MICROS 32 BITS
STM - PWM

ROBINSON JIMENEZ MORENO

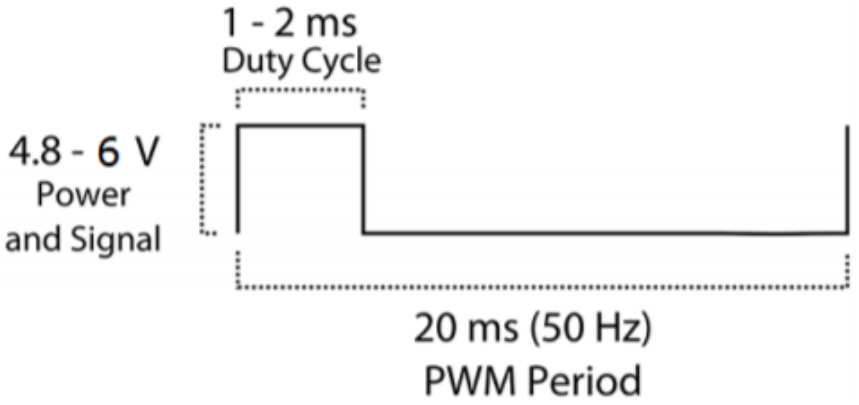
LUISA FERNANDA GARCIA

main.cpp*

```
#include <stdio.h>
#include "STM32F7xx.h"
int paso,i=0;

extern "C" {
  void SysTick_Handler ( void )
    {
    paso+=20;
      GPIOB->ODR=~GPIOB->ODR;
  GPIOC->ODR=1;
for(i=0;i<paso;i++){};
  GPIOC->ODR=0;
  if (paso>1200){paso=100;}
    }
}

  int main(void){

  RCC->AHB1ENR |=0xFF; //TODOS LOS
  GPIOB->MODER |=  0x000055;
  GPIOC->MODER |=  0x000055;
  GPIOC->OTYPER |= 0;
  GPIOC->OSPEEDR |= 0x555555;
  GPIOC->PUPDR |=  0x10000000;
//************************************
    SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);
  GPIOB->ODR=1;
    paso=100;
  while(true)    {

    }}
```

MG90S
Metal Gear Servo

PWM=Orange
Vcc = Red (+)
Ground=Brown (–)



Specifications

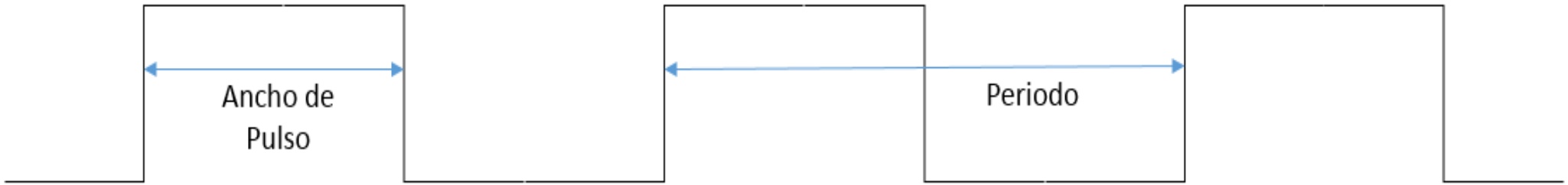
- Weight: 13.4 g
- Dimension: 22.5 x 12 x 35.5 mm approx.
- Stall torque: 1.8 kgf·cm (4.8V), 2.2 kgf·cm (6 V)
- Operating speed: 0.1 s/60 degree (4.8 V), 0.08 s/60 degree (6 V)
- Operating voltage: 4.8 V - 6.0 V
- Dead band width: 5 µs

https://youtu.be/swx1dCAQhzk

PWM - STM32F4

PWM:

La modulación por ancho de pulsos (PWM) consiste en la modificación del ciclo de trabajo o ancho del pulso de una señal periódica



La función de modulación por ancho de pulsos permite generar una señal con una frecuencia determinada por el valor del registro TIMx_ARR y un ciclo de trabajo determinado por el valor del registro TIMx_CCRx

General-purpose timers (TIM2/TIM3/TIM4/TIM5)

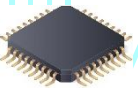
TIM2/TIM3/TIM4/TIM5 introduction

The general-purpose timers consist of a 16-bit or 32-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (*input capture*) or generating output waveforms (*output compare and PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

The timers are completely independent, and do not share any resources. They can be synchronized together as described in *Section 23.3.19: Timer synchronization*.



