

# Periféricos

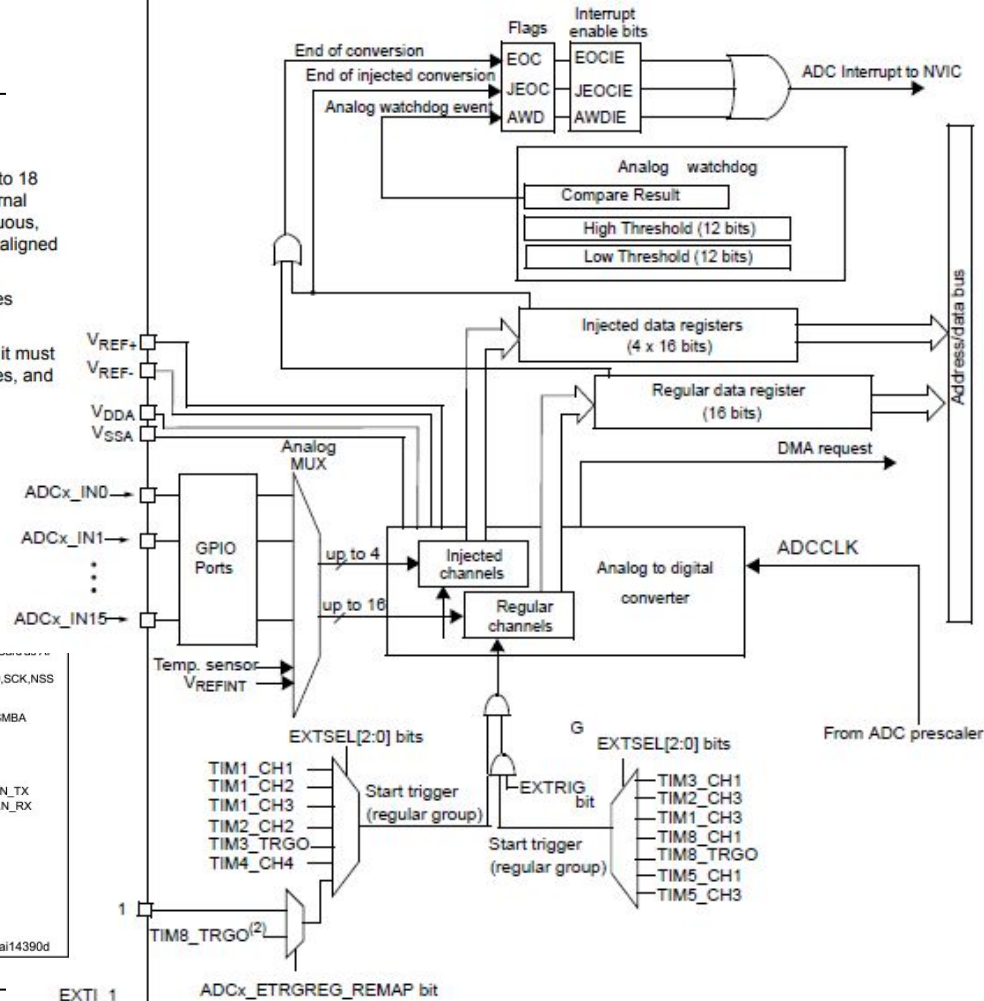
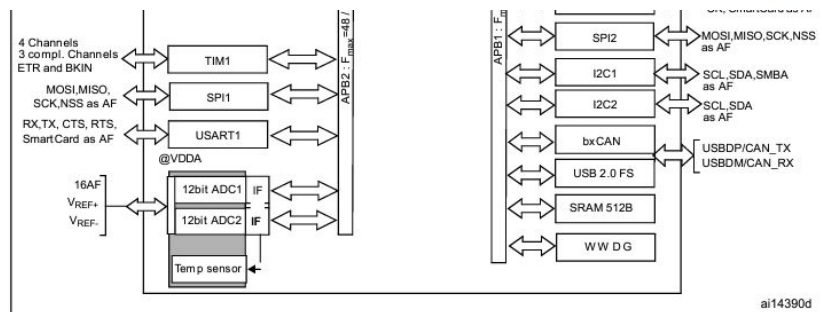
ADC

## 11.1 ADC introduction

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 18 multiplexed channels allowing it measure signals from sixteen external and two internal sources. A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left-aligned or right-aligned 16-bit data register.

The analog watchdog feature allows the application to detect if the input voltage goes outside the user-defined high or low thresholds.

The ADC input clock is generated from the PCLK2 clock divided by a prescaler and it must not exceed 14 MHz, refer to [Figure 8](#) for low-, medium-, high- and XL-density devices, and to [Figure 11](#) for connectivity line devices.



### 11.3.1 **ADC on-off control**

The ADC can be powered-on by setting the ADON bit in the ADC\_CR2 register. When the ADON bit is set for the first time, it wakes up the ADC from Power Down mode.

Conversion starts when ADON bit is set for a second time by software after ADC power-up time ( $t_{STAB}$ ).

The conversion can be stopped, and the ADC put in power down mode by resetting the ADON bit. In this mode the ADC consumes almost no power (only a few  $\mu A$ ).

### 11.3.2 **ADC clock**

The ADCCLK clock provided by the Clock Controller is synchronous with the PCLK2 (APB2 clock). The RCC controller has a dedicated programmable prescaler for the ADC clock,

### 11.3.4 Single conversion mode

In Single conversion mode the ADC does one conversion. This mode is started either by setting the ADON bit in the ADC\_CR2 register (for a regular channel only) or by external trigger (for a regular or injected channel), while the CONT bit is 0.

Once the conversion of the selected channel is complete:

- If a regular channel was converted:
  - The converted data is stored in the 16-bit ADC\_DR register
  - The EOC (End Of Conversion) flag is set
  - and an interrupt is generated if the EOCIE is set.

### 11.3.5 Continuous conversion mode

In continuous conversion mode ADC starts another conversion as soon as it finishes one. This mode is started either by external trigger or by setting the ADON bit in the ADC\_CR2 register, while the CONT bit is 1.

