

# Temporizadores

Ángel Grover Pérez Muñoz

*angel.perez.munoz@upm.es*

Informática Industrial

4 de noviembre de 2024



*datsi*

ETSIIInf – Informática Industrial – T6. Temporizadores

**Contenido original del Prof. Juan Zamorano**

# Índice

- 1) Temporizadores/contadores hardware. Estructura y modos de uso:
  - Relojes
  - Alarmas
  - Watchdog timers
  - Pulse Width Modulation (PWM)
- 2) Temporizadores en el ATmega32U4
- 3) Programación de temporizadores para el ATmega32U4
- 4) Casos de uso: Control de motores y servomotores
- 5) Control automático: Filtros Proporcional Integral Derivativo (PID)

# Contexto: Componentes del Microcontrolador

- Los microcontroladores son computadores “completos” e incluyen:
  - Microprocesador (CPU)
  - Memorias RAM y ROM
  - Periféricos:
    - **Osciladores y temporizadores**
    - Entrada/salida digital
      - I2C, SPI, USART, CAN, Ethernet, ...
    - Entrada/salida analógica
      - ADC y DAC
    - Almacenamiento (memoria Flash)

# 1) Temporizadores y contadores hardware

# ¿Qué es un temporizador/contador HW?

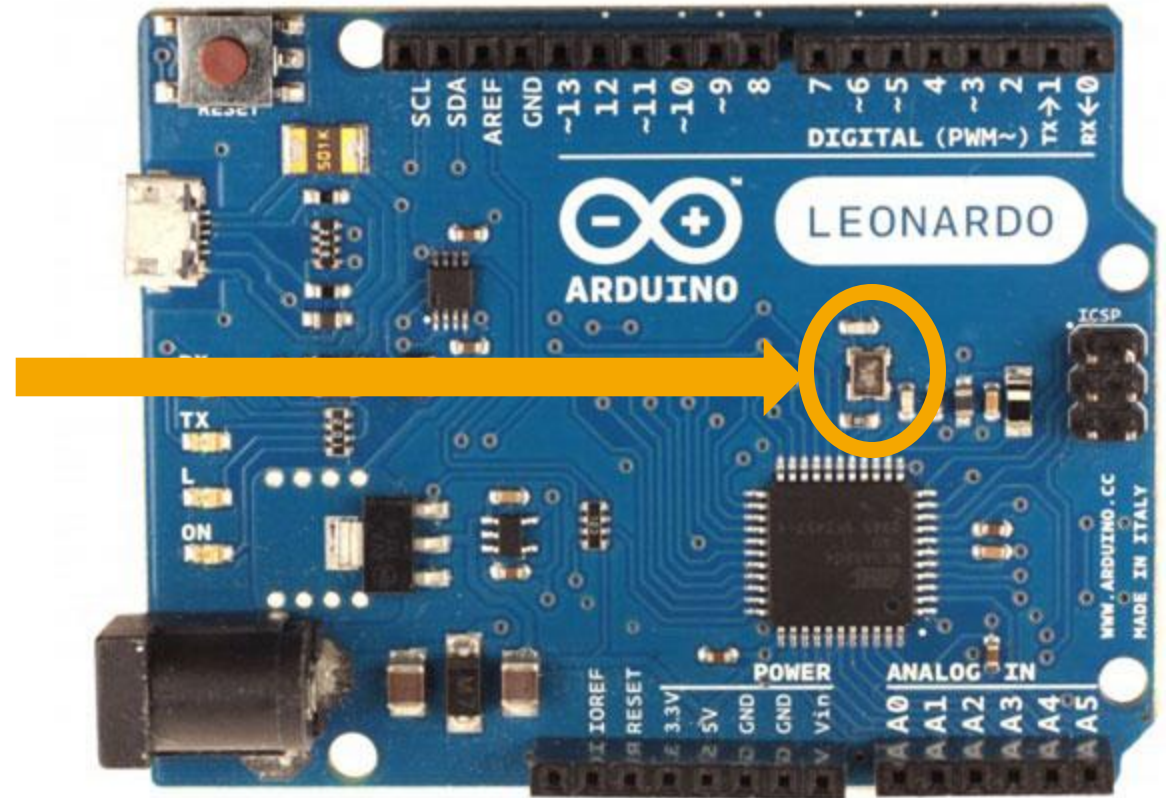
- Un **temporizador** hardware es un dispositivo incluido dentro del microcontrolador que permite **contar** sucesos periódicos o esporádicos.
  - *Suceso periódico*: Producidos internamente con un periodo fijo.  
Ejemplo: número de “tics” producidos cada 10 ms.
  - *Suceso esporádico*: Producidos externamente sin periodo fijo.  
Ejemplo: número de pulsaciones producidos por un botón.

# ¿Qué es un temporizador/contador hw?

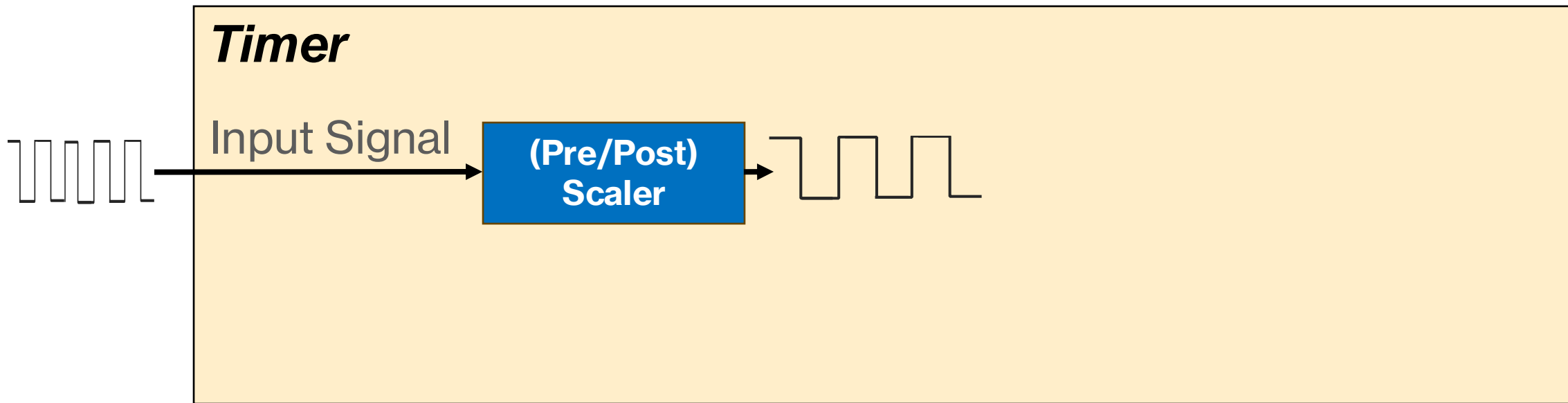
- Un **temporizador** hardware es un dispositivo incluido dentro del microcontrolador que permite **contar** sucesos periódicos o esporádicos.
  - *Suceso periódico*: Producidos internamente con un periodo fijo.  
Ejemplo: número de “tics” producidos cada 10 ms.
  - *Suceso esporádico*: Producidos externamente sin periodo fijo.  
Ejemplo: número de pulsaciones producidos por un botón.
- La frecuencia de los sucesos periódicos se puede configurar con un divisor de frecuencias programable denominado “scaler”.
  - La frecuencia base es la del reloj del sistema (oscilador).

# Generador de señal de reloj

- El  $\mu$ Controlador necesita una señal de reloj para su funcionamiento.
- Existen diversas alternativas, la más popular son los “**cristales de Cuarzo**”.
- Arduino LEONARDO cuenta con un oscilador de cristal de **16 MHz** montado en la superficie (SMD).



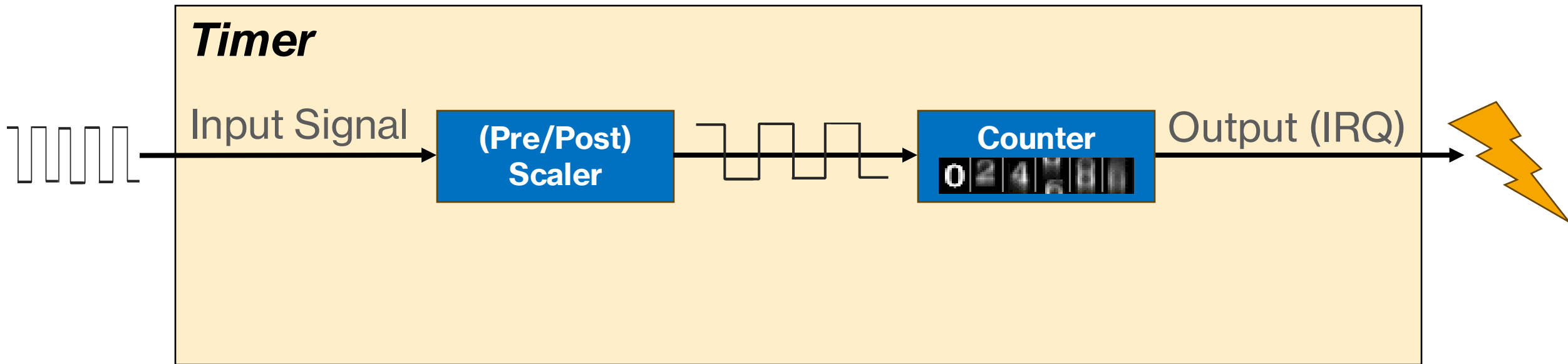
# Estructura de un temporizador HW



- Un **temporizador** hardware recibe una señal de reloj como entrada.
- Incluye un **escalador (scaler)** que permite decrementar la frecuencia del oscilador dividiéndola por un número entero configurable.
  - Es decir, el scaler actúa como un divisor de frecuencia.
  - Permite al programador elegir una variedad de frecuencias sin modificar el hardware.

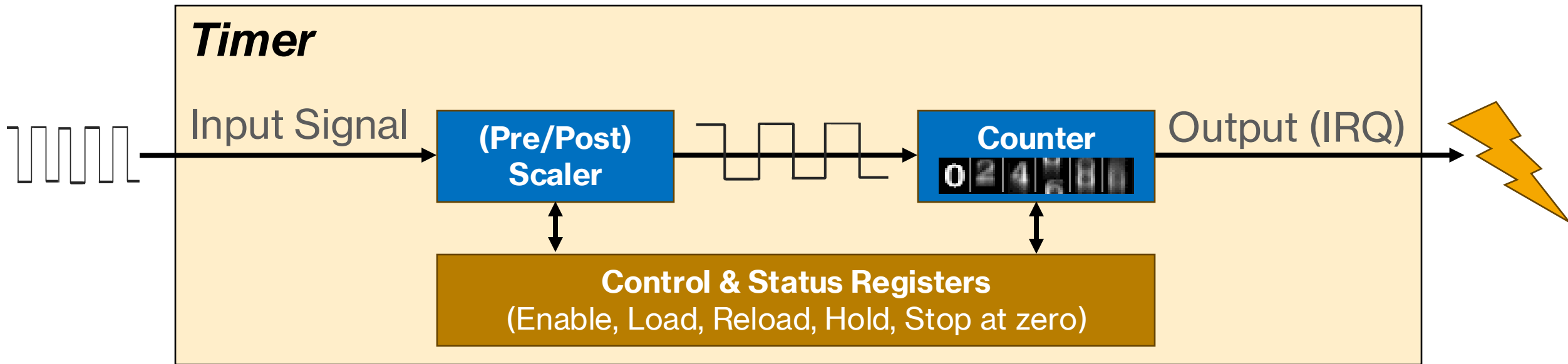


# Estructura de un temporizador HW



- Incluye un **contador**, que cuenta los pulsos o flancos que llegan por su entrada.
- La señal de reloj resultante del escalador decrementa/incrementa el registro contador. Cuando este registro alcanza un límite (cero o tope), se genera una interrupción.
- Permitiendo así, que el programador sea notificado de dicho suceso.

# Estructura de un temporizador HW

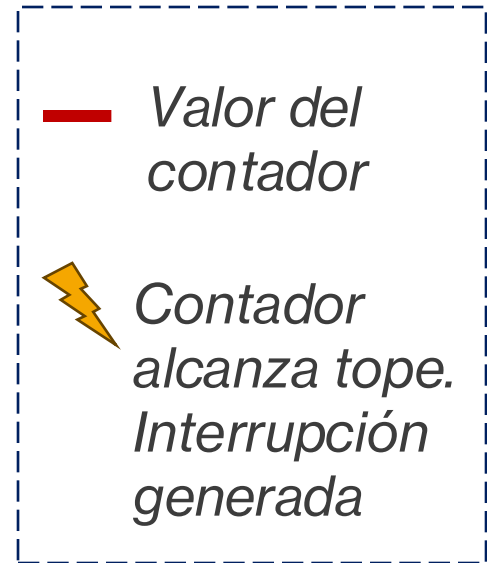
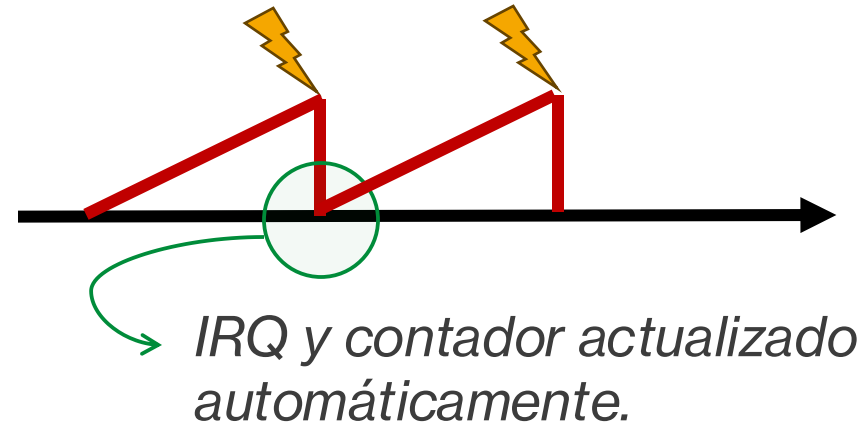


- El divisor entero del escalador, la cuenta atrás, etc. son registros configurables y mapeados en memoria principal.
  - El programador puede escribir y leer estos registros. Por ejemplo, para disminuir la frecuencia (escritura) o leer el valor de un contador (lectura).

# Modos de uso

## 1. Modo Contador Periódico:

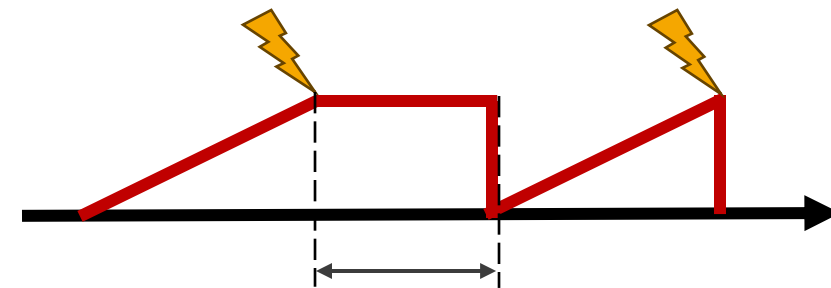
Cuando el contador alcanza el tope, este vuelve a tomar el valor inicial del contador automáticamente.



## 2. Modo Contador de Intervalos:

Cuando contador alcanza el tope, no actualiza su valor, permaneciendo con su valor.

Ejemplo: “delay”



*Tiempo durante el cual el programador trata la interrupción y reestablece el valor del contador.*

# Usos del temporizador/contador hardware

Dependiendo de estos modos de operación, y su conexión al computador se pueden implementar:

- **Temporizadores “watchdog”**. Configurado en modo “intervalos” y la interrupción se conecta a la línea de reset.
- **Temporizadores periódicos**. Configurado en modo “periódico” y la interrupción se conecta a una línea de petición de interrupción.
- **Temporizadores “single-shot”**. Configurado en modo “intervalo” y la interrupción se conecta a una línea de petición de interrupción. P. Ej: función `delay(int ms)`.
- **Generación de señales analógicas**. Configurado en modo “periódico” y la interrupción se conecta a un dispositivo analógico. P. Ej: PWM.

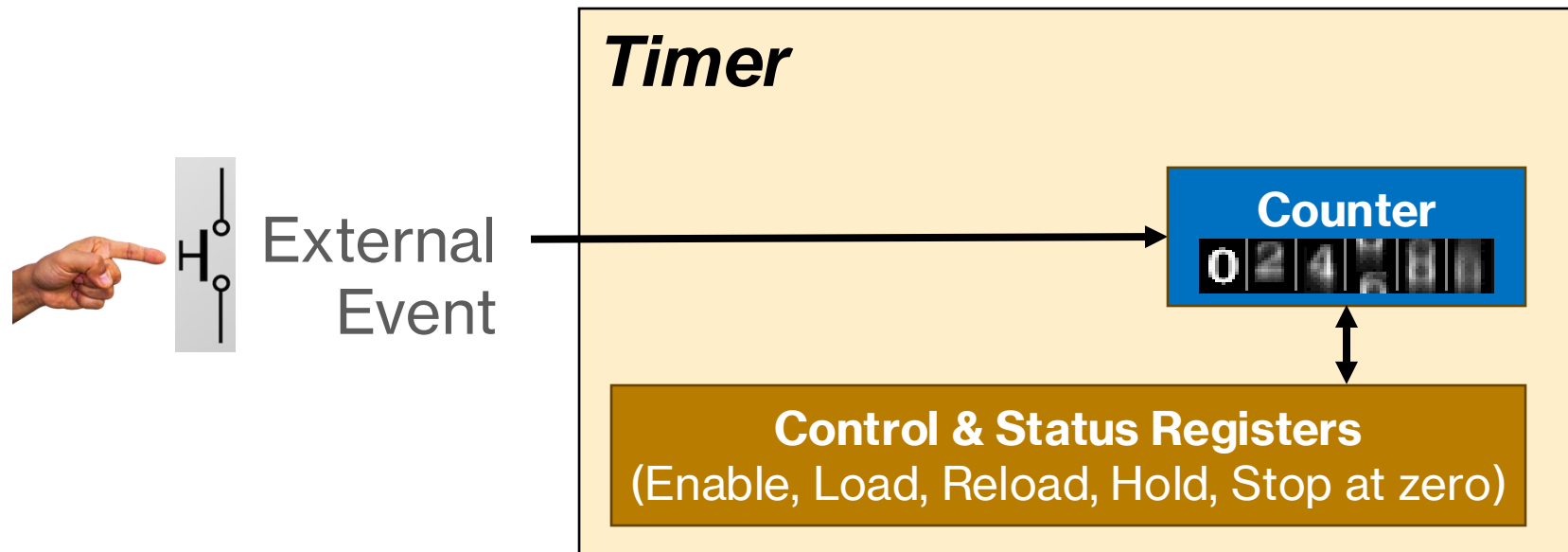
# Temporizadores Watchdog

- La salida del temporizador está conectada a una línea de reset del procesador.
- El programador debe “resetear” el contador vía software de forma periódica antes de que se genera la interrupción, evitando así el reinicio del procesador.
- Si el software se queda “colgado” no resetea el contador y el procesador se reinicia.
- Se usa en sistemas con poca accesibilidad, es decir, en casi todos los sistemas empotrados.
- ATmega32U4 incluye un temporizador *watchdog* y Arduino Leonardo cuenta con una línea de RESET.

# Modos de uso

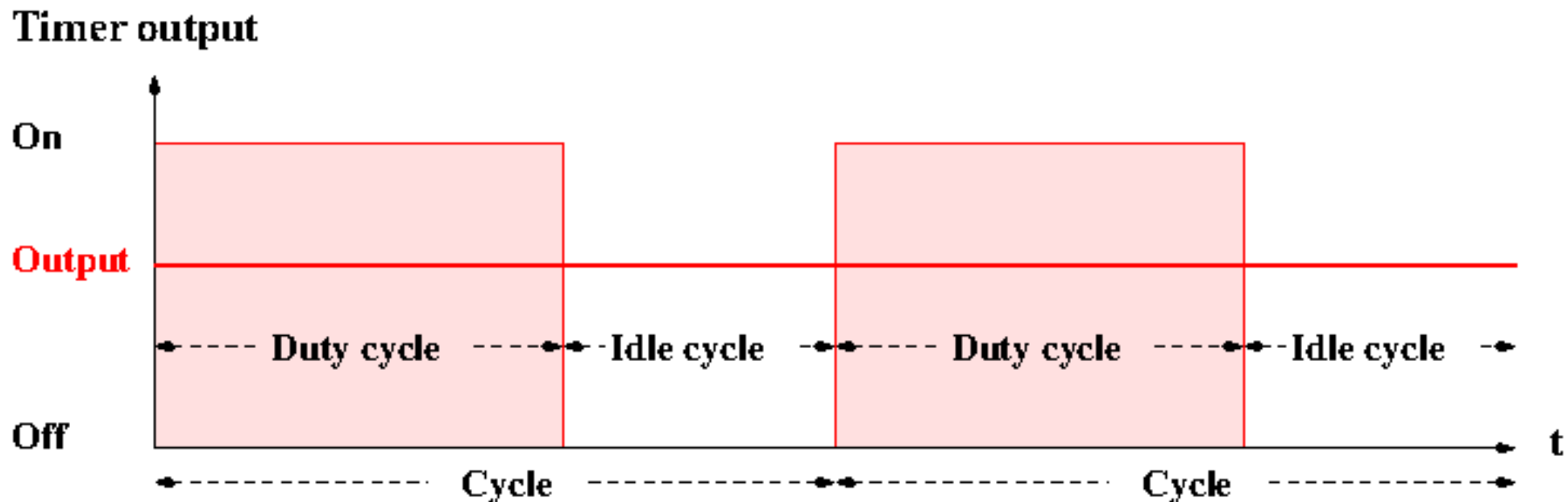
## 3. Modo Contador de sucesos:

Para ello, se conecta la señal digital a la línea de entrada del contador. Esto se puede configurar mediante los registros de control del temporizador.



