

# TEMA 8: TEMPORIZACIÓN

Sistemas Digitales basados en Microprocesador  
Grado en Ingeniería Telemática

© Raúl Sánchez Reillo





# ÍNDICE

- Conceptos generales, subsistema del reloj, estructura genérica de un temporizador y funcionalidades
- Arquitectura
- Registros de control
- Registros de datos
- Registros de estado
- Funcionalidades “Match” (TOC)
  - Funcionamiento
  - Ejemplos y Ejercicios
- Modulador por anchura de pulso (PWM)
  - Funcionamiento
  - Ejemplos y Ejercicios
- Funcionalidades “Capture” (TIC)
  - Funcionamiento
  - Ejemplos y Ejercicios

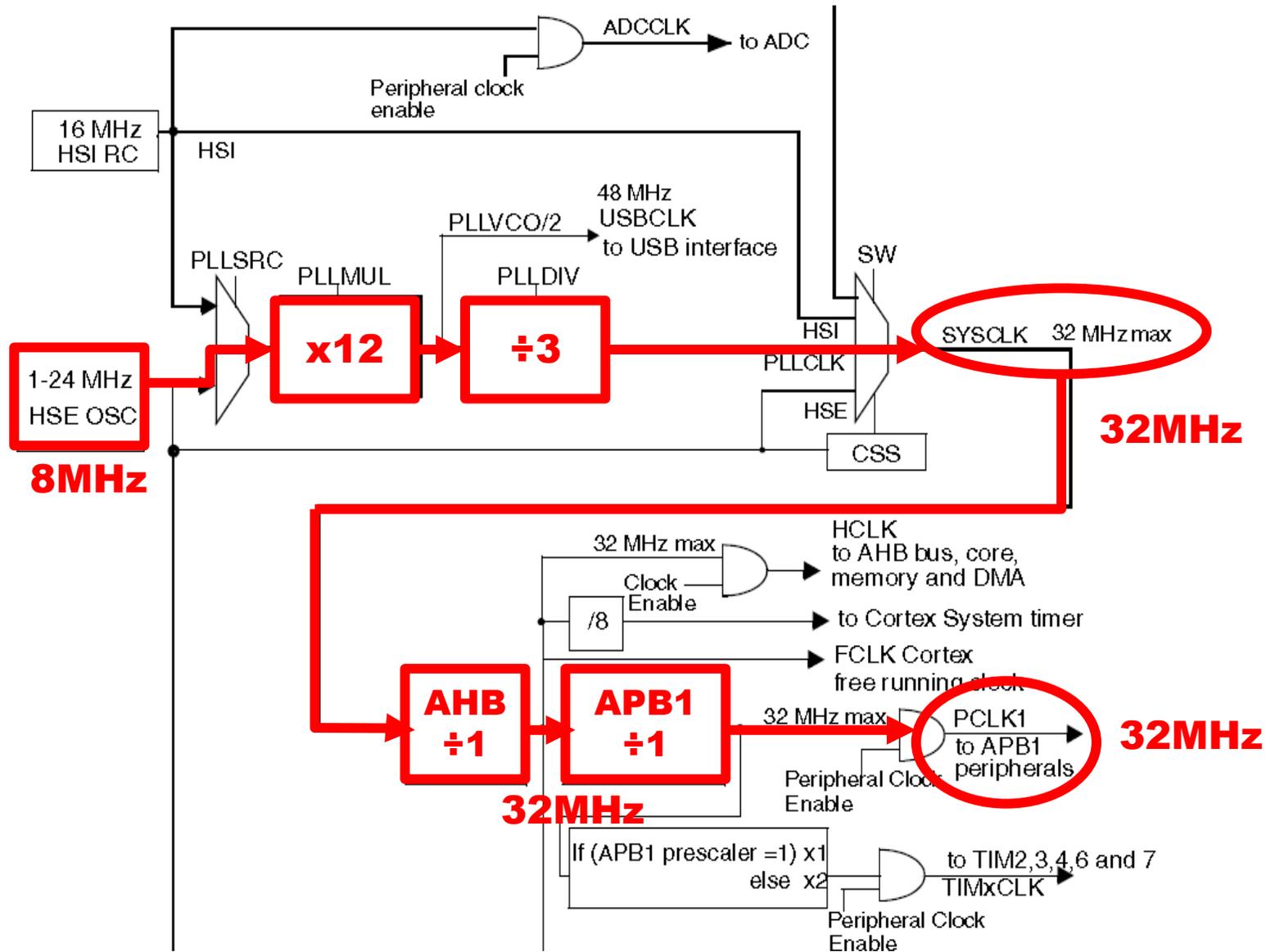


# CONCEPTOS GENERALES

- El subsistema de temporización sirve para generar “unidades de tiempo” tanto para él mismo, como para otros periféricos.
- Sin embargo, un temporizador **NO CUENTA TIEMPO, SINO CICLOS DE RELOJ**.
  - Dicho reloj procede de un **ESCALADO DEL RELOJ DEL SISTEMA (SYSCLK)**.
  - En concreto el escalado se hace respecto al reloj PCLK1, que es el reloj del bus APB1 y que a lo largo del curso será de 32MHz.
  - Adicionalmente a ese escalado, se puede programar unos escalados adicionales (denominados **preescalados**), los cuales suelen ser programables en tiempo de ejecución.
- El reloj resultante de los preescalados será el encargado de marcar el ritmo de cuenta del temporizador.
- El tiempo transcurrido se obtiene multiplicando las cuentas realizadas por el ritmo del temporizador (es decir, el PCLK1 y el preescalado seleccionado).
- Durante el curso los temporizadores a utilizar serán los denominados TIM2, TIM3 y TIM4.



# SUBSISTEMA DE RELOJ DURANTE EL CURSO





# ESTRUCTURA GENÉRICA DE UN TEMPORIZADOR



10MHz  
Tiempo entre pasos = 100ns

16 bits  
Tope de cuenta = 65.536

➔ Tope de tiempo =  $65.536 * 100\text{ns} = 6,553\text{ms}$



# ESTRUCTURA GENÉRICA DE UN TEMPORIZADOR



**PSC = 10000**

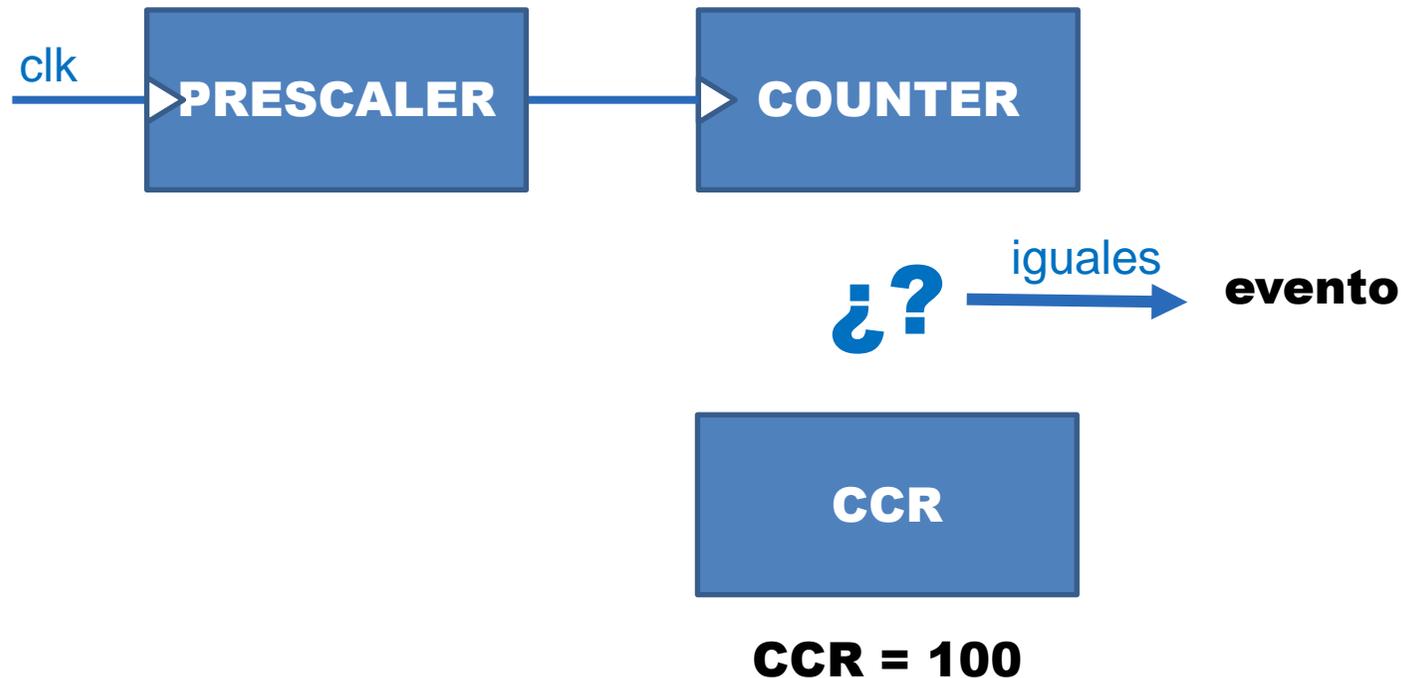
10MHz → Periodo =  $100\text{ns} \cdot \text{PSC} = 1\text{ms}$  → Tope de tiempo = 65,536s



# FUNCIONALIDADES

## ○ TOC (Timer Output Compare):

- Observa el valor del COUNTER hasta que éste llega a un valor previamente registrado en el CCR
- Cuando llega al valor del CCR, se dispara un evento.
- Es el funcionamiento típico de una alarma de un reloj.
- Una variante es el PWM

