

# Trabajo de Micros:

## Barrera de garaje

| INTEGRANTES DEL GRUPO    |           |
|--------------------------|-----------|
| APELLIDOS, NOMBRE        | MATRICULA |
| Marianini Rios, Gregorio | 55963     |
| Alonso Geijo, Javier     | 55714     |
| Vicente Gordón, Luis     | 56147     |

|                  |                 |
|------------------|-----------------|
| GRUPO TEORIA     | A404            |
| GRUPO DE TRABAJO | 21              |
| PROFESOR         | ALBERTO BRUNETE |

## Índice

|                               |   |
|-------------------------------|---|
| 1. Introducción.....          | 3 |
| 2. Sensores y actuadores..... | 3 |
| 3. Funcionalidad.....         | 4 |
| 4. Programación.....          | 5 |
| 5. Conclusión.....            | 8 |

## 1. Introducción



Un microcontrolador es un circuito integrado que concentra todos los componentes de un computador, diseñado para controlar tareas específicas. Su tamaño compacto permite integrarlo en el dispositivo que gobierna, lo que lo define como un "controlador incrustado" (embedded controller). Se le considera una "solución en un chip" porque reduce la cantidad de componentes y costos.



Inicialmente, los microcontroladores eran microprocesadores con memoria integrada, como RAM y ROM. Con el tiempo, evolucionaron para abarcar dispositivos diseñados para aplicaciones embebidas en automóviles, teléfonos móviles y electrodomésticos.

Hoy en día, existen marcas reconocidas como Intel, Motorola, Microchip, Philips, National y Atmel. En este contexto, utilizaremos el microcontrolador STM32F411E-DISCO de la familia STM32, basada en núcleos ARM Cortex-M de 32 bits. Esta familia incluye modelos desde M0 hasta M7, con diversas capacidades de memoria, periféricos y prestaciones, adecuados tanto para proyectos hobby como para aplicaciones profesionales avanzadas.



## 2. Sensores y actuadores

| Sensores   |   |
|--|---|
| <b>Sensor de ultrasonidos HC-SR04:</b> El funcionamiento se basa en emitir un sonido ultrasónico desde uno de sus transductores y esperar a que este rebote en un objeto. El eco resultante es captado por el segundo transductor, y la distancia se calcula en función del tiempo que tarda el eco en regresar. |   |
| <b>LDR:</b> Es un componente electrónico utilizado como sensor de luz, también conocido como fotorresistencia. Su resistencia eléctrica es muy alta en condiciones de oscuridad o con poca iluminación.  |  |

| Actuadores   |   |
|--|---|
| <b>LEDs:</b> Un LED es un diodo emisor de luz, es decir, un tipo particular de diodo que emite luz al ser atravesado por una corriente eléctrica.  |  |
| <b>Servomotor:</b> Un servomotor es un motor de corriente continua diseñado para girar a un ángulo específico en respuesta a una señal de control y mantenerse fijo en esa posición. Esta señal de control se envía a través de los pines digitales PWM. |   |

### 3.Funcionalidad

El sistema desarrollado controla una barrera de un garaje con una STM32F411. Este sistema permite gestionar el movimiento de la barrera mediante dos sensores de ultrasonidos HC-SR04 que detectan el vehículo que se aproxima. Gracias a estos sensores la barrera se abre, también se les ha incorporado un semáforo a ambos lados de la barrera para indicar si pueden pasar o no, para que en caso de que la barrera no se abra que no haya ningún accidente. Otra funcionalidad más que le hemos añadido es un led en el garaje que se enciende cuando la luz disminuye hasta cierto rango medido por un LDR.

Para todas estas funciones se utilizó interrupciones las cuales permiten responder de inmediato a eventos como cambios de flanco, liberando al procesador para otras tareas mientras se espera el próximo evento. Además, garantizan una medición precisa al capturar el valor exacto del temporizador en el momento de la interrupción. Sin ellas, sería necesario usar un bucle de espera para monitorear constantemente el estado del pin ECHO, lo que consumiría muchos recursos del microcontrolador.

También hemos utilizado DMA que nos resulta muy útil porque libera a la CPU para que se concentre en tareas críticas, como la captura de tiempos de los sensores ultrasónicos o el control del motor, reduciendo el consumo de energía al evitar interrupciones constantes para manejar el ADC. Además, garantiza datos actualizados en tiempo real al mantener el adc\_buffer con los últimos valores del ADC sin retrasos. Sin el DMA, sería necesario configurar el ADC para generar interrupciones tras cada conversión, implementar una rutina de interrupción para capturar los valores y copiarlos manualmente, lo que resultaría menos eficiente y podría saturar la CPU, complicando el manejo de temporizadores, interrupciones y cálculos.

