TIMER A

Relación entre la resolución de una señal PWM y la frecuencia de la misma:

Según hemos estudiado, la siguiente fórmula nos permite calcular el valor que hay que programar en el CCR0 para obtener una frecuencia deseada Fs:

$$CCR0 = \frac{F_{TACLK}}{F_s} - 1$$

Por otra parte, si se desea que la señal PWM tenga N bits de resolución, habrá que hacer

$$CCR0 = 2^N - 1$$

Igualando ambas tenemos:

$$CCR0 = 2^N - 1 = \frac{F_{TACLK}}{F_s} - 1 \Rightarrow 2^N = \frac{F_{TACLK}}{F_s} \Rightarrow F_{TACLK} = F_s 2^N$$

Que nos liga la frecuencia máxima de una señal PWM en función de la resolución. Si se aumenta la resolución, hay que bajar la frecuencia y viceversa. También se da opción a jugar con los divisores de reloj. La fórmula sólo liga los valores máximos de ambas magnitudes.

Hacer un temporizador con frecuencia precisa (usando decimales)[1].

Suponiendo que el reloj del TimerA tiene una frecuencia f_{TA} y que se quiere generar una señal o interrupción de frecuencia f_s, en general el cociente será un número real, que se truncará al entero inferior automáticamente provocando un error. Se propone la siguiente solución para reducirlo:

```
void taSet (unsigned divisor, unsigned cociente, unsigned resto);
```

donde divisor es f_s. cociente es la parte entera de la división f_{TA}/f_s y resto es el resto de dicha división.

La función guarda los parámetros en sendas variables para que los pueda usar la subrutina de interrupción que se va a encargar de modular el valor del CCR para que sea cociente o cociente+1 con objeto de que el error acumulado sea inferior a divisor en todo momento. En cada interrupción, se actualiza el ccr y se incrementa la variable de error que se guarda internamente para cada canal. Si la misma supera a frecuencia, se incrementa el ccr en cociente+1 y se resta frecuencia al error. Si no, se suma sólo cociente al ccr.

SOLUCIÓN

```
.bss   Divisor, 2    ;Frecuencia destino
.bss   Cociente, 2   ;Valor del CCR (cociente)
.bss   Resto, 2      ;Error absoluto (resto)
.bss   Error, 2      ;Error acumulado
```

1. Punto 8.5.3 (pág 326) del Davies

```
;-------------------------------------------------------------------------------
;  void  taSet  (unsigned  divisor,  unsigned  cociente,  unsigned  resto);
v1.0
;-------------------------------------------------------------------------------
taSet       mov.w  r12, &Divisor ;Guardar divisor
            mov.w  r13, &Cociente;Guardar cociente
            mov.w  r14, &Resto   ;Guardar resto
            clr.w  &Error        ;Empezar con error a 0

            add.w  &TA0R, r13    ;Preparar valor del CCR0 (ojo si no es síncrono)
            mov.w  r13, &TA0CCR0 ;Inicializar CCR0
            mov.w  #CCIE, &TA0CCTL0;Habilitar IRQ (y borrar IFG)
            ;Se supone el TA0 inicializado en modo CONT
            ret
;-------------------------------------------------------------------------------
; TA00ISR                                                                  v1.0
; Suma: 12w19c(media) 14c(modulador=0)/24c(modulador=1)
;-------------------------------------------------------------------------------
TA00ISR     add.w  &Cociente, &TA0CCR0      ;Actualizar tiempo
            add.w  &Resto, &Error           ;3w6c. Acumular error
            cmp.w  &Divisor, &Error         ;3w6c. Acumulada una unidad más?
            jlo    TA00ISRFin               ;1w2c. ...no. Salir
            inc.w  &TA0CCR0                 ;2w4c. ...sí. Acumular
            sub.w  &Divisor, &Error         ;3w6c. Descontar unidad al error
TA00ISRFin  reti
            .intvec TIMER0_A0_VECTOR, TA00ISR
```

Ejemplo: generar una señal de 1KHz con ACLK.
CCR0=32768/1000-1=32’768. Truncando, 31.

La frecuencia real es de 1024 Hz y el error cometido es de +2’4%

Si se usa la técnica anterior:
Dividendo = 32768
Divisor = 1000
Cociente = 32
Resto = 768

Iteración	Delta CCR	Error	Delta CCR ajustado	Error ajustado
0	32	768	32	768
1	32	1536	33	536
2	32	1304	33	304
3	32	1072	33	72
4	32	840	32	840
5	32	1608	33	608
6	32	1376	33	376
7	32	1144	33	144
8	32	912	32	912

9	32	1680	33	680
10	32	1448	33	448
11	32	1216	33	216
12	32	984	32	984
13	32	1752	33	752
14	32	1520	33	520
15	32	1288	33	288
16	32	1056	33	56

El valor promedio de la columa delta CCR es de 32’764, muy cerca del valor 32’768. El error es de -0’01%.

El precio a pagar es una ISR más larga (12W) y compleja que tarda más tiempo. 14c/19c/24c (mínimo/media/máximo). El mínimo se da cuando el modulador=0 y el máximo cuando el modulador=1.
La siguiente versión precalcula los incrementos del CCR en una variable moduladora de 16 bits por eficiencia. Cuanto mayor sea el número de bits de la variable Mod, mejor será la aproximación del error medio a 0.

```
            .bss   Cociente, 2   ;Valor del CCR (cociente)
            .bss   Mod, 2        ;Modulador
;-------------------------------------------------------------------------------
;  void  taSet  (unsigned  divisor,  unsigned  cociente,  unsigned  resto);
v2.0
;-------------------------------------------------------------------------------
taSet       mov.w  r13, &Cociente;Guardar cociente
            add.w  &TA0R, r13    ;Preparar valor del CCR0 (ojo si no es síncrono)
            mov.w  r13, &TA0CCR0 ;Inicializar CCR0
            mov.w  #CCIE, &TA0CCTL0;Habilitar IRQ (y borrar IFG)
            ;Se supone el TA0 inicializado en modo CONT
            ;Calcular modulador en R13
            clr.w  r15           ;R15 = Error
            mov.w  #16, r11      ;Contador de bits del modulador
taSetBuc    rla.w  r13           ;Meter un bit en el modulador (o por defecto)
            add.w  r14, r15      ;Acumular error
            cmp.w  r12, r15      ;El error es menor que el divisor?
            jlo    taSetNo       ;...sí, seguir
            inc.w  r13           ;...no. Meter un 1 en el modulador
            sub.w  r12, r15      ;Ajustar error
taSetNo     dec.w  r11           ;Siguiente bit
            jnz    taSetBuc
            mov.w  r13, &Mod     ;Guardar modulador
            ret
;-------------------------------------------------------------------------------
; TA00ISR                                                                  v2.0
; Suma: 6w12c
;-------------------------------------------------------------------------------
TA00ISR     add.w  &Cociente, &TA0CCR0;3w6c. Actualizar tiempo
            bit.w  #BIT0, &Mod   ;2w4c. Leer BIT0 para simular ROR
            rrc.w  &Mod          ;2w4c. ROR &Mod
            adc.w  &TA0CCR0      ;2w4c. Si bit=1, corregir CCR0
            reti
```

```
.intvec TIMER0_A0_VECTOR, TA00ISR
```