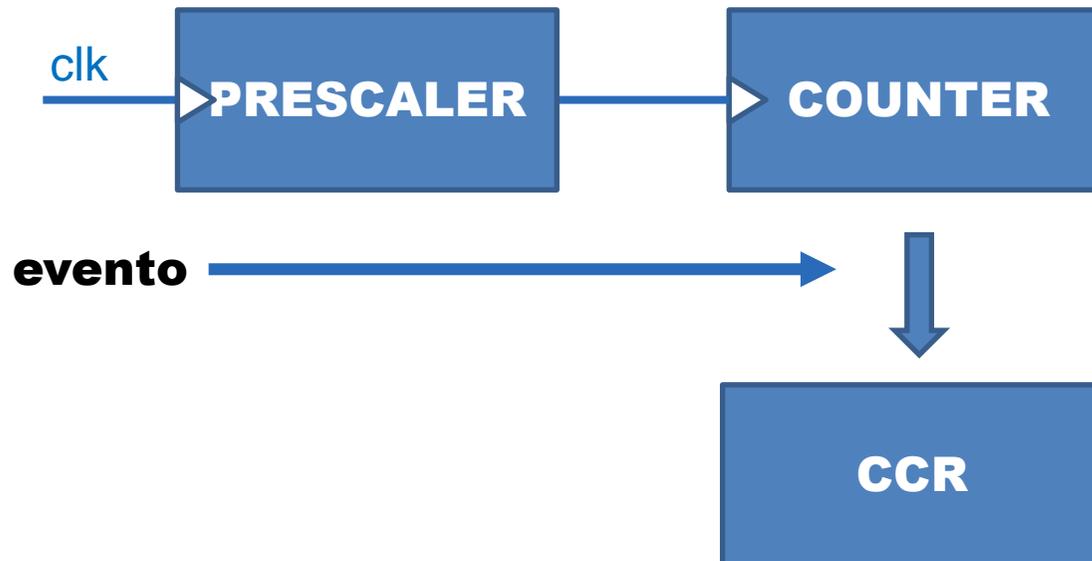




# FUNCIONALIDADES

## ○ TIC (Timer Input Capture):

- Se selecciona un determinado evento de entrada
- Se arranca el COUNTER
- Se queda esperando a que ocurra el evento de entrada
- En cuanto ocurre el evento, se copia el valor del COUNTER en el CCR
- Es la funcionalidad equivalente a cronometrar lo que tarda un atleta en llegar a meta



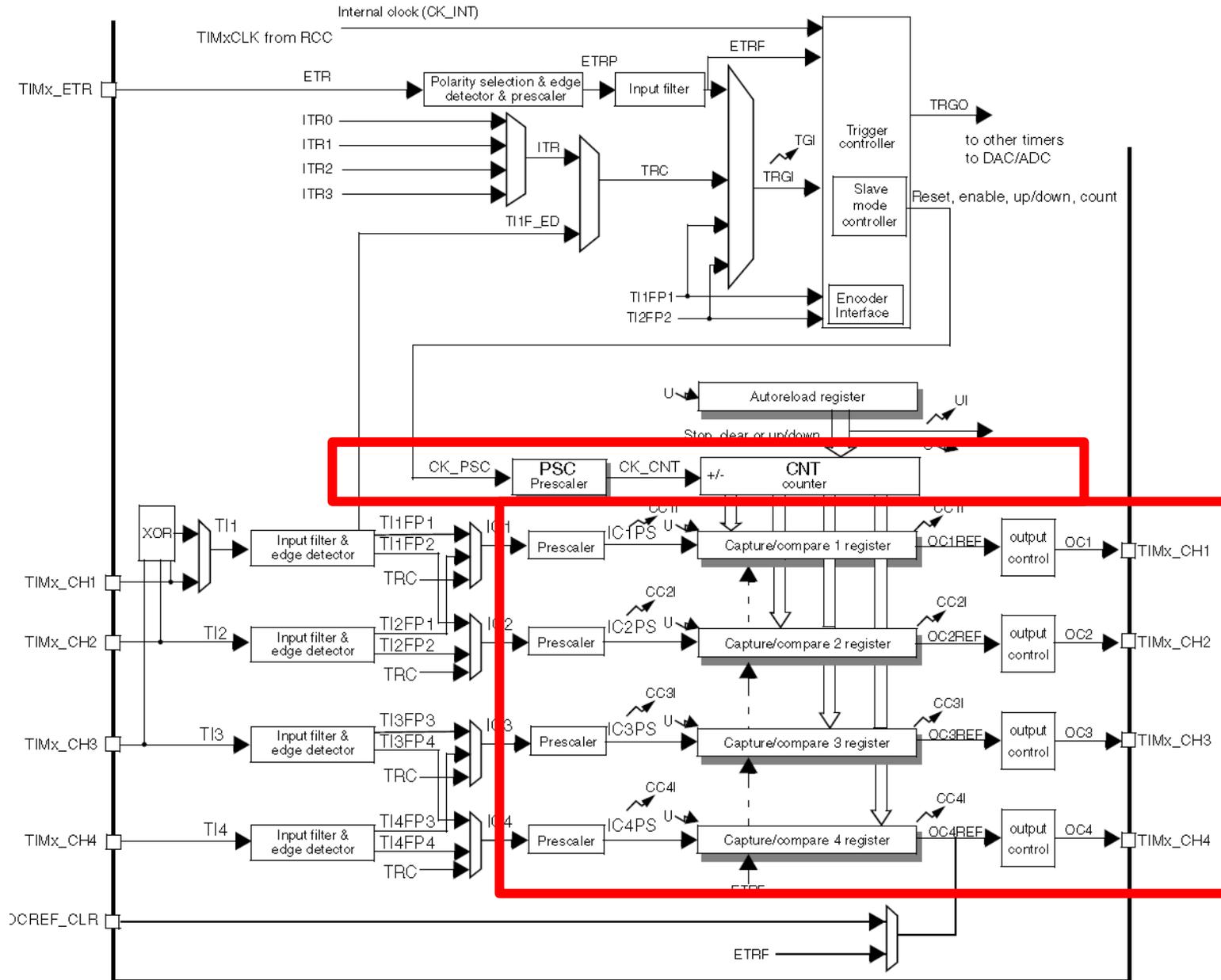


# ARQUITECTURA (1)

- Los TIM2-4 son totalmente independientes
- Cada TIMx posee:
  - Un contador de 16 bits:
    - Contador hacia arriba y hacia abajo (en el curso siempre hacia arriba)
    - Con auto-recarga (para alguna funcionalidad)
    - Con pre-escalado de 16 bits (división por cualquier valor entre 1-65535)
    - 4 canales, cada uno de ellos con funciones independientes (TOC, PWM, TIC) ya que cada canal tiene un CCR diferente, y por tanto cada canal puede implementar una funcionalidad diferente usando el mismo TIM
    - Distintos eventos para generar interrupciones:
      - TIC -> Interrupción al guardar en CCR el valor del temporizador cuando se cumple el evento adecuado
      - TOC -> Interrupción al llegar al valor de CCR
  - Diversidad de funcionalidades:
    - Creación de intervalos de tiempo y generación de formas de onda (TOC)
    - Creación de señales con modulación de la anchura de pulso (PWM)
    - Medida de tiempo (TIC)



# ARQUITECTURA (2)





## ARQUITECTURA (3)

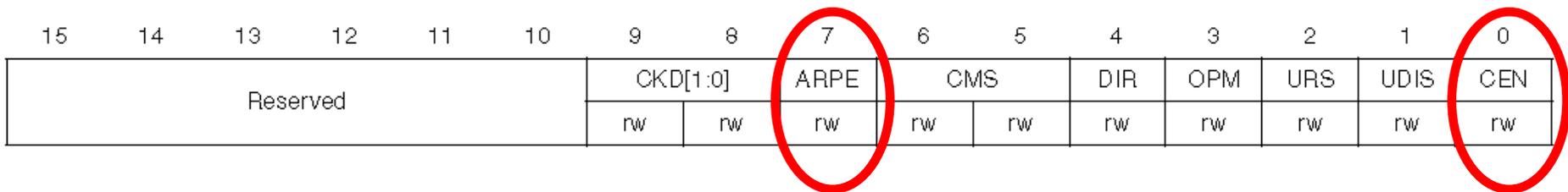
- El temporizador funciona basado en un contador cuyo reloj de entrada procede de:
  - El reloj del APB1 (denominado PCLK1)
  - El preescalado (TIMx\_PSC), que divide la frecuencia del PCLK1 por su valor +1
  - Se establece la base máxima de funcionamiento (TIMx\_ARR), no tiene porqué contar hasta 65.536 sino menos
- Además se escoge la modalidad de funcionamiento del temporizador:
  - Independiente
  - Se decide si se utiliza como TOC, TIC o PWM, y en el caso de TOC, si va a generar una señal de salida, o simplemente se va a utilizar como medidas de tiempo internas
  - Se inicializa el temporizador
  - Se configura la funcionalidad
- Se determina las formas de generación de eventos o interrupciones, y las causas de las mismas.
- Una vez todo configurado, se inicializa el temporizador (bit CEN = 1, en el registro TIMx\_CR1)



# TIMx: REGISTROS DE CONTROL

## ○ TIM<sub>x</sub> → CR1 – Control Register 1:

- Registro de 16 bits, con sólo 10 utilizables:
  - CKD – se pone a ‘0’
  - **ARPE** – Auto-reload preload enable (sólo se usará en PWM)
    - 0 – No se utilizará el TIM<sub>x</sub>\_ARR
    - 1 – Se utilizará el TIM<sub>x</sub>\_ARR
  - CMS, DIR, OPM, URS y UDIS – se ponen a ‘0’
  - **CEN** – Counter enable
    - 0 – contador deshabilitado
    - 1 – contador arrancado





# TIMx: REGISTROS DE CONTROL

## ○ TIMx → CR2 – Control Register 2:

- Registro de 16 bits, que se va a dejar a '0' al completo

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved								THS	MMS[2:0]			CCDS	Reserved			
								rw	rw	rw	rw	rw				

