DESARROLLO PROGRAMA EMBEBIDO CON MICROCONTROLADOR STM ORIENTADO AL CONTROL DE UN MOTOR TRIFASICO

IGNACIO ÁLVAREZ

MARCO TABORDA CARDONA

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

TECNOLOGÍA ELECTRONICA

ANTIOQUIA

2025

DESARROLLO PROGRAMA EMBEBIDO CON MICROCONTROLADOR STM ORIENTADO AL CONTROL DE UN MOTOR TRIFASICO

IGNACIO ÁLVAREZ

MARCO TABORDA CARDONA

Trabajo de grado para optar al título de Tecnólogo en ELECTRONICA

Asesor

Julio Pastor Restrepo Zapata

Magister en energía

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

FACULTAD DE INGENIERÍA

TECNOLOGÍA ELECTRONICA

ANTIOQUIA

2025

Contenido

	Pág.
Contenido	3
Lista de figuras	5
Lista de anexos	6
Resumen	7
Abstract	8
Glosario	9
Introducción	10
1. Planteamiento del problema	11
1.1 Descripción	11
1.2 Formulación	12
2. Justificación	13
3. Objetivos	14
3.1 Objetivo general	14
3.2 Objetivos específicos	14
4. Marco teórico	15
4.1 Microcontrolador STM32F302R8:	15
4.1.1 Características:	16
4.1.2 Escudo X-NUCLEO-IHM091:	19
4.1.2. Funciones	19
4.1.3 Modulo de potencia STEVAL-IHM028V1:	20
4.2 Motor trifásico:	21
4.2.1 Estator:	21
4.2.2 Rotor:	22
4.2.3 Escudos/carcasa:	22
4.3 Inversores tradicionales:	23
8.3.1 Convertidor de fase estáticos:	23
4.3.2 Convertidor de fase rotativa:	24
4.3.3 Convertidor de fase electrónico o variador de frecuencia:	24
5. Metodología	27

5.1 Tipo de proyecto 27
5.2 Método 27
5.2 Instrumentos de recolección de información 27
5.2.1 Fuentes primarias. 27
6. Resultados 28
7. Conclusiones 36
8. Recomendaciones 37
9. Referencias bibliográficas 39
10. Bibliografía 40
11. Anexos 41

Lista de figuras

Pág.

Figura 1. Microcontrolador SMT32F302R8 16
Figura 2. Conectores STM32F302R8 18
Figura 3. Núcleo F302R8 18
Figura 4. Escudo X-NUCLEO-IHM09M1 19
Figura 5. STEVAL-IHM023V3 20
Figura 6.Motor Trifásico y sus partes 21
Figura 7. Convertidor de fase estático (Capacitores) 23
Figura 8. Motor generador 24
Figura 9. Convertidor de fase Electrónico 25
Figura 10. Convertidor de fase electrónico y digital 26
Figura 11. Definición de variables 30
Figura 12. Void setup / pinMode 31
Figura 13. Void loop / digitalWrite / swicht / case 0 – 3 32
Figura 14. Void loop / digitalWrite / swicht / case 4 – 5 32
Figura 15. Muestra osciloscopio Ah-Bh 33
Figura 16. Muestra osciloscopio Bh-Ch 33
Figura 17. Muestra osciloscopio Ah-Al 34
Figura 18. Muestra osciloscopio Bh-Bl 34
Figura 19. Muestra osciloscopio Ch-Cl 35
Figura 20. Montaje final 35

Lista de anexos

Pag.

Anexo A. Primer contacto con el software STM32cubeIDE 41

Anexo B. Generar 2 PWM con stm32 53

Anexo C. MC WORKBENCH 72

Resumen

DESARROLLO PROGRAMA EMBEBIDO CON MICROCONTROLADOR STM ORIENTADO AL CONTROL DE UN MOTOR TRIFASICO

IGNACIO ÁLVAREZ
MARCO TABORDA CARDONA

En el presente proyecto se desarrollará un programa embebido con microcontrolador STM 32 orientado al control de un motor trifásico de inducción esto con un módulo académico ya armado e instalado con sus elementos eléctricos y electrónicos tales como: microcontrolador STM 32, tarjetas de potencia, variadores de frecuencia y convertidor de fase estática(capacitores), se adaptara un código de programación del microcontrolador STM32F302R8 para el control de encendido y de velocidad de un motor trifásico de inducción . El objetivo consiste en poder adaptar un código de programación con un motor de inducción y su variedad en su potencia, en evaluar el comportamiento de arranque y estabilidad ya cuando el motor este girando estable. Esta evaluación quedará registrada en el laboratorio de electrónica de la Institución Universitaria Pascual Bravo y estará como guía y modelo para los estudiantes interesados en la automatización del control del motor trifásico en conjunto con la electrónica y será más adaptable a entornos industriales y pequeñas empresas donde las condiciones pueden variar en la conexión del motor.

Como resultado de la realización del proyecto, se ha desarrollado el programa de bajo costo mediante el préstamo del módulo académico, motor trifásico y otros elementos eléctricos prestados por los laboratorios de la Institución Universitaria Pascual Bravo, se utilizará las laptops para la adaptación de los códigos de programación del STM32F302 para el control de encendido y de velocidad de un motor trifásico de inducción El resultado del proyecto se logró hacer un trabajo investigativo y de programación con el comportamiento de corriente alterna y directa al automatizar el microcontrolador STM32F302 para el control de un motor trifásico de inducción y así protegiendo y guardando la seguridad de los diferentes elementos eléctricos y electrónicos y de las personas o estudiantes que se encuentren haciendo diferentes prácticas.

Palabras claves: académico, micro controlador, programación, trifásico, control

Abstract

DEVELOPMENT OF EMBEDDED PROGRAM WITH STM MICROCONTROLLER ORIENTED TO THE CONTROL OF A THREE-PHASE MOTOR

Ignacio Alvarez

Marco Taborda Cardona

In this project, an embedded program will be developed using an STM 32 microcontroller for controlling a three-phase induction motor. This program will be used with an already assembled and installed academic module with its electrical and electronic components, such as an STM 32 microcontroller, power boards, frequency converters, and a static phase converter (capacitors). A programming code for the STM32F302R8 microcontroller will be adapted to control the start and speed of a three-phase induction motor. The objective is to adapt the programming code for the STM 32 microcontroller to an induction motor and its power range, and to evaluate its starting behavior and stability when the motor is running steadily. This evaluation will be recorded in the electronics laboratory at Pascual Bravo University Institution and will serve as a guide and model for students interested in automating three-phase motor control in conjunction with electronics. It will be more adaptable to industrial environments and small businesses where motor connection conditions may vary.

As a result of the project, the low-cost program was developed through the loan of the academic module, three-phase motor, and other electrical components from the laboratories of the Pascual Bravo University Institution. Personal laptops will be used to adapt the programming codes of the STM32F302 microcontroller to control the ignition and speed of a three-phase induction motor. The result of the project was research and programming work using alternating and direct current behavior by automating the STM32F302 microcontroller to control a three-phase induction motor, thus protecting and safeguarding the safety of the various electrical and electronic components, as well as the individuals or students performing the various practical work.

Keywords: academic, microcontroller, programming, three-phase, control

Glosario

Arduino IDE: entorno de desarrollo integrado IDE gratuito y de código abierto para programar placas Arduino y de otra marca como el STM32. Incluye herramientas para escribir, copilar y cargar código en el microcontrolador. Ideal para principiantes por su interfaz sencilla.

Código embebido: programa diseñado específicamente para ejecutarse en dispositivos como microcontroladores. Esta optimizado para tareas como leer sensores o activar motores y suele escribirse en lenguaje como C o C++.

Código embebido: programa diseñado específicamente para ejecutarse en dispositivos como microcontroladores. Esta optimizado para tareas como leer sensores o activar motores y suele escribirse en lenguaje como C o C++.

Electrónica: campo que estudia y aplica sistemas que controlan el flujo de electrones en circuitos. Incluye componentes como resistencias, transistores o microchips, se usa para diseñar dispositivos como teléfonos, computadores, sistemas de automatización y microcontroladores.

Microcontrolador: dispositivo electrónico que integra en un solo chip un procesador, memoria, puertos de entrada y salida. Se utiliza para controlar tareas específicas en sistemas embebidos, como robots y dispositivos automáticos. Ejemplos comunes el Arduino y STM32.

Motor trifásico: motor que funciona con corriente alterna (CA) que se alimenta de tres fases de energía simultáneamente. Es eficiente, potente y se usa en aplicaciones industriales como bombas y ventiladores.

STM32cubeIDE: herramienta de desarrollo creada por STMicroelectronics para programar microcontroladores STM32. Combina funciones avanzada de edición de código, depuración y configuraciones de hardware.

Introducción

El tema de trabajo es realizar un código embebido para el control de un motor trifásico. El trabajo se realiza por un motivo, que es incentivar a los estudiantes de la institución universitaria Pascual Bravo a conocer un poco sobre cómo realizar un código embebido con un microcontrolador en este caso el STM32F302R8 en los diferentes softwares, ya sea Arduino IDE o STM32cubeIDE. El trabajo se pensó para que los estudiantes, se empiecen a interesar por la electrónica y la utilidad de conocer las diferentes funciones de los microcontroladores. El método es seguir las indicaciones de los diferentes profesores que nos acompañan en el proceso y diferentes fuentes de información en internet, para poder hacer un código entendible y que cumpla la función del control del motor. La limitación son el tiempo, el material para realizar el proyecto ya que son costosas y el desarrollo del código.

1. Planteamiento del problema

1.1 Descripción

El control de motores trifásicos es fundamental en diversas aplicaciones industriales debido a su eficiencia y estabilidad en el suministro de energía. El control de estos motores suele ser complejo, ya que involucra la gestión de tres fases de corriente que deben sincronizarse para evitar problemas sobrecalentamientos o perdida de rendimiento.

El problema que se encontró es que las pequeñas empresas están adquiriendo motores trifásicos y están establecidos en zonas residenciales donde la distribución energética es monofásica.

Los sistemas inversores de señal tradicionales de control suelen ser limitados en cuanto a precisión y adaptabilidad, lo cual puede llevar a un consumo energético ineficiente y a un mayor desgaste de los componentes mecánicos y eléctricos.

El avance en la tecnología de microcontroladores, especialmente en la familia STM, permite implementar soluciones de control embebido que son más eficientes. Sin embargo, desarrollar un programa embebido que pueda manejar el control de un motor trifásico de forma óptima y adaptable a diferentes condiciones de operación sigue siendo un desafío técnico. Este proyecto busca abordar esta problemática mediante el diseño de un programa embebido que permita gestionar eficazmente un motor trifásico, mejorando así su rendimiento y reduciendo los costos de operación y mantenimiento.

1.2 Formulación

¿Cómo desarrollar un programa embebido basado en un microcontrolador STM32F302R8 que permita el control eficiente y preciso de un motor trifásico, optimizando su rendimiento en aplicaciones industriales y comerciales?

2. Justificación

El control preciso de motores trifásicos es crucial en aplicaciones industriales y pequeñas empresas, ya que permite mejorar la eficiencia energética. Es posible ajustar el rendimiento del motor en función de las necesidades específicas de la aplicación, reduciendo así el consumo de energía y costo operativos. La automatización con un microcontrolador permite reducir la necesidad de sistemas de control externos complejos y costosos. Al implementar el control directamente en el microcontrolador STM32F302R8, se puede disminuir costos de implementación, y facilitar el mantenimiento. Permite programar y adaptar el sistema a múltiples configuraciones de control de motor. Esto es especialmente útil en entornos industriales y en pequeñas empresas donde las condiciones pueden variar y se necesita adaptar rápidamente al funcionamiento del motor. Monitorear continuamente el estado del motor implementando sensores, detectando posibles fallas antes de que se conviertan en problemas mayores. El uso de microcontroladores STM en el control de motores es una tecnología de vanguardia que promueve la innovación en el desarrollo de sistemas de automatización. En la industria demanda sistemas de control cada vez más avanzados y accesibles, lo que hace el desarrollo de soluciones embebidas sea para el control de motores.

3. Objetivos

3.1 Objetivo general

Desarrollar un programa embebido para un inversor trifásico con microcontrolador STM 32F302R8 orientado al control de un motor trifásico de inducción ubicada en el bloque 10 de la I.U Pascual Bravo.

3.2 Objetivos específicos

- Adaptar el código de programación del microcontrolador STM32F302 para el control de encendido y de velocidad de un motor trifásico de inducción ubicado en el bloque 10 de la I.U Pascual Bravo.

- Conocer todos los elementos eléctricos y electrónicos para el funcionamiento y control de un motor trifásico de inducción por medio del microcontrolador STM32F302R8

- Explicar cómo se generó el código para el control del motor trifásico con Arduino IDE o STM32cubeIDE.

- Analizar los comportamientos del pwm al automatizar el microcontrolador STM32F302 para el control de un motor trifásico de inducción.

- Realizar muestreo del desface de 120 grados de las 3 señales PWM para el control del motor trifásico.

