9.1 Realice una rutina que configure el Timer_A TA0 para generar una onda cuadrada de 100 Hz por una de sus salidas y una interrupción periódica a la CPU. ¿Con qué frecuencia se recibe en la CPU la interrupción? ¿Cuál es el error en la frecuencia generada?

SOLUCIÓN

La interrupción se produce dos veces en cada periodo de la señal generada, esto es 200 Hz.

```
FTA0        .equ    4096                ;En Hz
FSalida     .equ    100                 ;En Hz
Firq        .equ    2*FSalida           ;En Hz
CCR0        .equ    FTA0/Firq-1

ConfTA0     ;Configurar una de las salidas disponibles: P1.5, P7.1, P10.1
            bis.b   #BIT5, &P1SELC      ;P1.5 función 3
            bis.b   #BIT5, &P1DIR       ;P1.5 salida
            ;Configurar el tiempo
            mov.w   #CCR0, &TA0CCR0
            ;CCR0 en modo comparación. Salida activa en conmutación. IRQ
            mov.w   #OUTMOD__TOGGLE|CCIE, &TA0CCTL0
            ;TA0: Fuente ACLK a 4096 Hz, modo Up, resetear
            mov.w   #TASSEL__ACLK|ID__8|MC__UP|TACLR,&TA0CTL
            eint                        ;Habilitar interrupciones
            ret

TA00ISR     ;Subrutina de interrupción del CCR0 del TA0
            reti

            .intvec TIMER0_A0_VECTOR, TA00ISR
```

Cálculo del error producido debido al redondeo:

CCR0 = 4096/200-1 = 19’48

El valor se trunca a 19, con lo que la frecuencia realmente producida es:

Firq = FTA0/(CCR0+1) = 4096/(19+1) = 204’8 Hz

Lo que supone un error de 204’8/200 = +2’4%

Si no se usa el divisor de frecuencia y se deja la frecuencia base de 32768Hz, el valor calculado del CCR0 es de la frecuencia generada sería 162’48 que, al truncarse, produce una frecuencia de 201’03 Hz, que da un error de 0’5%

9.2 Realice una rutina que configure el Timer_A TA0 para generar tres señales periódicas de 1, 60 y 100 Hz.

SOLUCIÓN

```
FTA0        .equ    4096                ;En Hz
FSal0       .equ    1                   ;En Hz
FSal1       .equ    60                  ;En Hz
FSal2       .equ    100                 ;En Hz
CCR0        .equ    FTA0/(2*FSal0)-1
CCR1        .equ    FTA0/(2*FSal1)-1
CCR2        .equ    FTA0/(2*FSal2)-1

;-------------------------------------------------------------------------------
; ConfTA0
;-------------------------------------------------------------------------------
ConfTA0     ;TA0.0 Configurar una de las salidas: P1.5, P7.1, P10.1
            bis.b   #BIT5, &P1SELC      ;P1.5 función 3
            ;TA0.1 Configurar una de las salidas: P1.0, P1.6, P7.2, P7.6
            bis.b   #BIT0, &P1SEL0      ;P1.0 función 1
            ;TA0.2 Configurar una de las salidas: P1.1, P1.7, P7.3, P7.5
            bis.b   #BIT1, &P1SEL0      ;P1.1 función 1
```

```
            bis.b   #BIT0|BIT1|BIT5, &P1DIR;P1.0,1,5 salidas
            ;Configurar los tiempos
            mov.w   #CCR0, &TA0CCR0
            mov.w   #CCR1, &TA0CCR1
            mov.w   #CCR2, &TA0CCR2
            ;Modo comparación. Salida activa en conmutación. IRQ
            mov.w   #OUTMOD__TOGGLE|CCIE, &TA0CCTL0
            mov.w   #OUTMOD__TOGGLE|CCIE, &TA0CCTL1
            mov.w   #OUTMOD__TOGGLE|CCIE, &TA0CCTL2
            ;TA0: Fuente ACLK a 4096 Hz, modo continuo, resetear
            mov.w   #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLR,&TA0CTL
            eint                        ;Habilitar interrupciones
            ret

;-------------------------------------------------------------------------------
; TA00ISR                                                               v1.0
;
;Subrutina de interrupción del CCR0 del TA0
;-------------------------------------------------------------------------------
TA00ISR     add.w   #CCR0, TA0CCR0
            reti

;-------------------------------------------------------------------------------
; TA01ISR                                                               v1.1
;
; Subrutina de interrupción de CCR1-CCR2 del TA0
; Control de cambios -------------
; v1.1 Corrección de errata. Los JZ eran JNZ por error
;-------------------------------------------------------------------------------
TA01ISR     bit.w   #CCIFG, &TA0CCTL1   ;IRQ del CCR1?
            jz      VerCCR2             ;...no, ver siguiente
            bic.w   #CCIFG, &TA0CCTL1   ;...sí, borrar IRQ
            add.w   #CCR1, &TA0CCR1     ;Actualizar tiempo de próxima IRQ
VerCCR2     bit.w   #CCIFG, &TA0CCTL2   ;IRQ del CCR2?
            jz      TA01ISRFin          ;...no, salir
            bic.w   #CCIFG, &TA0CCTL2   ;...sí, borrar IRQ
            add.w   #CCR2, &TA0CCR2     ;Actualizar tiempo de próxima IRQ
TA01ISRFin  reti

            .intvec TIMER0_A0_VECTOR, TA00ISR
            .intvec TIMER0_A1_VECTOR, TA01ISR
```

