SISTEMAS BASADOS EN MICROPROCESADOR **Segunda Convocatoria 2022-2023**

Apellidos:.....SOLUCIÓN.....

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

P1	P2	P3

Duración 4:00 horas

1.- [3 Puntos] Preguntas cortas:

a) La instrucción `NOP` no hace nada, pero ocupa una palabra de código y tarda un ciclo en ejecutarse. Proponga una solución alternativa que ocupe una palabra y pierda 2 ciclos.

b) Cuando se produce una interrupción, el procesador termina de ejecutar la instrucción en curso y, si las interrupciones están habilitadas, inicia el proceso de servir la más prioritaria de las pendientes. Por tanto, una instrucción en sí misma es atómica, ya que no se puede dividir. Sin embargo, un trozo de código cualquiera formado por más de una instrucción no es atómico. Indique cómo lo convertiría en atómico.

c) Escriba las instrucciones necesarias para que la frecuencia del ACLK sea 1'17 kHz (NOTA: Flfxt=32768Hz, Fvlo=9400Hz, Flfmodclk=37500Hz).

d) Haga un programa en ensamblador que tenga dos tareas cooperativas. La primera hace parpadear el led1 del *Launchpad* a 3 Hz (es decir, se apaga y se enciende 3 veces en un segundo) y la segunda hace lo propio con el led2 a 7Hz. Dispone del módulo `st.asm` desarrollado en prácticas. Escoja la mínima frecuencia del TA2 para que el error de las frecuencias esté por debajo del 0'1%. Led1 en P1.0; Led2 en P9.7.

SOLUCIÓN

a) Hay muchas alternativas. Por poner algunas:

- JMP <siguiente instrucción>
- mov.w pc, pc
- mov.w @r0, r3

Es importante que no se altere el SR, ya que estrictamente, no debe hacer nada.

b) Basta con deshabilitar las interrupciones antes del bloque de instrucciones y restaurar su estado al final. De esa forma no se puede ejecutar una IRQ entre dos instrucciones cualesquiera del bloque. Antes de deshabilitar las interrupciones, se guarda el estado del SR en un registro (si está disponible) o la pila (solución más lenta):

```
mov.w     sr, r15   ;Salvar GIE (y resto de SR) en R15
;push.w   sr        ;Versión alternativa si no hay un registro disponible
dint                ;Deshabilitar interrupciones
;inicio del bloque atómico
;...
;fin del bloque atómico
mov.w     r15, sr   ;Recuperar estado original del GIE
;pop.w    sr        ;Versión alternativa si no hay un registro disponible
```

c) EL ACLK sólo puede tener como fuentes los osciladores lentos, LFXT, VLO y LFMODCLK. Cualquiera de esos relojes está en el rango de los 10KHz mínimo, por lo que habrá que usar un divisor para conseguir una velocidad tan lenta. Tenemos tres opciones:

- LFXT / 1,17 = 32'768 / 1,17 = 28'01
- VLO / 1,17 = 9'4 / 1,17 = 8,03
- LFMODOSC / 1,17 = 37'5 / 1,17 = 32,05

El divisor para LFXT se sale de rango, ya que sólo son válidos divisores que sean potencia de 2 (1, 2, 4, 8, 16 y 32). En cambio, los otros dos osciladores dan aproximaciones válidas a un divisor. Dada la variablidad de ambos osciladores, no podemos dar ninguna de las dos opciones como mejor respuesta, siendo ambas igualmente válidas. La única razón que puede decantar la elección de la tercera opción es que el VLO está por defecto apagado y el LFMODCLK encendido. No obstante, el consumo del VLO es ridículo, por lo que encenderlo

no supondría más que un incremento del consumo de 100nA. En el siguiente código se muestra la configuración de la tercera opción.

```
;Descomentar instrucciones si no se parte del estado de reset
mov.b  #CSKEY_H, &CSCTL0_H;Desbloquear módulo de reloj
bis.w  #SELA1, &CSCTL2;Hacer LFMODCLK fuente de ACLK
;bic.w #SELA2|SELA0, &CSCTL2;Hacer LFMODCLK fuente de ACLK
bis.w  #DIVA2|DIVA0, &CSCTL3;DIVA=5 (dividir por 32)
;bic.w #DIVA1, &CSCTL3;DIVA=5 (dividir por 32)
clr.b  &CSCTL0_H     ;Bloquear módulo de reloj
```

d) Escogeremos como frecuencias para las tareas 6 y 14, el doble de las frecuencias según el enunciado, para contabilizar tanto los encendidos como los apagados.

