SISTEMAS BASADOS EN MICROPROCESADOR **Segundo parcial 2023-2024**

Apellidos:.....**SOLUCIÓN**.....

Nombre:.....

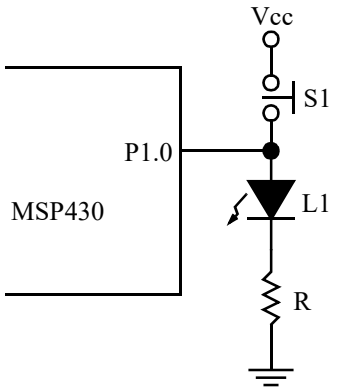
P1		P2	

Duración 2:00 horas

1.- (23/24 C2 5/10) Considere el circuito de la figura. Haga un programa en ensamblador del MSP430 ***basado en multitarea cooperativa*** que inicialmente configura el puerto como entrada para poder leer la tecla. Cuando el usuario la pulsa, el led se enciende sin intervención del MSP. El objetivo del programa es alargar el encendido del led para asegurar que el mismo está encendido al menos 3 segundos. Para ello inicialmente el puerto está en modo entrada, de forma que se pueda leer la tecla y medir el tiempo que ha estado pulsada. Si dicho tiempo es menor de 3 segundos, enciende el led desde el puerto poniéndolo como salida. Una vez completado el tiempo hasta los tres segundos, el puerto se vuelve a poner como entrada para un nuevo ciclo. Para evitar observar el apagado momentaneo del led, muestree la tecla con un periodo máximo de 10ms.

Notas:

- Nótese que se pide un programa que use multitarea cooperativa, no que lea las teclas por interrupciones ni use el TimerA.
- Minimice el consumo del sistema manteniéndolo todo el tiempo posible en modo de bajo consumo.
- Considere el pulsador libre de rebotes, el perro guardián desactivado, los puertos desbloqueados, pila inicializada y LFXT en marcha con cristal de 32768Hz.
- Dispone del módulo `st.asm`.



SOLUCIÓN

Dado que el pulsador no tiene resistencia externa, hay que usar la interna. En este caso debe ser de *pull-down*. El control del led es con lógica positiva (1 enciende, 0 apaga).

Hay que medir tiempos con una resolución de 10ms (más que suficiente para una pulsación de tecla). Eso supone que el proceso de medida debe tener una frecuencia de al menos 100Hz. Se ha escogido 128Hz.

Inicialmente el proceso establece una frecuencia de trabajo de 128Hz con la que escaneará la tecla. Cuando la misma se pulse (es decir, en una lectura estaba suelta y en la siguiente pulsada), se guardará el instante actual más 3 segundos en la variable `TFin`. Una vez que se suelte se comparará el tiempo actual con `TFin`. Si es mayor o igual, no se hará nada. En cambio, si es menor, se cambiará a modo encendido, en el que el puerto se pondrá como salida con un 1 y se ajustará la variable de próxima ejecución a `TFin`. No se volverá a entrar en la tarea hasta que se hayan completado los 3 segundos, momento en el que se volverá al estado inicial (puerto como entrada, led apagado y esperando pulsación). Una forma sencilla de gestionar todo esto es con una variable `Estado` que pasará por lo modos `ENOPUL`, `EPUL` y `EENC`. Se han añadido otros dos estados adicionales para encapsular el código de inicialización (`EINI`) y de error (`EULTIMO`). Nótese cómo, en el estado `ENOPUL`, cuando se pulsa la tecla, se lee el tiempo actual y se almacena en `TFin` dicho tiempo más 3 segundos.

Después se pasa al estado EPUL en el que se va a esperar hasta que se suelte la tecla. Una vez que se suelte, se va a comparar dicho instante con TFin, para comprobar si se ha alcanzado la cifra de 3 segundos. En ese caso, se pasa al estado ENOPUL a la espera de otra pulsación. En caso contrario, se pasa al estado EENC y se actualiza la variable teclaPE para que sea igual a TFin y se pone el puerto como salida con un 1, para que se encienda manualmente el led. Esto permite un control sencillo del tiempo de encendido y evita numerosas ejecuciones del proceso para ver si ha transcurrido el tiempo.

