Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

**Trabajo Fin de Grado**

# Diseño de driver para transmisión de información con lámparas led



**Autor:** Javier de Marcos Alonso

**Tutor/es:** José Luis Lázaro Galilea
Álvaro de la Llana Calvo

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

# Trabajo Fin de Grado
# Diseño de driver para transmisión de información con lámparas led

Autor: Javier de Marcos Alonso

Tutor/es: José Luis Lázaro Galilea
Álvaro de la Llana Calvo

TRIBUNAL:

Presidente: Daniel Pizarro Pérez

Vocal 1º: M.ª Del Rosario Fernández Ruiz

Vocal 2º: José Luis Lázaro Galilea

FECHA: 22 de Julio del 2021

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE CÓDIGOS



# ÍNDICE DE TABLAS

# i. Resumen

El proyecto consiste en el diseño de un driver para transmitir información modulada mediante los sistemas de iluminación de un edificio. El objetivo es poder modular la luz emitida por las luminarias (de emisores led) de las diferentes estancias, de manera que se pueda transmitir información modulada, pero sin que el usuario perciba alteraciones en la iluminación.

A través de un microcontrolador de bajo coste incorporado en la luminaria generaremos las señales que transmitirán la información (PWM) y serán la entrada del driver de conmutación. Gracias a la tecnología Wi-Fi podremos configurarlas sin contacto directo estableciendo una red a la que se conectarán las luminarias y el usuario que desee configurarla.

Se trata de desarrollar unas luminarias que, para llevar a cabo un sistema de posicionamiento VLP solo se requiera sustituir las lámparas actuales por las que aquí se desarrollan.

# ii. Summary

The project consists the design of a driver to transmit modulated information through the lighting systems of a building. The aim is to be able to modulated the light emitted by the luminaries (led emitters) in the different rooms, so that modulated information can be transmitted, but the user can't perceiving any alterations in the lighting.

Through of a low cost microcontroller incorporate in the luminaries, we will generate the signals that will transmit the information (PWM) and will be the inputs of the switching driver. Thanks to Wi-Fi technology we will be able to configure them without direct contact by establishing a network to which the luminaries and the user who wishes to configure them will be connected.

The aim is to develop luminaires which, in order to implement a VLP positioning system, only require to be replace the existing lamps by the ones developed here.

## iii. Resumen extendido

El objetivo de este trabajo de fin de grado es el diseño de un *driver* para transmitir información modulada mediante los sistemas de iluminación de los edificios. Se basa en poder modular la luz emitida por las luminarias (de emisores led) de las diferentes estancias de un edificio, de manera que se pueda transmitir información, pero sin que el usuario perciba cambios de iluminación, es decir, manteniendo una iluminación media constante.

El sistema completo constará de tres partes claramente diferenciadas. Un microcontrolador de bajas prestaciones con tecnología inalámbrica para poder configurar las señales que se transmitirán, sin contacto directo, desde un ordenador o un dispositivo móvil. Una segunda parte de conmutación basada en un controlador de puerta y transistores MOSFET que se encargarán de conmutar la corriente que circulará por los led y, por último, una fuente de tensión conmutable que se encargará de alimentar el circuito completo generando todas las tensiones necesarias.

Inicialmente se hará un pequeño estudio de la selección del microcontrolador, así como de los entornos disponibles para poder trabajar con ellos. Detallaremos los motivos de la elección del chip, así como la descripción del módulo en concreto, aportando las tecnologías, periféricos y modos de funcionamiento que soporta.

Una vez realizada la elección pasaremos a desarrollar la aplicación software que se encargará de la transmisión de información mediante modulación por ancho de pulsos (PWM), así como de la configuración de estas mismas mediante tecnología inalámbrica.

Generaremos una red a través de un ordenador o un *router* (el cual será nuestro punto de acceso) a la que se conectará el microcontrolador (función de servidor) y el usuario (función de cliente) que desee establecer la configuración. Con los datos deseados se creará una trama TCP (protocolo de comunicación establecido para el envío y la recepción de los datos) que el cliente mandará y el servidor se encargará de decodificar, actualizar y almacenar.

Describiremos el material utilizado para su implementación y el desarrollo en sí, aportando resultados teóricos y prácticos.

Por último, con respecto a las partes de la lógica de conmutación y la fuente de tensión conmutable, mencionaremos el procedimiento de diseño de cada una de ellas, así como una descripción *hardware* de los componentes utilizados y un apartado para los resultados obtenidos.

# iv. Acrónimos y abreviaturas

- ESP-IDF: *Espressif IoT Development Framework*.
- MESH: Protocolo de red construido sobre el protocolo Wi-Fi.
- PWM: *Pulse width modulation* (modulación por ancho de pulsos).
- TCP: *Transmission control protocol* (protocolo de control de transmisión).
- MIPS: Millones de instrucciones por segundo (medida de rendimiento).
- MHz: Mega hercios (medida de frecuencia).
- Mbit/s: Mega bits por segundo (medida de transmisión).
- GND: *Ground* (toma de tierra).
- GPIO: *General purpose input/output* (pin digital de propósito general de entrada y salida).
- PLL: *Phase-locked loop* (lazo de seguimiento de fase).
- RTC: *Real time clock* (reloj de tiempo real).
- UART: *Universal asynchronous receiver-transmitter* (transmisor-receptor asíncrono universal).
- *I2C: Inter-integrated circuit, synchronous serial communication bus* (circuito inter-integrado, bus serie de comunicación síncrono).
- SPI: *Serial peripheral interface* (bus de interfaz de periféricos serie).

# 1. Introducción y objetivos

La tecnología se encuentra en constante evolución. Cada día en la prensa de todo el mundo observamos una gran cantidad de avances científico-tecnológicos.

Una de las ramas que se encuentra en evolución constante es la de la energía, cualquier industria o multinacional del mundo desearía poder suministrar o poseer energía ilimitada a coste cero. Este hecho es físicamente imposible, pero los ingenieros nos dedicamos a acercarnos lo máximo posible a lo que llamamos "mundo ideal", es decir, un mundo donde las cosas no consumen energía, los componentes eléctricos no disipan potencia ni tienen pérdidas, etc.

A día de hoy tenemos una gran contaminación lumínica en nuestras ciudades debido a la gran cantidad de luz que requieren todos los sectores del ámbito doméstico o industrial, y esto conlleva un gran gasto de suministros. Debido a este problema y dada la necesidad de disminuir el coste que supone, han surgido diversas alternativas, como las bombillas de bajo consumo y las bombillas led.

