;CCR0 lleve la cuenta del número de ciclos PWM que está la tecla pulsada.
;Cuando se libera la tecla, se calcula el tiempo que ha estado pulsada.
;Si es menor del tiempo de rebote, se ignora. Si es menor del tiempo de pulsación
;corta, se conmuta el estado de la luz. Si es mayor, se cancela el modo de cambio
;de brillo.
;-------------------------------------------------------------------------------
P1ISR       xor.b  #PUL, &P1IES  ;Cambiar flanco activo
            bic.b  #PUL, &P1IFG  ;Borrar flag
            bit.b  #PUL, &P1IN   ;Tecla pulsada?
            jnz    TeclaNoPul    ;...no. Se ha liberado
TeclaPul    ;...sí. Tecla pulsada. Inicializar contador de ciclos
            clr.b  &CiclosPul    ;Empezar cuenta de ciclos PWM desde pulsación
            jmp    P1ISRFin      ;Salir
TeclaNoPul  ;Tecla liberada
            ;Calcular tiempo desde pulsación
            cmp.b  #NREBOTE+1, &CiclosPul;Ha pasado el tiempo de rebote?
            jlo    FinTecla      ;...no. Salir ignorando pulsación
            cmp.b  #NCORTA, &CiclosPul;...sí. Ha pasado el tiempo de pulsación corta?
            jhs    FinTecla      ;...sí. Parar el cambio de brillo
            ;...no. Pulsación corta. Conmutar estado de la luz
            bit.b  #BRON, &BrilloFlags;Está la luz encendida?
            jnz    ApagarLuz     ;...sí. Apagar luz
EncenderLuzmov.w  &Brillo, &TA0CCR1;Encender led
            jmp    ConmLuz       ;Salir
ApagarLuz   clr.w  &TA0CCR1      ;Apagar led
ConmLuz     xor.b  #BRON, &BrilloFlags;Conmutar flag de encendido
FinTecla    mov.w  #-1, &CiclosPul;Desactivar contador ciclos tecla
P1ISRFin    reti

;-------------------------------------------------------------------------------
; TA00ISR                                                                 v1.0
;
;ISR del CCR0. Gestiona la señal ISR del CCR0. Lleva la cuenta de ciclos PWNM que la
;tecla ha estado pulsada (si CiclosPul!=-1) y cambia el brillo si procede.
;Si el brillo es estático, no hace nada. Si hay que cambiar el brillo, primero
;se decrementa el número de ciclos en este brillo.
;Cuando se llegue a 0, cambiar en función de BrilloDir una unidad hasta llegar
;al máximo o al mínimo. En ese caso, dar la vuelta a BrilloDir.
;-------------------------------------------------------------------------------
TA00ISR     ;Contar ciclos PWM si tecla pulsada y no liberada
            tst.w  &CiclosPul    ;Contador de ciclos activo?
            jn     TA00ISRFin    ;...no, salir
            cmp.w  #NCORTA-1, &CiclosPul;...sí. Alcanzado tiempo máx pulsación corta?
            jhs    VerBrillo     ;...sí, gestionar brillo dinámico
            inc.w  &CiclosPul    ;...no. Contar un ciclo más
            jmp    TA00ISRFin    ;Salir
VerBrillo   ;Encender luz si apagada
            bit.b  #BRON, &BrilloFlags;Luz encendida?
            jnz    LuzOn         ;...sí, cambiar brillo
            bis.b  #BRON, &BrilloFlags;..no. Encender
            mov.w  &Brillo, &TA0CCR1;Que se vea
            ;Gestionar cambio de brillo en modo dinámico
LuzOn       dec.b  &BrilloCont   ;Decrementar contador de ciclos. Fin?
            jnz    TA00ISRFin    ;...no, salir
            mov.b  #VBRILLO, &BrilloCont;...sí, reponer contador de ciclos
            bit.b  #BRDIR, &BrilloFlags;Ver pendiente
            jz     PendNeg       ;...negativa
PendPos     inc.w  &TA0CCR1      ;...positiva. Incrementar brillo
            cmp.w  &TA0CCR0, &TA0CCR1;Ha llegado al 100%?
            jlo    TA00ISRFin    ;...no, salir
            jmp    CambiarDir    ;...sí, invertir dirección
PendNeg     dec.w  &TA0CCR1      ;Decrementar brillo
            cmp.w  #BRMIN, &TA0CCR1;Ha llegado CCR1 al 25%?
            jnz    TA00ISRFin    ;...no, salir
```

