SISTEMAS BASADOS EN MICROPROCESADOR **Primera convocatoria 2022/2023**

Apellidos:.....**SOLUCIÓN**..... | P1 | P2 | P3 |

Nombre:..... **Puesto:**.....

Duración 4:00 horas

1.- (2223 Feb) [3 Puntos] Analice el siguiente código e indique qué hace. Proponga un código alternativo que haga lo mismo, de forma más eficiente. 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
;-------------------------------------------------------------------------------
hb          mov.w  r12, r13     ;_________________|___|_____________________________
            mov.w  #0x8000, r12 ;_________________|___|_____________________________
hbb1        rra.w  r13          ;_________________|___|_____________________________
            jc     hbs          ;_________________|___|_____________________________
            rrc.w  r12          ;_________________|___|_____________________________
            jnc    hbb1         ;_________________|___|_____________________________
            jmp    hbf          ;_________________|___|_____________________________
hbb2        rra.w  r13          ;_________________|___|_____________________________
            xor.w  #1, sr       ;_________________|___|_____________________________
hbs         rrc.w  r12          ;_________________|___|_____________________________
            jnc    hbb2         ;_________________|___|_____________________________
hbf         ret                 ;_________________|___|_____________________________
```

SOLUCIÓN

a) Análisis:

```
hb          mov.w  r12, r13     ;R13=Parametro X
            mov.w  #0x8000, r12 ;R12=1000 0000 0000 0000
hbb1        rra.w  r13          ;Desplazar un bit a la derecha X. LSB al carry
            jc     hbs          ;Si el bit era 1, salir de este bucle
            rrc.w  r12          ;Si no, desplazar R12 metiendo 0 por la izqda
            jnc    hbb1         ;Si sale cero, cerrar bucle
            jmp    hbf          ;Si sale 1, hemos terminado. Salir
hbb2        rra.w  r13          ;Desplazar un bit a la derecha X. LSB al carry
            xor.w  #1, sr       ;Conmutar C
hbs         rrc.w  r12          ;Desplazar R12 a dcha metiendo C por izqda
            jnc    hbb2         ;Si ha salido 0, otra vuelta más
hbf         ret
```

La subrutina recibe un número de 16 bits en R12, que se copia a R13. Llamémosle X. En R12 se carga el número 0x8000. Después se entra en el bucle hbb1 que va sacando los bits de X de uno en uno por la derecha y los va metiendo por la izquierda en R12. Este proceso termina si el bit que sale de X es 1, en cuyo caso se va al bucle hbb2, o si el bit que sale de R12 es 1. Dado el número que se cargó en R12, esto sucederá después de 16 desplazamientos. Una vez que se detecta el primer 1 en X empezando por el lsb, se va al bucle hbb2 que es esencialmente igual al anterior, pero ahora los bits que salen de X son invertidos antes de entrar en R12. Se sale cuando encontramos el primer 1 en R12.

En resumen, se trata de un algoritmo que hace un bucle de 16 iteraciones que va copiando los bits de X a R12 hasta que se copia el primer 1. El resto de los bits se copian invertidos. Es decir, se está calculando el complemento a 2 de X. Dado que X entra por R12, 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 hb (int x);
```

donde x es un entero con signo al que se le calcula el complemento a 2, que se devuelve como un número con signo de 16 bits.

Este procedimiento, aunque válido, es muy ineficiente. Alternativamente, podríamos hacer:

```
hb          inv.w  r12           ;R12=Ca1(X)
            inc.w  r12           ;R12=Ca2(X)
hbf         ret
```

Que ocupa 3 palabras y tarda 5 ciclos en ejecutarse, incluyendo el retorno de subrutina.

Tiempo de ejecución:

```
;                               Emulación       Ciclos
;-------------------------------------------------------------------------------
hb          mov.w  r12, r13     ;                 | 1 |
            mov.w  #0x8000, r12 ;                 | 2 |
hbb1        rra.w  r13          ;                 | 1 |\
            jc     hbs          ;                 | 2 | | N iteraciones
            rrc.w  r12          ;                 | 1 | |
            jnc    hbb1         ;                 | 2 |/
            jmp    hbf          ;                 | 2 |
hbb2        rra.w  r13          ;                 | 1 |\
            xor.w  #1, sr       ;                 | 1 | | 16-N iteraciones
hbs         rrc.w  r12          ;                 | 1 | |
            jnc    hbb2         ;                 | 2 |/
hbf         ret                 ;mov.w @sp+,pc    | 3 |
```