Probablemente la mayoría de la gente tendrá en sus domicilios bombillas de led, y otro gran grupo de gente no las comprará porque son de mayor precio que las convencionales. Directamente esta evidencia es cierta, pero si vamos más allá de lo que nos indica una etiqueta con un simple precio, veremos que no es cierto.

Si comparamos las especificaciones técnicas de una bombilla halógena de 20W [1] con una bombilla led de 2,6W [2], veremos que la primera es tres veces más barata que la segunda, pero la vida útil es veinticinco veces menor.

Haciendo un cálculo rápido para obtener el precio por hora de vida de cada una de las bombillas, obtendremos que la halógena tiene un coste por hora de 0,094 céntimos, mientras que en la led es de 0,0139. Por tanto, se demuestra que la evidencia anterior es falsa, puesto que, por hora de vida útil, la bombilla led es casi 7 veces más barata que la halógena.

Las ventajas de este tipo de bombillas van más allá del precio, el consumo puede llegar a ser de hasta un 70% inferior al de una incandescente, son más respetuosas con el medio ambiente, con lo que la diferencia de las luminarias led con respeto a las competidores es claramente desfavorable para el resto, etc.

Este tipo de bombillas necesita el uso de un *driver* (o fuente de alimentación), la función de éste es la de transformar la corriente alterna de la red, en España (230V a 50Hz), en corriente continua. Por este motivo, se propone el desarrollo de un *driver* electrónico de emisores led para iluminar las estancias de un edificio que además de la conversión AC/DC pueda incorporar entradas de señales moduladoras.

El campo de aplicación es muy amplio, y aunque en nuestro caso vayamos a aplicarlo a un edificio en concreto, podemos llevarlo a nuestros hogares para introducir diversas mejoras como poder regular la intensidad de la luz o apagarlas remotamente. Otras aplicaciones que ahora se están desarrollando tienen que ver con la comunicación de información (VLC –Visible Light Communication-) o con el uso de las señales emitidas para posicionar agentes (VLP –Visible Light Positioning-).

El objetivo principal del proyecto es el diseño de un *driver* con comunicación inalámbrica con calidad suficiente que nos asegure la estabilidad de los led ante diversos inconvenientes como pueden ser saltos de tensión en la red. Además, en nuestro caso también tendremos como objetivo la transmisión de información modulada, con el uso de un microcontrolador empotrado sin que se vea comprometida la calidad de la luz (iluminación).

El campo de aplicación será desarrollar "bombillas inteligentes" que se conecten a un módulo central o punto de acceso (red generada por un equipo o un *router*) y que realicen en paralelo, tanto el proceso de iluminación, como el de envío y recepción de información. Adicionalmente, podremos comunicarnos con ellas a través de un protocolo preestablecido desde cualquier punto de acceso o a través de una aplicación preinstalada en nuestro *smartphone*. Esto es de gran utilidad, puesto que si podemos conectarnos a través de un módulo de tecnología inalámbrica podremos manipularlas a nuestra conveniencia para transmitir información determinada.

Los resultados de este proyecto serán incluidos en el proyecto de investigación GUIA financiado por la Junta de Comunidades de Castilla La Mancha, para llevar a cabo el posicionamiento dentro de naves logísticas, centros comerciales, lugares de ocio, etc. En primer lugar, se utilizará en una aplicación que se está llevando a cabo en el Museo Provincial de Guadalajara.

Como objetivos específicos del proyecto destacamos los siguientes:

- Iluminación y trasmisión de información con luminarias led configurables.
- Reducción de coste a la hora de elegir un sistema de iluminación determinado.
- Maximizar el control sobre la iluminación de una bombilla, etc.
- Transmisión de información modulada sin verse afectada la calidad de la luz a través de señales PWM (modulación por ancho de pulsos, en inglés *pulse width modulation*).

# 2. Descripción del módulo ESP32

## 2.1. ESP32-WROOM-32

El módulo ESP32 es un System on Chip (SoC) creado y desarrollado por Espressif Systems, aunque es fabricado por TSCM (Taiwan Semiconductor Manufacturing Company), que fue la primera empresa del mundo en dedicarse exclusivamente a la fabricación de semiconductores.

A día de hoy estos módulos son una opción interesante en microcontroladores con tecnología inalámbrica o incluso como punto de acceso a la red para otros microcontroladores que no posean esta tecnología. En la Figura 1 se muestra el diagrama de bloques del microcontrolador WROOM-32

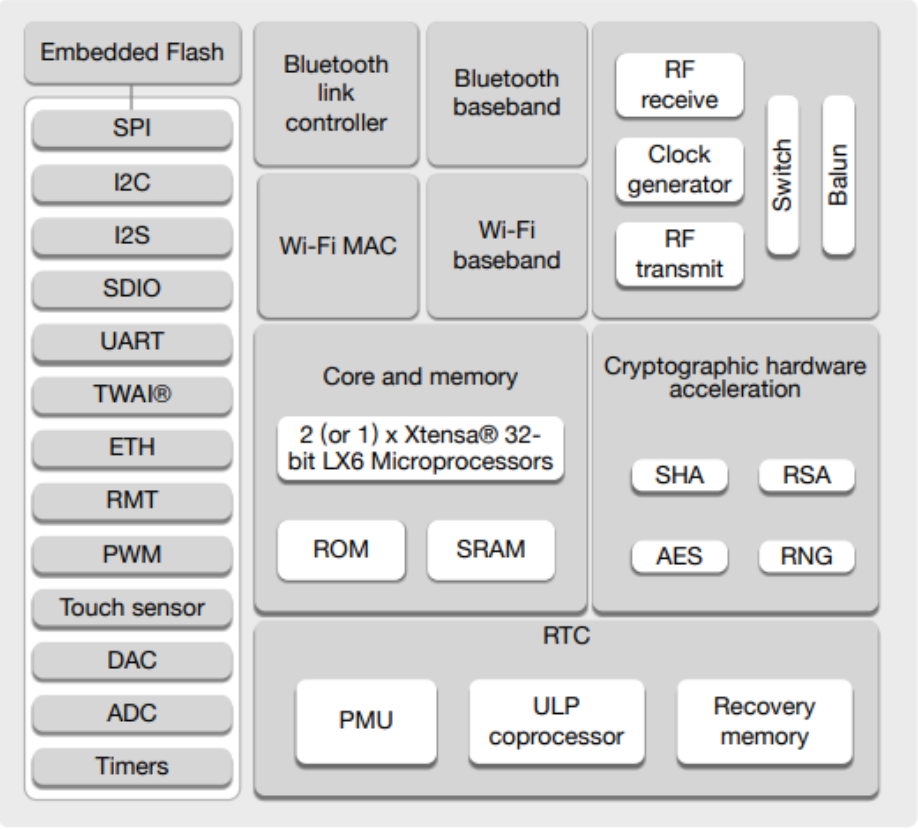


*Figura 1. Diagrama de bloques WROOM-32 [3].*

En nuestro caso haremos uso del módulo WROOM-32, ya que es un módulo de la familia ESP32 con una amplia memoria *flash* y con antena MIFA. Internamente tiene el chip ESP32-D0WDQ6, que posee tecnología Wi-Fi b/g/n y BT/BLE modo dual, además de poseer un *dual core*.