4. Marco teórico

4.1 Microcontrolador STM32F302R8:

Un STM32 es una familia de microcontroladores desarrollada por STMcroelectronics. Estos microcontroladores están basados en la arquitectura ARM cortex-M y son ampliamente utilizados en aplicaciones embebidas debido a su versatilidad, potencia y eficiencia energética. Los STM32 son muy utilizados en proyectos de control de motores, sistemas de sensores, dispositivos de internet de las cosas (loT) y aplicaciones de consumo industrial.

- Arquitectura ARM Córtex-M: Utilizan núcleos de la familia ARM Cortex-M (M0,M3,M4,M7)lo que permite una gran variedad de conexiones en términos de potencia de procesamiento y consumo de energía.
- Amplia Gamas de periféricos: incluyen una variedad de periféricos integrados, como convertidores analógicos-digitales (ADC), temporizadores, controladores de comunicación (UART, SPI, I2C, CAN, USB) y en algunos modelos avanzados, controladores de ethernet.
- Rango de aplicaciones: Los STM32 están diseñados para ser versátiles por, lo que son adecuados para aplicaciones desde tareas sencillas hasta procesos de tiempo real que requieren control complejo, como el manejo de motores trifásicos.
- Consumo de Energía controlada: Disponen de modos de bajo consumo de energía, lo cual es ideal para dispositivos alimentados por batería o aplicaciones que deben optimizar el consumo energético.
- Herramientas de desarrollo: STMicroelectronics ofrece herramientas de desarrollo y entornos integrados (IDE) como STM32CubeIDE y STM32CubeMX, que facilitan la configuración y programación de estos microcontroladores.
- Soporte para aplicaciones de control: Los STM32 son comunes en el desarrollo de sistemas de control, con en este proyecto, donde se utiliza para controlar un motor trifásico. (Chat gpt, 2024)

En este caso utilizaremos el STM32F302 para el control del motor trifásico.

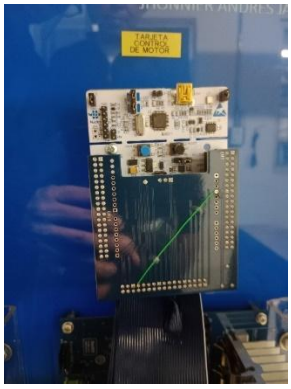


Figura 1. Microcontrolador SMT32F302R8
Fuente: La imagen es propia.

4.1.1 Características:

1. Núcleo: CPU ARM Cortex-M4 de 32 bits con FPU (72 MHz max), multiplicación de ciclo único y división de HW, instrucción DSP y MPU (unidad de protección de memoria).
2. Rango de voltaje VDD, VDDA: 2.0V a 3.6V.
3. Memoria: 128 a 256 Kbytes de memoria Flash.
4. Hasta 40Kbytes de SRAM, con verificación de paridad de HW implementada en los primeros 16 Kbytes.
5. Unidad de cálculo CRC
6. Gestión de reinicio y suministro.
7. Reinicio de on/off (POR/PDR).
8. Detector de voltaje programable (PVD).
9. Modos de bajo consumo: suspensión, detención y espera.
10. Alimentación VBAT para RTC y registros de respaldo.
11. Gestión de reloj: oscilador de cristal de 4 a 32 MHz.
12. Oscilador de 32 KHz para RTC con calibración.
13. RC interno de 8 MHz con opción x 16 PLL.
14. Oscilador interno de 40kHz.
15. Hasta 87 E/S rápidas.
16. Todas asignables a vectores de interrupción externos.
17. Varios tolerantes a 5V.

18. Matriz de interconexión.
19. Controlador DMA de 12 canales.
20. Dos ADC de 0.20uS (hasta 17 canales) con resoluciones seleccionable de 12/10/8/6 bits, rango de conversión de 0 a 3.6 V, entrada diferencial de extremo único, suministro analógico separado de 2 a 3.6 V.
21. Un canal DAC de 12 btis con suministro analógico de 2.4 a 3.6 V.
22. Cuatro comparadores analógicos rápidos de riel con suministro analógico de 2 a 3.6 V.
23. Dos amplificadores operacionales que se pueden usar en modo PGA, todos los terminales accesibles con suministros analógicos de 2.4 a 3.6 V.
24. Hasta 24 canales de detección capacitiva que admiten sensores táctiles lineales, rotativos y de tecla táctil .
25. Hasta 11 temporizadores.
26. Un temporizador de 32bits y dos temporizadores de 16 bits con hasta 4 IC/OC/PWM o contador de pulsos y entrada de codificador de cuadratura(incremental).
27. Un temporizador de control avanzado de 16 bits y 6 canales, con hasta 6 canales PWM, generación de tiempo muerto y parada de emergencia.
28. Un temporizador de 16 bits con 2 IC/OC, 1 OCN/PWM, generación de tiempo muerto y parada de emergencia.
29. Dos temporizadores de 16 bits con IC/OC/OCN/PWM, generación de tiempo muerto y parada de emergencia.
30. Dos temporizadores de vigilancia (independientes, ventana) temporizador.
31. SysTick: contador regresivo de 24bits.
32. Un temporizador básico de 16 bits para controlar el DAC.
33. Calendario RTC con alarma, activación periodica desde Stop/Stanby.
34. Interfaces de comunicación.
35. CAN (2.0B activa).
36. Dos I2C Fast mode plus (1 Mbit/s) con disipador de corriente de 20mA, SMBus/PMBus, activación desde STOP.
37. Hasta cinco USART/UART (interfaz ISO 7816, LIN, IrDA, control de modem).
38. Hasta tres SPI, dos con interfaz I2S multiplezado hal/full duplez, 4 a 16tramas de bits programables.

39. Interfaz USB 2.0 de velocidad completa.
40. Transmisor infrarrojo.
41. Depuración de cable serial, Cortex-M4 con FPU ETM, JTAG.
42. Identificación única de 96 bits.

(Alldatasheet, 2023)

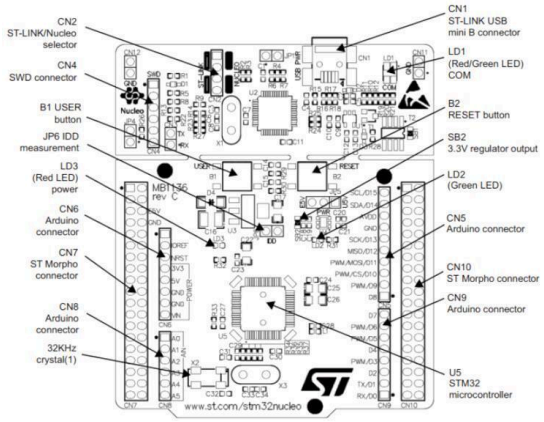


Figura 2. Conectores STM32F302R8
Fuente: extraído de https://docs.zephyrproject.org/latest/_images/nucleo_f302r8_connectors.jpg

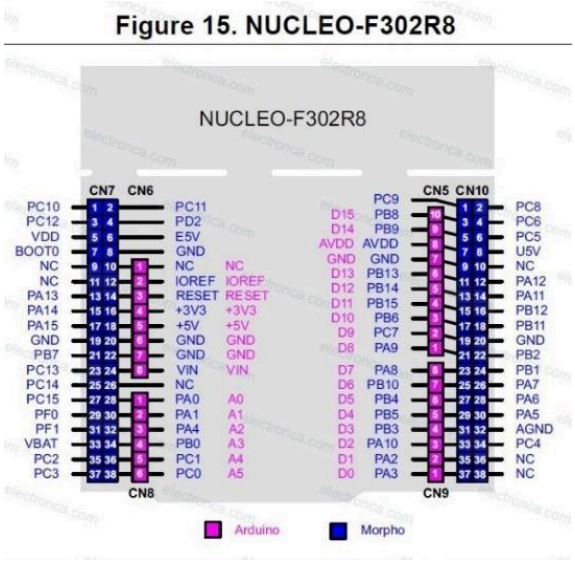


Figura 3. Núcleo F302R8
Fuente: extraído de https://electronca.com/wp-content/uploads/2023/02/Nucleo-f302r8-pinout-diagram.jpg

4.1.2 Escudo X-NUCLEO-IHM091:

X-NUCLEO-IHM09M1 es una placa de expansión de conector de control de motor para STM32 Nucleo. Ofrece una manera sencilla de evaluar soluciones de control de motor para motores trifásicos mediante la adaptación de la placa STM32 Nucleo con una placa de alimentación de control de motor ST externa, gracias a la tecnología ST Morpho y al conector de control de motor. El conector de control de motor de 34 pines es compatible con las principales placas de alimentación de control de motor ST, que requieren una sección digital (MCU) externa para controlar un motor trifásico. El conector DAC facilita el desarrollo y las pruebas de código de usuario con fácil acceso a los periféricos de la MCU.

Figura 4. Escudo X-NUCLEO-IHM09M1
Fuente: Imagen propia

4.1.2. Funciones

1. Conector de control de motor ST (34 pines) compatible con las principales placas de potencia de control de motor ST.
2. Compatibilidad con STM32 Nucleo, gracias a los conectores morfo ST.
3. Compatible con la biblioteca de firmware de control de motor de seis pasos y FOC de ST.
4. Conector de depuración para DAC, GPIO, etc.
5. Concepción de placa completamente poblada con puntos de prueba.

6. LED para condición de falla o indicación de estado.
7. Potenciómetro disponible (es decir, para referencia de velocidad).
8. Cumple con RoHS.

https://www.st.com/en/ecosystems/x-nucleo-ihm09m1.html

4.1.3 Modulo de potencia STEVAL-IHM028V1:

El objetivo de la placa de demostración STEVAL-IHM028V1 es presentar un diseño universal, completamente probado y completo, compuesto por un puente inversor trifásico basado en el módulo de potencia inteligente STGIPS20K60 de 600 V y 17 A. El propio IPM está compuesto por IGBT resistentes a cortocircuitos con coeficiente de temperatura negativo. También incluye una amplia gama de funciones auxiliares, como el bloqueo por subtensión y el apagado inteligente.

Gracias a estas características avanzadas, el sistema ha sido diseñado específicamente para lograr un acondicionamiento preciso y rápido de la realimentación de corriente, cumpliendo así los requisitos típicos de un control orientado a campo (FOC).

https://www.st.com/en/evaluation-tools/steval-ihm028v1.html#overview

Figura 5. STEVAL-IHM023V3
Fuente: Imagen propia

4.2 Motor trifásico:

El motor trifásico debe el termino o su nombre a que se alimenta de energía eléctrica trifásica. Las instalaciones monofásicas son más usadas en los hogares, que tensiones que van de 120 a 230 voltios y potencias que quedan por debajo de los 10 kW. El motor trifásico es muy usado en instalaciones industriales o comerciales. Esto se debe, por un lado, a que suelen ser más pequeños y manejables que motores monofásicos de la misma potencia. La potencia del motor trifásico varía dependiendo su uso y se fabrican en un rango muy grande de potencias, medidas de kilovatios o caballos de fuerza. Están destinados al accionamiento de máquinas como bombas, montacargas, ventiladores, grúas, elevadores y el uso académico en las universidades.

(Solerpalau, 2022)

Partes y componentes de un Motor trifásico: El motor trifásico se divide en tres partes importantes, estas son: el estator, el rotor y los escudos o carcaza.

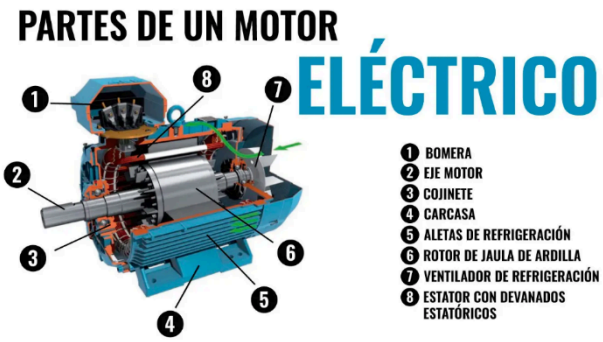


Figura 6. Motor Trifásico y sus partes
Fuente: Extraída de
https://motorba.com.ar/wp-content/uploads/cuales-son-las-partes-principales-de-un-motor-trifasico.webp

4.2.1 Estator:

El estator es la parte fija y opera como la base del motor esta parte es constituida por una carcasa en la que se fijan una corona de chas de hierro al silicio o acero al silicio, en las que están presentes unas ranuras. En estas ranuras es donde se presentan, al tratarse del motor trifásico, encontramos 3 bobinas y 3 circuitos diferentes. En cada circuito hay tantas bobinas como polos

tiene el motor.

4.2.2 Rotor:

El rotor es la parte móvil que se sitúa en el interior del estator. En el eje se inserta un núcleo magnético ranurado de acero al silicio en cuyas ranuras se colocan unas barras de cobre o aluminio (son lo que realizan la función de conductores) en una disposición que se conoce como ‘jaula de ardilla’ esto se debe a que las barras están unidas en cortocircuito por dos anillos, en la parte inferior y superior, confiriéndole una forma de jaula.

4.2.3 Escudos/carcasa:

Estos son los que constituyen la parte exterior del motor trifásico, generalmente producidos en aluminio o hierro colado. Están diseñados de tal forma que contienen unas cavidades para acoger los componentes esenciales en el interior. Sobre unos cojinetes descansa el eje del rotor, los escudos deben estar perfectamente ajustados para evitar distorsiones en el giro del rotor, estos como vibraciones y/o ruido.