TIM2/TIM3/TIM4/TIM5 main features

General-purpose TIMx timer features include:

- 16-bit (TIM3, TIM4) or 32-bit (TIM2 and TIM5) up, down, up/down auto-reload counter.
- 16-bit programmable prescaler used to divide (also “on the fly”) the counter clock frequency by any factor between 1 and 65535.
- Up to 4 independent channels for:
  - Input capture
  - Output compare
  - PWM generation (Edge- and Center-aligned modes)
  - One-pulse mode output
- Synchronization circuit to control the timer with external signals and to interconnect several timers.
- Interrupt/DMA generation on the following events:
  - Update: counter overflow/underflow, counter initialization (by software or internal/external trigger)
  - Trigger event (counter start, stop, initialization or count by internal/external trigger)
  - Input capture
  - Output compare

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

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

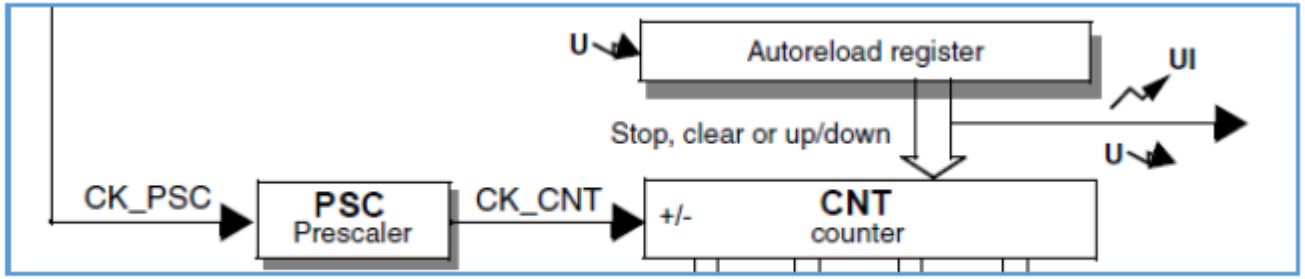


Figura 1. Diagrama de bloques del circuito de base de tiempos

https://riunet.upv.es/bitstream/handle/10251/39538/ArticuloTimers.pdf?sequence=1

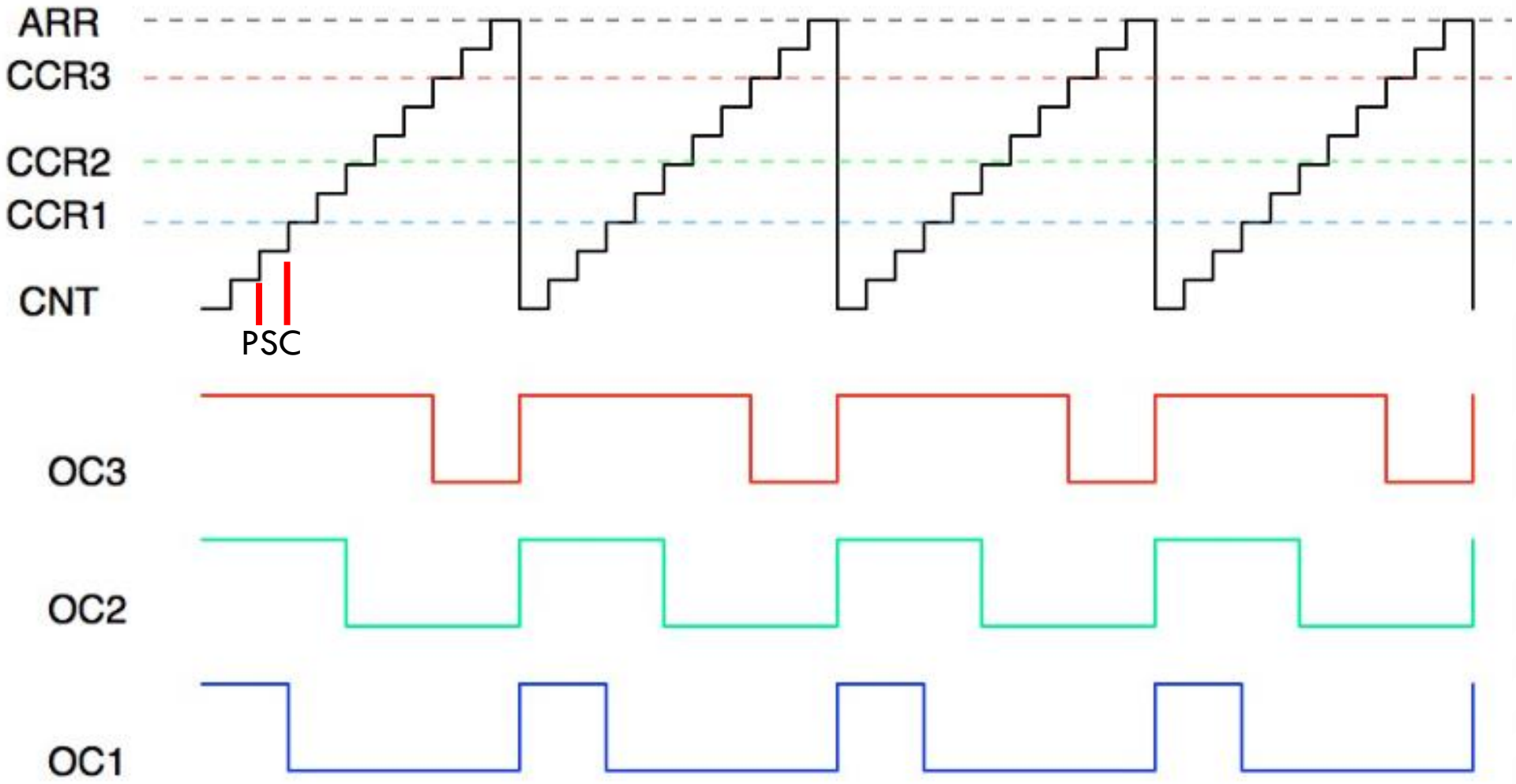
PWM mode

Pulse width modulation mode allows you to generate 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.

The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing 110 (PWM mode 1) or ‘111 (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register.

As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register.

Three PWM signals from the Output Compare Channels of a general purpose timer