After each conversion:

- If a regular channel was converted:
  - The converted data is stored in the 16-bit ADC\_DR register
  - The EOC (End Of Conversion) flag is set
  - An interrupt is generated if the EOCIE is set.

## 11.7 Conversion on external trigger

Conversion can be triggered by an external event (e.g. timer capture, EXTI line). If the EXT-TRIG control bit is set then external events are able to trigger a conversion. The EXTSEL[2:0] and JEXTSEL[2:0] control bits allow the application to select decide which out of 8 possible events can trigger conversion for the regular and injected groups.

*Note: When an external trigger is selected for ADC regular or injected conversion, only the rising edge of the signal can start the conversion.*

**Table 67. External trigger for regular channels for ADC1 and ADC2**

Source	Type	EXTSEL[2:0]
TIM1_CC1 event	Internal signal from on-chip timers	000
TIM1_CC2 event		001
TIM1_CC3 event		010
TIM2_CC2 event		011
TIM3_TRGO event		100
TIM4_CC4 event		101
EXTI line 11 / TIM8_TRGO event <sup>(1)(2)</sup>	External pin / Internal signal from on-chip timers	110
SWSTART	Software control bit	111

**Figure 27. Right alignment of data****Figure 28. Left alignment of data**

## 11.6 Channel-by-channel programmable sample time

ADC samples the input voltage for a number of ADC\_CLK cycles which can be modified using the SMP[2:0] bits in the ADC\_SMPR1 and ADC\_SMPR2 registers. Each channel can be sampled with a different sample time.

The total conversion time is calculated as follows:

$$T_{\text{conv}} = \text{Sampling time} + 12.5 \text{ cycles}$$

Example:

With an ADCCLK = 14 MHz and a sampling time of 1.5 cycles:

$$T_{\text{conv}} = 1.5 + 12.5 = 14 \text{ cycles} = 1 \mu\text{s}$$

Bits 23:0 **SMPx[2:0]**: Channel x Sample time selection

These bits are written by software to select the sample time individually for each channel. During sample cycles channel selection bits must remain unchanged.

000: 1.5 cycles

001: 7.5 cycles

010: 13.5 cycles

011: 28.5 cycles

100: 41.5 cycles

101: 55.5 cycles

110: 71.5 cycles

111: 239.5 cycles



Equation 1:  $R_{AIN}$  max formula:

$$R_{AIN} < \frac{T_s}{f_{ADC} \times C_{ADC} \times \ln(2^{N+2})} - R_{ADC}$$

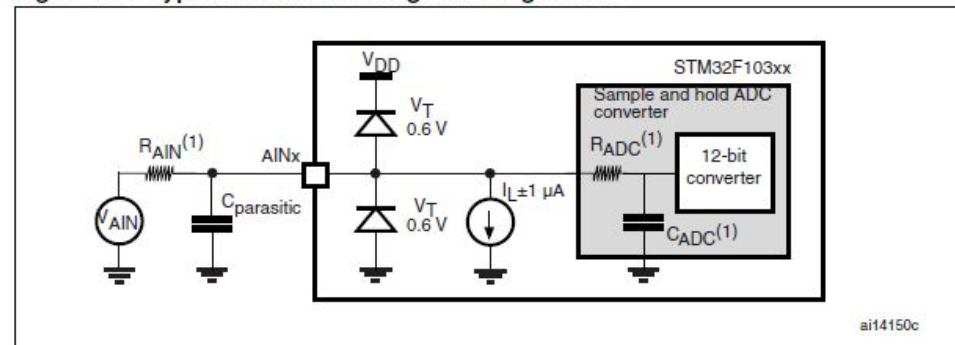
The formula above ([Equation 1](#)) is used to determine the maximum external impedance allowed for an error below 1/4 of LSB. Here N = 12 (from 12-bit resolution).

Table 46.  $R_{AIN}$  max for  $f_{ADC} = 14 \text{ MHz}^{(1)}$

T <sub>s</sub> (cycles)	t <sub>s</sub> (μs)	R <sub>AIN</sub> max (kΩ)
1.5	0.11	0.4
7.5	0.54	5.9
13.5	0.96	11.4
28.5	2.04	25.2
41.5	2.96	37.2
55.5	3.96	50

$R_{AIN}^{(2)}$	External input impedance	50	k $\Omega$
$R_{ADC}^{(2)}$	Sampling switch resistance	1	k $\Omega$
$C_{ADC}^{(2)}$	Internal sample and hold capacitor	8	pF

**Figure 37. Typical connection diagram using the ADC**

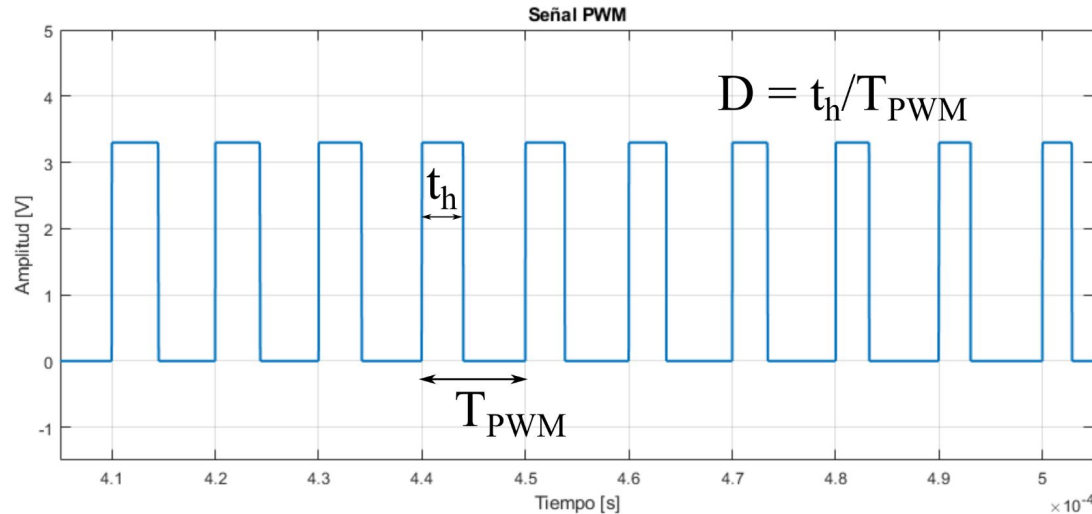


1. Refer to [Table 45](#) for the values of  $R_{AIN}$ ,  $R_{ADC}$  and  $C_{ADC}$ .
2.  $C_{parasitic}$  represents the capacitance of the PCB (dependent on soldering and PCB layout quality) plus the pad capacitance (roughly 7 pF). A high  $C_{parasitic}$  value will downgrade conversion accuracy. To remedy this,  $f_{ADC}$  should be reduced.