Como funciona un motor trifásico:

Como hemos mencionado las partes del motor trifásico, el estator está compuesto por una estructura que conforma electroimanes y por eso esta parte también se denomina inductor. El bobinado en tres fases, al recibir una corriente eléctrica, genera un campo magnético que a su vez “induce” corriente en las barras del rotor. Su funcionamiento está basado en el principio de inducción mutua de Faraday En el campo magnético se genera precisamente por la aplicación de una corriente alterna de tres fases. La electricidad de corriente alterna cuenta con una onda que cambia de negativo a positivo muchas veces por segundo se trata de una onda llamada” onda sinusoidal” esa corriente alterna se compone de tres fases que están desfasadas 120° una respecto de la otra volviendo al motor trifásico, es la acción de tres ondas simultaneas la que genera un

flujo magnético que induce corriente en las barras del rotor creando un par motor que pone un movimiento al rotor, lo que es lo mismo, que hace que el rotor gire.

4.3 Inversores tradicionales:

Los convertidores de fase son dispositivos que permiten alimentar motores trifásicos en sistemas eléctricos monofásicos. Existen varios tipos de convertidores de fase y cada uno funciona de manera diferente para lograr este propósito.

(Machinetoolproducts, 2020)

8.3.1 Convertidor de fase estáticos:

Los convertidores de fase estáticos usan capacitores para generar una tercera fase. Durante el arranque, los capacitores generar un desfase momentáneo y una vez que el motor arranca, el sistema se estabiliza, manteniendo las tres fases necesarias.

El suministro monofásico está conectado a dos de los tres cables del motor. El tercer cable del motor está conectado a una de las salidas monofásicas en serie con los capacitores y los cables de salida del convertidor van conectados a los tres terminales del motor.

Este tipo de inversores es más adecuado para aplicaciones de baja potencia, ya que una vez que el motor arranca, la tercera fase generada puede no ser tan estables, el motor puede operar a menos del 100% de su capacidad nominal, por lo que es más eficiente en motores de uso ligero o intermitente.

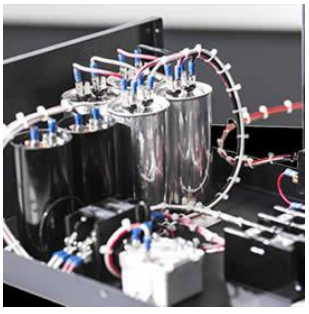


Figura 7. Convertidor de fase estático (Capacitores)

Fuente: Extraído de https://cdn.shopify.com/s/files/1/0680/2538/5243/files/Phase-Converter-7.jpg?v=1671354824

4.3.2 Convertidor de fase rotativa:

Se utiliza un motor generador adicional (generalmente un motor trifásico) conectado a una fuente monofásica. Este motor generador crea una salida trifásica real y balanceada, que se usa para alimentar el motor trifásico principal.

Los convertidores rotativos ofrecen una fase adicional más estable y equilibrada que los estáticos, permitiendo que los motores funcionen cerca de su potencia nominal. Son ideales para aplicaciones de mayor carga y uso continuo, aunque son más grandes y costosos.

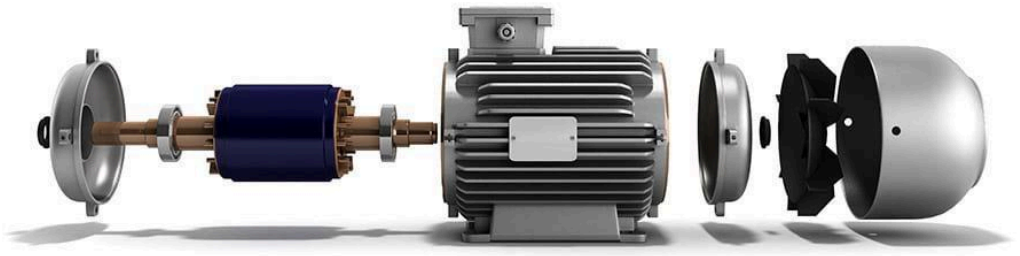


Figura 8. Motor generador
Fuente: Extraído de
https://www.phoenixphaseconverters.com/cdn/shop/files/Motorexplodedview.jpg?v=1718387230

El siguiente convertidor es el que vamos a utilizar para realizar este proyecto.

4.3.3 Convertidor de fase electrónico o variador de frecuencia:

Este tipo de convertidor rectifica la corriente alterna monofásica para convertirla en corriente continua y luego la convierte nuevamente en corriente alterna trifásica mediante un inversor. Además, permiten controlar la velocidad del motor al ajustar la frecuencia de salida. Son compactos, eficientes y ofrecen un control preciso de velocidad y torque. Son muy adecuados para motores trifásicos de cualquier tamaño y proporcionan una fase trifásica bien equilibrada.

El primer paso es convertir la corriente alterna monofásica en corriente continua mediante un rectificador. Esto se logra mediante el uso de diodos o tiristores, que permiten que la corriente fluya en una sola dirección generando una señal de corriente continua. Después de la rectificación se utiliza un filtro de capacitores para suavizar la señal CC y reducir las fluctuaciones de voltaje, produciendo una señal más estable. La corriente continua estabilizada pasa a través de un inversor, el cual utiliza transistores de potencia (IGBTs) que funcionan en alta frecuencia. Este inversor convierte la CC en una corriente alterna trifásica y ajusta la frecuencia y el voltaje de salida para controlar la velocidad del motor.

Controlando la frecuencia de señal de salida, el Variador de frecuencia puede ajustar la velocidad del motor. La frecuencia está directamente relacionada con la velocidad de rotación del motor a menor frecuencia, el motor gira más lento, a mayor frecuencia, gira más rápido.

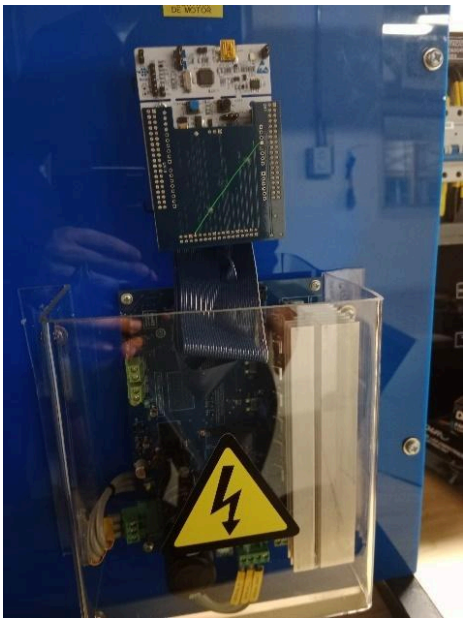


Figura 9. Convertidor de fase Electrónico
Fuente: La foto es propia

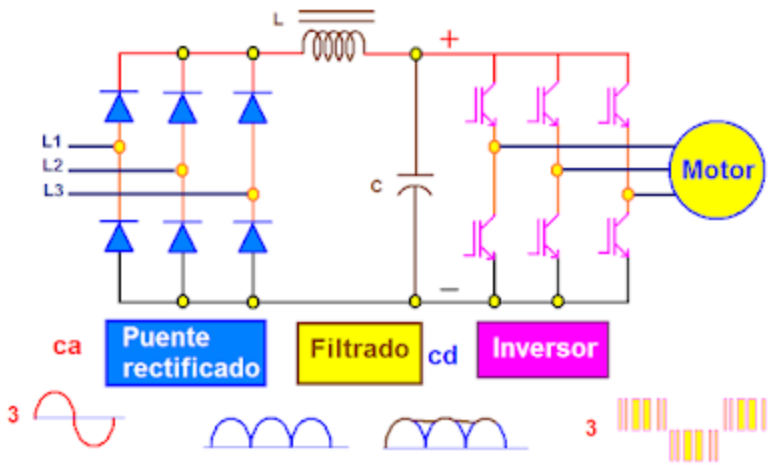


Figura 10. Convertidor de fase electrónico y digital

Fuente: Extraída de

https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhJxs6uHoik8fQqj8f82ZoZ9Qw0PrrpYnqQcTHqXeyd4cxMl6jxE2Trchnq0Ahj4J6aDH9KTQ1ajHeG2EYMa3lhWpIeHvY0LcyjJzr9ftcLp3gN3w7FX0V3tUgEr2eDWLJZHN_BInd9vVfK/s400/convertidor+de+frecuencia.PNG

5. Metodología

5.1 Tipo de proyecto

Este trabajo se considera de tipo investigativo, educativo y productivo, con respecto al tipo investigativo se determina descriptivo y explicativo porque se realiza una descripción de todos los elementos a utilizar en el proyecto, tanto eléctricos, electrónicos y mecánicos y explicativo porque a la medida de ejecución del proyecto se analizan las causas y los efectos con las distintas variables. Se realizará un inversor electrónico donde podamos cambiar la AC monofásica a una señal DC estable y esta convertirla en una señal AC trifásica con transistores TRIAC y IGBT para controlar la potencia y el funcionamiento del motor en cada etapa.

5.2 Método

Se realizarán tomas de datos cada vez que variamos la velocidad del motor y del cambio de giro y así poder saber cuánto es el consumo de energía, que cambios obtenemos cuando pasamos de una señal AC monofásica a una señal AC trifásica.

5.2 Instrumentos de recolección de información

5.2.1 Fuentes primarias.

Se recogió información de diferentes artículos que se encontró en internet como en Wikipedia, inteligencias artificiales, videos y proyectos parecidos.

6. Resultados

Con la ayuda de Arduino IDE se generó un código donde se puede evidenciar 6 fases con pwm con un desface de 120 grados, adaptando el STM32F302R8 logrando dichos objetivos.

Lo primero fue crear el código que seria este:

```
#define PinAh PA8
#define PinBh PA9
#define PinCh PA10
#define PinAl PA7
#define PinBl PB0
#define PinCl PB1
int count=0;

void setup() {

  pinMode(PA8, OUTPUT);
  pinMode(PA9, OUTPUT);
  pinMode(PA10, OUTPUT);
  pinMode(PA7, OUTPUT);
  pinMode(PB0, OUTPUT);
  pinMode(PB1, OUTPUT);
}

// the loop function runs over and over again forever
void loop(){
     digitalWrite(PinAh, LOW);
     digitalWrite(PinAl, LOW);
     digitalWrite(PinBh, LOW);
     digitalWrite(PinBl, LOW);
     digitalWrite(PinCh, LOW);
     digitalWrite(PinCl, LOW);
  delayMicroseconds(10);

  switch(count++){
    case 0:
     digitalWrite(PinAh, HIGH);
     digitalWrite(PinAl, LOW);
     digitalWrite(PinBh, LOW);
     digitalWrite(PinBl, HIGH);
     digitalWrite(PinCh, HIGH);
     digitalWrite(PinCl, LOW);
     break;
    case 1:
     digitalWrite(PinAh, HIGH);
```

```
      digitalWrite(PinAl, LOW);
      digitalWrite(PinBh, LOW);
      digitalWrite(PinBl, HIGH);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
    case 2:
      digitalWrite(PinAh, HIGH);
      digitalWrite(PinAl, LOW);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
    case 3:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
    case 4:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, HIGH);
      digitalWrite(PinCl, LOW);
      break;
    case 5:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, LOW);
      digitalWrite(PinBl, HIGH);
      digitalWrite(PinCh, HIGH);
      digitalWrite(PinCl, LOW);
      count = 0;
      break;
    default:
      break;
  }
  delayMicroseconds(2257);
}
```

En la primera parte es definir los puertos o periféricos del microcontrolador. En este caso utilizamos un #define que es una directiva del preprocesador que se utiliza para crear constantes. Esta directiva permite asignar un nombre a un valor constante antes de la compilación del código. Las constantes definidas con #define no ocupan espacio de memoria en el programa, ya que el compilador reemplaza el nombre de la constante con su valor durante la fase de preprocesamiento. PinAh(A-HIGH) se definio en el puerto PA8, PinBh(B-HIGH) se definio en el puerto PA9, En el PinCh(C-HIGH) PA10, en el PinAl(A-LOW) PA7, en el PinBl(B-LOW) PB0 y en el PinCl(C-LOW) PB1. También se definio la variable (count) con un entero (INT) es un tipo de variable que se utiliza para almacenar números enteros (sin decimales) igual a 0.

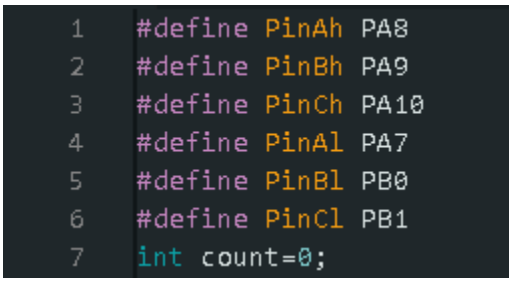


Figura 11. Definición de variables
Fuente: Imagen propia

En la parte de void setup es la función que se ejecuta una sola vez cuando se inicia el programa o se reinicia la placa. utilizamos PinMode que es una función que se utiliza para configurar el modo de trabajo de un pin digital. Puedes usarla para definir si un pin funcionará como entrada o como salida. Es este caso definimos todos lo pines para que sean salidas (OUTPUT).