La propia nomenclatura nos aporta mucha información sobre las características del propio chip; si indagamos en la *datasheet* veremos qué significa cada parte del nombre que recibe. En la Figura 2 se muestra dicha nomenclatura.

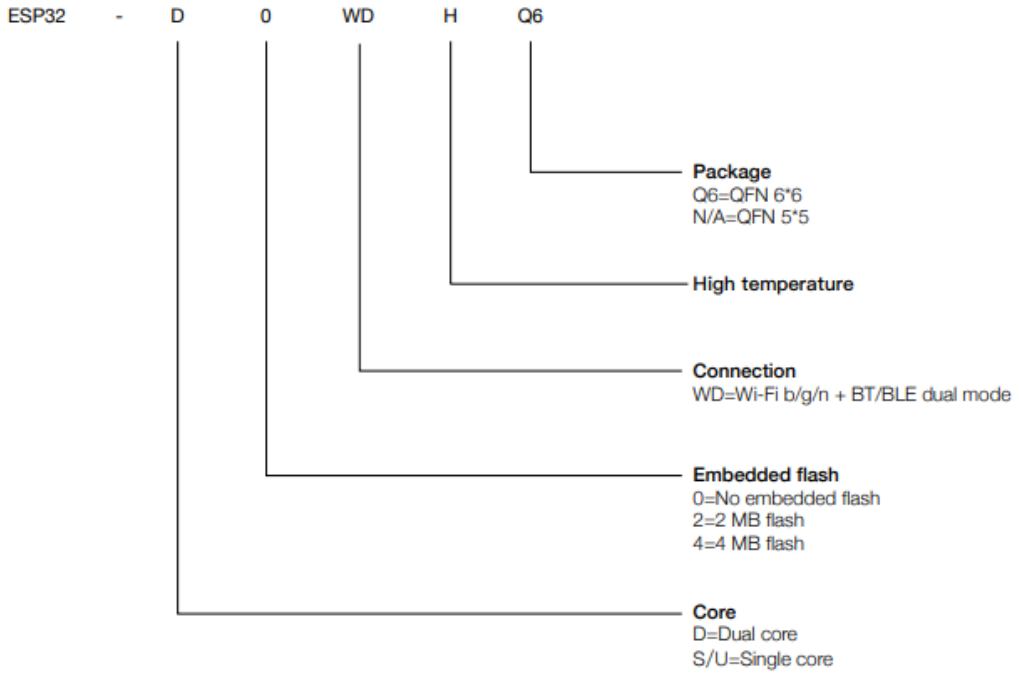


*Figura 2. Nomenclatura chip ESP32 [3].*

Como procesador tenemos un Tenslica Xtensa dual-core 32-bit LX6 con un rendimiento de 600 MIPS (millones de instrucciones por segundo) y reloj programable desde los 80 MHz hasta los 240 MHz.

En lo que a memoria se refiere, posee una serie de memorias internas:

- ROM de 448 KB para funciones del núcleo.
- SRAM de 520 KB para datos e instrucciones.
- RTC FAST SRAM de 8KB para almacenar datos.
- RTC SLOW SRAM de 8KB.

Para el desarrollo del proyecto haremos uso de la placa de desarrollo DevKitC (modelo 30 pines). Se trata de la placa más comercial de la familia de Espressif Systems y que incorpora en su interior el módulo WROOM-32 con interfaz micro USB. Utiliza un CP210x como conversor USB-Serie que soporta velocidades de descarga de hasta 3 Mbit/s. Se puede alimentar de 3 maneras distintas, con el micro USB que es la alimentación por defecto o bien a través de los pines correspondientes con 3V3/GND o 5V/GND. Además, posee dos botones para *reset* (EN) y para descarga en placa (BOOT). La Figura 3 muestra la tarjeta de desarrollo usada en el proyecto.

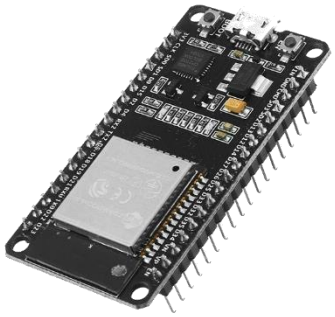

*Figura 3. ESP32-DevKitC v1 (30 pines) [12].*

## 2.2. Especificaciones

Ya hemos introducido algunas características del chip, módulo y placa que usaremos a lo largo del proyecto como el procesador, las memorias internas, tipos de alimentaciones de la placa, etc. Ahora vamos a detallar técnicamente la mayoría de los bloques que podemos observar en la Figura 1, incluyendo tecnologías permitidas y periféricos.

El ESP32 soporta las tecnologías inalámbricas Wi-Fi y Bluetooth, con respecto a Wi-Fi soporta el estándar 802.11 b/g/n con velocidades máximas de 150 Mbit/s, mientras que la versión bluetooth soportada es la v4.2 BR/EDR.

Cuenta con varios relojes de sistema, el más importante es el de la CPU cuya frecuencia de reloj por defecto es de 160 MHz, gracias a un cristal de cuarzo externo de 40 MHz que se conecta internamente a un PLL También posee un RTC (reloj de tiempo real, en inglés *real time clock*) y un oscilador interno de 8 MHz.

Alberga una gran cantidad de periféricos que pueden ser de gran utilidad para diversos proyectos. Dispone de 4 temporizadores (*timer*) de propósito general de 64 bits y *prescaler* de 16 bits, utiliza por defecto un reloj de 80 MHz. Gracias a la resolución tal alta de los temporizadores podemos realizar contadores de tiempo del rango de días o años. Pueden ser configurados para que cuenten de manera ascendente o descendente, además de generar una interrupción por nivel o flanco.