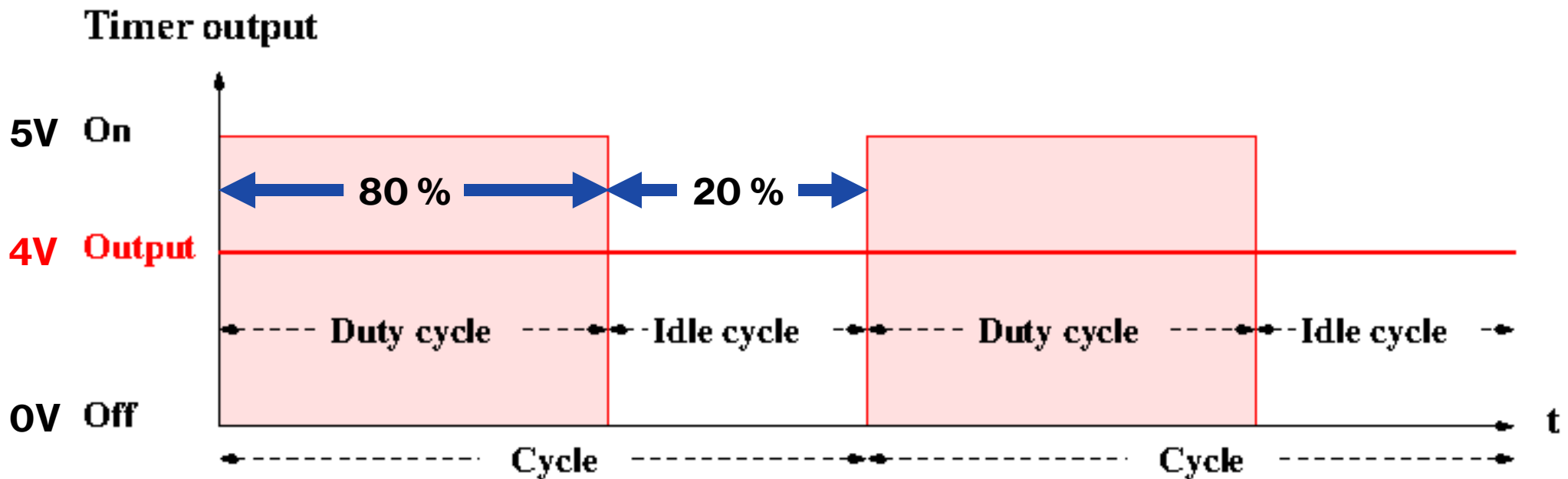
# Modulación por ancho de impulso (PWM)

- Permite generar salidas analógicas cambiando (modulando) el ancho de una onda cuadrada.
- La proporción de tiempo en que la onda está a nivel alto se conoce como **Duty Cycle**. Por contra, el **Idle Cycle** es la proporción de tiempo en que está a nivel bajo.



# Modulación por ancho de impulso (PWM)

- Por ejemplo, en un sistema cuya salida a nivel alto es de 5 V y tiene programada un PWM con un Duty Cycle al 80%, tendrá una salida promedio de  $4\text{ V} = 5\text{ V} * 0.8$ .

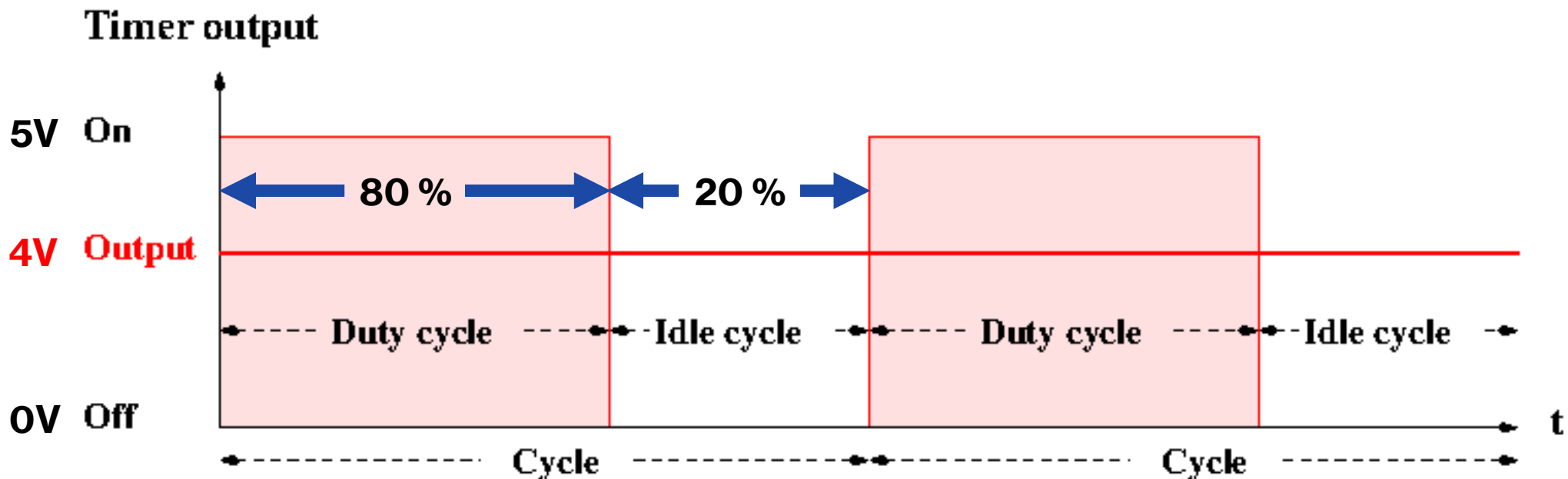




# Modulación por ancho de impulso (PWM)

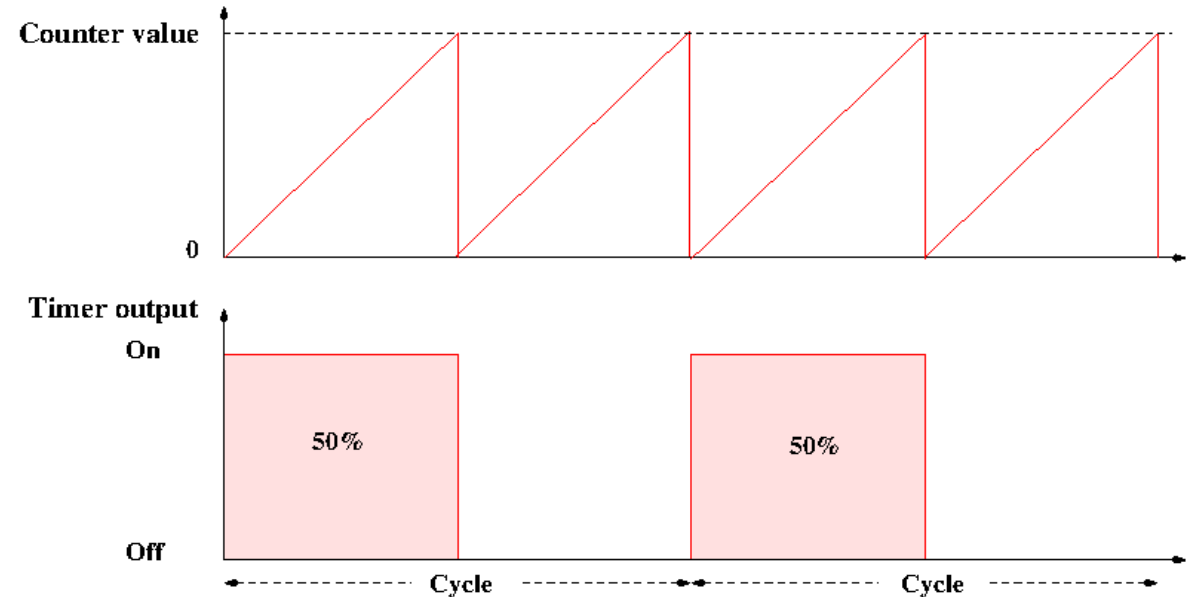
- Por ejemplo, en un sistema cuya salida a nivel alto es de 5 V y tiene programada un PWM con un Duty Cycle al 80%, tendrá una salida promedio de  $4\text{ V} = 5\text{ V} * 0.8$ .

*¡IMPORTANTE!: Este valor medio (4 V) es independiente de la frecuencia del pulso. Sin embargo, la frecuencia debe configurarse de acuerdo a la carga que controle.*



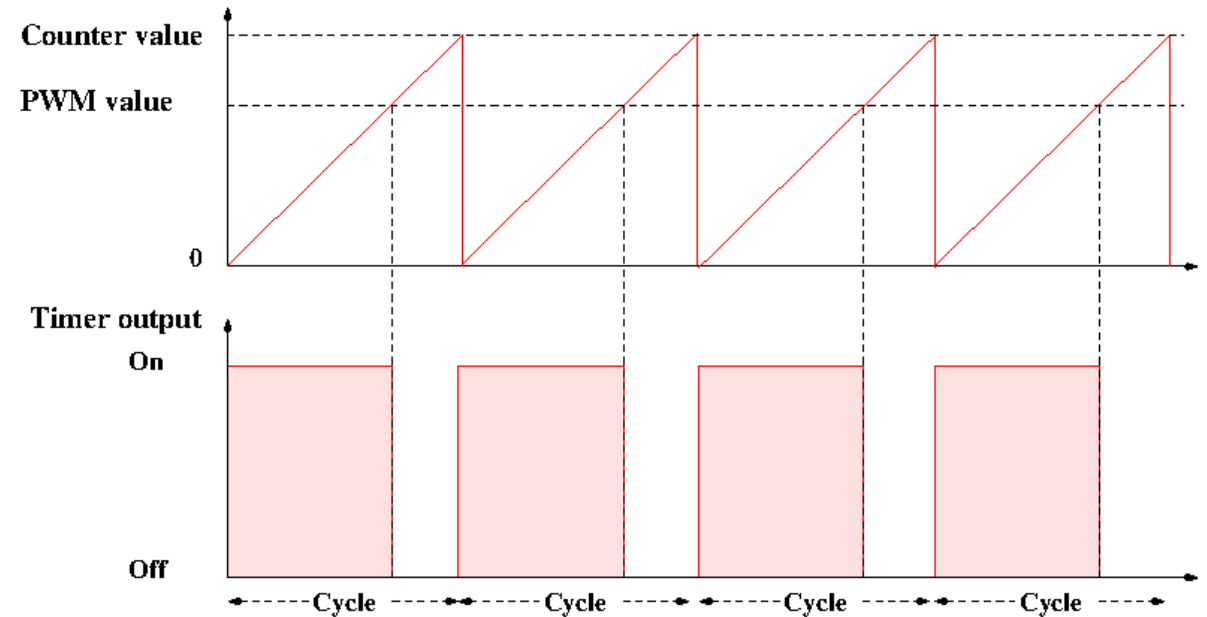
# Temporizadores PWM

- Los temporizadores básicos reciben a su entrada pulsos con un duty-cycle del 50%.
  - En cada IRQ, se invierte el valor de la salida (de ON a OFF y viceversa).
  - **PROBLEMA:** El programador está limitado a un PWM del 50%.
- La PWM requiere que el ancho de los pulsos sea variable.



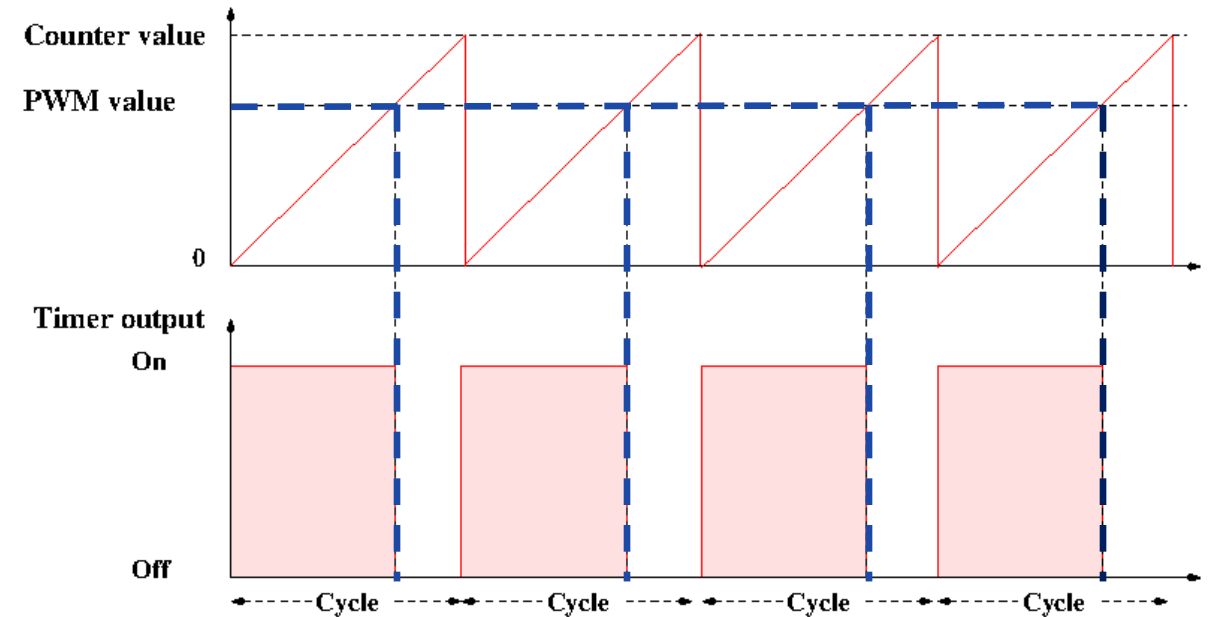
# Temporizadores PWM

- Los “PWM Timers” incluyen registros adicionales que permiten configurar el duty cycle del PWM.



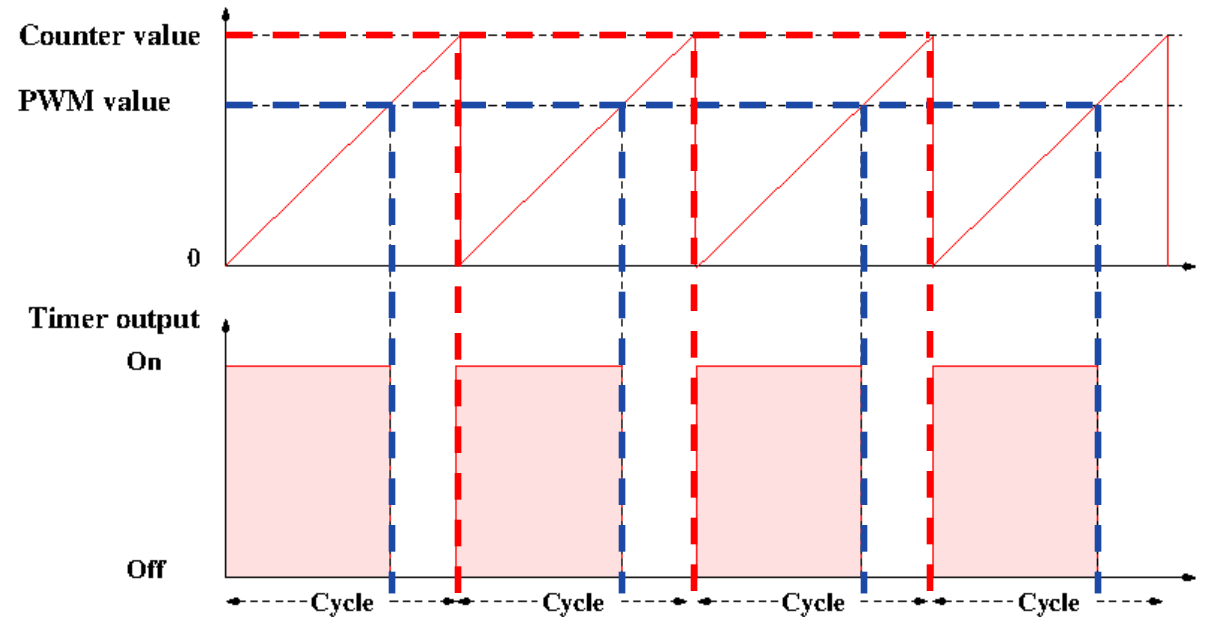
# Temporizadores PWM

- Los “PWM Timers” incluyen registros adicionales que permiten configurar el duty cycle del PWM.
- Cuando el contador alcanza el valor del PWM, la salida se invierte.



# Temporizadores PWM

- Los “PWM Timers” incluyen registros adicionales que permiten configurar el duty cycle del PWM.
- Cuando el contador alcanza el valor del PWM, la salida se invierte.
- Cuando el contador alcanza su valor máximo, la salida se vuelve a invertir



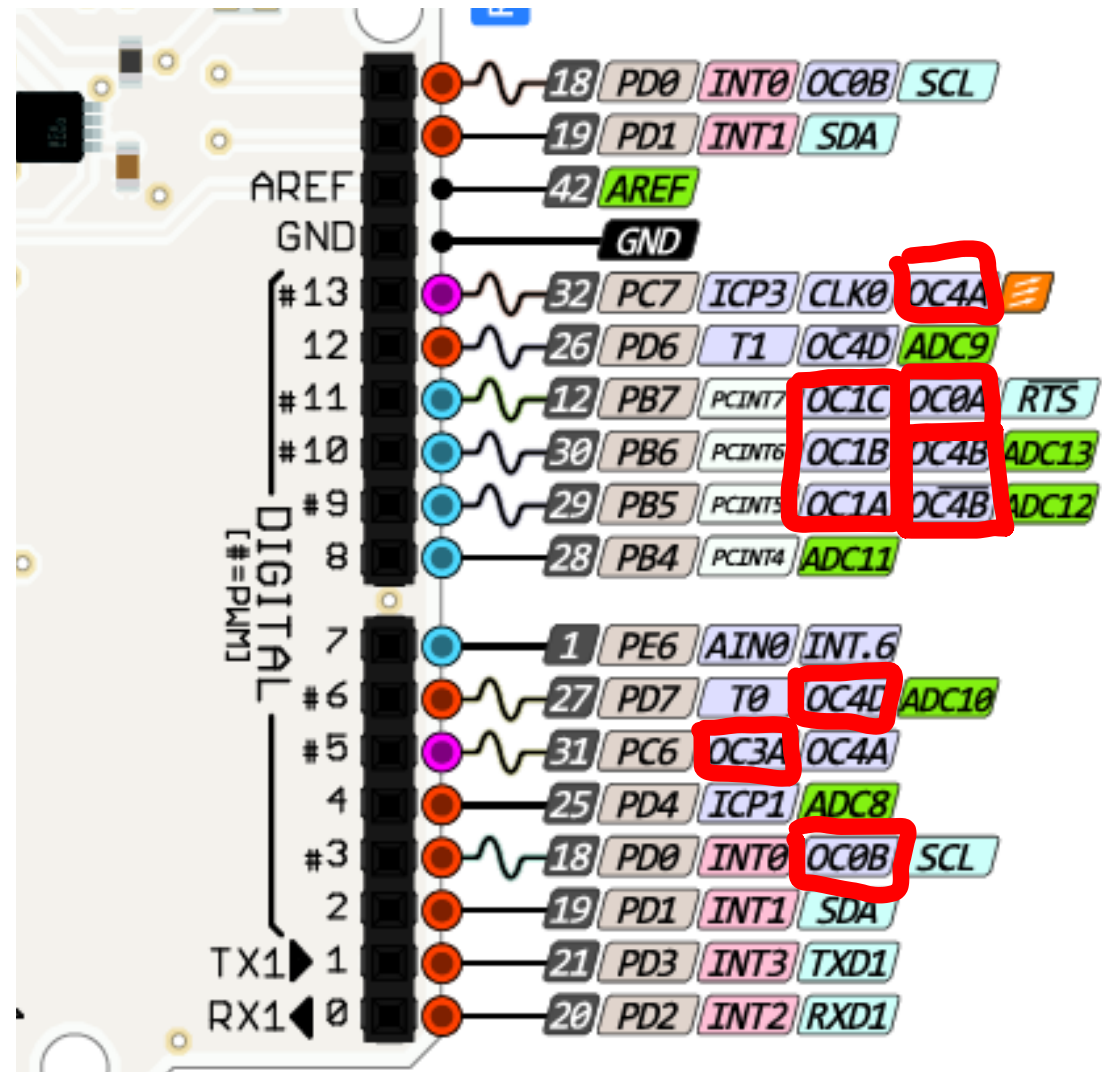
# Programación en ATmega32U4

# Temporizadores/Contadores en el ATmega32U4

- El  $\mu$ Controlador ATmega32U4 cuenta con cuatro temporizadores:
  - **T0**: Timer/Counter0 (8-bit, PWM, fast PWM, compare mode)
  - **T1**: Timer/Counter1 (16-bit, PWM, fast PWM, compare & capture mode)
  - **T3**: Timer/Counter3 (16-bit, PWM, fast PWM, compare & capture mode)
  - **T4**: Timer/Counter4 (8/10-bit, PWM, fast PWM, compare mode, High-Speed)
- Los T0, T1 y T3 pueden ser configurados con distintos valores de prescaler (divisores de frecuencia).
  - Reciben reloj CLK\_I/O (16 MHz) con un factor de scaler de /8, /64, /256, y /1024.
- El T4 (High-Speed Timer) alcanza un máximo de 64 MHz @ 5V.
  - Recibe el reloj PLL (96 MHz) con un factor de scaler de /1, /1.5 y /2.