La existencia de 2 estados para saber si la tecla está pulsada o no (ENOPUL y EPUL) puede parecer caprichosa, pero es necesario para detectar los flancos de la tecla. Nótese que la liberación de la tecla ocurre cuando, en dos lecturas consecutivas de la tecla, la primera da pulsada y la segunda no. Muy distinto de, simplemente, ver si la tecla está libre, ya que, en ese caso, generaríamos un segundo de encendido del led cada vez que se leyera la tecla suelta.

```
            .cdecls C,LIST,"msp430ports.h"
;------------------------------
; Datos de configuración
;------------------------------
; *** Puertos de E/S ***
LEDBIT      .equ   BIT0
LEDPORT     .equ   P1IN

;Frecuencia de los distintos procesos. En Hz
FTECLA      .equ   128           ;Frecuencia de escaneo de la tecla

; *** Frecuencia del System Timer ***
FTA         .equ   32768         ;Frecuencia del reloj del TA. Hz
FST         .equ   128           ;Frecuencia del SystemTimer. Hz
;Tiempo mínimo de pulsación de tecla (TICS). Alargar hasta aquí si es necesario
TMIN        .equ   3*FST         ;Tiempo mínimo de encendido. TICS

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

;------------------------------
; Variables
;------------------------------
            .bss   teclaPE, 4    ;Próxima Ejecución del prceso de Parpadeo
            .bss   TFin, 4       ;Instante de tiempo que marca la pulsación de 3s
            .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
EENC        .equ   3*2           ;Estado led encendido manualmente
EULTIMO     .equ   EENC          ;Último estado. Sólo a efectos de comprobación
;-------------------------------------------------------------------------------
; main                                                                    v1.0
;-------------------------------------------------------------------------------
main        ;Inicializar SystemTimer
            mov.w  #CCR0, r12
            call   #stIni

            ;Inicializar procesos
            call   #Inicializa

superbucle  call   #AlargaTecla  ;Tarea que alarga la pulsación de la tecla
            bis.w  #LPM3+GIE, sr ;Entrar en bajo consumo
            jmp    superbucle
            .intvec RESET_VECTOR, main
            .text
```

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

;-------------------------------------------------------------------------------
; Proceso AlargaTecla                                                   v1.0
;-------------------------------------------------------------------------------
AlargaTeclamov.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    MdError
           add.w  r14, pc
           jmp    MdIni         ;Inicialización
           jmp    MdNoPul       ;No pulsada
           jmp    MdPul         ;Pulsada
           jmp    MdEnc         ;Encendido

           ;Estado Error. Desactivar tarea con el puerto como entrada
MdError    bic.b  #LEDBIT, &PDIR+LEDPORT;Puerto como entrada
           bic.b  #LEDBIT, &PREN+LEDPORT;sin resistencia de pulldown
           mov.w  #-1, &teclaPE ;Próxima ejecución... en el infinito
           mov.w  #-1, &teclaPE+1
           jmp    teclaFin
           ;Estado inicialización. Puerto como entrada con resistencia de pulldown
MdIni      ;bic.b #LEDBIT, &PDIR+LEDPORT;Puerto como entrada
           bis.b  #LEDBIT, &PREN+LEDPORT;Resistencia ...
           bic.b  #LEDBIT, &POUT+LEDPORT;... de pulldown
           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
MdNoPul    bit.b  #LEDBIT, &PIN+LEDPORT;Tecla pulsada?
           jz     teclaFin      ;...no. Salir
           call   #stTime       ;...sí. Dejar hora actual en R13:R12
           add.w  #TMIN, r12    ;R13:R12=ahora+Tiempo mínimo de tecla
           adc.w  r13
           mov.w  r12, &TFin    ;Guardar tiempo fin de pulsación mínima
           mov.w  r13, &TFin+2
           mov.b  #EPUL, &Estado;Cambiar de estado a esperar liberación
           jmp    teclaFin
           ;Estado tecla pulsada. Esperar a que se suelte
MdPul      bit.b  #LEDBIT, &PIN+LEDPORT;Tecla pulsada?
           jnz    teclaFin      ;...sí. Salir
           mov.w  &TFin, r12    ;...no. Ver si ha estado pulsada el tiempo mínimo
           mov.w  &TFin+2, r13
           call   #stComp
           tst.w  r12           ;Ha pasado?
           jhs    MdANoPul      ;...sí. Pasar a ENOPUL
           bis.b  #LEDBIT, &PDIR+LEDPORT;...no. Alargar encendido
           bis.b  #LEDBIT, &POUT+LEDPORT;Puerto salida. Led encendido
           mov.b  #EENC, &Estado;Cambiar de estado a encendido
           mov.w  &TFin, &teclaPE;Próxima ejecución: final de tiempo mínimo
           mov.w  &TFin+2, &teclaPE+2
           jmp    teclaFin
           ;Estado led encendido. Apagar el led, poner puerto como entrada
```

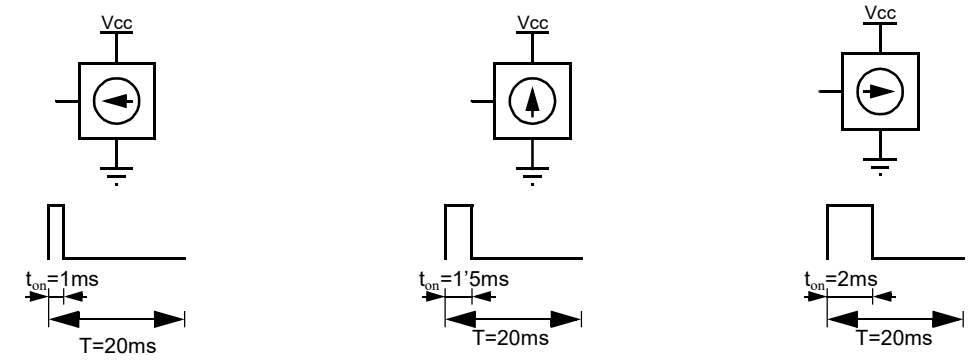
```
MdEnc       bic.b  #LEDBIT, &PDIR+LEDPORT;Puerto como entrada
            bic.b  #LEDBIT, &POUT+LEDPORT;con resistencia de pulldown
MdANoPul    mov.b  #ENOPUL, &Estado;Pasar a leer tecla
            ;jmp   teclaFin

