

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

P1	P2	P3

Duración 4:00 horas

- 1.- [3 Puntos] Analice el siguiente código e indique qué hace. Proponga un código alternativo que haga lo mismo, pero 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 $f_{MCLK}=1\text{MHz}$. Ensamble manualmente el programa y dé el código máquina en hexadecimal (excepto las instrucciones repetidas, que están en negrita). NOTA: la etiqueta C se refiere al bit de carry y está definida como BIT0.

		Emulación	Ciclos	Código máquina

vuas	mov.w r12, r13	;		
	mov.w #0x7FFF, r12	;		
vuasb1	rra.w r13	;		
	jc vuas2	;		
	rrc.w r12	;		
	jc vuasb1	;		
	jmp vuasfin	;		
vuasb2	rra.w r13	;		
	xor.w #C, sr	;		
vuas2	rrc.w r12	;		
	jc vuasb2	;		
vuasfin	ret	;		

SOLUCIÓN**a) Análisis:**

```

vuas      mov.w    r12, r13      ;Copia el parámetro de entrada en R13
          mov.w    #0x7FFF, r12;R12=0111 1111 1111 1111
vuasb1    rra.w    r13          ;Extrae el lsb de R13 y lo pasa al carry
          jc       vuas2        ;Es 1? Sí, ir a vuas2
          rrc.w    r12          ;...no, meter el carry por la izquierda en R12
          jc       vuasb1       ;Sale un 1 de R12? Sí, repetir una vuelta más
          jmp      vuasfin      ;...no. Hemos terminado. Han sido 16 iteraciones
vuasb2    rra.w    r13        ;Extrae el lsb de R13 y lo pasa al carry
          xor.w    #C, sr       ;Invertir el carry
vuas2     rrc.w    r12        ;meterlo por la izquierda en R12
          jc       vuasb2       ;Sale un 1 de R12? Sí, repetir una vuelta más
vuasfin    ret              ;...no. Hemos terminado. Han sido 16 iteraciones

```

Esta rutina va sacando los bits del parámetro de entrada por la derecha y los va introduciendo en otro registro por la izquierda. Cuando se encuentra el primer uno, se pasa a un segundo bucle que realiza el mismo proceso, pero introduce el inverso del bit extraído. El proceso se repite 16 veces. En resumen, se van copiando los bits de R13 a R12 (empezando por la derecha) hasta que se copia el primer 1. A partir de entonces se copian los bits invertidos. Éste es el algoritmo de la operación complemento a 2.

La función sigue el convenio de C, puesto que tanto la entrada como la salida se hace por R12 y no se alteran los registros R4 a R10. El prototipo sería:

```
uint16_t vuas(uint16_t n);
```

Otras formas de calcular el complemento a 2 son más eficientes porque hacen el proceso con la palabra al completo, no bit a bit. Se puede usar la definición básica de la operación (complemento a 1 más 1):

```

inv.w     r12      ;1w1c. Complemento a 1
inc.w     r12      ;1w1c. ...más 1

```

```
ret ;1w3c
```

que usa 3 palabras y 5 ciclos. O también se puede aprovechar que los números con signo usan la codificación en complemento a 2:

```
clr.w    r13    ;1w1c. R13=0
sub.w    r12, r13 ;1w1c. R13=0-R12=-R12=Ca2(R12)
mov.w    r13, r12 ;1w1c. R12=Ca2(R12)
ret      ;1w3c
```

que son 4 palabras y 6 ciclos. Se podría ahorrar 1 palabra y 1 ciclo si no es estrictamente necesario que la salida sea por R12, pero entonces no seguiría el convenio de llamada de C.