```
CambiarDir xor.b  #BRDIR, &BrilloFlags;...sí, cambiar dirección
TA00ISRFin reti

            .intvecRESET_VECTOR, main
            .intvecPORT1_VECTOR, P1ISR
            .intvecTIMER0_A0_VECTOR, TA00ISR
```

Tratemos ahora el caso de que no sea aceptable la medida de tiempos en base al periodo de la señal PWM. Lo normal sería usar el modo UP para generar las señales PWM fácilmente. Sin embargo, eso dificulta la medida de los tiempos de rebote (10ms) y de pulsación corta (250ms), mayores al periodo de la señal PWM (1/128=7'812ms). Por ello, se va a usar el modo CONTINUOUS, lo que nos facilita las medidas de tiempo, aunque se va a complicar la generación de la señal del led. No obstante, dado que se quiere variar dinámicamente el brillo, de todas formas era necesario capturar la IRQ del CCR0.

De momento supongamos que no se va a variar el brillo de forma dinámica. ¿Cómo se genera una señal PWM en modo continuo? La respuesta es: modificando en cada IRQ del CCR0 los valores de CCR0 y CCR1. Inicialmente se hace CCR0=31 y CCR1=8, lo que implica una señal de 128Hz con 25% de ciclo de trabajo para la frecuencia de reloj programada (4096Hz). En cada interrupción por comparación del CCR0 se suman 32 tanto a CCR0 como CCR1. Esto debe hacerse lo antes posible dentro de la ISR para que tengan efecto valores bajos de CCR1. Los subíndices a, b, c y d indican la secuencia que van tomando ambos registros.:



No obstante, este mecanismo no es perfecto y sería posible que los ciclos de trabajo más bajos no funcionaran correctamente en función de la frecuencia de MCLK y del reloj del TA0. En nuestro caso en el que el brillo mínimo es del 25% y la frecuencia es baja, no debería haber problemas. Si, además, se quiere cambiar el brillo, se aprovecha esa misma ISR para cambiar el valor del CCR1.

Inicialmente el CCR0 se cargará con RESPWM-1 y el CCR1 empezará en 0 (brillo al 0%) en modo RESET/SET (lógica positiva). En cada IRQ se incrementarán ambos en RESPWM unidades. Además, el CCR1 podrá ser incrementado/decrementado en una unidad si se está en modo dinámico.

El control del brillo se hará con la ayuda de tres variables.

- `Brillo`, que recogerá el valor actual de brillo aunque la luz esté apagada y que será un número entre 8 (25%) y 31 (100%).
- `BrilloFlags` que recoge un puñado de banderas: el bit 0 (`BRCAMBIA`) indicará si el brillo es estático (0, brillo estático, un valor entre 25% y 100%) o está cambiando (1, brillo cambiando); el bit 1 es `BRDIR`, que indicará la dirección de cambio (0 para bajar brillo y 1 para subirlo); y el bit 2 es `BRON` que indica si la luz está encendida (1) o apagada (0).
- La pendiente de subida o bajada es de 100%/2s, lo que significa variar 32 tonos de brillo en 2 segundos. Para una señal PWM de 128Hz (128 ciclos en un segundo), significa variar 32 veces en 2*128 ciclos, o 1 vez cada 8 ciclos. Se usará la variable `BrilloCont` como un contador de ciclos de la señal PWM. Inicialmente se cargará a 8 y, cada vez que se entre en la ISR (si `BRCAMBIA=1`), se decrementará. Al llegar a 0, se incrementará/decrementará el brillo actual en función de `BRDIR` y se volverá a reponer `BrilloCont` a 8. Aquí también se controla que el brillo nunca suba del 100% ni baje del 25% y que se cambie la dirección cuando se llegue a alguno de los topes.

La gestión de la tecla debe tener en cuenta los posibles rebotes y el tiempo de pulsación. Si dicho tiempo es menor de 10ms, se considerará un rebote y se ignorará. Si es mayor de 10ms y menor de 250ms, será una pulsación breve y se cambiará el estado de la luz entre encendido y apagado. Nótese que esto tiene el efecto de conmutar la luz cuando se libera la tecla y no cuando se pulsa. Finalmente, si la pulsación es mayor, se entrará en modo dinámico (encendiendo la luz si estaba apagada) y procediendo al cambio del brillo haciendo `BRCAMBIA=1`. Para gestionar todos los posibles eventos, es necesario en un determinado momento detectar la liberación de la tecla y la expiración del tiempo máximo de pulsación corta. Esto sólo es posible teniendo el puerto P1.1 en función E/S digital con detección de flanco y el CCR2 en modo comparación esperando el tiempo fin de la pulsación corta.

Se procederá como sigue: inicialmente, P1.1 sensible en flanco de bajada y CCR2 desactivado. Cuando se detecte la pulsación de la tecla, se leerá en el CCR2 el valor actual del `TA0R`, se guardará este tiempo en una variable local `TIni`, se pasará a modo COMPARACIÓN y se le sumará el tiempo de pulsación corta (250 ms) expresado en tics. También se cambiará el flanco activo del puerto. En este estado se podrán producir dos eventos distintos: evento de comparación y liberación de la tecla. Si se produce primero el del *timer*, eso implica una pulsación larga, lo que nos llevaría a activar el modo dinámico de brillo (`BRCAMBIA=1`) y resetear el contador de brillo (`BrilloCont=8`). Si, en cambio, llega antes la liberación de la tecla, se desactivará la IRQ del CCR2, se medirá el tiempo que ha estado pulsada restando al valor actual del `TA0R`, el valor que se guardó en `TIni`. Si dicho tiempo es menor al de rebote, se ignorará la pulsación de la tecla. Si no, se trata de una pulsación corta y se conmutará el estado del led. Finalmente, si el tiempo es mayor del de una pulsación corta, ya se habrá entrado en la ISR del CCR2 que habrá activado el modo dinámico. Ahora sólo toca finalizar dicho modo y dejar el brillo memorizado en el estado en el que esté, haciendo `BRCAMBIA=0`.