Si se escoge el mínimo común multiplo de las frecuencias de las tareas como frecuencia del SystemTimer (ST), el error en las frecuencias vendrá determinado por la precisión con la que se ajuste la frecuencia del ST en el timer A.

Por tanto, el error en las frecuencias de las tareas dependerá del divisor que se aplique al reloj del timer A. En la siguiente tabla se pueden encontrar los cálculos para todos los posibles divisores que se pueden aplicar (hay algunos que se pueden obtener de varias formas al tener 2 divisores separados):

Divisor	FTA (Hz)	CCR0	Freal (Hz)	Error	Válido
1	32768	779	42,01026	0,02%	VERDADERO
2	16384	389	42,01026	0,02%	VERDADERO
3	10922,67	259	42,01026	0,02%	VERDADERO
4	8192	194	42,01026	0,02%	VERDADERO
5	6553,6	155	42,01026	0,02%	VERDADERO
6	5461,333	129	42,01026	0,02%	VERDADERO
7	4681,143	110	42,17246	0,41%	FALSO
8	4096	96	42,2268	0,54%	FALSO
10	3276,8	77	42,01026	0,02%	VERDADERO
12	2730,667	64	42,01026	0,02%	VERDADERO
14	2340,571	54	42,55584	1,32%	FALSO
16	2048	47	42,66667	1,59%	FALSO
20	1638,4	38	42,01026	0,02%	VERDADERO
24	1365,333	31	42,66667	1,59%	FALSO
28	1170,286	26	43,34392	3,20%	FALSO
32	1024	23	42,66667	1,59%	FALSO
40	819,2	18	43,11579	2,66%	FALSO
48	682,6667	15	42,66667	1,59%	FALSO
56	585,1429	12	45,01099	7,17%	FALSO
64	512	11	42,66667	1,59%	FALSO

En principio, cuando más pequeño sea dicho divisor, más grande será el valor del CCR0 y menor influencia va a tener el truncamiento de los decimales en el cálculo del CCR0. Pero también es posible encontrar errores mayores con divisores menores: el error para el divisor 8 es de 0'54%, mientras que para el divisor 20 es de 0'02% (o 200 ppm). De hecho, el divisor de 20 es la mejor elección porque es el mayor que produce un error menor al 0'1% como se pide.

