**TIMER**

**Características clave:**

1. **Misma base de tiempo**: Todos los canales de un timer comparten:
   * El **reloj** (clock) que alimenta al timer.
   * El **prescaler** (divisor de frecuencia).
   * El **contador principal** (CNT), que cuenta de 0 a un valor configurable (Auto-Reload Register, **ARR**).
2. **Registros independientes por canal**:
   * Cada canal tiene su propio **Capture/Compare Register** (**CCRx**, donde "x" es el número de canal). Esto permite configurar acciones específicas para cada canal (ej: generar PWM con diferente ciclo de trabajo).

**CONFIGURACION TIMER**

**1. Reloj (Clock) del Timer**

* **Qué es**: La señal de reloj es el "latido" que determina la velocidad a la que el timer cuenta.
* **Fuentes comunes**:
  + **Reloj interno (HSI/HSE)**: Por ejemplo, 16 MHz (HSI) o 8-25 MHz (HSE, usando un cristal externo).
  + **Reloj del sistema (SYSCLK)**: Deriva del HSI, HSE, o PLL (Phase-Locked Loop), que multiplica la frecuencia.
  + **Reloj externo**: Puede usarse una señal externa (como ETR, que veremos más adelante).
* **Importancia**: La frecuencia del reloj determina la **resolución temporal** del timer.
  + Ejemplo: Si el reloj es de 72 MHz, el timer cuenta 72 millones de pulsos por segundo.

**2. Prescaler (Preescalador)**

* **Qué es**: Un divisor de frecuencia que reduce la velocidad del reloj que llega al timer.
* **Fórmula**:

Frecuencia timer=Frecuencia relojPrescaler + 1Frecuencia timer=Prescaler + 1Frecuencia reloj​

* **Ejemplo**:
  + Si el reloj es 72 MHz y el prescaler es 71:

Frecuencia timer=72 MHz71+1=1 MHz(el timer cuenta 1 milloˊn de veces por segundo).Frecuencia timer=71+172MHz​=1MHz(el timer cuenta 1 milloˊn de veces por segundo).

* **Para qué sirve**:
  + Ajustar la velocidad del timer para aplicaciones específicas (ej: generar PWM de baja frecuencia o medir señales lentas).

**3. ARR (Auto-Reload Register)**

* **Qué es**: Un registro que define el valor máximo que alcanza el contador del timer antes de reiniciarse (overflow).
* **Fórmula de la frecuencia de salida** (para PWM o temporizaciones):

Frecuencia sen˜al=Frecuencia timerARR + 1Frecuencia sen˜al=ARR + 1Frecuencia timer​

* **Ejemplo**:
  + Si el timer está a 1 MHz y el ARR es 999:

Frecuencia sen˜al=1 MHz999+1=1 kHz(sen˜al con periodo de 1 ms).Frecuencia sen˜al=999+11MHz​=1kHz(sen˜al con periodo de 1 ms).

* **Para qué sirve**:
  + Definir el periodo de la señal (en PWM) o el tiempo entre interrupciones.

**Relación entre Reloj, Prescaler y ARR**

* **Ejemplo completo**:
  + **Objetivo**: Generar una señal PWM de 1 kHz con un STM32 de 72 MHz.
  + **Prescaler**: 71 → Frecuencia timer = 1 MHz.
  + **ARR**: 999 → Frecuencia PWM = 1 kHz.
  + **Duty Cycle**: Si el **CCR** (Capture/Compare Register) es 300, el duty cycle será:

Duty cycle=300999+1×100%=30%.Duty cycle=999+1300​×100%=30%.

**4. Timer ETR (External Trigger)**

El **ETR** (External Trigger) es una entrada especial en algunos timers que permite **sincronizar o controlar el timer con una señal externa**. No es un canal como los CH1, CH2, etc., sino una función alternativa.

**Funcionalidades del ETR:**

1. **Modo de reloj externo**:
   * El timer usa una señal externa (por ejemplo, un pulso de otro dispositivo) como reloj, en lugar del reloj interno.
   * Ejemplo: Contar eventos externos (como pulsos de un encoder).
2. **Trigger externo**:
   * Una señal externa inicia, detiene, reinicia, o sincroniza el timer.
   * Ejemplo: Iniciar una conversión ADC cuando llega un pulso por ETR.
3. **Filtrado y prescaler externo**:
   * El ETR puede configurarse con un filtro digital para eliminar ruido y un prescaler para dividir la señal externa.

