MICROS 32 BITS STM - PWM

ROBINSON JIMENEZ MORENO

UNIVERSIDAD MILITAR NUEVA GRANADA

INGENIERÍA MECATRÓNICA UMNG

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

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.

Pulse Width Modulation is a way of modifying a signal my changing the proportion of time it is on and off. There are other uses of the term but, for now, I am only interested in signals where the amount of time that the signal is active – the duty cycle – can be varied from 0 to 100%. Signals such as these are used a lot in power control applications like light dimmers and motor speed control. The brightness of your computer screen is almost certainly controlled by a PWM signal.

The STM32 general purpose timers like TIM3 and TIM4 have hardware that makes it easy to generate PWM signals. In fact they have several modes for just this purpose. I will consider only simplest type which is good for the great majority of application. There are four channels available and each can have a different duty cycle although the basic frequency will be the same for each.

Basic PWM mode is similar to the output compare toggle mode except that the output pin is cleared whenever there is a match between the CCRx and the CNT registers and then set again when the counter reloads.

The frequency of the PWM signal can be important parameter of the setup. For changing the brightness of an LED, any frequency above a few tens of Hertz will not be seen by the eye. Below that and the light will seem to flicker. Even at relatively high frequencies you may sometimes see spotty trails left by PWM dimmed lights as your vision tracks across them. For motors, it is often a good idea to have PWM frequencies of well above the range of normal hearing. Otherwise, your motor controller will appear to whine a lot. It is someties difficult to distinguish the whining of the motor from the whining of the customers who have to listen to it.

Another factor to take into account is the minimum resolution you need from the PWM system. That is, how many different steps in duty cycle are needed. The most obvious choices might be 100 steps, corresponding to percentages, or 256 steps, corresponding to the range of values in an 8 bit integer. Perhaps you only want 16 different intensities of a backlight. It is generally OK to have more resolution than you need but to have less might be a problem.

As with other timer problems, there are a number of constraints to consider when setting up the timer timebase. These are:

TIMER_Frequency – the input clock to the timer module

PWM_Steps – the number of different duty cycles needed

PWM_Frequency – the repetition rate of the PWM signal

These numbers determine the reload register and prescaler values used to configure the timer.

Suppose I want to have 100 different PWM levels and a frequency of 100Hz for a simple LED dimming application. First, I need to calculate the frequency needed to drive the counter

COUNTER_Frequency = PWM_Frequency * PWM_Steps = 100 * 100 = 10000 = 10KHz

Now I need to find a value for the prescaler that will give me a 10kHz clock from the Timer_Frequency

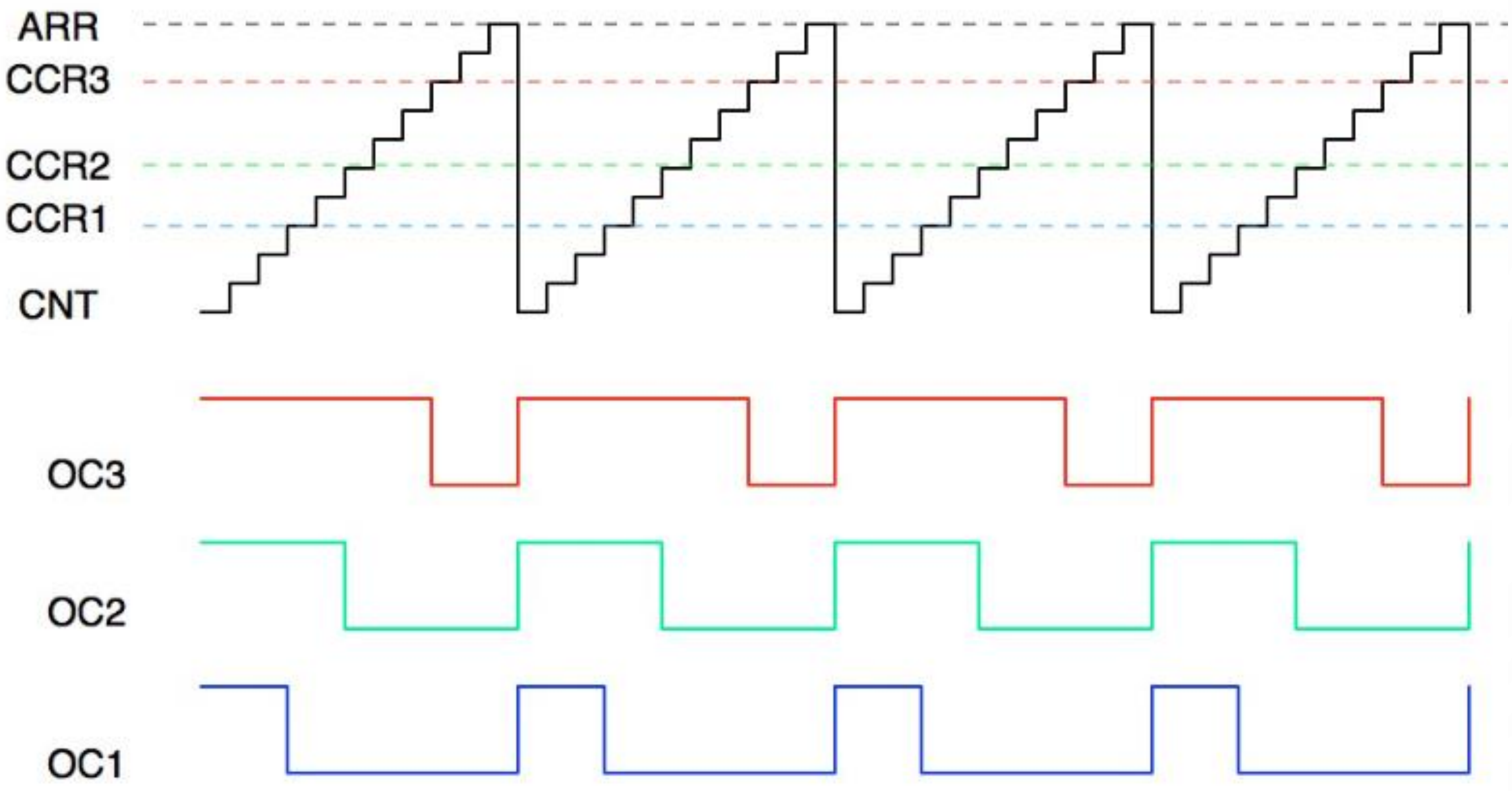
TIMER_Prescale = (TIMER_Frequency / COUNTER_Frequency) – 1 = 72000000/10000 – 1 = 7199