Tiempo de ejecución:

			Emulación	Ciclos

vuas	mov.w	r12, r13	;	1
	mov.w	#0x7FFF, r12	;	2
vuasb1	rra.w	r13	;	1 \
	jc	vuas2	;	2 N vueltas
	rrc.w	r12	;	1
	jc	vuasb1	;	2 /
	jmp	vuasfin	;	2
vuasb2	rra.w	r13	;	1 \
	xor.w	#1, sr	;	1 16-N vueltas
vuas2	rrc.w	r12	;	1
	jc	vuasb2	;	2 /
vuasfin	ret		;mov.w @sp+, pc	3

Sea N el índice primer bit empezando por la derecha que vale 1. El bucle vuasb1 da N vueltas. El resto hasta contar 16 las da el bucle vuasb2. Por tanto, el número total de ciclos de la subrutina es $2+6*N+5*(16-N)+3/5 = 5/7+16*5+N$. El código de salida tarda 5 ciclos si la palabra es 0 y 3 en el resto de los casos. Puesto que el primer bucle tarda 6 ciclos y el segundo (el que invierte los bits) sólo 5, la subrutina es más rápida cuanto antes se encuentre el primer 1, siendo crítico el caso en el que el parámetro de entrada sea 0, porque el bucle vuasb2 no se llega a ejecutar y hay que hacer un salto al final para saltarlo. En ese caso, N=16 y el número de ciclos total es 103. El mejor caso es cuando el lsb es 1, en cuyo caso tarda 89 ciclos. A 1MHz, los tiempos serían 89/103µs.

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, es decir, -4 para las dos jc vuasb*, que en complemento a 2 es 1111111100; +5 para jc vuas2 (0000000101) y +4 para jmp vuasfin (0000000100). Cada instrucción ocupa una palabra excepto la segunda que ocupa 2 al tener que almacenar el número en inmediato, dando un total de 13 palabras o 26 bytes.

			Emulación	Código máquina
				OpCd -Rf- DdBdf -Rd-(tipo I)
				000100 OpC Bdf -Rd-(tipo II)
				001 Cnd Desplazami (salto)

vuas	mov.w	r12, r13	;	0100 1100 0000 1101 = 4C0D
	mov.w	#0x7FFF, r12	;	0100 0000 0011 1100 = 403C 7FFF
vuasb1	rra.w	r13	;	000100 010 000 1101 = 110D
	jc	vuas2	; +5	001 011 0000000101 = 2C05
	rrc.w	r12	;	000100 000 000 1100 = 100C
	jc	vuasb1	; -4	001 011 1111111100 = 2FFC
	jmp	vuasfin	; +4	001 111 0000000100 = 3C04
vuasb2	rra.w	r13		
	xor.w	#1, sr	;	1110 0011 0001 0010 = E312
vuas2	rrc.w	r12		
	jc	vuasb2	; -4	001 011 1111111100 = 2FFC

CRITERIO DE CORRECCIÓN

- Análisis: 40% (instrucciones 20%, prototipo 20%, función 30%, alternativa 30%)
- Tiempo de ejecución: 20% (instrucciones 50%, cálculo total 50%)
- Codificación: 40%

2.- [3 Puntos] Realice en ensamblador del MSP430 las funciones `colaInserta` y `colaExtrae` que siguen la convención de llamada tipo C y atienden a los siguientes prototipos:

```
int colaInserta (cola_t *cola, uint8_t c);
int colaExtrae (cola_t *cola);
```

donde `cola` es el puntero a una cola y `c` es un elemento a insertar en la misma. `colaInserta` devuelve 0 si todo fue bien y -1 en caso de error. `colaExtrae` devuelve el elemento extraído si todo fue bien y -1 si la cola estaba vacía. El tipo `cola_t` se define como sigue:

```
typedef struct
{
    uint8_t *pini;           //Puntero a cabeza
    uint8_t *pfin;          //Puntero a cola
    uint8_t cola[TAMCOLA];   //La cola en sí
} cola_t;
```

Suponga que las colas se crean vacías y se inicializan de forma que `pini=pfin=cola[0]` para que ambos punteros apunten al primer elemento de la cola.

- Explique con palabras el algoritmo que usará para resolver el problema.
- Escriba `colaInserta`.
- Escriba `colaExtrae`.