23.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. So you must take care that the same bit can have a different meaning for the input stage and for the output stage.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 3 **OC1PE**: Output compare 1 preload enable

0: Preload register on TIMx_CCR1 disabled. TIMx_CCR1 can be written at anytime, the new value is taken in account immediately.

1: Preload register on TIMx_CCR1 enabled. Read/Write operations access the preload register. TIMx_CCR1 preload value is loaded in the active register at each update event.

*Note: **1:** These bits can not be modified as long as LOCK level 3 has been programmed (LOCK bits in TIMx_BDTR register) and CC1S=00 (the channel is configured in output).*

***2:** The PWM mode can be used without validating the preload register only in one-pulse mode (OPM bit set in TIMx_CR1 register). Else the behavior is not guaranteed.*

Bit 2 **OC1FE**: Output compare 1 fast enable

This bit is used to accelerate the effect of an event on the trigger in input on the CC output.

0: CC1 behaves normally depending on counter and CCR1 values even when the trigger is ON. The minimum delay to activate CC1 output when an edge occurs on the trigger input is 5 clock cycles.

1: An active edge on the trigger input acts like a compare match on CC1 output. Then, OC is set to the compare level independently from the result of the comparison. Delay to sample the trigger input and to activate CC1 output is reduced to 3 clock cycles. OCFE acts only if the channel is configured in PWM1 or PWM2 mode.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

OC2M – OC1M

0110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF=‘0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).