9.3 (1718sep) Genere un pulso positivo de un cuarto de segundo en el puerto P1.0 usando el TimerA y programación por interrupciones. Datos:

- Suponga el perro guardián detenido, que se han desbloqueado los puertos, se ha inicializado el SP al final de la RAM y que el ACLK ha sido configurado a 32768 Hz.
- En el P1.0 está la salida TA0.1 en su función 1.

SOLUCIÓN

El proceso de generación del pulso consiste en:

- Configurar el P1.0 para que sea controlado por el TA0.1.
- Configurar el TA0 en modo continuo, con divisor a 64 (hay que programar los dos registros de división a 8), con lo que la frecuencia de entrada será de 32768/64=512. Una frecuencia tan baja minimiza el consumo del módulo, sin perjuicio de la precisión de medida, ya que 0'25seg es un divisor exacto de la frecuencia de reloj.
- Configurar el TA0.1 en modo comparación y actuando sobre la salida en modo Set con CCR1=1. Esto hace que 1 ciclo de TACLK después de poner en marcha el TA0, la señal de salida (P1.0) se ponga a 1 y se solicite una interrupción.
- La primera vez que se entre en la ISR se cambia el modo de salida a Reset y se

actualiza el CCR1 con el número de ciclos del pulso (32768/64)/4=128.
- Cuando se produzca la siguiente comparación (un cuarto de segundo después), el TA0 pone a 0 automáticamente la salida y solicita una interrupción.
- Una vez dentro de la ISR, se desactiva la IRQ y, opcionalmente si el TA0 no se usa para otra cosa, se para el TA0.

La forma de la solución es una subrutina llamada Pulso y una ISR llamada TA01ISR que hacen todo el trabajo después de llamar a la primera.

```
FACLK         .equ    32768/64              ;En Hz
TPULSO        .equ    250                   ;En ms
NPULSO        .equ    TPULSO*FACLK/1000     ;Ojo orden de operaciones

;-------------------------------------------------------------------------------
; Pulso                                                                    v1.0
;-------------------------------------------------------------------------------
Pulso         ;TA0 Configurar puerto de salida: P1.0(1)
              bis.b   #BIT0, &P1SEL0        ;P1.0 función 1
              bis.b   #BIT0, &P1DIR         ;P1.0 salida
              ;CCR1. Modo comparación, Salida a 0 inmediata
              mov.w   #OUTMOD_0, &TA0CCTL1
              ;CCR1. Modo comparación, Salida a 1 después de 1 ciclo de TACLK, IRQ
              mov.w   #1, &TA0CCR1          ;produce flanco subida e IRQ en 1 ciclo
              mov.w   #OUTMOD_1|CCIE, &TA0CCTL1
              ;TA0: TACLK(32768Hz) a 512Hz (div=8*8), modo continuo, resetear
              mov.w   #7,&TA0EX0
              mov.w   #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLR,&TA0CTL
              eint                          ;Habilitar interrupciones
              ret

;-------------------------------------------------------------------------------
; TA01ISR                                                                  v1.0
;
;Subrutina de interrupción de CCR1 del TA0
;-------------------------------------------------------------------------------
TA01ISR       bic.w   #CCIFG, &TA0CCTL1     ;Borrar IRQ de CCR1
              ;La primera vez que se entre en la ISR, cambiar el modo de salida a
              ;Reset y cambiar el CCR1 para que cuente los 0'25seg. La segunda vez,
              ;desactivar la IRQ. Opcionalmente se puede parar el temporizador
              bit.w   #OUTMOD2, &TA0CCTL1   ;Qué modo hay configurado?
              jz      TA01ISR1              ;...set. Preparar flanco de bajada
              bic.w   #CCIE, &TA0CCTL1      ;...reset. Desactivar IRQ
              bic.w   #MC1|MC0, &TA0CTL     ;Parar TA0. Comentar si otros usos
              jmp     TA01ISRFin
TA0ISR1       add.w   #NPULSO, &TA0CCR1     ;Programar tiempo del pulso (ojo ADD)
              bis.w   #OUTMOD2, &TA0CCTL1   ;Modo de salida Reset
TA01ISRFin    reti

              .intvec TIMER0_A1_VECTOR, TA01ISR
```