```
;Constantes de configuración
LED         .equ   BIT0
PUL         .equ   BIT1

FACLK       .equ   32768         ;Frecuencia de ACLK. En Hz
FPWM        .equ   128           ;Frecuencia del PWM. En Hz
RESPWM      .equ   32            ;Resolución del PWM. En unidades
TREBOTE     .equ   10            ;Tiempo de rebote. En ms
TCORTA      .equ   250           ;Tiempo de pulsación corta. En ms

;Constantes calculadas
FTA         .equ   FPWM*RESPWM   ;Frecuencia del TA0. En Hz
DIVTA       .equ   FACLK/FTA     ;Divisor del TA0
NREBOTE     .equ   TREBOTE*FTA/1000;Tiempo de rebote (en tics de TACLK)
NCORTA      .equ   TCORTA*FTA/1000;Tiempo de pulsación corta (en tics de TACLK)
VBRILLO     .equ   2*FPWM/RESPWM;Velocidad de la pendiente de brillo
BRMIN       .equ   25*RESPWM/100;Brillo mínimo
BRMAX       .equ   RESPWM-1      ;Brillo máximo

;Variables
            .bss   TPul,2        ;Instante de pulsación de la tecla
            .bss   Brillo,2      ;Brillo actual si estuviera encendida
            .bss   BrilloCont,1  ;Contador de ciclos con mismo brillo
            .bss   BrilloFlag,1  ;Conjunto de flags de brillo (ver más abajo)
BRDIR       .equ   0             ;0:Bajando brillo; 1:subiendo brillo
BRON        .equ   1             ;0:luz apagada; 1:luz encendida
BRCAMBIA    .equ   2             ;0:Brillo estático; 1:brillo dinámico

;-------------------------------------------------------------------------------
;main                                                                      v2.0
;-------------------------------------------------------------------------------
main        ;Inicializar variables
            clr.b  &BrilloFlag   ;Luz apagada, bajando brillo, estático
            mov.b  #BRMAX, &Brillo;Brillo al 100%

            ;P1.1 entrada pulldown sensible a flanco de bajada (IRQ)
            ;bic.b #PUL, &P1DIR  ;Entrada
            bis.b  #PUL, &P1REN  ;Resistencia...
            bic.b  #PUL, &P1OUT  ;...de pulldown
```

```
            bis.b  #PUL, &P1IES  ;Sensible a flanco de bajada
            bic.b  #PUL, &P1IFG  ;Borrar flag
            bis.b  #PUL, &P1IE   ;Habilitar IRQ