PWM

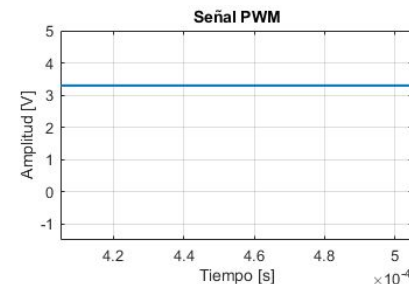
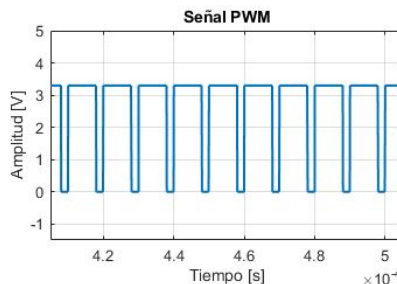
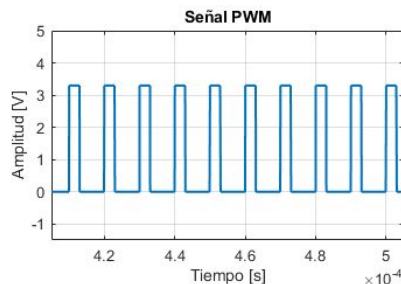
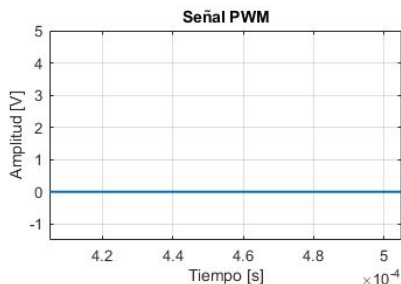
# Modulación por ancho de pulso

- El PWM permite codificar la información en el tiempo en lugar de la amplitud o de un código binario particular.
- El ciclo de trabajo, duty cycle,  $D$  codifica el valor.

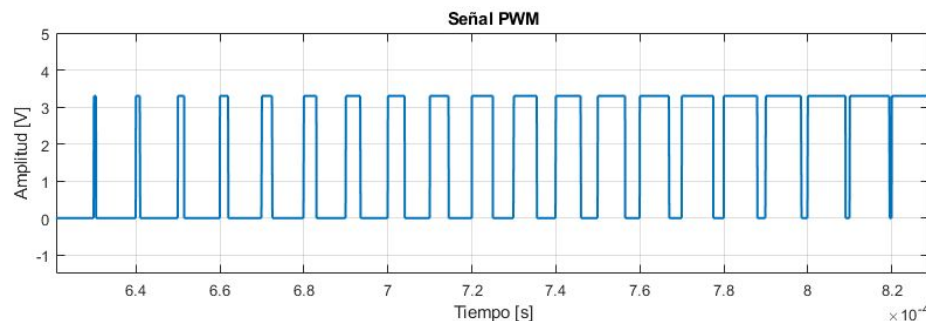


# Ancho de pulso o “duty cycle”

- Ejemplos para 0%, 30%, 80%, 100%

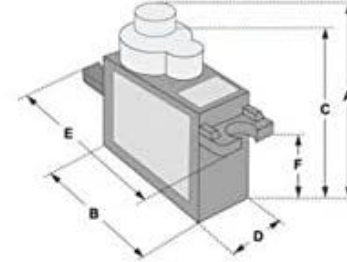


- Rampa con incrementos de 5%



# Usos del PWM

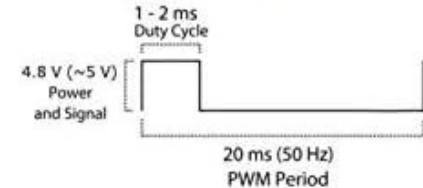
- Un uso es la codificación directa del valor en tiempo
  - e.g. servomecanismos reciben un valor de 0 a 255 y lo mapean en un ángulo de 0 a 180°
- Otro uso es variar la potencia promedio entregada a un motor, luz, etc
- El otro uso es un DAC de bajo costo
  - Una forma intuitiva de pensarlo es que el promedio de la señal de PWM en un período devuelve el valor codificado.
  - El PWM produce una modulación no lineal de la señal codificada. Si está acotada en frecuencia, puede recuperarse con un filtro pasa bajos.
  - Cuanta más alta la frecuencia del PWM más fácil será recuperar la onda.



Position "0°" (1.5 ms pulse) is middle, "90°" (~2ms pulse) is middle, is all the way to the right, "-90°" (~1ms pulse) is all the way to the left.