Se ha comentado el código para expresar las instrucciones emuladas y el número de ciclos de cada instrucción. El bucle `hbb1` se ejecuta N veces, hasta que se copia el primer 1 de X. Una vez se localiza dicho 1 (si lo hubiere), el bucle `hbb2` se ejecuta 16-N veces. El número total de ciclos sería 3+6*N+5*(16-N)+3, donde N es igual al bit menos significativo que vale 1 en X más 1 o 16 si X=0. Dado que ambos bucles son muy parecidos en tiempo (6 y 5 ciclos respectivamente), podemos tomar un valor medio de ambos, quedando la fórmula 3+5'5*16+3=94 ciclos. El mejor caso se da cuando x es impar, tardando 92 ciclos. El peor cuando X=0 y tarda 104 ciclos (hay 2 ciclos extra no contemplados en la fórmula general debidos a la instrucción `jmp hbf` que salta al final). Si se sustituye por `ret`, se ahorran esos dos ciclos. Para un reloj de 1MHz, los tiempos mínimo/medio/máximo serían 92/94/104 μs.

Ensamblado:

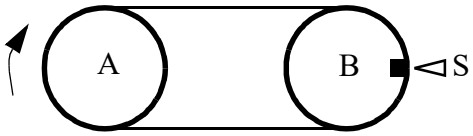
Se ha comentado el código para expresar la codificación de las instrucciones de tipo I (las que tienen 2 operandos), las de tipo II (las que tienen 1 operando) y las de salto relativo. En éstas sólo hay que codificar la condición y el desplazamiento, que se calcula como el número de palabras desde el origen (instrucción posterior al salto), al destino y que se anota en los comentarios. Cada instrucción ocupa una palabra excepto la segunda, que ocupa 2, dando un total de 13 palabras. Se han omitido, por claridad, las instrucciones repetidas.

```
;                               Emulación           Código máquina
;                                                   OpCd -Rf- DdBDf -Rd-(tipo I)
;                                                   000100 OpC BDf -Rd-(tipo II)
;                                                   001 Cnd Desplazami (salto)
;-------------------------------------------------------------------------------
hb          mov.w  r12, r13     ;                   0100 1100 0000 1101 = 4C0D
            mov.w  #0x8000, r12 ;                   0100 0000 0011 1100 = 403C 8000
hbb1        rra.w  r13          ;                   000100 010 000 1101 = 110D
            jc     hbs          ;+5                 001 011 0000000101  = 2C05
            rrc.w  r12          ;                   000100 000 000 1100 = 100C
            jnc    hbb1         ;-4                 001 010 1111111100  = 2BFC
            jmp    hbf          ;+4                 001 111 0000000100  = 3C04
hbb2        rra.w  r13          ;
            xor.w  #1, sr       ;                   1110 0011 0001 0010 = E312
hbs         rrc.w  r12          ;
            jnc    hbb2         ;-4
hbf         ret                 ;mov.w @r1+,r0      0100 0001 0011 0000 = 4130
```

CRITERIO DE CORRECCIÓN

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

2.- (2223 Feb) [3 Puntos] En el mecanismo de la figura, las ruedas A y B tienen un perímetro de 50cm y están conectadas por una correa que las hace mover de forma solidaria. La rueda A es movida por un motor que se activa con la señal M. La B dispone de un sensor S que permite contar el número de vueltas de la misma (suponga que inicialmente B está posicionado de forma que S=1; conforme B gira, S se desactivará y se volverá a activar después de hacer un giro completo; vea la figura). Además, hay un pulsador con salida P que puede ser accionado por el usuario.



Se desea realizar un controlador para el mecanismo anterior cuya funcionalidad sea la siguiente: cuando el usuario pulsa P, el controlador activa el motor para que la cinta se mueva 1m controlando el estado de S. Una vez completado el movimiento, el controlador parará el motor incluso si P=1 y se asegurará de que P=0 antes de iniciar otro posible ciclo con una nueva pulsación de P.

Realice un programa en ensamblador usando la técnica de multitarea cooperativa. Dispone del módulo st.asm y la subrutina cmp32. Considere perro guardián desactivado, los puertos desbloqueados, pila inicializada y LFXT en marcha con cristal de 32768Hz.

SOLUCIÓN

a) Usaremos el sensor S para determinar cuándo se da una vuelta. Como el perímtero de B es de 0'5m, el controlador deberá dar dos vueltas para avanzar la cinta 1m. Para gestionar la lógica de inicio (se debe pulsar P, pero se debe garantizar que se libere P antes de iniciar un nuevo ciclo), usaremos la siguiente máquina de estados:

A: Motor parado, se espera la pulsación de la tecla.