            ;P1.0 salida función 1 (TA0.1)
            bis.b  #LED, &P1DIR  ;Salida
            bis.b  #LED, &P1SEL0 ;Función 1

            ;TA0.1 en modo comparación con salida PWM RESET-SET. Brillo al 0%
            mov.w  #OUTMOD_7, &TA0CCTL1
            clr.w  &TA0CCR1

            ;TA0.0 fijando excursión del TA0. IRQ
            mov.w  #CCIE, &TA0CCTL0
            mov.w  #RESPWM-1, &TA0CCR0

            ;TA0 con ACLK/DIVTA modo UP
            mov.w  #DIVTA-1, &TA0EX0;Divisor extra
            mov.w  #TASSEL__ACLK|ID__1|MC__CONTINUOUS|TACLR, &TA0CTL
            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; P1ISR                                                                   v2.0
;
;ISR del P1. Gestiona las pulsaciones y liberaciones de la tecla.
;Cuando se pulsa la tecla (flanco de bajada), programa el tiempo de pulsación corta
;en el TA0CCR2 y cambia el flanco activo para detactar la liberación.
;Cuando se libera la tecla, se calcula el tiempo que ha estado pulsada con la ayuda
;del CCR2. Si es menor del tiempo de rebote, se ignora. Si es menor del tiempo de
; pulsación corta, se conmuta el estado de la luz. Si es mayor, se cancela el
; modo de cambio de brillo.
;-------------------------------------------------------------------------------
P1ISR       xor.b  #PUL, &P1IES  ;Cambiar flanco activo
            bic.b  #PUL, &P1IFG  ;Borrar flag
            bit.b  #PUL, &P1IN   ;Tecla pulsada?
            jnz    TeclaNoPul    ;...no. Se ha liberado
TeclaPul    ;...sí. Tecla pulsada. Programar instante de pulsación corta
            mov.w  #CAP|CM_1|CCIS_2, &TA0CCTL2;CCR2 captura, flanco subida, ent. GND
            mov.w  #CAP|CM_1|CCIS_3, &TA0CCTL2;CCR2 captura, flanco subida, ent. VCC
            mov.w  &TA0CCR2, &TPul;CCR2=instante de pulsación. Guardar en TPul
            add.w  #NCORTA, &TA0CCR2;CCR2=fin de pulsación corta
            mov.w  #CCIE, &TA0CCTL2;CCR2 comparación, IRQ
            jmp    P1ISRFin      ;Salir
TeclaNoPul  ;Tecla liberada
            bic.w  #CCIE, &TA0CCTL2;Desactivar tenmporizador de tecla
            ;Calcular tiempo desde pulsación
            mov.w  #CAP|CM_1|CCIS_2, &TA0CCTL2;CCR2 captura, flanco subida, ent. GND
            mov.w  #CAP|CM_1|CCIS_3, &TA0CCTL2;CCR2 captura, flanco subida, ent. VCC
            sub.w  &TPul, &TA0CCR2;CCR2=tiempo desde pulsación
            cmp.w  #NREBOTE, &TA0CCR2;Ha pasado el tiempo de rebote?
            jlo    P1ISRFin      ;...no. Salir ignorando pulsación
            cmp.w  #NCORTA, &TA0CCR2;...sí. Ha pasado el tiempo de pulsación corta?
            jhs    FinCambio     ;...sí. Parar el cambio de brillo
            ;...no. Pulsación corta. Conmutar estado de la luz
            bit.b  #BRON, &BrilloFlags;Está la luz encendida?
            jnz    ApagarLuz     ;...sí. Apagar luz
            call   #EncenderLuz  ;...no. Encender luz
            jmp    P1ISRFin      ;Salir
ApagarLuz   push.w r4            ;Salvar R4
            mov.w  &TA0CCR0, r4  ;R4=Valor actual del brillo al 100%
            sub.w  #RESPWM-1, r4 ;R4=Valor actual del brillo al 0%
            mov.w  r4, &TA0CCR1  ;Apagar led
            pop.w  r4            ;Recuperar R4
            bic.b  #BRON, &BrilloFlags;Desactivar flag de encendido
            jmp    P1ISRFin      ;Salir
FinCambio   bic.b  #BRCAMBIA, &BrilloFlags;Desactivar flag de cambio de brillo
P1ISRFin    reti
```

