uc3m

Departamento Tecnología Electrónica
dte UC3M

Tema 8: Temporizadores

Sistemas Digitales Basados en Microprocesadores

Universidad Carlos III de Madrid

Dpto. Tecnología Electrónica

Nota: Las figuras utilizadas para ilustrar las características y funcionalidades del microcontrolador del curso se han obtenido de la documentación técnica disponible en https://www.st.com/en/microcontrollers-microprocessors/stm32l151-152.html

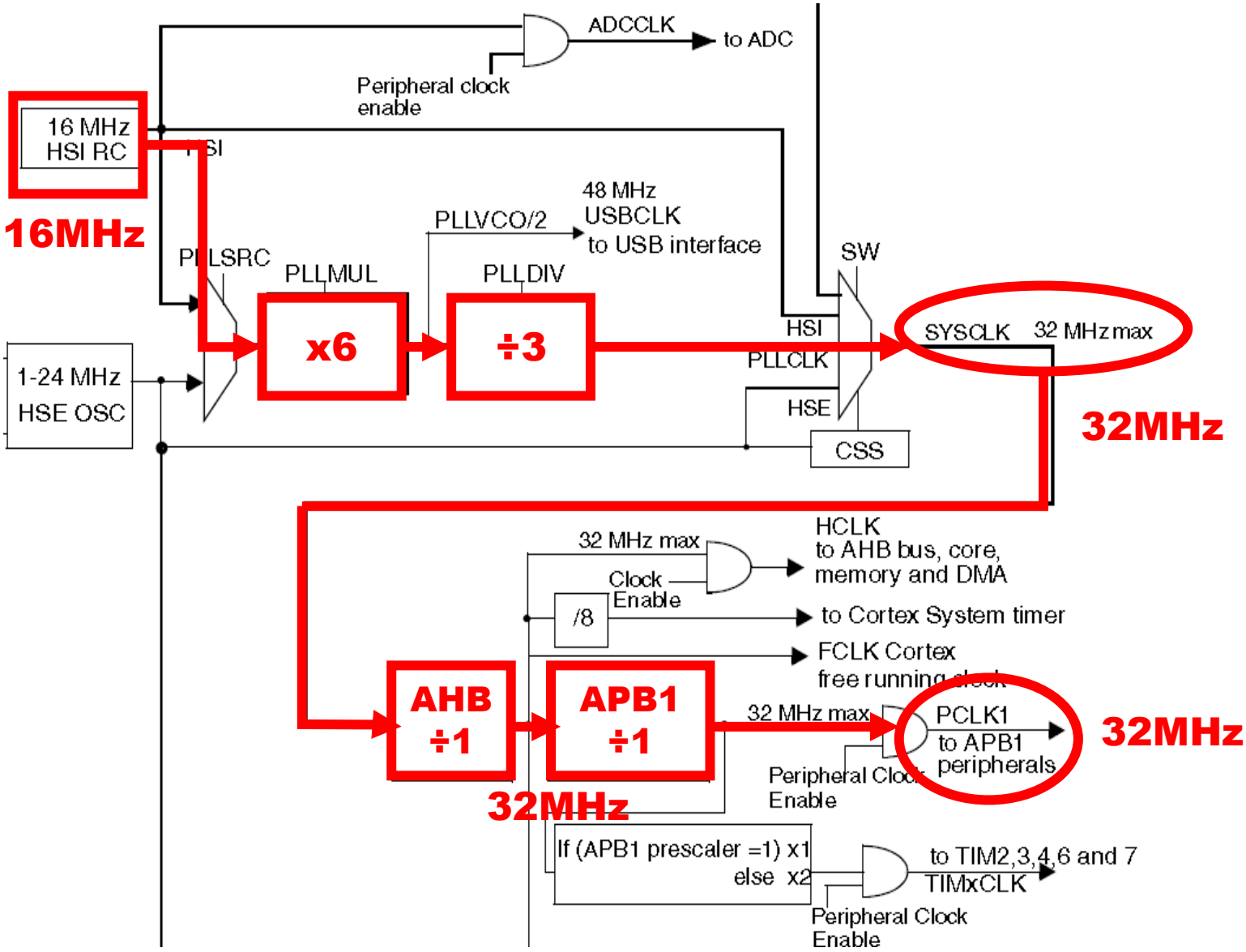
Índice

Conceptos Generales

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

16 bits

Tope de cuenta = 65.536

10MHz

Tiempo entre pasos = 100ns

Tope de tiempo = 65.536 * 100ns = 6,553ms

Estructura Genérica de un Temporizador

16 bits
Tope de división = 65.536

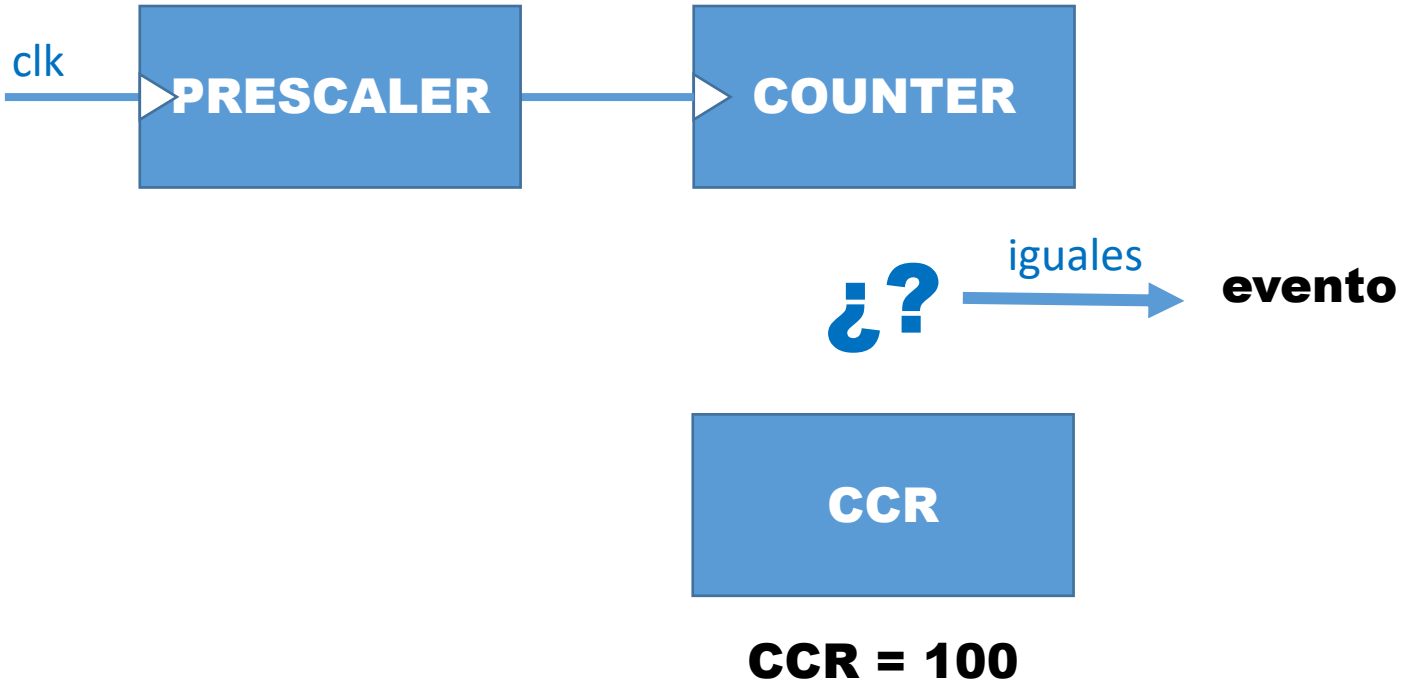
16 bits
Tope de cuenta = 65.536

PSC = 10000

10MHz → Periodo = 100ns*PSC = 1ms → Tope de tiempo = 65,536s

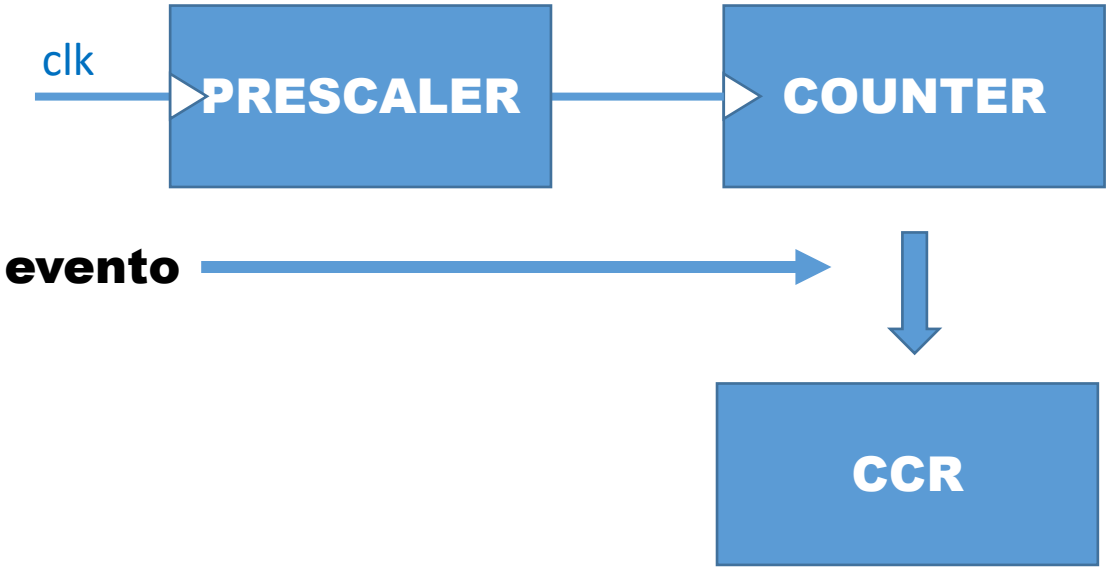