# Timer/Counter Pins

- T0
  - OC0A: D11
  - OC0B: D3
- T1
  - OC1A: D9
  - OC1B: D10
  - OC1C: D11
- T3
  - OC3A: D5
- T4
  - OC4A: D13
  - OC4B: D10
  - OC4B: D9
  - OC4D: D6

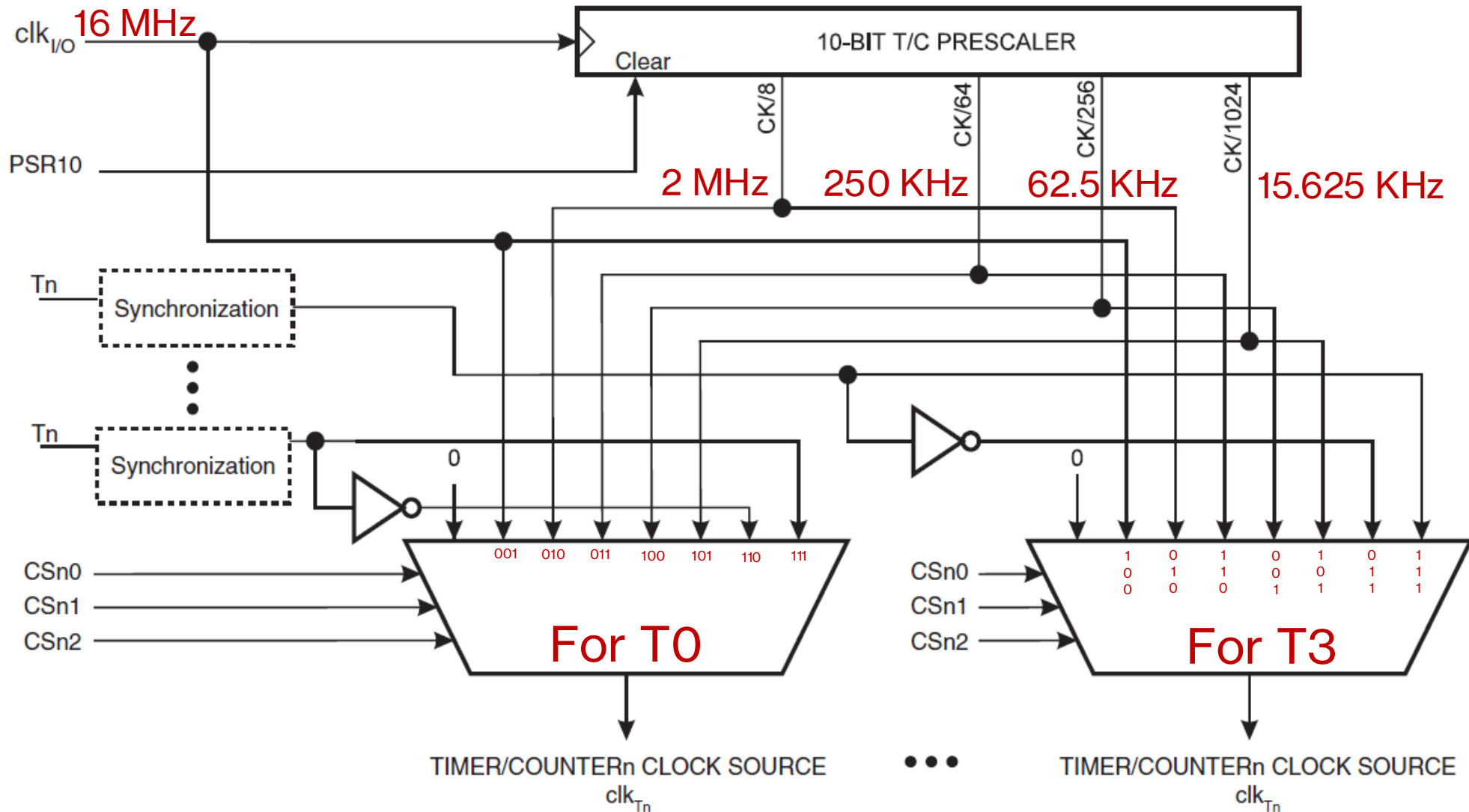


- █ Power
- █ GND
- Serial Pin
- Analog Pin
- Control
- INT
- Physical Pin
- Port Pin
- Pin function
- Interrupt Pin
- PWM Pin
- ● ● Port Power



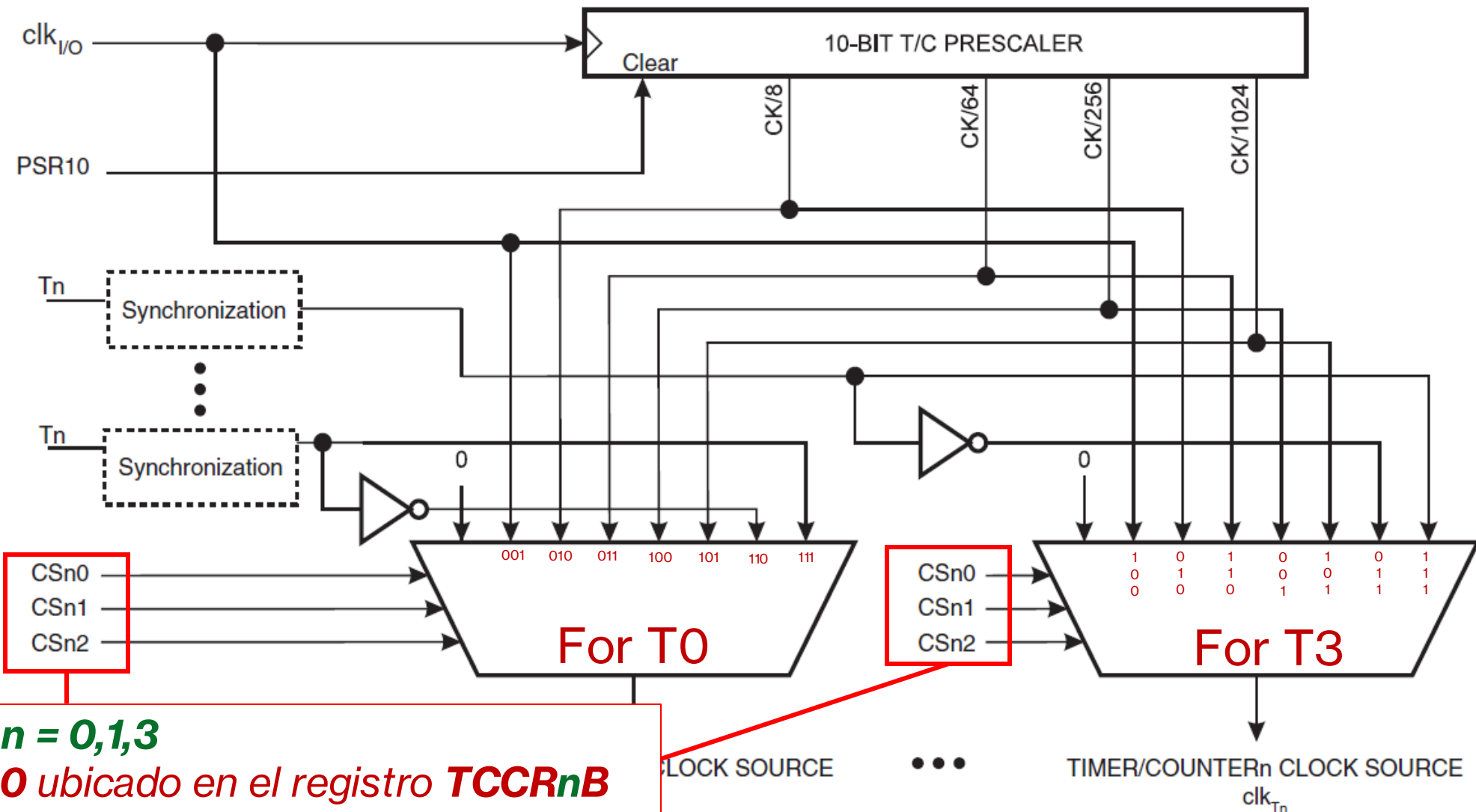
# Timer/Counter T0, T1 y T3

Figure 12-2. Prescaler for synchronous Timer/Counters



# Timer/Counter T0, T1 y T3

Figure 12-2. Prescaler for synchronous Timer/Counters



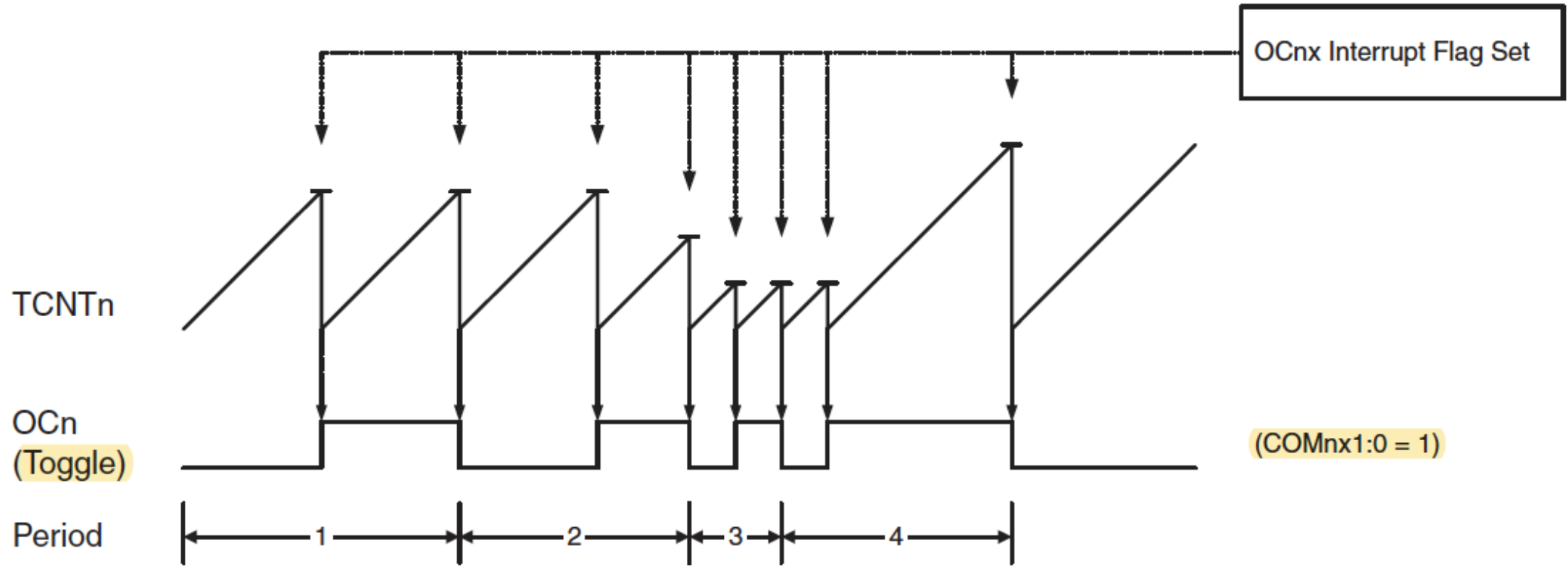


# Timer/Counter T0

- Timer de 8-bits.
- Modos de operación:
  - **Normal mode**: El contador tiene como valor de comparador el máximo valor: 0xFF. Véase la entrada “Fixed TOP Value” al multiplexor.
  - **Clear Timer on Compare Match Mode**: El contador es periódico y admite “cualquier” frecuencia.
  - **Fast PWM mode**: El contador PWM incrementa y baja a 0 de inmediato.
  - **Phase Correct PWM Mode**: El PWM incrementa y decrementa para bajar a 0.
- Modos configurables mediante los bits
  - WGM0[2:0], repartidos en los registros TCCR0A y TCCR0B.
  - COM0A[1:0] para el contador A. Ubicado en el registro TCCR0A.
  - COM0B[1:0] para el contador B. Ubicado en el registro TCCR0A.

# T0: Modo CTC

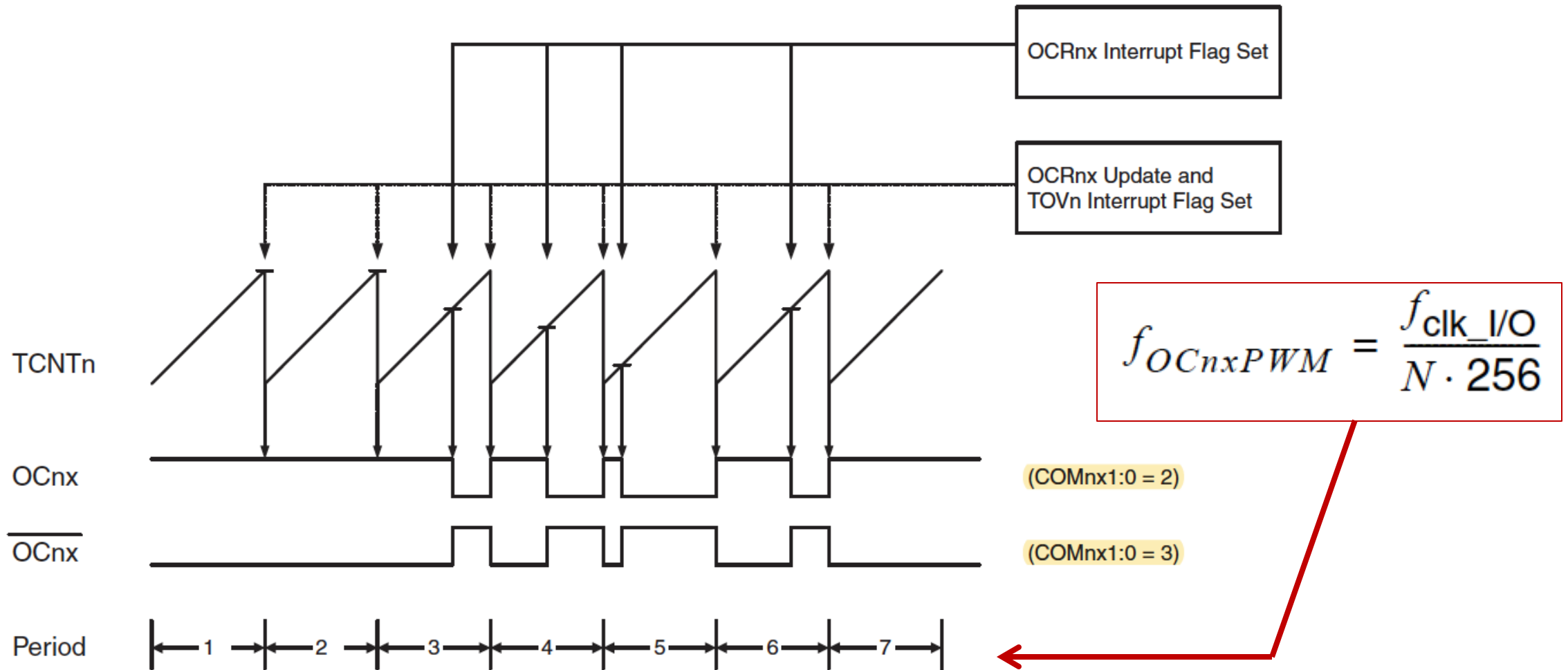
Figure 13-5. CTC Mode, Timing Diagram



$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

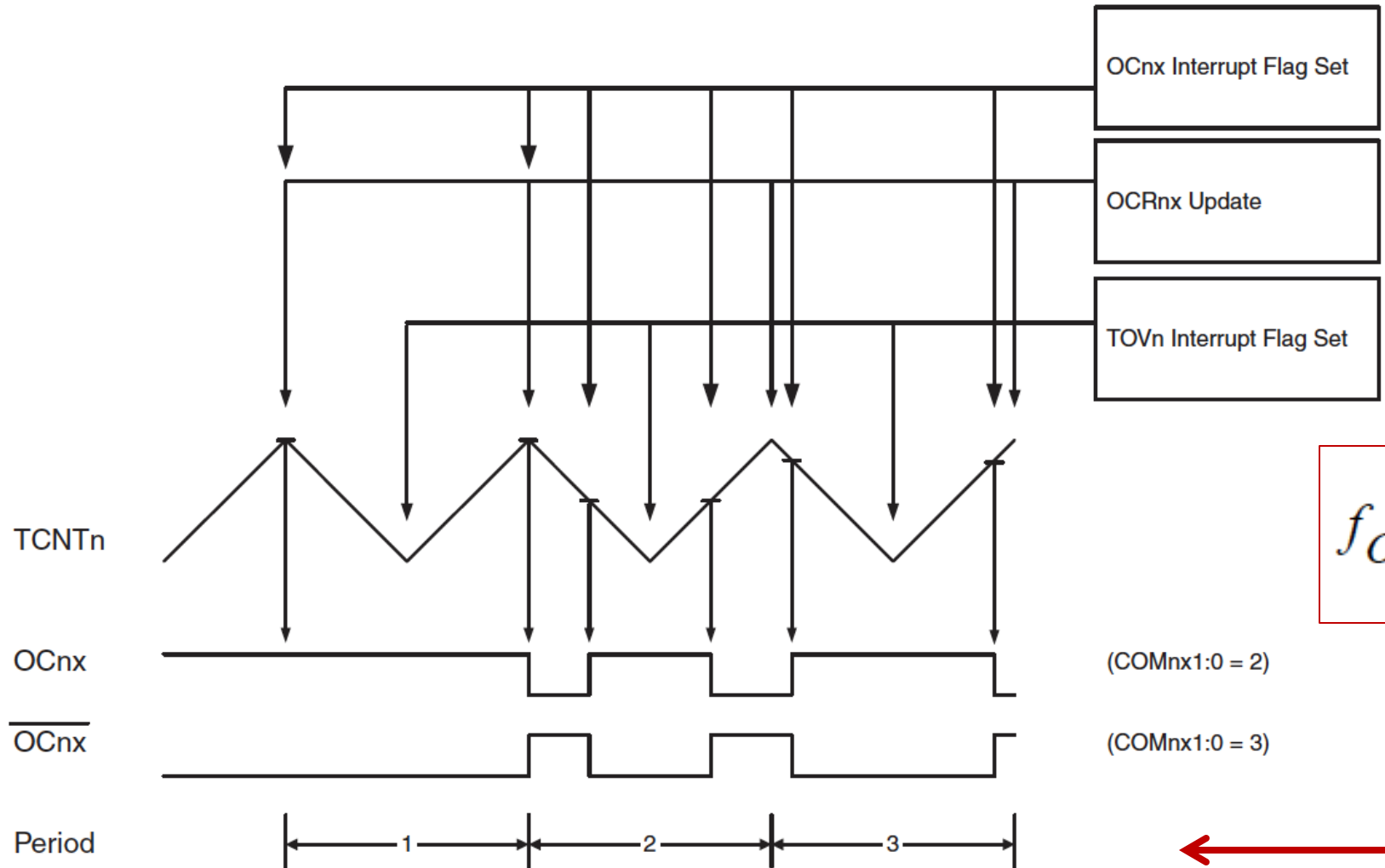
# T0: Modo Fast PWM

Figure 13-6. Fast PWM Mode, Timing Diagram



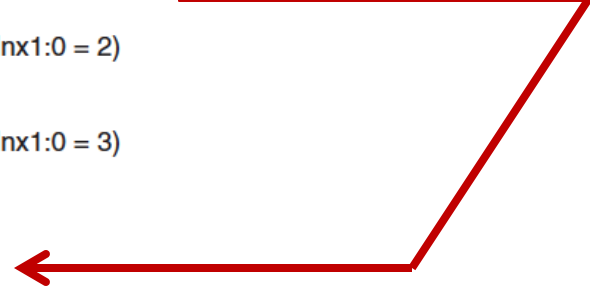
# T0: Modo Phase correct PWM

Figure 13-7. Phase Correct PWM Mode, Timing Diagram



$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{N \cdot 510}$$

(COMnx1:0 = 2)  
(COMnx1:0 = 3)