Posee dos conversores analógico-digitales de aproximaciones sucesivas de 12 bits con hasta 18 canales de conversión, incluso funciona cuando el dispositivo se encuentra en ahorro de energía. La velocidad de conversión es configurable para los distintos canales, llegando al orden de MSamples/s. Por el contrario, solo posee un conversor digital analógico de 8 bits con dos canales, pudiendo convertir dos señales al mismo tiempo.

Es capaz de generar 16 señales PWM gracias a que cualquier GPIO (pin digital de propósito general de entrada y salida, en inglés *general purpose input/output*) puede ser configurado como salida PWM (a excepción de los que solo son entradas) y a los 16 canales internos que posee el microcontrolador, en sus modos *high-speed* y *low-speed*. También posee 10 puertos que pueden desempeñar la función de detección capacitiva y

16 puertos que pueden funcionar como RTC para despertar al microcontrolador cuando se encuentre suspendido.

Finalizamos con el apartado de comunicaciones. El ESP32 posee 3 interfaces UART (transmisor-receptor asíncrono universal, en inglés *universal asynchronous receiver-transmitter*) de hasta 5 Mbit/s, así como dos interfaces I2C (bus serie de comunicación síncrono, en inglés *inter-integrated circuit, synchronous serial communication bus*) configurables como maestro o esclavo y relojes de hasta 5 MHz, dos interfaces I2S (sonido inter-ic, bus serie eléctrico para conectar dispositivos de audio, en inglés *inter-ic sound, electrical serial bus used for connecting digital audio devices)* y 3 interfaces SPI (bus de interfaz de periféricos serie, en inglés *serial peripheral interface)* de hasta 80 MHz.

## 2.3. Entorno de desarrollo (IDE)

Como bien sabemos, un entorno de desarrollo integrado, en inglés *Integrated development environment* (IDE), es una aplicación informática que pone a disposición del ingeniero una serie de servicios/características para facilitar el desarrollo de un determinado *software*.

Principalmente, un IDE consiste en un editor de texto para la modificación del código fuente, herramientas de automatización y un depurador. El objetivo principal de un entorno de desarrollo es maximizar la productividad reduciendo la configuración para reconstruir múltiples utilidades de desarrollo.

Es importante estudiar distintos entornos para conocer cuál se ajusta más a nuestras necesidades, ya que hay características que son específicas de cada uno de ellos. Por ejemplo, algunos se construyeron con la idea de estar dedicados a un único lenguaje de programación o poseen un compilador e interprete propios como es el caso de *NetBeans* y *eclipse*. Algunos de estos entornos soportan lenguajes alternativos a través de un *plug-in* que puede ser instalado en el mismo IDE al mismo tiempo. Esto es una gran ventaja para el ingeniero, puesto que mejoramos la productividad y la simplicidad de los proyectos desarrollados.

En nuestro caso, tenemos una serie de necesidades *hardware* y otras *software*. Para poder seleccionar el entorno de desarrollo adecuado, debemos centrarnos en los requerimientos del *software*.

### 2.3.1. Selección del entorno

Como detallamos anteriormente, hay diversos entornos entre los que podemos optar. En nuestro caso, siguiendo las recomendaciones del fabricante, utilizaremos el entorno ESP-IDF con el *software* de e*clipse*; el motivo por el cual se ha decidido utilizar dicho entorno es por su simplicidad en la comprensión, así como por las funcionalidades que aporta.

A continuación, vamos a detallar las características de algunos uno de ellos. De esta manera, concienciaremos al usuario de todas y cada una de las funcionalidades de los diversos entornos, aportando las ventajas de cada uno de ellos.

#### *2.3.1.1. ESP-IDF*

La familia de chips Espressif Systems posee su propio entorno de desarrollo y un gran número de recursos y facilidades a la hora de desarrollar nuestra aplicación basándonos en el módulo ESP32.

El lenguaje de programación utilizado por el fabricante es C/C++. Se encuentra dotado de una gran cantidad de funciones amigables y previamente definidas, que facilitan la programación del dispositivo y una gran documentación técnica, la cual, el fabricante pone a nuestra disposición a través de su página web.

Una de las ventajas del uso de dicho entorno es que no es necesario un *software* de terceros para el desarrollo de cualquier aplicación, puesto que desde un terminal es accesible (con el *toolchain* necesario).

Nosotros utilizaremos *eclipse*, con el complemento ESP-IDF (también disponible para *VS Code*). El motivo principal es que, con el uso de un terminal como *VS Code*, la simplicidad que proporciona *eclipse* la perdemos; también, la compilación desde el propio terminal conlleva más lentitud. Aunque también tendría aspectos positivos, como es el caso de un menú en el cual podemos configurar una gran cantidad de características y aspectos del módulo ESP, como pueden ser tabla de particiones, *tick_rate* del FreeRTOS, funciones de seguridad, etc. Además del entorno ESP-IDF, el fabricante ha desarrollado otra gran cantidad de entornos cuyo uso se hace más específico para determinadas aplicaciones y funcionalidades. Es el caso de ESP-ADF (*Audio development framework*), cuyo uso se encuentra especializado en aplicaciones de audio, ESP-MDF (*Mesh development framework*), en este caso, su uso está especializado para aplicaciones que hagan uso del protocolo MESH.

MESH, es un protocolo de red que fue construido sobre el protocolo Wi-Fi. Permite que una larga lista de dispositivos que se encuentran dispersos en un área determinada, se interconecten a través de una única WLAN. A diferencia de la red Wi-Fi tradicional, en la cual un único nodo central (o punto de acceso) se encuentra conectado al resto de nodos (o estaciones), el protocolo MESH permite que los nodos que se encuentren alejados del nodo central se puedan conectar a nodos vecinos, que serán los encargados de transmitir hacia el resto de nodos. Esto permitirá que una red bajo el protocolo MESH posea un área de trabajo mucho mayor [7]. En la Figura 4 se muestra la arquitectura de una red Wi-Fi, y en la Figura 5 se muestra la arquitectura del protocolo MESH.

