Universidad Europea

LAUREATE INTERNATIONAL UNIVERSITIES

TEMPORIZADORES Y WATCHDOG

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

Índice

© Universidad Europea de Madrid. Todos los derechos reservados.

Presentación

En este recurso continuamos revisando funcionalidades específicas para microcontroladores. En concreto, contadores/temporizadores y sistemas de vigilancia ante cuelgues (*watchdog*).

Cuando un microprocesador funciona de manera aislada (empotrado en otro dispositivo), debe disponer de un sistema de reinicio automático si un programa se cuelga, dado que no hay nadie que lo reinicie manualmente. Pensemos por ejemplo en un satélite: un error de programa o la aparición de una situación imprevista en el código que provoque un cuelgue (o una falta de respuesta durante un tiempo largo) sería catastrófica. Un **sistema de reinicio automático** debe darse cuenta de que esto ha ocurrido y reiniciar el microprocesador para que vuelva a ser operativo, Este mecanismo se conoce como ***watchdog*** (perro guardián).

Otra tarea habitual para un microcontrolador es simplemente **contar el número de veces que ocurre un cierto evento**, para tomar alguna acción cuando esta cuenta llegue a un valor dado. Aunque esto puede hacerse por programa, para evitar cargar al microprocesador se disponen de circuitos secuenciales específicos que de manera autónoma cuentan pulsos en determinadas señales, y avisan al micro cuando el valor de cuenta es igual a uno fijado, se desborda, o se hace 0.

Además, si la señal que contamos es una **señal periódica** (produce flancos a una cierta frecuencia), contar estos pulsos equivale a contar tiempo (temporizar). Por lo tanto, estos circuitos específicos se denominan **temporizadores/contadores**, ya que realizan ambas funciones simplemente cambiando la señal a contar.

Los **objetivos** que se pretenden alcanzar con este recurso son los siguientes:

- Analizar el **uso de temporizadores** en un ensamblador para el ATmega328P.
- Comprender el papel de **mecanismos de seguridad** como el *watchdog*.
- Practicar el uso de **interrupciones temporizadas** para realizar tareas.

© Universidad Europea de Madrid. Todos los derechos reservados.

Sistema de reinicio automático (*watchdog*)

Un *watchdog* es un tipo de dispositivo de seguridad que en otros ámbitos se conoce como "**dispositivo de hombre muerto**", que trata de asegurar que un **sistema no se quede sin control**. Por ejemplo, los conductores de locomotoras de tren han de activar periódicamente un pulsador para asegurar a la máquina que están ahí y activos. Si el conductor se desmaya, deja de activar el pulsador, por lo que la máquina puede tomar medidas como detener el tren y realizar una llamada de emergencia.

El sistema *watchdog* emplea un **temporizador propio** (WDT, WatchDog Timer) que cuenta pulsos de un reloj (también separado del reloj principal) que oscila a 128 kHz. Si se activa esta funcionalidad, el programa principal debe enviar una señal de "*reset*" para reiniciar el temporizador del *watchdog* antes de que este haya llegado a contar un tiempo prefijado (ejecutando la instrucción específica **WDR**, WatchDog Reset):

Situación normal: el código ejecuta WDR a tiempo	Si la instrucción WDR se ejecuta antes de que el tiempo del *watchdog* se haya cumplido, el *watchdog* vuelve a **comenzar** a temporizar **desde 0** y no ocurre nada.
Situación anómala: el código no consigue ejecutar WDR a tiempo	Si por alguna razón el micro no ha sido capaz de ejecutar la instrucción WDR a tiempo (por ejemplo porque se ha quedado colgado), el plazo de tiempo del ***watchdog* expira**. Entonces este lanza una **interrupción** como señal **de aviso**, y además una señal que puede **reiniciar automáticamente el micro** (este comportamiento es configurable).

La instrucción específica **WDR** debe ser ejecutada dentro del **bucle principal** del programa, usualmente en cada iteración. No supone una carga apreciable puesto que únicamente toma un ciclo de reloj.

Obviamente, hay que configura el tiempo de expiración del *watchdog* para que sea mayor que el tiempo de bucle de nuestro programa. Puede ajustarse entre 16 ms y 8 segundos.