teclaFin    ret
```

CRITERIO DE CORRECCIÓN

- Constantes: 10%
- Prinicipal: 30%
- Proceso: 60%

2.- (2324 C2 5/10) Se desea realizar el velocímetro de un coche con un MSP430. Se dispone de un servomotor SG90 conectado a la aguja del velocímetro y un tacómetro que mide la rotación de la rueda. El tacómetro proporciona una señal de salida de frecuencia variable en función de la velocidad de giro. Ha sido calibrado para proporcionar una salida de $120*10^3$ rpm (revoluciones por minuto) cuando el vehículo circula a 250km/h.

El servomotor es un sistema formado por un motor de CC, acoplado a unos engranajes, un sensor de posición del eje y circuitería de control. Con todo ello se puede controlar de forma precisa el ángulo del eje y enclavarlo en la posición deseada. Dispone de un par de terminales de alimentación (+3'3V y masa) y una entrada de control tipo PWM (*Pulse Width Modulation*) que determina la posición del mismo. Por ejemplo, el SG90 tiene un rango de giro de 180º y se controla con una señal de 50Hz con tiempo de encendido entre 1 y 2 ms:



Posición de giro: 0º | Posición de giro: 90º | Posición de giro: 180º

La posición de 0º se alcanza cuando la señal de control tiene un t_{on} de 1ms. La posición de 180º se alcanza cuando t_{on} = 2ms. Variando linealmente t_{on} se puede conseguir cualquier posición entre 0º y 180º.

Se desea realizar un programa en ensamblador del MSP430 que lea la velocidad de rotación de la rueda y muestre en el velocímetro la velocidad del coche con resolución de 0'5km/h. Gestione el programa por interrupciones en el mejor modo de bajo consumo.

Datos:

- Cuando la aguja está a 0º marca 0 km/h y cuando está a 180º marca 250 km/h.
- La entrada de control del servo se ha conectado a P1.0 cuya función primaria es TA0.1.
- La salida del tacómetro se ha conectado al puerto P1.1, cuya función secundaria es TA1CLK.
- La entrada CCI2B del TA1 está conectada internamente a ACLK.
- Considere el perro guardián desactivado, los puertos desbloqueados y la pila inicializada.

SOLUCIÓN

Se requiere el uso de dos temporizadores para hacer el trabajo. El TA0 servirá para generar la señal PWM por TA0.1 (P1.0) con SMCLK como fuente de reloj en modo UP. El CCR0 se encargará para fijar la frecuencia de la señal PWM (50Hz) y la resolución (0'5km/h).

El TA1 medirá la salida del tacómetro. Dicha señal se conectará como entrada de reloj de TA1 y se comparará con ACLK que entra por CCI2B.

Temporización del servo: sabemos que la velocidad del vehículo es un parámtero entre 0 y 250 km/h con una resolución de 0'5km/h. Eso implica 500 cuentas. Por tanto, la señal t_{on} del servo variará entre 1 y 2 ms con 500 cuentas. Es decir, una resolución de (2000µs-1000µs)/ 500=2µs. Esto descarta el uso de ACLK como señal de reloj ya que su periodo es de más 30µs y obliga a usar SMCLK que tiene una frecuencia por defecto de 1MHz. Para conseguir la resolución de 2µs será necesario dividir por 2 la misma. El mayor tiempo a medir es el periodo de la señal del servo, 20ms, que supone 20ms/2µs = 10^4 ciclos, por debajo del límite del temporizador, 65536. El CCR0 se establecerá, por tanto, en 9999 (10000 ciclos menos 1) y el CCR2 se usará para controlar la señal de posición.

Con estos ajustes, la relación será uno a uno (cada estado de cuenta, medio km/h). La generación de la señal PWM es completamente automática y no necesita mayor intervención de la CPU.

La medida del tacómetro se hará introduciendo la señal de salida del mismo por P1.1 hacia la entrada de reloj externa de TA1, el TA1CLK. Se aprovechará que el CCI2B está conectado internamente con ACLK para comparar la señal de reloj desconocida con una señal conocida. La idea es medir cuántos ciclos de TA1CLK (el tacómetro) suceden en 1 segundo. Esto se puede hacer capturando TA1 en un flanco de ACLK y 1 segundo después con lo que se obtiene directamente la medida en Hz del tacómetro.

Aplicando la fórmula que relaciona los CCR con el ciclo de trabajo (DC), tenemos:

$$DC_x = \frac{CCR_x}{CCR_0 + 1} 100\% \Rightarrow CCR_x = DC_x(CCR_0 + 1)$$

Esto nos permite montar la siguiente tabla que resume los valores mínimo y máximo de las señales de entrada y salida:

Tacóm. (rpm)	Tacóm. (Hz)	Velocidad (km/h)	Ángulo servo (º)	t_{on} (ms)	DC (%)	CCR2
0	0	0	0	1'0	5'0	500
120000	2000	250	180	2'0	10'0	1000

Con esta tabla es fácil ver que hay que mapear la variable de entrada `Tacóm. (Hz)` a la variable de salida, `CCR2`. Es decir, el rango `[0, 2000]` al rango `[500, 1000]`. Esto nos da la relación `CCR2 = Tacom/4 + 500`. O mejor, se puede hacer la medida del tacómetro durante un cuarto de segundo, con lo que la señal de entrada estaría directamente en el rango `[0, 500]` y nos evitamos tener que dividir por 4 la medida y esperar durante un segundo.

Tras inicializar los temporizadores, todo el proceso se hará por interrupciones (medida del tacómetro) o por hardware (generación de señal PWM del velocímetro). Dado que el TA0 se alimenta con SMCLK, el máximo modo de bajo consumo alcanzable es LPM1, ya que el DCO, de quien deriva SMCLK, debe estar encendido.