# T0: Mode configuration

TCCR0A:

7	6	5	4	3	2	1	0
COM0A 1	COM0A 0	COM0B 1	COM0B 0	–	–	WGM0 1	WGM0 0

TCCR0B:

7	6	5	4	3	2	1	0
FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00

	WGM0 2	WGM01	WGM00	Modo configurado del timer T0:	TOP	OCRA/B actualizado	TOV0 Flag actualizado
<b>0</b>	0	0	0	<i>Normal</i>	0xFF	Inmediato	MAX = 255
<b>1</b>	0	0	1	<i>PWM, Phase Correct</i>	0xFF	TOP	BOTTOM = 0
<b>2</b>	0	1	0	<i>CTC</i>	OCR0A	Inmediato	MAX = 255
<b>3</b>	0	1	1	<i>Fast PWM</i>	0xFF	TOP	MAX = 255
<b>5</b>	1	0	1	<i>PWM, Phase Correct</i>	OCR0A	TOP	BOTTOM = 0
<b>7</b>	1	1	1	<i>Fast PWM</i>	OCR0A	TOP	TOP



# T0: Mode configuration (channel A)

TCCR0A:

7	6	5	4	3	2	1	0
COM0A	COM0A	COM0B	COM0B	–	–	WGM0	WGM0
1	0	1	0			1	0

La salida del canal A (pin OC0A) depende del modo de operación (bits WGM) y los bits COM0A0 y COM0A1:

COM0A	COM0A0	Modo <i>Normal</i> o <i>CTC</i>	Modo <i>Fast PWM</i>	Modo <i>Phase Correct PWM</i>
1	1			
0	0	OC0A desconectado (no se genera onda)		
0	1	OC0A $\leftarrow$ !OC0A si match	WGM02 = 0: OC0A $\leftarrow$ !OC0A si match WGM02 = 1: OC0A desconectado	
1	0	OC0A $\leftarrow$ 0 si match	OC0A $\leftarrow$ 0 si match OC0A $\leftarrow$ 1 si TOP	
1	1	OC0A $\leftarrow$ 1 si match	OC0A $\leftarrow$ 1 si match OC0A $\leftarrow$ 0 si TOP	

# T0: Mode configuration (channel B)

TCCR0A:

7	6	5	4	3	2	1	0
COM0A	COM0A	COM0B	COM0B	–	–	WGM0	WGM0
1	0	1	0			1	0

La salida del canal B (pin OC0B) depende del modo de operación (bits WGM) y los bits COM0B0 y COM0B1:

COM0B	COM0B0	Modo Normal y CTC	Modo Fast PWM	Modo Phase Correct PWM
1	0	OC0B ← 0 si match	OC0B ← 0 si match OC0B ← 1 si TOP	OC0B ← 0 si match OC0B ← 1 si TOP
1	1	OC0B ← 1 si match	OC0B ← 1 si match OC0B ← 0 si TOP	OC0B ← 1 si match OC0B ← 0 si TOP
0	0	OC0B desconectado (no se genera onda por el pin)		
0	1	OC0B ← !OC0B si match	Reservado	

# T0: Prescaler selection

TCCR0B:

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

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)

$$clk_{I/O} = 16 \text{ MHz}$$

# T0: Interrupt Requests

## TIFRO:

Bit	7	6	5	4	3	2	1	0
	–	–	–	–	–	<b>OCF0B</b>	<b>OCF0A</b>	<b>TOV0</b>
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Biestables de **petición** de interrupción para el Timer 0:

- **OCF0B**: Puesto a 1 cuando OCR0B es igual al contador (TCNT0)
- **OCF0A**: Puesto a 1 cuando OCR0A es igual al contador (TCNT0)
- **TOV0**: Puesto a 1 cuando hay overflow en el contador (TCNT0).
  - El overflow puede ocurrir por encima ( $>255$ ) o por debajo ( $<0$ ), dependiendo del modo de configuración.

# T0: Interrupt Requests

## TIMSK0:

Bit	7	6	5	4	3	2	1	0
	–	–	–	–	–	<b>OCIE0B</b>	<b>OCIE0A</b>	<b>TOIE0</b>
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Biestables de **máscara** de interrupción para el Timer 0:

- **OCIE0B**: Puesto a 1 para habilitar interrupciones cuando  $OCR0B == TCNT0$ .
- **OCIE0A**: Puesto a 1 para habilitar interrupciones cuando  $OCR0A == TCNT0$ .
- **TOIE0**: Puesto a 1 para habilitar interrupciones de overflow  $TOV0$ .

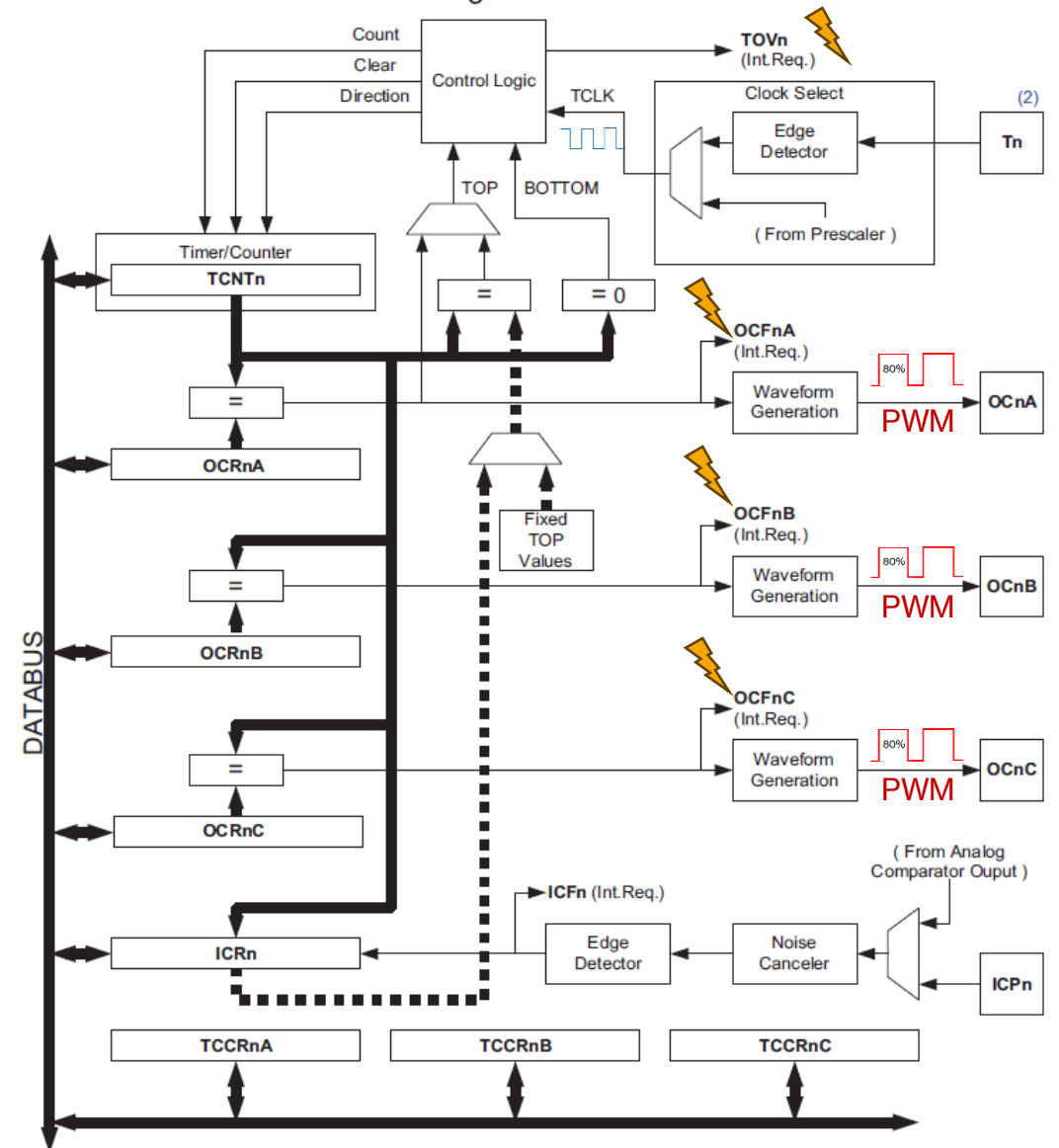


# Timer/Counter T1 y T3

## Registros (n = 1, 3):

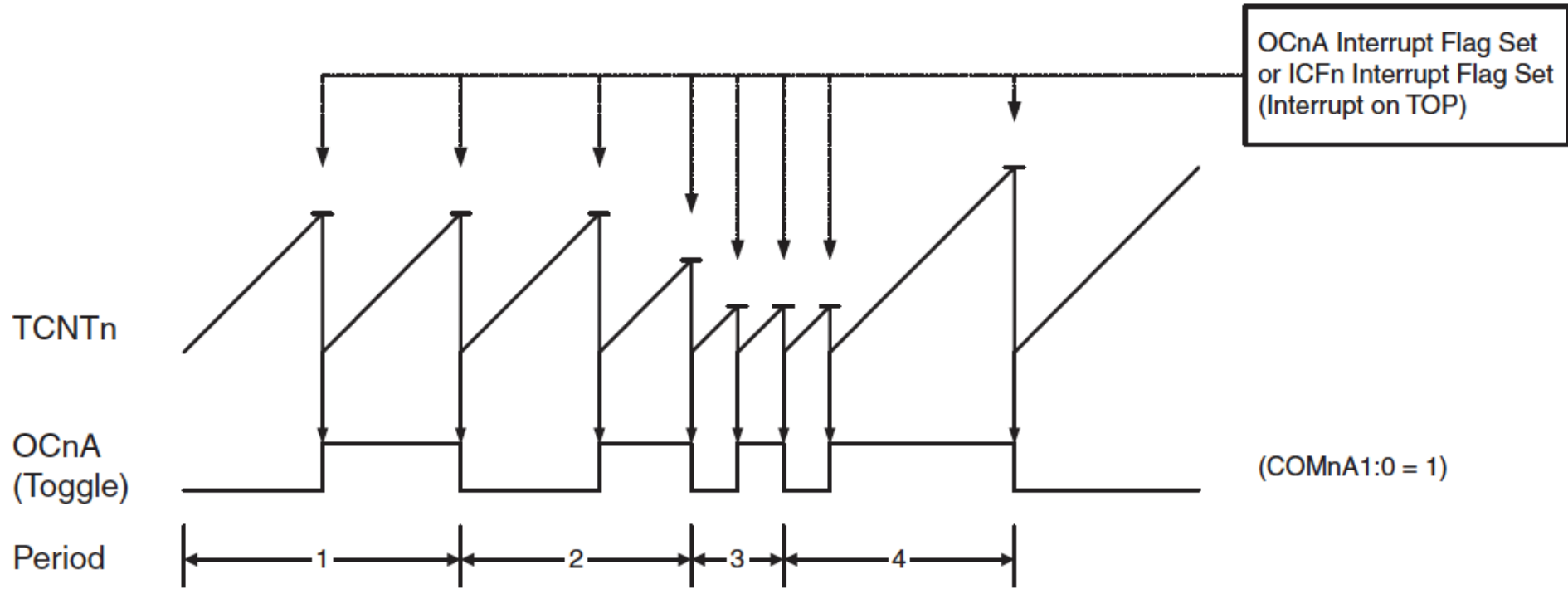
- **TCNTn[H:L]:**  
Contiene el valor del contador.
- **OCRnA[H:L], OCRnB[H:L] y OCRnC[H:L]:**  
“Output compare registers”
- **ICRn[H:L]:**  
“Input Capture unit Register”.
- **TCCRnA, TCCRnB y TCCRnC:**  
“Timer/Counter control registers”.
- **TIMSK0:**  
“Timer/Counter Interrupt Mask Register”
- **TIFR0:**  
“Timer/Counter Interrupt Flag Register”

Figure 14-1. 16-bit Timer/Counter Block Diagram<sup>(1)</sup>



# T1, T3: Modo CTC

Figure 14-6. CTC Mode, Timing Diagram



$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

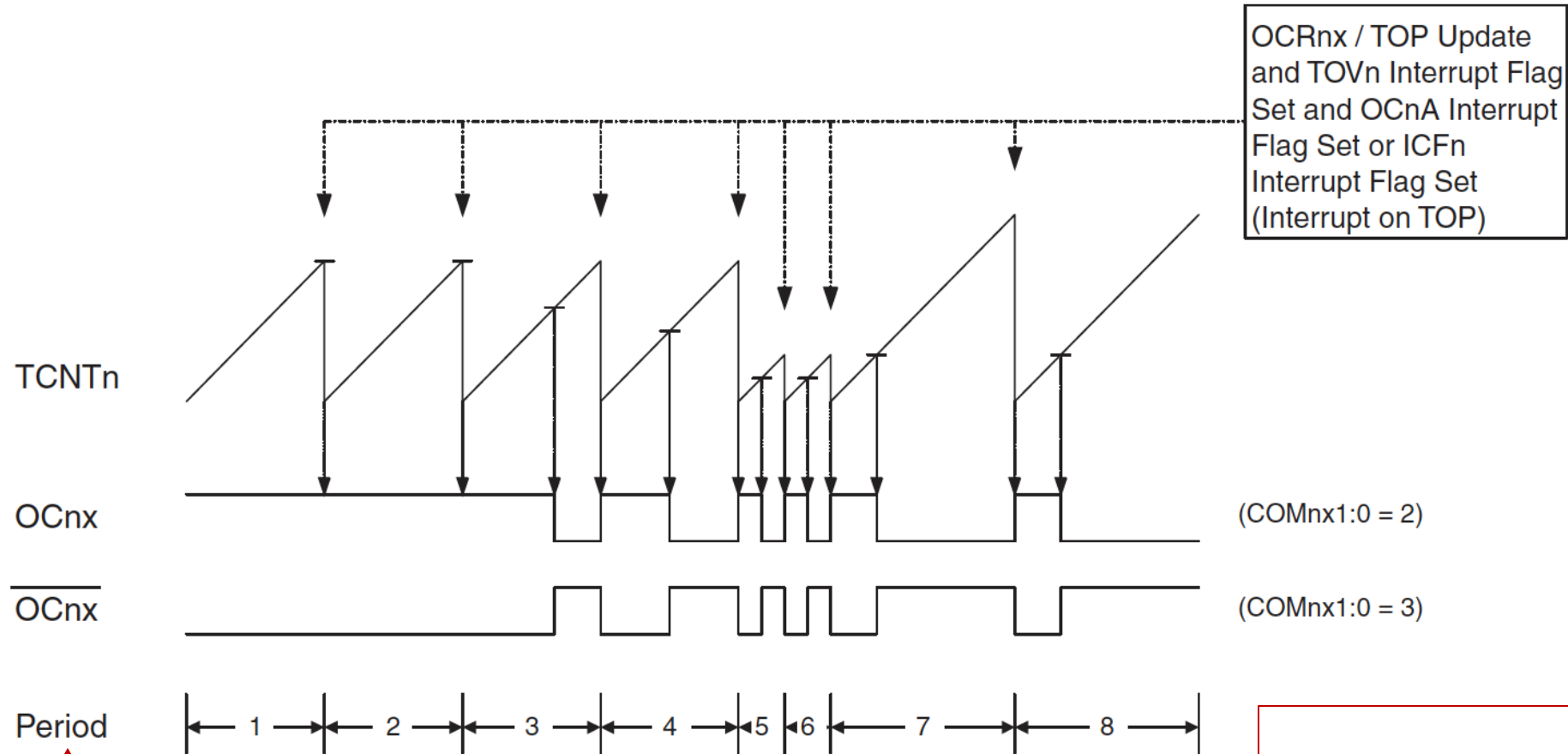


# T1, T3: Modo CTC

- El valor de tope puede configurarse a OCRnA o ICRn.
- Cuando se alcanza el tope:
  - El contador vuelve a 0x0000.
  - Se activa el biestable de petición de interrupción OCFnA o ICFn, dependiendo del valor elegido como tope.
- El valor de tope se puede actualizar en la RTI.
  - No existe double buffering, el cambio se efectúa inmediatamente.
  - Peligro: si tope < contador, el contador incrementará hasta 0xFFFF.
- La salida de su canal OCRnA puede configurarse para que se inviertan en cada match.

# T1, T3: Modo Fast PWM

Figure 14-7. Fast PWM Mode, Timing Diagram



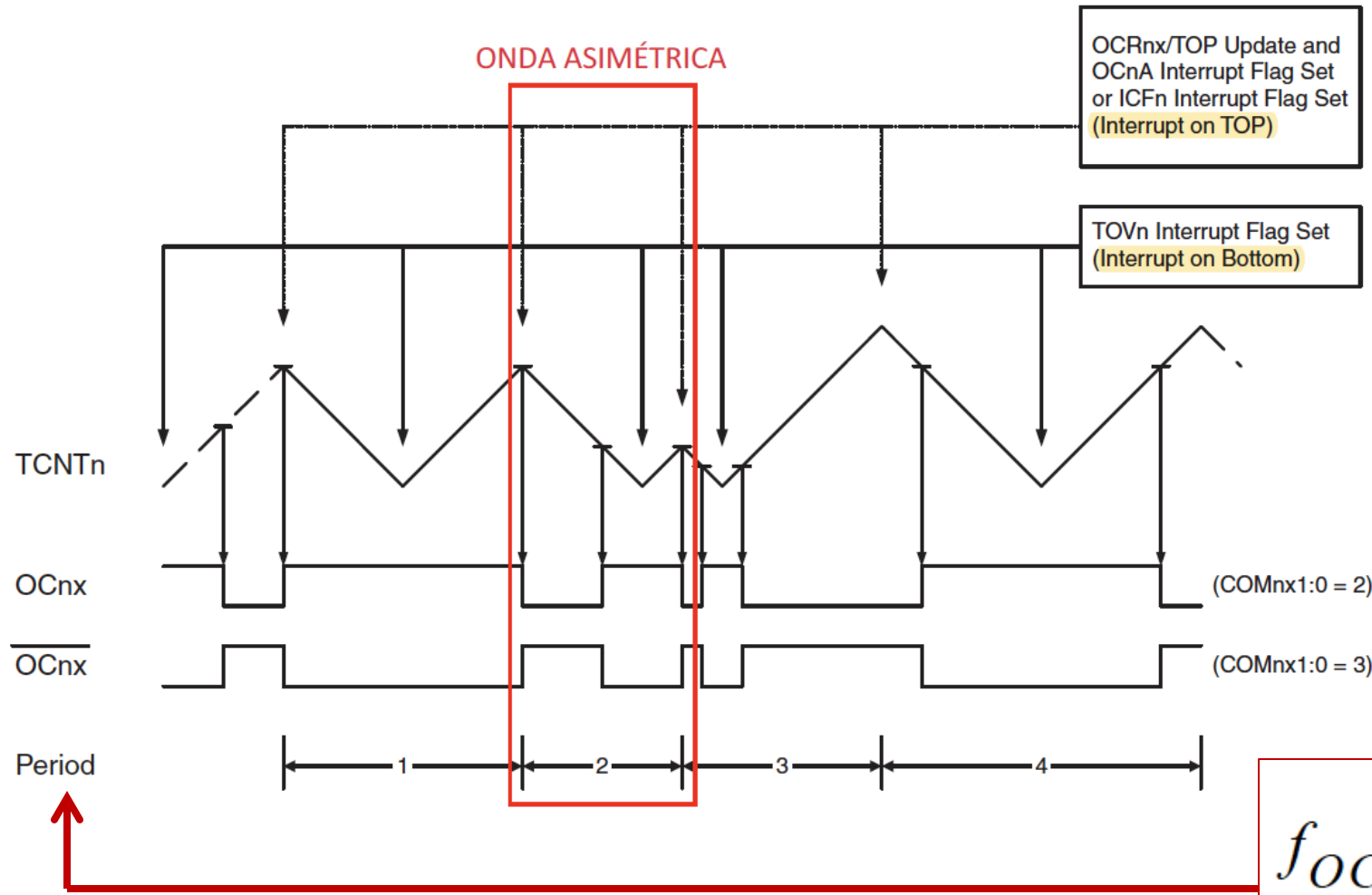
$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

# T1, T3: Modo Fast PWM

- El valor de tope puede configurarse a OCRnA, ICRn, o valores fijos (0x0FF, 0x1FF, 0x3FF).
  - Cuando OCRnA es usado como tope, este no se puede usar para generar ondas PWM.
- Cuando el contador alcanza el tope:
  - El contador vuelve a 0x0000.
  - Se activa el biestable de petición de interrupción de overflow TOVn.
  - Se activa el biestable de petición de interrupción OCFnA o ICFn, dependiendo del valor elegido como tope.
  - Se produce un cambio a nivel ALTO en la salida.
- Cuando el contador es igual al valor de OCRn{A/B/C}:
  - La salida del pin correspondiente se invierte.
  - Se activa el biestable de petición de interrupción del canal correspondiente OCFn{A/B/C}.
- Hay double buffering para OCRnA (tope). El cambio de este registro se hace efectivo cuando el contador vuelva a alcanzar el tope.
- La salida de sus canales OCRn{A/B/C} pueden configurarse para que se invierta.