Para la configuración del periodo hemos optado por configurar el período del temporizador en 0xFFFF ya es ideal para aplicaciones de Input Capture porque aprovecha al máximo el rango de un temporizador de 16 bits, permitiendo medir intervalos de tiempo largos sin preocuparse por desbordamientos frecuentes. Esto simplifica el código al minimizar la necesidad de manejar desbordamientos y asegura una medición eficiente y precisa dentro del rango completo del temporizador.

## 4.Programación

```
42 /* Private variables ----- */
43 ADC_HandleTypeDef hadc1;
44 DMA_HandleTypeDef hdma_adc1;
45
46 TIM_HandleTypeDef htim2;
47 TIM_HandleTypeDef htim3;
48
49 /* USER CODE BEGIN PV */
50 uint32_t adc_buffer[1]; // Buffer para almacenar los valores de ADC
51
52 TIM_HandleTypeDef htim2;
53 TIM_HandleTypeDef htim3;
54
55 /* USER CODE BEGIN PV */
56
57 /* USER CODE END PV */
58
59 /* Private function prototypes ----- */
60 void SystemClock_Config(void);
61 static void MX_GPIO_Init(void);
62 static void MX_DMA_Init(void);
63 static void MX_TIM3_Init(void);
64 static void MX_TIM2_Init(void);
65 static void MX_ADC1_Init(void);
66 /* USER CODE BEGIN PFP */
67
68 /* USER CODE END PFP */
69
70 /* Private user code ----- */
71 /* USER CODE BEGIN 0 */
72 volatile uint32_t valor1 = 0, valor2 = 0, periodo = 0;
73 volatile uint8_t es_pv = 0;
74 int distancia = 0;
75
76 volatile uint32_t valor1s = 0, valor2s = 0, periodos = 0;
77 volatile uint8_t es_pvs = 0;
78 int distanciaSalida = 0;
79
80 int angulo=0;
81
82 /**
83  * @brief Callback de interrupción para la captura de pulsos de los timers.
84  * @param htim Puntero al handler del timer que genera la interrupción.
85  */
86
87 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
88     if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4) {
89         if (es_pv == 0) {
90             valor1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Captura el primer valor (flanco de subida)
91             es_pv = 1;
92             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_FALLING); // Cambia al flanco de bajada
93         } else {
94             valor2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // Captura el segundo valor (flanco de bajada)
95             __HAL_TIM_SET_COUNTER(htim, 0); // Reinicia el contador del timer
96             if (valor2 > valor1) {
97                 periodo = valor2 - valor1; // Calcula el periodo.
98             }
99             es_pv = 0;
100             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4, TIM_INPUTCHANNELPOLARITY_RISING); // Vuelve a cambiar al flanco de subida
101         }
102     } else if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3) {
103         if (es_pvs == 0) {
104             valor1s = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3);
105             es_pvs = 1;
106             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_3, TIM_INPUTCHANNELPOLARITY_FALLING);
107         } else {
108             valor2s = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3);
109             __HAL_TIM_SET_COUNTER(htim, 0);
110             if (valor2s > valor1s) {
111                 periodos = valor2s - valor1s;
112             }
113             es_pvs = 0;
114             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_3, TIM_INPUTCHANNELPOLARITY_RISING);
115         }
116     }
117 }
118 /**
119  * @brief Genera un pulso para activar los sensores ultrasonido de entrada y salida.
120  */
121 void Trigger_HCSR04(void) {
122     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
123     HAL_Delay(10);
124     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET);
125 }
126 void Trigger_HCSR04Salida(void) {
127     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
128     HAL_Delay(10);
129     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
130 }
131 /**
132  * @brief Calcula la distancia detectada por los sensores de entrada y salida.
133  */
134
135 void CalcularDistancia(void) {
136     distancia = periodo / 58;
137 }
138 void CalcularDistanciaSalida(void) {
139     distanciaSalida = periodos / 58;
140 }
```

```

142 * @brief Mueve el motor a un angulo especifico.