Funcionalidades

- 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

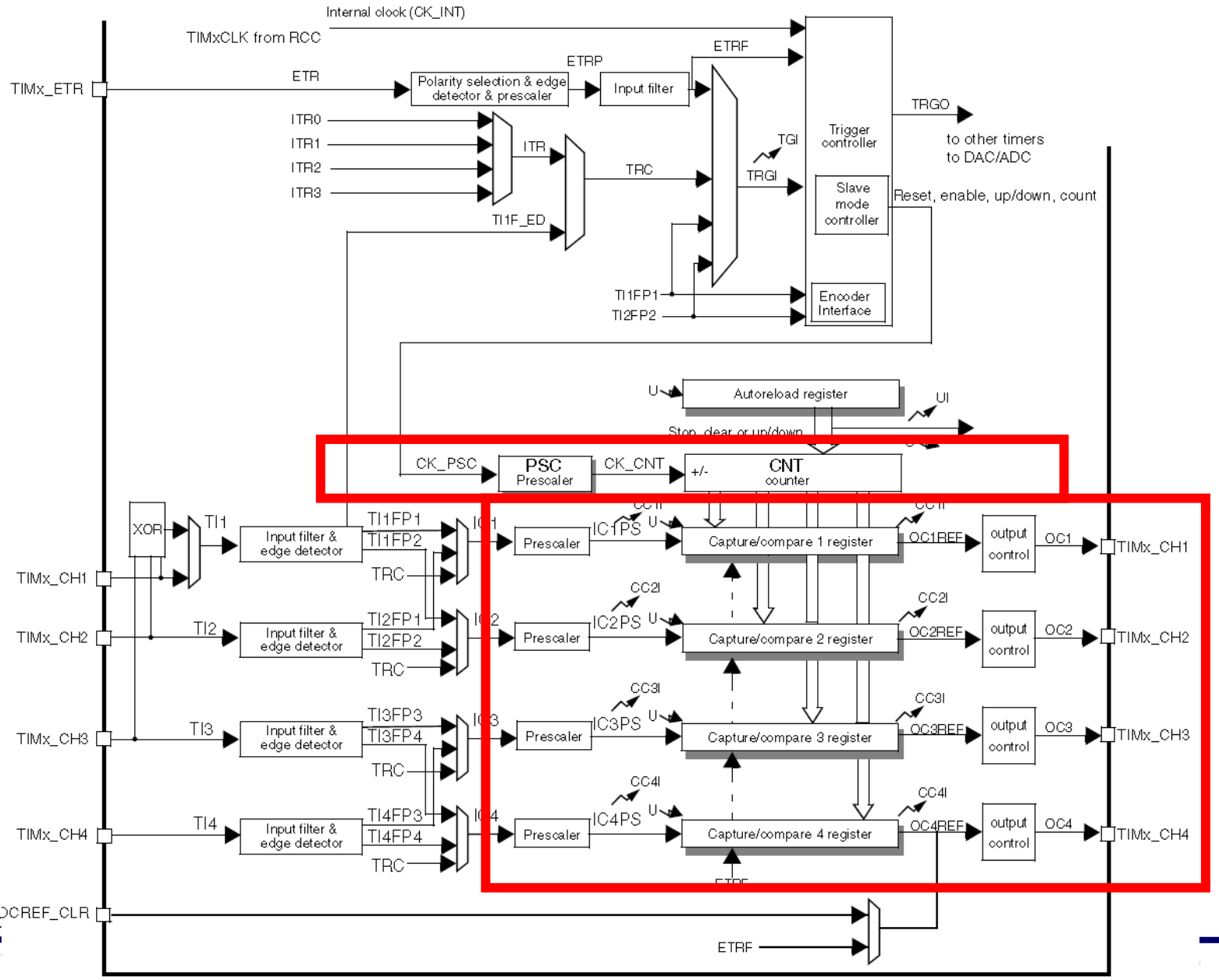


Arquitectura

Arquitectura

- 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

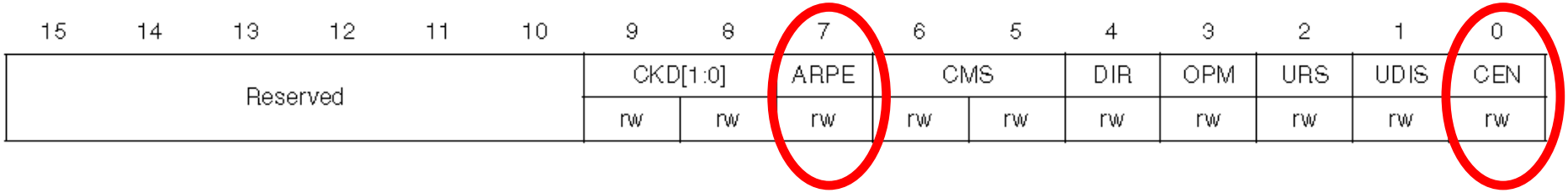


Arquitectura

- 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

- TIMx→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 TIMx_ARR
      - 1 – Se utilizará el TIMx_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

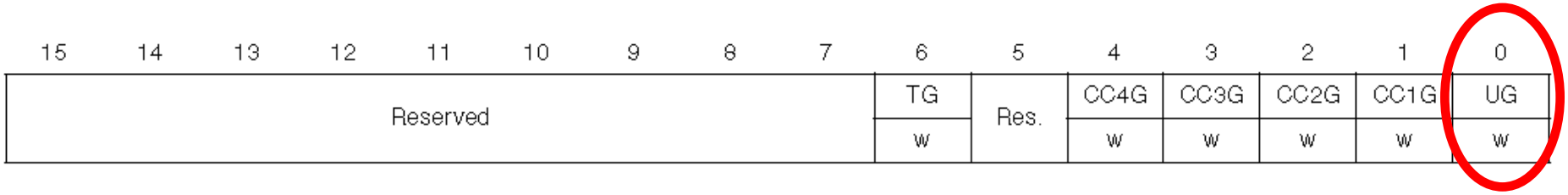
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TI1S	MMS[2:0]			CCDS	Reserved		
								rw	rw	rw	rw	rw			

- 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

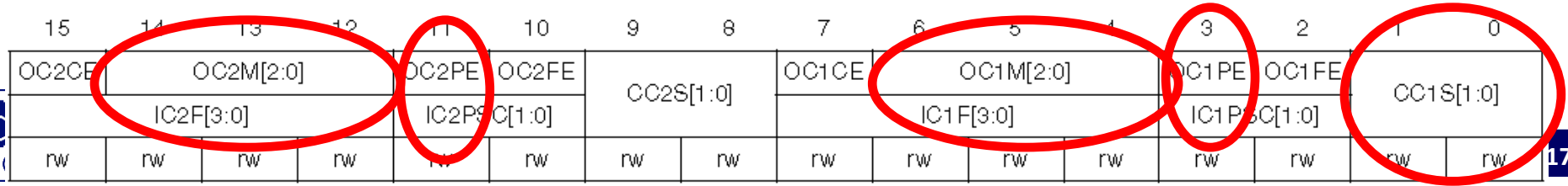
TIMx: Registros de Control

- TIMx→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 – Toogle
      - 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)