# T1,T3: Modo Phase Correct PWM

Figure 14-8. Phase Correct PWM Mode, Timing Diagram



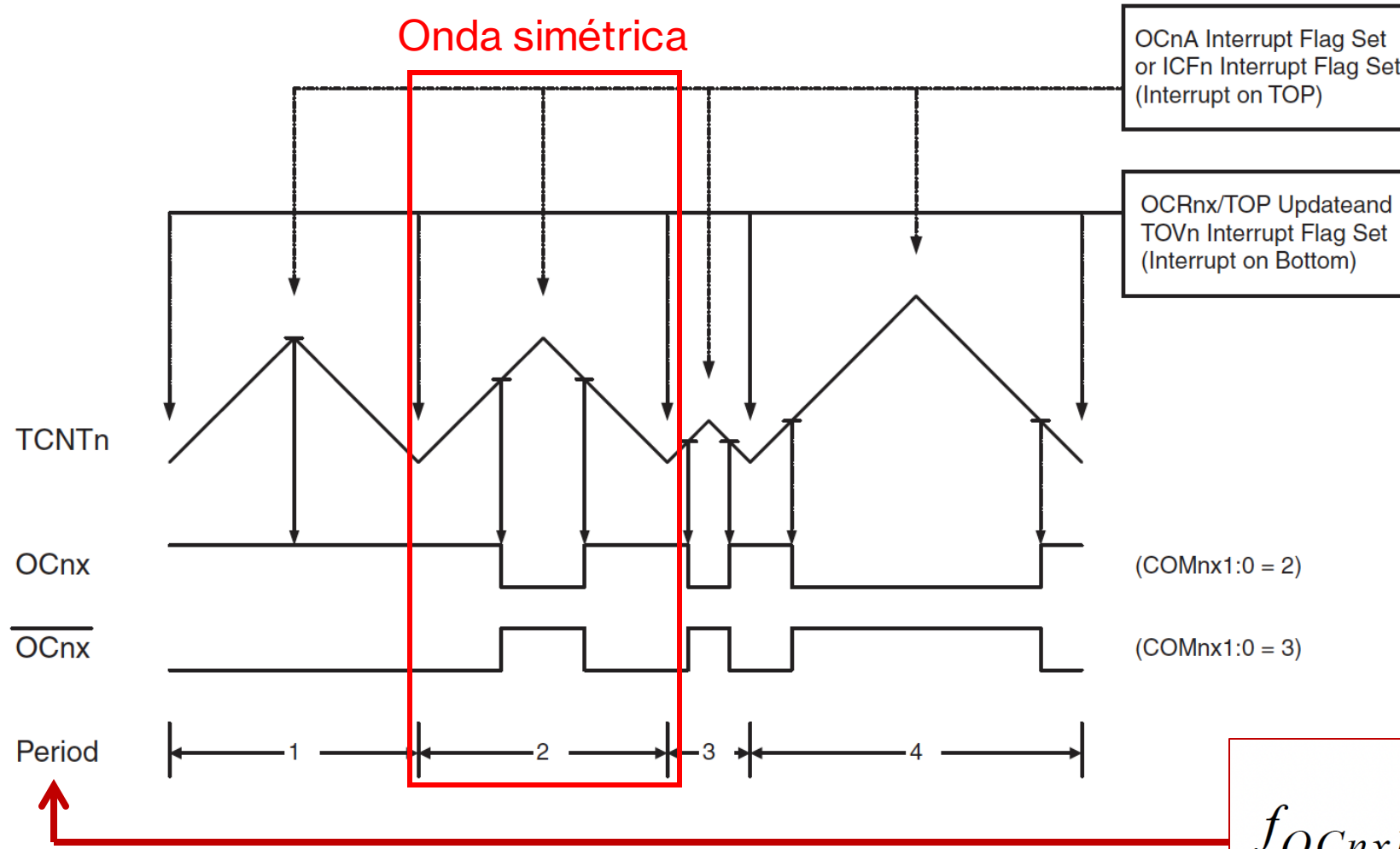
$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

# T1, T3: Modo Phase Correct PWM

- El valor de tope puede configurarse a OCRnA, ICRn o valores fijos (0x0FF, 0x1FF, 0x3FF).
  - Cuando OCRnA es usado como tope, este no se puede usar para generar ondas PWM.
- Cuando el contador alcanza el tope:
  - El contador vuelve a 0x0000 escalonadamente (rampa de descenso).
  - Se activa el biestable de petición de interrupción OCRnA o ICRn, dependiendo del valor elegido como tope.
  - Hay double buffering para OCRnx. El cambio de este registro se hace efectivo cuando el contador vuelva a alcanzar el **tope**.
- Cuando el contador es igual al valor de OCRn{A/B/C}:
  - Si es rampa de subida: salida a nivel BAJO.
  - Si es rampa de bajada: salida a nivel ALTO.
- Cuando el contador alcanza el 0x0000
  - Se genera la interrupción TOVn.
- Hay ondas asimétricas porque cada ciclo incluye dos Tops que pueden variar (∨)
  - Evitable con el modo correcto en fase y frecuencia.

# T1, T3: Modo Phase & Freq Correct PWM

Figure 14-9. Phase and Frequency Correct PWM Mode, Timing Diagram



$$f_{OCnxPFCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

# T1, T3: Modo Phase Correct PWM

- El valor de tope puede configurarse a OCRnA, ICRn o valores fijos (0x0FF, 0x1FF, 0x3FF).
  - Cuando OCRnA es usado como tope, este no se puede usar para generar ondas PWM.
- Cuando el contador alcanza el tope:
  - El contador vuelve a 0x0000 escalonadamente (rampa de descenso).
  - Se activa el biestable de petición de interrupción OCRnA o ICRn, dependiendo del valor elegido como tope.
- Cuando el contador es igual al valor de OCRn{A/B/C}:
  - Si es rampa de subida: salida a nivel BAJO.
  - Si es rampa de bajada: salida a nivel ALTO.
- Cuando el contador alcanza el 0x0000
  - Se genera la interrupción TOVn.
  - Hay double buffering para OCRnx. El cambio de este registro se hace efectivo cuando el contador vuelva a alcanzar el **cero**.
- Hay ondas simétricas porque el periodo incluye un solo top (/\\)

# TI: Programación de interrupción periódica

## Example 1

1. Setup the timer 1 in normal mode:
  - a. Pin configuration: nothing!
  - b. Mode configuration: in normal mode.
  - c. IRQ configuration: enable overflow IRQs.
  - d. Clock configuration: set the prescaler (/1).
2. Handle overflow IRQs:
  - a. Increase a software counter variable.
  - b. Toggle the internal LED.
3. Main loop:
  - a. Atomic read of this software counter.
  - b. Print its value.
  - c. Calculate the passage of time.

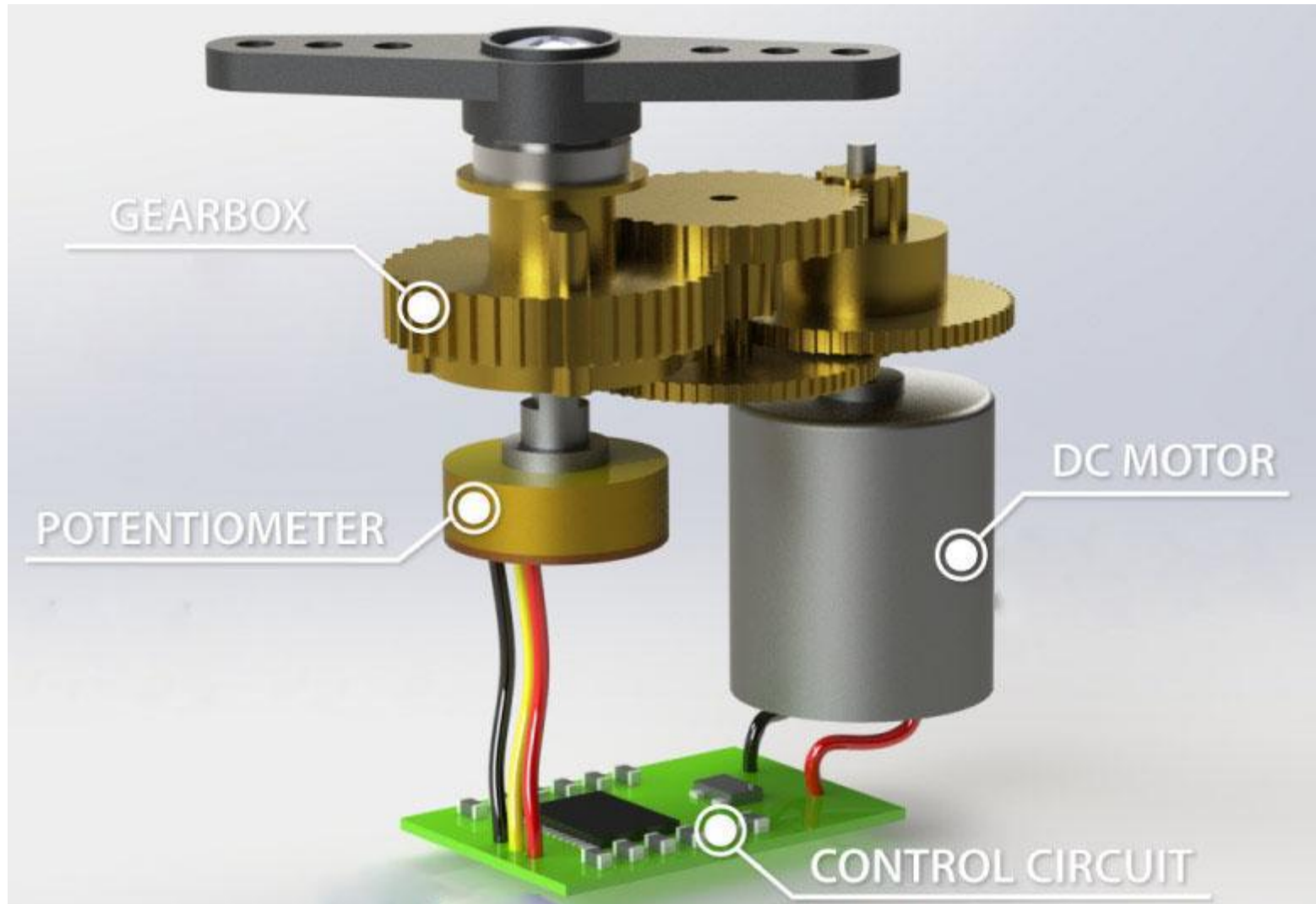
## Example 2

1. Setup the timer 1 in CTC mode:
  - a. Pin configuration: Toggle channels A & B
  - b. Mode configuration: in CTC mode with OCR1A as the top.
  - c. IRQ configuration: enable “compare match” IRQs on channels A & B.
  - d. Clock configuration: 10 Hz.
2. Handle overflow IRQs:
  - a. Increase software counters variable.
  - b. Toggle the internal LED.
3. Main loop:
  - a. Atomic read of this software counter.
  - b. Print its value.



# Montaje del Servomotor

# Servomotor



# Servomotor de posición

- Tienen un rango de giro de  $90^\circ$ .
- Concretamente, la posición se puede controlar desde los  $-45^\circ$  hasta los  $+45^\circ$ .
- Responden a una frecuencia de **50 Hz** de PWM, es decir, con un periodo de **20 ms**.
- **El valor de duty cycle controla el ángulo del servomotor:**

Duty Cycle	Ángulo
0.5 ms	$-90^\circ$
1 ms	$-45^\circ$
1.5 ms	$0^\circ$
2 ms	$45^\circ$
2.5 ms	$90^\circ$



# Servomotor de rotación continua

- Tienen un rango de giro de  $360^\circ$ .
- Se controla la velocidad y dirección de giro desde las **180 rpm** hasta las **-180 rpm** (revoluciones por minuto).
- Responden a una frecuencia de **50 Hz** de PWM, es decir, con un periodo de **20 ms**.
- **El valor de duty cycle controla la velocidad y dirección de giro:**

Duty Cycle	Velocidad
1.40 ms	180 rpm
1.45 ms	90 rpm
1.50 ms	0 rpm
1.55 ms	-90 rpm
1.60 ms	-180 rpm



# Sevomotor

- Cuenta con tres cables, típicamente:
  - Potencia (rojo)
  - Tierra (marrón)
  - Señal de control PWM (típicamente naranja, amarillo, blanco o azul)
- Conexiones:

Servomotor	Arduino
Power	5V
Ground	GND
PWM signal	Pin con soporte PWM



# Servo Library

- Uso de esta librería para realizar pruebas de montaje, por ejemplo: que el servomotor está bien conectado.
- Los pines PWM están marcados con ~ o #: 3, 5, 6, 9, 10, 11, 13
- Sin embargo, la Servo Library solo soporta los pines D9 y D10.
- La función `attach(int pin, int min, int max)` permite configurar un pin como PWM.
  - **pin**: Arduino digital pins with PWM support (9 or 10)
  - **min**: The pulse width, in microseconds, corresponding to the minimum (-90 degree) angle on the servo (defaults to 544)
  - **max**: The pulse width, in microseconds, corresponding to the maximum (90 degree) angle on the servo (defaults to 2400)
- **IMPORTANTE**: A diferencia de Arduino Library (`AnalogWrite`), Servo Library admite frecuencias de 50 Hz (periodo de 20 ms).
- La función `writeMicroseconds(int microsecs)` permite controlar el servomotor:
  - **microsecs**: Valor de duty-cycle en la salida del pin configurado.

# Ejemplo de control con la librería Servo

## 1. Inicialización y configuración

```
#include <Servo.h>

Servo myservo; // Crea objeto servo
void setup() {
  // Configura objeto myservo al pin D9 y
  // 500us (0.5ms) : -90°
  // 2500us (2.5ms) : +90°
  myservo.attach(9, 500, 2500);

  // Configura pin con duty-cycle de 1500us (1.5ms),
  // i.e.: centra el servo en la posición de 45°
  myservo.writeMicroseconds(1500);
}
```

# Ejemplo de control con la librería Servo

- Ejemplo de funcionamiento con llamadas a la función writeMicroseconds.
- Los giros son bruscos.

```
void loop() {  
  static bool left = true;  
  if (left) {  
    // Mueve a la Izqda: -90º  
    // 500us de duty-cycle  
    myservo.writeMicroseconds(500);  
  } else {  
    // Mueve a la Dcha: +90º  
    // 2500us de duty-cycle  
    myservo.writeMicroseconds(2500);  
  }  
  left != left;  
  
  delay(3000);  
}
```



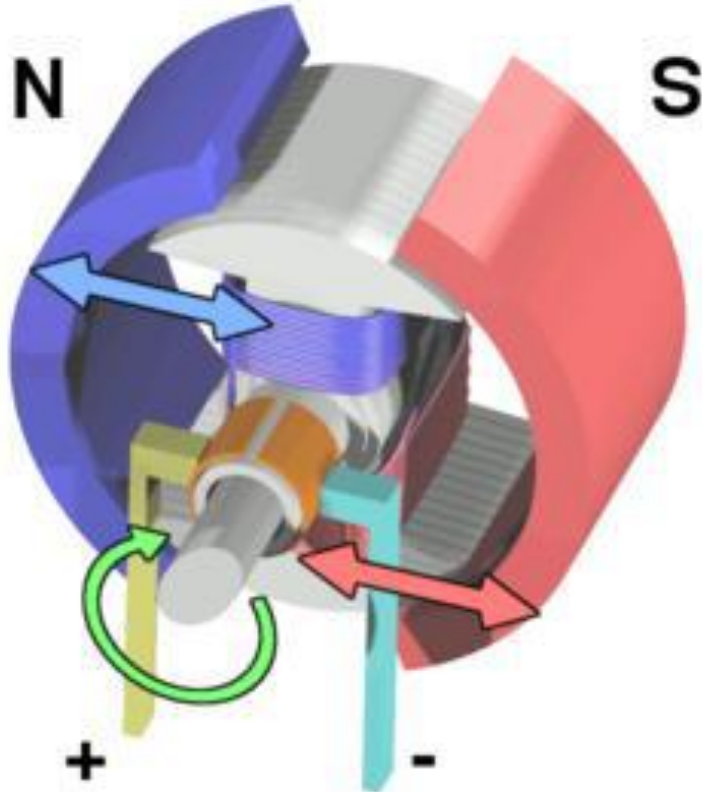
# Ejemplo de control con la librería Servo

- Ejemplo de funcionamiento con llamadas a la función write, que recibe el ángulo como parámetro de entrada
- Los giros son escalados.

```
void loop() {
  static int pos = 0;
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);                // tell servo to go to 'pos' degrees
    delay(15);                          // waits 15 ms to reach position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);                // tell servo to go to 'pos' degrees
    delay(15);                          // waits 15 ms to reach the position
  }
}
```

# Montaje del Motor DC

# DC Motor



Dos direcciones:

Clockwise: VCC to + and GND to -

Counterclockwise: VCC to - and GND to +

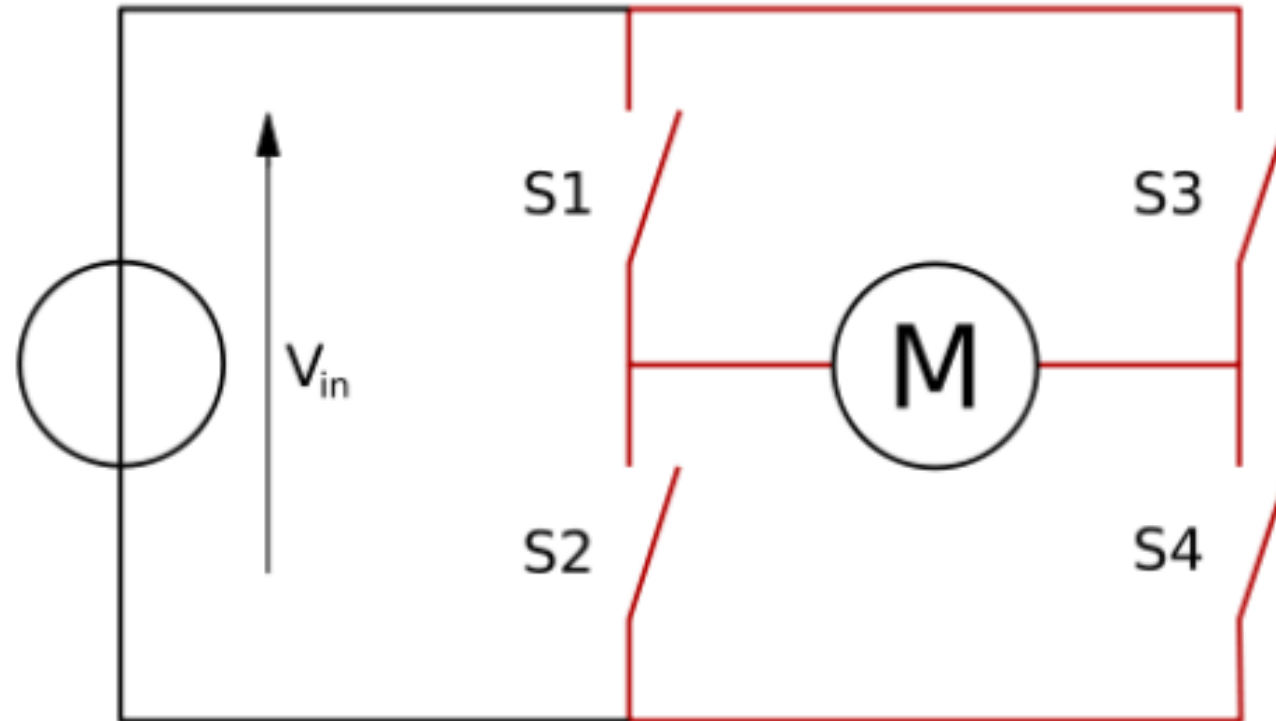
**¿Cómo invertirlas?**

Mediante un driver, como el Puente H.

