UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Fundamentos de los timers de los microcontroladores STM32F4


| | |
|---|---|
| **Apellidos, nombre** | Yuste Pérez, Pedro<br>(pyuste@disca.upv.es) |
| **Departamento** | Departamento de Informática de Sistemas y Computadores |
| **Centro** | Universidad Politécnica de Valencia |

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# 1 Resumen de las ideas clave

En este artículo se introducen los fundamentos de funcionamiento de los timers de la familia STM32F4 de microcontroladores ARM Cortex. Se enumeran los diferentes componentes y modos de funcionamiento y se explica, paso a paso, cómo utilizar las librerías CMSIS para configurarlos. Se suponen conocimientos básicos del sistema de relojes de esta familia de microcontroladores y de la utilización de las librerías CMSIS.

# 2 Introducción

Una característica fundamental de los sistemas empotrados basados en microcontrolador es la capacidad para trabajar con tiempos con una gran precisión. El periférico que realiza estas funciones son los timers. Cada fabricante tiene un diseño propio para los timers, aunque los fundamentos son los mismos. En este artículo se introduce el funcionamiento de los timers de la familia STM32F4 y la configuración mediante librerías CMSIS. Se asume que el lector tiene un conocimiento básico de esta familia de microcontroladores, del sistema de relojes y de las librerías CMSIS.

# 3 Objetivos

Una vez que el alumno se lea con detenimiento este documento, será capaz de:

- Comprender el funcionamiento de los timers de los microncontroladores STM32F4
- Calcular los parámetros necesarios para generar y medir tiempos con la precisión deseada.
- Escribir programas que utilizan los timers para generar o medir tiempos.

# 4 Desarrollo

La familia de microcontroladores STM32F4 tiene una serie de timers de propósito general (TIM2 a TIM5) que tienen las siguientes características:

- Contador de autorecarga de 16 o 32 bits, con cuenta creciente o decreciente.
- Prescaler programable de 16 bits para dividir la frecuencia de reloj por un valor entre 1 y 65535.
- 4 canales independientes que se pueden utilizar para:
  - Captura de señal (de entrada)
  - Comparación de cuenta (para salida)
  - Generación automática de señales PWM
  - Salida en modo “un pulso”
- Generación automática de señales de interrupción en diferentes eventos.

Podemos dividir el timer en dos grandes bloques, la circuitería de base de tiempos y los canales de captura/comparación. Vamos a centrarnos en el primero.

## 4.1 Base de tiempos

El circuito de base de tiempos está formado por tres bloques, cada uno de ellos con un registro asociado:

- El contador. Es un contador ascendente o descendente de 16 o 32 bits. Es el corazón del timer. Tiene asociado el Counter Register (TIMx_CNT), en el que se puede leer y escribir. El contador se incrementa o decrementa en una unidad cuando lega un flanco del siguiente bloque.
- El prescaler. Es un divisor de frecuencia programable. En realidad es un timer dentro del timer. Tiene un registro asociado, el Prescaler Register (TIMx_PSC). La frecuencia del reloj a la salida del prescaler sigue la fórmula de la ecuación 1, por lo que el valor a escribir en el prescaler vendrá dado por la ecuación 2.

$$CK_{CNT} = \frac{CK_{PSC}}{TIMxPSC + 1}$$

*Ecuación 1. Frecuencia de salida del prescaler.*

$$TIMxPSC = {CK_{CNT}}/{CK_{PSC}} - 1$$

*Ecuación 2. Cálculo del valor del prescaler.*

Donde:

  - CK_PSC es la frecuencia de entrada al prescaler (Clock Prescaler)
  - CK_CNT es la frecuencia de salida del prescaler (Clock Counter)
  - TIMx_PSC es el valor contenido en el Prescaler Register

- El registro de autorecarga, Auto-Reload Register (TIMx_ARR). Es un registro que almacena el periodo que va a contar el timer. El funcionamiento es diferente si la cuenta es ascendente o descendente.
  - Contador ascendente. El valor del contador crece hasta que alcanza al valor contenido en el TIMx_ARR. En la siguiente cuenta el contador se pone a cero y vuelve a empezar.
  - Contador descendente. El valor del contador disminuye hasta llegar a 0. En la siguiente cuenta se escribe el contenido de TIMx_ARR en el contador y se vuelve a empezar.