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

TIMx: Registros de Datos

- TIMx→CNT – Counter
- TIMx→PSC – Prescaler
  - El valor que se introduzca aquí hace que la frecuencia del reloj del CNT sea:
    - PSC = 0 -> fck/1 (sin preescalado)
    - PCS = 1 -> fck/2 (divido por 2)
    - PSC = 2 -> fck/3 (divido por 3)

      ……
- TIMx→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.
- TIMx→CCRy – 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

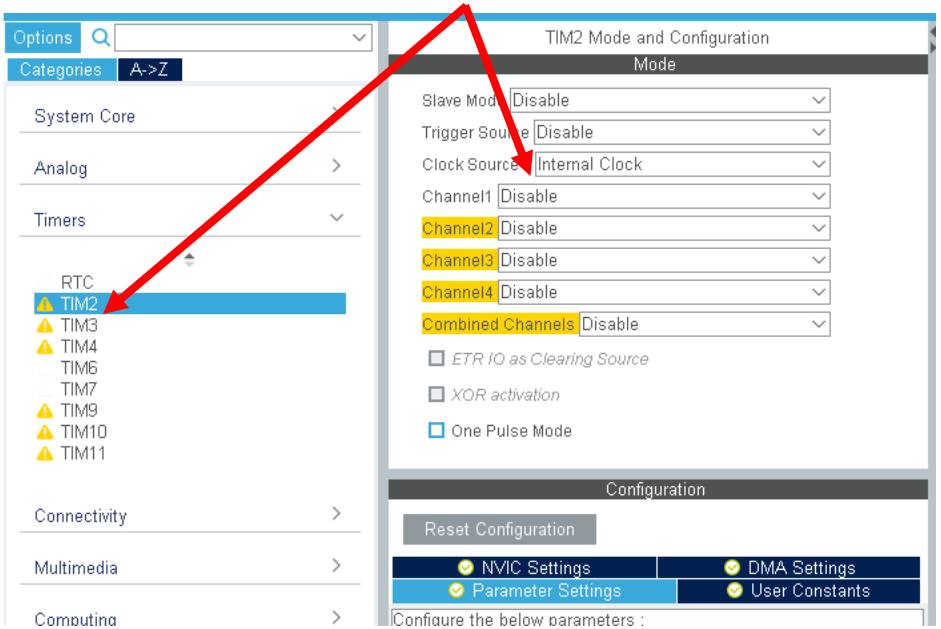
- TIMx→SR – Status Register:
  - Registro de 16 bits, que contiene los 10 flags de eventos:
    - CCyOF – 4 flags que indican con un '1' si ha ocurrido un error de overrun (un flag para cada uno de los canales)
      - Para limpiar el flag hay que escribir un '0'
    - TIF y UIF – (no se van a utilizar)
    - 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.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