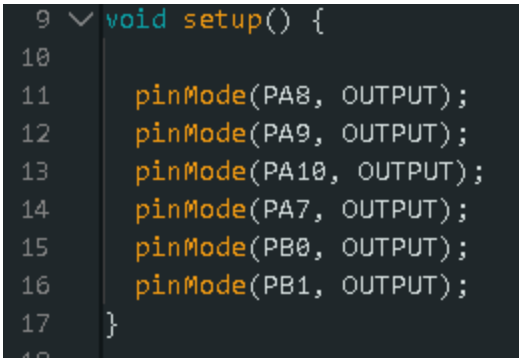


Figura 12. Void setup / pinMode
Fuente: Imagen propia

Pasamos al void loop que es una función que se ejecuta de manera cíclica y repetidamente mientras la placa esté encendida. En la primera parte utilizamos digitalWrite es una función de programación utilizada para enviar una señal digital (HIGH o LOW) a un pin digital. En este caso empezamos con todos los pines en bajo y utilizamos un delayMicroseconds para pausar la ejecución del programa durante un tiempo determinado en microsegundos.

En la segunda parte utilizamos un switch que se refiere a la instrucción switch-case. Es una estructura de control que permite ejecutar diferentes bloques de código dependiendo del valor de una variable o expresión, y case se utiliza dentro de la instrucción switch para especificar los diferentes valores de la variable a evaluar. Al final de cada case ponemos un break que fuerza la interrupción del flujo de ejecución actual y pasa a la siguiente instrucción que se encuentra después del bucle o estructura que se estaba ejecutando. El default se refiere a una instrucción que se ejecuta cuando ninguna de las opciones case coincide con el valor que se está evaluando. Sirve como una caja de seguridad para manejar casos no previstos y terminamos con un delayMicroseconds de 2257. En las siguientes imágenes se muestra el proceso explicado.

```
void loop(){
    digitalWrite(PinAh, LOW);
    digitalWrite(PinAl, LOW);
    digitalWrite(PinBh, LOW);
    digitalWrite(PinBl, LOW);
    digitalWrite(PinCh, LOW);
    digitalWrite(PinCl, LOW);
  delayMicroseconds(10);

  switch(count++){
    case 0:
      digitalWrite(PinAh, HIGH);
      digitalWrite(PinAl, LOW);
      digitalWrite(PinBh, LOW);
      digitalWrite(PinBl, HIGH);
      digitalWrite(PinCh, HIGH);
      digitalWrite(PinCl, LOW);
      break;
    case 1:
      digitalWrite(PinAh, HIGH);
      digitalWrite(PinAl, LOW);
      digitalWrite(PinBh, LOW);
      digitalWrite(PinBl, HIGH);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
    case 2:
      digitalWrite(PinAh, HIGH);
      digitalWrite(PinAl, LOW);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
    case 3:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, LOW);
      digitalWrite(PinCl, HIGH);
      break;
```

Figura 13. Void loop / digitalWrite / swicht / case 0 – 3
Fuente: Imagen propia

```
    case 4:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, HIGH);
      digitalWrite(PinBl, LOW);
      digitalWrite(PinCh, HIGH);
      digitalWrite(PinCl, LOW);
      break;
    case 5:
      digitalWrite(PinAh, LOW);
      digitalWrite(PinAl, HIGH);
      digitalWrite(PinBh, LOW);
      digitalWrite(PinBl, HIGH);
      digitalWrite(PinCh, HIGH);
      digitalWrite(PinCl, LOW);
      count = 0;
      break;
    default:
      break;
  }
  delayMicroseconds(2257);
}
```

Figura 14. Void loop / digitalWrite / swicht / case 4 – 5
Fuente: Imagen propia

Después de realizar el respectivo código, cargamos los datos al STM32F02R8 y con la ayuda un oscilospio y verificaremos los pwm. Las 2 primeras imágenes se muestran el desface de Ah-Bh y Bh-Ch.

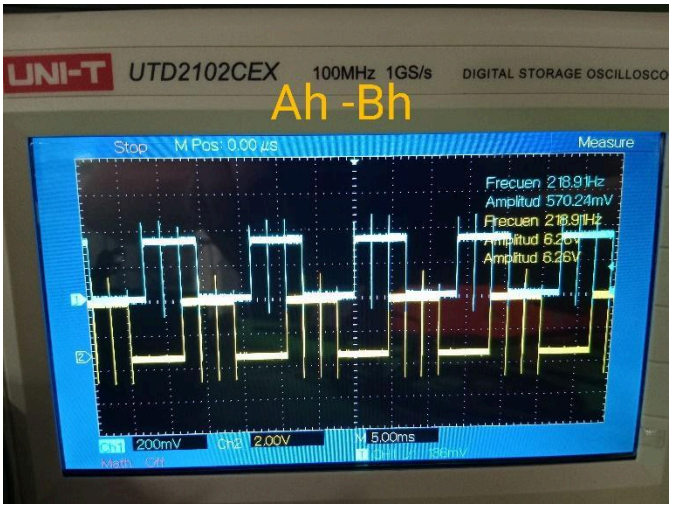


Figura 15. Muestra osciloscopio Ah-Bh
Fuente: Imagen propia

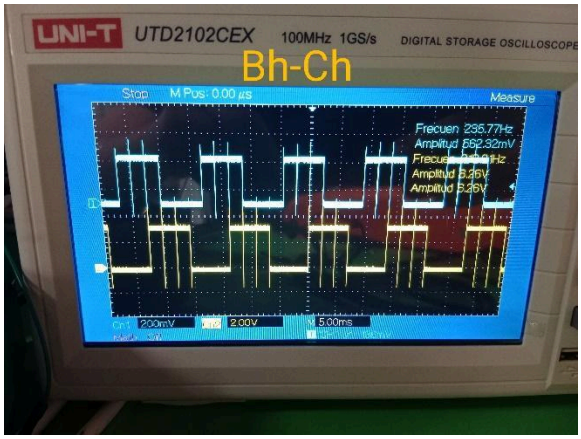


Figura 16. Muestra osciloscopio Bh-Ch
Fuente: Imagen propia

Las siguientes imágenes muestran el PWM con el espejo de su otra señal en este case si es Ah-Al , Bh-Bl, Ch-Cl. Se muestra cuando la señal azul esta en alto, la otra señal está en bajo y así sucesivamente. En la mitad de cada cambio de señal aparece una línea en la mitad eso significa los tiempos muertos para que un transistor se apague antes de activar el siguiente.

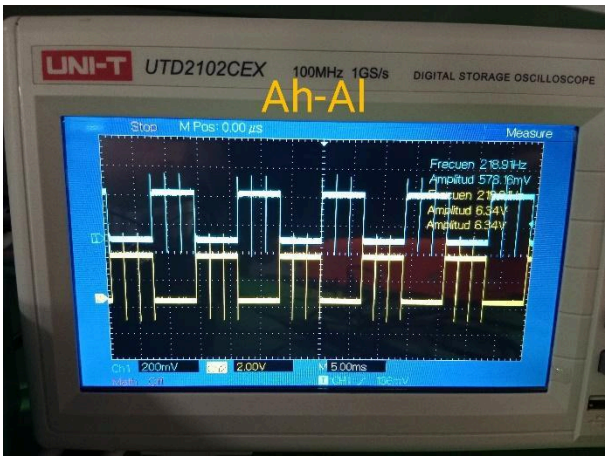


Figura 17. Muestra osciloscopio Ah-Al
Fuente: Imagen propia

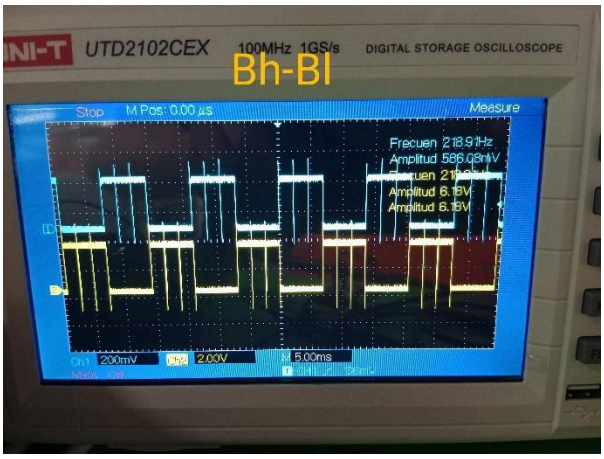


Figura 18. Muestra osciloscopio Bh-Bl
Fuente: Imagen propia

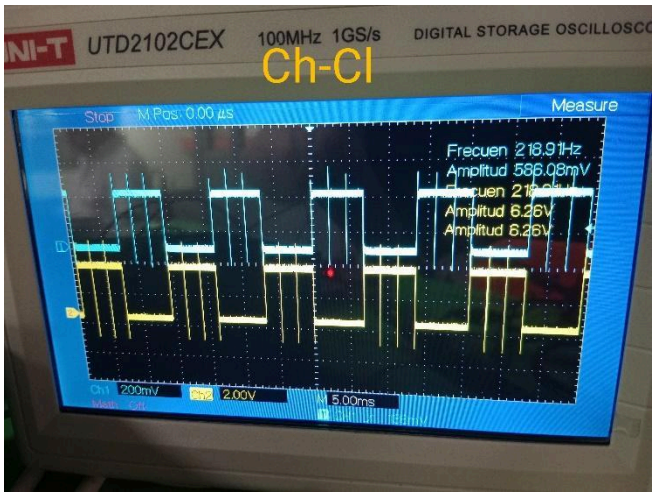


Figura 19. Muestra osciloscopio Ch-Cl
Fuente: Imagen propia

El siguiente paso fue realizar el montaje, con la parte de potencia que sería el STEVAL-IHM023V3. Conectamos los periféricos del microcontrolador STM32F302R8 al puerto J5 del modulo de potencia. Se alimenta el módulo con un variac y se conecta a una señal a 110 y se variar el voltaje de salida, esto permite el arranque del motor.

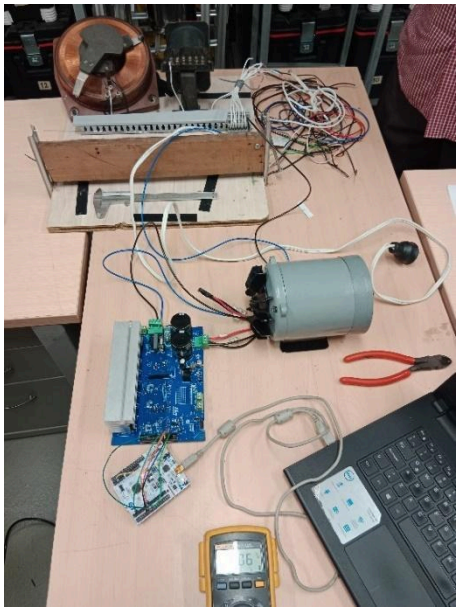


Figura 20. Montaje final
Fuente: Propia

7. Conclusiones

Para activar los IGBT es necesario conectar el pin 1 del STEVAL-IHM023V3 enviando una señal en alto del microcontrolador.

Para realizar la activación del relevo del modulo de potencia STEVAL-IHM023V3, es necesario adminístrale el voltaje suficiente para el correcto funcionamiento del motor.

Para generar una señal de 60Hz es necesario modular un PWM a una frecuencia mínima es de 1200Hz.

8. Recomendaciones

Las recomendaciones que surgieron en el desarrollo del proyecto son, conocer sobre varios tipos de microcontroladores que se adapten al proyecto. investigar que microcontroladores tengan características parecidas ya que hay una amplia gama de estos, ya sea de STM32 o de Arduino.
Conocer y aprender de los 2 tipos de software que en este caso es STM32cubeIDE Y Arduino IDE, buscar en varias fuentes de información para ampliar el conocimiento del manejo de estos programas ya que abarcan un gran desarrollo de código donde en este caso podemos controlar un motor o para otros tipos de proyecto como el análisis de sensores.

En este caso que era el control de un motor trifásico, se recomienda primero con la ayuda de un osciloscopio identificar las 3 faces antes de conectarlas con el circuito de potencia, ya que los materiales son costosos y no se puede dar el lujo de dañarlos y reponerlos.

También se recomienda buscar ayuda a profesores o otras personas que tengan conocimiento de estos temas, de tanto el código como la instalación y conexión de estas herramientas. Realizar pruebas en el laboratorio con un motor pequeño, que tenga un voltaje y corriente baja para que se prevengan accidentes.

Muy importante tener conocimientos básicos tanto de electrónica como de mecatrónica y de programación, para que al mismo tiempo que estén investigando puedan asimilar mejor la información y no tengan percances con el tiempo y tener buena comunicación con los compañeros de investigación y profesores.

Los trabajos a futuro se pueden realizar de una mejor manera, investigando gracias a las

inteligencias artificiales, ya que si uno le sabe preguntar e interpretar bien la respuesta se puede hacer un desarrollo de proyecto de investigación coherente y con una amplia información exacta de lo que se está buscando. dejar a un lado los tabúes sobre estas herramientas que están dando un giro importante a la educación y a la búsqueda de información.