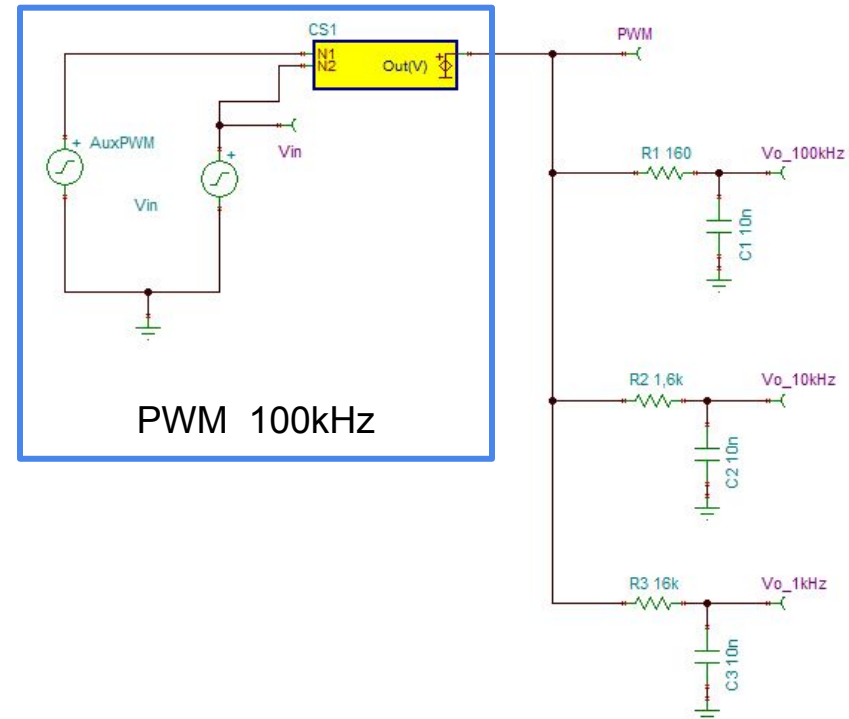
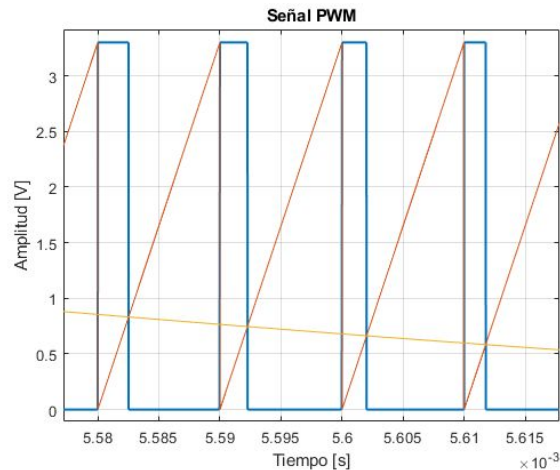
Dimensions & Specifications	
A (mm) :	32
B (mm) :	23
C (mm) :	28.5
D (mm) :	12
E (mm) :	32
F (mm) :	19.5
Speed (sec) :	0.1
Torque (kg-cm) :	2.5
Weight (g) :	14.7
Voltage :	4.8 - 6

PWM=Orange (⏏)  
Vcc=Red (+)  
Ground=Brown (-)