0111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

23.4.9 TIMx capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

Bit 3 **CC1NP**: *Capture/Compare 1 output Polarity.*

CC1 channel configured as output: CC1NP must be kept cleared in this case.

CC1 channel configured as input: This bit is used in conjunction with CC1P to define TI1FP1/TI2FP1 polarity. refer to CC1P description.

Bit 1 **CC1P**: *Capture/Compare 1 output Polarity.*

CC1 channel configured as output:

0: OC1 active high

1: OC1 active low

Bit 0 **CC1E**: *Capture/Compare 1 output enable.*

CC1 channel configured as output:

0: Off - OC1 is not active

1: On - OC1 signal is output on the corresponding output pin

CC1 channel configured as input: This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled

1: Capture enabled

Note: The state of the external IO pins connected to the standard OCx channels depends on the OCx channel state and the GPIO and AFIO registers.

23.4.13 TIMx capture/compare register 1 (TIMx_CCR1)

CCR2 para canal 2

Address offset: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

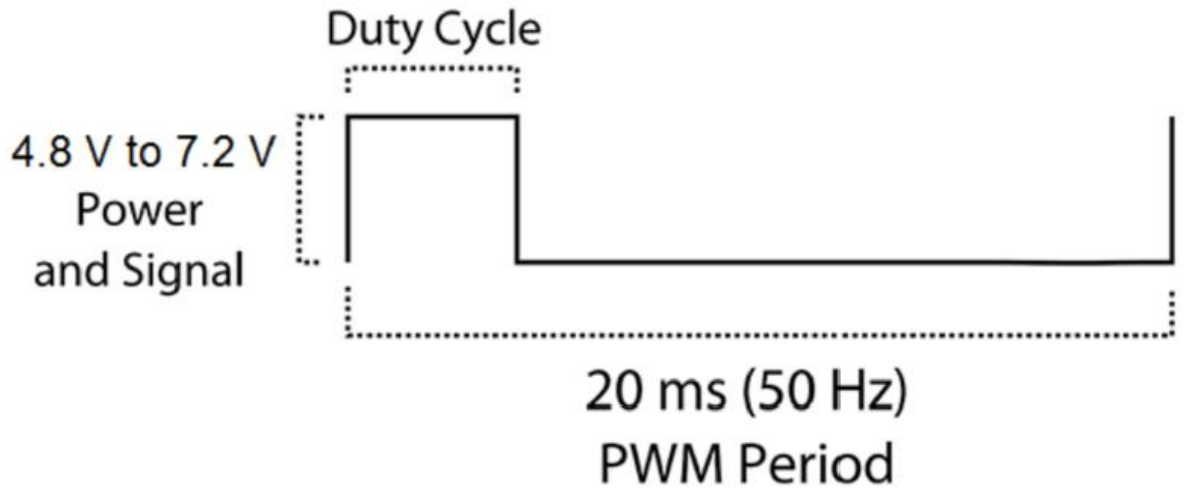
Bits 31:16 **CCR1[31:16]**: High Capture/Compare 1 value (on TIM2 and TIM5)