9. Referencias bibliográficas

Alldatasheet. (14 de Enero de 2023). *Alldatasheet*. Recuperado el 11 de Noviembre de 2024, de https://www.alldatasheet.com/datasheet-pdf/pdf/513396/STMICROELECTRONICS/STM32F302.html

Machinetoolproducts. (12 de Agosto de 2020). *Machinetoolproducts*. Recuperado el 11 de Noviembre de 2024, de https://www.machinetoolproducts.com/phase-converters/static-vs-rotary-phase-converters

Solerpalau. (20 de Julio de 2022). *Solerpalau*. Recuperado el 23 de Noviembre de 2024, de https://www.solerpalau.com/es-es/blog/motor-trifasico/

Chat gpt. (2 de Noviembre de 2024). *Chat gpt*. Recuperado el |0 de Noviembre de 2024, de https://chatgpt.com/c/673240c9-82b4-800a-87fc-a50823d5da55

Documento de datasheet del F302R8

file:///C:/Users/Marco/Downloads/stm32f302r8.pdf

Arduino IDE

https://www.arduino.cc/en/software/

X-NUCLEO-IHM09M1

https://www.st.com/en/ecosystems/x-nucleo-ihm09m1.html

STEVAL-IHM023V3

https://www.st.com/en/evaluation-tools/steval-ihm023v3.html

VARIAC 110V

https://variac.com/staco/PDFCutSheets/VT%20designengine.pdf

10. Bibliografía

Documentación de las funciones de HAL
https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf

Módulo de potencia steval-ihm028v1
https://www.st.com/en/evaluation-tools/steval-ihm028v1.html#overview

Foros STM32
https://community.st.com/t5/forums/searchpage/tab/message?q=control%20motor%20trifasico&noSynonym=false&collapse_discussion=true

STMcubeMX
https://www.st.com/en/development-tools/stm32cubemx.html

Proyectos similares ATlab
https://www.youtube.com/watch?v=pji4pLFAV3o&ab_channel=ATLab

sistema RCC
https://www.st.com/resource/en/product_training/STM32F7_System_RCC.pdf

STM32cubeIDE
https://www.st.com/en/microcontrollers-microprocessors/stm32f302r8.html

11. Anexos

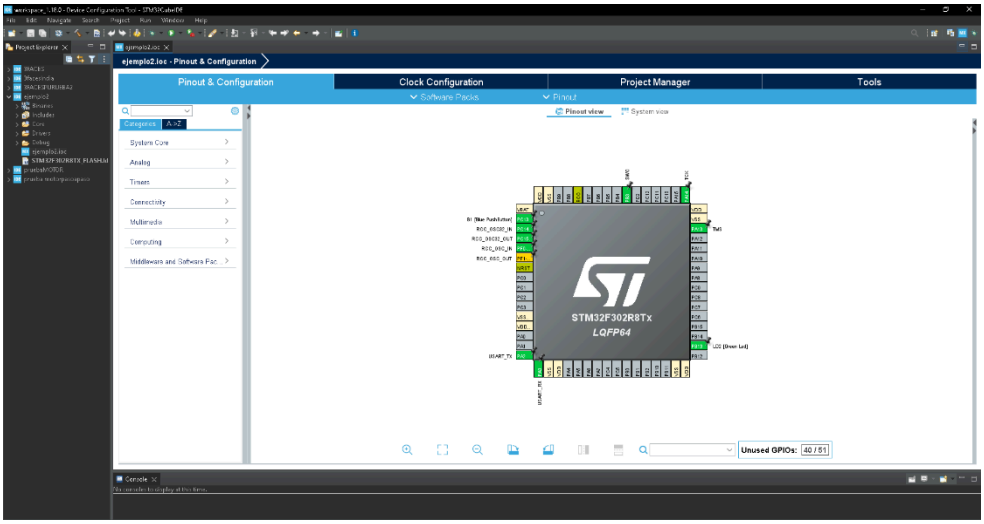
Anexo A. Primer contacto con el software STM32cubeIDE

Cuando se empezó con la investigación de como utilizar un microcontrolador STM32 para el control de un motor, tuvimos varias preguntas de por donde empezar a investigar. Se le pregunto a varios profesores y lo que nos dijeron fue primero identificar con que software se utilizara para el desarrollo del código. Unos nos recomendaron trabar en ArduinoIDE porque es un programa común que se utiliza en casi todas las materias para hacer tareas, trabajos o investigaciones. otros profesores nos recomendaron trabajar con el software oficial que es STMcubeIDE, porque este software es muy completo para lo que nosotros necesitábamos y podía cumplir los objetivo. Pero el problema es que no teníamos el suficiente conocimiento para manejar los 2 programas y generar un código por nuestra propia cuenta. Entonces esto nos llevó a investigar y buscar información en internet.

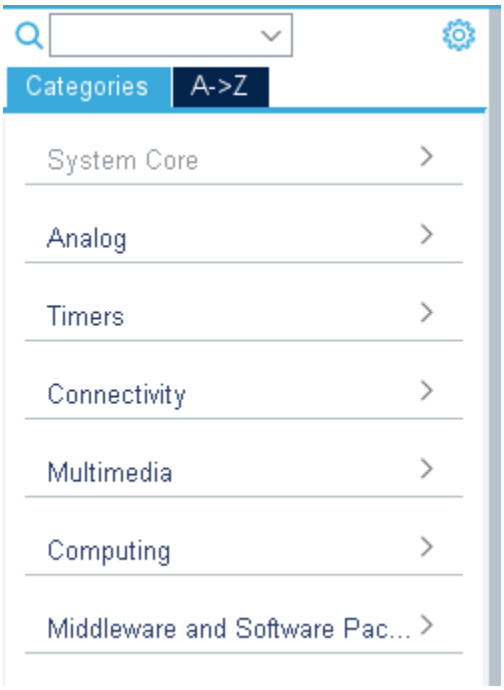
Lo primero era descargar los 2 software e instalarlos.

Lo segundo era buscar como hacer un código simple para prender y apagar un led, en este caso el led que esta incluido en el microcontrolador STM32F302R8. para así verificar que estuviera en buen estado el microcontrolador y que las instalaciones de los softwares estuvieran correctas.

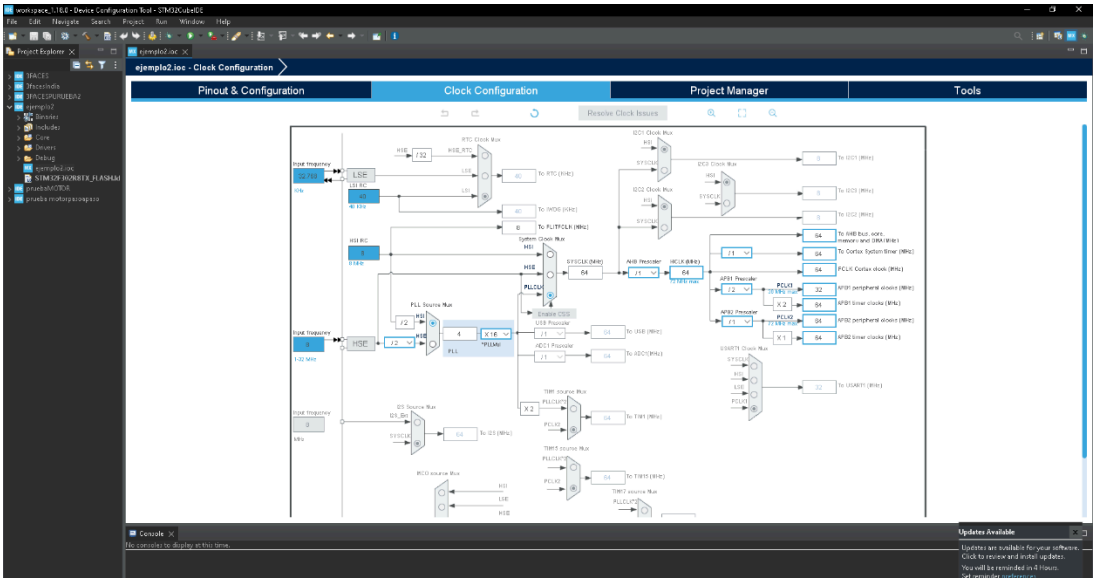
En este caso se utilizo el STM32cubeIDE donde conocimos la interfaz y sus diferentes herramientas.



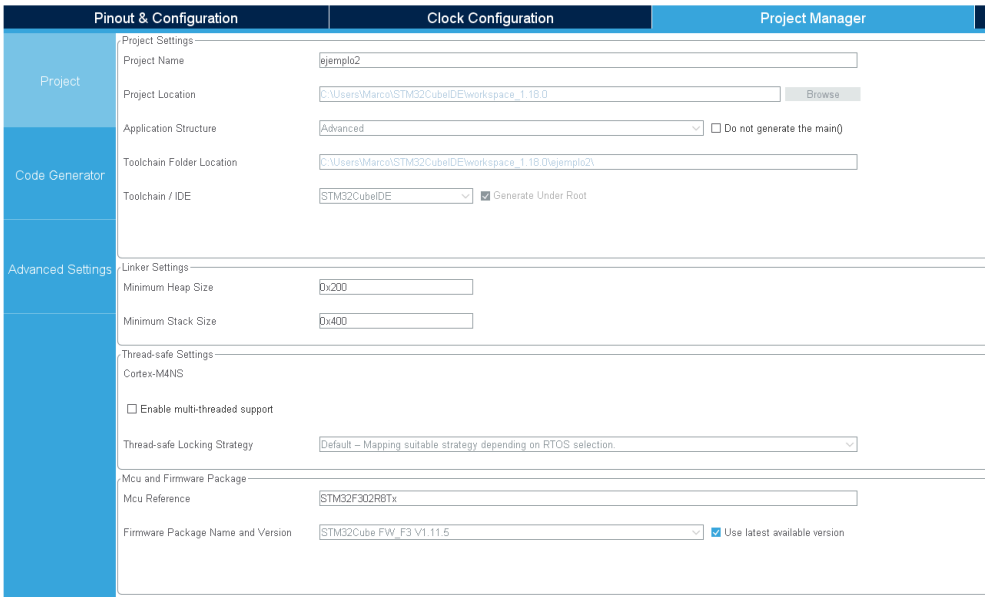
Esta seria la interfaz del software STM32cubeIDE y en la imagen se aprecia el procesador de microcontrolador STM32F302R8 con sus diferentes pines.



En la parte izquierda de la interfaz podemos encontrar los periféricos disponibles de la tarjeta, system core, analog, timers, connectivity, multimedia, computing y middleware and software.



En esta sección encontraremos los multiplicadores y divisores de los relojes de la tarjeta.



En la otra pestaña Project manajer, es una sección propia de lo que es la configuración del proyecto.

Se deja todo por defecto y presionamos este símbolo que es para generar el código. Lo que hace la plataforma es instanciar y realizar todas las funciones de código que necesita el microcontrolador para funcionar.

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
```

```
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
int State = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
```

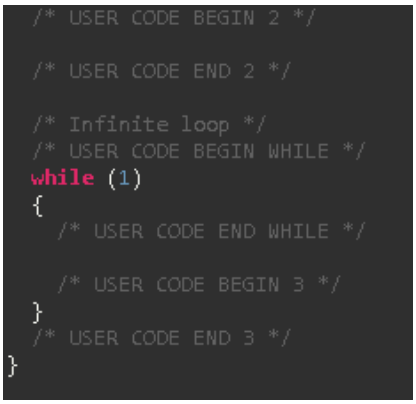
```
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
          HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
          HAL_Delay(500);
          State = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  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 USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 38400;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @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_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @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 */
```

En el archivo main.c es donde vamos a escribir toda nuestra lógica. Por defecto se nos genera algunas funciones y se instancias algunas librerías, están son propias de la configuración de la tarjeta y es muy importante no tocarlas.



```
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Vamos a trabajar principalmente en el ciclo infinito, para poner dentro del ciclo infinito toda nuestra lógica que queremos que se repita siempre. Antes del ciclo vamos a poner lo que queremos que se ejecute solo una vez, como por ejemplo seria la configuración de los periféricos.