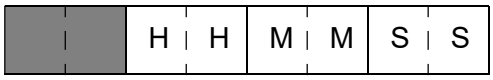
```
;-------------------------------
; Datos de configuración
;-------------------------------
; *** Puertos de E/S ***
L1          .equ   BIT0
L2          .equ   BIT7

; *** Frecuencia del System Timer ***
FTACLK      .equ   32768              ;Frecuencia del reloj del TA. Hz
FL1         .equ   6                  ;Frecuencia de eventos del led1
FL2         .equ   14                 ;Frecuencia de eventos del led2
FST         .equ   42                 ;Frecuencia del SystemTimer. Hz

;-------------------------------
; Constantes calculadas
;-------------------------------
CCR0        equ    FTACLK/FST-1       ;Valor a programar en CCR0
l1PERIODO   equ    FST/FL1            ;Tiempo entre ejecuciones del L1(TICs)
l2PERIODO   equ    FST/FL2            ;Tiempo entre ejecuciones del L2(TICs)

;-------------------------------
; Variables
;-------------------------------
            .bss   l1PE, 4            ;Próxima Ejecución del proceso del led 1
            .bss   l2PE, 4            ;Próxima Ejecución del proceso del led 2

;-------------------------------------------------------------------------------
; main                                                                    v1.0
;-------------------------------------------------------------------------------
main        mov.w  #CCR0, r12
            call   #stIni             ;Inicializar System Timer
            call   #IniVars           ;Inicializar variables
            call   #IniPuertos        ;Inicializar puertos E/S

superbucle  call   #Led1              ;Tarea de parpadeo del led1
            call   #Led2              ;Tarea de parpadeo del led1
            bis.w  #LPM3+GIE, sr      ;Entrar en bajo consumo
            jmp    superbucle
            .intvec RESET_VECTOR, main
            .text

;-------------------------------------------------------------------------------
; IniVars                                                                 v1.0
;-------------------------------------------------------------------------------
IniVars     ;Inicializar variables del sistema
            clr.w  &l1PE              ;Ejecutar desde el principio
            clr.w  &l1PE+2
            clr.w  &l2PE              ;Ejecutar desde el principio
            clr.w  &l2PE+2
            ret

;-------------------------------------------------------------------------------
; IniPuertos                                                              v1.0
;
; Inicializa los puertos de E/S.
;-------------------------------------------------------------------------------
IniPuertos  bic.b  #L1, &P1OUT        ;L1 apagado
            bic.b  #L2, &P9OUT        ;L2 apagado
            bis.b  #L1, &P1DIR        ;L1 salida
            bis.b  #L2, &P9DIR        ;L2 salida
            ret
```

```
;-------------------------------------------------------------------------------
; Proceso del led 1                                                        v1.0
;-------------------------------------------------------------------------------
Led1        call   #stTime             ;Leer tiempo
            mov.w  &l1PE, r14          ;R15:R14=Instante de próxima ejecución
            mov.w  &l1PE+2, r15
            call   #cmp32              ;Comparar. Toca?
            jlo    Led1Fin             ;...no. Salir
            add.w  #l1PERIODO, &l1PE   ;...sí.Actualizar instante próxima ejecución
            adc.w  &l1PE+2

            xor.b  #L1, &P1OUT         ;Conmutar led
Led1Fin     ret

;-------------------------------------------------------------------------------
; Proceso del led 2                                                        v1.0
;-------------------------------------------------------------------------------
Led2        call   #stTime             ;Leer tiempo
            mov.w  &l2PE, r14          ;R15:R14=Instante de próxima ejecución
            mov.w  &l2PE+2, r15
            call   #cmp32              ;Comparar. Toca?
            jlo    Led2Fin             ;...no. Salir
            add.w  #l2PERIODO, &l2PE   ;...sí.Actualizar instante próxima ejecución
            adc.w  &l2PE+2

            xor.b  #L2, &P9OUT         ;Conmutar led
Led2Fin     ret
```

2.- **[3 Puntos]** Se desea realizar la función `IncTime` que recibe una información de hora y la incrementa en un segundo:

```
int IncTime (unsigned long *hms);
```

`hms` es un puntero a un entero largo sin signo que almacena, en BCD empaquetado, información temporal en formato HHMMSS (dos dígitos BCD para hora, minutos y segundos):



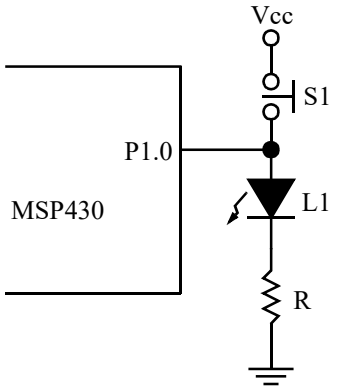
La función sigue el convenio de llamada de C y devuelve 1 si se produce un cambio de día (la hora de entrada es 23:59:59), -1 si la hora es incorrecta y 0 en el resto de los casos. Ignore el byte alto de la información horaria (zona rayada). La hora se guarda en formato de 24h. Escriba `IncTime` en ensamblador del MSP430. NOTA: La nueva hora se guarda en el mismo sitio que la original. Se recomienda el uso de las instrucciones que soporten aritmética BCD.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int IncTime (unsigned long *hms);                                  v1.0
;
; Incrementa la hora hms (en BCD empaquetado y en formato de 24h) en un
; segundo. En caso de cambio de día (hms=235959) devuelve 1. En caso de
; error (hora incorrecta) devuelve -1. En caso contrario, 0.
;-------------------------------------------------------------------------
IncTime     mov.w  r12, r13       ;Dejar libre R12 para la salida
            clr.w  r12            ;Por defecto, salir con Ok
            clrc                  ;Empezar con C=0