This value is safely within the range of an unsigned 16 bit register so I should be safe to proceed.

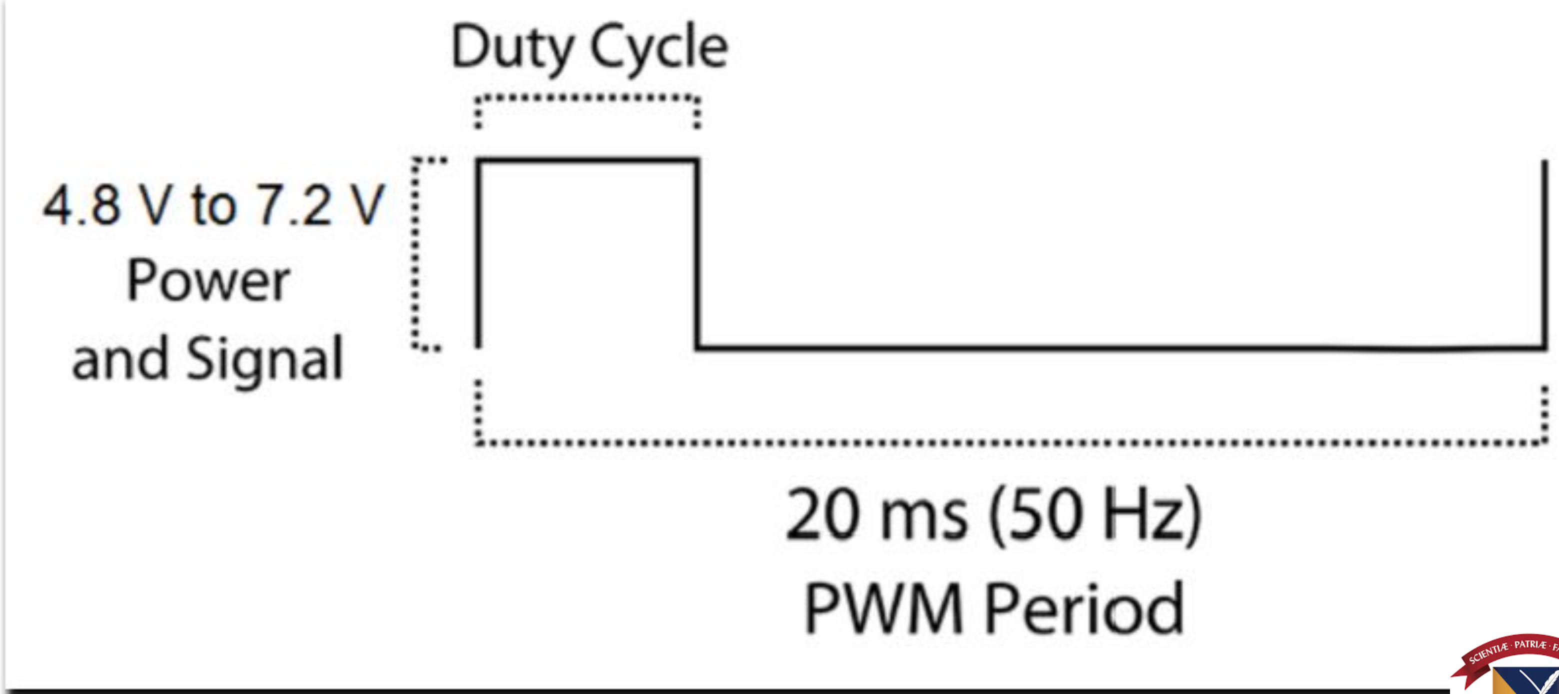
The ARR register will get a value that is PWM_Steps – 1 and I am ready to configure the timer timebase.