```
;-------------------------------------------------------------------------------
; TA00ISR                                                                 v2.0
;
;ISR del CCR0. Gestiona la señal PWM en modo CONT y cambia el brillo si procede.
;Si el brillo es estático, no hace nada. Si hay que cambiar el brillo, primero
;se decrementa el número de ciclos en este brillo.
;Cuando se llegue a 0, cambiar en función de BrilloDir una unidad hasta llegar
;al máximo o al mínimo. En ese caso, dar la vuelta a BrilloDir.
;-------------------------------------------------------------------------------
TA00ISR    ;Actualizar CCRs para siguiente ciclo. Sólo hay que sumar la resolución
           ;módulo 16 bits. Descartar el posible desbordamiento
           add.w  #RESPWM, &TA0CCR1;Actualizar CCR1
           add.w  #RESPWM, &TA0CCR0;Actualizar CCR0
           ;Gestionar el cambio de brillo
           bit.b  #BRCAMBIA, &BrilloFlags;Qué modo está programado?
           jz     TA00ISRFin    ;...estático, salir
           dec.b  &BrilloCont   ;...dinámico. Decrementar contador de ciclos. Fin?
           jnz    TA00ISRFin    ;...no, salir
           mov.b  #VBRILLO, &BrilloCont;...sí, reponer contador de ciclos
           bit.b  #BRDIR, &BrilloFlags;Ver pendiente
           jz     PendNeg       ;...negativa
PendPos    inc.w  &TA0CCR1      ;...positiva. Incrementar brillo
           cmp.w  &TA0CCR0, &TA0CCR1;Ha llegado al 100%?
           jlo    TA00ISRFin    ;...no, salir
           jmp    CambiarDir    ;...sí, invertir dirección
PendNeg    dec.w  &TA0CCR1      ;Decrementar brillo
           push   r4            ;Salvar R4
           mov.w  &TA0CCR0, r4  ;R4=CCR0 (Brillo máximo actual)
           sub.w  #RESPWM-1+BRMIN, r4;R4=Valor mínimo que puede tomar CCR1 (25%)
           cmp.w  r4, &TA0CCR1  ;Ha llegado CCR1 al 25%?
           pop    r4            ;Recuperar R4
           jnz    TA00ISRFin    ;...no, salir
CambiarDir xor.b  #BRDIR, &BrilloFlags;...sí, cambiar dirección
TA00ISRFin reti

;-------------------------------------------------------------------------------
; TA01ISR                                                                 v2.0
;
;ISR del CCR2. Si se entra aquí es porque ha expirado el tiempo de pulsación corta.
;Activar el modo de brillo dinámico.
;-------------------------------------------------------------------------------
TA01ISR    bic.w  #CCIFG|CCIE, &TA0CCTL2;Borrar flag y desactivar IRQ
           bit.b  #BRON, &BrilloFlags;La luz está encendida?
           jnz    CambiaBril    ;...sí, sequir y activar cambio de brillo
           ;...no. La luz estaba apagada. Encender con el brillo actual
           call   #EncenderLuz
CambiaBril ;Activar cambio de brillo
           bis.b  #BRCAMBIA, &BrilloFlags;Activar cambio de brillo
           mov.b  #VBRILLO, &BrilloCont;Reponer contador de ciclos
           reti

;-------------------------------------------------------------------------------
; EncenderLuz                                                             v2.0
;
;Enciende la luz con el brillo memorizado. No altera ningún registro, para que
; se pueda llamar desde una ISR.
;-------------------------------------------------------------------------------
EncenderLuzpush.w r4            ;Salvar R4
           mov.w  &TA0CCR0, r4  ;R4=Valor actual del brillo al 100%
           sub.w  #RESPWM-1, r4 ;R4=Valor actual del brillo al 0%
           add.w  &Brillo, r4   ;R4=Nivel de brillo actualizado a este ciclo
           mov.w  r4, &TA0CCR1  ;Encender led al brillo actual
           pop.w  r4            ;Recuperar R4
           bis.b  #BRON, &BrilloFlags;Activar flag de encendido
           ret

           .intvecRESET_VECTOR, main
           .intvecPORT1_VECTOR, P1ISR
```

```
        .intvecTIMER0_A0_VECTOR, TA00ISR
        .intvecTIMER0_A1_VECTOR, TA01ISR
```

CRITERIO DE CORRECCIÓN

- main: 20%
- ISR P1.1: 40%
- ISR CCR0: 40%