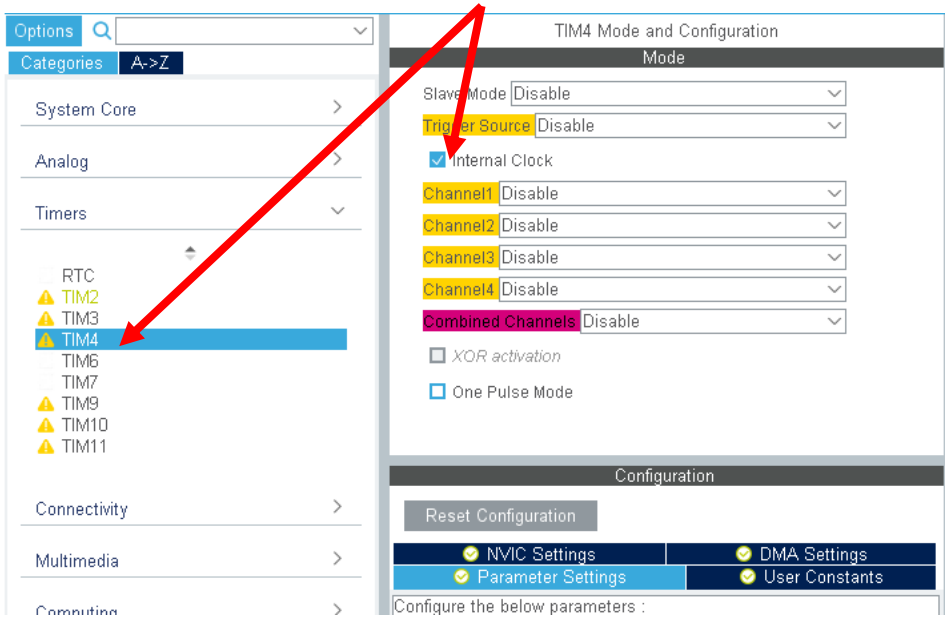
Peculiaridades de la Discovery y la perspectiva CubeMX

- Por defecto todos los timers están desconectados del reloj interno para ahorrar energía
- Si necesitas el Timer, debes activar en CubeMX la fuente de reloj como Internal Clock

TIM2, TIM3



TIM4



Funcionalidades "Match" (TOC)

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 TIMx→CCMRy
- Cuando hay una comparación exitosa:
  - No activa ningún pin de salida o bien se pone el pin de salida al valor (OCyM en TIMx→CCMRy). 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 TIMx→SR)
  - Y genera una IRQ (si el bit CCyIE en TIMx→DIER está a 1)
- Los TIMx→CCRy se pueden programar usando o no preload (bit OCxPE en TIMx→CCMRz, 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

1. Selección del reloj interno
   - TIMx→CR1, TIMx→CR2, TIMx→SMCR
2. Configuración del funcionamiento del contador
   - TIMx→PSC, TIMx→CNT, TIMx→ARR, TIMx→CCRy
3. CCyIE = 1 (en TIMx→DIER) si se quiere utilizar IRQ
4. Selección del modo de salida
   - TIMx→CCMR1, TIMx→CCMR2, TIMx→CCER
   - Por ejemplo:
     - OCyM=011 (toogle 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 GPIOx→MODER y GPIOx→AFR
5. Habilitación del contador (CEN=1 en TIMx→CR1)
6. Para actualizar el valor de comparación en TIMx→CCRy:
   - Escribirlo directamente con OCyPE=0 (sin preload)
     - Si no, entonces sólo se actualizaría al provocar un “update event”

Ejemplo de Uso por Espera Activa

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

```
/* USER CODE BEGIN 2 */
short numero=0;
unsigned char cadena[6];

BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();
// 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 -> F_contador=32000000/32000 = 1000 pasos/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.
                     // 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

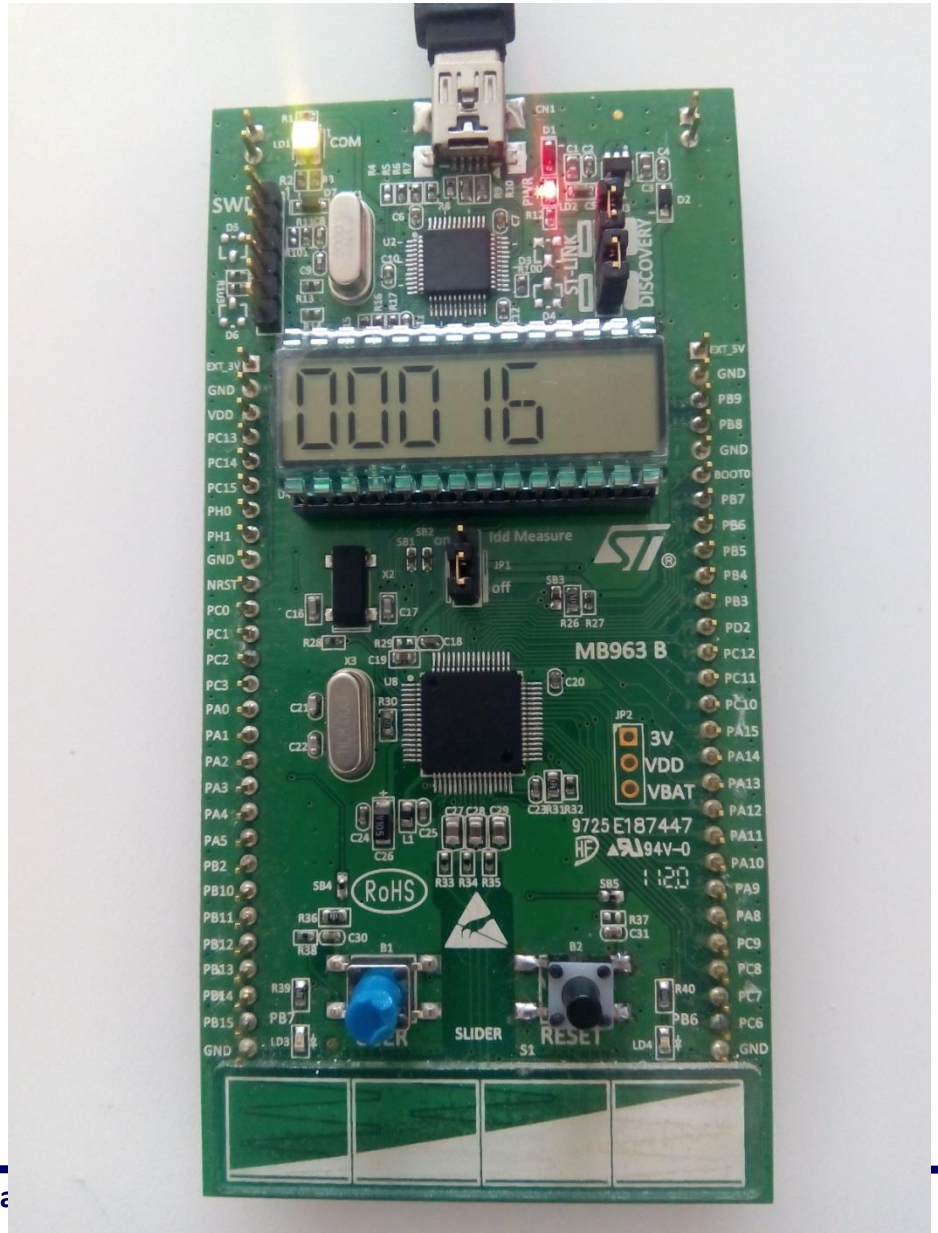
- Inicialización (cont.):