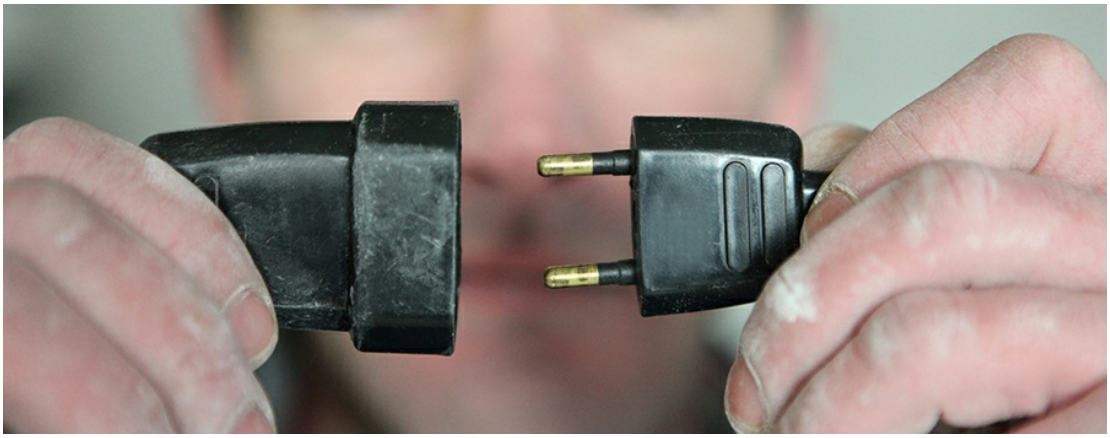
© Universidad Europea de Madrid. Todos los derechos reservados.

© Universidad Europea de Madrid. Todos los derechos reservados.

Modos de trabajo del *watchdog*

El dispositivo de *watchdog*, si se activa, dispone de **tres modos de trabajo** que podemos elegir:

Modo "solo interrupción"	Si el tiempo del *watchdog* llega a expirar, este simplemente **lanza una interrupción** (WDT, que provoca la ejecución de la instrucción presente en la posición de programa 0x00C). Esto permite diversos usos, por ejemplo: • Se puede usar como un **temporizador normal** (con tiempos largos), para ejecutar una cierta tarea periódicamente. • Permite "**despertar**" automáticamente al **procesador** cuando lo hemos puesto en alguno de los modos de reposo (*sleep*) para ahorrar energía. • También puede usarse para **limitar un tiempo de espera** para una cierta operación (por ejemplo, esperar una señal externa que por alguna razón no está llegando).
Modo System Reset	Si el tiempo del *watchdog* llega a expirar, este **reinicia automáticamente el procesador**. Se evita así un colgado indefinido en caso de un problema en el código.
Modo mixto	El *watchdog* lanza la interrupción y poco después reinicia el micro. Esto permite que, por ejemplo, el **código asociado** a la interrupción **guarde** algunos **valores críticos** en la EEPROM antes del reseteo.



© Universidad Europea de Madrid. Todos los derechos reservados.

Configuración del watchdog

El *watchdog* se gestiona utilizando el registro de configuración, **WDTCSR** (WatchDog Timer Control Register).

- Los bits para seleccionar el modo de operación son **WDIE** (WatchDog Interrupt Enable) y WDE (WatchDog reset Enable), según la siguiente tabla:

WDE	WDIE	Mode	Action on Time-out
0	0	Stopped	None
0	1	Interrupt Mode	Interrupt
1	0	System Reset Mode	Reset
1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode

Fuente: cortesía de Atmel

- Los bits para elegir el plazo de tiempo (*timeout*) son **WDP3:0** (WatchDog Preset) según la tabla:

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s

Fuente: cortesía de Atmel

© Universidad Europea de Madrid. Todos los derechos reservados.