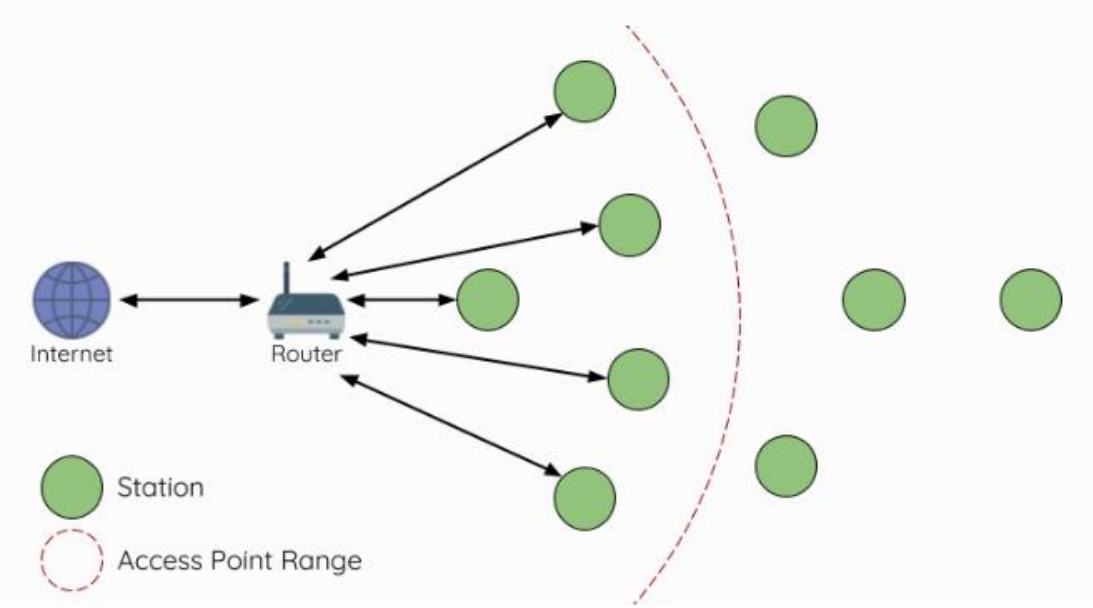


*Figura 4. Arquitectura Wi-Fi [7].*

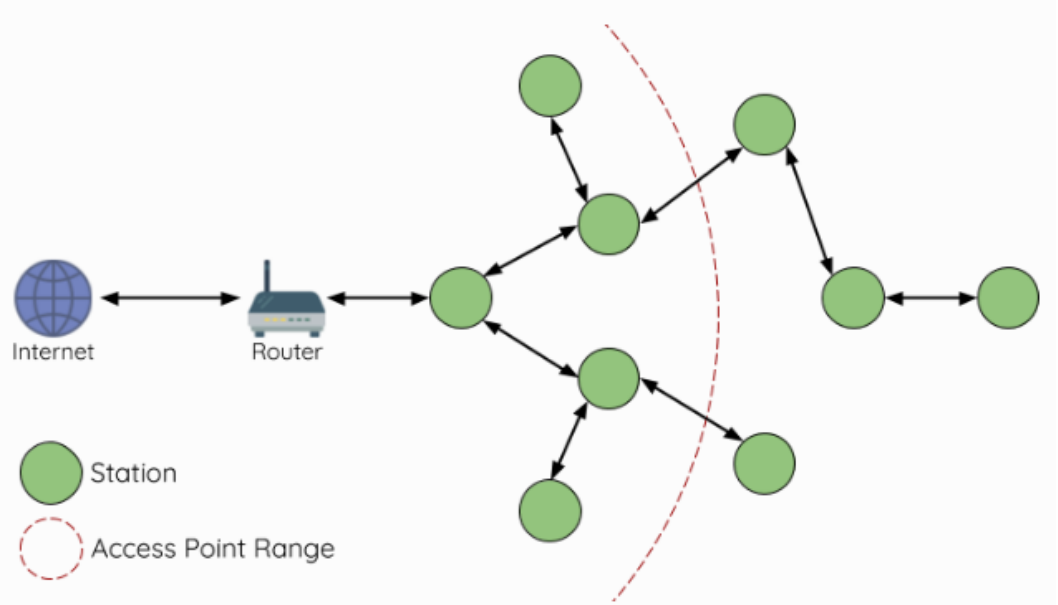


*Figura 5. Arquitectura MESH [7].*

El fabricante nos proporciona una amplia y detallada guía de instalación y configuración para poder poner el entorno a punto. Se encuentra disponible en su página web oficial [6].

#### *2.3.1.1.1. Requisitos*

Tanto si utilizamos *eclipse* como *VS Code* con el *plug-in* de ESP-IDF, los requisitos que debemos cubrir son los siguientes:

- Versión de Java 11 o superior.

- Versión de Python 3.5 o superior.
- GIT.
- Repositorio ESP-IDF.
- Eclipse 2020-12 CDT o superior/ *VS Code*.

En la Figura 6 se muestra la interfaz de *eclipse*. En la parte izquierda se puede ver el directorio de trabajo seleccionado y en la parte central, el código fuente.

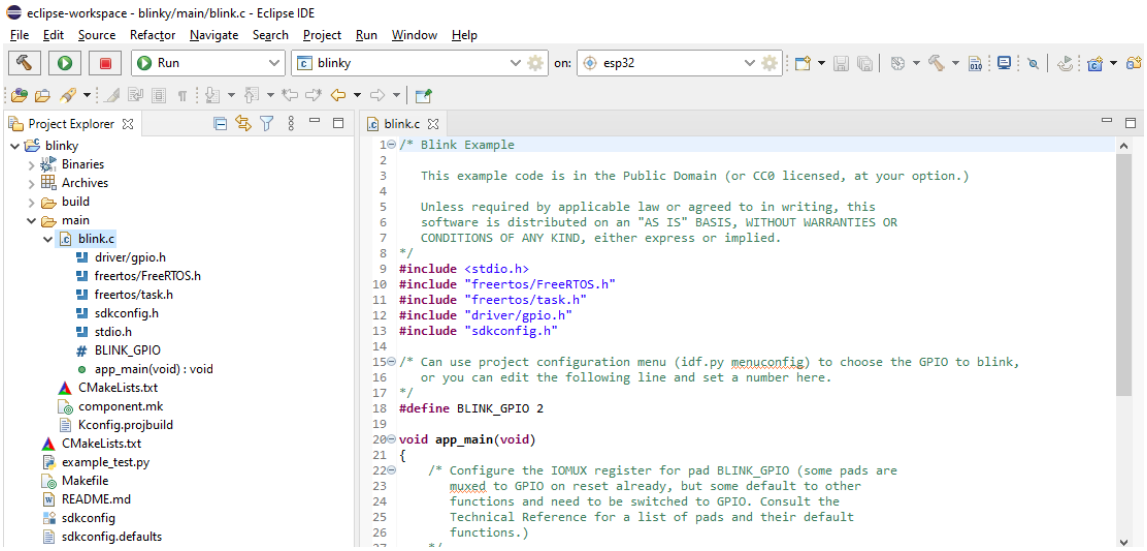


*Figura 6.Entorno ESP-IDF sobre eclipse.*

#### *2.3.1.2. Arduino IDE*

Posiblemente, el entorno de *Arduino* sea uno de los más demandados para poder trabajar con los módulos ESP, y ello es debido a su gran facilidad de uso, así como de una interfaz amigable para el desarrollo de aplicaciones.