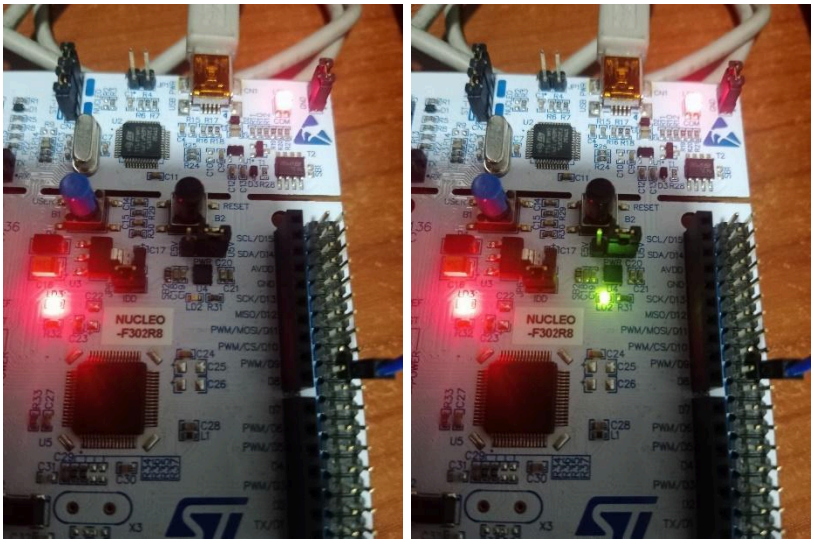
Una ventaja que nos da STMcubeIDE es que nos instala por defecto lo que son los drivers de CMSIS y HAL. Esto son una serie de funciones y librerías para el control de registros y periféricos, que nos va a simplificar todo el proceso de codificación y programación del microcontrolador.

```
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
      HAL_Delay(500);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Dentro del WHILE, en la sección destinada para la codificación del usuario vamos a escribir HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13); la tarjeta por defecto ya tiene unos pines, vamos a trabajar con el pin PB13 que es el primer led de la tarjeta que ya esta configurado en la opción GPIO_OUTPUT . para ver el parpadeo de led utilizamos otra función HAL_Delay(500); le damos un delay de medio segundo, que corresponde en 500 ciclos de máquina.

Le damos en RUN, que hace correr el código, el automáticamente copila, busca errores y si todo

es correcto descarga el código en la tarjeta. Se confirma que todo está bien.

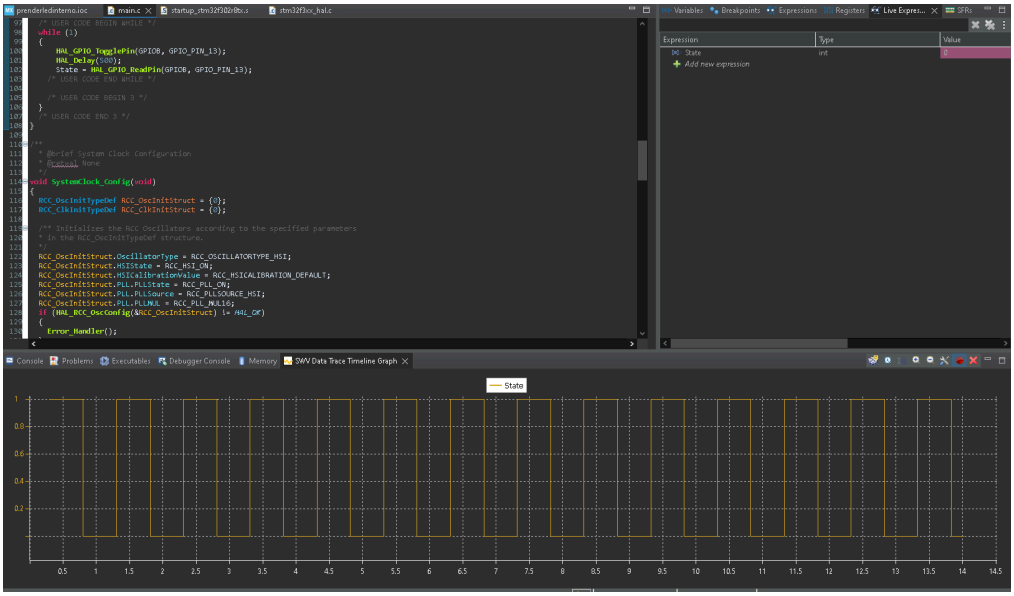


Podemos mirar que el LD2 parpadea con un delay de medio segundo que sería 500ms.

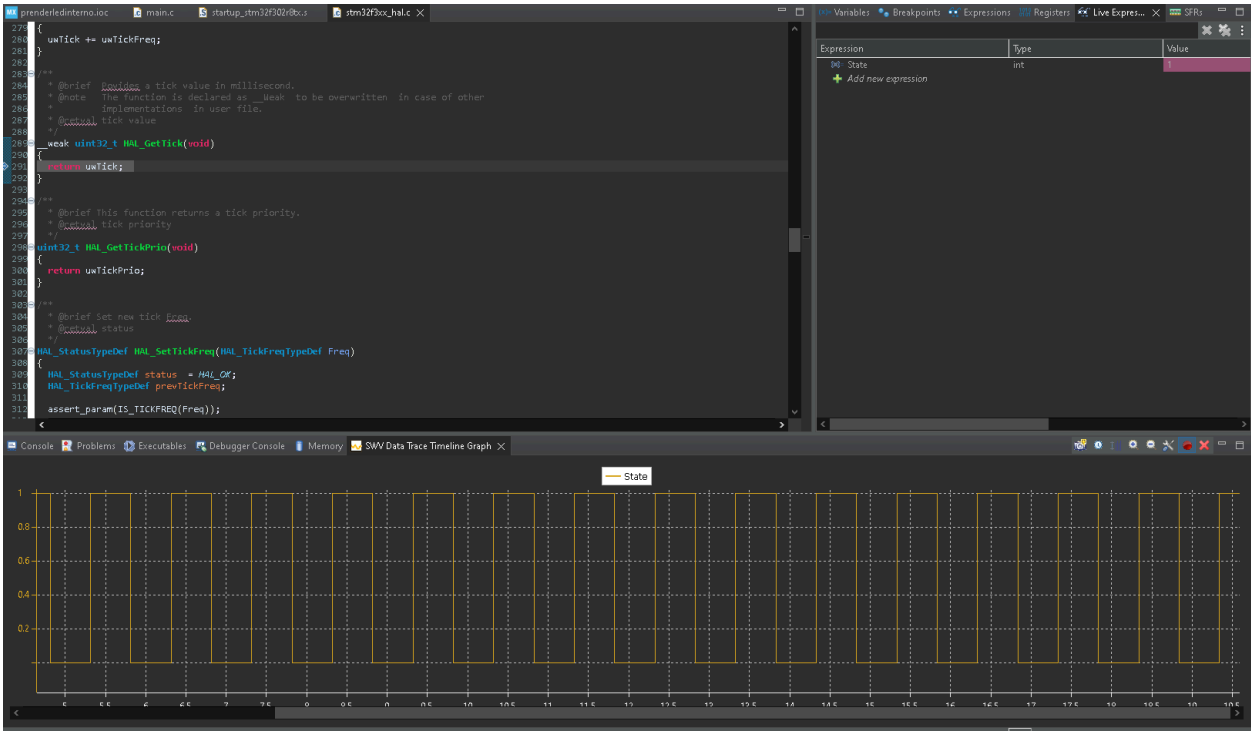
```
/* USER CODE BEGIN PV */

int State = 0;

/* USER CODE END PV */
```

```
  while (1)
  {
      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
      HAL_Delay(500);
      State = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Ahora lo que vamos a hacer es crear una variable para leer el estado del led que sería State = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13); pero antes debemos inicializar esta variable el la parte destinada del usuario. La ponemos como una variable entera INT y la iniciamos en 0.



Ahora nos vamos para la herramienta debug para revisar el estado de la variable. Podemos mirar el cambio de la variable que va de 1 a 0 y en la grafica confirma este cambio de estado.



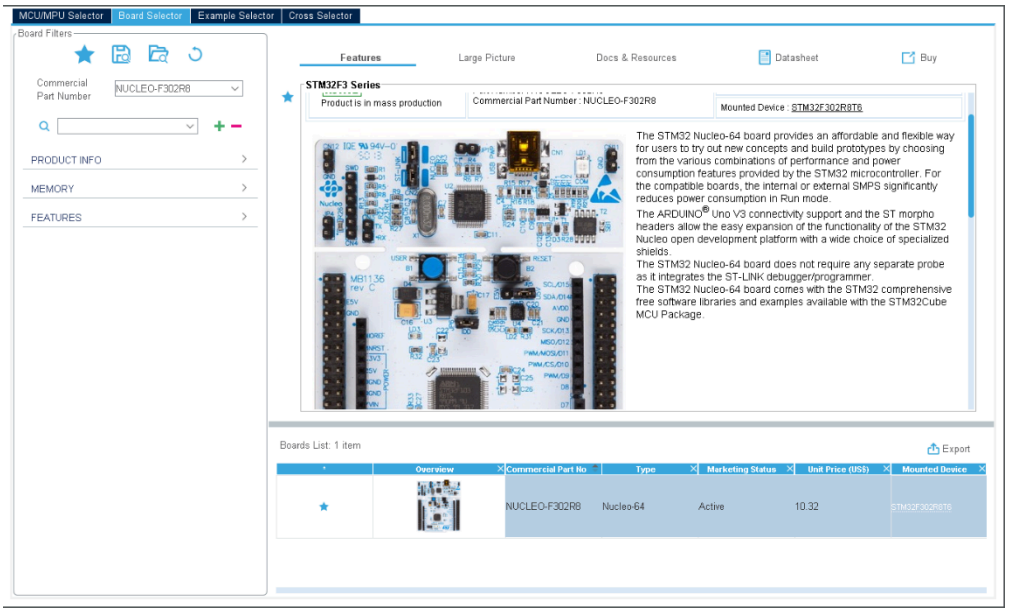
Esto seria un paso importante para conocer como función el software STM32cubeIDE, en un simple código que es encender y apagar el led interno del microcontrolador STM32302R8. miramos varias herramientas como el debugger que nos permite mirar los estados y la gráfica.

Anexo B. Generar 2 PWM con stm32

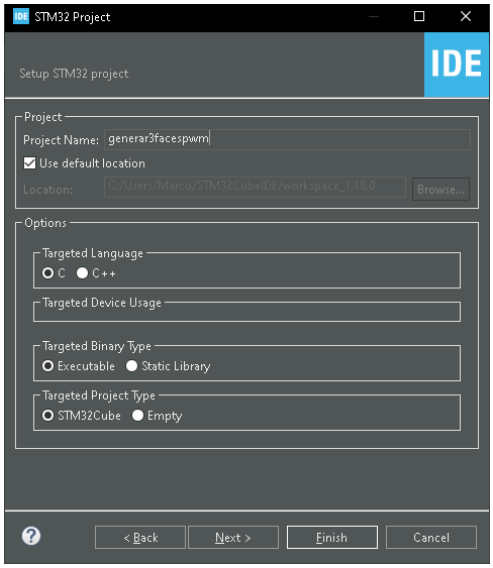
Después de una ardua investigación de como generar un código embebido para el control de un motor, se encontró y se realizo un código que pudiera resolver los objetivos planteados que era generar 2 PWM con un desface de 120 grados para poder realizar el control.

En este caso se utilizó el software STM32cudeIDE y un microcontrolador STM32F302R8.

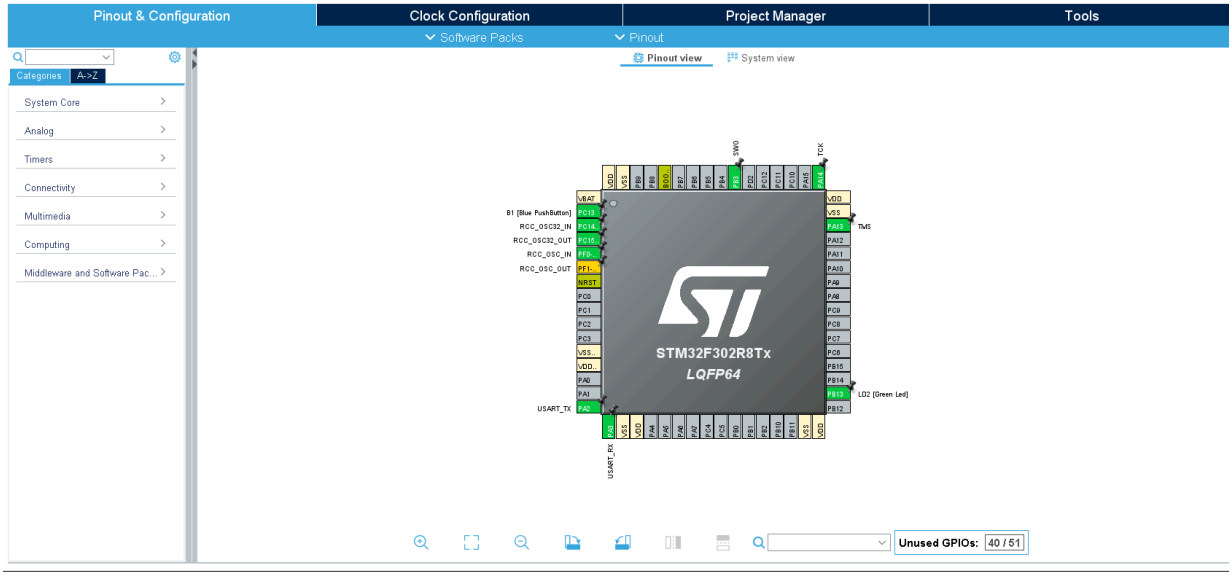
Lo primero es ir al software y crear un nuevo proyecto, en la parte de BOARD SELECTOR seleccionamos el microcontrolador que vamos a utilizar, en este caso es el NUCLEO-F302R8.



Le danos NEXT, después le ponemos un nombre al código que deseamos, en este caso es GENERAR 3 FACES PWM y le damos FINISH.