Una alternativa es usar las capacidades PWM para generar los dos tiempos y flancos que se necesitan para definir el pulso usando CCR0 y CCR1. Usando el modo Up o Contiunuous se puede usar el modo Set/Reset o el Toggle/Reset del CCR1 para hacer todo el trabajo en el timer sin necesidad de interrupciones (salvo para parar el timer).

```
FACLK         .equ    32768/64              ;En Hz
TPULSO        .equ    250                   ;En ms
NPULSO        .equ    TPULSO*FACLK/1000     ;Ojo orden de operaciones
NESPERA       .equ    1                     ;Ciclos antes del pulso
;-------------------------------------------------------------------------------
; Pulso                                                                    v2.0
;-------------------------------------------------------------------------------
Pulso         ;TA0 Configurar puerto de salida: P1.0(1)
```

```
                bis.b   #BIT0, &P1SEL0          ;P1.0 función 1
                bis.b   #BIT0, &P1DIR           ;P1.0 salida
                ;CCR1. Salida a 0 inmediata
                mov.w   #OUTMOD_0, &TA0CCTL1
                ;CCR1 modo comparación, salida Set/Reset; salida a 1 después de
                ;NESPERA ciclos de TACLK y a 0 después de NPULSO ciclos
                mov.w   #NESPERA, &TA0CCR1      ;Tiempo antes del flanco de subida
                mov.w   #OUTMOD_3, &TA0CCTL1 ;Salida Set/Reset
                ;CCR0 modo comparación, sin salida asociada. IRQ
                mov.w   #NESPERA+NPULSO, &TA0CCR0;Tiempo antes del flanco bajada. IRQ
                mov.w   #CCIE, &TA0CCTL0        ;IRQ para conocer tiempo final
                ;TA0: TACLK(32768Hz) a 512Hz (div=8*8), modo continuo, resetear
                mov.w   #7,&TA0EX0
                mov.w   #TASSEL__ACLK|ID__8|MC__CONTINUOUS|TACLR,&TA0CTL
                eint                            ;Habilitar interrupciones
                ret

;-------------------------------------------------------------------------------
; TA00ISR                                                                 v2.0
;
;Subrutina de interrupción de CCR0 del TA0. Se ha terminado de generar el pulso.
;Desactivar la IRQ y parar el timer si no se usa para otros menesteres.
;-------------------------------------------------------------------------------
TA00ISR         bic.w   #CCIE, &TA0CCTL0        ;Desactivar IRQ
                bic.w   #MC1|MC0, &TA0CTL       ;Parar TA0. Comentar si otros usos
                reti

                .intvec TIMER0_A0_VECTOR, TA00ISR
```

9.4 Se dispone de una señal de reloj ECLK de 1 MHz con la que se quiere medir la anchura del pulso positivo de una señal A con una resolución de 10µs. La anchura máxima del pulso es de 1 minuto. Deje valor de la medida en una variable `Medida`. Indique cómo hacer la medida con un MSP430FR6989. ¿Qué cambios debería hacer si se desea ampliar la resolución a 1µs?