```
// Modo de salida del contador
TIM4->CCMR1 = 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

/* USER CODE END 2 */
```

- Funcionalidad Continua:

```
/* USER CODE BEGIN WHILE */
while (1) {
   while ((TIM4->SR&0x0002)==0);  // Si no se produce un evento en el canal 1 del TIM4 (TOC),
                                  // es decir, 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
   BSP_LCD_GLASS_Clear();
   BSP_LCD_GLASS_DisplayString((uint8_t *)cadena); // Saco el valor en el LCD

   /* USER CODE END WHILE */
   }
```

Prueba del Ejemplo



Ejemplo de Uso por IRQs

- El mismo ejemplo, pero esta vez con IRQs
  - Variables Globales y RAI:

```
/* USER CODE BEGIN PV */
unsigned short tiempo = 1000;
unsigned short numero=0;
/* USER CODE END PV */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 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->CCR1 += tiempo;       // Actualizo el valor de la comparación incrementándolo en
                                  // 1000 pasos = 1 segundo
      TIM4->SR = 0x0000;          // Limpio los flags del contador
      }
  }

/* USER CODE END 0 */
```

Ejemplo de Uso por IRQs

- Inicialización (cont.):

```
/* USER CODE BEGIN 2 */
unsigned char numero_ant = 0;
unsigned char cadena[6];

BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();

// 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

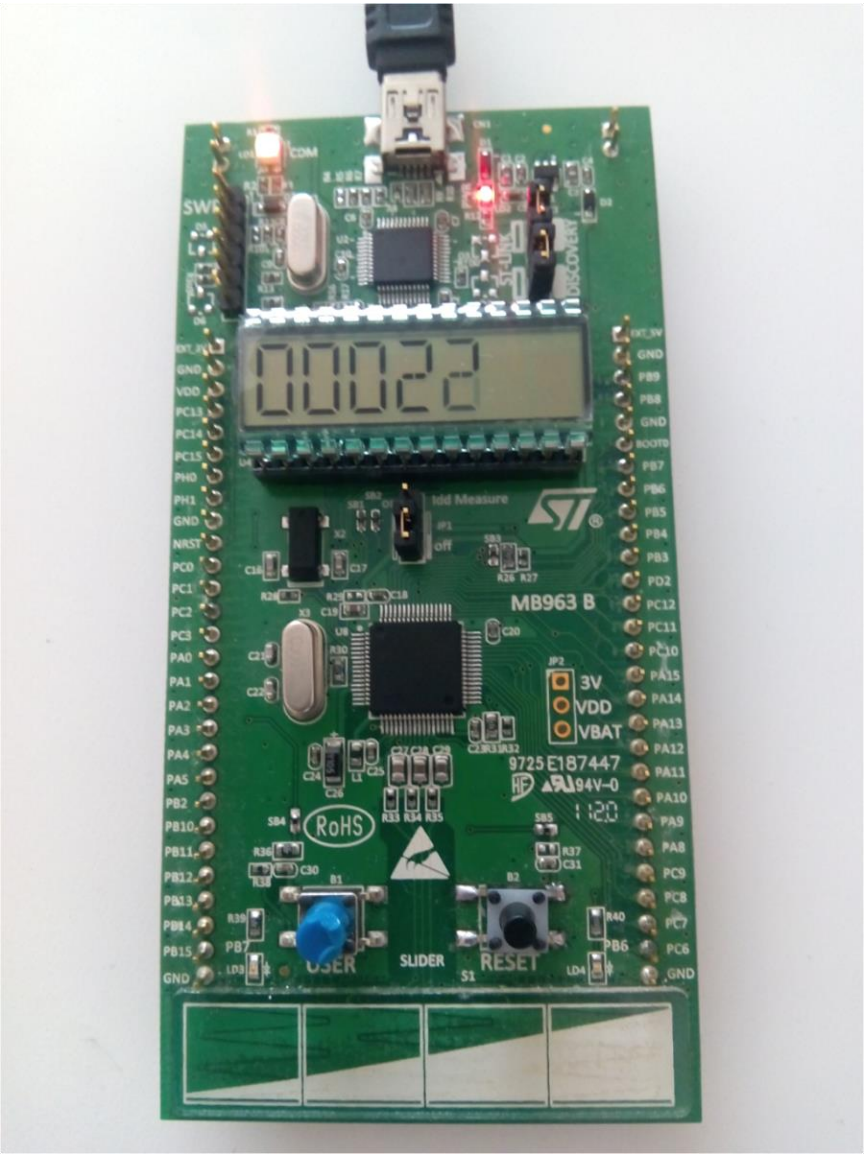
- Inicialización (cont.):

```
// Modo de salida del contador
TIM4->CCMR1 = 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);

/* USER CODE END 2 */
```

- Funcionalidad continua:

```
/* USER CODE BEGIN WHILE */
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
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)cadena); // Saco el número por el LCD
      }
   /* USER CODE END WHILE */
   }
```