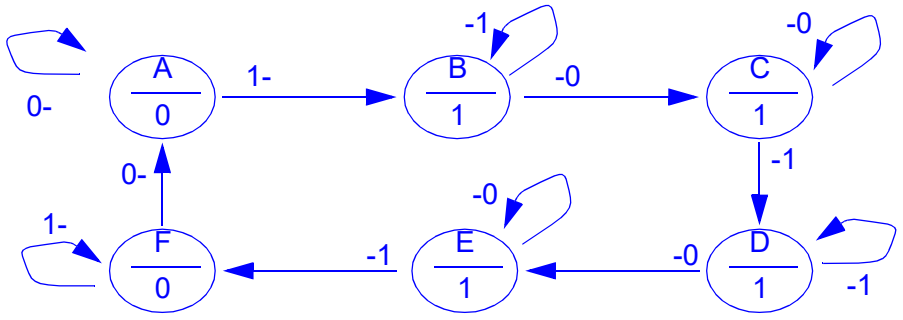
B: Motor funcionando. Se espera a que se desactive el sensor de posición S.

C: Motor funcionando. Se espera a que se active S (eso sería una vuelta).

D: Motor funcionando. Se espera a que se desactive S (inicio de la segunda vuelta).

E: Motor funcionando. Se espera a que se active S (fin de la segunda vuelta).

F: Motor parado, se espera la liberación de la tecla.



No es necesario repetir los estado B y C en D y E. Podría usarse en su lugar una variable que llevara la cuenta del número de vueltas que ha dado la rueda.

```
;-------------------------------
; Datos de configuración
;-------------------------------
; *** Puertos de E/S ***
P           .equ   BIT0
S           .equ   BIT1
```

```
M           .equ   BIT2

;Frecuencia de los distintos procesos. En Hz
FTECLA      .equ   100          ;Frecuencia de escaneo de la tecla
FLEDS       .equ    50          ;Frecuencia de parpadeo de leds

; *** Frecuencia del System Timer ***
FTA         .equ   32768        ;Frecuencia del reloj del TA. Hz
FRUEDA      .equ   100          ;Frecuencia del proceso de control de la ryueda. Hz
FST         .equ   100          ;Frecuencia del SystemTimer. Hz

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

;-------------------------------
; Variables
;-------------------------------
            .bss   ruedaPE, 4   ;Próxima Ejecución del prceso rueda
            .bss   EstRueda, 1  ;Estado del proceso
EA          .equ   0*2          ;Posibles valores de estado
EB          .equ   1*2          ;Posibles valores de estado
EC          .equ   2*2          ;Posibles valores de estado
ED          .equ   3*2          ;Posibles valores de estado
EE          .equ   4*2          ;Posibles valores de estado
EF          .equ   5*2          ;Posibles valores de estado
EULTIMO     .equ   EF           ;Posibles valores de estado

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

            ;Inicializar procesos
            call   #Inicializa

superbucle  call   #Rueda       ;Control de la rueda
            bis.w  #LPM3+GIE, sr ;Entrar en bajo consumo
            jmp    superbucle
            .intvec RESET_VECTOR, main
            .text

;-------------------------------------------------------------------------------
; Inicializa                                                              v1.0
;
; Inicializar proceso
;-------------------------------------------------------------------------------
Inicializa  clr.w  &ruedaPE     ;Ejecutar desde el principio
            clr.w  &ruedaPE+2
            mov.b  #EINI, &EstRueda
            ret

;-------------------------------------------------------------------------------
; Proceso Tecla                                                           v1.0
;-------------------------------------------------------------------------------
Rueda       call   #stTime      ;R13:12 = Ahora
            mov.w  &ruedaPE, r14 ;R15:R14=Instante de próxima ejecución
            mov.w  &ruedaPE+2, r15
            call   #cmp32       ;Comparar. Toca?
            jlo    ruedaFin     ;...no. Salir
            add.w  #PERRUEDA, &ruedaPE;...sí.Actualizar instante próxima ejecución
            adc.w  &ruedaPE+2

            ;Proceso útil de la tarea
```

```
            mov.b  &EstRueda, r14
            cmp.b  #EULTIMO+1, r14
            jhs    RdError
            add.w  r14, pc
            jmp    RdIni          ;Inicialización
            jmp    RdA            ;Estado A
            jmp    RdB            ;Estado B
            jmp    RdC            ;Estado C
            jmp    RdD            ;Estado D
            jmp    RdE            ;Estado E
            jmp    RdF            ;Estado F

            ;Estado Error. Desactivar tarea
RdError     mov.w  #-1, &ruedaPE ;Próxima ejecución... en el infinito
            mov.w  #-1, &ruedaPE+1
            bic.b  #M, &P1OUT     ;Parar motor
            jmp    ruedaFin
            ;Estado inicialización. Sensores como entrada
RdIni       ;bic.b #P|S, &P1DIR  ;Sensores como entrada
            mov.b  #EA, &EstRueda;Pasar a estado inicial real
            ;jmp   ruedaFin      ;Descomentar para esperar un ciclo
            ;Estado A. Esperar a que se pulse P
RdA         bit.b  #P, &P1IN      ;Tecla pulsada?
            jz     ruedaFin       ;...no. Salir
            mov.b  #EB, &EstRueda;...sí, cambiar de estado
            bis.b  #M, &P1OUT     ;Arrancar motor
            jmp    ruedaFin
            ;Estado B. Esperar a que se desactive S (primera vez)
RdB         bit.b  #S, &P1IN      ;Sensor de rueda?
            jnz    ruedaFin       ;...sí. Salir
            mov.b  #EC, &EstRueda;...no, cambiar de estado
            jmp    ruedaFin
            ;Estado C. Esperar a que se active S (primera vez)
RdC         bit.b  #S, &P1IN      ;Sensor de rueda?
            jz     ruedaFin       ;...no. Salir
            mov.b  #ED, &EstRueda;...sí, cambiar de estado
            jmp    ruedaFin
            ;Estado D. Esperar a que se desactive S (segunda vez)
RdD         bit.b  #S, &P1IN      ;Sensor de rueda?
            jnz    ruedaFin       ;...sí. Salir
            mov.b  #EE, &EstRueda;...no, cambiar de estado
            jmp    ruedaFin
            ;Estado E. Esperar a que se active S (primera vez)
RdE         bit.b  #S, &P1IN      ;Sensor de rueda?
            jz     ruedaFin       ;...no. Salir
            mov.b  #EF, &EstRueda;...sí, cambiar de estado
            bic.b  #M, &P1OUT     ;Parar motor
            jmp    ruedaFin
            ;Estado F. Esperar a que se suelte P
RdF         bit.b  #P, &P1IN      ;Tecla pulsada?
            jnz    ruedaFin       ;...sí. Salir
            mov.b  #EA, &EstRueda;...no, cambiar de estado
            ;jmp   ruedaFin

ruedaFin    ret
```