Espressif Systems desarrolló *Arduino Core*. Gracias a este diseño, todas las funcionalidades y ventajas que nos aporta el entorno de ESP-IDF, las tendremos disponibles en este otro entorno.

*Arduino* es un entorno multiplataforma de código abierto, apoyado por una gran comunidad que día a día mejora las prestaciones y pone a disposición de cualquier usuario los avances en la herramienta, este es el motivo principal de su gran éxito.

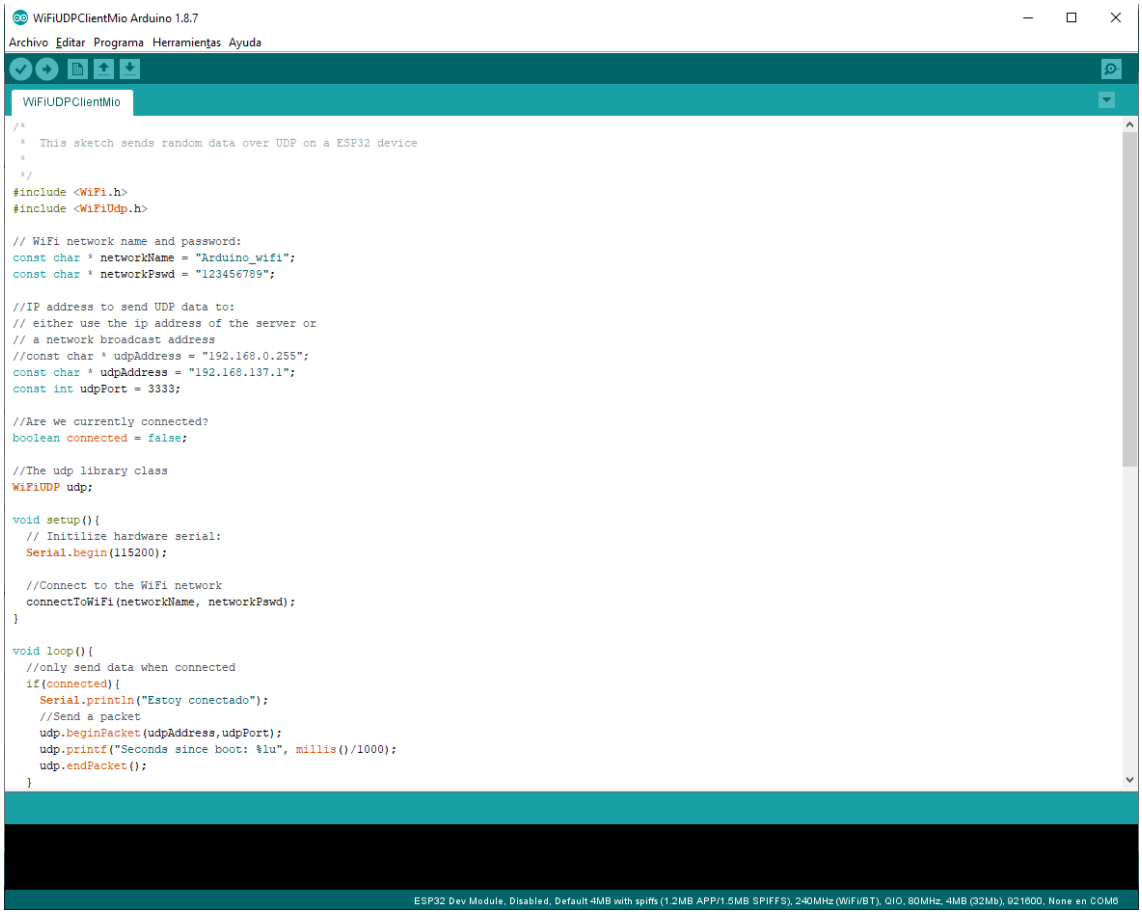


*Figura 7.Entorno Arduino.*

#### *2.3.1.2.1. Requisitos*

En el caso del entorno *Arduino*, los requisitos que debemos cubrir son:

- Equipo con sistema operativo Windows (disponible también para Mac y Linux).
- Placa de desarrollo compatible con el entorno.
- Cable de transmisión de datos para conectar la placa al ordenador (normalmente micro USB B).

### *2.3.1.3. PlatformIO*

*PlatformIO* es un entorno de desarrollo pensado para aplicaciones IoT. Es un entorno de código abierto el cual se ejecuta sobre un editor de texto, como podría ser *Visual Studio Code* o *Sublime Text*. La principal ventaja que posee es su compatibilidad con la mayoría de las placas existentes en el mercado y su gran cantidad de librerías.

Al igual que *Arduino*, posee una interfaz gráfica amigable, además de otras características destacables como un gestor de librerías, diversidad de opciones multiplataforma, o

detección de código automático y permite además el desarrollo remoto. Posee un depurador propio.

##### *2.3.1.3.1.* *Requisitos*

Para poder hacer uso del entorno, necesitamos una serie de requerimientos:

- Equipo con sistema operativo Windows (disponible también para Mac y Linux).
- Editor de texto sobre el que se ejecutará *PlatformIO* (*VS Code, Sublime Text, eclipse*…).
- Versión de Python 3.5 o superior.
- Placa de desarrollo compatible con el entorno.
- Cable de transmisión de datos para conectar la placa al ordenador (normalmente micro USB B).