**Configuración del ETR en STM32CubeIDE**

1. **Selecciona el timer** que tenga la función ETR (ej: TIM2, TIM3, etc.).
2. **Habilita el modo ETR**:
   * En la pestaña **Parameter Settings** del timer, busca **Clock Source** y selecciona **External Trigger**.
3. **Configura los parámetros**:
   * **ETR Prescaler**: Divide la señal externa (ej: 1, 2, 4, 8).
   * **ETR Polarity**: Flanco de subida (Rising) o bajada (Falling).
   * **ETR Filter**: Filtro para eliminar ruido (0 a 15 ciclos).

**Ejemplo práctico: Contar pulsos externos con ETR**

Supongamos que queremos contar los pulsos de un encoder conectado a la entrada ETR del TIM2:

1. **Configuración**:
   * **Clock Source**: External Trigger.
   * **ETR Prescaler**: 1 (sin división).
   * **ETR Polarity**: Rising Edge (flanco de subida).
   * **ETR Filter**: 10 (filtrar ruido de alta frecuencia).
2. **Código**:

c

Copy

// Iniciar el timer en modo cuenta externa

HAL\_TIM\_Base\_Start(&htim2);

// Leer el valor del contador

uint32\_t pulsos = \_\_HAL\_TIM\_GET\_COUNTER(&htim2);

**Diferencias entre ETR y los canales (CH1, CH2, etc.)**

| **Característica** | **ETR** | **Canales (CH1, CH2...)** |
| --- | --- | --- |
| **Propósito** | Sincronización externa o reloj | Generar PWM, capturar pulsos, etc. |
| **Dirección** | Entrada única | Pueden ser entrada o salida |
| **Registros asociados** | TIMx\_SMCR (Slave Mode Control) | TIMx\_CCRx (Capture/Compare) |
| **Ejemplo de uso** | Contar pulsos externos | Controlar un servo con PWM |

**Casos de uso avanzados con ETR**

1. **Sincronizar múltiples timers**:
   * Un timer maestro envía un trigger (por ETR) a un timer esclavo para iniciar su conteo.
2. **Medir frecuencia de señales**:
   * Usar ETR como entrada de reloj y el timer como contador para medir la frecuencia de una señal externa.
3. **Controlar motores paso a paso**:
   * Sincronizar el timer con pulsos externos para generar secuencias precisas.

**Consejos clave**

* **Documentación**: Consulta el **Reference Manual** de tu STM32 para ver qué pines soportan ETR (varía por modelo).
* **Frecuencia máxima**: Asegúrate de que la señal externa no exceda la frecuencia máxima del timer.
* **Filtrado**: Usa el filtro del ETR si trabajas en entornos ruidosos.

Si tienes un proyecto específico donde quieras usar ETR o ajustar reloj/prescaler/ARR, ¡dímelo y te ayudo con la configuración! 😊

**TIMER CONFIGURATION CODE**

htim8.Instance = TIM8; // Usa el TIM8.

htim8.Init.Prescaler = 0; // Prescaler = 0 → Sin división (reloj directo: 180 MHz).

htim8.Init.CounterMode = TIM\_COUNTERMODE\_UP; // Cuenta ascendente.

htim8.Init.Period = 65535; // ARR = 65535 (máximo valor para un timer de 16 bits).

htim8.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1; // Sin división adicional.

htim8.Init.RepetitionCounter = 0; // Sin repetición (solo en timers avanzados).

htim8.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE; // ARR se actualiza inmediatamente.

sConfigOC.OCMode = TIM\_OCMODE\_PWM1; // Modo PWM 1 (el pin se pone en HIGH cuando CNT < CCR).

sConfigOC.Pulse = 0; // CCR inicial = 0 (duty cycle del 0%).

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH; // Señal HIGH durante el duty cycle.

sConfigOC.OCNPolarity = TIM\_OCNPOLARITY\_HIGH; // Polaridad del canal complementario (no usado aquí).

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE; // Deshabilita el modo rápido (cambios en CCR no son inmediatos).

sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET; // Estado del pin cuando el timer está inactivo.

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET; // No genera trigger a otros periféricos.

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE; // No sincroniza con otros timers.

sBreakDeadTimeConfig.DeadTime = 0; // Tiempo muerto entre canales complementarios (0 aquí).

sBreakDeadTimeConfig.BreakState = TIM\_BREAK\_DISABLE; // Deshabilita la función de "Break" (protección para motores).