Ejemplo

Ejemplo de código de configuración del watchdog

Ejemplo de código de configuración del *watchdog*

```
1.  .ORG 0x0000
2.    RJMP start
3.  .ORG 0x0072
4.
5.  start:
6.    ;configure watchdog to system reset mode (WDE=1, WDIE=0), timeout=1 second
    (WDP3:0 = 0110)
7.    LDI R16, (1<<WDE)|(1<< WDP2) |(1<< WDP1)
8.    OUT WDTCSR, R16
9.        ……
10.   loop:  ;main program loop
11.             ……
12.             ;do the tasks the program is supposed to do
13.     ……
14.     WDR ;avoids Watchdog time-out. The processor will be reset if this is not
    executed
15. within 1 second
16.   RJMP loop
```

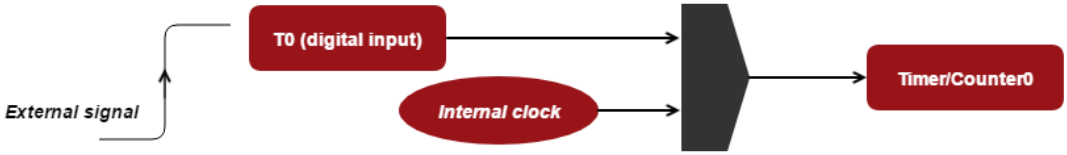
En el código anterior, si algo ocasiona que el bucle principal (loop:) se cuelgue, o quede **irresponsivo** por al menos 1 s, el micro se **reiniciará automáticamente** para recomenzar el programa.

Obviamente, si el reseteo se produce a menudo es que hay algún error en el programa. Esto solo debe ocurrir si el programa se cuelga por razones ajenas al mismo (típicamente una situación infrecuente que no somos capaces de prever).

© Universidad Europea de Madrid. Todos los derechos reservados.

Temporizadores/contadores (*timer/counter*)

Los microcontroladores proporcionan varios **circuitos** Temporizador/Contador (T/C) que se pueden usar para **múltiples tareas**. Se debe tener en cuenta que medir tiempo no es otra cosa que contar pulsos de un reloj, por lo que los circuitos temporizadores son simples contadores:



Si contamos pulsos de reloj a una frecuencia f_{CLK} (pulsos por segundo, Hz), cada pulso supone un periodo de tiempo $T_{CLK}=1/f_{CLK}$. La **relación** entre el valor de cuenta del circuito T/C (value) y el tiempo transcurrido es:

Algunos **ejemplos** de aplicaciones típicas serían los siguientes:

- Realizar una tarea que se repite de manera programada, o **sincronizar acciones**.
- **Esperar** un tiempo antes de hacer algo (o limitar un tiempo de espera de una respuesta).
- **Contar** un número de **eventos** (vueltas de una rueda, parpadeos de una luz, pulsaciones de un botón, etc.). Si contamos durante un tiempo dado, estamos calculando una frecuencia. Por ejemplo, contando el número de vueltas de una rueda durante un minuto, tenemos su velocidad de giro en rpm (revoluciones por minuto).
- **Generar formas de onda** (como las PWM).

El número de bits m de un T/C establece el máximo valor que puede contar (MAX = 2^m-1). El mínimo valor de cuenta es BOTTOM=0. Por ejemplo, un contador de 16 bits puede contar desde 0 hasta 0xFFFF=65535.

Los T/Cs pueden usarse de dos formas:

Activa (por *polling*)	En cualquier momento podemos leer el valor de cuenta del T/C y utilizar **comparaciones** para decidir qué hacer a continuación. También podemos resetear o cambiar el valor de cuenta.

© Universidad Europea de Madrid. Todos los derechos reservados.

Pasiva (por interrupciones)	Una vez configurado, el T/C opera de forma desatendida. Cuando el valor de cuenta llega a **MAX**, a **BOTTOM** o a un valor preestablecido, **lanza una interrupción**. En el código asociado a esa interrupción se realizan las tareas oportunas.

© Universidad Europea de Madrid. Todos los derechos reservados.

Modos de funcionamiento de un Temporizador/Contador (T/C)