SOLUCIÓN

a) La cola se gestionará circularmente, de forma que se evita el tener que moverla continuamente en cada extracción, así como buscar el final en las inserciones. Ello conlleva algunas precauciones para hacer la gestión correctamente: cada vez que se incrementa un puntero (`pini` o `pfin`) se comprobará si se ha salido del vector y, en ese caso, se conectará con el principio del mismo, haciendo el incremento circular. La condición de cola vacía es que `pini=pfin`. Y el de cola llena, `pini=pfin+1`.

b) El algoritmo primero comprueba que la cola no esté llena. Para ello calcula primero circularmente `pfin+1`. Si es igual a `pini`, sale con el código de error -1. En caso contrario, guarda el elemento `c` en el valor anterior de `pfin` y finalmente escribe el `pfin` incrementado en su sitio.

```
-----
; int colaInserta (cola *cola, uint8_t c)                                v1.0
;
; Inserta un elemento c en la cola. Devuelve 0 si todo fue bien o -1 en caso de
; error (cola llena).
-----
colaInserta:
    mov.w    r12, r11 ;R11=cola
    mov.w    2(r11), r15;R15=pfin
    mov.w    r15, r14 ;R14=pfin
    inc.w    r14      ;R14=pfin+1
    ;Asegurarse de que el incremento es circular
    mov.w    r11, r12 ;R12=Dirección inicio del objeto
    add.w    #TAMCOLA+4, r12;R12=Dirección de fin del objeto (+1)
    cmp.w    r12, r14 ;R14 se ha pasado?
    jlo      colaInsOk ;...no. Seguir
    sub.w    #TAMCOLA, r14;...sí, R14 apunta al inicio del vector
colaInsOk:
    ;Comprobar si cola llena. Si llena, salir con error
    mov.w    #-1, r12 ;por defecto, devolver error
    cmp.w    @r11, r14 ;pini = pfin+1?
    jeq      colaInsFin;...sí, cola llena. Salir
    ;
```

```

mov.b    r13, 0(r15);...no. Insertar elemento por el final
mov.w    r14, 2(r11);Guardar pfin+1 en la cola
clr.w    r12      ;Operación OK
colaInsFin ret

```

c) La función primero comprueba si la cola está vacía ($pini=pfin$). En ese caso, sale con el código de error (-1). En caso contrario, lee el elemento apuntado por $pini$ e incrementa circularmente dicho puntero antes de guardarlo en su sitio.

```

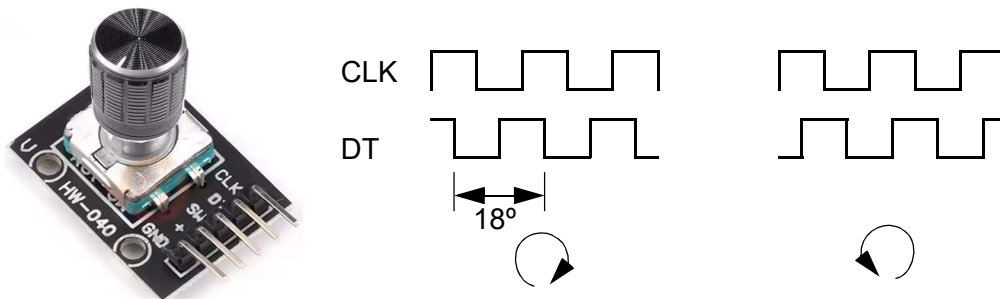
;-----
; int colaExtrae(cola *cola)                                v1.0
;
; Extrae un elemento de la cola. Devuelve el elemento si todo fue bien o -1 en caso de
; error (cola vacia).
;-----
colaExtrae mov.w    r12, r11 ;R11=cola
mov.w    #-1, r12 ;Por defecto devolver error
mov.w    @r11, r14 ;R14=pini
cmp.w    r14, 2(r11);pini = pfin?
jeq      colaExtFin;...sí, cola vacía. Salir
mov.b    @r14+, r12;...no. R12=elemento, R14=pini+1
;Asegurarse de que el incremento es circular
mov.w    r11, r15 ;R15=Dirección inicio del objeto
add.w    #TAMCOLA+4, r15;R15=Dirección de fin del objeto (+1)
cmp.w    r15, r14 ;R14 se ha pasado?
jlo      colaExtOk ;...no. Sequir
sub.w    #TAMCOLA, r14;...sí, R14 apunta al inicio del vector
colaExtOk mov.w    r14, 0(r11);Guardar pini+1 en la cola
colaExtFin ret

```

CRITERIO DE CORRECCIÓN

- colaInserta 50%
- colaExtrae 50%

3.- [4 Puntos] El KY-040 es un encoder diferencial de 20 pulsos por vuelta. Se trata de un dispositivo con un eje accionable que proporciona 2 salidas indicando la dirección y velocidad de giro (CLK y DT). Además, dispone de un pulsador asociado al eje Z (salida SW) y dos terminales de alimentación (+ y GND).