CRITERIO DE CORRECCIÓN

- Constantes, principal e inicializaciones: 20%
- Máquina de estados: 20%
- Programación de la máquina de estados: 60%

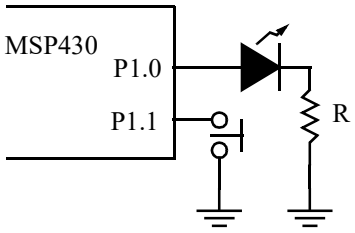
3.- (2223 Feb) [4 Puntos] Considere el circuito de la figura. Se trata de un sistema de luces con encendido/apagado y control de intensidad en una sola tecla con rebotes. Cada vez que se pulsa momentaneamente el *switch*, se conmuta el estado del led. Inicialmente el led se enciende con un 100% de su capacidad. 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 liberación del *switch* se paraliza el cambio de intensidad de la luz y se memoriza para posteriores encendidos/apagados.

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.

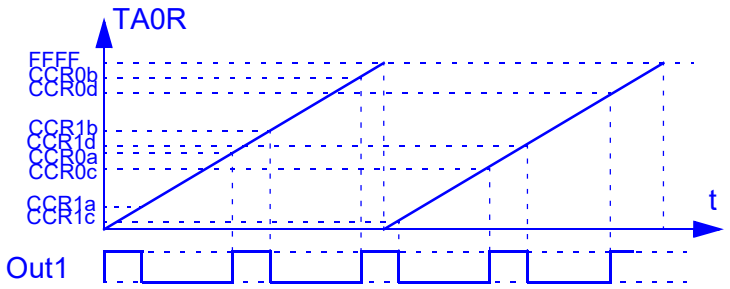
```
;Constantes de configuración
LED         .equ   BIT0
PUL         .equ   BIT1

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%