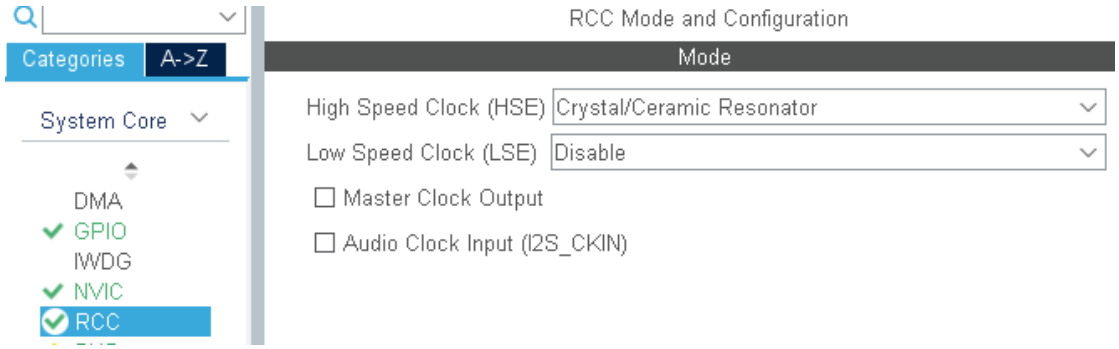


Después de darle FINISH el software empieza a generar un menú donde podemos visualizar el núcleo del microcontrolador con sus periféricos y varias pestañas. Pero en este caso solo nos vamos a concentrar en lo que es PIN OUT Y CONFIGURATIONS y en CLOCK CONFIGURATIONS.

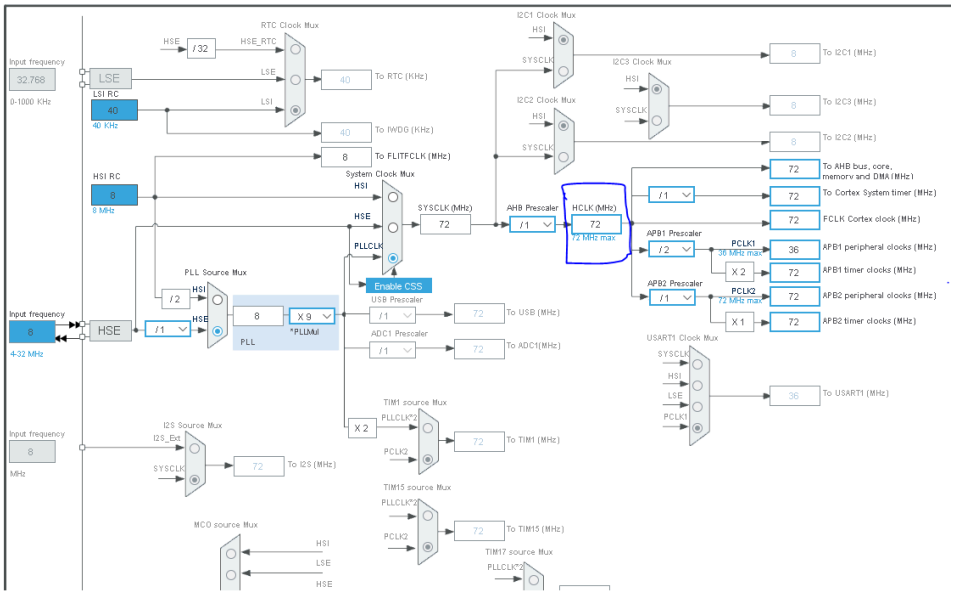


El prime paso es ir a SYSTEM CORE, después nos vamos a RCC. ¿qué significa?

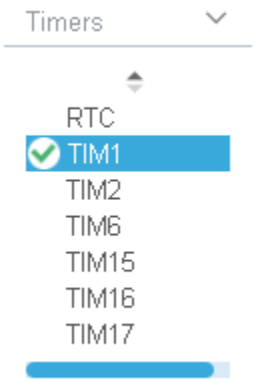
(RCC significa "Reset and Clock Control" Control de Reinicio y Reloj). Después vamos a donde dice HIGH SPEED CLOCK (HSE) ¿qué es? (es una fuente de reloj externa de alta velocidad que se utiliza para generar el reloj principal del sistema. Generalmente, se utiliza un oscilador de cristal o un reloj externo para proporcionar esta señal de alta frecuencia) Y en esa pestaña seleccionamos CRYSTAL/CERAMIC RESONATOR (son componentes que generan la señal de reloj principal, que controla la velocidad de operación del microcontrolador).



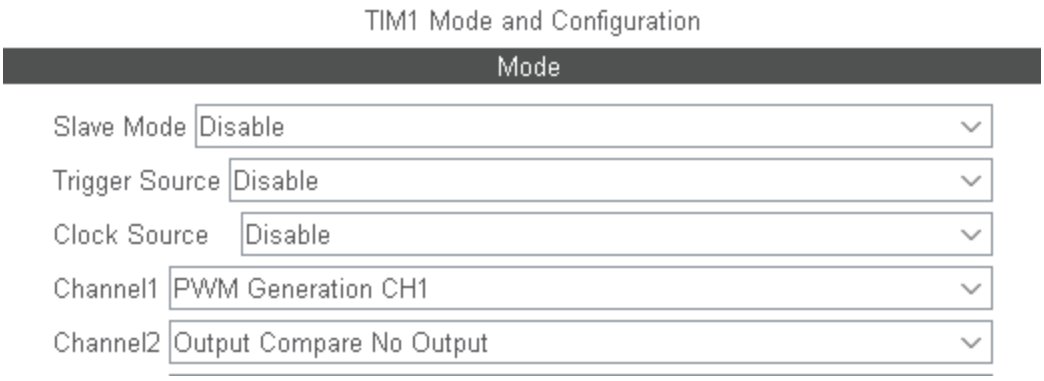
El siguiente paso es ir a la parte de CLOCK CONFIGURATION, vamos a donde dice HCLK(MHz), los ponemos a la máxima frecuencia en la que trabaja el reloj principal, en este caso es 72MHz. Al cambiar esta configuración automáticamente se actualiza el APB y así poder que los ambos buces APB trabajen a la misma frecuencia con esto podemos utilizar la misma configuración para los 3 temporizadores. (APB significa Advanced Peripheral Bus (Bus Periférico Avanzado). Es un bus de comunicación utilizado para conectar periféricos a la CPU y otros módulos del microcontrolador. En esencia, el APB facilita el intercambio de datos entre la CPU y componentes como temporizadores, UART, GPIO, etc.)



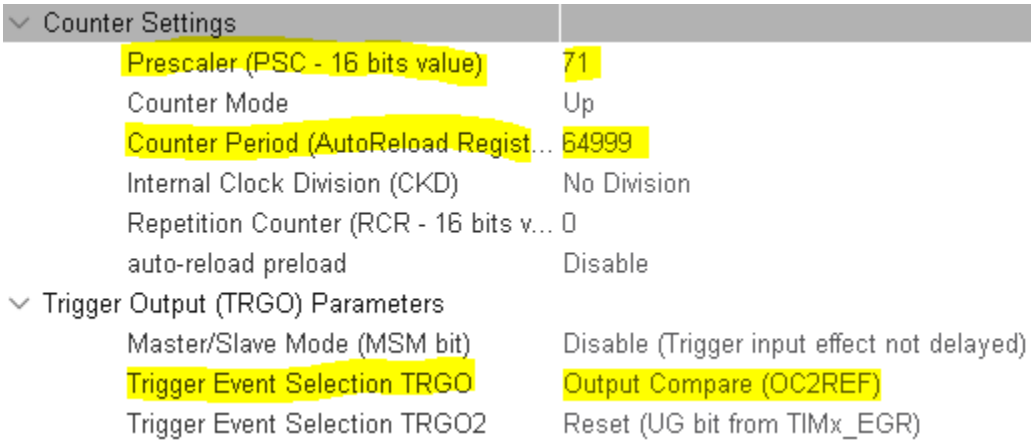
El siguiente paso, nos devolvemos a la pestaña PINOUT Y CONFIGURATION, después a la pestaña de los TIMERS. Configuramos el primer TIMER1, que será el maestro del TIMER2.



Aquí generaremos la señal PWM en el canal 1 (PWM generationCH1) y también compararemos LA SALIDA SIN SALIDA en el canal 2 (Output Compare No Output), ya que esta se usará para el disparo interno.

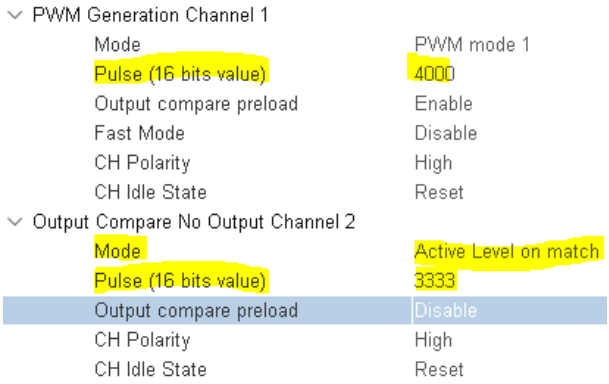


Después vamos a la parte en donde dice PARAMETER SETTINGS modificamos la frecuencia del PWM y el ciclo de trabajo. El temporizador esta trabajando a 72MHz, por lo que un prescaler de 71 (prescaler PSC - 16 BIT VALUE) reducirá la frecuencia a 1MHz. Luego la recarga automática (COUNTER PERIOD) de 64999 reducirá la frecuencia a 100 Hz. Ahora para la selección del evento de disparo (Trigger event selectionTRGO) iremos con la comparación de salida para el canal 2. (Oc2ref) se usa como señal de disparo y también dispara el temporizador.



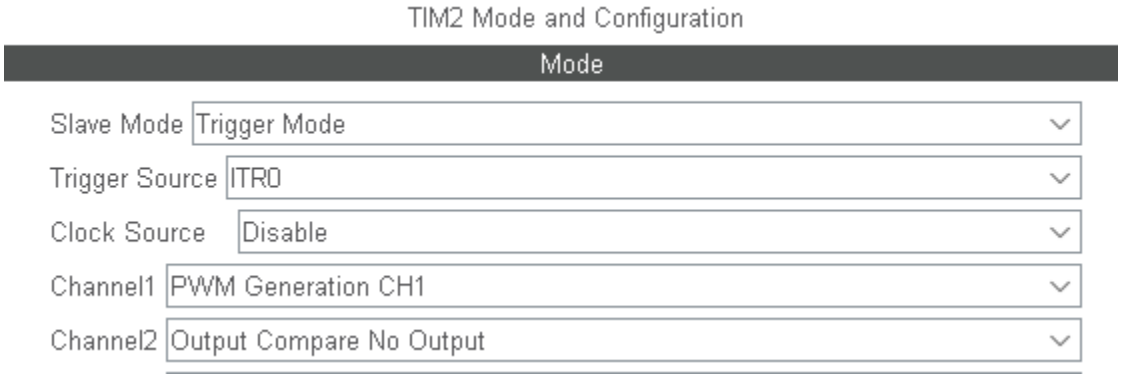
En PWM generation channel 1 en PULSE se configura en ciclo del trabajo de la señal pwm en 40 por ciento que seria 4000. Para la comparación de salida en OUTPUT COMPARE NO OUTPUT CHANNEL, en MODE cambie el modo a nivel activo en Match. Esto básicamente activara la señal una vez que el valor del contador coincida con el valor de comparación y este valor de comparación lo es 33 por ciento que seria 3333 del

valor de recarga automática.



Esta seria la parte del temporizador uno.

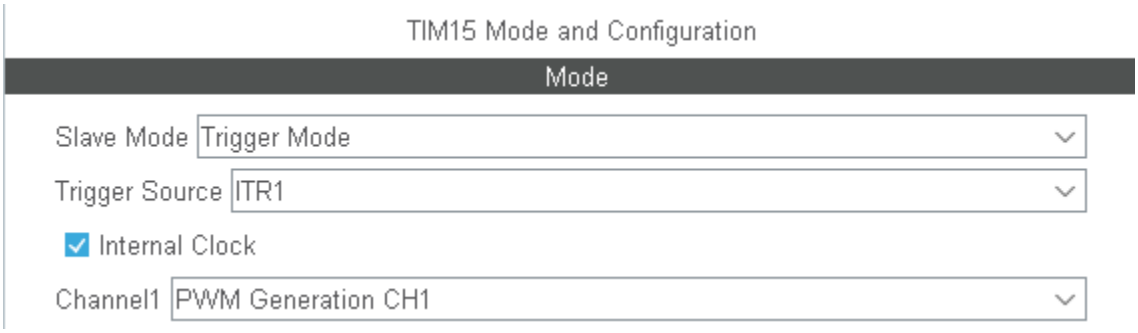
Ahora modificaremos el temporizador 2 que actuara como esclavo para el temporizador 1 y maestro para el temporizador 3. Se usa el modo de disparo TRIGGER MODE (modo de disparo) permite que un temporizador MAESTRO active el inicio de otro temporizador ESCLAVO o de otros periféricos como el ADC. para que el contador comience a contar una vez que el temporizador reciba la señal ITR0.



El resto de configuración es exactamente la misma que la del temporizador 1. Así que básicamente, el temporizador 2 comenzara a contar al recibir la señal del temporizador 1. Una vez que el contador alcance el 33 por ciento del valor de recarga automática, la salida de comparación de volverá alta, lo que generara el evento ITR1.