HAL\_TIM\_PWM\_Init(&htim8); // Inicializa el timer en modo PWM.

HAL\_TIMEx\_MasterConfigSynchronization(...); // Aplica la configuración maestro/esclavo.

HAL\_TIM\_PWM\_ConfigChannel(...); // Configura el canal 1 del TIM8.

HAL\_TIMEx\_ConfigBreakDeadTime(...); // Aplica la configuración de dead time.

HAL\_TIM\_MspPostInit(&htim8); // Inicializa GPIOs asociados al PWM (definido en otro archivo).

**Duty Cycle**: Inicialmente 0% (Pulse = 0). Para cambiarlo, modifica htim8.Instance->CCR1 = X;

**Dead Time**: Útil en control de motores para evitar cortocircuitos, pero aquí está desactivado.

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2024 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes \*/

#include "main.h"

#include <stdint.h>

/\* Private defines \*/

#define NUM\_ADC 10

#define VAL\_MIN 200

#define VAL\_MAX 800

#define KP 15420

#define KD 53970

#define VELOCIDAD 40000

#define LINEA\_CENTRO 3500

#define LINEA\_MAX 7000

#define PWM\_MAX 65535

/\* Private variables \*/

ADC\_HandleTypeDef hadc1;

DMA\_HandleTypeDef hdma\_adc1;

TIM\_HandleTypeDef htim1, htim2, htim3, htim4;

/\* Global variables optimizadas \*/

volatile uint16\_t adcVal[NUM\_ADC];

int16\_t proporcional = 0, derivativo = 0, proporcional\_pasado = 0;

int16\_t position = 0, last\_value = 0;

/\* Function Prototypes \*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM1\_Init(void);

static void MX\_TIM2\_Init(void);

static void MX\_ADC1\_Init(void);

static void MX\_TIM4\_Init(void);

static void MX\_TIM3\_Init(void);

void PID(int16\_t velocidad, int16\_t Kp, int16\_t Kd, int16\_t LINEA);

void motores(int32\_t m\_izq, int32\_t m\_der);

void RGB(uint8\_t red, uint8\_t green, uint8\_t blue);

void ADC\_Select\_Channel(uint8\_t channel);

void sensado(void);

int16\_t posicion(void);

uint8\_t control(void);

/\* Main Function \*/

int main(void) {

HAL\_Init();

SystemClock\_Config();

/\* Initialize peripherals \*/

MX\_GPIO\_Init();

MX\_TIM1\_Init();

MX\_TIM2\_Init();

MX\_ADC1\_Init();

MX\_TIM4\_Init();

MX\_TIM3\_Init();

/\* Start Timers \*/

HAL\_TIM\_Base\_Start\_IT(&htim1);

HAL\_TIM\_Base\_Start\_IT(&htim2);

HAL\_TIM\_Base\_Start\_IT(&htim3);

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim4, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim4, TIM\_CHANNEL\_2);

HAL\_TIM\_PWM\_Start(&htim4, TIM\_CHANNEL\_3);

HAL\_TIM\_PWM\_Start(&htim4, TIM\_CHANNEL\_4);

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_12, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_13, GPIO\_PIN\_SET);

while (1) {

sensado();

GPIO\_PinState booleano = HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_15); // Read button state

int16\_t linea = posicion(); // Read sensor position

if (booleano == GPIO\_PIN\_SET) { // If button is pressed

if (linea != 0 && linea != LINEA\_MAX) {

proporcional = linea - LINEA\_CENTRO;

derivativo = proporcional - proporcional\_pasado;

proporcional\_pasado = proporcional;

int16\_t pwm\_pid = ((KP \* proporcional) + (KD \* derivativo)) / 3500;

motores(VELOCIDAD + pwm\_pid, VELOCIDAD - pwm\_pid);

} else {

motores((linea == 0) ? -VELOCIDAD / 2 : VELOCIDAD / 2,

(linea == 0) ? VELOCIDAD / 2 : -VELOCIDAD / 2);

}

} else {

motores(0, 0); // Stop motors if button is not pressed

}

// LED indication for different sensor readings

if (linea > 2500 && linea < 4500) {

RGB(0, 0, 255); // Blue (Centered)