## ○ TIMx → SMCR – Slave Mode Control Register

- Se deja inicializado a 0x0000 puesto que no se va a utilizar este modo

## ○ TIMx → DIER – DMA/Interrupt Enable Register

- TDE, CCyDE, UDE, TIE y UIE –se van a poner a '0'
- CCyIE se pondrán a '1' cuando se vaya a utilizar esa fuente de IRQ, y se mantendrán a '0' cuando no

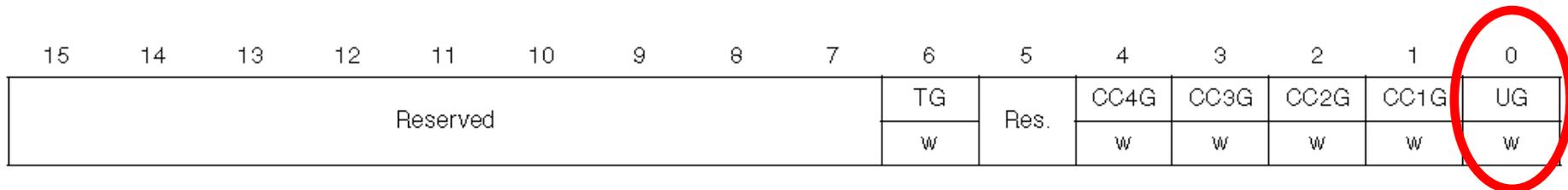
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw



# TIMx: REGISTROS DE CONTROL

## ○ TIM<sub>x</sub> → EGR – Event Generation Register:

- Registro de 16 bits que se utiliza para generar eventos de forma manual.
- El registro no se utilizará de forma habitual.
- De hacerlo, todos los bits se mantendrán a '0' salvo UG, que se pondrá a '1' para refrescar los valores de los registros internos.
  - UG – Se pone un '1' para generar un evento de update, con lo que se actualizan los registros. El hardware pone el bit automáticamente a '0'. Siempre se activa.





# TIMx: REGISTROS DE CONTROL

## TIMx → CCMR1 – Capture/Compare Mode Register 1:

- Registro de 16 bits, para configurar los canales 1 y 2 (existe otro registro equivalente para los canales 3 y 4: TIMx\_CCMR2):
- La configuración inicial se hace escribiendo el bit **CCyS** – Capture/Compare y Selection:
  - 00 – como TOC
  - 01 – como TIC asociado al su propia entrada TIMx
  - 10 – como TIC asociado a la entrada TIMx del otro canal del registro (no se utiliza este modo en el curso)
  - 11 – (no se utiliza este modo en el curso)
- **Para la funcionalidad TOC y PWM**
  - OCyCE – Output Compare y Clear Enable (se pondrá siempre a 0)
  - OCyM – Output Compare y Mode
    - 000 – Sin salida
    - 001 – La salida se pone a 1 tras la comparación exitosa
    - 010 – La salida se pone a 0 tras la comparación exitosa
    - 011 – Toggle
    - 100 – Se fuerza la salida a 0
    - 101 – Se fuerza la salida a 1
    - 110 – PWM con el primer semiciclo a 1
    - 111 – PWM con el primer semiciclo a 0
  - OCyPE – Preload Enable,
    - Se habilita con ‘1’ (se toma el valor de CCRy al generar un update event) y se deshabilita con ‘0’ (CCRy coge el valor inmediatamente, según se escribe en él). Se usa para PWM.
  - OCyFE – (se utiliza ‘0’)
- **Para la funcionalidad TIC**
  - ICyF – Filter (se utiliza 0000 – sin filtrado previo)
  - ICyPSC – Input Capture Prescaler (se utiliza 00 – sin pre-escalado específico)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OC2CE	OC2M[2:0]				OC2PE	OC2FE	CC2S[1:0]			OC1CE	OC1M[2:0]		OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]					IC1F[3:0]		IC1PSC[1:0]					
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	



# TIMx: REGISTROS DE CONTROL

## TIMx → CCER – Capture/Compare Enable Register:

- Para cada canal:
  - Para la funcionalidad TOC y PWM
    - CCyNP – Se deja a ‘0’ en TOC y PWM
    - CCyP – Se deja a ‘0’ en TOC y PWM
    - CCyE – Output Enable
      - Un ‘0’ desactiva la salida hardware, y un ‘1’ la activa.
  - Para la funcionalidad TIC
    - CCyNP:CCyP – Polarity:
      - 00 – activo por flanco de subida
      - 01 – activo por flanco de bajada
      - 10 – (reservado)
      - 11 – activo por ambos flancos
    - CCyE – Output Enable:
      - Un ‘0’ deshabilita la captura, y un ‘1’ la habilita

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw												



# TIMx: REGISTROS DE DATOS

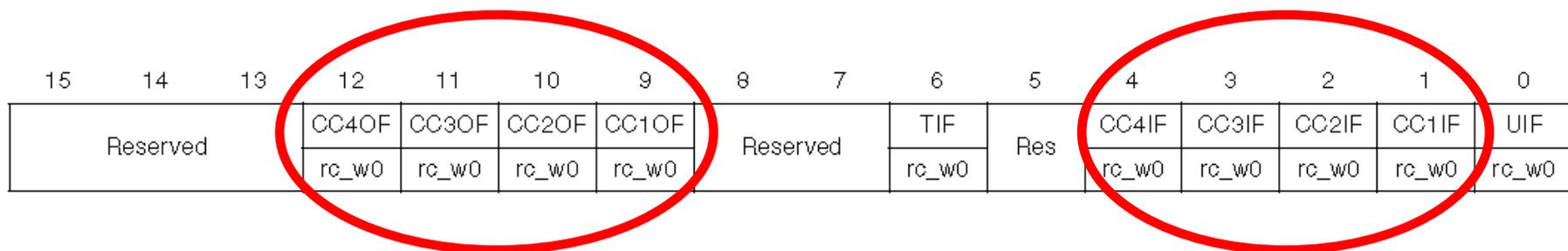
- **TIM<sub>x</sub>→CNT** – Counter
- **TIM<sub>x</sub>→PSC** – Prescaler
  - El valor que se introduzca aquí hace que la frecuencia del reloj del CNT sea:  $f_{CK\_PSC} / (PSC[15:0] + 1)$ 
    - PSC = 0 -> fck/1 (sin preescalado)
    - PCS = 1 -> fck/2 (divido por 2)
    - PSC = 2 -> fck/3 (divido por 3)
  - .....
- **TIM<sub>x</sub>→ARR** – Auto-Reload Register
  - Es el valor que se cargará en el registro interno de auto-reload (sólo lo vamos a usar en PWM)
  - Aunque no se use, hay que escribir un valor que no sea 0 para TOC y TIC por lo que se aconseja ponerlo a 0xFFFF si no se usa.
- **TIM<sub>x</sub>→CCR<sub>y</sub>** – Capture/Compare Register (Channel y)
  - En modo TOC y PWM marca en valor en el que ocurrirá la comparación exitosa
  - En el modo TIC es donde se almacenará el valor del CNT cuando el evento externo ocurra



# TIMx: REGISTROS DE ESTADO

## ○ TIMx→SR – Status Register:

- Registro de 16 bits, que contiene los 10 flags de eventos:
  - CCyOF – 4 flags que indican con un ‘1’ si ha ocurrido un error de overrun (un flag para cada uno de los canales)
    - Para limpiar el flag hay que escribir un ‘0’
  - TIF y UIF – (no se van a utilizar)
  - CCyIF – 4 flags de indican con un ‘1’ que ha ocurrido el evento programado en ese canal.
    - El flag se limpia leyendo el TIMx\_CCRy correspondiente, o escribiendo un ‘0’ en ese bit.



# TEMA 8: TEMPORIZACIÓN

## Funcionalidad Output Compare (TOC)

19





# FUNCIONALIDAD TOC

- Consiste en empezar a contar y cuando llegue a un valor configurado se genera una interrupción o un evento, por ejemplo, una alarma horaria
- Viene caracterizada por configurar el canal en Output Mode:  
**CCyS=00 en  $TIM_x \rightarrow CCMR_y$**
- Cuando hay una comparación exitosa:
  - No activa ningún pin de salida o bien se pone el pin de salida al valor (**OCyM en  $TIM_x \rightarrow CCMR_y$** ). Si se activa, con la polaridad deseada
    - Sin salida:  $OCyM = 000$
    - Activarse:  $OCyM = 001$
    - Desactivarse:  $OCyM = 010$
    - Toogle:  $OCyM = 011$
  - Siempre activa el flag (**CCyIF en  $TIM_x \rightarrow SR$** )
  - Y genera una IRQ (si el bit **CCyIE en  $TIM_x \rightarrow DIER$**  está a 1)
- Los  $TIM_x \rightarrow CCR_y$  se pueden programar usando o no preload (bit **OCxPE en  $TIM_x \rightarrow CCMR_z$** , sólo para PWM pero no lo vamos a usar en el curso)
- Los eventos de update no tienen ningún efecto en la salida



# PROCEDIMIENTO PARA USAR TOC

- Selección del reloj interno
  - $TIM_x \rightarrow CR1$ ,  $TIM_x \rightarrow CR2$ ,  $TIM_x \rightarrow SMCR$
- Configuración del funcionamiento del contador
  - $TIM_x \rightarrow PSC$ ,  $TIM_x \rightarrow CNT$ ,  $TIM_x \rightarrow ARR$ ,  $TIM_x \rightarrow CCR_y$
- $CCyIE = 1$  (en  $TIM_x \rightarrow DIER$ ) si se quiere utilizar IRQ
- Selección del modo de salida
  - $TIM_x \rightarrow CCMR1$ ,  $TIM_x \rightarrow CCMR2$ ,  $TIM_x \rightarrow CCER$
  - Por ejemplo:
    - $OCyM=011$  (toggle en el pin HW asociado al TIMER)
    - $CCyE = 1$  (habilitado)
  - En cualquiera de los casos en los que se use salida hardware, habrá que configurar los correspondientes  $GPIO_x \rightarrow MODER$  y  $GPIO_x \rightarrow AFR$
- Habilitación del contador ( $CEN=1$  en  $TIM_x \rightarrow CR1$ )
- Para actualizar el valor de comparación en  $TIM_x \rightarrow CCR_y$ :
  - Escribirlo directamente con  $OCyPE=0$  (sin preload)
    - Si no, entonces sólo se actualizaría al provocar un “update event”