Counter Settings	
Prescaler (PSC - 16 bits value)	71
Counter Mode	Up
Counter Period (AutoReload Regist...	64999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Slave Mode Controller	Trigger Mode
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Output Compare (OC2REF)
Clear Input	
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (32 bits value)	4000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
Output Compare No Output Channel 2	
Mode	Active Level on match
Pulse (32 bits value)	3333
Output compare preload	Disable
CH Polarity	High

Configuramos el temporizador 3 que en este caso utilizaremos el temporizador 15, que es esclavo del temporizador 2 y se puede controlar usando el disparador ITR1.



No necesitamos generar la comparación de salida para el temporizador 3 y el resto de configuración es la misma de los otros temporizadores.

Counter Settings

Prescaler (PSC - 16 bits value)	71
Counter Mode	Up
Counter Period (AutoReload Regist...	64555
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits va...	0
auto-reload preload	Disable
Slave Mode Controller	Trigger Mode

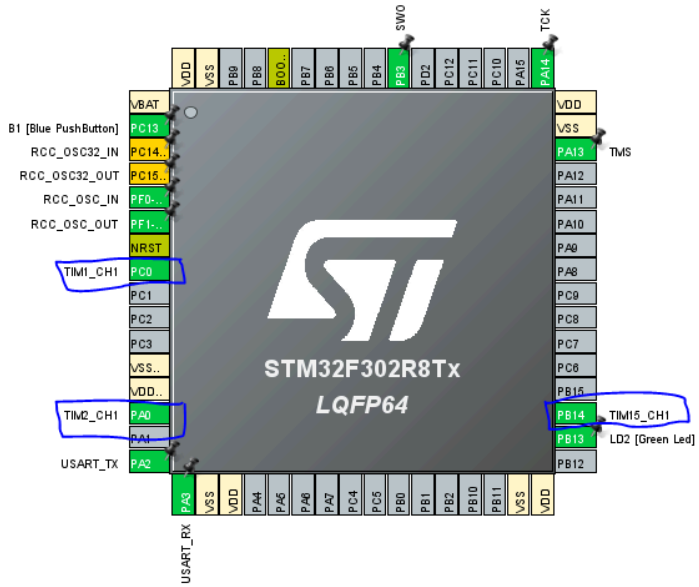
Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

PWM Generation Channel 1

Mode	PWM mode 1
Pulse (16 bits value)	4000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
CH Idle State	Reset

El firmware genera tres pines de salida pwm que están conectados al analizador lógico. El TIM1_CH1 queda en el pin PC0, el TIM2_CH1 queda en el pin PA0 y el TIM15_CH1 queda en el pin PB14.



Guardamos el proyecto para generar el código. En la parte de codificación, tenemos que

iniciar el PWM para el canal 1y la comparación de la salida para el canal 2. Solo habilitaremos el PWM para el temporizador 15 ya que no hemos configurado la comparación de salida.

```
/* USER CODE BEGIN 2 */

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_OC_Start(&htim1, TIM_CHANNEL_2);

HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_2);

HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1);

/* USER CODE END 2 */
```

se guarda, se construye (BUILD DEBUG) y se carga (RUN) en la placa.

Este sería el código:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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"
```

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim15;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM15_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
```

```
 */
int main(void)
{

 /* USER CODE BEGIN 1 */

 /* USER CODE END 1 */

 /* MCU Configuration--------------------------------------------------------*/

 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 MX_TIM1_Init();
 MX_TIM2_Init();
 MX_TIM15_Init();
 /* USER CODE BEGIN 2 */

 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
 HAL_TIM_OC_Start(&htim1, TIM_CHANNEL_2);

 HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
 HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_2);

 HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1);

 /* USER CODE END 2 */

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

   /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
```

```
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** 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.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_TIM1|RCC_PERIPHCLK_TIM15;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  PeriphClkInit.Tim15ClockSelection = RCC_TIM15CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @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_MasterConfigTypeDef sMasterConfig = {0};
 TIM_OC_InitTypeDef sConfigOC = {0};
 TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

 /* USER CODE BEGIN TIM1_Init 1 */

 /* USER CODE END TIM1_Init 1 */
 htim1.Instance = TIM1;
 htim1.Init.Prescaler = 71;
 htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim1.Init.Period = 64999;
 htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim1.Init.RepetitionCounter = 0;
 htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
 {
  Error_Handler();
 }
 if (HAL_TIM_OC_Init(&htim1) != HAL_OK)
 {
  Error_Handler();
 }
 sMasterConfig.MasterOutputTrigger = TIM_TRGO_OC2REF;
 sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
 if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
 {
  Error_Handler();
 }
 sConfigOC.OCMode = TIM_OCMODE_PWM1;
 sConfigOC.Pulse = 4000;
 sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
 sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
 sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
 sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
 if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
 {
  Error_Handler();
 }
 sConfigOC.OCMode = TIM_OCMODE_ACTIVE;
```

```
  sConfigOC.Pulse = 3333;
  if (HAL_TIM_OC_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
  sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
  sBreakDeadTimeConfig.Break2Filter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @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_SlaveConfigTypeDef sSlaveConfig = {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 = 71;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 64999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

```
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_TRIGGER;
  sSlaveConfig.InputTrigger = TIM_TS_ITR0;
  if (HAL_TIM_SlaveConfigSynchro(&htim2, &sSlaveConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_OC2REF;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 4000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_ACTIVE;
  sConfigOC.Pulse = 3333;
  if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM15 Initialization Function
  * @param None
  * @retval None
  */
```

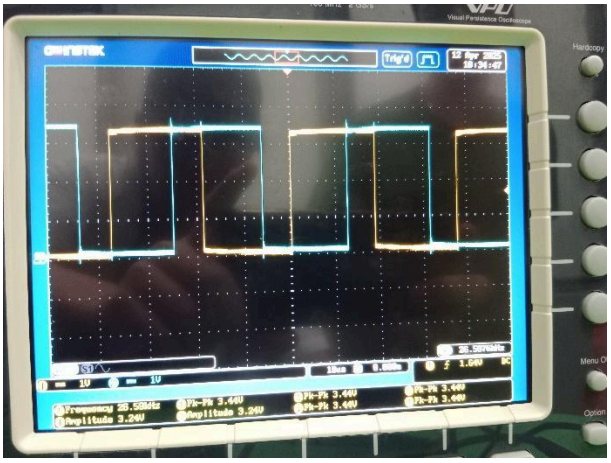
```
static void MX_TIM15_Init(void)
{

  /* USER CODE BEGIN TIM15_Init 0 */

  /* USER CODE END TIM15_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_SlaveConfigTypeDef sSlaveConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM15_Init 1 */

  /* USER CODE END TIM15_Init 1 */
  htim15.Instance = TIM15;
  htim15.Init.Prescaler = 71;
  htim15.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim15.Init.Period = 64555;
  htim15.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim15.Init.RepetitionCounter = 0;
  htim15.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim15) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim15, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim15) != HAL_OK)
  {
    Error_Handler();
  }
  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_TRIGGER;
  sSlaveConfig.InputTrigger = TIM_TS_ITR1;
  if (HAL_TIM_SlaveConfigSynchro(&htim15, &sSlaveConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim15, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 4000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
```

```
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim15, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim15, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM15_Init 2 */

  /* USER CODE END TIM15_Init 2 */
  HAL_TIM_MspPostInit(&htim15);

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 38400;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
```

```
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @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_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
```

```
/* USER CODE END 4 */

/**
  * @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 */
```

Después de cargar el código a la tarjeta. Con la ayuda de un osciloscopio verificamos los 2 PWM con un desface de 120 grados.



Anexo C. MC WORKBENCH

El MC Workbench para STM32 es un software de PC que simplifica la configuración de la biblioteca de firmware de control de motores (FOC) para microcontroladores STM32. Permite a los desarrolladores configurar de manera eficiente los parámetros de la aplicación de control de motores, como la biblioteca PMSM (motor de imán permanente síncrono), y genera los archivos de proyecto necesarios para su integración con STM32CubeMX.

Legacy or partial content

Up to date of Full content

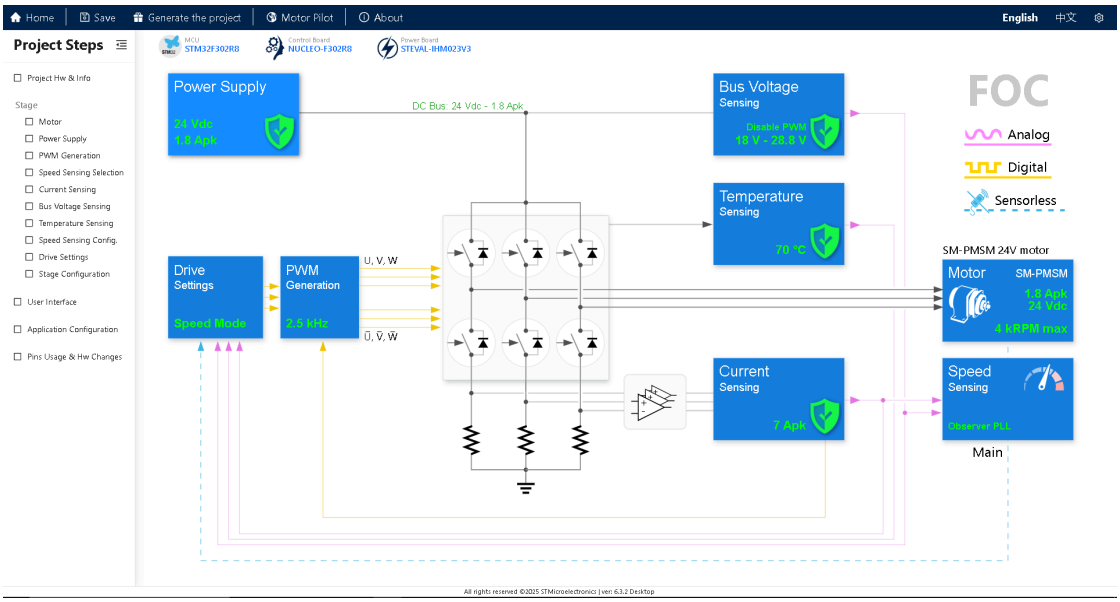
STM32 MC SDK compatibility		STM32 MC SDK 5.4 (legacy)	STM32 MC SDK 6.4 (May-2025 release)
STM32 series supported		9 series (STM32F0, F1, F3, F7, L4, G0, G4, H7) + STSPIN32	12 series (STM32F0, G0, C0, L4, F3, G4, F4, F7, H5, H7, U5) + STSPIN32
Hardware boards supported		ST Boards (control & power)	ST Boards (control & power, board manager) User boards (JSON text format description) Board designer tool
Algorithms	BLDC 6-steps	Voltage and Current drive 6-step solution (examples)	Legacy support + full support of STM32F0, F3, F4, G0, C0, G4 series Enhanced Rotor Speed and position feedback (sensor and sensor-less mode) On the Fly and Initial position detection
	BLDC FOC	Legacy support	Legacy and STM32U5 support STM32C0 single shunt support New High sensivity Observer (HSO) support STM32 ZeST support (for selected customers only)
	Observer	ST Observer (STO PLL)	ST Observer (STO PLL) High Sensivity Observer (HSO)
	STM32 ZeST	-	STM32 ZeST: Max Torque at Zero Speed (available for selected customers only)
Variable monitoring		Simple variable monitoring tool	Motor pilot tool (advanced variable monitoring, control and new trigger functionality)
Motor parameters measurements		-	Motor profiler tool

NB: The MCSDK5.Y is no longer available for download. The latest MCSDK6 version must be considered instead.

Los microcontroladores STM32 ofrecen el rendimiento de los núcleos Arm® Cortex® - M ,

estándar de la industria . Funcionan en modo de control vectorial o FOC, ampliamente utilizados en variadores de alto rendimiento para aire acondicionado, electrodomésticos, drones, automatización industrial y de edificios, aplicaciones médicas y de bicicletas eléctricas.

El firmware STM32 MC SDK (kit de desarrollo de software de control de motor) incluye la biblioteca de firmware del motor síncrono de imán permanente (PMSM) (control FOC) y el banco de trabajo de control de motor STM32 (para configurar los parámetros de la biblioteca de firmware FOC), con su interfaz gráfica de usuario (GUI). El banco de trabajo de control de motor STM32 es un software para PC que reduce el esfuerzo de diseño y el tiempo necesarios para la configuración del firmware FOC del PMSM STM32. El usuario genera un archivo de proyecto mediante la interfaz gráfica de usuario e inicializa la biblioteca según las necesidades de la aplicación. Algunas variables del algoritmo se pueden monitorizar y modificar en tiempo real.



Esto sería el menú del MC WORKBENCH que me permite seleccionar el microcontrolador, que estoy utilizando en este caso el STM32F302R8 , el módulo de potencia STEVEAL-IHM023V3 y un motor trifásico. El automáticamente genera unas opciones predeterminadas que pueda modificar, para que funcione correctamente el motor. En este caso no

lo modificamos y procedemos a generar el código en STM32cubeMX donde nos permite modificar el código y expórtalo directamente en STM32cudeIDE.