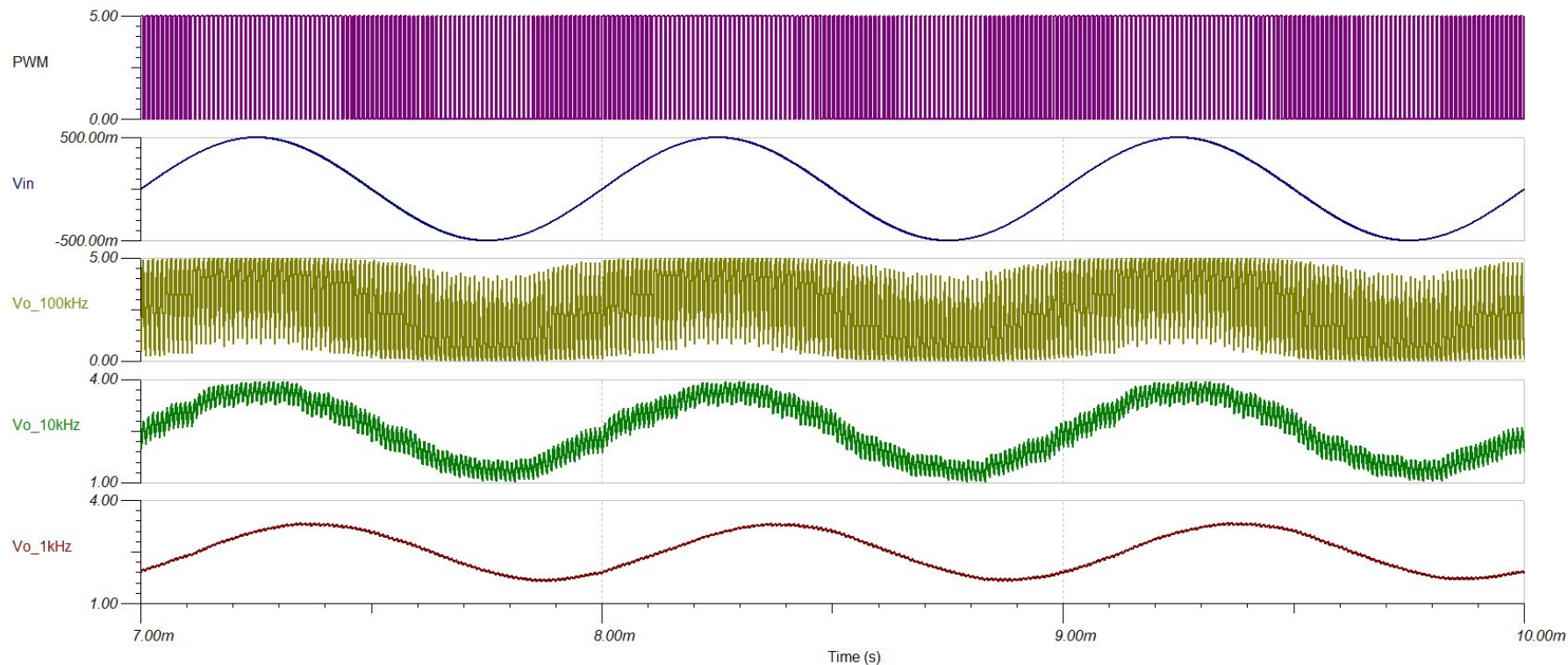


# Ejemplo simulado de generación y recuperación de señal

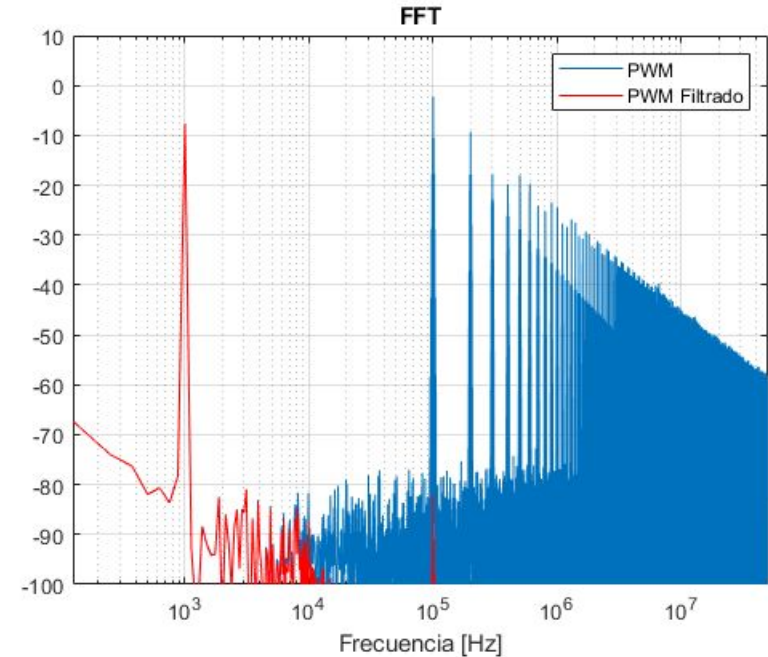
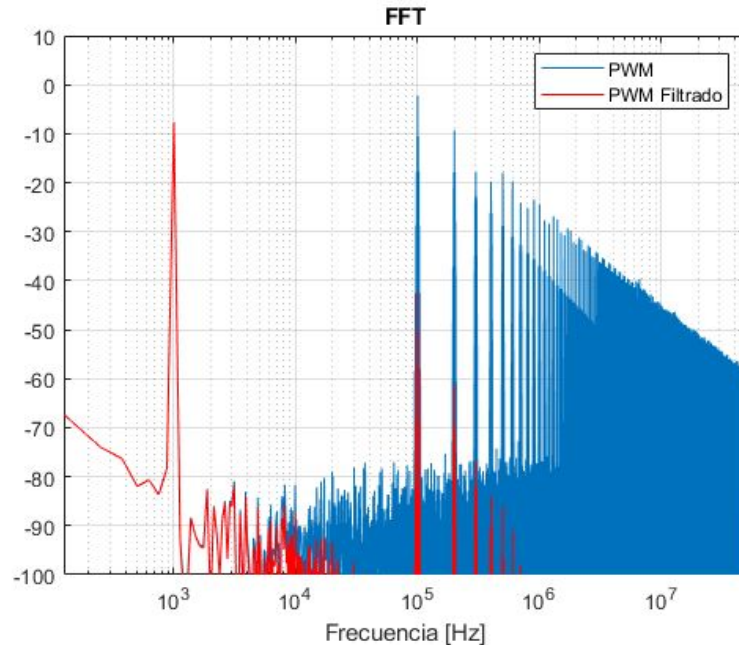
- Simulación en Spice Tina de TI
- Generamos el PWM con un diente de sierra DS y un seno SEN
- Siempre que  $SEN > DS$  la señal será alta, si no, será 0



# Recuperación de señal



# Efecto del filtrado en onda sinusoidal



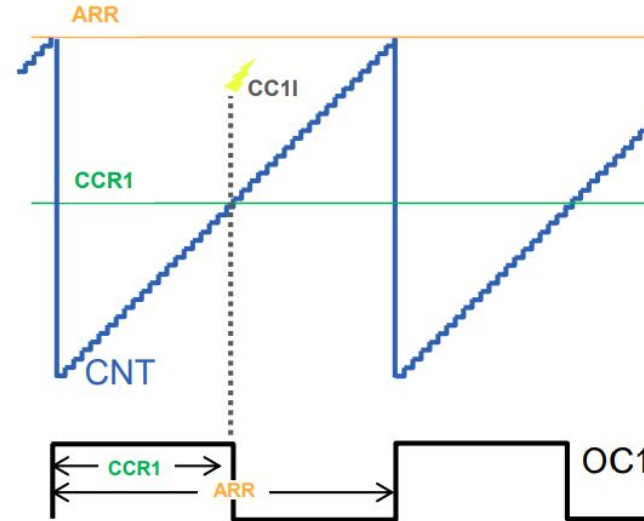
PWM 100 kHz, señal codificada seno 1 kHz

Filtro pasa bajos butterworth  $f_c = 10$  kHz, 1 polo vs 2 polos



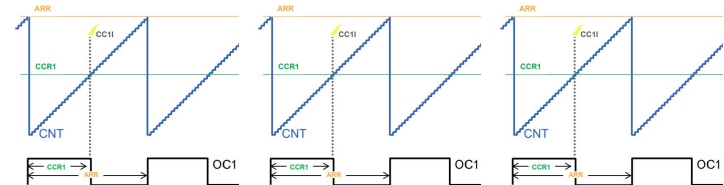
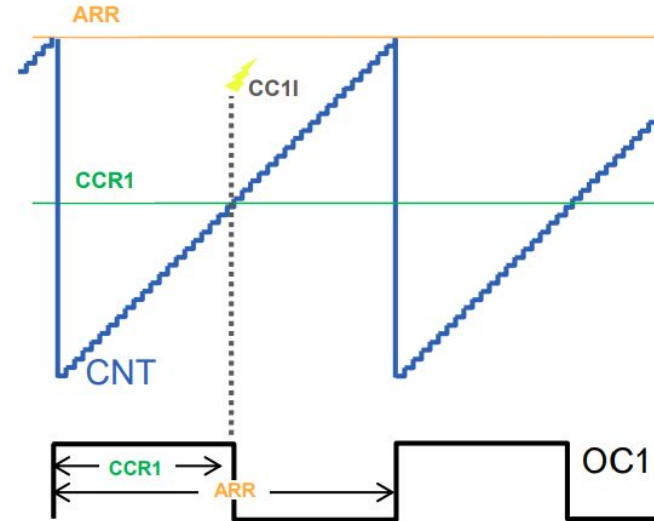
# Implementación del PWM en microcontroladores