} else if ((linea > 0 && linea <= 2500) || (linea >= 4500 && linea < LINEA\_MAX)) {

RGB(255, 0, 0); // Red (Off-centered)

}

}

}

/\* Optimized ADC Read \*/

void ADC\_Select\_Channel(uint8\_t channel) {

ADC\_ChannelConfTypeDef sConfig = {0};

sConfig.Channel = channel;

sConfig.Rank = 1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_480CYCLES;

if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK) {

Error\_Handler();

}

}

void sensado() {

for (uint8\_t i = 0; i < NUM\_ADC; i++) {

ADC\_Select\_Channel(ADC\_CHANNEL\_0 + i);

HAL\_ADC\_Start(&hadc1);

HAL\_ADC\_PollForConversion(&hadc1, 1);

adcVal[i] = HAL\_ADC\_GetValue(&hadc1);

HAL\_ADC\_Stop(&hadc1);

}

for (uint8\_t i = 0; i < NUM\_ADC; i++) {

adcVal[i] = (adcVal[i] <= VAL\_MIN) ? 50 : (adcVal[i] >= VAL\_MAX) ? 950 : adcVal[i];

}

}

/\* Get Position Function \*/

int16\_t posicion() {

int32\_t avg = 0, sum = 0;

uint8\_t on\_line = 0;

for (uint8\_t i = 0; i < 7; i++) {

int16\_t value = adcVal[i];

if (value > 300) {

avg += value \* (i \* 1000);

sum += value;

on\_line = 1;

}

}

return on\_line ? (avg / sum) : (last\_value < 3500 ? 0 : LINEA\_MAX);

}

/\* Motor Control Function \*/

void motores(int32\_t m\_izq, int32\_t m\_der) {

// Aplicamos los límites aquí para evitar sobrepasar el rango de PWM

m\_izq = (m\_izq > PWM\_MAX) ? PWM\_MAX : (m\_izq < -PWM\_MAX) ? -PWM\_MAX : m\_izq;

m\_der = (m\_der > PWM\_MAX) ? PWM\_MAX : (m\_der < -PWM\_MAX) ? -PWM\_MAX : m\_der;

// Configuración del PWM y dirección para el motor derecho

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_10, m\_der > 0);

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_11, m\_der <= 0);

TIM4->CCR1 = abs(m\_der);

// Configuración del PWM y dirección para el motor izquierdo

HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_14, m\_izq <= 0);

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_3, m\_izq > 0);

TIM4->CCR2 = abs(m\_izq);

}

/\* RGB Function \*/

void RGB(uint8\_t red, uint8\_t green, uint8\_t blue) {

TIM2->CCR1 = 65535 - green \* 257;

TIM4->CCR3 = 65535 - blue \* 257;

TIM4->CCR4 = 65535 - red \* 257;

}

//////////////////////////////////////////////////

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLM = 25;

RCC\_OscInitStruct.PLL.PLLN = 144;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief ADC1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_ADC1\_Init(void)

{

/\* USER CODE BEGIN ADC1\_Init 0 \*/

/\* USER CODE END ADC1\_Init 0 \*/

ADC\_ChannelConfTypeDef sConfig = {0};

/\* USER CODE BEGIN ADC1\_Init 1 \*/

/\* USER CODE END ADC1\_Init 1 \*/

/\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

\*/

hadc1.Instance = ADC1;

hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV2;

hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

hadc1.Init.ScanConvMode = ENABLE;

hadc1.Init.ContinuousConvMode = ENABLE;

hadc1.Init.DiscontinuousConvMode = DISABLE;

hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

hadc1.Init.NbrOfConversion = 1;

hadc1.Init.DMAContinuousRequests = ENABLE;

hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

{

Error\_Handler();

}

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_0;

// sConfig.Rank = 1;

// sConfig.SamplingTime = ADC\_SAMPLETIME\_3CYCLES;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_1;

// sConfig.Rank = 2;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_2;

// sConfig.Rank = 3;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_3;

// sConfig.Rank = 4;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_4;

// sConfig.Rank = 5;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_5;

// sConfig.Rank = 6;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_6;

// sConfig.Rank = 7;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_7;

// sConfig.Rank = 8;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_8;

// sConfig.Rank = 9;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

//

// /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

// \*/

// sConfig.Channel = ADC\_CHANNEL\_9;

// sConfig.Rank = 10;

// if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

// {

// Error\_Handler();