Prueba del Ejemplo



Modulador por Anchura de Pulso (PWM)

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

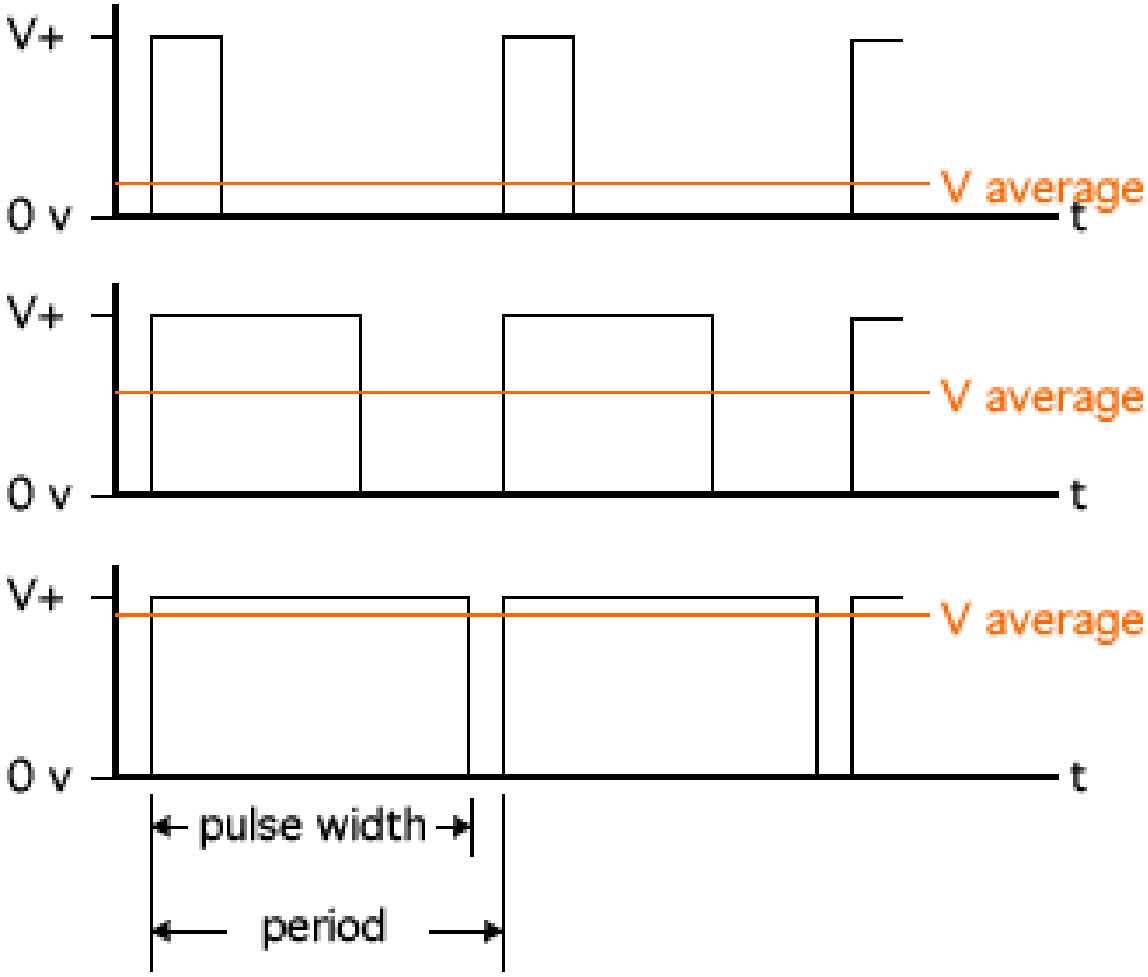


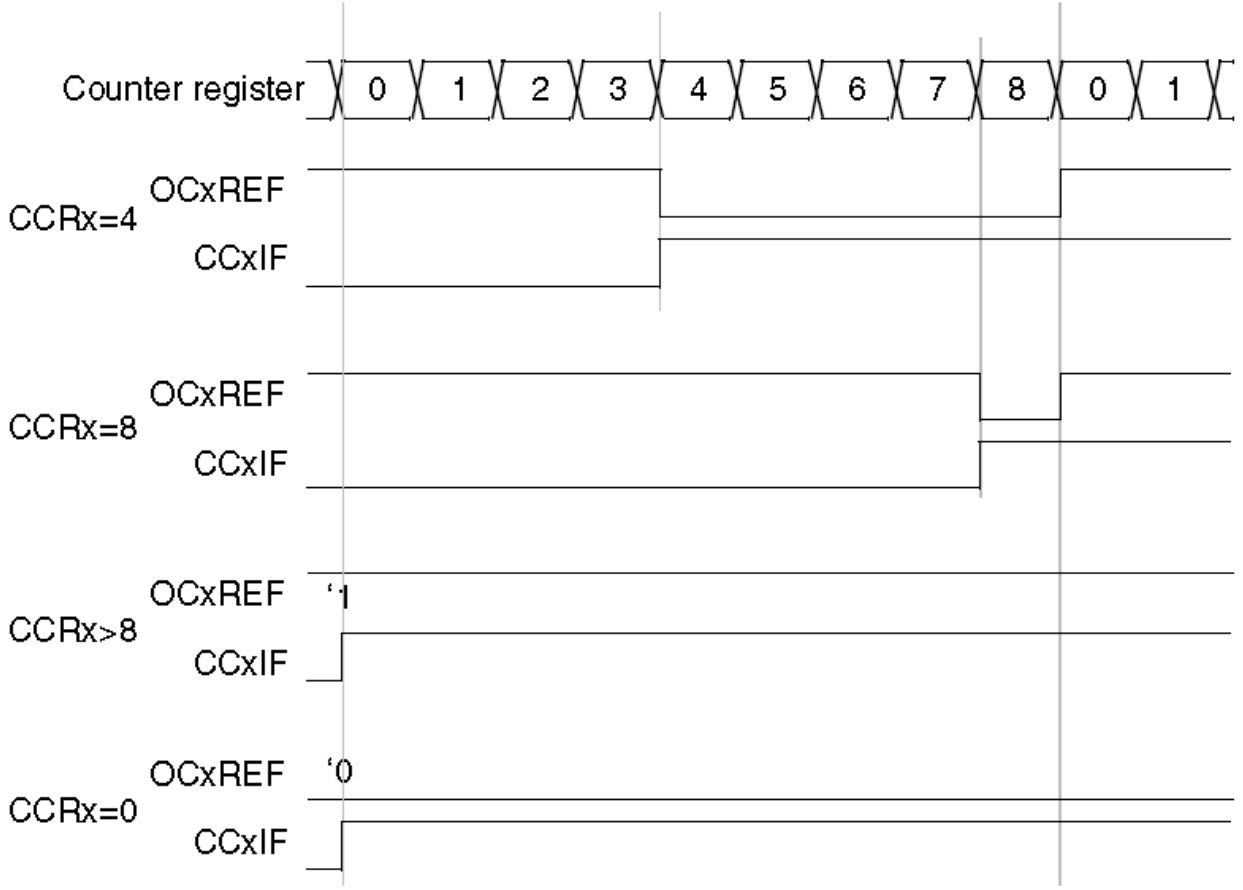
Imagen tomada de:
http://meteoyelectronica.blogspot.com/2010/12/pwm-una-manera-sencilla-de-controlar-un.html