SOLUCIÓN

Dada la latencia del sistema de interrupciones (finalizar instrucción actual, hasta 6 ciclos, apilar PC y SR, 5 ciclos,...) el pulso mínimo de 10µs sólo se podrá medir si se acelera la ejecución de instrucciones del valor por defecto de MCLK = 1MHz (sólo habría 10 ciclos para evitar un overflow del TA0).

```
                .bss    Medida, 4               ;60 segundos en TICS de 10us = 6MTic
                .bss    TA0HWord, 2             ;TA0 de 32 bits. Palabra alta
                .bss    TIni, 2                 ;Valor CCR0 en flanco de subida

;-------------------------------------------------------------------------------
; ConfTA0                                                                 v1.0
;-------------------------------------------------------------------------------
ConfTA0         ;TA0 Configurar entrada reloj externa: P1.2(2), P6.7(1) o P7.0(1)
                bis.b   #BIT2, &P1SEL1          ;P1.2 función 2
                ;TA0.0 Configurar entrada CCIA: P1.5(3)
                bis.b   #BIT5, &P1SELC          ;P1.5 función 3
                ;bic.b  #BIT2|BIT5, &P1DIR      ;P1.2 y P1.5 entrada
                ;TA0: TACLK a 100KHz(TACLK/(2*5)), modo continuo, IRQ, resetear
                mov.w   #4,&TA0EX0
                mov.w   #TASSEL__TACLK|ID__2|MC__CONTINUOUS|TACLR|TAIE,&TA0CTL
                ;CCR0. Modo captura, flanco de subida, entrada CCIA, IRQ
                mov.w   #CM_1|CCIS_0|SCS|CAP|CCIE, &TA0CCTL0
                eint                            ;Habilitar interrupciones
                ret

;-------------------------------------------------------------------------------
```

```
; TA00ISR                                                                    v1.0
;
; Subrutina de interrupción del CCR0 del TA0.
;-------------------------------------------------------------------------------
TA00ISR      ;IRQ de CCR0 en modo captura. El flanco de subida es el inicio
             ;de medida y el de bajada el final
             xor.w   #CM1|CM0, &TA0CCTL0  ;Conmutar flanco
             bit.w   #CM0, &TA0CCTL0      ;Que flanco ha quedado?
             jnz     FinCaptura           ;...subida. Era bajada. Terminar medida
             mov.w   &TA0CCR0, &TIni      ;...bajada. Capturar instante de inicio
             clr.w   &TA0HWord            ;Empezar a contar tiempo
             jmp     TA00ISRFin
FinCaptura   ;Flanco de bajada de la señal. Terminar medida
             mov.w   &TA0CCR0, &Medida
             mov.w   &TA0HWord, &Medida+2 ;Medida=TA0HWord:TA0CCR0(Fin)
             sub.w   &TIni, &Medida
             sbc.w   &Medida+2            ;Medida = Medida - 0:TA0CCR0(Ini)
TA00ISRFin   reti

;-------------------------------------------------------------------------------
; TA01ISR                                                                    v1.0
;
; Subrutina de interrupción de TAR,CCR1-CCR2 del TA0.
;-------------------------------------------------------------------------------
TA01ISR      bic.w   #TAIFG, &TA0CTL      ;Borrar IRQ de TA
             inc.w   &TA0HWord            ;Contabilizar desbordamiento
TA01ISRFin   reti

             .intvec TIMER0_A0_VECTOR, TA00ISR
             .intvec TIMER0_A1_VECTOR, TA01ISR
```

9.5 (20/21 C2 5/10) Por P1.2 se introduce una señal cuya frecuencia, entre 1Hz y 50KHz, se desea medir con la ayuda de un MSP430FR6989 usando el TA0 e interrupciones con el procesador en un modo de bajo consumo. Configure el sistema de forma que la señal sea la fuente de reloj del TA0 y aproveche que ACLK está conectado a la entrada CCI2B para medir cuántos ciclos de reloj de la señal desconocida entran en 1 segundo. El programa realizará las medidas continuamente y dejará dicho valor (un número entre 1 y 50.000) en la variable `Frecuencia` de 16 bits sin signo. En caso de que la frecuencia de la señal esté fuera de rango, escribirá el valor FFFF. Datos: TA0CLK en P1.2 (función secundaria) y CCI2B es ACLK (conectado internamente). Considere el perro guardián desactivado, los puertos desbloqueados, pila inicializada y ACLK=LFXT en marcha con cristal de 32768Hz.

SOLUCIÓN

Hay 2 formas de medir una frecuencia:

- Usar un reloj conocido como fuente del TA y medir con él cuántos ciclos dura un perido completo de la señal a medir. Una vez conocido el periodo, su inversa es la frecuencia.
- Usar la señal a medir como fuente del TA y medir cuántos ciclos hay en una señal conocida, como ACLK. Se mide directamente la frecuencia.

La elección de uno u otro va a depender del rango de la señal a medir y de los relojes disponibles.