- El PWM suele estar asociado a los timers y es un modo de operación de los mismos
- Supongamos un timer que cuenta ascendentemente en su registro CNT desde 0 y se resetea cada vez que alcanza un valor ARR
- Es esencialmente una onda diente de sierra
- Se produce una salida positiva siempre que un valor  $CCR1 > CNT$  y 0 en el caso contrario
- Si actualizamos el valor de CCR1 ciclo a ciclo estaremos codificando las muestras de una señal



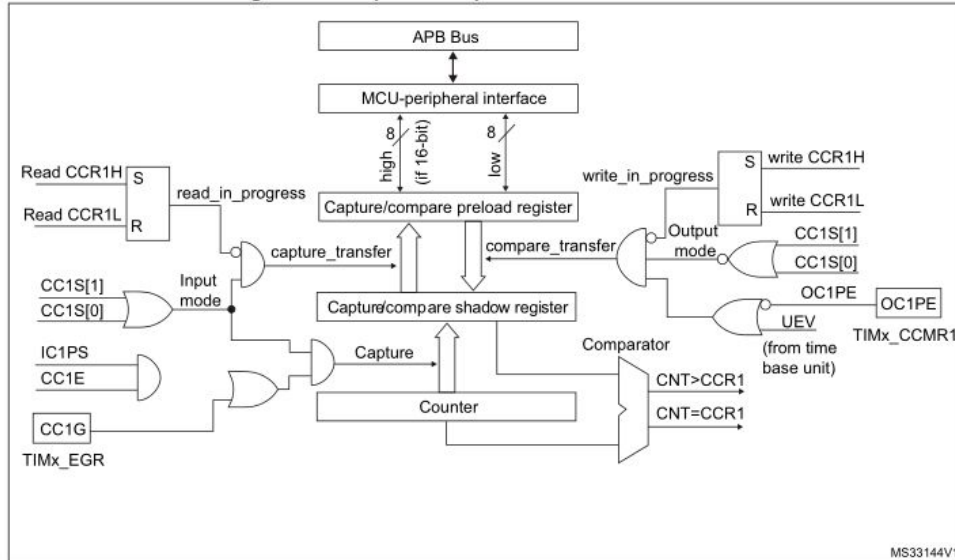
## Preguntas:

- ¿Cuál será la resolución de nuestro “DAC” basado en el PWM?
- ¿Qué relación habrá entre la frecuencia del timer y la resolución del PWM? (para hacerlo más fácil, pensemos que queremos lograr la máxima frecuencia posible por “muestra”)

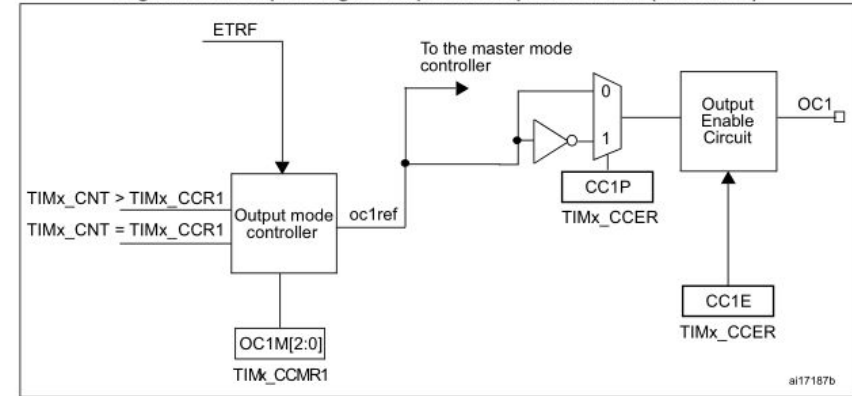


# PWM en el STM32F103C8

**Figure 126. Capture/compare channel 1 main circuit**



**Figure 127. Output stage of capture/compare channel (channel 1)**

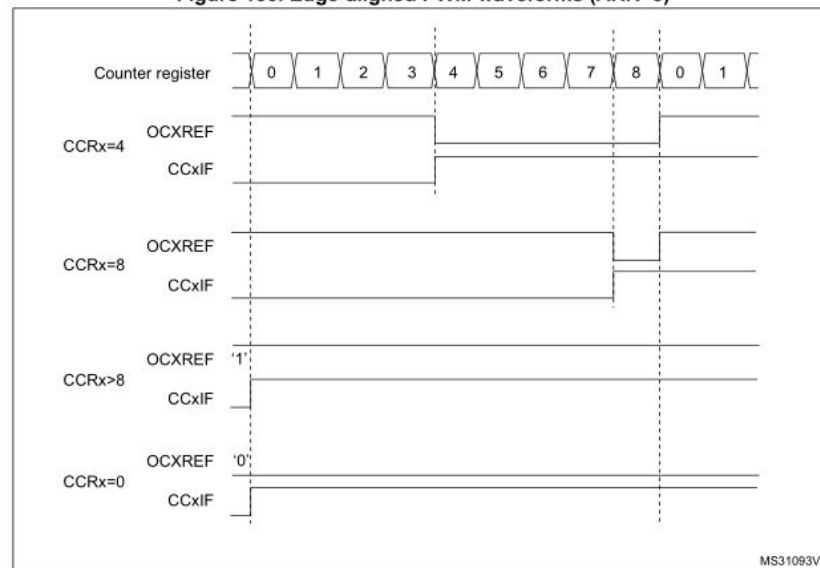


# PWM en el STM32F103C8

## 15.3.9 PWM mode

Pulse width modulation mode allows generating a signal with a frequency determined by the value of the TIMx\_ARR register and a duty cycle determined by the value of the TIMx\_CCRx register.