```
            .cdecls C,LIST,"msp430ports.h"
; Configuración de puertos -------------------------------------------------
SERVOB      .equ   BIT0
SERVOP      .equ   P1IN
TACOB       .equ   BIT1
TACOP       .equ   P1IN
; Constantes de configuración ----------------------------------------------
FSMCLK      .equ   1000000       ;Frecuencia del SMCLK. En Hz
FACLK       .equ   32768         ;Frecuencia del ACLK. En Hz
FPWM        .equ   50            ;Frecuencia de la señal PWM. En Hz
TONMIN      .equ   1000          ;Valor mínimo activo de PWM. En us
TONMAX      .equ   2000          ;Valor máximo activo de PWM. En us
VMAX        .equ   250000        ;Velocidad máxima en m/h
VRES        .equ   500           ;Resolución de medida de velocidaden m/h
TACOMAX     .equ   2000          ;Salida máxima del tacómetro (Hz)
; Constantes calculadas ----------------------------------------------------
VCONT       .equ   VMAX/VRES     ;Información de la salida (número de cuentas)
RESPWM      .equ   (TONMAX-TONMIN)/VCONT;Resolución de PWM (us/unidad de medida)
OFFSETPWM   .equ   TONMIN/RESPWM ;Número de ciclos de la parte fija de PWM
DIVTA       .equ   FSMCLK*RESPWM/1000000;Divisor de reloj para TA0
FTA         .equ   FSMCLK/DIVTA  ;Frecuencia del TA0
CCR0        .equ   FTA/FPWM-1    ;Valor del CCR0 para conseguir FPWM
DIVACLK     .equ   TACOMAX/VCONT ;Divisor del periodo de ACLK para medir Tacómetro
FTEST       .equ   FACLK/DIVACLK ;Número de ciclos de ACLK para medida de tacómetro
; Variables ----------------------------------------------------------------
            .bss   NCapturas, 2  ;Contador de capturas con ACLK
            .bss   TIni, 2       ;Tiempo de la primera captura
;-------------------------------------------------------------------------------
; void main (void);                                                      v1.0
;-------------------------------------------------------------------------------
```