Implementación del primer método: se usa el reloj externo de 1MHz divido por 20 (1MHz/20=50kHz) como fuente del timer en modo continuo y la señal a medir como entrada de uno de los CCR configurado en modo captura. Después se hacen 2 capturas (da igual el flanco elegido) de la misma y se restan las medidas capturadas (llamemos DIF a esa resta). Si DIF=1, tenemos una señal de 50kHz. Si DIF=50000, la señal es 1Hz. SI DIF<1 o DIF>50000

sería un error de la señal de entrada, que estaría fuera de rango. La frecuencia medida sería Frec=50000/DIF.

El segundo método tiene la ventaja de que no hay que hacer divisiones. Una forma sería poner la señal desconocida como entrada del TA y medir uno o varios ciclos de ACLK, que es conocida y precisa. En función del número de ciclos que usemos tendremos más o menos precisión. Si se quiere llegar a medir señales de 1 Hz, hay que medir durante 1 segundo, lo que implica hacer dos capturas con 32768 ciclos entre ellas, o mejor dicho, 32768 capturas, restando la primera a la última. Veamos esta segunda opción en detalle.

Se usará `TA0CLK` como fuente de reloj del `TA0` y `ACLK` como entrada B del `CCR2`. Para realizar las medidas se capturarán 32768 flancos de subida (o de bajada, es indistinto), para que el tiempo de medida sea de un segundo. La frecuencia será el valor de la diferencia entre la útima captura y la primera. Salvo la configuración inicial, todo el proceso se hará por interrupciones en LPM3 (para que esté activo el `ACLK`). Hay tres posibles acciones cuando se produce la IRQ:

- Primera: guardar la captura en `TIni`.
- Última: calcular `Frecuencia = CCR2 - TIni`.
- Resto: nada

En todos los casos hay que borrar el flag de interrupción y llevar la cuenta del número de capturas. Una vez hecha la medida, se empezará con la siguiente. Se tomará la última captura como valor inicial de la siguiente medida. Nótese que no hay que tener en cuenta el posible desbordamiento del `TA0R` a la hora de calcular la frecuencia.

Con un ejemplo se enterá mejor el concepto: suponga que el valor de la primera captura es `TIni = 26789`. Suponga también que la frecuencia de la señal que queremos medir es 40kHz. Un segundo después de la primera captura, el `TA0R` se habrá incrementado 40000 veces, por lo que debería tener un valor de 26789+40000=66789. Como el contador es de 16 bits, una vez que alcanza el valor máximo de 65535, se resetea, con lo que el valor real sería 66789-65536=1253. Cuando finalmente se calcule TFin-TIni=1253-26789=-25536_{10}. Se obtiene un número negativo. Pero si obviamos este hecho e interpretamos el número sin signo, tendríamos -25536_{10} = $9C40_{16}$ = 40000_{10}. Es decir, la representación del 40000 y del -25536 es la misma. Lo que está pasando que es estamos trabajando con aritmética módulo 16. Podemos hacer las sumas y las restas y olvidarnos de los posibles desbordamientos siempre que la señal no supere los máximos previstos. Si hubiéramos llevado la cuenta con 17 bits las cuentas habrían sido `TFin=TIni+40000=66789`. `Frencuencia = TFin-Tini = 66789-26789 = 40000`. Hacerla con 16 bits no tiene repercusión. El único límite es que la frecuencia máxima se pueda representar con 16 bits, como pasa en nuestro caso.

```
FTEST         .equ    32768               ;Frecuencia del reloj de medida
FMAX          .equ    50000               ;Frecuencia máxima a medir

              .bss    Frecuencia, 2       ;Valor medido
              .bss    TIni, 2             ;Valor de la primera captura
              .bss    NCaptura, 2         ;Número de capturas

;-------------------------------------------------------------------------------
; MideFrec                                                                 v1.0
;-------------------------------------------------------------------------------
MideFrec      ;Inicializar variables
              mov.w   #0xFFFF, &Frecuencia ;Frecuencia = "no hay medida"
              clr.w   &NCaptura           ;No hay capturas aún

              ;Configurar puertos
              ;bic.b  #BIT2, &P1DIR        ;P1.2 entrada
              bis.b   #BIT2, &P1SEL1       ;P1.2 función secundaria (TA0CLK)
```

```
              ;TA0.2 en modo captura, flanco de subida, CCIB (ACLK). IRQ actva.
              ;Como la señal a capturar (ACLK) es asíncrona con la del reloj (señal
              ;desconocida, es conveniente activar la sincronización (SCS)
              mov.w   #CAP|CM_1|CCIS_1|SCS|CCIE, &TA0CCTL2
              ;TA0 entrada externa (TA0CLK), divisor 1, modo continuo
              mov.w   #TASSEL__TACLK|ID__1|MC__CONTINUOUS|TACLR, &TA0CTL
              bis.w   #GIE|LMP3, sr         ;Habilitar interrupciones y bajo consumo

;-------------------------------------------------------------------------------
; TA01ISR                                                                  v1.0
;-------------------------------------------------------------------------------
TA01ISR       bic.w   #CCIFG, &TA0CCTL2     ;Borrar IRQ
              inc.w   &NCaptura             ;Actualizar el número de capturas
              cmp.w   #1, &NCaptura         ;Es la primera?
              jnz     NoPrimera             ;...no, seguir
              mov.w   &TA0CCR2, &TIni       ;...sí. Guardar instante inicial
              jmp     TA01ISRFin            ;Salir
NoPrimera     cmp.w   #FTEST+1, &NCaptura   ;Se ha completado un segundo?
              jlo     TA01ISRFin            ;...no. Salir
              mov.w   #-1, &Frecuencia      ;...sí. Calcular frec. Por defecto,error
              ;Calcular medida
              push.w  r12                   ;No alterar registros
              mov.w   &TA0CCR2, r12         ;R12=TFin
              sub.w   &TIni, r12            ;R12=TFin-TIni=Frecuencia
              jz      TA01ISRError          ;Frecuencia=0? Error
              tst.w   #FMAX+1, r12          ;...no. Frecuencia>FMax?
              jhs     TA01ISRError          ;...sí. Error
TA01ISRSigue  mov.w   r12, &Frecuencia      ;...no. Valor medido correcto. Guardar
              pop.w   r12
              mov.w   &TA0CCR2, &TIni       ;Últ. captura es primera de sig. medida
              mov.w   #1, &NCaptura         ;Inicializar contador para próx medida
TA01ISRFin    reti
TA01ISRError  mov.w   #-1, r12              ;Marcar como lectura no válida
              jmp     TA01ISRSigue
              .intvec RESET_VECTOR, MideFrec
              .intvec TIMER0_A1_VECTOR, TA01ISR
```