Figure 130. Edge-aligned PWM waveforms (ARR=8)



# Ejemplo de PWM en el STM32F103C8

## Configuración en Cube MX

### TIM2 Mode and Configuration

#### Mode

Slave Mode	Disable	▼
Trigger Source	Disable	▼
Clock Source	Internal Clock	▼
Channel1	PWM Generation CH1	▼
Channel2	Disable	▼
Channel3	Disable	▼

Parameter Settings

User Constants

Configure the below parameters :

Search (Ctrl+F)

◀ ▶

i

Counter Settings

Prescaler (PSC - 16 bits v...

71

Counter Mode

Up

Counter Period (AutoReloa...

24

Internal Clock Division (CKD)

No Division

auto-reload preload

Disable

Parameter Settings

User Constants

Configure the below parameters :

Search (Ctrl+F)

◀ ▶

i

Master/Slave Mode (MSM ...

Disable (Trigger input effect not delay...

Trigger Event Selection

Reset (UG bit from TIMx\_EGR)

PWM Generation Channel 1

Mode

PWM mode 1

Pulse (16 bits value)

0

Output compare preload

Enable

Fast Mode

Disable

CH Polarity

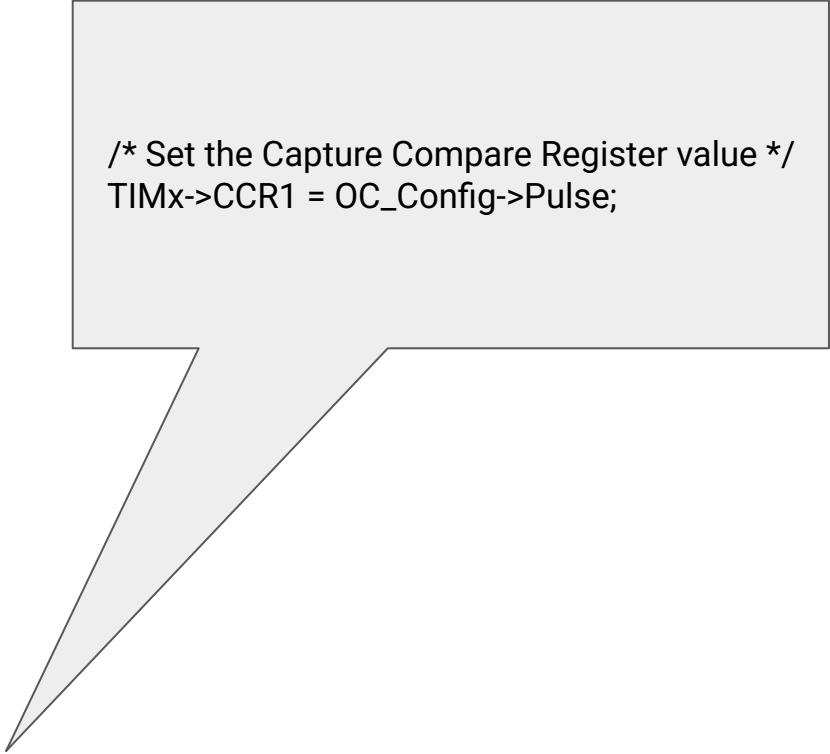
High

# Ejemplo de PWM en el STM32F103C8

Código generado:

```
htim2.Instance = TIM2;
htim2.Init.Prescaler = 71;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 999;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
HAL_TIM_Base_Init(&htim2);
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig);
HAL_TIM_PWM_Init(&htim2);

sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig);
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 500;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1);
```



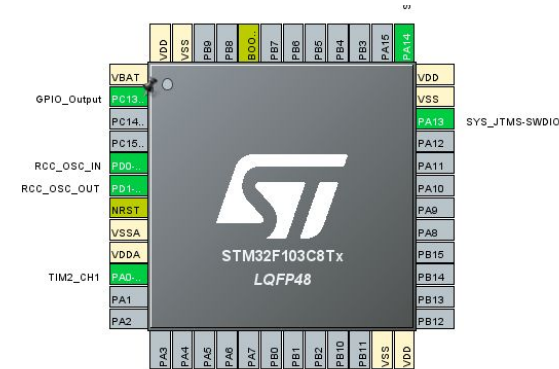
```
/* Set the Capture Compare Register value */
TIMx->CCR1 = OC_Config->Pulse;
```

# Ejemplo de PWM en el STM32F103C8

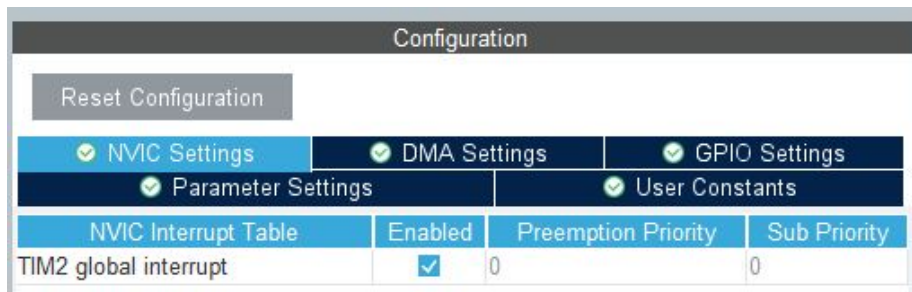


```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
}
```