### 2.3.2. ESP-IDF con Eclipse

De los entornos expuestos, se han probado *eclipse* y *VS Code* con el *plug-in* de ESP-IDF y *PlatformIO*. Finalmente, se ha seleccionado el primero por los motivos referidos en la sección 2.3.1

ESP-IDF (*Espressif IoT Development Framework*) es el repositorio oficial desarrollado por la familia de Espressif Systems. Se basa en el código original de FreeRTOS para el desarrollo y la implementación de todas sus funciones.

FreeRTOS se trata de un sistema operativo embebido de tiempo real y código abierto. El núcleo (o *kernel*) está implementado en lenguaje C, debido a que uno de los objetivos era la portabilidad y legibilidad, aunque existen algunas funciones implementadas en lenguaje máquina (o ensamblador). Proporciona métodos para múltiples procesos, hilos, semáforos etc., y es completamente gratuito para aplicaciones comerciales [27].

#### *2.3.2.1.* *Requisitos previos*

Los requisitos previos comentados en la sección 2.3.1.1.1 son indispensables para el funcionamiento de nuestro entorno. Podemos seguir dos procedimientos para su instalación, el primero de ellos es usar una herramienta que nos facilita el fabricante y con la cual se instalará automáticamente todo lo necesario para el funcionamiento.

El segundo procedimiento es ir instalando manualmente cada uno de los requisitos previos a través de las páginas *web* oficiales de cada uno de ellos. En la sección 9 se incluye un anexo con todo el proceso de instalación detallado paso a paso.

# 3. Desarrollo de la aplicación software

## 3.1. Descripción general

La parte de generación de señales de control del driver de excitación de las luminarias, así como de la recepción de la configuración de la modulación a llevar a cabo por vía Wi-Fi se desarrollará utilizando el módulo ESP32 DevKitC en su versión de 30 pines (Figura 3). Como hemos comentado, se trata de un módulo de la familia Espressif Systems con tecnología Wi-Fi.

La aplicación a desarrollar generará las señales de control, que posteriormente se usarán como entrada para los circuitos de conmutación, así como de recibir por Wi-Fi la configuración de dichas señales (número de señales a emitir, frecuencias, ciclo de trabajo, etc.) y actualizarla.

Las señales de control son PWM (combinando diferentes frecuencias), y para generarlas haremos uso del módulo LED_PWM que posee el microcontrolador. Se trata de un módulo diseñado principalmente para el control de la intensidad de un led, aunque puede generar señales para distintos propósitos. Por otro lado, haciendo uso del módulo Wi-Fi recibiremos la configuración de dichas señales, en concreto recibiremos una trama TCP, ya que es el protocolo de comunicación que vamos a establecer para la transmisión y recepción de los datos.

## 3.2. Material utilizado

### 3.2.1. Ordenador Toshiba Satellite C660-10H

En primer lugar, necesitamos un equipo con el *software* del fabricante instalado, para poder programar nuestra aplicación y descargarla en la placa.

Cualquier equipo que cumpla los requisitos mínimos para ejecutar el *software* es válido para nuestro proyecto [11]. En la Figura 8 se muestra el modelo de PC usado en este trabajo.

*Figura 8. Toshiba Satellite C660-10H [11].*

- Windows® 10 Home Premium 64-bit.
- Procesador Intel® Core™ i3-370M.
- 39,6 cm (15.6”), Toshiba *TruBrite*® HD TFT *High Brightness display with 16: 9 aspect ratio and LED backlighting*.
- Disco duro HDD de 500 GB.
- 4,096 (2,048 + 2,048) MB, DDR3 RAM (1,066 MHz).

### 3.2.2. ESP32 DevKitC

Como hemos mencionado anteriormente, se trata de la placa de la familia Espressif con tecnología Wi-Fi. En la Figura 3 se muestra la placa usada en este trabajo.

Las tarjetas DevKitC, en sus distintas versiones, tienen una serie de características concretas en lo que a pines se refiere, puesto que hay algunos que poseen una funcionalidad predeterminada y otros que solamente se pueden configurar como salida o como entrada. A continuación, mostraremos el *pinout* en la Figura 9 y detallaremos dichas características:

- Pines GPIO34, GPIO35, GPIO36(VN) y GPIO39(VP) solamente funcionan como entrada.
- Pines GPIO2, GPIO4, GPIO12-15, GPIO25-27, GPIO32-36 y GPIO39 entradas ADC.
- Pines GPIO25-26 salidas DAC.
- Pines GPIO21-22 comunicación por I2C.
- Pines GPIO17-18 como UART2 y GPIO1 junto con GPIO3 como UART0.

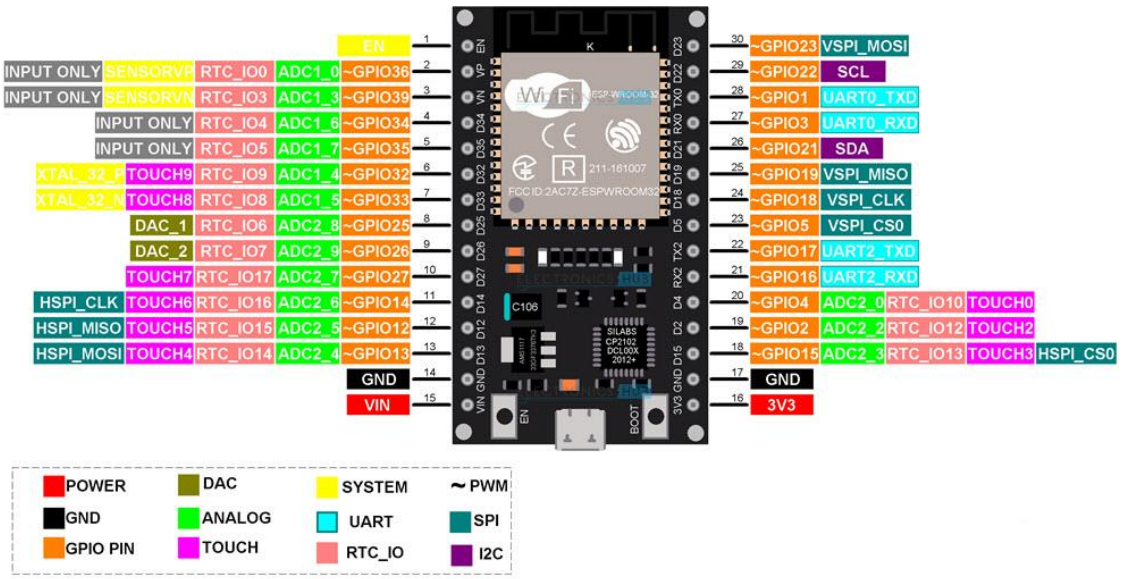


*Figura 9. Pinout ESP32-DevKitC v1 (30 pines) [12].*

Para desarrollar la aplicación, en nuestro proyecto haremos uso de unos pines determinados, los cuales pasamos a enumerar a continuación en la Tabla 1:

| GPIO | Función |
|---|---|
| GPIO26 | PWM INA (original) |
| GPIO25 | PWM INB (inversa) |

*Tabla 1.Reparto de pines en la aplicación.*

## 3.3. Diagrama de bloques

El diagrama de bloques mostrado consta de la conexión del controlador con la conexión Wi-Fi (con el cliente TCP), la tarjeta procesadora y el control del driver mediante las señales generadas por el microcontrolador. Internamente, en el microcontrolador, el diagrama de bloques de las funciones desarrolladas es el mostrado en la Figura 10.