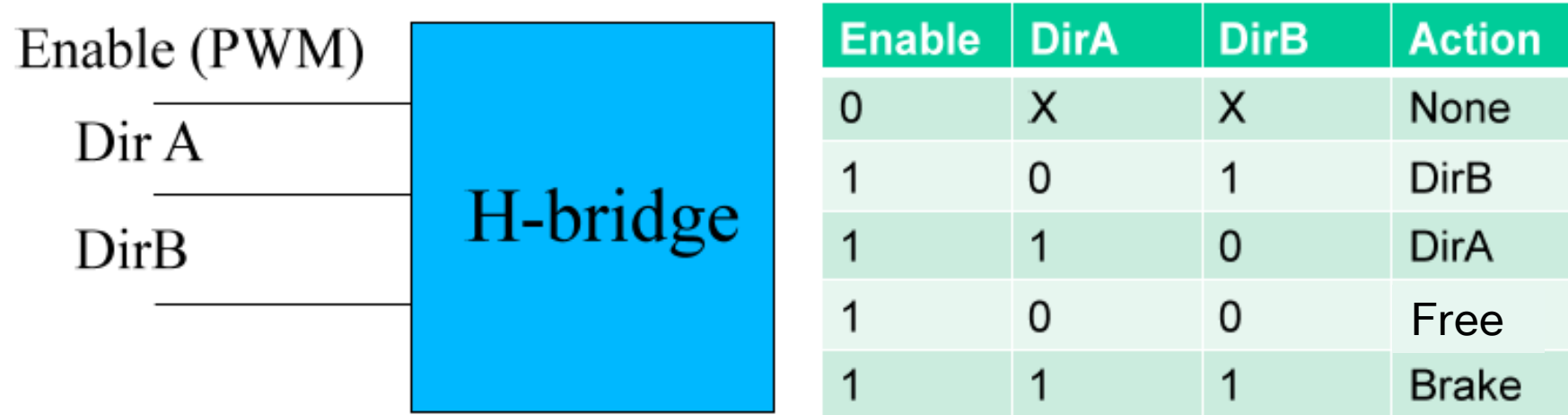
**¿Cómo determino la velocidad/posición del motor?**

Mediante un sensor, como el encoder en cuadratura.

# Puente en H (H-Bridge)



# H-Bridge

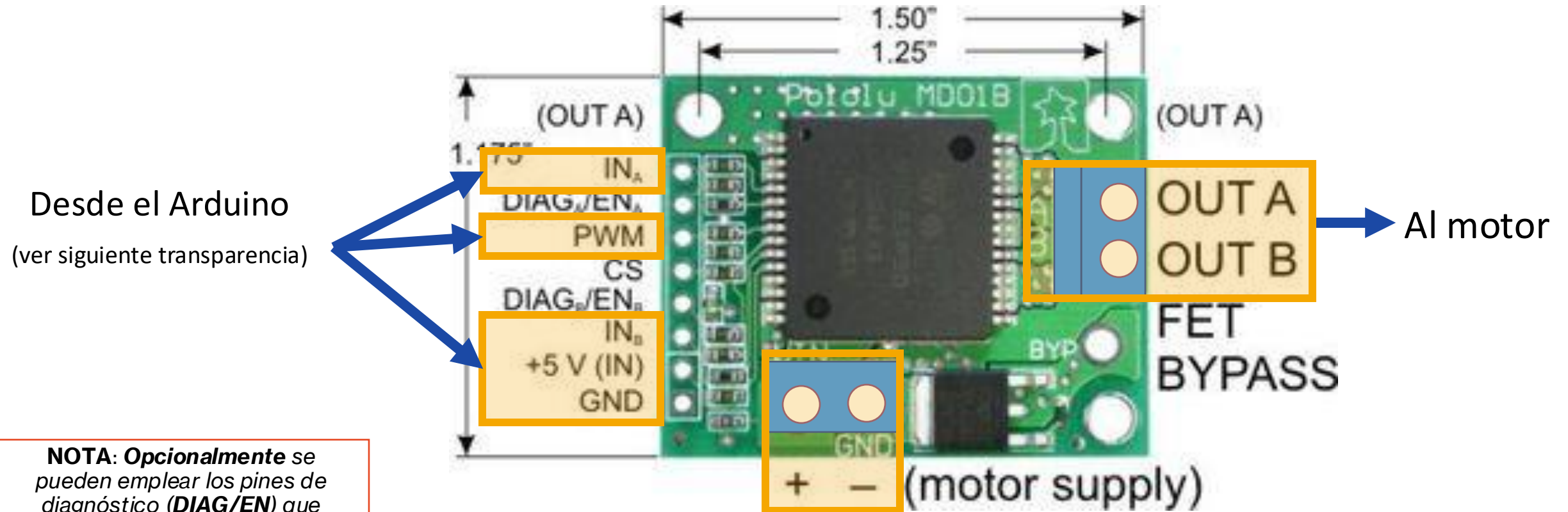


Problems with electric motors:

Non linearity. i.e. duty cycles do not match with the “corresponding” motor speeds.

Electric motors need power to start. For instance, 50% duty cycles.

# Montaje (H-Bridge Pololu)



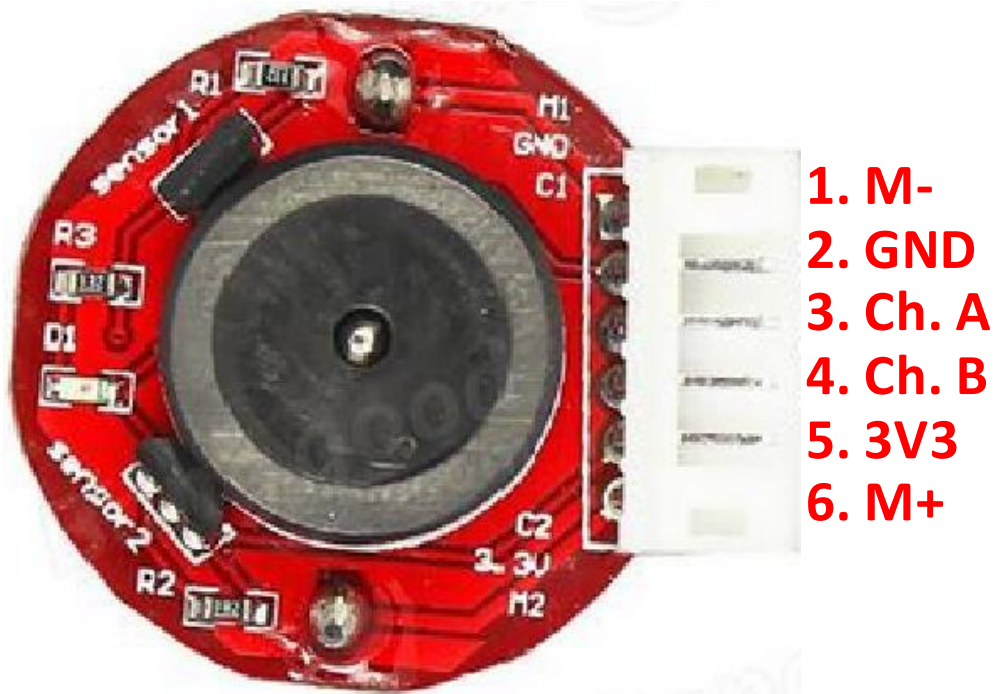
Desde el Arduino  
(ver siguiente transparencia)

**NOTA:** Opcionalmente se pueden emplear los pines de diagnóstico (**DIAG/EN**) que permiten monitorizar las condiciones de fallo del controlador del motor

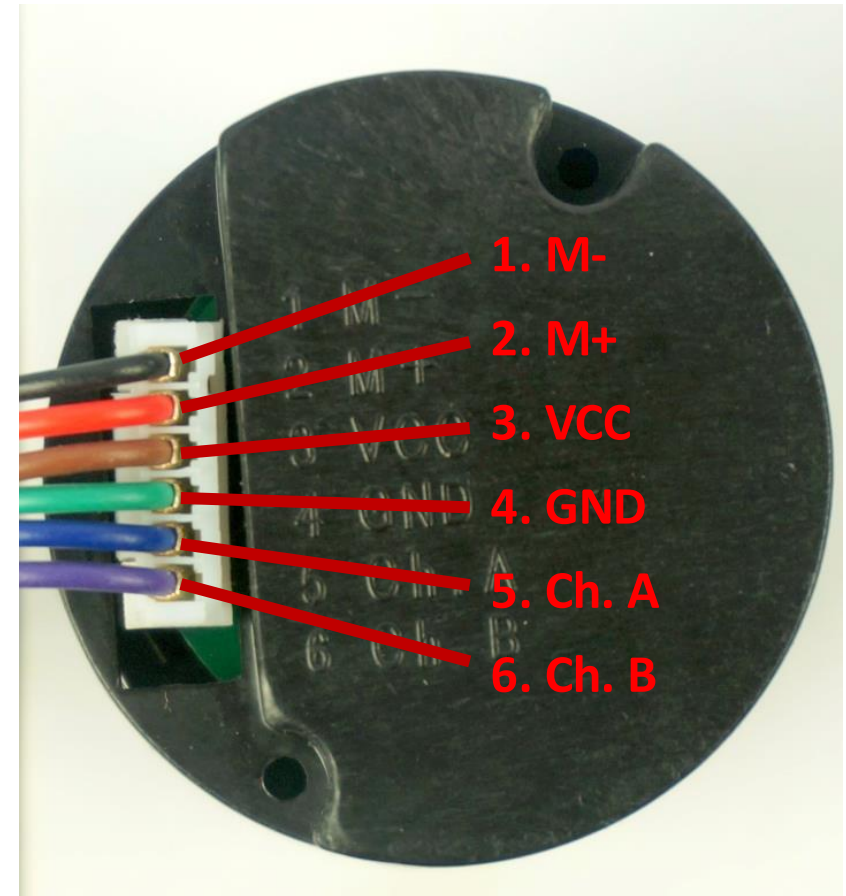
Alimentación externa, del adaptador ( $\pm 6V$ )

# Montaje (Encoder)

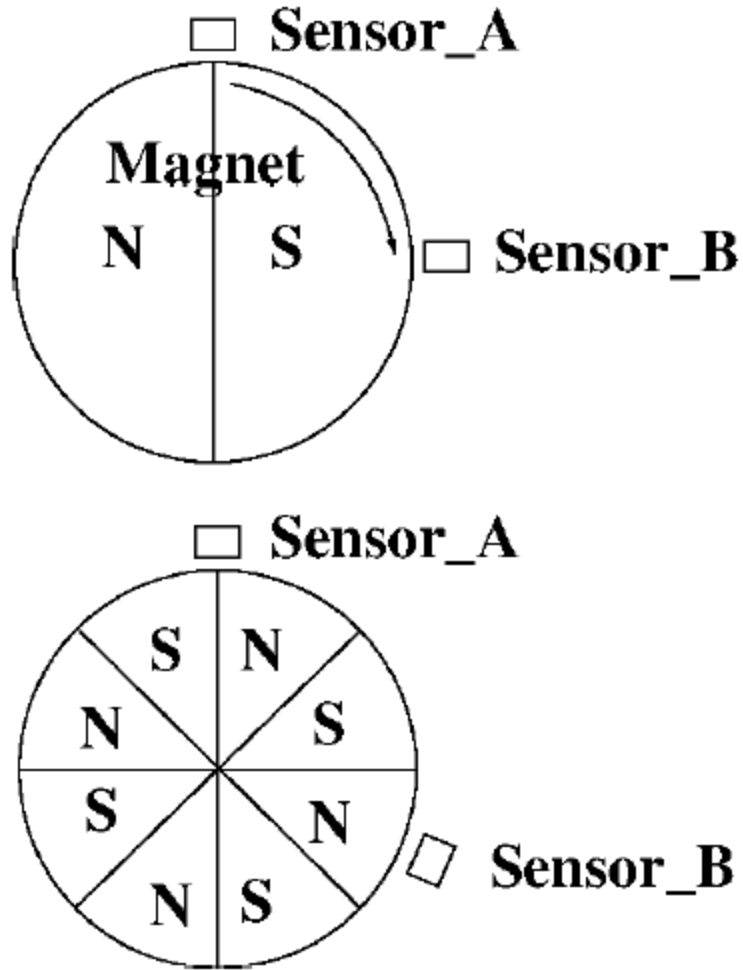
Bangood



Roboshop pins



# ENCODER



An artifact that gives pulses as a function of a rotational speed of an axis.

Motors usually have gearboxes with a coefficient of reduction in order to augmenting the resulting torque on the secondary axes.

Encoders can be attached to secondary axes.

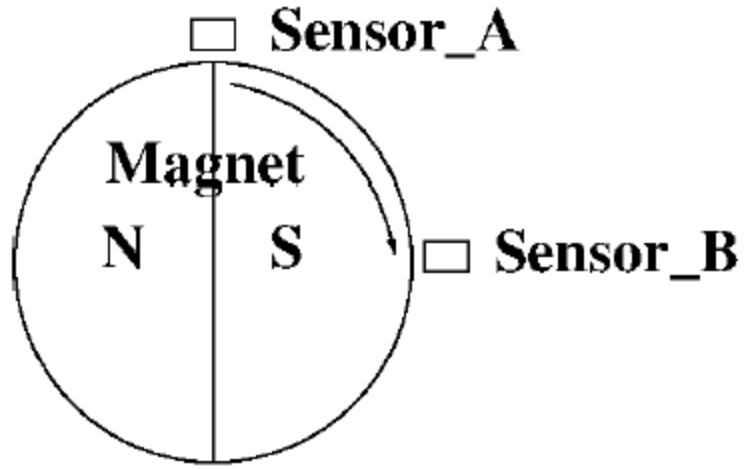
In this way the number of pulses per second is lowered:

12V, 58RPM 60: 1 gear motor with encoder.

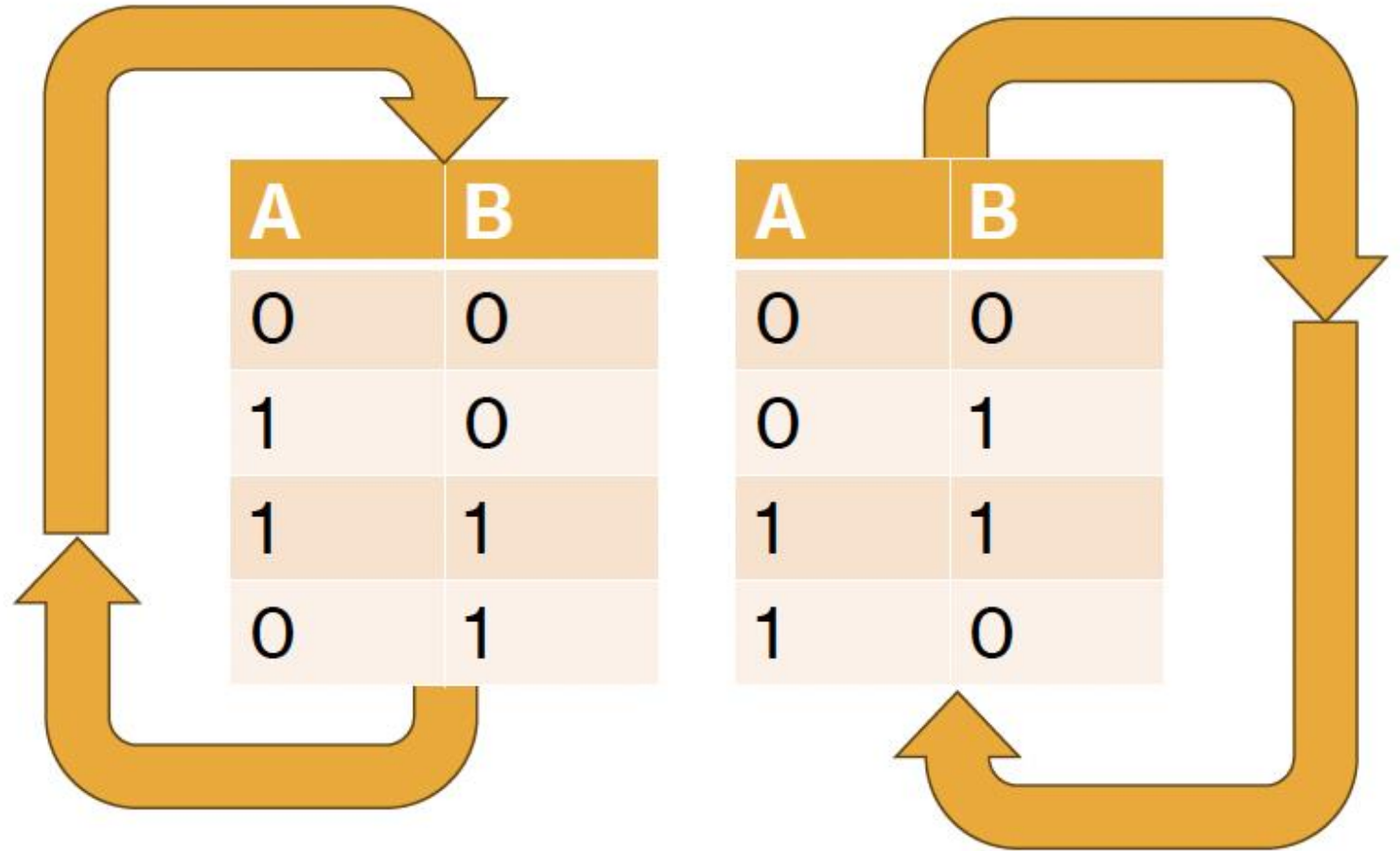
Encoders can be mechanical, optic, electronic or magnetic.