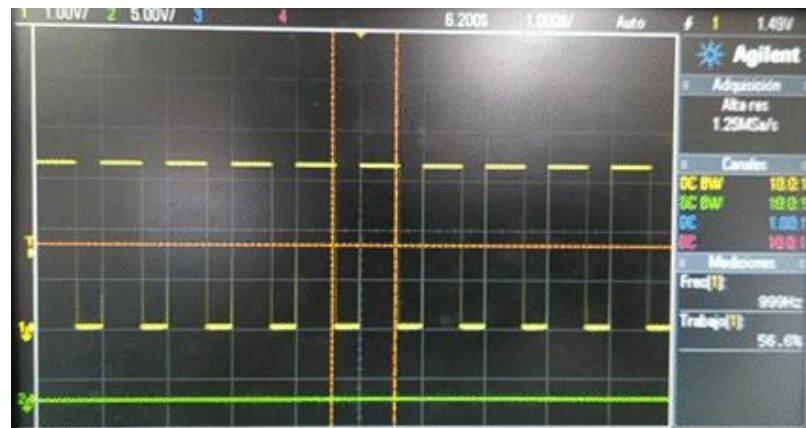
# Ejemplo de PWM en el STM32F103C8 - Actualización CCR1



```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start_IT(&htim2, TIM_CHANNEL_1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
}
```

```
220
229 /* USER CODE BEGIN 4 */
230 void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim){
231
232     idx = (idx+1)%1000;
233     TIM2->CCR1 = idx;
234
235 }
```

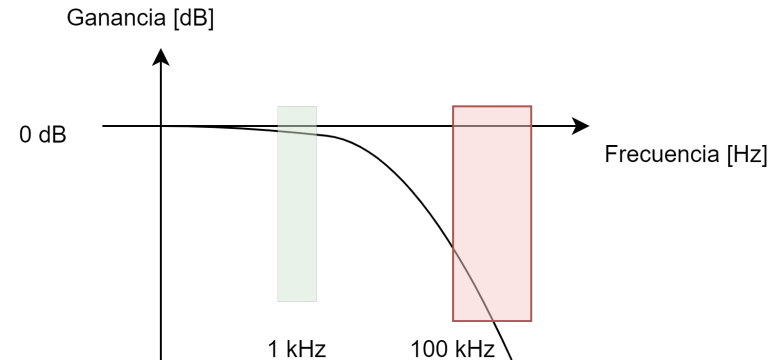
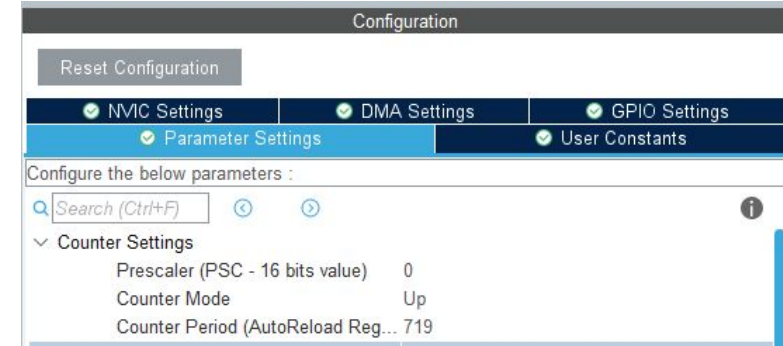




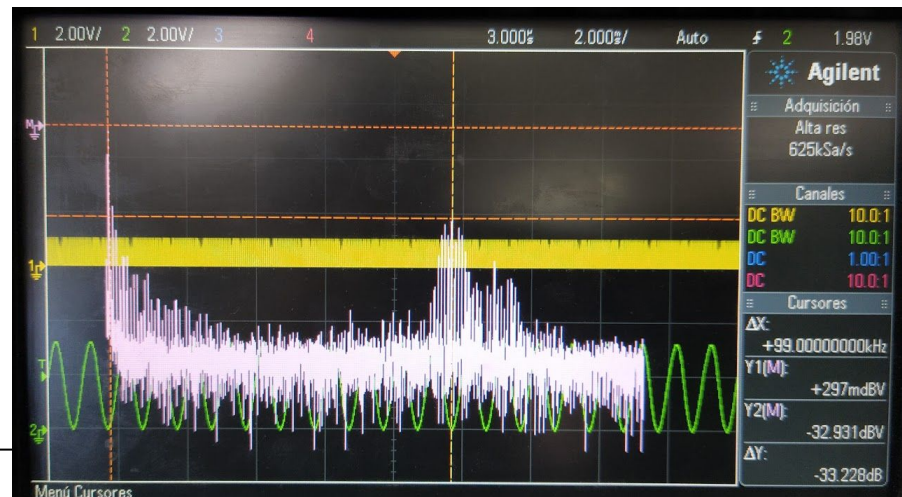
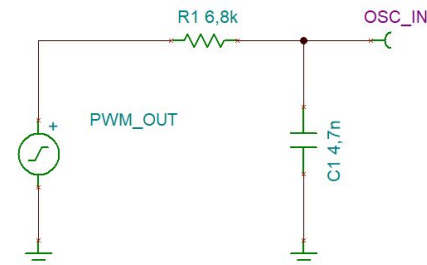
# Ejemplo de PWM en el STM32F103C8 - Sinusoide

```
#include "main.h"
#include <math.h>
#define NSEN 100
#define VMAX 720
uint16_t onda[NSEN] = {0};
int idx=0;
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_TIM2_Init();

    /* USER CODE BEGIN 2 */
    for(idx = 0; idx < NSEN ; idx++){
        onda[idx] = ( uint16_t ) ((float)VMAX/2.0*(sin(2.0*M_PI*idx/( float)NSEN)+1.0));
    }
    HAL_TIM_PWM_Start_IT(&htim2, TIM_CHANNEL_1);
    /* USER CODE END 2 */
    while (1) { }
}
/* USER CODE BEGIN 4 */
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef * htim){
    GPIOC->BSRR |= GPIO_BSRR_BS13_Msk;
    idx = (idx+1)%NSEN;
    TIM2->CCR1 = onda[idx];
    GPIOC->BSRR |= GPIO_BSRR_BR13_Msk;
}
/* USER CODE END 4 */
```



# Ejemplo de PWM en el STM32F103C8 - Sinusoide



# Ejemplo de PWM en el STM32F103C8 - Sinusoide