# EJEMPLO DE USO POR ESPERA ACTIVA (1)

- Partiendo de un PCLK1 de 32MHz, se genera contador de segundos

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"
#include "Utiles_SDM.h"

int main(void){

    short numero=0;
    unsigned char cadena[6];
    Init_SDM();
    Init_LCD();
    LCD_Limpia();

    // Selección del reloj interno: CR1, CR2, SMCR
    TIM4->CR1 = 0x0000; // ARPE = 0 -> No es PWM, es TOC
                        // CEN = 0; Contador apagado
    TIM4->CR2 = 0x0000; // Siempre "0" en este curso
    TIM4->SMCR = 0x0000; // Siempre "0" en este curso

    // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
    TIM4->PSC = 32000; // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000 = 1000 pasos por segundo
    TIM4->CNT = 0; // Inicializo el valor del contador a cero
    TIM4->ARR = 0xFFFF; // Valor recomendado = FFFF
    TIM4->CCR1 = 1000; // Registro donde se guarda el valor que marca la comparación existosa en TOC.
                    // Lo inicializo al valor que quiero llegar = 1000 pasos = 1 sg

    // Selección de IRQ o no: DIER
    TIM4->DIER = 0x0000; // No se genera INT al terminar de contar -> CCyIE = 0
```



## EJEMPLO DE USO POR ESPERA ACTIVA (2)

- Partiendo de un PCLK1 de 32MHz, se genera contador de segundos

```
// Modo de salida del contador
TIM4->CCMR2 = 0x0000; // CCyS = 0 (TOC)
                        // OCyM = 000 (no hay salida por el pin HW asociado al TIM4)
                        // OCyPE = 0 (sin precarga)
TIM4->CCER = 0x0000; // CCyP = 0 (siempre en TOC)
                        // CCyE = 0 (desactivada la salida hardware)

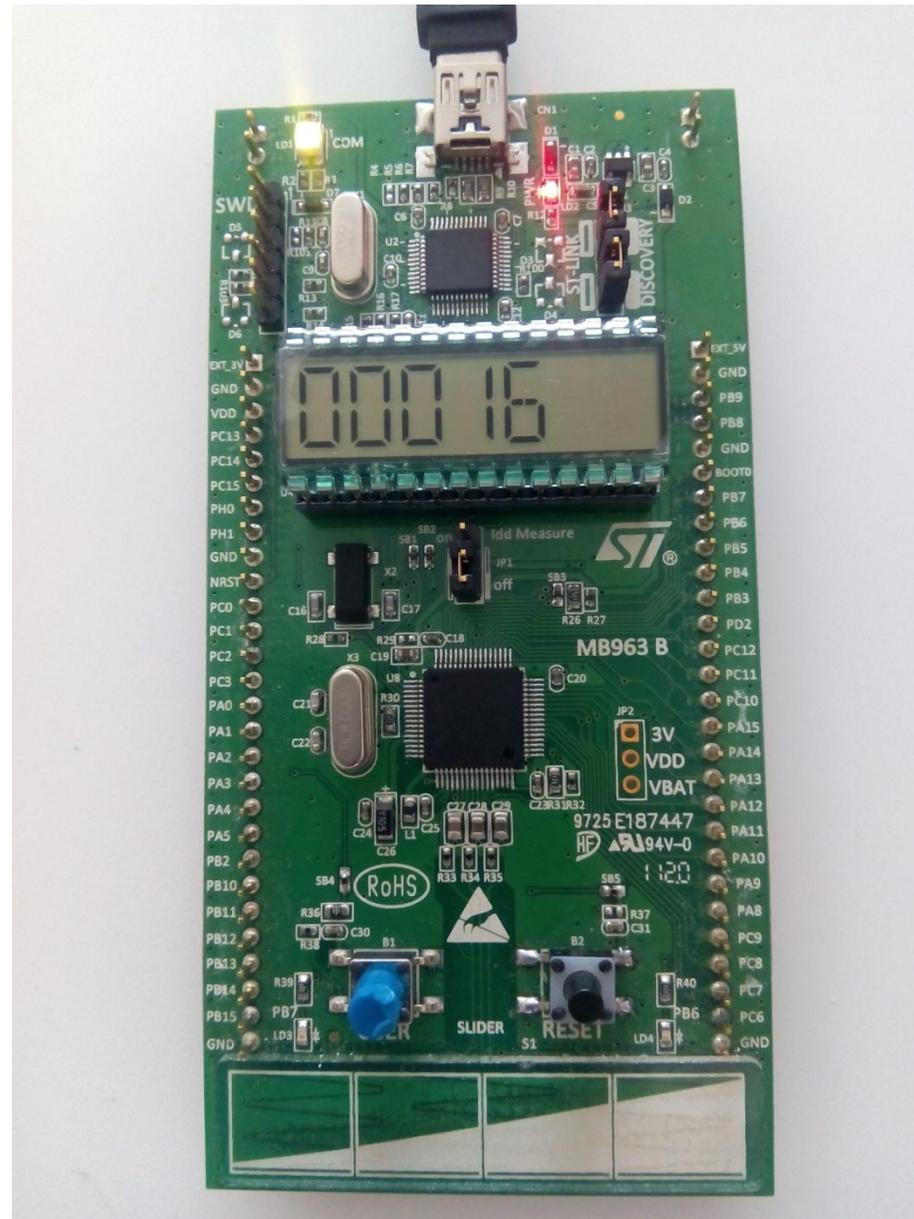
// Habilitación de contador
TIM4->CR1 |= 0x0001; // CEN = 1 -> Arranco el contador
TIM4->EGR |= 0x0001; // UG = 1 -> Se genera evento de actualización
TIM4->SR = 0; // Limpio los flags del contador

while (1)
{
    while ((TIM4->SR&0x0002)==0); // Si no se produce un evento en el canal 1 del TIM4 (TOC), o sea,
                                // no han pasado 1000 pasos = 1 segundo, espero

    TIM4->SR &= ~(0x0002); // Si he llegado, sigo en el programa y limpio el flag

    TIM4->CCR1 += 1000; // Aumento en 1000 pasos mas = 1 sg el valor a contar
    numero++; // Aumento en uno el número a sacar en el LCD
    Bin2Ascii(numero, cadena); // Convierto a ASCII el valor para el LCD
    LCD_Texto(cadena); // Saco el valor en el LCD
}
}
```

# PRUEBA DEL EJEMPLO EXPLICADO





# EJEMPLO DE USO POR IRQs (1)

- El mismo ejemplo pero con IRQs

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"
#include "Utiles_SDM.h"

unsigned short tiempo = 1000;
unsigned short numero=0;

void TIM4_IRQHandler(void) {
    if ((TIM4->SR & 0x0002)!=0) // Si se produce un evento en el canal 1 del TIM4 (TOC), o sea,
                                // han pasado 1000 pasos = 1 segundo, ejecuto la interrupción
    {
        numero++; // Aumento en uno el número a sacar en el LCD
        TIM4->CNT = 0; // Inicializo el valor del contador a cero
        TIM4->CCR1 = tiempo; // Registro donde se guarda el valor que marca la comparación
                                // existosa en TOC.
                                // Vuelvo a ponerlo a 1000 pasos = 1 segundo
        TIM4->SR = 0x0000; // Limpio los flags del contador
    }
}
```



# EJEMPLO DE USO POR IRQs (2)

## ○ El mismo ejemplo pero con IRQs

```
int main(void){
    unsigned char numero_ant = 0;
    unsigned char cadena[6];
    Init_SDM();
    Init_LCD();
    LCD_Limpia();

    // Selección del reloj interno: CR1, CR2, SMRC
    TIM4->CR1 = 0x0000; // ARPE = 0 -> No es PWM, es TOC
                        // CEN = 0; Contador apagado
    TIM4->CR2 = 0x0000; // Siempre "0" en este curso
    TIM4->SMCR = 0x0000; // Siempre "0" en este curso

    // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
    TIM4->PSC = 32000; // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000
                    // = 1000 pasos por segundo
    TIM4->CNT = 0; // Inicializo el valor del contador a cero
    TIM4->ARR = 0xFFFF; // Valor recomendado = FFFF
    TIM4->CCR1 = 1000; // Registro donde se guarda el valor que marca la comparación existosa
                    // en TOC.
                    // Lo inicializo al valor que quiero llegar = 1000 pasos = 1 sg

    // Selección de IRQ o no: DIER
    TIM4->DIER = 0x0002; // Se genera INT al terminar de contar -> CCyIE = 1
```