Procedimiento de uso del PWM

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

Ejemplo de Uso del PWM (gráfico)

- Ejemplo de uso con TIMx→ARR = 8 y distintos TIMx→CCRy
  - Se muestra tanto la salida OCyREF, como el flag CCyIF



Ejemplo de Uso de PWM

- 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
  - Inicialización:

```
/* USER CODE BEGIN 2 */
short DC=1;
unsigned char cadena[6];
BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();
// 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] para decir que el PB7 tiene la AF2 (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));
// Selección del reloj interno: CR1, CR2, SMRC
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 -> f_contador=32000000/32000 = 1000 pasos/segundo
TIM4->CNT = 0;          // Inicializo el valor del contador a cero
TIM4->ARR = 9;          // Pongo una frecuencia PWM de 100 Hz y sólo cuento 10 pasos
TIM4->CCR2 = DC;        // El Duty cycle se pone a 1 inicialmente
```

Ejemplo de Uso de PWM

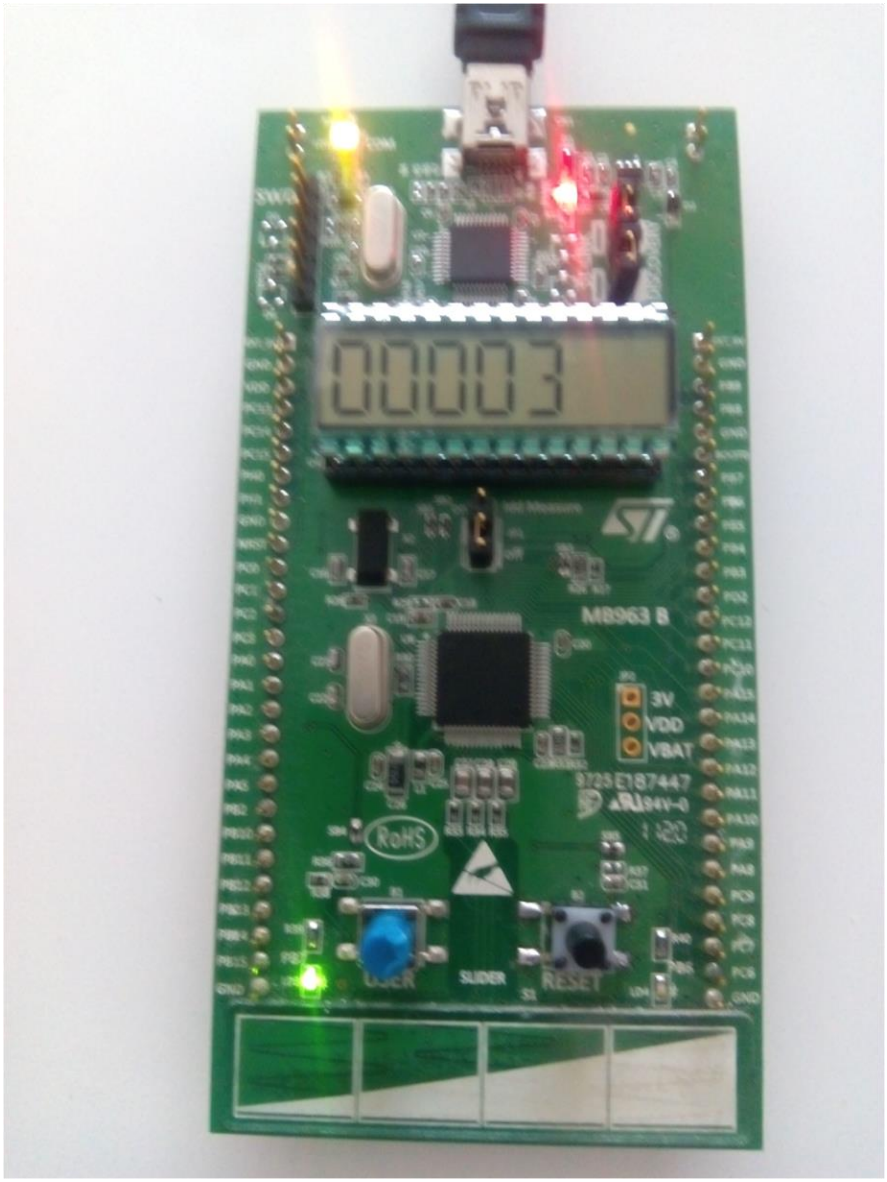
- Inicialización (cont.):

```
 // 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
 /* USER CODE END 2 */
```

- Funcionalidad continua:

```
 /* USER CODE BEGIN WHILE */
 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
       BSP_LCD_GLASS_Clear();
       BSP_LCD_GLASS_DisplayString((uint8_t *)cadena); // Saca el DC por el LCD
       }
    /* USER CODE END WHILE */
    }
```

Ejemplo de Uso de PWM



Funcionalidades “Capture” (TIC)

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→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

1. Configurar el pin para funcionalidad TIMx
   - GPIOx→MODER, GPIOx→AFR
2. Configurar el reloj
   - TIMx→CR1, TIMx→CR2, TIMx→SMCR
3. Configurar el ritmo de cuenta del contador
   - TIMx→PSC, TIMx→CNT, TIMx→ARR
4. Configurar la entrada activa (CCyS=01 en TIMx→CCMRx)
5. Seleccionar el flanco activo para el evento (CCyP y CCyNP en TIMx→CCER)
6. Habilitar la captura (CCyE=1 en TIMx→CCER)
7. Si se quieren utilizar interrupciones, activar el CCxIE de TIMx→DIER
8. Limpiar el flag (TIMx→SR)
9. Iniciar el contador (CEN=1 en TIMx→CR1)
10. Al producirse el evento de captura:
    - Leer el contenido de TIMx→CCRy
    - Limpiar el flag en TIMx→SR