143 * @param angulo Angulo al que se desea mover el motor (0-180 grados).
144 */
145
146 void mover_motor(int angulo) {
147     int pulso_min = 500;
148     int pulso_max = 2500;
149
150     int pulso = pulso_min + ((pulso_max - pulso_min) * angulo) / 180;
151
152     int periodo_timer = 20000; // Periodo del PWM en microsegundos porque el motor funciona a 50Hz
153     int compare_value = (pulso * _HAL_TIM_GET_AUTORELOAD(&htim2)) / periodo_timer; // Calcula el valor de comparación
154
155     _HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, compare_value); // Actualiza el valor del pulso
156 }
157
158
159
160 /* USER CODE END 0 */
161
162 int main(void)
163 {
164     /* USER CODE BEGIN 1 */
165
166     /* USER CODE END 1 */
167
168     /* MCU Configuration-----*/
169
170     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
171     HAL_Init();
172
173     /* USER CODE BEGIN Init */
174
175     /* USER CODE END Init */
176
177     /* Configure the system clock */
178     SystemClock_Config();
179
180     /* USER CODE BEGIN SysInit */
181
182     /* USER CODE END SysInit */
183
184     /* Initialize all configured peripherals */
185     MX_GPIO_Init();
186     MX_DMA_Init();
187     MX_TIM3_Init();
188     MX_TIM2_Init();
189     MX_ADC1_Init();
190
191     /* USER CODE BEGIN 2 */
192     HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_4); // Habilita la captura de entrada con interrupción (sensor de entrada)
193     HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_3); // Habilita la captura de entrada con interrupción (sensor de salida)
194     HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // Inicia el PWM para el motor
195     HAL_ADC_Start_DMA(&hadc1, adc_buffer, 1); // Inicia la conversión ADC con DMA
196
197     /* USER CODE END 2 */
198     /* Infinite loop */
199     /* USER CODE BEGIN WHILE */
200     while (1)
201     {
202         /* USER CODE END WHILE */
203
204         /* USER CODE BEGIN 3 */
205         if (adc_buffer[0] < 1450)
206         {
207             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); // Encender LED
208         }
209         else
210         {
211             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); // Apagar LED
212         }
213         Trigger_HCSR04();
214         Trigger_HCSR04Salida();
215         CalcularDistancia();
216         CalcularDistanciaSalida();
217         if (distancia < 25) {
218             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
219             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
220             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_3, GPIO_PIN_RESET);
221             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_RESET);
222             mover_motor(90);
223             HAL_Delay(3000);
224         } else {
225             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
226             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
227             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_3, GPIO_PIN_SET);
228             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET);
229             mover_motor(0);
230             if (distanciaSalida < 25) {
231                 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
232                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN SET);
233                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_3, GPIO_PIN RESET);
234             }
235             if (distanciaSalida < 25) {
236                 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN SET);
237                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN SET);
238                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_3, GPIO_PIN RESET);
239                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN RESET);
240                 mover_motor(90);
241                 HAL_Delay(3000);
242             } else {
243                 HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN RESET);
244                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN RESET);
245                 mover_motor(0);
246             }
247         }
248     }
249     /* USER CODE END 3 */
250 }