            ;Incrementar SS
            dadd.b #1, 0(r12)     ;Sumar 1 a SS
            cmp.b  #0x60, 0(r12) ;SS <=> 60?
            jlo    IncTimeFin     ;<, trabajo terminado
            jeq    IncTimeMM      ;=, incrementar minutos
            jmp    IncTimeErr     ;>, error

            ;Hacer SS=0 e incrementar MM
IncTimeMM   clr.b  0(r12)         ;SS=0
            dadd.b #1, 1(r12)     ;Sumar 1 a MM
```

```
            cmp.b  #0x60, 1(r12) ;MM <=> 60?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeHH     ;=, incrementar minutos
            jmp    IncTimeErr    ;>, error

            ;Hacer MM=0 e incrementar HH
IncTimeHH   clr.b  1(r12)        ;MM=0
            dadd.b #1, 2(r12)    ;Sumar 1 a HH
            cmp.b  #0x24, 2(r12) ;HH <=> 24?
            jlo    IncTimeFin    ;<, trabajo terminado
            jeq    IncTimeDD     ;=, incrementar minutos
            jmp    IncTimeErr    ;>=, error

            ;Hacer HH=0 y salir con cambio de día
IncTimeDD   clr.b  2(r12)        ;HH=0
            mov.w  #1, r12       ;Código de salida=1 (nuevo día)
            jmp    IncTimeFin

IncTimeErr  mov.w  #-1, r12      ;Salida Error
IncTimeFin  ret
```

3.- **[4 Puntos]** Observe el circuito de la figura. Inicialmente el puerto se configurará en modo entrada para poder leer el estado del pulsador. Cuando el usuario lo acciona, se enciende el led sin intervención del MSP. Se desea hacer un programa que alargue el encendido del led hasta 3s cuando la pulsación sea menor a ese tiempo. Para ello, se medirá el tiempo de la pulsación y, si es menor, colocará el puerto en modo salida y encenderá el led el tiempo restante hasta completar los 3s. Nótese que el led se apagará brevemente desde que se suelte la tecla hasta que se encienda por programa. Una vez terminado el tiempo, volverá a colocar el puerto en modo entrada para esperar la siguiente pulsación. En caso de que la pulsación fuera de 3s o más, no se hará ninguna actuación. El programa en ensamblador debe funcionar totalmente por interrupciones, con ayuda del Timer A0 y operar en el modo de mayor bajo consumo. Considere los pulsadores libres de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada, ACLK=LFXT en marcha con cristal de 32768Hz y TA0.CCR0 como función primaria del P1.0, CCI0A cuando es entrada y OUT0 cuando es salida.



SOLUCIÓN

Inicialmente se configurará el puerto como entrada con resistencia de *pull-down*. La tecla pulsada se detectará con un valor alto (lógica positiva). El control del led es con lógica positiva (1 enciende, 0 apaga).

Además se configurará en su función primaria para que sea controlado por el TA0 directamente. El CCR0 se configurará en modo de captura sensible a flanco de subida en el canal CCIA con las interrupciones habilitadas.

El TA0 se configurará con entrada de ACLK, en modo continuo con el divisor máximo (64), ya que permite medir con exactitud el tiempo de mínimo de 3s.

El CCR0 puede estar en tres modos distintos:

- Captura, sensible a flanco de subida (modo inicial). Se espera a la pulsación de la tecla. En la ISR se guarda el valor del CCR0 (instante actual) más 3s en la variable `TFin` para poder calcular el momento límite del encendido mínimo. Se cambiará el flanco activo a

bajada, para detectar la liberación.

- Captura, sensible a flanco de bajada. Se espera a la liberación de la tecla. En la ISR se mira si el tiempo actual (capturado en el CCR0) es mayor o igual que el instante final de tiempo mínimo de encendido (guardado en la variable `TFin`). Si es así, no se actúa sobre el led y sólo se configura el TA0 para la siguiente pulsación. Si no, se pone el puerto como salida y se enciende el led manualmente. También se configura el CCR0 en modo de comparación para que produzca un evento al final de dicho tiempo `TFin` y apague el led automáticamente.
- Comparación. Se espera al fin de los 3s desde la pulsación. En la ISR sólo hay que poner el puerto en modo entrada y configurar el CCR0 en modo captura en flanco de subida para detectar la siguiente pulsación.