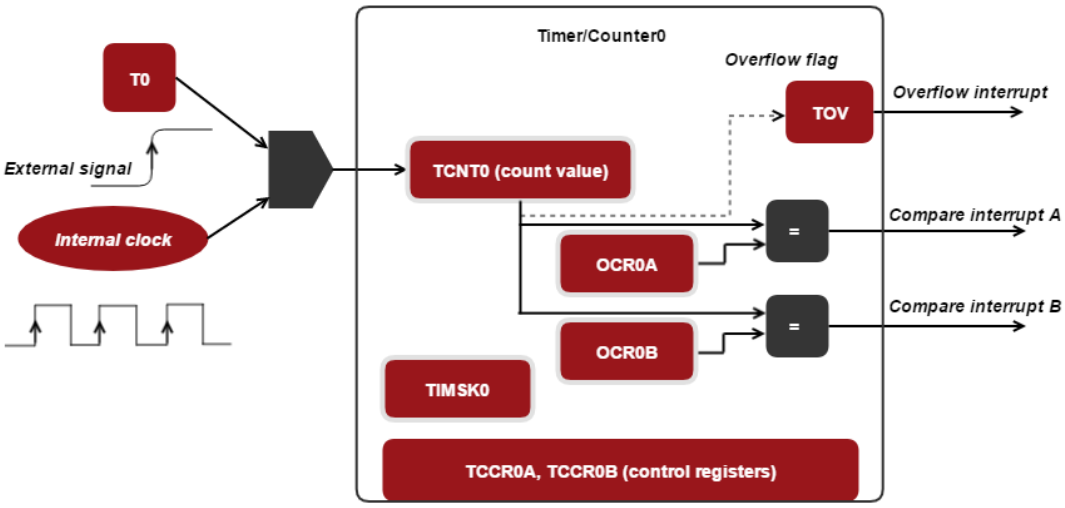
Un T/C puede permitir varias formas de funcionamiento. En el caso de micros Atmel *, los **modos disponibles** son:

- **Modo normal**: el T/C incrementa la cuenta en cada evento a contar. Cuando el valor de cuenta pasa de MAX (desbordamiento), este regresa a BOTTOM (0) y se pone a 1 un flag de desbordamiento (TOV, Timer OVerfow). Además se puede lanzar la interrupción de desbordamiento.

- **Modo CTC** (Clear Time on Compare match): en este modo la cuenta se incrementa normalmente y se compara con un valor de referencia (TOP) que se almacena en un cierto registro dado. Cuando los valores se igualan, se lanza una interrupción de comparación, y se devuelve el valor de cuenta a BOTTOM (0).

- **Modo PWM**.

La operación del temporizador/contador (por ejemplo, el Timer/Counter0) se realiza con los siguientes **registros**:



Valor de cuenta: TCNT0

Contiene el valor de cuenta. Puede ser leído (o escrito) por la CPU en cualquier momento.

© Universidad Europea de Madrid. Todos los derechos reservados.

Valor de comparación A: OCR0A

Guarda el **valor** de **TOP** para el modo de trabajo CTC. En modo normal se puede usar para guardar un valor de interés cualquiera, ya que lanza una interrupción cuando se igualan TCNT0=OCR0A. También se puede activar/desactivar un pin de salida del procesador (OC0A).

Valor de comparación B: OCR0A

Guarda un **segundo valor** de comparación que se compara con el valor de cuenta, lanzando su propia interrupción cuando se igualan (TCNT0=OCR0B). También se puede activar/desactivar un pin de salida del procesador (OC0B).

Gestión de interrupciones: TIMSK0

La habilitación/deshabilitación de las posibles interrupciones se realiza con el registro TIMSK0:

- El **bit TOIE0** habilita (1)/deshabilita (0) la interrupción de desbordamiento. Si se habilita, la interrupción pone a 0 automáticamente el flag de desbordamiento (vector 0x0020).
- El **bit OCIE0A** habilita/deshabilita la interrupción cuando el valor de cuenta iguala al PRESET en OCR0A (vector 0x001C).
- El **bit OCIE0B** habilita/deshabilita la interrupción cuando el valor de cuenta iguala el valor en OCR0B (vector 0x001E).

Recordemos, también que para usar cualquier interrupción hay que dejar habilitadas las interrupciones **globalmente** con la instrucción específica **SEI**, de lo contrario ninguna interrupción será lanzada en la máquina.

En la página 65 del manual del procesador (ver referencias bibliográficas) puede encontrarse el listado completo de **interrupciones** y sus correspondientes **vectores de memoria**.

* *Esta marca es una marca registrada que se cita para uso exclusivamente docente.*

© Universidad Europea de Madrid. Todos los derechos reservados.

Configuración del temporizador/contador0

El resto de la configuración se realiza en dos circuitos de configuración (**TCCR0A** y **TCCR0B**).