```

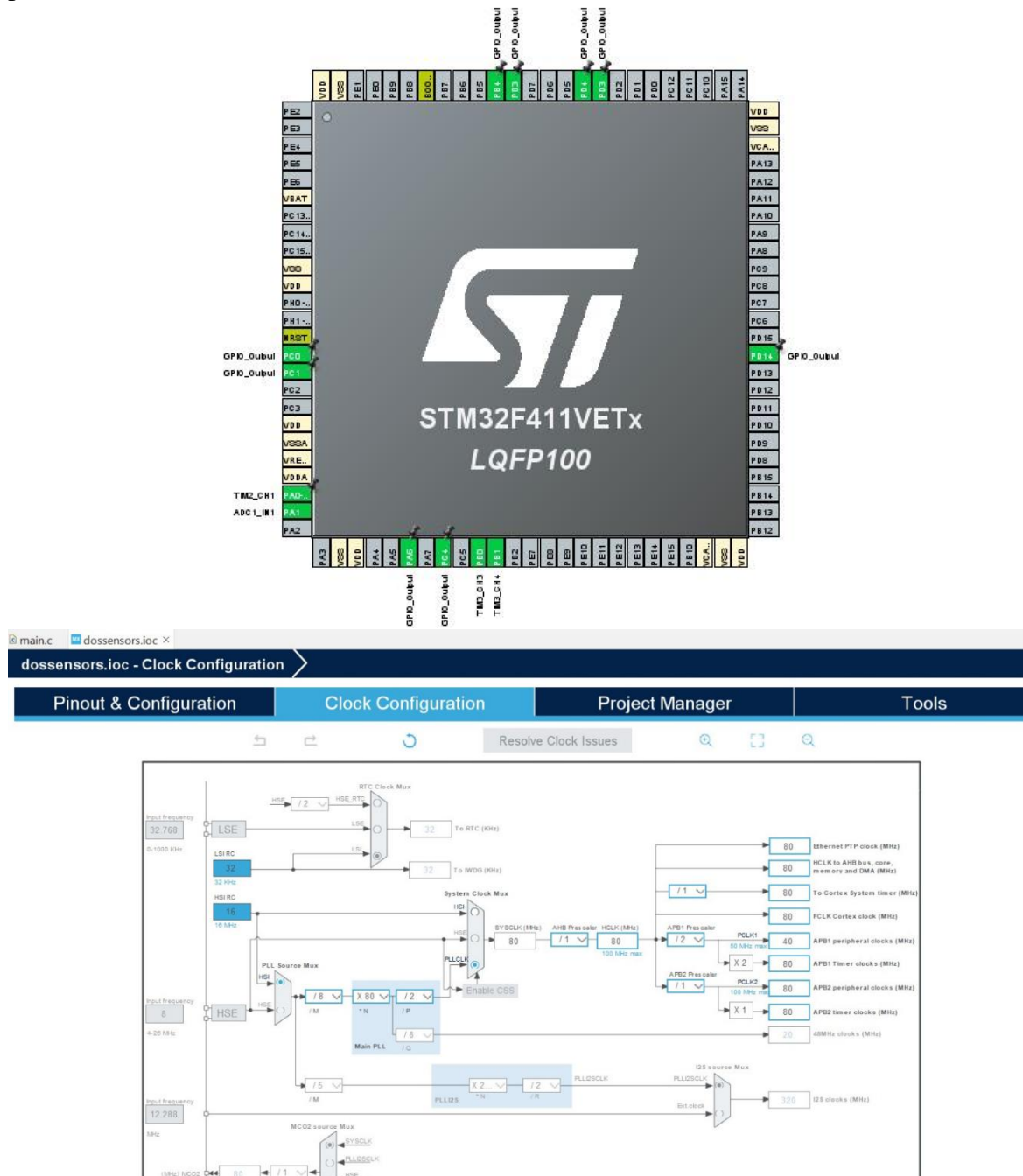
El código desarrollado es el anterior mostrado, en él se puede ver diferentes funciones. Para medir los ultrasonidos, primero es necesario configurar un temporizador que registre el tiempo de captura, determinar el pin del trigger y activar la interrupción del temporizador. En la rutina de interrupción, se captura el momento en que se emite el ultrasonido y se cambia la polaridad del impulso para