```
PUERTO      .equ   BIT0

FACLK       .equ   32768         ;Frecuencia de ACLK. En Hz
DIVTA       .equ   64            ;Divisor del TA0
FTA         .equ   FACLK/DIVTA   ;Frecuencia del TA0. En Hz
TMINLED     .equ   3*FTA         ;Tiempo mínimo de led encendido (en TICS)
            .bss   TFin, 2       ;Guarda instante final de tiempo mínimo de encendido

;-------------------------------------------------------------------------------
;main                                                                      v1.0
;-------------------------------------------------------------------------------
main        ;P1.0 entrada pulldown función 1 (TA0.0)
            bis.b  #PUERTO, &P1REN;Resistencia...
            bic.b  #PUERTO, &P1OUT;...de pulldown
            bis.b  #PUERTO, &P1SEL0;Función 1

            ;TA0.0 en modo captura con entrada CCIA en flanco de subida. IRQ
            mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL0

            ;TA0 con ACLK/64 modo continuo
            mov.w  #7, &TA0EX0   ;Divisor extra a 8
            mov.w  #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLR, &TA0CTL
            bis.w  #LPM3|GIE, sr ;Ea, a dormir

;-------------------------------------------------------------------------------
; TA00ISR                                                                  v1.0
;-------------------------------------------------------------------------------
TA00ISR     ;Tres posibles eventos de entrada:
            ;1. Flanco de subida en CCIA. Se ha pulsado la tecla
            ;2. Flanco de bajada en CCIA. Se ha soltado la tecla
            ;3. Evento de comparación. Se ha terminado el tiempo minimo
            bit.w  #CAP, &TA0CCTL0;Qué modo está programado?
            jz     Comparac      ;...comparación; led se apaga sólo. Esperar pulsación
            bit.w  #CM0, &TA0CCTL0;...captura. Qué flanco está programado?
            jz     FBajada       ;...bajada. Se ha soltado la tecla
            ;Si flanco subida, calcular fin del tiempo mínimo de encendido
FSubida     mov.w  &TA0CCR0, &TFIn;TFin=Instante de pulsación
            add.w  #TMINLED, &TFin;TFin=Instante final de tiempo mínimo de encendido
            xor.w  #CM1|CM0, &TA0CCTL0;Conmutar flanco de entrada (sensible a bajada)
            jmp    TA00Fin
            ;Si es flanco de bajada, ver si se ha cumplido el tiempo mínimo
FBajada     cmp.w  &TA0CCR0, &TFin;Hemos llegado al tiempo mínimo de encendido?
            jlo    Comparac2     ;...sí. Esperar siguiente pulsación
            ;...no. Encender led manualmente y programar apagado tras el tiempo mínimo
            mov.w  #OUTMOD_0|OUT, &TA0CCTL0;Pasar a modo comparación y encender led
            bis.b  #PUERTO, &P1DIR;Puerto salida
            mov.w  &TFin, &TA0CCR0;Programar tiempo de apagado
            mov.w  #OUTMOD_5|CCIE, &TA0CCTL0;Apagar led automáticamente y pedir IRQ
            jmp    TA00Fin
            ;Evento de comparación, poner puerto como entrada sensible a flanco subida
Comparac    bic.b  #PUERTO, &P1DIR;Puerto como entrada
Comparac2   mov.w  #CAP|CM_1|CCIS_0|CCIE, &TA0CCTL0;CCR0 modo captura Sensible a tecla
TA00Fin     reti

            .intvecTIMER0_A0_VECTOR, TA00ISR
```

```
.intvecRESET_VECTOR, main
```