Es fácil determinar sentido/velocidad de giro (vea la figura). Así, se gira sentido horario si, al detectar un flanco en CLK, ambas salidas son iguales ($CLK=DT$), mientras que si son complementarias, el sentido es el contrario al de las agujas del reloj. La velocidad de giro se puede medir sabiendo que un ciclo completo de CLK o DT se produce después de girar 18° el eje (20 ciclos por vuelta).

Se ha conectado un KY-040 al Launchpad del MSP430: CLK en P1.1, DT en P1.2 y SW en P1.3, además de los leds rojo (P1.0) y verde (P9.7) habituales.

Realice un programa en ensamblador del MSP430 usando programación por interrupciones y con el mínimo consumo posible que controle el brillo de ambos leds con la ayuda del encoder y los temporizadores. Inicialmente los dos leds están apagados y el led activo es el rojo. Girando el encoder en sentido horario es posible aumentar el brillo del led activo. Se baja el brillo girando en sentido antihorario. Si la velocidad de giro es menor a 60 rpm, cada pulso del encoder cambiará un 1% el brillo. En cambio, si la velocidad es mayor, cada pulso cambiará un 5%. Cada vez que se pulse el eje hacia dentro, se cambiará el led activo.

- La resolución del brillo es de un 1%. Use una frecuencia mínima de 64Hz para evitar ver el parpadeo de los leds.
- Suponga que la función primaria de P1.0 es TA0.1 y de P9.7 es TA0.2. La función secundaria de P1.1 es TA1.0 (CCI1A).
- CLK y DT tienen resistencias de *pullup* a Vcc, pero no SW.
- Suponga SW libre de rebotes.
- Considere el perro guardián desactivado, los puertos desbloqueados, ACLK funcionando a 32768Hz y la pila inicializada.

SOLUCIÓN

El control del brillo de los leds se hará con salidas PWM. Se requiere una resolución de 1% y un fondo de escala de 100, así que TA0CCR0=99. Para que el parpadeo no se note se requiere que la frecuencia de la señal PWM sea de al menos 64 Hz. Atendiendo a la fórmula que liga resolución y frecuencia, podemos calcular la velocidad del timer (>6400 Hz) y, por tanto, su divisor ($32768/6400=5.12$). Se ha escogido un divisor de 5 que proporciona una $F_{PWM}=6554$, que cumple las especificaciones.

$$F_{TA0} = BRILLOMAX \times F_{PWM}$$

Inicialmente se configurarán P1.0 (led rojo) y P9.7 (led verde) como salidas en sus funciones primarias (TA0.1 y TA0.2 respectivamente) para que sean controlados por los CCR1 y CCR2 del TA0. El valor inicial de ambos será un brillo de 0%. El bit 0 de la variable `LedActual` indicará el led activo y se inicializará a 0. El control del brillo de ambos leds se hace por PWM con la ayuda del TA0, sin intervención del software, más allá de indicar cuál va a ser el valor del brillo.

El P1.3 se inicializará como entrada con resistencia de *pullup* sensible en flanco de bajada y con las interrupciones activadas. La ISR correspondiente (`P1ISR`) se encargará de conmutar el led activo cada vez que se pulse el eje del encoder (variable `LedActivo`).

La medida del encoder se hará con la ayuda del TA1.0, configurando el P1.1 como entrada en su función secundaria (CCI1A) y sensible al flanco de bajada. El CCR0 del TA1 estará en modo captura para medir el instante del flanco de bajada de la señal CLK y se usará para medir el tiempo entre flancos. En función del mismo se variará el brillo del led activo en 1 o 5. Todo el proceso se hace dentro de la ISR del CCR0 del TA1. En cada flanco de bajada de CLK se calculará el tiempo que ha pasado desde el flanco anterior (`TFAnt`) y se actualizará dicha variable para la siguiente medida. En función del tiempo pasado se guardará en R5 1 o 5. También se determinará cuál es el led activo y se dejará en R4 la dirección de su CCR para posterior tratamiento. Después se leerá el valor de DT para determinar la dirección de giro. Finalmente, el led actual es incrementado/decrementado en el valor correspondiente asegurándose que no se pasa del valor válido de brillo [0, 100].