# EJEMPLO DE USO POR IRQs (3)

## ○ El mismo ejemplo pero con IRQs

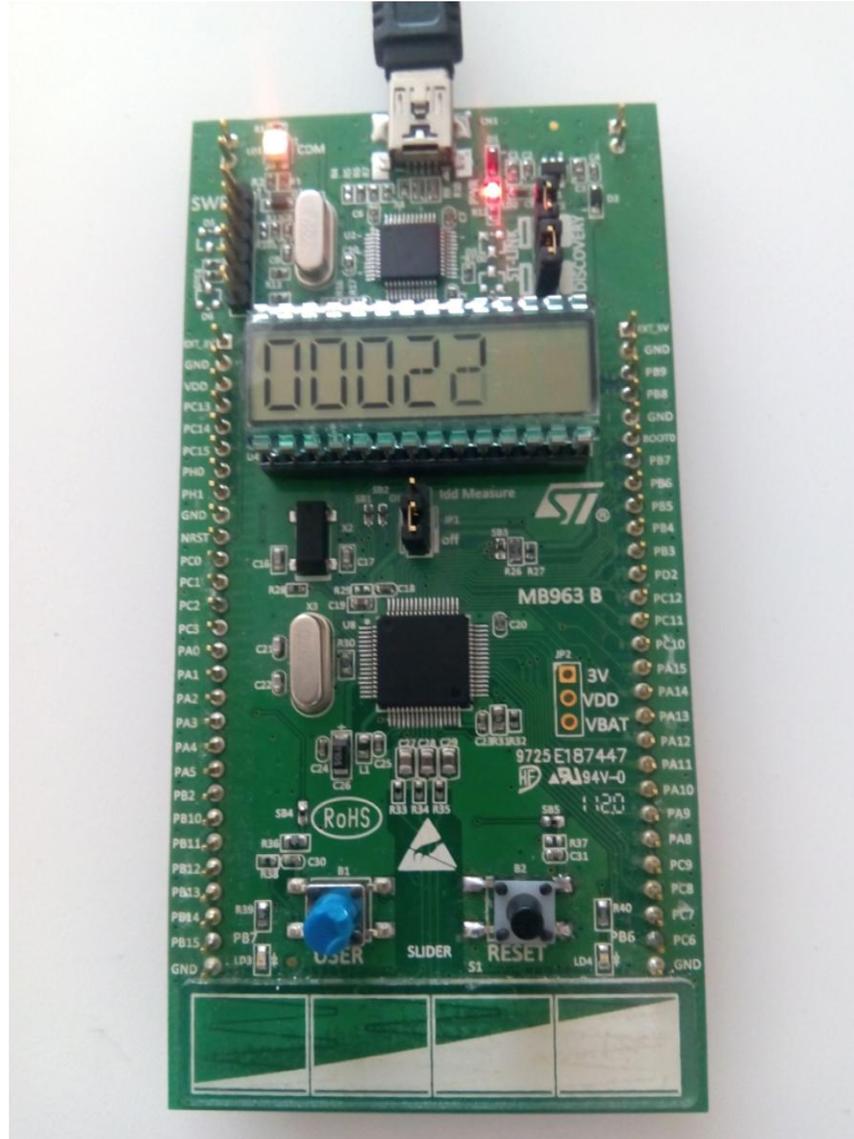
```
// Modo de salida del contador
TIM4->CCMR2 = 0x0000; // CCyS = 0 (TOC)
                        // OCyM = 000 (no hay salida por el pin HW asociado al TIM4)
                        // OCyPE = 0 (sin precarga)
TIM4->CCER = 0x0000; // CCyP = 0 (siempre en TOC)
                        // CCyE = 0 (desactivada la salida hardware)

// Habilitación de contador
TIM4->CR1 |= 0x0001; // CEN = 1 -> Arranco el contador
TIM4->EGR |= 0x0001; // UG = 1 -> Se genera evento de actualización
TIM4->SR = 0; // Limpio los flags del contador

// Habilitación de la interrupción TIM4_IRQ en el NVIC (posición 30).
NVIC->ISER[0] |= (1 << 30);

while (1) {
    if (numero_ant != numero) // Si la interrupción ha cambiado el número a sacar
    {
        numero_ant = numero; // Guardo el número nuevo para la próxima comparación
        Bin2Ascii(numero, cadena); // Convierto el número a texto
        LCD_Texto(cadena); // Saco el número por el LCD
    }
}
```

# PRUEBA DEL EJEMPLO EXPLICADO





## EJEMPLOS Y EJERCICIOS

- 1) Análisis de los ejemplos: Realice el diagrama de flujo de los ejemplos, cree los proyectos y al escribir el código comente lo que hace cada línea (a nivel funcional). Ejecútelos y depúrelo.
- 2) Utilice el TOC por espera activa para hacer conversiones en el ADC cada 2 segundos y muestre el valor en el LCD, y compruebe que sólo se actualiza el valor convertido (y que se saca por el LCD) cada 2 segundos.
- 3) Modifique el programa anterior, para que se use el TOC por interrupciones.
  - El ADC se debe arrancar desde la RAI, pero la comprobación de final de conversión y mostrarlo en el LCD se hará en el programa principal.
- 4) Utilice el TOC por interrupciones, para encender y apagar el LED verde cada 0,5 segundos, 1 segundo, 2 segundos y 5 segundos. El cambio del intervalo se hará cada vez que se pulse el botón de USER, volviendo a 0,5 cuando se pulse el botón, estando en el intervalo de 5 segundos.

# TEMA 8: TEMPORIZACIÓN

## Modulador por Anchura de Pulsos (PWM)

30





# FUNCIONALIDAD PWM

- Existen muchos dispositivos que se controlan mediante la cantidad de energía que se les suministra.
  - Dicha energía se genera como una forma de onda cuadrada de una determinada frecuencia, y donde se controla la cantidad de tiempo que la señal se encuentra a nivel alto (es decir, el “duty cycle” – DC)
    - A mayor DC, mayor energía.
    - A menor DC, menor energía.
  - Por lo tanto el periodo de la señal no se varía nunca, sino solamente el DC.
  - Con el mismo periodo de señal se pueden generar distintas señales de control PWM.
- Esto es aplicable a:
  - Motores
  - Control de potencia en reguladores conmutados
  - Control de intensidad luminosa de LEDs de baja y alta potencia
  - Etc.



# FUNCIONALIDAD PWM

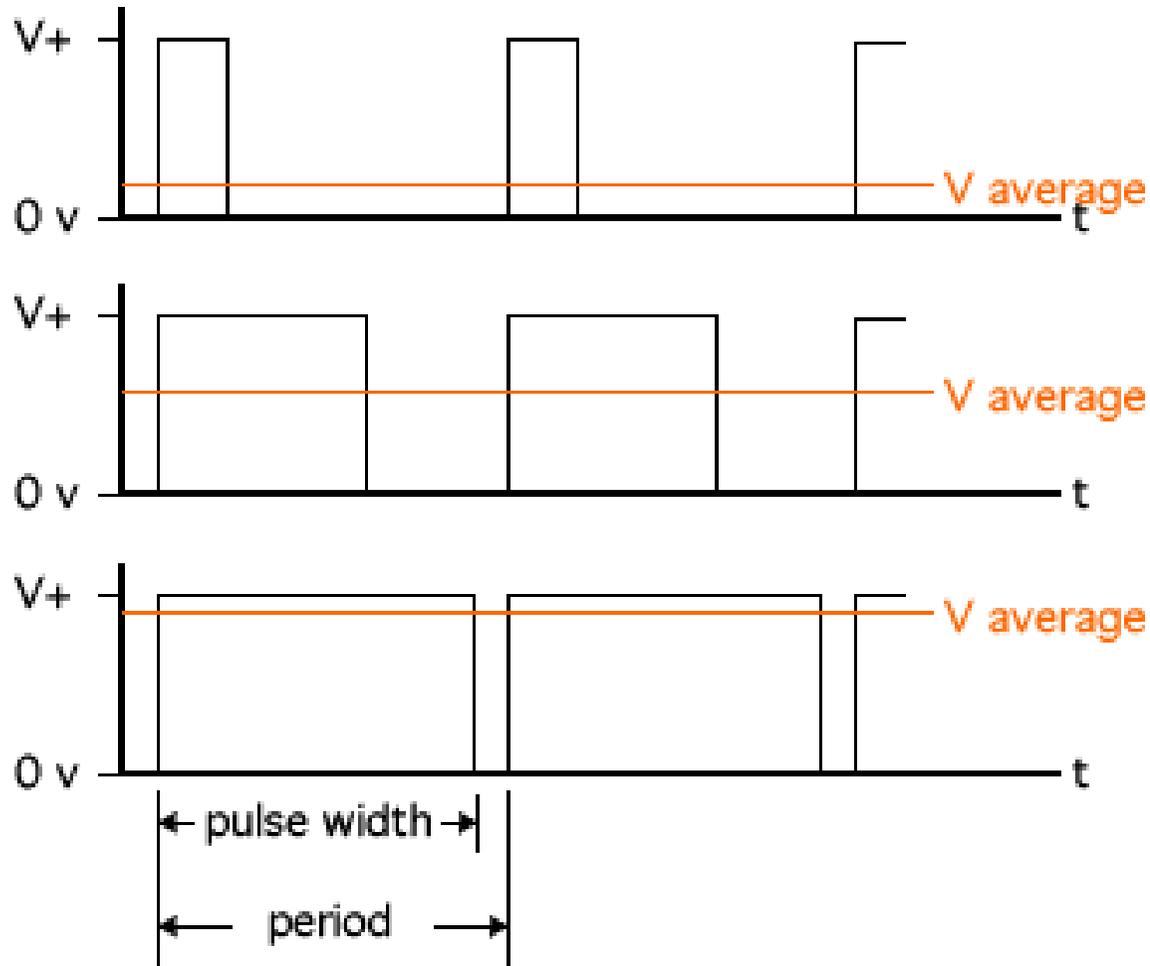


Imagen tomada de:

<http://meteoyelectronica.blogspot.com/2010/12/pwm-una-manera-sencilla-de-controlar-un.html>