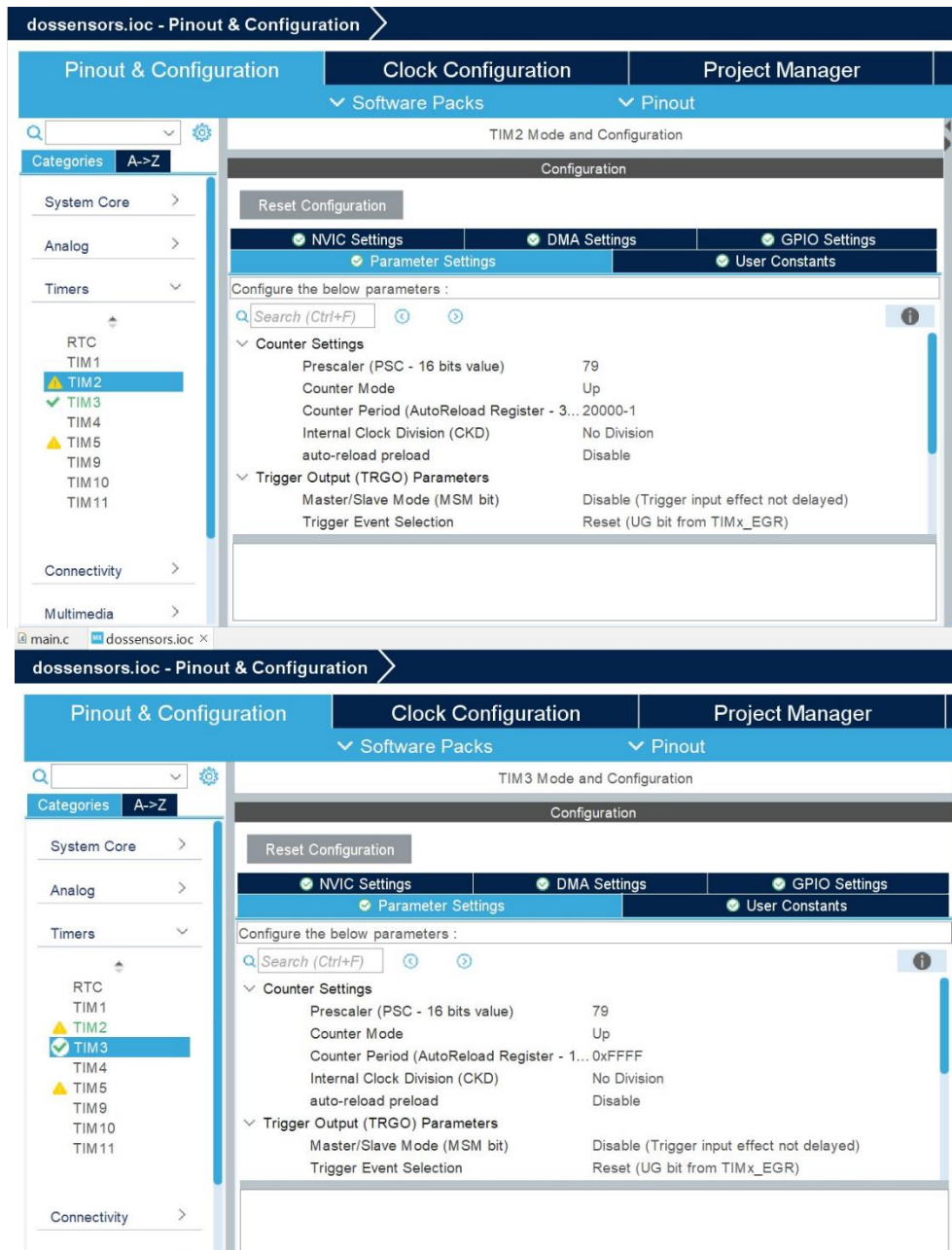
preparar la recepción del eco. Cuando el sonido regresa y se detecta el flanco de bajada, la diferencia entre los periodos se almacena en la variable periodo, y se ajusta nuevamente la polaridad para la siguiente medición. Con la variable periodo se calcula la distancia del objeto al sensor y si es menor que 25 centímetros la barrera se abre y los cambian según el sensor que se active. Para activar el trigger de los sensores se crea una función TRIGGER\_HCSR04 que es la encargada de esa función. Además, se crea la función MOVER\_MOTOR que como su nombre indica es la encargada de accionar el motor.

Una vez dentro del main en el while(1) se enciende el led del LDR cuando el valor menor que 1450, después se llama a las funciones anteriormente nombradas y según la distancia del vehículo a el sensor se activan los LEDs correspondientes y el motor.

Los pines utilizados y la configuración del reloj para este código son los siguientes:



La configuración de los dos timmers son las siguientes:



## 5.Conclusión

Este proyecto ofrece una solución completa y funcional que integra sensores ultrasónicos, control de motores y procesamiento de señales a través de un microcontrolador STM32, con el objetivo de medir distancias y ejecutar acciones automatizadas según las condiciones detectadas. La implementación efectiva de tecnologías avanzadas, como el uso de DMA, temporizadores en modo Input Capture y PWM, asegura un rendimiento óptimo, permitiendo que la CPU se libere para gestionar múltiples tareas de manera simultánea. Gracias a este enfoque, el sistema puede responder de manera eficiente y precisa a las mediciones y controles necesarios.

El diseño modular del código contribuye a una alta precisión en la medición de distancias utilizando los sensores HC-SR04, mientras que el control del servomotor se realiza de forma fluida y estable.



Esta combinación lo convierte en una herramienta ideal para aplicaciones en automatización, robótica y sistemas de seguridad. Además, el uso del ADC para monitorear señales analógicas amplía la flexibilidad y la versatilidad del sistema, permitiendo su adaptación a diversas necesidades.

En resumen, este proyecto demuestra la efectividad de integrar hardware y software de manera eficiente para resolver problemas del mundo real. No solo optimiza los recursos del microcontrolador STM32, sino que también aprovecha al máximo las capacidades de sus periféricos, creando una base sólida para futuras mejoras o adaptaciones en aplicaciones más complejas.

[https://github.com/gmarianini/Micros\\_puertagaraje](https://github.com/gmarianini/Micros_puertagaraje)