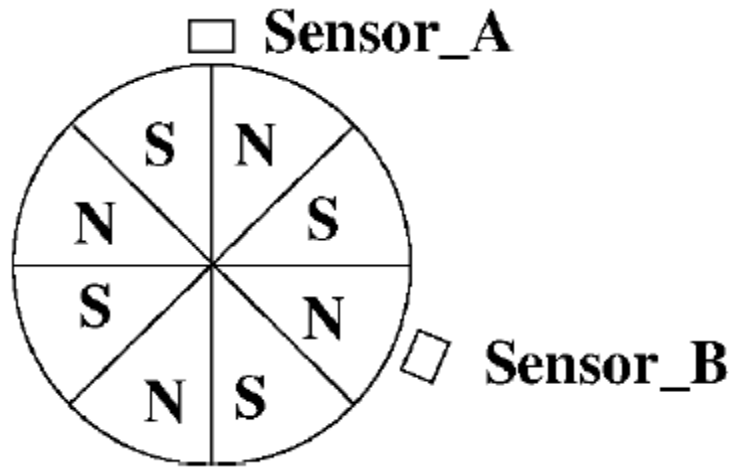
# ENCODER



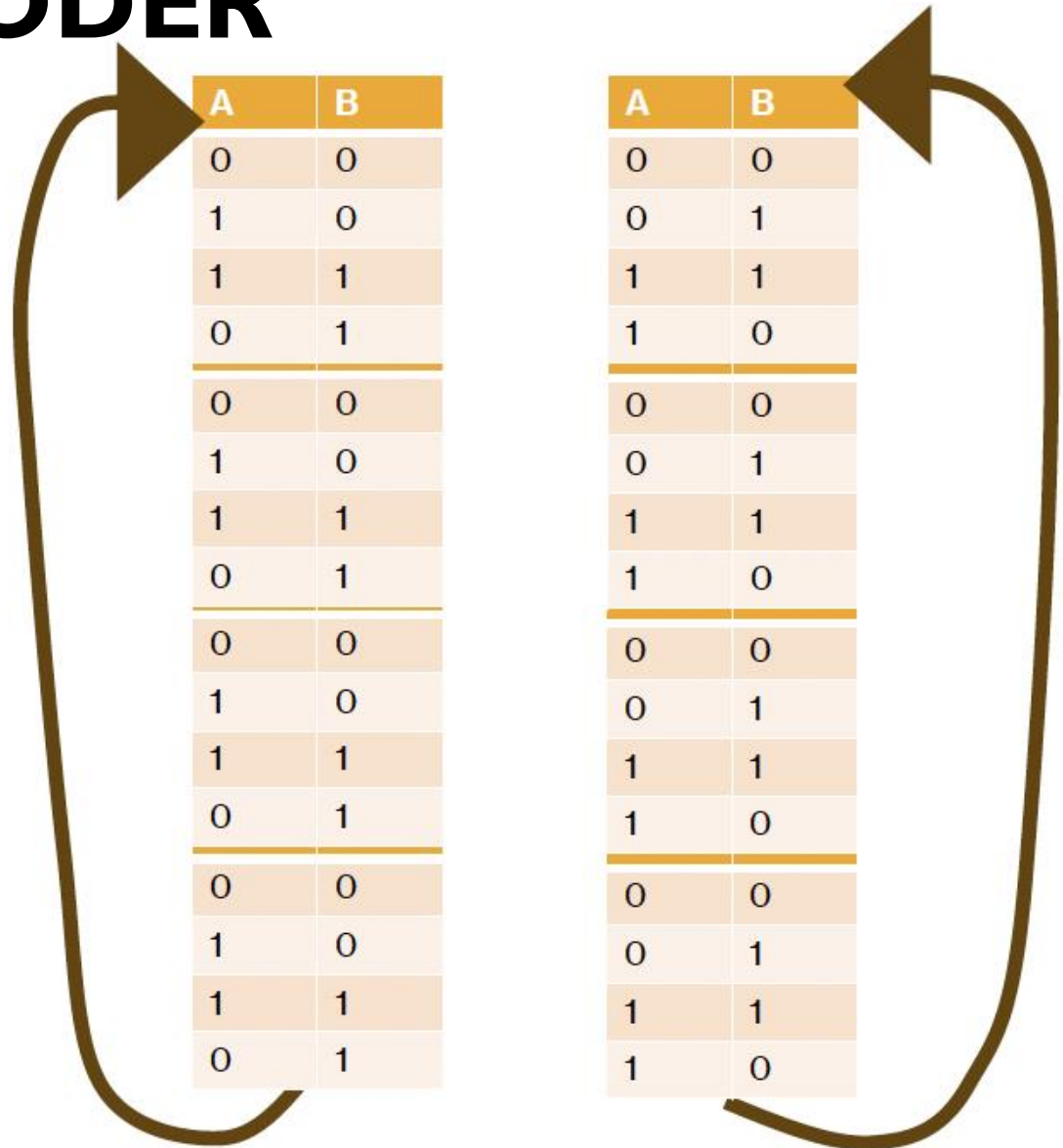
- **En ambos sentidos hay 4 cambios por revolución.**
  - La resolución para medir la posición del eje es de  $360^\circ/4 = 90$



# ENCODER



- **En ambos sentidos hay 16 cambios por revolución.**
  - Tengo más resolución para medir la posición del eje, es decir, el ángulo.
  - $360^\circ/16 = 22.5^\circ$  de resolución.



# Encoder, ejemplo mediante polling

```
volatile int pos = 0; // Variable global
ISR(...) { // Activación periódica, p.ej.: cada 100 us
    static int8_t value_old_new;
    static int8_t value_old;
    int8_t value_new = PORTD & 0x03;
    if (value_new != value_old) {
        value_old_new = value_old_new << 2;
        value_old_new = value_old_new | value_new;
        value_old = value_new; // Mantengo para Sgte. vez
        // ¿Cómo interpreto la lectura del encoder?
        // Continúa...
    } // End if
} // End ISR
```

# Encoder

Sentido de giro	Últimos 4 bits de la variable value_old_new				value_old_new en decimal
	OLD B	OLD B	NEW B	NEW A	
	0	0	0	1	1
	0	1	1	1	7
	1	1	1	0	14
	1	0	0	0	8
	0	0	1	0	2
	1	0	1	1	11
	1	1	0	1	13
	0	1	0	0	4

# Encoder, ejemplo mediante polling

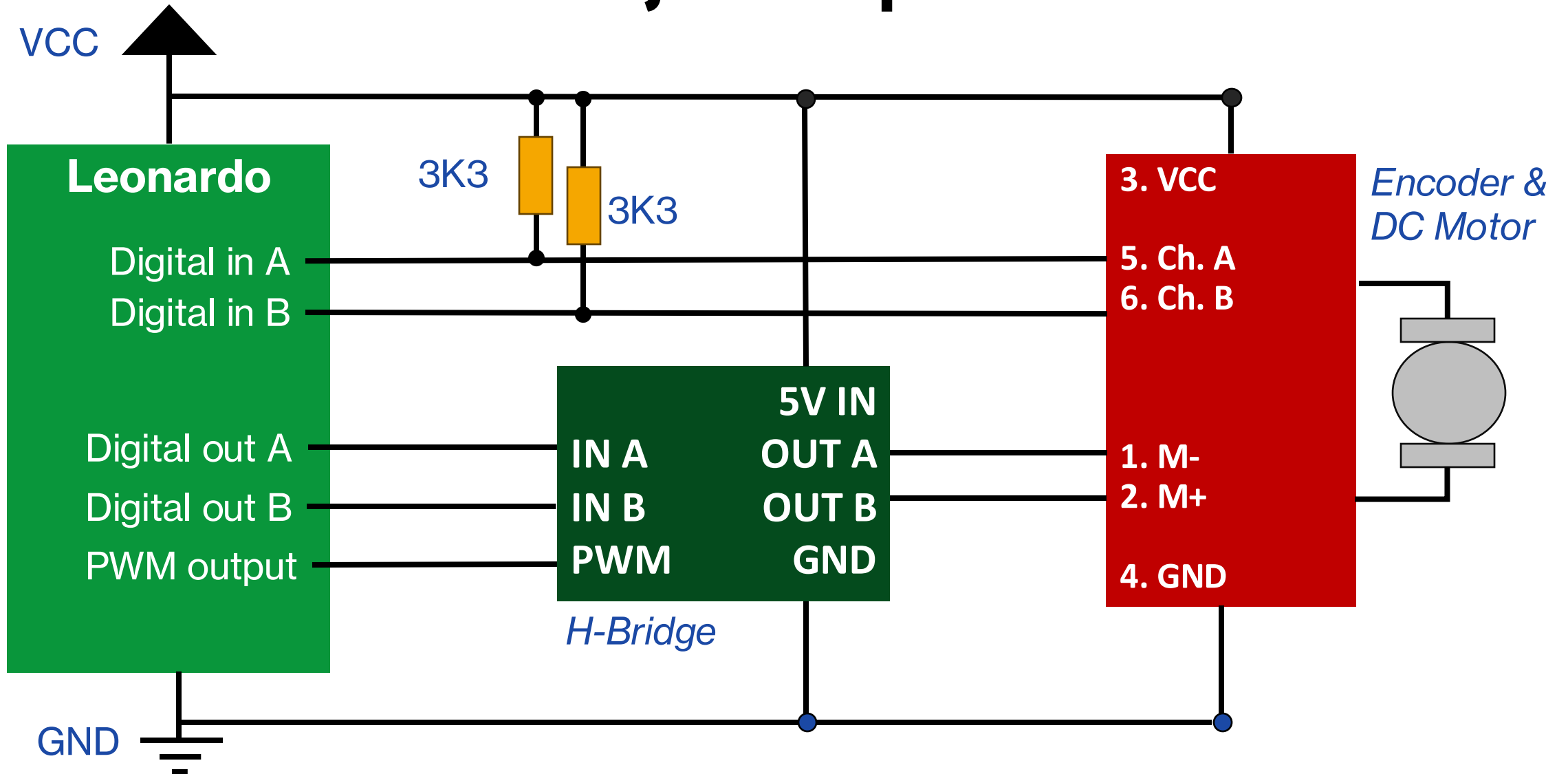
```
// ... Continuación:  
// ¿Cómo interpreto la lectura del encoder?  
if (value_old_new == 1 || value_old_new == 7 ||  
    value_old_new == 14 || value_old_new == 8)  
{  
    encoder++; // Incremento contador  
}  
if (value_old_new == 2 || value_old_new == 11 ||  
    value_old_new == 13 || value_old_new == 4)  
{  
    encoder--; // Decremento contador  
}  
} // End if  
} // End ISR
```

# Encoder, ejemplo mediante polling

```
// 1. Posición en grados. Si es un encoder de 48 PPR,  
// hay  $360^\circ/16 = 22.5^\circ$  por cada incremento del encoder.  
posicion_deg = (encoder * 22.5) % 360;
```

```
// 2. ¿Y la velocidad?  
// Es el número de pulsos leídos por unidad de tiempo.  
// Si mi ciclo de control ejecuta cada 40 ms,  
// Obtendré la variación del encoder en un tramo de 40 ms.  
ISR(...) { // Executes each 40ms  
    static int encoder_last = encoder;  
    int delta_encoder = encoder - encoder_last;  
    speed_pp40ms = delta_encoder;  
    encoder_last = encoder;  
}
```

# Montaje completo



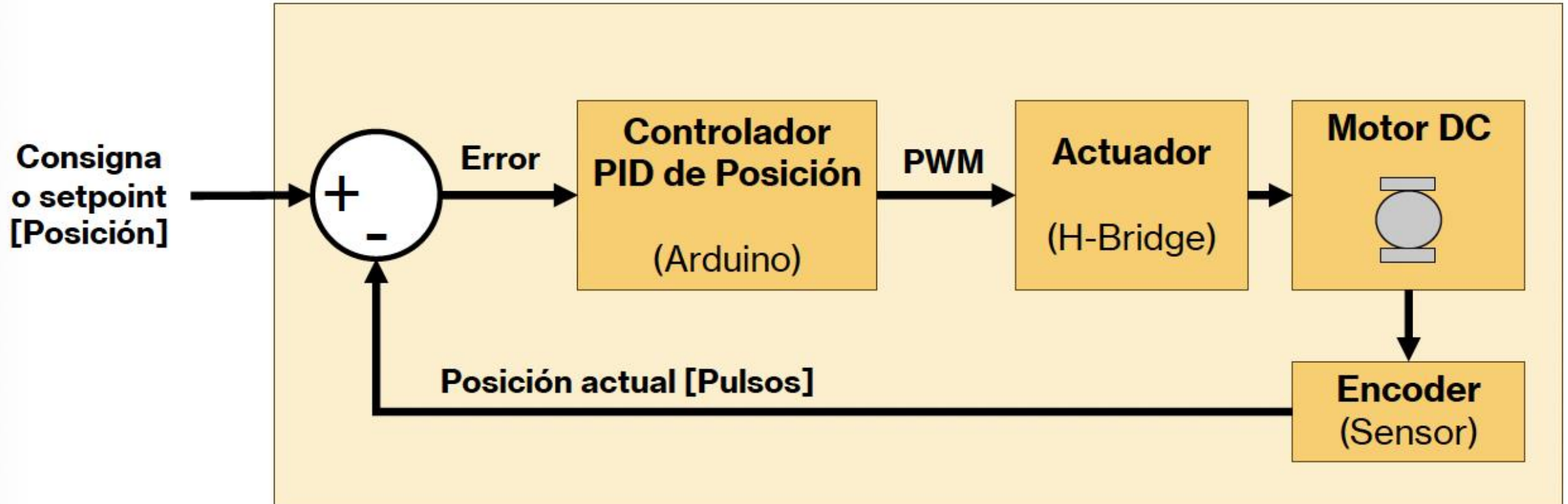
# Arduino library

- Uso de esta librería para realizar pruebas de montaje del motor.
- `AnalogWrite(int pin, int value)`
  - `pin`: Arduino digital pins with PWM support.
  - `value`: Ranges between 0 (OFF, 0% duty-cycle) and 255 (ON, 100%).
- PWM are digital pins marked with ~: 3, 5, 6, 9, 10, 11, 13
- **NOTES & WARNINGS:**
  - The PWM outputs on D5 and D6 will have higher-than-expected duty cycles because of interactions with `millis()` and `delay()` functions that share the same internal timer used to generate those PWM outputs.
  - This will be noticed mostly on low duty-cycle settings (e.g.: 0 up to 10) and may result in a value of 0, not fully turning off the output on pins D5 and D6.

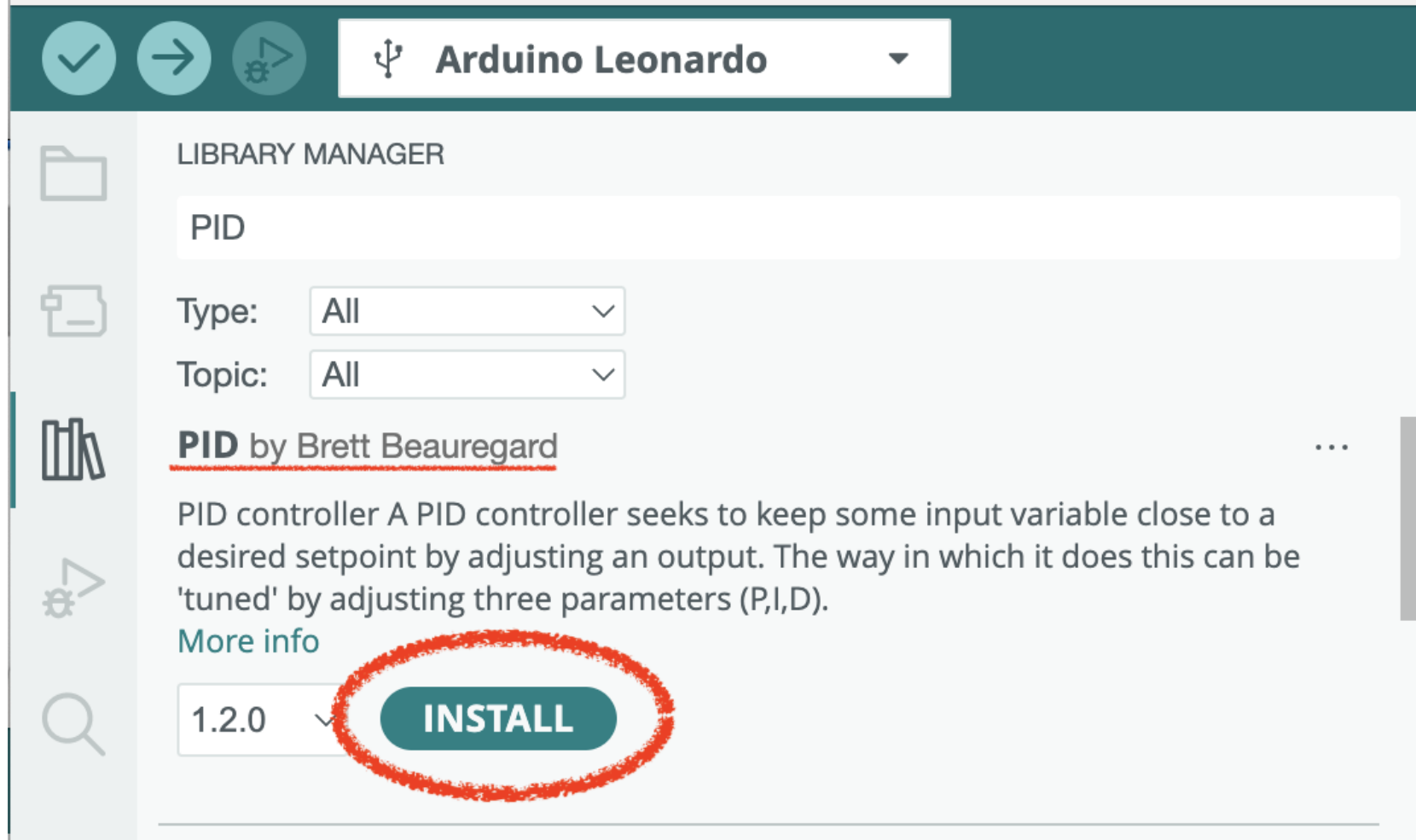


# PID Control

# Control PID de Posición



# Biblioteca PID de Brett Beauregard



The screenshot shows the Arduino IDE interface with the Library Manager open. At the top, the board is set to 'Arduino Leonardo'. The Library Manager search bar contains 'PID'. The 'Type' and 'Topic' filters are both set to 'All'. The search results show the 'PID' library by Brett Beauregard. The library description states: 'PID controller A PID controller seeks to keep some input variable close to a desired setpoint by adjusting an output. The way in which it does this can be 'tuned' by adjusting three parameters (P,I,D)'. Below the description, the version '1.2.0' is shown, and the 'INSTALL' button is highlighted with a red hand-drawn circle.

LIBRARY MANAGER

PID

Type: All

Topic: All

**PID** by Brett Beauregard

PID controller A PID controller seeks to keep some input variable close to a desired setpoint by adjusting an output. The way in which it does this can be 'tuned' by adjusting three parameters (P,I,D).

[More info](#)

1.2.0 **INSTALL**

# Biblioteca PID de Brett Beauregard

- Documentación en Español sobre la implementación de la librería (lectura recomendada):

<http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Guía-de-uso-PID-para-Arduino.pdf>

- API oficial de la librería:

<https://playground.arduino.cc/Code/PIDLibrary/>

- Constantes PID orientativas:

- $K_P = 1.1$
- $K_I = 0.50$
- $K_D = 0.08$

# Biblioteca PID de Brett Beauregard

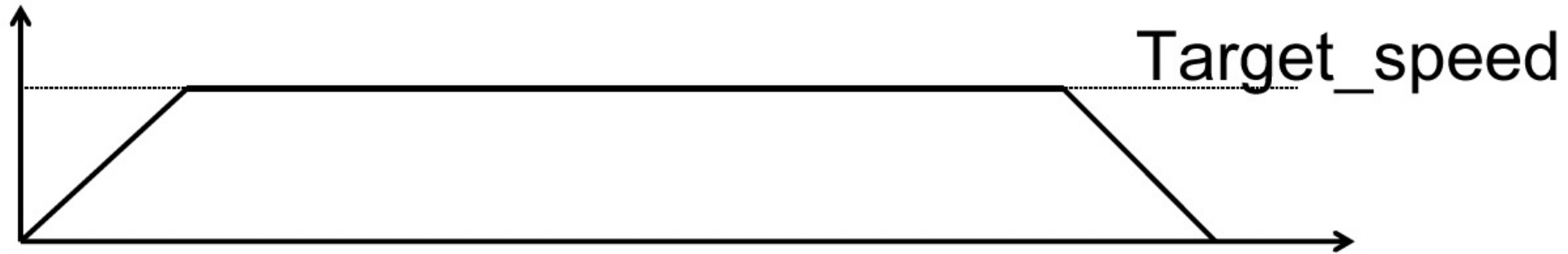
```
#include <PID_v1.h> // PID library
double Setpoint, Input, Output;
const double KP = 1.00;
const double KI = 0.35;
const double KD = 0.20;

// Creo objeto PID
PID myPID(&Input, &Output, &Setpoint,0.2,0.1,0.0, DIRECT);
void setup() {
    myPID.SetOutputLimits(0,800); // Restringir salida para el PWM.
    //...
```

# Control PID de velocidad

```
ISR(...) { // Ciclo de control cada 40 ms
    velocidad_objetivo_pp40ms = velocidad_objetivo_pps * 0.04;
    posicion_objetivo = posicion_actual +
                        velocidad_objetivo_pp40ms;
    Input = encoder; // La posicion actual
    Setpoint = posicion_objetivo - Input; // El error
    // Llama la librería que actualiza la variable Output
    myPID.Compute();
}
```

# Rampa



```
acceleration = (1..10); // pulses  
speed = 0;  
if (speed < target_speed)  
    speed = speed + acceleration/10;  
else  
    speed = target_speed;
```