9.6 Realice una función `pwmSet10` que sigue el convenio de llamada de C y que activa una señal PWM de 8 bits de resolución y con el ciclo de trabajo deseado en el puerto P1.0. ¿Cuál es la frecuencia de la señal resutante?

```
void pwmSet10 (unsigned char dc);
```

SOLUCIÓN

En el datasheet del launchpad se puede ver que el puerto P1.0 tiene la salida del TA0.1 en su función 1. Se usará el modo Up con CCR0 estableciendo la frecuencia base y CCR1 para controlar el ciclo de trabajo con modo de salida 7 (reset/set). Para tener una resolución de 8 bits, se hará CCR0=255, lo que dará una frecuencia de la señal generada de $TA0_{CLK}/256$. Si se usa ACLK como fuente de reloj, la frecuencia será 128Hz, más que suficiente para controlar el led 1 de usuario (control de brillo). No se necesitan interrupciones, ya que el TA lo hace todo por nosotros.

```
;-------------------------------------------------------------------------------
; void pwmSet10 (unsigned char dc);                                        v1.0
;-------------------------------------------------------------------------------
pwmSet10      bis.b   #BIT0, &P1SEL0        ;P1.0 función 1 (TA0.1)
              bis.b   #BIT0, &P1DIR         ;P1.0 salida
              mov.w   #OUTMOD_7, &TA0CCTL1 ;Modo de salida 7 para TA0.1
              bic.w   #0xFF00, r12          ;Borrar byte alto de dc
              mov.w   r12, &TA0CCR1         ;Establecer ciclo de trabajo
              mov.w   #255, &TA0CCR0        ;Frecuencia de TA0
              ;TA0 con ACLK/1, modo Up, Reset
```

```
        mov.w   #TASSEL__ACLK|ID__1|MC__UP|TACLR, &TA0CTL
        ret
```

9.7 Realice una función `pwmSet` que sigue el convenio de llamada de C y que activa una señal PWM de 8 bits de resolución y con el ciclo de trabajo deseado en el puerto indicado. Si `dc` es -1, desactiva la señal PWM de ese puerto. Puerto sigue el convenio `P*8+B`, donde `P` es el número de puerto (1<=P<=10) y B es el número de bit (0<=B<=7). Devuelve 0 si todo fue bien y -1 en caso de error. Debe controlar al menos 8 señales simultaneamente.

```
int pwmSet (int puerto, unsigned char dc);
```