- El **modo de operación** se escoge con los bits WGM02 (en **TCCR0B**) y WGM01:0 (en **TCCR0A**) según la tabla:

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX

Fuente: cortesía Atmel

- El **origen de los eventos a contar** se escoge con los bits CS02:0 del registro **TCCR0B**. La combinación 000 **detiene** el T/C. Las combinaciones 001 a 101 **temporizan** (usando divisores del reloj principal $clk_{I/O}$=16 MHz). Las combinación 110 y 111 permite contar flancos (bajada o subida) de una **señal externa** conectada al pin T0:

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Fuente: cortesía Atmel

Por último, en la página 104 del manual del procesador (ver referencias bibliográficas) se indica cómo se configuran los **bits de salida digital OC0A y OC0B** para indicar al "mundo exterior" que la cuenta ha igualado los valores en OCR0A o OCR0B.

© Universidad Europea de Madrid. Todos los derechos reservados.

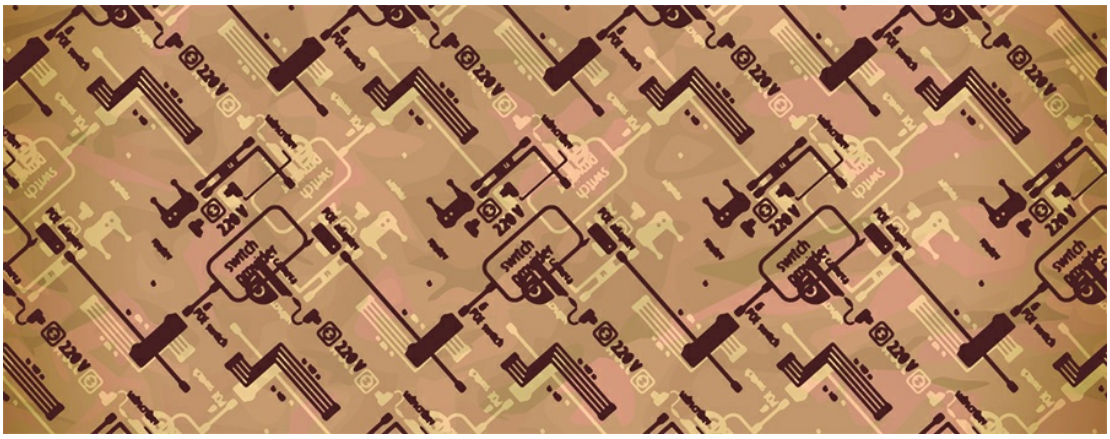
Ejemplo de configuración del Timer/Counter0

La siguiente rutina escribe los registros de configuración para hacer funcionar el Timer/Counter0 en modo CTC, usando el valor guardado en OCR0A para lanzar una interrupción y realizar una **tarea periódica** con ella cada 5 ms:

```
1.  config_TC0:  ;configure timer/counter0 as CTC with 5msec interrupt
2.    LDI R16, 78  ;set number of clock ticks to count TOP=78 (78 ticks x 15256
    ticks/second = 5msec)
3.    OUT OCR0A, R16
4.    LDI R16, (1<<WGM01)|(0<< WGM00) ;set operation mode = CTC
5.    OUT TCCR0A, R16
6.    LDI R16, (1<<OCIE0A) ;enable OCR0A match interrupt (vector 0x00E)
7.    STS TIMSK0, R16
8.    LDI R16, (1<<CS02)|(0<<CS01) |(1<<CS00) ;set input clock to
    clkI/O/1024=15265kHz, this also starts timing!
9.    OUT TCCR0A, R16
10.   SEI  ;enable interrupts globally
11.   RET
```

Tras ejecutar este código, cada 5 ms se lanzará la interrupción de comparación con el valor TOP. Su vector es el 0x00E, por lo que tenemos que escribir en esta dirección de memoria de programa una instrucción de salto a la rutina que realice la **tarea temporizada**:

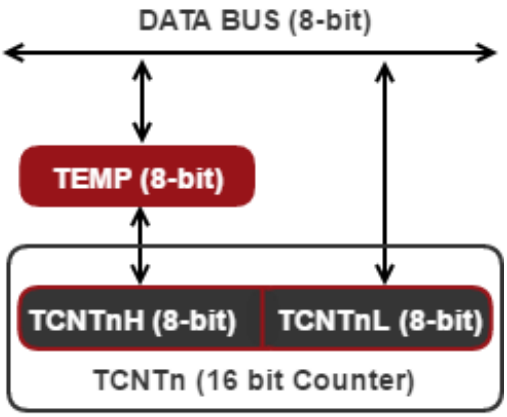
```
1.  .ORG 0x00E ;interrupt vector for timer/counter0 OCR0A match (count value=OCR0A)
2.    RJMP timer0_handler  ;jump to the interrupt handler routine to do the required task
```



© Universidad Europea de Madrid. Todos los derechos reservados.