```
main        ;Inicializar variables
            clr.w  &NCapturas
            ;TA0. Generación de la señal PWM que contola el servo
            ;Puertos. P1.0 salida función primaria
            bis.b  #SERVOB, &SERVOP+PDIR
            bis.b  #SERVOB, &SERVOP+PSEL0
            ;CCR0. Fija frecuencia de PWM
            mov.w  #CCR0, &TA0CCR0
            ;CCR1. PWM en modo Reset/Set (PWM Positivo). Inicialmente velocidad = 0
            mov.w  #OUTMOD_7, &TA0CCTL1
            mov.w  #OFFSETPWM, &TA0CCR1
            ;TA0, en modo UP con fuente SMCLK y divisor 2
            mov.w  #DIVTA-1, &TA0EX0;Divisor de reloj
            mov.w  #TASSEL__SMCLK|ID__1|MC__UP|TACLR, &TA0CTL

            ;TA1. Medida del tacómetro
            ;Puertos. P1.1 entrada TA1CLK función secundaria
            ;bic.b #TACOB, &TACOP+PDIR
            bis.b  #TACOB, &TACOP+PSEL1
            ;CCR2=ACLK. Captura, entrada por CCI2B sensible en flanco subida, IRQ
            mov.w  #CAP|CM_1|SCS|CCIS_1|CCIE, &TA1CCTL2
            ;TA1, en modo CONT con fuente TA1CLK
            mov.w  #TASSEL__TACLK|ID__1|MC__CONTINUOUS|TACLR, &TA1CTL

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

;-------------------------------------------------------------------------------
; TA11ISR                                                                 v1.0
;
; Subrutina de servicio de la IRQ de CCR2 de TA1. Mide el tacómetro.
;-------------------------------------------------------------------------------
TA11ISR     bic.w   #CCIFG, &TA1CCTL2    ;Borrar IRQ
            inc.w   &NCapturas           ;Actualizar el número de capturas
            cmp.w   #1, &NCapturas       ;Es la primera?
            jnz     NoPrimera            ;...no, seguir
            mov.w   &TA1CCR2, &TIni      ;...sí. Guardar instante inicial
            jmp     TA11ISRFin           ;Salir
NoPrimera   cmp.w   #FTEST+1, &NCapturas ;Se ha completado el periodo de medida?
            jlo     TA11ISRFin           ;...no. Salir
                                         ;...sí. Calcular frecuencia de tacómetro
            ;Calcular medida
            push.w  r12                  ;En una ISR no se alteran los registros
            mov.w   &TA1CCR2, r12        ;R12=TFin
            sub.w   &TIni, r12           ;R12=TFin-TIni=Frecuencia
            cmp.w   #TACOMAX+1, r12      ;Frecuencia>TACOMAX?
            jlo     TA11ISRSigue         ;...no. Refrescar velocímetro
            mov.w   #TACOMAX, r12        ;...sí. Saturar en TACOMAX y refrescar
TA11ISRSigue add.w  #OFFSETPWM, r12      ;Calcular ton
            mov.w   r12, &TA0CCR1        ;Actualizar PWM velocímetro
            pop.w   r12
            mov.w   &TA1CCR2, &TIni      ;Últ. captura es primera de sig. medida
            mov.w   #1, &NCapturas       ;Inicializar contador para próx medida
TA11ISRFin  reti

            .intvec TIMER1_A1_VECTOR, TA11ISR
            .intvec RESET_VECTOR, main
```

CRITERIO DE CORRECCIÓN

- Main servo: 20%
- Main taco: 20%
- Proceso: 60%