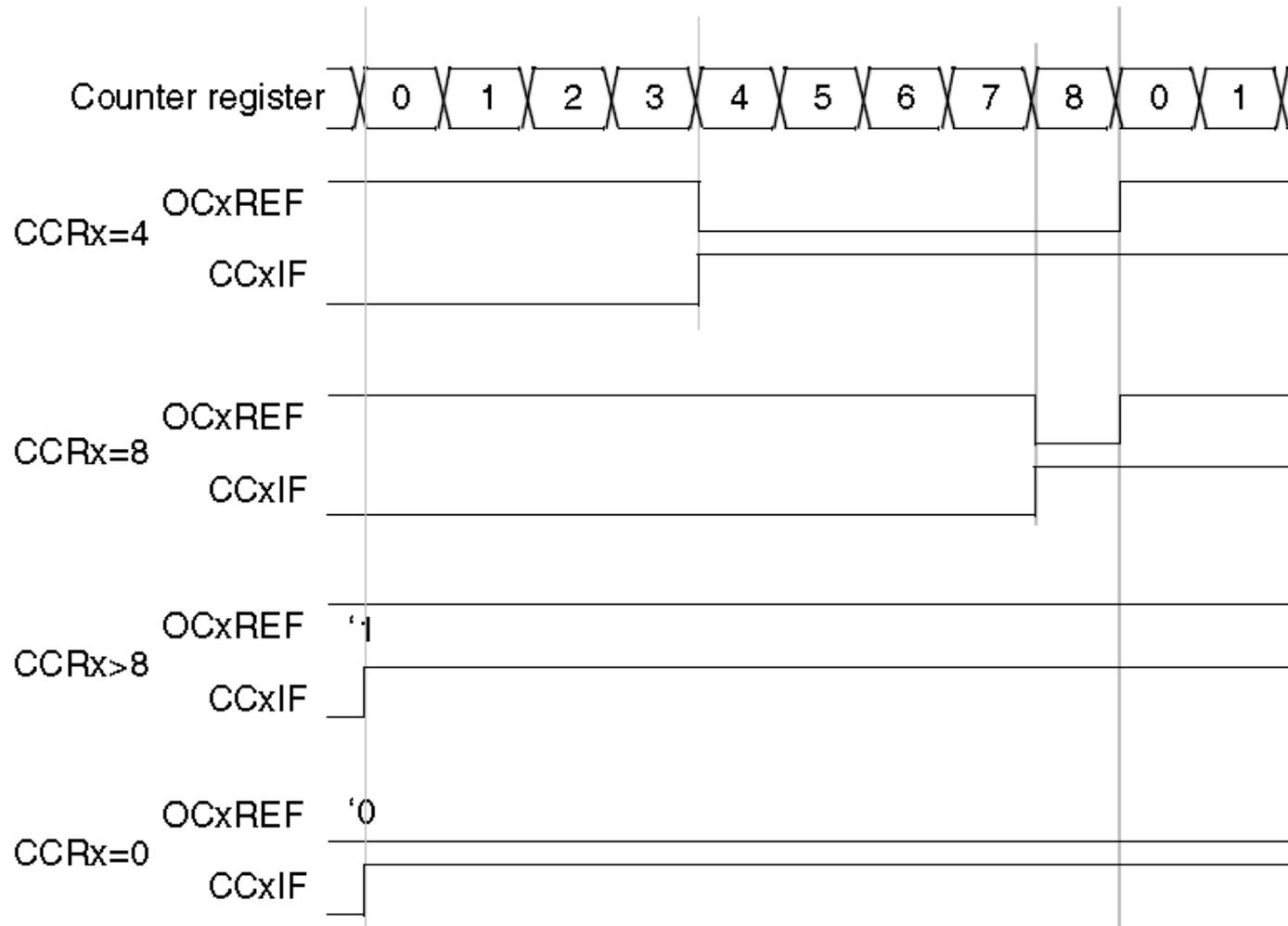
# PROCEDIMIENTO DE USO DEL PWM

- Hay que configurar el reloj interno y el preescalado para tener una definición del tiempo de cuenta del contador
  - $TIM_x \rightarrow CR1$ ,  $TIM_x \rightarrow CR2$ ,  $TIM_x \rightarrow SMCR$ ,  $TIM_x \rightarrow PSC$
- El periodo lo determina el valor de  $TIM_x \rightarrow ARR$
- El DC viene controlado por  $TIM_x \rightarrow CCRy$
- Se selecciona la funcionalidad PWM en aquellos canales que se quiera. Cada salida puede configurarse independiente con uno de los 2 modos que tiene ( $TIM_x \rightarrow CCMRz$ )
  - $OCyM = 110$  (inicio de ciclo a nivel alto -> lógica positiva)
  - $OCyM = 111$  (inicio de ciclo a nivel bajo -> lógica negativa)
- En la funcionalidad PWM es obligatorio:
  - El preload en la carga de los  $CCRy$  (para evitar DCs inesperados) ( $OCyPE = 1$  en  $TIM_x \rightarrow CCMRz$ )
  - El autoreload preload register ( $ARPE=1$  en  $TIM_x \rightarrow CR1$ )
  - Antes de iniciar el temporizador, inicializar los registros con  $UG=1$  en  $TIM_x \rightarrow EGR$
- La polaridad de la señal se programa con los bits  $CCyP$ , y la salida se habilita activando  $CCyE$  ( $TIM_x \rightarrow CCERy$ )



# EJEMPLO DE USO DEL PWM (GRÁFICO)

- Ejemplo de uso con  $TIM_x \rightarrow ARR = 8$  y distintos  $TIM_x \rightarrow CCR_y$ 
  - Se muestra tanto la salida  $OCyREF$ , como el flag  $CCyIF$





# EJEMPLO DE USO DE PWM (1)

- Partiendo de un reloj de 32MHz, se genera una señal PWM de 100Hz, cambiando el DC de 10 en 10%, cada vez que se pulsa PA0. La salida se hace por PB7 y el LED verde cambiará de intensidad según varíe el DC, mientras que el DC se muestra por el LCD

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"
#include "Utiles_SDM.h"

int main(void){
    short DC=1;
    unsigned char cadena[6];
    Init_SDM();
    Init_LCD();
    LCD_Limpia();

    // PB7 como salida para el TIM4
    GPIOB->MODER|=0x00000001 << (2*7 +1); // MODER = 10 (AF) para el bit 7 del puerto B
    GPIOB->MODER&=~(0x00000001 << (2*7));
    GPIOB->AFR[0]|=(0x02 << (7*4)); // AFR[0] = 0x0200000000000000 para decir que el pin
    // 7 del puerto B tiene la función especial 2 (0010) = TIM4

    // PA0 (Boton USER) como entrada
    GPIOA->MODER &= ~(1 << (0*2 +1)); // MODER = 00 (entrada) para el bit 0 del puerto A
    GPIOA->MODER &= ~(1 << (0*2));
    GPIOA->PUPDR &= ~(11 << (0*2)); // PUPDR = 00 - Sin pull-up ni pull-down

    // Selección del reloj interno: CR1, CR2, SMCR
    TIM4->CR1 = 0x0080; // ARPE = 1 -> Es PWM
    // CEN = 0; Contador apagado
    TIM4->CR2 = 0x0000; // Siempre "0" en este curso
    TIM4->SMCR = 0x0000; // Siempre "0" en este curso

    // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
    TIM4->PSC = 32000; // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000 = 1000 pasos por segundo
    TIM4->CNT = 0; // Inicializo el valor del contador a cero
    TIM4->ARR = 9; // Pongo una frecuencia PWM de 100 Hz, sólo cuento 10 pasos (de 0 a 9) y empiezo de nuevo
    TIM4->CCR2 = DC; // El Duty cycle se pone a 1 inicialmente
```



## EJEMPLO DE USO DE PWM (2)

- Partiendo de un reloj de 32MHz, se genera una señal PWM de 100Hz, cambiando el DC de 10 en 10%, cada vez que se pulsa PA0. La salida se hace por PB7 y el LED verde cambiará de intensidad según varíe el DC, mientras que el DC se muestra por el LCD

```
// Selección de IRQ o no: DIER
TIM4->DIER = 0x0000; // No se genera INT al terminar de contar -> CCyIE = 0

// Modo de salida
TIM4->CCMR1 = 0x6800; // CCyS = 0 (TOC, PWM)
// OCyM = 110 (PWM con el primer semiciclo a 1)
// OCyPE = 1 (con precarga)

TIM4->CCER = 0x0010; // CCyP = 0 (siempre en PWM)
// CCyE = 1 (activada la salida hardware)

// Habilitación de contador
TIM4->CR1 |= 0x0001; // CEN = 1 -> Arranco el contador
TIM4->EGR |= 0x0001; // UG = 1 -> Se genera evento de actualización
TIM4->SR = 0; // Limpio los flags del contador

while (1) {
    if ((GPIOA->IDR&0x00000001)!=0) // Si se pulsa el botón USER
    {
        while ((GPIOA->IDR&0x00000001)!=0) // Espero un poco para evitar los rebotes
        {
            espera(70000);
        }