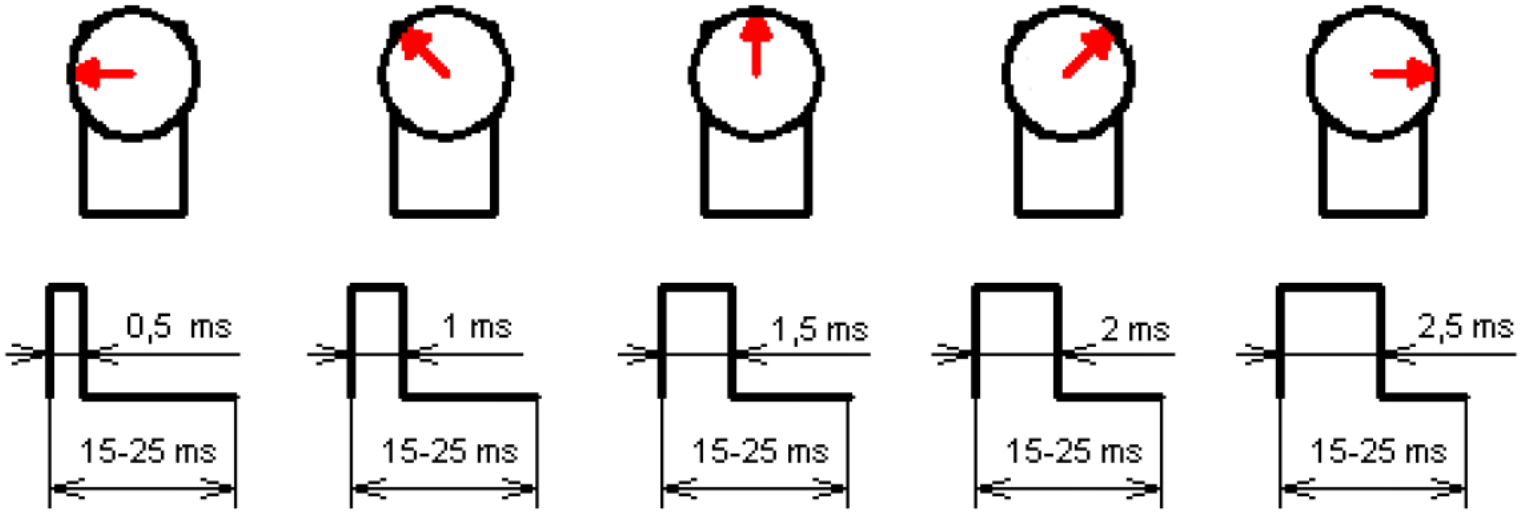
Bits 15:0 **CCR1[15:0]**: Low Capture/Compare 1 value

If channel CC1 is configured as output:
CCR1 is the value to be loaded in the actual capture/compare 1 register (preload value).
It is loaded permanently if the preload feature is not selected in the TIMx_CCMR1 register (bit OC1PE). Else the preload value is copied in the active capture/compare 1 register when an update event occurs.
The active capture/compare register contains the value to be compared to the counter TIMx_CNT and signaled on OC1 output.

If channel CC1is configured as input:
CCR1 is the counter value transferred by the last input capture 1 event (IC1). The TIMx_CCR1 register is read-only and cannot be programmed.

Aplicación en servomotores





EJERCICIO: Generar una señal PWM para el servomotor anterior mediante timer 3 en el canal uno empleando el pin PA6.

SOLUCIÓN:

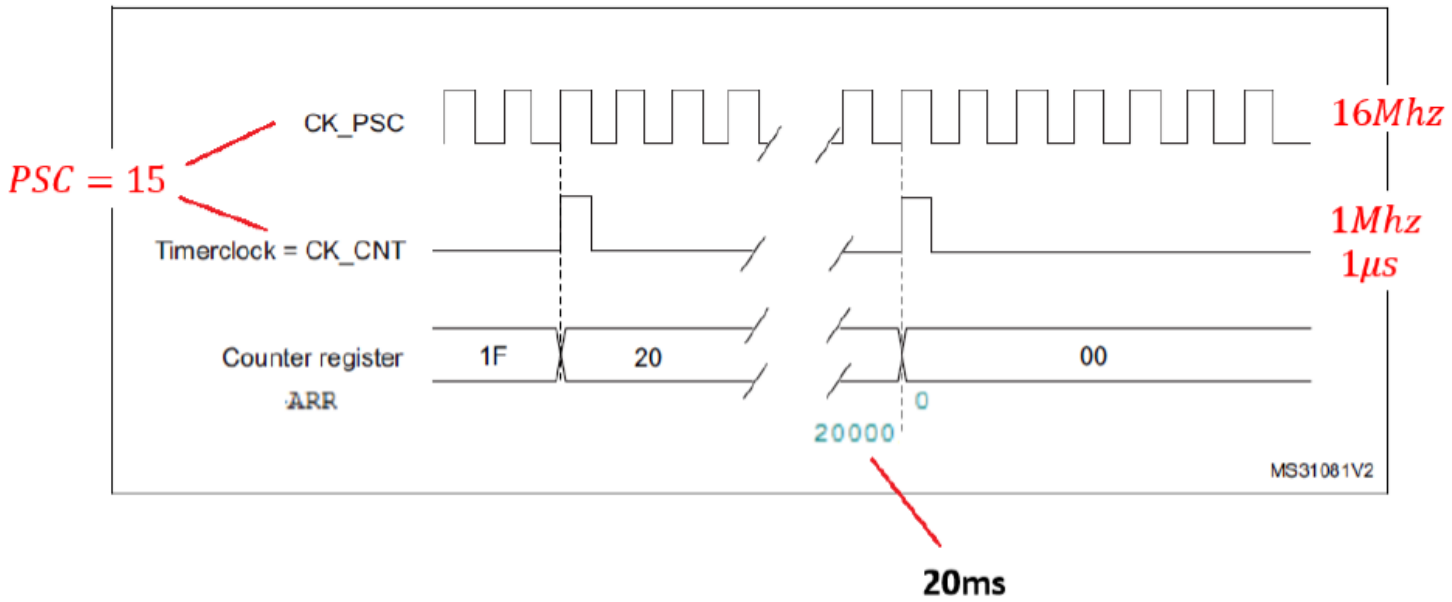
- **Requerimientos de la señal:**

$$1Mhz = \frac{16Mhz}{PSC + 1} \qquad PSC + 1 = \frac{16Mhz}{1Mhz}$$

$$PSC = 15$$

- **Frecuencia:** 50Hz
- **Periodo:** 20ms

- **Referencia de cambio de acuerdo al ancho de pulso:**
  - Valor mínimo: 0,5ms - Cero grados
  - Valor máximo: 2,5ms



ARR=FrecIncrPWM/ FrecSeñalPWM
ARR=T_señalPWM/T_IncrPWM

Table 12. STM32F745xx and STM32F746xx alternate function mapping

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/LPTIM1/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4/5/6	SPI3/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI2/USART6/UART4/5/7/8/SPDIFRX	CAN1/2/TIM12/13/14/QUADSPI/LCD	SAI2/QUADSPI/OTG2_HS/OTG1_FS	ETH/OTG1_FS	FMC/SDMMC1/OTG2_FS	DCMI	LCD	SYS
Port A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	SAI2_SD_B	ETH_MII_CRS	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	QUADSPI_BK1_IO3	SAI2_MCK_B	ETH_MII_RX_CLK/ETH_RMII_REF_CLK	-	-	LCD_R2	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	SAI2_SCK_B	-	-	ETH_MDIO	-	-	LCD_R1	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	-	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS/I2S1_WS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_SOF	DCMI_HSYNC	LCD_VSYNC	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK/I2S1_CK	-	-	-	-	OTG_HS_ULPI_CK	-	-	-	LCD_R4	EVENTOUT
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	TIM13_CH1	-	-	-	DCMI_PIXCLK	LCD_G2	EVENTOUT
							SPI1_M						ETH_MII_				

```
GPIOA->MODER |=0x2000;
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

0X 2 0 0 0

Bits 2y+1:2y **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O mode.

00: Input mode (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

```
GPIOA->AFR[0] = 0x2000000;
```

0X **2** **0** **0**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

0 **0** **0** **0**

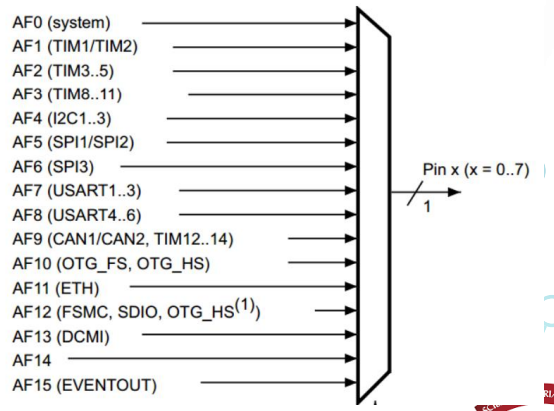
Bits 31:0 **AFRy[3:0]:** Alternate function selection for port x pin y (y = 0..7)

These bits are written by software to configure alternate function I/Os

AFSELy selection:

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: AF8
1001: AF9
1010: AF10
1011: AF11
1100: AF12
1101: AF13
1110: AF14
1111: AF15

Cada GPIO tiene multiplexadas hasta 16 funciones alternativas como se muestra a continuación.



23.4.7 TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	1	1	0	0	0	0	0