http://www.micromouseonline.com/2016/02/06/pwm-basics-on-the-stm32-general-purpose-timers/

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



EJEMPLO: Diseñar un programa que permita controlar el giro de un Servomotor (en un solo sentido) por medio de un PWM. El incremento del ancho de pulso del PWM se deberá realizar por medio del pulsador de la tarjeta. Cada vez que se pulse el botón, el servomotor deberá recorrer N-grados. Cuando el PWM llegue al máximo permitido para el servomotor, el sistema deberá reiniciarse al punto cero grados.



EJEMPLO: Diseñar un programa que permita controlar el giro de un Servomotor (en un solo sentido) por medio de un PWM. El incremento del ancho de pulso del PWM se deberá realizar por medio del pulsador de la tarjeta. Cada vez que se pulse el botón el servomotor deberá recorrer N-grados. Cuando el PWM llegue al máximo permitido para el servomotor, el sistema deberá reiniciarse al punto cero grados.

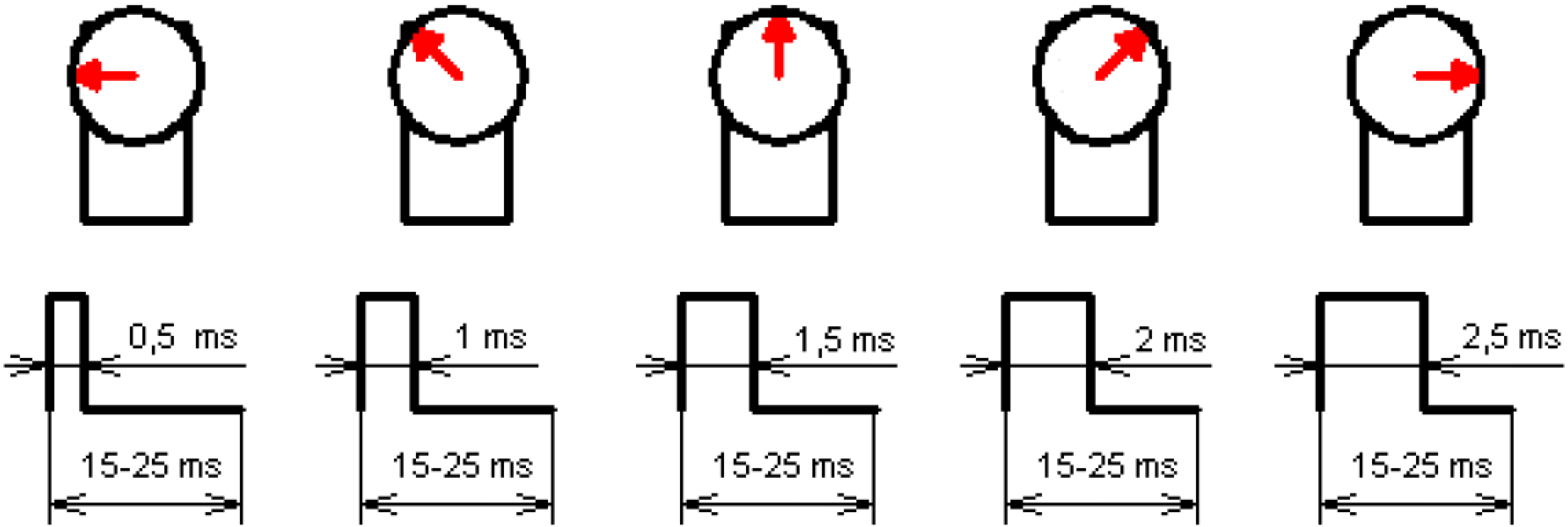


Table 12. STM32F745xx and STM32F746xx alternate function mapping

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
	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
Port A	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_				