# Referencias

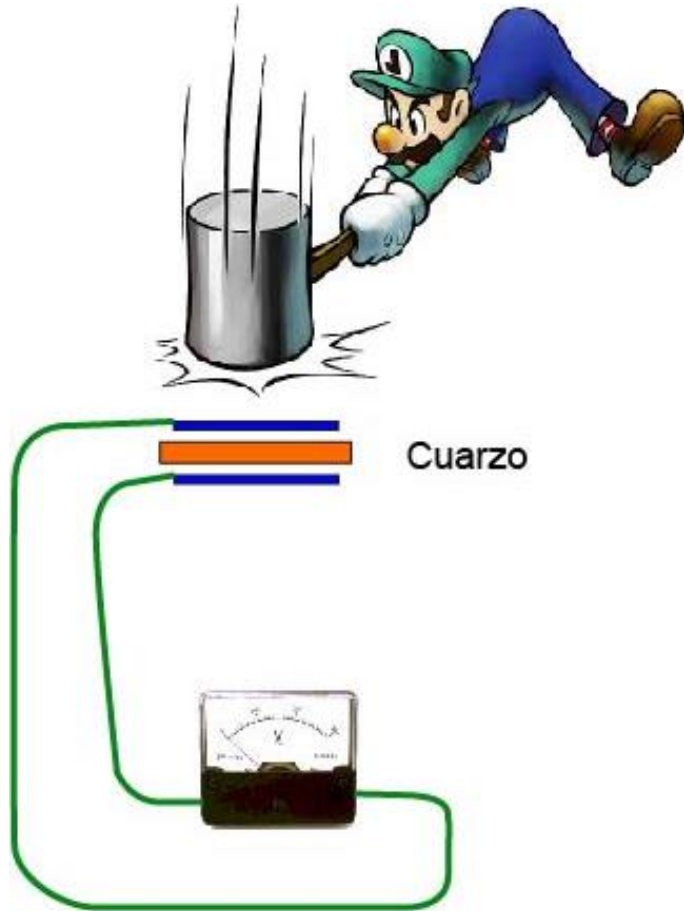


# Referencias

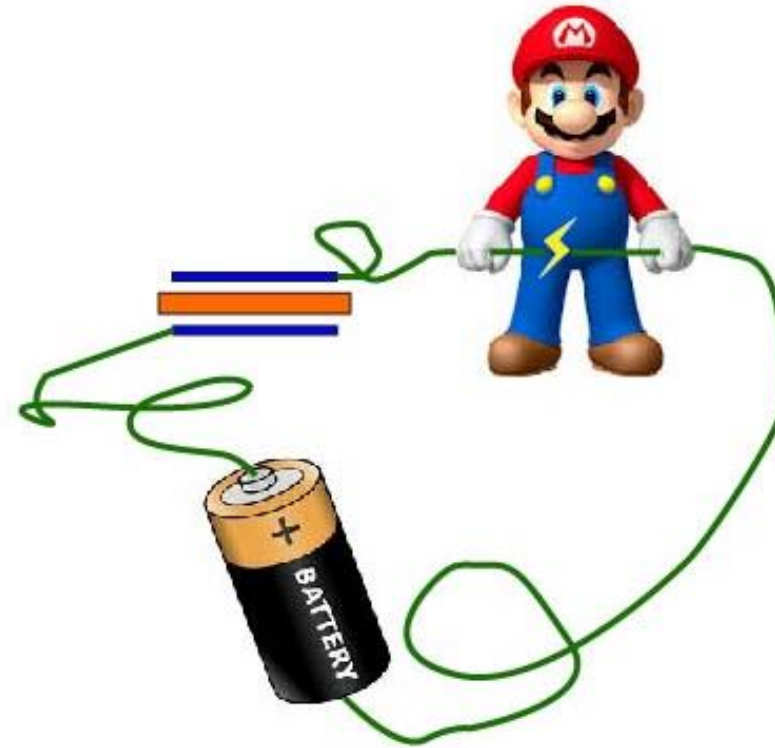
- [1] Juan Zamorano, José F. Ruiz, and Juan Antonio de la Puente. 2001. *Implementing Ada.Real\_Time.Clock and Absolute Delays in Real-Time Kernels*. In Proceedings of the 6th Ade-Europe International Conference Leuven on Reliable Software Technologies (Ada Europe '01). Springer-Verlag, Berlin, Heidelberg, 317–327.
- [2] ATMEL. ATmega16/32U4 datasheet. [https://www.datsi.fi.upm.es/docencia/Informatica\\_Industrial/DMC/pdf/atmega32u4.pdf](https://www.datsi.fi.upm.es/docencia/Informatica_Industrial/DMC/pdf/atmega32u4.pdf)
- [3] Arduino. AnalogWrite API. <https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>
- [4] Elecia White. *Making Embedded Systems, 2<sup>nd</sup> Edition*. O'Reilly Media, Inc. <https://learning.oreilly.com/library/view/making-embedded-systems/9781098151539/>
- [5] Arduino. Servo library API. <https://www.arduino.cc/reference/en/libraries/servo/>

Anexo

# A. PIEZOELECTRICIDAD

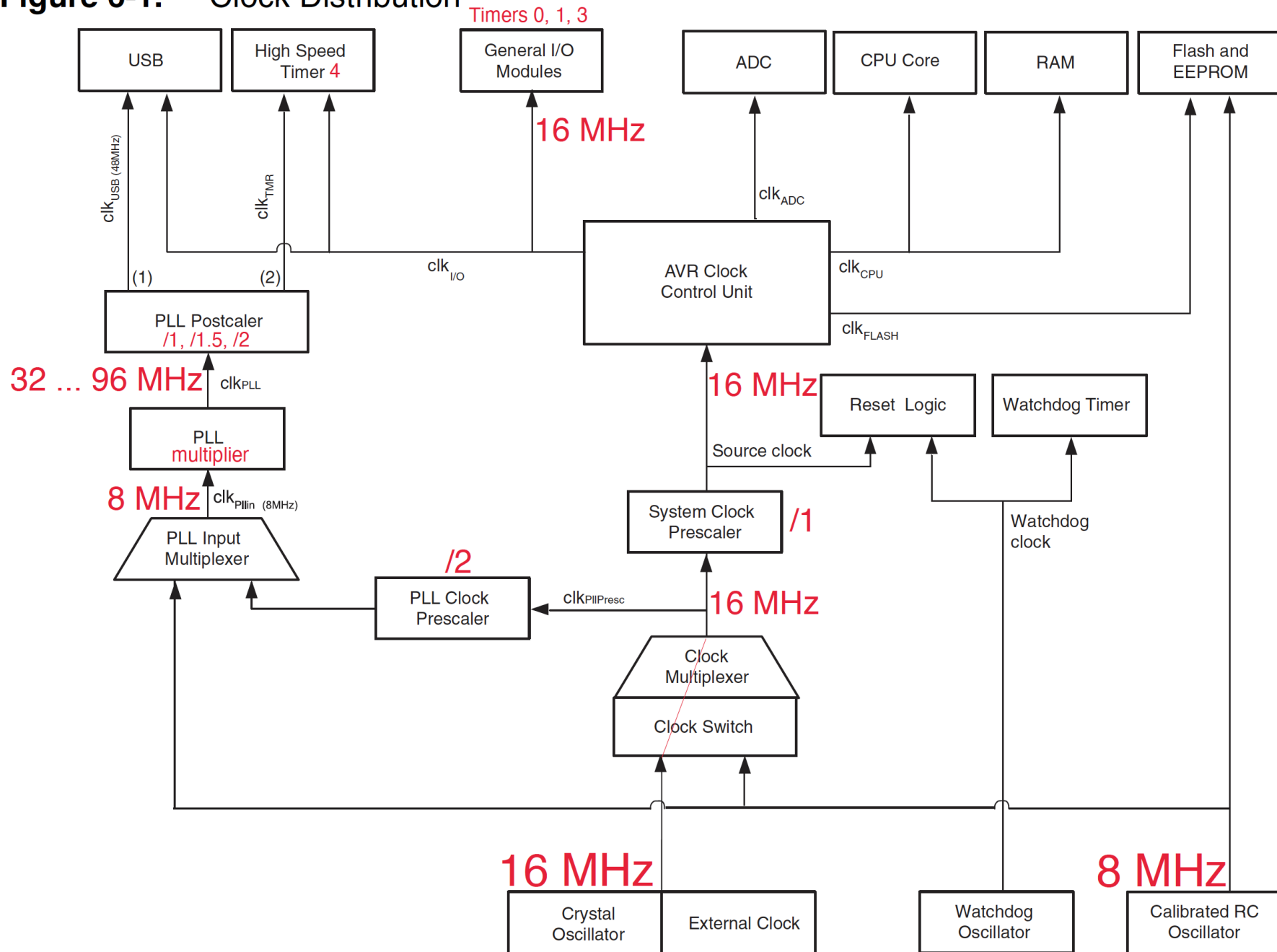


Cuando el cuarzo es sometido a compresion entonces un voltaje aparece en sus caras

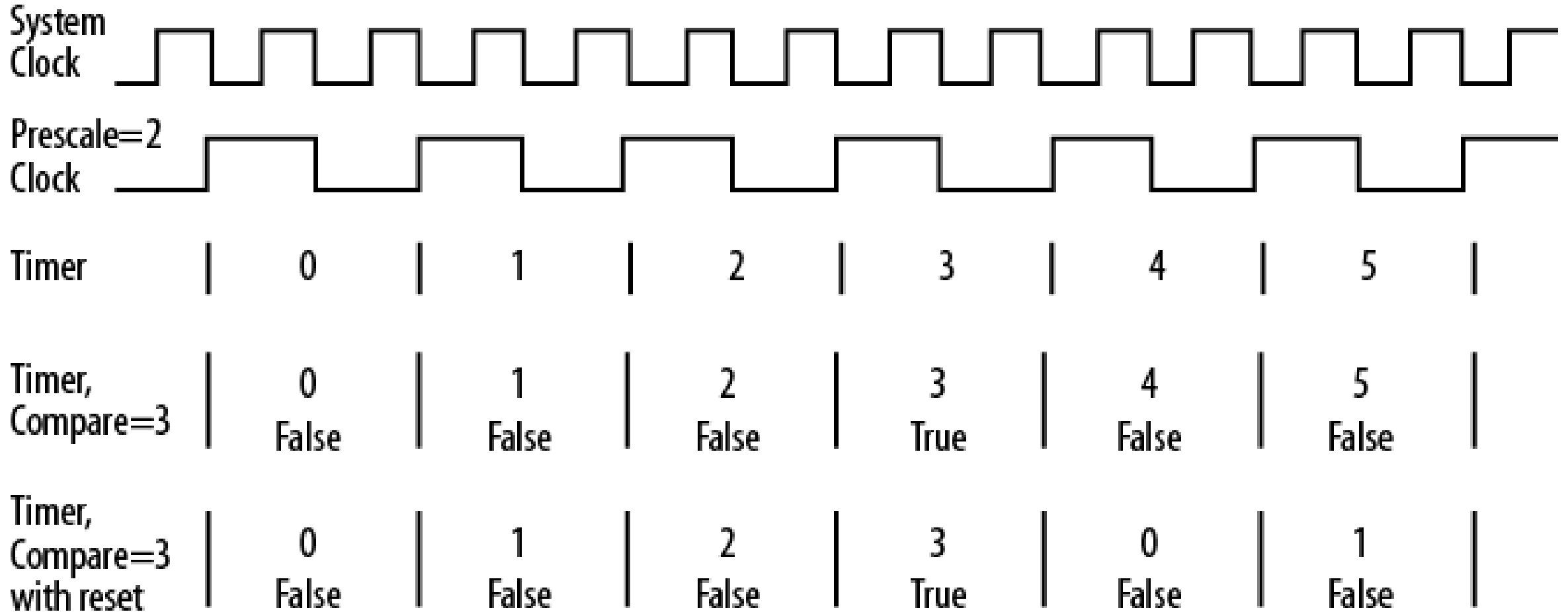


Cuando se conecta un voltaje entre las caras del Cuarzo entonces el empieza a vibrar.

**Figure 6-1. Clock Distribution**



# B. Diagrama temporal de un timer [4]



# B. Registros típicos de un Timer/Counter [4]

## *Timer counter*

This holds the changing value of the timer (the number of ticks since the timer was last reset).

## *Compare register (or capture compare register or match register)*

When the timer counter equals this register, an action is taken. There may be more than one compare register for each timer.

## *Action register (or auto reload register)*

This register sets up an action to take when the timer and compare register are the same. (For some timers, these actions are also available when the timer overflows, which is like having a compare register set to the maximum value of the timer counter.) There are four types of possible actions to be configured (one or more may happen):

- Interrupt
- Stop or continue counting
- Reload the counter
- Set an output pin to high, low, toggle, or no change

# B. Registros típicos de un Timer/Counter [4]

## *Clock configure register (optional)*

This register tells a subsystem which clock source to use, though the default may be the system clock. Some processors have timers that even allow a clock to be attached to an input pin. You can often choose whether to count up or down, I'll be using count-up with these examples but the process is similar.

## *Prescaler register*

As shown in [Figure 4-7](#), this reduces the fast incoming clock so that it runs more slowly, allowing timers to happen for slower events.

## *Control register*

This sets the timer to start counting once it has been configured. Often the control register also has a way to reset the timer.

## *Interrupt register (may be multiple)*

If you have timer interrupts, you will need to use the appropriate interrupt register to enable, clear, and check the status of each timer interrupt.

# C. Interrupt Vector Table (IVT)

Vector No	Program Address	Source	Interrupt Definition	Arduino/C++ ISR() Macro Vector Name
1	0x0000	RESET	Reset	
2	0x0002	INT0	External Interrupt Request 0 (pin D0)	(INT0_vect)
3	0x0004	INT1	External Interrupt Request 1 (pin D1)	(INT1_vect)
4	0x0006	INT2	External Interrupt Request 2 (pin D2)	(INT2_vect)
5	0x0008	INT3	External Interrupt Request 3 (pin D3)	(INT3_vect)
6	0x000A	Reserved	Reserved	
7	0x000C	Reserved	Reserved	
8	0x000E	INT6	External Interrupt Request 6 (pin E6)	(INT6_vect)
9	0x0010	Reserved		
10	0x0012	PCINT0	Pin Change Interrupt Request 0 (pins PB7 to PB0)	(PCINT0_vect)
11	0x0014	USB General	USB General Interrupt request	(USB_GENERAL_vect)
12	0x0016	USB Endpoint	USB Endpoint Interrupt request	(USB_ENDPOINT_vect)
13	0x0018	WDT	Watchdog Time-out Interrupt	(WDT_vect)
14	0x001A	Reserved	Reserved	
15	0x001C	Reserved	Reserved	
16	0x001E	Reserved	Reserved	



# C. Interrupt Vector Table (IVT)

17	0x0020	TIMER1 CAPT	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
18	0x0022	TIMER1 COMPA	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
19	0x0024	TIMER1 COMPB	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
20	0x0026	TIMER1 COMPC	Timer/Counter1 Compare Match C	(TIMER1_COMPC_vect)
21	0x0028	TIMER1 OVF	Timer/Counter1 Overflow (see note)	(TIMER1_OVF_vect)
22	0x002A	TIMERO COMPA	Timer/Counter0 Compare Match A	(TIMERO_COMPA_vect)
23	0x002C	TIMERO COMPB	Timer/Counter0 Compare Match B	(TIMERO_COMPB_vect)
24	0x002E	<b>TIMERO OVF</b>	<b>Timer/Counter0 Overflow</b>	<b>(TIMERO_OVF_vect)</b>
25	0x0030	SPI, STC	SPI Serial Transfer Complete	(SPI_STC_vect)
26	0x0032	USART, RX	USART Rx Complete	(USART_RX_vect)
27	0x0034	USART, UDRE	USART, Data Register Empty	(USART_UDRE_vect)
28	0x0036	USART, TX	USART, Tx Complete	(USART_TX_vect)

# C. Interrupt Vector Table (IVT)

29	0x0038	ANALOG COMP	Analog Comparator	(ANALOG_COMP_vect)
30	0x003A	ADC	ADC Conversion Complete	(ADC_vect)
31	0x003C	EE READY	EEPROM Ready	(EE_READY_vect)
32	0x003E	TIMER3 CAPT	Timer/Counter3 Capture Event	(TIMER3_CAPT_vect)
33	0x0040	TIMER3 COMPA	Timer/Counter3 Compare Match A	(TIMER3_COMPA_vect)
34	0x0042	TIMER3 COMPB	Timer/Counter3 Compare Match B	(TIMER3_COMPB_vect)
35	0x0044	TIMER3 COMPC	Timer/Counter3 Compare Match C	(TIMER3_COMPC_vect)
36	0x0046	TIMER3 OVF	Timer/Counter3 Overflow	(TIMER3_OVF_vect)
37	0x0048	TWI	2-wire Serial Interface (I2C)	(TWI_vect)
38	0x004A	SPM READY	Store Program Memory Ready	(SPM_READY_vect)
39	0x004C	TIMER4 COMPA	Timer/Counter4 Compare Match A	(TIMER4_COMPA_vect)
40	0x004E	TIMER4 COMPB	Timer/Counter4 Compare Match B	(TIMER4_COMPB_vect)
41	0x0050	TIMER4 COMPD	Timer/Counter4 Compare Match D	(TIMER4_COMPD_vect)
42	0x0052	TIMER4 OVF	Timer/Counter4 Overflow	(TIMER4_OVF_vect)
43	0x0054	TIMER4 FPF	Timer/Counter4 Fault Protection Interrupt	(TIMER4_FPF_vect)

# D. Programación de RTIs en C

- Macros para habilitar e inhibir interrupciones:
  - #define sei()
  - #define cli()
- Para escribir una RTI, se debe emplear la macro `ISR`, Por ejemplo:

```
// Primer parámetro: nombre del vector de la IVT
ISR(TIMER1_OVF_vect) {
    // Código del programador...
}
```

# E. LCD Keypad Shield (16x2)

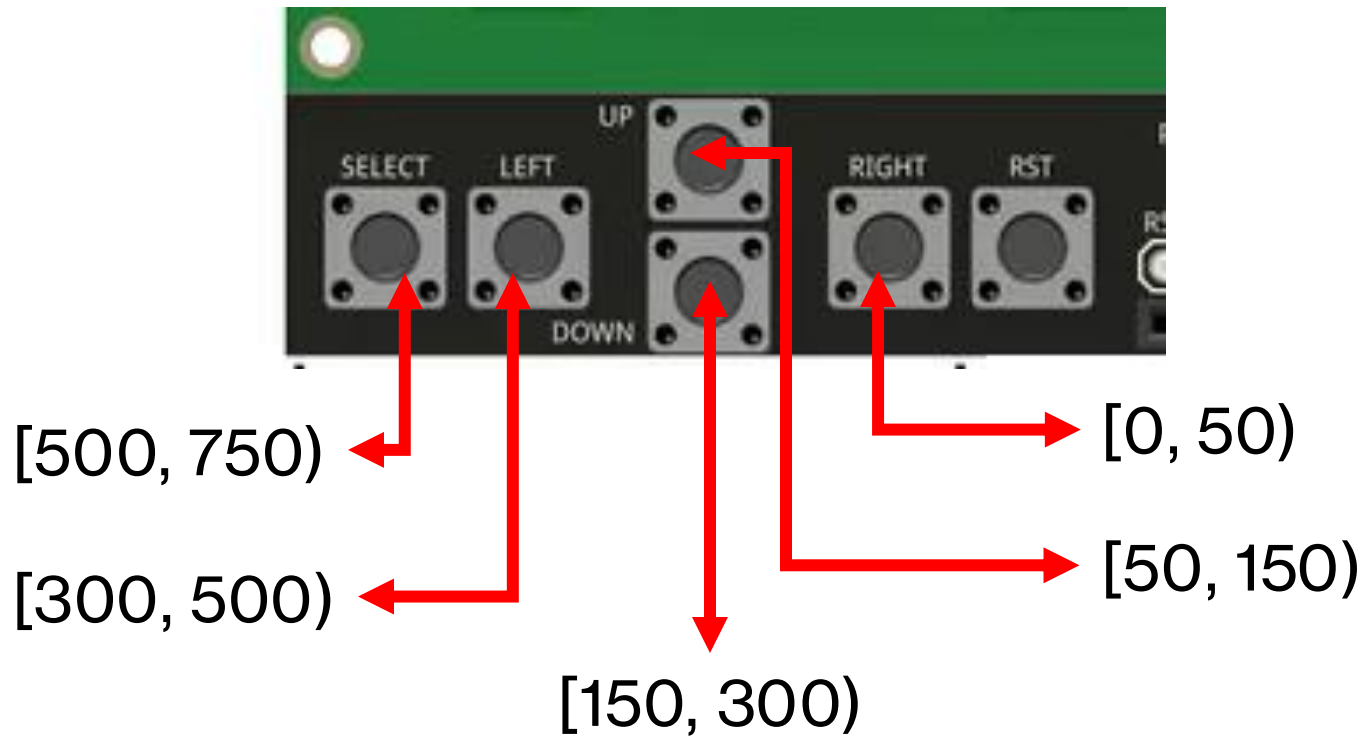
Potenciómetro (control de contraste)



Todos los botones están conectados al pin A0

LCD Pin	Arduino Pin
Reset (RS)	DB8
Enable (E)	DB9
D4	DB4
D5	DB5
D6	DB6
D7	DB7
Botones	A0

# E. LCD Keypad Shield (16x2)



Botón pulsado	Rango de lectura en el ADC
RIGHT	[0, 50)
UP	[50, 100)
DOWN	[150, 300)
LEFT	[300, 500)
SELECT	[500, 750)