Ejemplo de Uso por Espera Activa

- Partiendo de un PCLK1 de 32MHz, se cuenta el tiempo que se tarda entre pulsaciones del botón USER, mostrándolo en ms en el LCD.
  - Inicialización:

```
/* USER CODE BEGIN 2 */
unsigned char cadena[6];
unsigned short tiempo_inicio = 0;
int tiempo;

BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();
// 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] para decir que el PA0 tiene AF1 (TIM2)
GPIOA->AFR[0]&=~(0x0E << (0*4));
// Selección del reloj interno: CR1, CR2, SMRC
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 -> F_contador=32000000/32000 = 1000 pasos/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

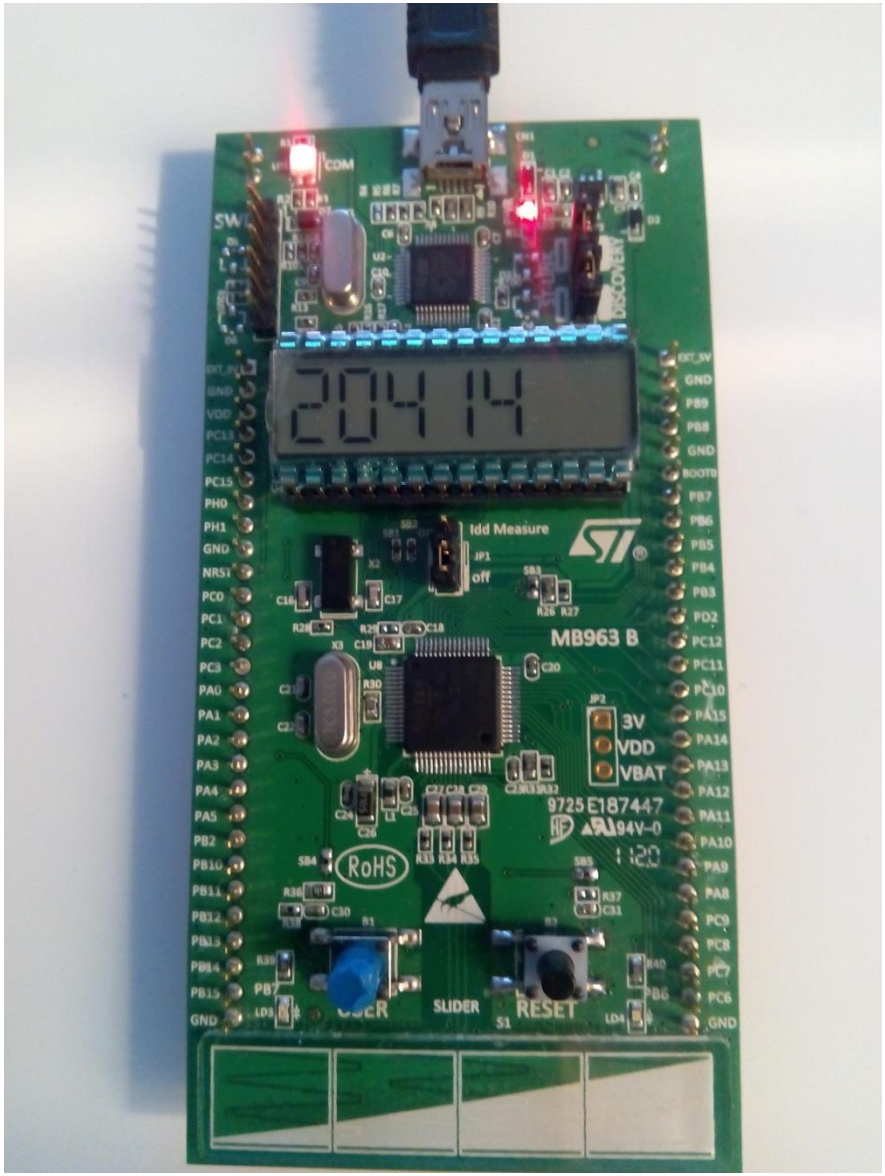
- Inicialización (cont.):

```
// 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
/* USER CODE END 2*/
```

- Funcionalidad continua:

```
/* USER CODE BEGIN WHILE */
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 de 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
                                         // yo la vuelta y me aseguro que 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
   BSP_LCD_GLASS_Clear();
   BSP_LCD_GLASS_DisplayString((uint8_t *)cadena); // Saco el texto por el LCD CD
   /* USER CODE END WHILE */
   }
```

Ejemplo de Uso por Espera Activa



Ejemplo de Uso por IRQs

- Mismo ejemplo pero usando las IRQs de TIM2
  - Variables Globales y RAI:

```
/* USER CODE BEGIN PV */
unsigned char pulsacion=0;
unsigned short tiempo_inicio = 0;
int tiempo;
/* USER CODE END PV */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void TIM2_IRQHandler(void) {
   if ((TIM2->SR & 0x0002)!=0) {            // Si se cumple el evento del TIC
      pulsacion=1;                          // Actualizo para que lo sepa el programa principal
      tiempo = TIM2->CCR1 - tiempo_inicio;  // El tiempo transcurrido es la resta del tiempo
      if (tiempo<0) tiempo += 0x0FFFF;      // Aseguro la diferencia correcta
      tiempo_inicio = TIM2->CCR1;           // Actualizo el nuevo tiempo inicio para la próxima
      TIM2->SR = 0x0000;                    // Limpio los flags del contador
      }
   }

/* USER CODE END 0 */
```

Ejemplo de Uso por IRQs

- Inicialización:

```
/* USER CODE BEGIN 2 */
unsigned char cadena[6];
BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();
// 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 para decir que el P0A es AF1 (TIM2)
GPIOA->AFR[0]&=~(0x0E << (0*4));
// Selección del reloj interno: CR1, CR2, SMRC
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
// Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
TIM2->PSC = 32000;      // Preescalado=32000 -> F_contador=32000000/32000 = 1000 pasos/seg
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 y 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)
/* USER CODE END 2 */
```

Ejemplo de Uso por IRQs

- Funcionalidad continua:

```
/* USER CODE BEGIN WHILE */
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
   BSP_LCD_GLASS_Clear(); // Saco el tiempo por el LCD
   /* USER CODE END WHILE */
   }
```

Ejemplo de Uso por IRQs