En la siguiente figura se pueden ver los tres bloques de la base de tiempos. Está extraída de [1].

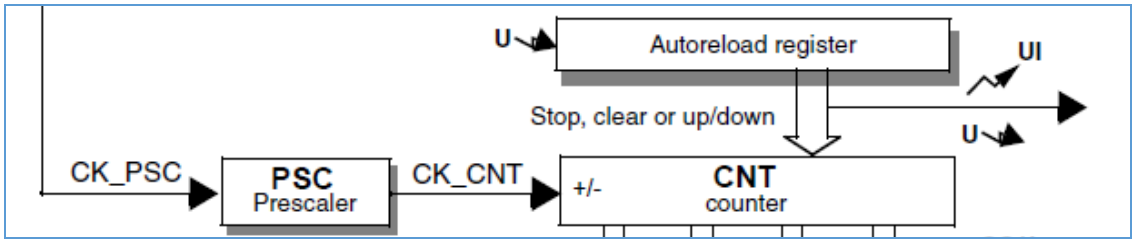


*Figura 1. Diagrama de bloques del circuito de base de tiempos*

## 4.2 ¿Y esto para qué sirve?

Pues para medir tiempos. Lo importante de un timer es la capacidad para medir un tiempo con tanta precisión como queramos (dentro de los límites del timer). Supongamos, por ejemplo, que queremos generar un pulso de 10ms y queremos que la precisión sea de 1ms. Con esta precisión el contador debe hacer 10 cuentas de 1ms cada una, por lo que en TIM_x_ARR deberíamos escribir el número máximo de cuentas menos uno (9 = 10-1) y en TIM_x_PSC deberíamos escribir el divisor necesario para conseguir que el contador cuente con una frecuencia de 1KHz (1/1ms).

A partir de aquí se plantean las siguientes preguntas al lector:

¿Cuál sería el tiempo máximo y mínimo que podría medir un timer como este modificando sólo TIM_x_ARR?

¿Cómo tendríamos que modificar estos dos parámetros si queremos que la precisión sea de 1us?

¿Cuál sería la precisión máxima que podemos alcanzar si fijamos el tiempo de cuenta en 10ms?

## 4.3 Interrupciones

Para sincronizar los eventos del timer con la ejecución del programa principal podemos utilizar interrupciones. El timer tiene varias fuentes posibles de interrupción. Dentro de la base de tiempos se puede generar una interrupción cuando se produzca un evento de actualización. Esto quiere decir, si está funcionado en modo creciente, cuando el contador alcance su cuenta máxima. En el instante en que la cuenta pasa del valor máximo a cero se genera un evento de actualización. Si se ha habilitado la interrupción por actualización, este evento producirá una interrupción y se comenzará a ejecutar el código de la función de tratamiento de esta interrupción.

Hay que recordar que en el STM32F4, para habilitar una interrupción es necesario habilitarla en el periférico y habilitar el canal correspondiente en el controlador de interrupciones (NVIC). Por último, en la función de tratamiento de la interrupción será necesario borrar el flag correspondiente al evento de actualización.

## 4.4 Implementación, librerías.

Para implementar un programa que utilice los timers podemos utilizar las librerías de periféricos que nos proporciona el fabricante del microcontrolador. En concreto la librería stm32f4xx_tim incluye funciones para tratamiento de timers. Los pasos para utilizar la base de tiempos son los siguientes:

- Habilitar el reloj del periférico con la función RCC_APBxPeriphClockCmd(RCC_APBxPeriph_TIMx, ENABLE)
- Rellenar una estructura del tipo TIM_TimeBaseInitStruct con los parámetros deseados.