```
TIM1->CCMR1=  0x60;

//PWM modo 1, preload del CCR1 deshabilitado, CH1 configurado como salida
```

CICLO UTIL FIJO

```
//////////////////////////////////////

#include "STM32F7xx.h"
int dato=0;
int main(void){
  RCC->AHB1ENR |= 7;     //HABILITA EL CLOCK DEL PTB
  RCC->APB1ENR |= (1UL << 1);     //HABILITA CLOCK TIM3
  GPIOA->MODER |=0x2000;      //PTA6 en MODO ALTERNO
  GPIOA->AFR[0] = 0x2000000;        //PTA6 funcion alterna  AF2= TIM3_CH1
  TIM3->EGR |= (1UL<<0);            //UG = 1 ,  RE-inicializar el contador
  TIM3->PSC = 15;      //señal de reloj HSI=16Mhz, se necesita generar 1Mhz por lo tanto PSC=15
  TIM3->ARR = 20000;           //con una frecuencia de 1Mhz -> T=1uS :
  TIM3->DIER |= (1UL<<0);
  TIM3->CR1 |= (1UL<<0);     //Enable counter
  TIM3->CCMR1 = 0x60;        //PWM modo 1, preload del CCR1 deshabilitado, CH1 configurado como salida
  TIM3->CCER |= (1UL<<0);   //OC1 signal is output on the corresponding output pin
  TIM3->CCR1 = 1000;         //conteo hasta 1000 significa 1000*1uS = 1ms
  while(true){  }//cierra while
}//cierra main
```

CICLO UTIL VARIABLE

```c
#include "STM32F7xx.h"

int dato=0;

int main(void)
{
  //************************************************************
  //CONFIGURACION "CLOCK"
  RCC->AHB1ENR |= 7;    //HABILITA EL CLOCK DEL PTA,B,C
  RCC->APB1ENR |= (1UL << 1);    //HABILITA CLOCK TIM3
  GPIOB->MODER = 0x10004001;        //PTB0, PTB7 y PTB 14 -> OUTPUT
  GPIOC->MODER= 0; //pulsador como entrada (PC13)
  //CONFIGURACION PTA6 -> TIM3_CH1
  GPIOA->MODER |= 0x2000;
  GPIOA->AFR[0] = 0x2000000;        //PTA6 funcion alterna  AF2= TIM3_CH1
  //CONFIGURACION DEL TIM3_CH1
  TIM3->EGR |= (1UL<<0);            //UG = 1 ,  RE-inicializar el contador
  TIM3->PSC = 15;                   //señal de reloj HSI=16Mhz, se necesita generar 1Mhz por lo tanto PSC=15
  TIM3->ARR = 20000;                //con una frecuencia de 1Mhz -> T=1uS :
  TIM3->DIER |= (1UL<<0);           //UIE = 1, update interrupt enable
                                    //conteo hasta 20000 significa 20000*1uS = 20ms //periodo de la señal de control del servo
  TIM3->CR1 |= (1UL<<0);            //Enable counter
  TIM3->CCMR1 = 0x60;               //PWM modo 1, preload del CCR1 deshabilitado, CH1 configurado como salida
  TIM3->CCER |= (1UL<<0);           //OC1 signal is output on the corresponding output pin
  TIM3->CCR1 = 510;                 //conteo hasta 510 significa 510*1uS = 0,51ms

  //************************************************************
```

```
22    while(1){
23      if (GPIOC->IDR &= 0x2000){
24        if (dato>18000){
25          dato=510;
26        }
27      TIM3 ->CCR1 = dato;
28        dato=dato+510;
29      for(int i=0; i<200000; i++);
30      }
31
32    }
33  }
34
```

https://youtu.be/E9fc3ONdVi8