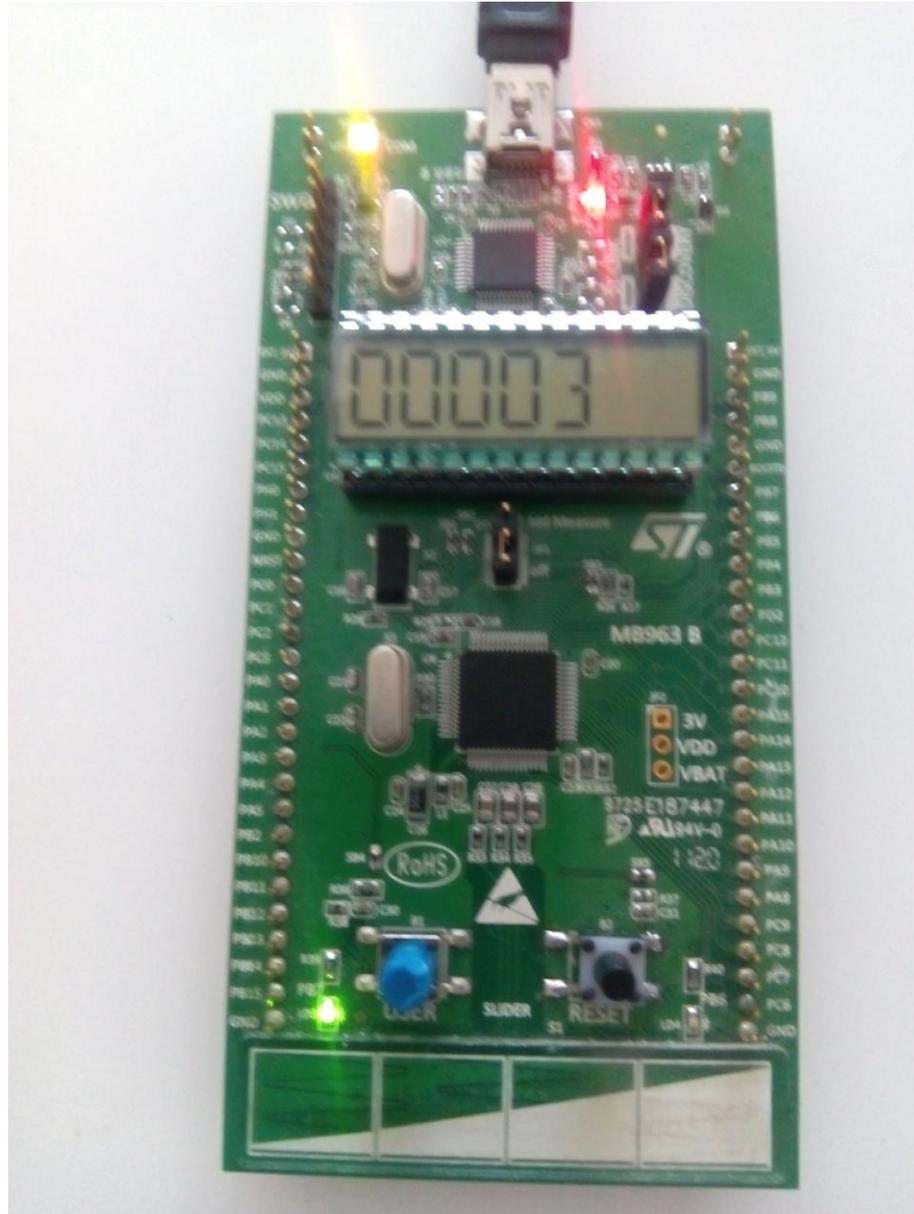
        DC+=1; // Modifico el DC cada vez que pulse
        if (DC >= TIM4->ARR) DC = 1; // Si el DC es mayor que 10, se pone a uno de nuevo

        TIM4->CCR2 = DC; // Guardo el nuevo DC en CCR2

        Bin2Ascii(DC,cadena); // Convierte el DC a texto
        LCD_Texto(cadena); // Saca el DC por el LCD
    }
}
```



# PRUEBA DEL EJEMPLO EXPLICADO





## EJERCICIOS

- 1) Análisis del ejemplo: Realice el diagrama de flujo del ejemplo, cree el proyecto y al escribir el código comente lo que hace cada línea (a nivel funcional). Ejecútelo y depúrelo.
- 2) Modifique el ejemplo para conseguir el mismo efecto pero mostrando un parpadeo en el LED.
- 3) Conecte un filtro RC paso bajo a la salida del PWM (por la salida PB7) y lleve el resultado a una entrada del conversor ADC. Modifique el ejemplo para sacar el valor medio de la señal por el LCD, para cada una de las señales generadas en PWM al pulsar el botón USER.

# TEMA 8: TEMPORIZACIÓN

## Funcionalidad Capture (TIC)

39





# FUNCIONALIDAD TIC

- La funcionalidad TIC se utiliza para medir el tiempo que pasa entre eventos externos:
  - Por lo tanto hay que definir el tipo de evento que se va a medir:
    - Flanco de subida, flanco de bajada o ambos flancos
  - Hay que mantener el conocimiento del valor del contador antes de empezar a contar
  - Una vez ocurrido el evento, el valor del contador se copiará en el registro  $TIMx \rightarrow CCRx$  correspondiente
  - El programa calcula el tiempo con la diferencia, teniendo en cuenta que el contador puede haber dado la vuelta y teniendo en cuenta el tiempo de cuenta programado.



# PROCEDIMIENTO PARA USAR EL TIC

- Configurar el pin para funcionalidad TIM<sub>x</sub>
  - GPIO<sub>x</sub>→MODER, GPIO<sub>x</sub>→AFR
- Configurar el reloj
  - TIM<sub>x</sub>→CR1, TIM<sub>x</sub>→CR2, TIM<sub>x</sub>→SMCR
- Configurar el ritmo de cuenta del contador
  - TIM<sub>x</sub>→PSC, TIM<sub>x</sub>→CNT, TIM<sub>x</sub>→ARR
- Configurar la entrada activa (CCyS=01 en TIM<sub>x</sub>→CCMR<sub>x</sub>)
- Seleccionar el flanco activo para el evento (CCyP y CCyNP en TIM<sub>x</sub>→CCER)
- Habilitar la captura (CCyE=1 en TIM<sub>x</sub>→CCER)
- Si se quieren utilizar interrupciones, activar el CCxIE de TIM<sub>x</sub>→DIER
- Limpiar el flag (TIM<sub>x</sub>→SR)
- Iniciar el contador (CEN=1 en TIM<sub>x</sub>→CR1)
- Al producirse el evento de captura:
  - Leer el contenido de TIM<sub>x</sub>→CCR<sub>y</sub>
  - Limpiar el flag en TIM<sub>x</sub>→SR



# EJEMPLO DE USO POR ESPERA ACTIVA (1)

- Partiendo de un PCLK1 de 32MHz, se cuenta el tiempo que se tarda entre pulsaciones del botón USER, mostrándolo en ms en el LCD.
  - **NOTA:** Como el PA0 también se utiliza como LCD, al ubicar el PA0 como entrada, se pueden producir interferencias en el LCD

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"
#include "Utiles_SDM.h"

int main(void){
    unsigned char cadena[6];
    unsigned short tiempo_inicio = 0;
    int tiempo;
    Init_SDM();
    Init_LCD();
    LCD_Limpia();

    // PA0 (Boton USER) como AF para el TIM2
    GPIOA->MODER|=0x00000001 << (2*0 +1); // MODER = 10 (AF) para el bit 0 del puerto A
    GPIOA->MODER&=~(0x00000001 << (2*0));
    GPIOA->AFR[0]|=(0x01 << (0*4)); // AFR[0] = 0x0000000000000001 para decir que el pin
    // 0 del puerto A tiene la función especial 1 (0001) = TIM2
    GPIOA->AFR[0]&=~(0x0E << (0*4));

    // Selección del reloj interno: CR1, CR2, SMCR
    TIM2->CR1 = 0x0000; // ARPE = 0 -> No es PWM, es TIC
    // CEN = 0; Contador apagado
    TIM2->CR2 = 0x0000; // Siempre "0" en este curso
    TIM2->SMCR = 0x0000; // Siempre "0" en este curso

    // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
    TIM2->PSC = 32000; // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000 = 1000 pasos por segundo
    TIM2->CNT = 0; // Inicializo el valor del contador a cero
    TIM2->ARR = 0xFFFF; // Valor recomendado si no es PWM
```



## EJEMPLO DE USO POR ESPERA ACTIVA (2)

- Partiendo de un PCLK1 de 32MHz, se cuenta el tiempo que se tarda entre pulsaciones del botón USER, mostrándolo en ms en el LCD.
  - **NOTA:** Como el PA0 también se utiliza como LCD, al ubicar el PA0 como entrada, se pueden producir interferencias en el LCD

```
// Selección de IRQ o no: DIER
TIM2->DIER = 0x0000;    // No se genera INT al terminar de contar -> CCyIE = 0

// Modo de salida del contador
TIM2->CCMR1 = 0x0001;  // CCyS = 1          (TIC)
                       // OCyM = 000      (siempre en TIC)
                       // OCyPE = 0       (siempre en TIC)
TIM2->CCER = 0x0001;  // CCyNP:CCyP = 00 (activo a flanco de subida)
                       // CCyE = 1        (captura habilitada para TIC)

// Habilitación de contador
TIM2->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
TIM2->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
TIM2->SR = 0;         // Limpio los flags del contador

while (1) {
    while ((TIM2->SR&0x0002)==0);    // Mientras no se cumpla el evento del TIC, espero

    TIM2->SR &= ~(0x0002);           // Una vez que se activa el evento, limpio el flag asociado
    tiempo = TIM2->CCR1 - tiempo_inicio; // El tiempo transcurrido es la resta del tiempo TIM2->CCR1 menos
                                        // el tiempo de inicio, que inicialmente es 0

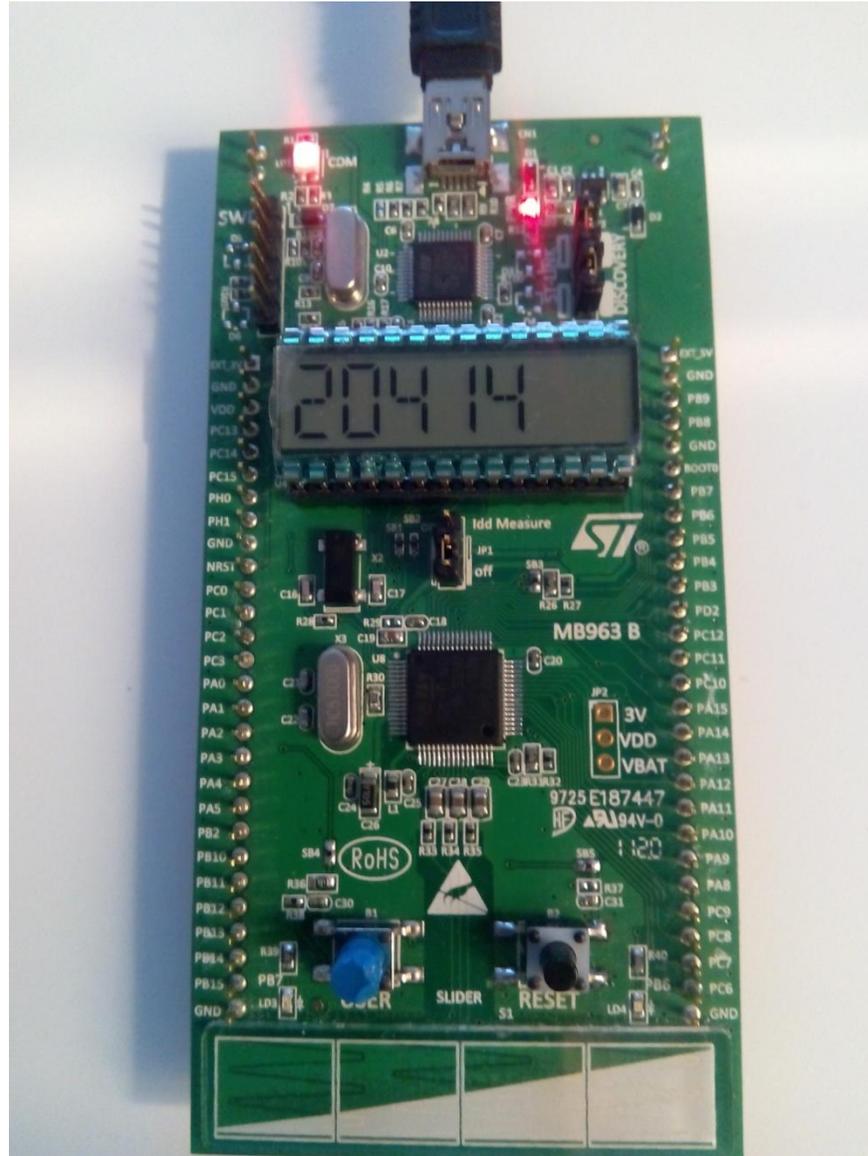
    if (tiempo<0) tiempo += 0x0FFFF; // Para evitar que de la vuelta al contador, le doy la vuelta
                                        // y me aseguro que la diferencia es correcta

    tiempo_inicio = TIM2->CCR1;      // El nuevo tiempo inicio para la próxima vez es el tiempo actual

    Bin2Ascii((unsigned short) tiempo, cadena); // Convierto el número a texto
    LCD_Texto(cadena);                       // Saco el texto por el LCD
}
}
```