;Constantes de configuración

```
LEDR_B      .equ    BIT0      ;Led rojo. Puerto y bit
LEDR_P      .equ    P1IN
LEDV_B      .equ    BIT7      ;Led verde. Puerto y bit
LEDV_P      .equ    P9IN
CLK_B       .equ    BIT1      ;Salida CLK del enconder. Puerto y bit
CLK_P       .equ    P1IN
DT_B        .equ    BIT2      ;Salida DT del enconder. Puerto y bit
DT_P        .equ    P1IN
SW_B        .equ    BIT3      ;Salida SW del enconder. Puerto y bit
SW_P        .equ    P1IN

FACLK       .equ    32768     ;Frecuencia de ACLK. En Hz
RPM         .equ    60       ;Revoluciones del enconder para distinguir velocidad
PPV         .equ    20       ;Pulsos por vuelta del encoder
BRILLOMAX   .equ    100      ;Máximo nivel de brillo

DIVPRI0     .equ    1        ;Divisor primario del TA0
DIVSEC0     .equ    5        ;Divisor secundario del TA0
```

;Constantes calculadas

```
FTA0        .equ    FACLK/(DIVPRI0*DIVSEC0) ;Frecuencia del TA0. En Hz
```

[illegible]

```

TA10ISR    push.w r4                ;No alterar registros en una IRQ
           push.w r5
           ;Calcular valor del cambio de brillo en función del tiempo pasado
           ;desde último flanco.
           mov.w  &TA1CCR0, r4      ;R4=Ahora
           sub.w  &TFAnt, r4        ;R4=Tiempo entre flancos
           mov.w  &TA1CCR0, &TFAnt;Guardar instante flanco actual para sig medida
           mov.w  #1, r5            ;Incrementar/decrementar brillo en 1 por defecto
           cmp.w  #NRPM, r4         ;Es mayor o igual que la velocidad incrementada?
           jlo    TA10ISRLED        ;...no. Inc/dec en 1
           mov.w  #5, r5            ;...sí. Inc/dec en 5
TA10ISRLED ;Cargar en R4 la dirección del led que hay que cambiar el brillo
           mov.w  #TA0CCR0, r4      ;Por defecto, cambiar brillo del led rojo
           bit.b  #BIT0, &LedActivo;Qué led hay que variar?
           jz     TA10ISRDir        ;...Rojo. Seguir
           mov.w  #TA0CCR1, r4      ;...verde. Cargar en R4 su dirección
TA10ISRDir ;Se ha producido un flanco de bajada de CLK. Si DT=0, incrementar
           ;brillo del led activo. Si DT=1, decrementar
           bit.b  #DT_B, &DT_P+PIN;DT=0?
           jz     TA10ISRInc        ;...sí, incrementar brillo
           ;Decrementar el brillo sin pasarse
TA10ISRDec inv.w  r5                ;R5=-R5
           inc.w  r5
           add.w  @r4, r5           ;R5=Brillo final
           jhs    TA10ISREsc        ;Brillo positivo o 0? Escribir
           clr.w  r5                ;...negativo. Mínimo 0
           jmp    TA10ISREsc        ;Escribir
TA10ISRInc ;Incrementar el brillo sin pasarse
           add.w  @r4, r5           ;R5=Brillo final
           cmp.w  #BRILLOMAX, r5;Brillo menor a 100%?
           jlo    TA10ISRDec        ;...sí. Escribir
           mov.w  #BRILLOMAX, r5;...no. Saturar
TA10ISREsc mov.w  r5, 0(r4)         ;Establecer nuevo brillo
           pop.w  r5
           pop.w  r4
           reti

.intvecRESET_VECTOR, main
.intvecPORT1_VECTOR, P1ISR
.intvecTIMER1_A0_VECTOR, TA10ISR

```

CRITERIO DE CORRECCIÓN

- Definiciones e inicialización (main): 30% (Variables 10%, leds 10%, enconder 20%, TA0 30%, TA1 30%)
- P1ISR: 10%
- ISR TA1.0: 60%