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

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

SOLUCIÓN:

- Puertos y pines necesarios:

  - Pulsador = PC13

  - MODULO TIMER = TIM3
    - Canal del TIM3 - PWM : Canal 1 – Pin PA6
    - Interrupciones deshabilitadas

- Requerimientos de la señal:

  - Frecuencia: 50Hz
  - Periodo: 20ms

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

SOLUCIÓN:

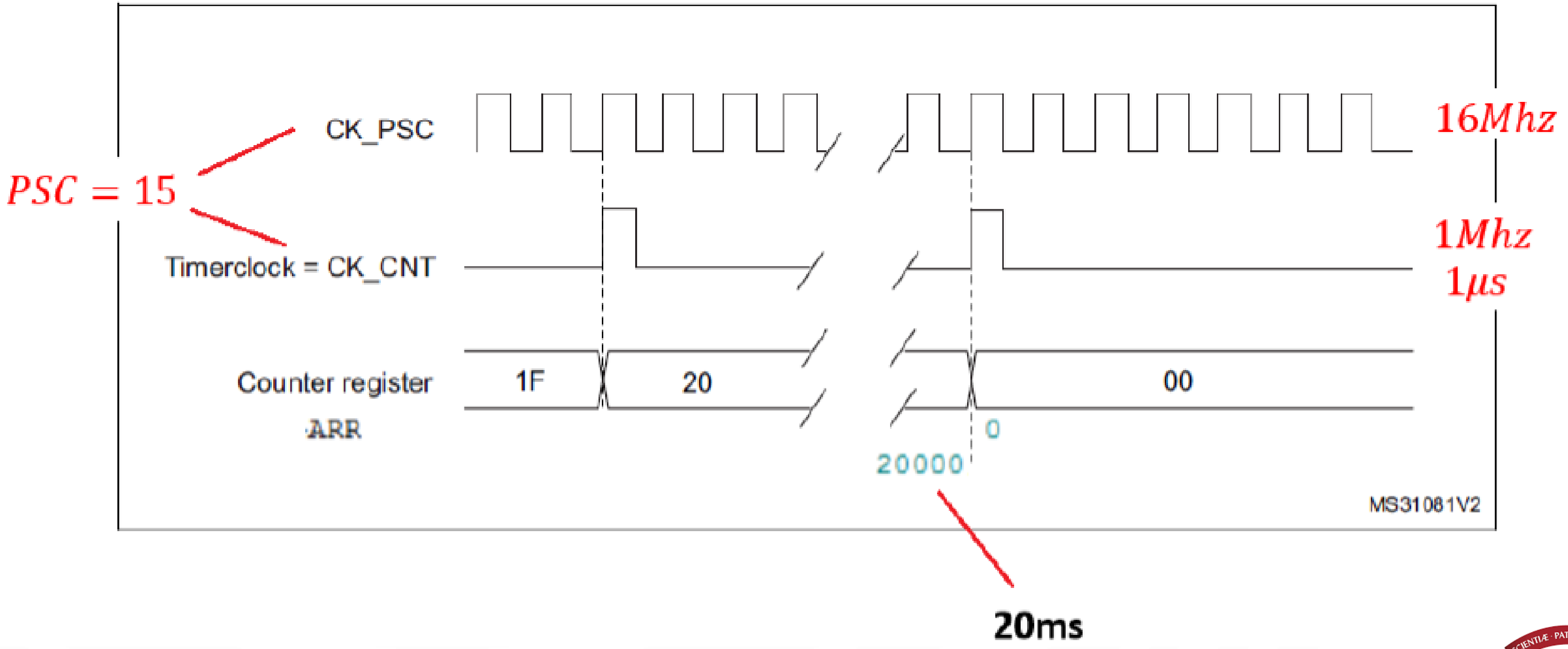
- **Requerimientos de la señal:**

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

$$PSC = 15$$

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

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



```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

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

INGENIERÍA MECATRÓNICA

UNIVERSIDAD MILITAR NUEVA GRANADA

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

while(true){
  if(GPIOC->IDR &= 0x2000){
      if(dato>2400){                //limite del PMW en este caso para un servo de referecia
        dato=510;                   //MG996R
      }
      TIM3->CCR1 = dato;             //Actualiza el valor del PWM
      for(int i=0;i<200000;i++);       //anti-rebote por software
      }
  }//cierra while
}//cierra main
```