- Llamar a la función TIM_TimeBaseInit(TIMx, &TIM_TimeBaseInitStruct) para inicializar la base de tiempos con los parámetros configurados en el paso anterior.
- Habiltar el NVIC si se van a generar interrupciones.
- Habilitar la interrupción de actualización con la función TIM_ITConfig(TIMx, TIM_IT_Update)
- Llamar a la función TIM_Cmd(ENABLE) para poner en marcha el timer.
- Por último, si se han configurado las interrupciones, cuando el timer alcance el valor almacenado en el registro TIM_x_ARR se ejecutará automáticamente la función asociada al la interrupción, donde habrá que borrar el flag que ha provocado la interrupción.

Veamos todo el proceso con un ejemplo:

```
void ConfigTimer4(void) {
/* Definimos variables*/
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
uint32_t PrescalerValue;

/* Habilitamos reloj Timer4 */
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

 /*Suponemos TIM4 input clock (TIM4CLK) es igual a  2 * APB1 clock (PCLK1),
Como el prescaler APB1 es diferente de 1 (APB1 Prescaler = 4 tenemos que
 TIM4CLK = 2 * PCLK1  y   PCLK1 = HCLK / 4
    => TIM4CLK = 2*(HCLK / 4) = HCLK/2 = SystemCoreClock/2
          Para conseguir TIM4 a 100 KHz, el prescaler se calcula como:
    Prescaler = (TIM4CLK / TIM1 counter clock) - 1
    Prescaler = (168 MHz/(2 * 100 KHz)) - 1 = 839
 La variable SystemCoreClock está definida en CMSIS y mantiene la frecuencia de HCLK.*/

 /* Cálculo del prescaler */
 PrescalerValue = (uint16_t) ((SystemCoreClock /2) / 100000) - 1;

 /* Configuración de la base de tiempos */
 TIM_TimeBaseStructure.TIM_Period = 1999;
 TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
 TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
```

```
/*Inicializamos la base de tiempos*/
 TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

/*Habilitamos Tim4 en NVIC  */
 NVIC_EnableIRQ(TIM4_IRQn);

/*Habilitamos interrupción de actualización en tim4*/
 TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);

 /* Ponemos en marcha TIM4 */
 TIM_Cmd(TIM4, ENABLE);
 }

void TIM4_IRQHandler()
{
//Todo lo que escribamos aquí se ejecutará cuando salte la interrupción del timer 4
//Y tedremos que asegurarnos de borrar el flag:
TIM_ClearITPendingBit(TIM4, TIM_IT_Update)
}
```

Se recomienda al lector la búsqueda de la definición del tipo TIM_TimeBaseInitTypeDef en el archivo de librería stm32f4xx_tim.h y de las funciones TIM_TimeBaseInit(), TIM_Cmd (), TIM_ITConfig () y TIM_ClearITPendingBit() en el archivo de librería stm32f4xx_tim.c

# 5 Cierre

A lo largo de este objeto de aprendizaje hemos tratado los fundamentos de los timers de propósito general de la familia STM32F4 y se ha visto el código necesario para configurarlos para generar una base de tiempos cualquiera. Para comprobar que se ha entendido se recomienda hacer los siguientes ejercicios:

- Revisar la configuración del ejemplo y determinar cada cuánto tiempo saltará la interrupción del timer 4.
- Modificar el programa para conseguir que esta interrupción salte una vez por segundo.
- Modificar el programa para conseguir que esta interrupción salte una vez por microsegundo.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# 6 Bibliografía

[1] STMicroelectronics: "RM0090 Reference manual. STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced ARM-based 32-bit MCUs", Doc ID 018909 Rev 4. 2013. URL: http://www/st.com