# PRUEBA DEL EJEMPLO EXPLICADO





# EJEMPLO DE USO POR INTERRUPCIONES (1)

## o Mismo ejemplo pero usando las IRQs de TIM2

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"
#include "Utiles_SDM.h"

unsigned char pulsacion=0;
unsigned short tiempo_inicio = 0;
int tiempo;

void TIM2_IRQHandler(void) {
    if ((TIM2->SR & 0x0002)!=0) // Si se cumple el evento del TIC
    {
        pulsacion=1; // Pongo esto a 1 para que lo sepa el programa principal
        tiempo = TIM2->CCR1 - tiempo_inicio; // El tiempo transcurrido es la resta del tiempo TIM2->CCR1 menos 0
        if (tiempo<0) tiempo += 0x0FFFF; // Para evitar que de la vuelta al contador, le doy la vuelta
        // y me aseguro que la diferencia es correcta
        tiempo_inicio = TIM2->CCR1; // El nuevo tiempo inicio para la próxima vez es el tiempo actual
        TIM2->SR = 0x0000; // Limpio los flags del contador
    }
}

int main(void){
    unsigned char cadena[6];
    Init_SDM();
    Init_LCD();
    LCD_Limpia();

    // PA0 (Boton USER) como AF para el TIM2
    GPIOA->MODER|=0x00000001 << (2*0 +1); // MODER = 10 (AF) para el bit 0 del puerto A
    GPIOA->MODER&=~(0x00000001 << (2*0));
    GPIOA->AFR[0]|=(0x01 << (0*4)); // AFR[0] = 0x0000000000000001 para decir que el pin
    // 0 del puerto A tiene la función especial 1 (0001) = TIM2
    GPIOA->AFR[0]&=~(0x0E << (0*4));

    // Selección del reloj interno: CR1, CR2, SMCR
    TIM2->CR1 = 0x0000; // ARPE = 0 -> No es PWM, es TIC
    // CEN = 0; Contador apagado
    TIM2->CR2 = 0x0000; // CCyIE = 0 -> No se provoca interrupción con el TIMER2
    TIM2->SMCR = 0x0000; // Siempre "0" en este curso
```



# EJEMPLO DE USO POR INTERRUPCIONES (2)

## o Mismo ejemplo pero usando las IRQs de TIM2

```
// Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
TIM2->PSC = 32000;      // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000 = 1000 pasos por segundo
TIM2->CNT = 0;         // Inicializo el valor del contador a cero
TIM2->ARR = 0xFFFF;   // Valor recomendado si no es PWM

// Selección de IRQ o no: DIER
TIM2->DIER = 0x0002;   // Se genera INT al terminar de contar -> CCyIE = 1

// Modo de salida del contador
TIM2->CCMR1 = 0x0001;  // CCyS = 1          (TIC)
                        // OCyM = 000      (siempre en TIC)
                        // OCyPE = 0       (siempre en TIC)
TIM2->CCER = 0x0001;  // CCyNP:CCyP = 00 (activo a flanco de subida)
                        // CCyE = 1        (captura habilitada para TIC)

// Habilitación de contador
TIM2->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
TIM2->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
TIM2->SR = 0;         // Limpio los flags del contador

NVIC->ISER[0] |= (1 << 28); // Habilita la IRQ del TIM 2 en el NVIC (posición 28)

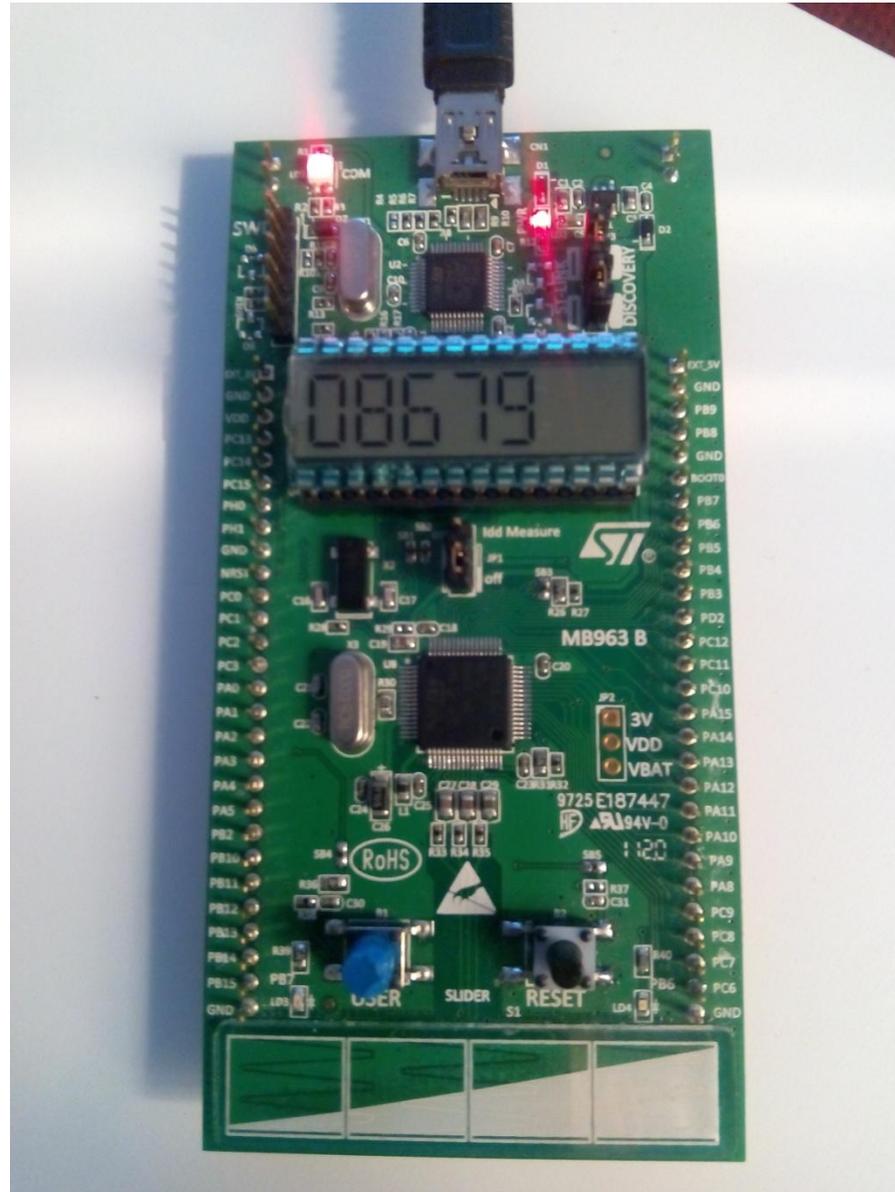
while (1) {
    while (pulsacion==0);    // Si no pulso, espero

    pulsacion=0;            // Si pulso lo pongo a 0 para la próxima interrupción

    Bin2Ascii((unsigned short)(tiempo), cadena); // Convierto el valor de tiempo a texto
    LCD_Texto(cadena);      // Saco el tiempo por el LCD
}
}
```



# PRUEBA DEL EJEMPLO EXPLICADO





## EJERCICIOS

- 1) Análisis de los ejemplos: Realice el diagrama de flujo de los ejemplos, cree los proyectos y al escribir el código comente lo que hace cada línea (a nivel funcional). Ejecútelos y depúrelos.
- 2) Basándose en el ejemplo del PWM y del TIC, genere la señal PWM y use el TIC para medir la frecuencia.
  - Para ello la salida por PB7 se debe conectar a la entrada PA5.
  - Considere si prefiere utilizar interrupciones o no.
- 3) Modifique el ejemplo anterior, para medir el duty cycle en lugar de medir la frecuencia.