Temporizador/contador de 16 bits Timer/Counter1

El Timer/Counter1 gestiona un valor de cuenta de **16 bits** (MAX=0xFFFF=65535). Esto implica que los registros para guardar el valor de cuenta (**TCNT1**) y los de comparación (**OCR1A** y **OCR1B**) necesitan ser de 16 bits. Como la máquina es de 8 bits, un registro de 16 bits es internamente una pareja de registros de 8 bit; por lo tanto, deben ser accedidos dos veces siguiendo un orden dado.



- **Escritura**: en primer lugar, se escribe el bite alto (H), que queda en un registro temporal. A continuación, se escribe el byte bajo (L), la electrónica provoca que ambos se combinen y se escriban al mismo tiempo en el registro de 16 bit. Por ejemplo, para establecer el valor de comparación en **OCR1A** a 0x01FA:

```
1. LDI R16, 0xFA
2. LDI R17, 0x01
3. OUT OCR1AH, R17  ;esto guarda el byte alto en un registro temporal (aún no se escribe en TCNT1)
4. OUT OCR1AL, R16  ;escribe a la vez el byte bajo proporcionado aquí y el alto del temporal en TCNT1 (16bit)
```

- **Lectura**: en primer lugar, se lee el byte bajo (L) del registro de 16 bit, la electrónica provoca que el byte alto se copie al mismo tiempo a un registro temporal. A continuación, se lee el byte alto (H), que automáticamente se toma del registro temporal. Por ejemplo, para leer el valor de cuenta de **TCNT1**:

```
1. IN R16, TCNT1L ;esto copia el byte bajo a R16 y guarda el byte alto en un registro temporal a la vez
2. IN R17, TCNT1H ;copia el byte alto (previamente guardado en el registro temporal) a R17
```

Si se emplean interrupciones, puede ocurrir que se lance una entre ambas instrucciones de lectura (o escritura) y que modifique el registro temporal para el valor alto, con lo que se obtendría una parte alta errónea. Para evitarlo, se deshabilitan interrupciones globalmente (instrucción **CLI**) antes de comenzar el par de instrucciones de lectura/escritura y se habilitan de nuevo (instrucción **STI**) al acabar.

La configuración es análoga a la del Timer/Counter0, solo que usando los registros de configuración **TCCR1A**, **TCCR1B**, **TCCR1C**, y el de control de interrupciones **TIMSK1**.

© Universidad Europea de Madrid. Todos los derechos reservados.

Notas adicionales sobre temporizadores/contadores

A continuación, vamos a describir alguna **característica** más de los temporizadores/contadores:

Timer/Counter2 de 8 bits: preescalado propio

Hay disponible un segundo T/C de 8 bits, el **Timer/Counter2**, que tiene su propio preescalado (por lo que puede usar una base de tiempos distinta a la de los Timer/Counter0 y Timer/Counter1.

Table 24-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Fuente: cortesía de Atmel

Timer/Counter1 de 16 bits: reducción del preescalado

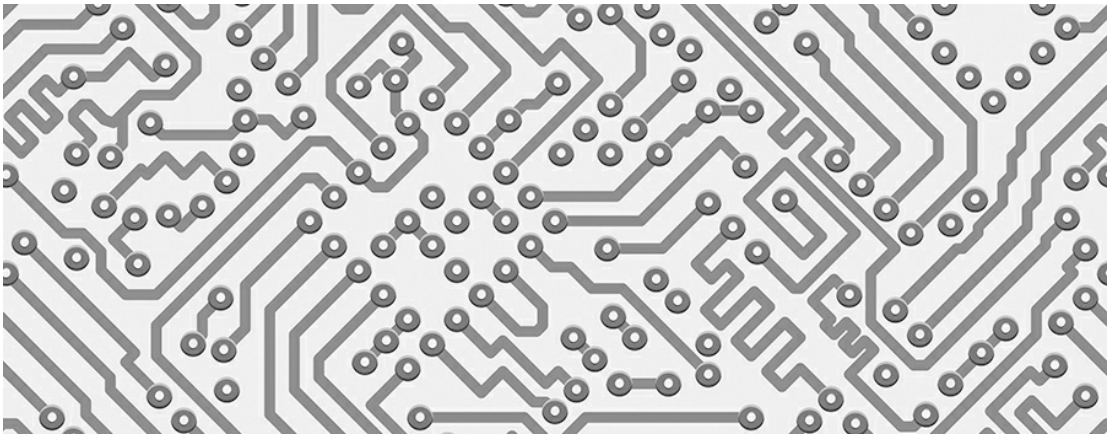
El Timer/Counter1 es de 16 bits, permitiendo por tanto una **temporización más larga** (hasta 65535 pulsos). También permite una **temporización más precisa**, reduciendo el preescalado del reloj principal $clk_{I/}$para aumentar la frecuencia de los pulsos). Por ejemplo, 5 milisegundos pueden obtenerse:

- Contando 78 pulsos (0x4E) a $clk_{I/O}/1024$ (15635Hz). El error de ±1 pulso es 1/15635Hz= 64 µseg.
- Contando 10000 pulsos (0x2710) a $clk_{I/O}/8$ (2MHz). El error de ±1 pulso es 1/2MHz= 0.5 µseg.

Timer/Counter1: funcionalidad especial

El Timer/Counter1 tiene una funcionalidad especial, la **captura de eventos de entrada**. Cuando una entrada al micro se activa, este módulo hace una copia del valor de cuenta al registro de 16 bit **ICR1**, lo que puede usarse como una "marca de tiempo" (*timestamp*) del evento. De este modo es muy sencillo calcular el tiempo transcurrido entre eventos (por ejemplo, para medir frecuencias), y también sirve para hacer un log de eventos. Para una descripción detallada, consultar las páginas 117-119 del manual del micro (ver bibliografía).

© Universidad Europea de Madrid. Todos los derechos reservados.



© Universidad Europea de Madrid. Todos los derechos reservados.

Resumen

En este recurso, se han revisado los **circuitos de conteo y temporización** que se emplean para liberar a la CPU de este tipo de tareas que, de tenerse que ejecutar por código resultan una gran carga de CPU (por ejemplo, estar consultando el valor de una entrada digital, contando uno más, y comparando el valor con un valor objetivo). Con estos dispositivos, todo esto se realiza automáticamente fuera del núcleo de la CPU.

La temporización es básicamente una cuenta de pulsos de reloj. El tiempo contado es el número de pulsos por el tiempo transcurrido entre pulsos (que es el inverso de la frecuencia de reloj).

El reloj del que contamos pulsos se obtiene mediante un **preescalado** (dividir por un factor dado la seña del reloj principal $clk_{I/)}$, que en Arduino funciona a 16 MHz (1 pulso cada 62.5 nanosegundos).

Las temporizaciones pueden usarse para sincronizar o programar tareas, cronometrar señales externas, limitar un tiempo de espera, generar formas de ondas (como PWM), y un largo etcétera.

Existe un temporizador especial, denominado ***watchdog***, que además de poderse usar como temporizador permite al micro salir automáticamente de situaciones en las que se pueda haber quedado colgado. Si el bucle principal no reinicia el temporizador *watchdog* y este llega al tiempo límite, se puede hacer que provoque un "*reset*" del micro. Es muy útil en sistemas aislados para recuperar el control ante fallos imprevisibles o código errático.

Aunque sea un **micro de 8 bits**, dispone de un temporizador/contador de 16 bits. El **acceso** a estos **registros** se ha de realizar por tanto en **dos veces** (parte alta H y parte baja L) en un orden determinado.

U n **buen diseño** que haga uso de la circuitería adicional al núcleo (temporizadores, conversores A/D, contadores, comparadores, etc.) y emplee interrupciones como medio de comunicación con el núcleo de la CPU permitirá una **enorme eficiencia** del equipo con consumo energético mínimo.

¡Enhorabuena! Has finalizado con éxito.

© Universidad Europea de Madrid. Todos los derechos reservados.

Referencias bibliográficas

- Manual detallado (datasheet) del fabricante del procesador ATmega328 (Atmel). Páginas 93-110, y 237-252. Disponible en: <http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf> [Consultado el 20 de abril de 2016].

© Universidad Europea de Madrid. Todos los derechos reservados.