// }

/\* USER CODE BEGIN ADC1\_Init 2 \*/

/\* USER CODE END ADC1\_Init 2 \*/

}

/\*\*

\* @brief TIM1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM1\_Init(void)

{

/\* USER CODE BEGIN TIM1\_Init 0 \*/

/\* USER CODE END TIM1\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM1\_Init 1 \*/

/\* USER CODE END TIM1\_Init 1 \*/

htim1.Instance = TIM1;

htim1.Init.Prescaler = 35999;

htim1.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim1.Init.Period = 1999;

htim1.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim1.Init.RepetitionCounter = 0;

htim1.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim1) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM1\_Init 2 \*/

/\* USER CODE END TIM1\_Init 2 \*/

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = 0;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 65535;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim2);

}

/\*\*

\* @brief TIM3 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM3\_Init(void)

{

/\* USER CODE BEGIN TIM3\_Init 0 \*/

/\* USER CODE END TIM3\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM3\_Init 1 \*/

/\* USER CODE END TIM3\_Init 1 \*/

htim3.Instance = TIM3;

htim3.Init.Prescaler = 35999;

htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim3.Init.Period = 1999;

htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim3.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim3) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM3\_Init 2 \*/

/\* USER CODE END TIM3\_Init 2 \*/

}

/\*\*

\* @brief TIM4 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM4\_Init(void)

{

/\* USER CODE BEGIN TIM4\_Init 0 \*/

/\* USER CODE END TIM4\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM4\_Init 1 \*/

/\* USER CODE END TIM4\_Init 1 \*/

htim4.Instance = TIM4;

htim4.Init.Prescaler = 0;

htim4.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim4.Init.Period = 65535;

htim4.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim4.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

if (HAL\_TIM\_Base\_Init(&htim4) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim4) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim4, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_ConfigChannel(&htim4, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_ConfigChannel(&htim4, &sConfigOC, TIM\_CHANNEL\_3) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_ConfigChannel(&htim4, &sConfigOC, TIM\_CHANNEL\_4) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM4\_Init 2 \*/

/\* USER CODE END TIM4\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim4);

}

/\*\*

\* Enable DMA controller clock

\*/

//static void MX\_DMA\_Init(void)

//{

//

// /\* DMA controller clock enable \*/

// \_\_HAL\_RCC\_DMA2\_CLK\_ENABLE();

//

// /\* DMA interrupt init \*/

// /\* DMA2\_Stream0\_IRQn interrupt configuration \*/

// HAL\_NVIC\_SetPriority(DMA2\_Stream0\_IRQn, 0, 0);

// HAL\_NVIC\_EnableIRQ(DMA2\_Stream0\_IRQn);

//

//}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_13|GPIO\_PIN\_14, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_12|GPIO\_PIN\_3, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_10|GPIO\_PIN\_11|GPIO\_PIN\_12, GPIO\_PIN\_RESET);

/\*Configure GPIO pins : PC13 PC14 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_13|GPIO\_PIN\_14;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\*Configure GPIO pin : PC15 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_15;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\*Configure GPIO pins : PB12 PB3 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_12|GPIO\_PIN\_3;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\*Configure GPIO pin : PB15 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_15;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\*Configure GPIO pins : PA8 PA9 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_8|GPIO\_PIN\_9;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\*Configure GPIO pins : PA10 PA11 PA12 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_10|GPIO\_PIN\_11|GPIO\_PIN\_12;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

/\*if (htim->Instance == TIM3){

//contador++;

//if (bandera==0){TIM4->CCR2=4294967295/2;bandera=1;}

//if (bandera==1){TIM4->CCR2=4294967295;bandera=2;}

//if (bandera==2){TIM4->CCR2=0;bandera=0;}

if (TIM2->CCR1<=64000){TIM2->CCR1+=1000;}

if (TIM2->CCR1>64000){TIM2->CCR1=0;}

//if (TIM2->CCR1<=4294960000){TIM2->CCR1+=1000;}

//if (TIM2->CCR1>4294960000){TIM2->CCR1=0;}

if (TIM4->CCR1<=64000){TIM4->CCR1+=1000;}

if (TIM4->CCR1>64000){TIM4->CCR1=0;}

}\*/

}

/\* USER CODE END 4 \*/

//void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef \*hadc)

//{

// TIM4->CCR1=adcVal[0]\*16;

//}

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/