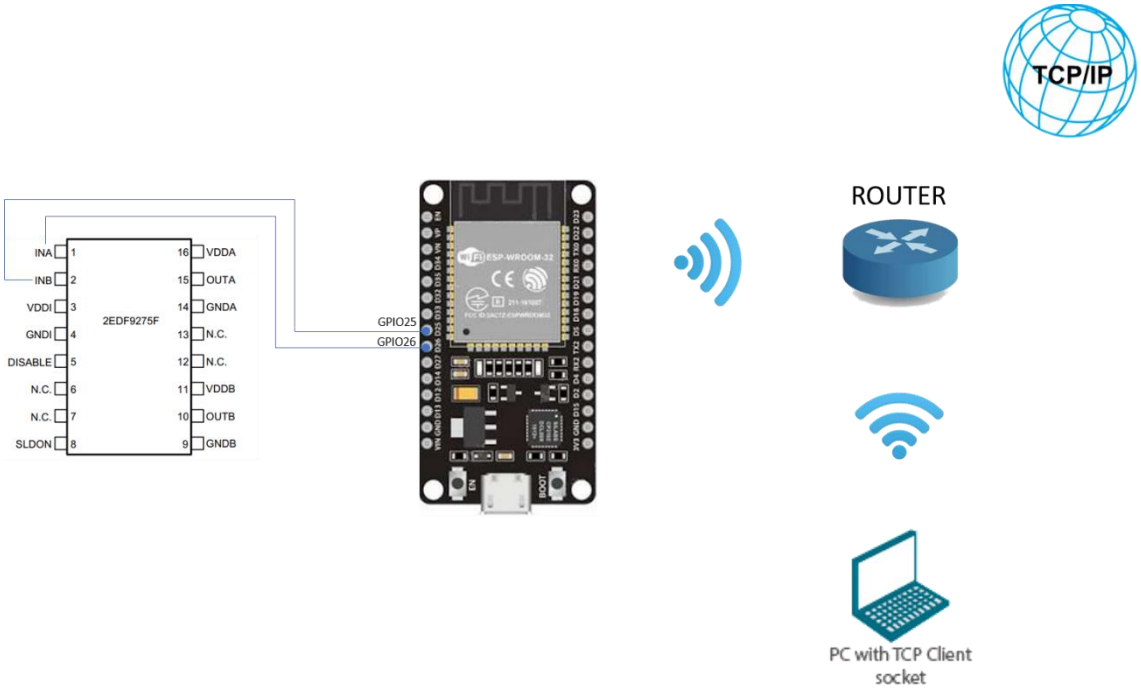


*Figura 10. Diagrama de bloques de la aplicación.*

## 3.4. Implementación del código

Como hemos mencionado en la descripción general, haremos uso de dos módulos concretos del ESP32, LED_PWM y Wi-Fi. Vamos a detallar cada uno de ellos para poder

comprender mejor el uso que haremos de los mismos cada una de las partes en las que se ha dividido el código fuente, así como el propio código.

### 3.4.1. LED_PWM

Se trata de un módulo diseñado específicamente para la generación de señales PWM. Su objetivo principal es el de controlar la intensidad de diodos leds, pero su uso se generaliza a una gran cantidad de aplicaciones, como puede ser el control de un servomotor o de un ventilador de PC, etc.

Contiene dos tipos de canales, denominados como *high-speed* y *low-speed*, en concreto tenemos 8 canales de cada tipo. Además, contiene 4 *timers* internos para controlar cada uno de los tipos de canales existentes (*high-speed* y *low-speed*) que abarcan los 16 canales en total. En la Figura 11 se puede ver el diagrama de bloques interno del módulo LED_PWM.

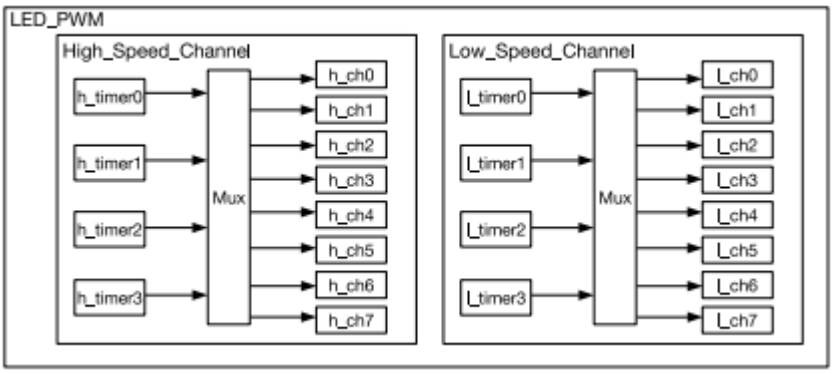


*Figura 11. Diagrama de bloques del módulo LED_PWM [5].*

Los *timers* internos de ambos tipos de canales pueden ser configurados con 2 relojes distintos, concretamente *REF_CLK (1 MHz)* o *APB_CLK (80 MHz)*. Posee un *prescaler* para que la cuenta del *timer* aumente cada vez que llegue a final de cuenta de dicho *prescaler*, en lugar de cada ciclo de reloj. Su configuración es muy sencilla, bastaría con escribir en el registro LEDC_CLK_DIV_NUM_HSTIMERx cuyos 10 bits de mayor peso se corresponden con la parte entera y los 8 bits de menor peso con la parte fraccionaria de la división según la siguiente expresión:

$$LEDC\_CLK\_DIV\_NUM\_HSTIMERx = A\frac{B}{256}$$

siendo A la parte entera y B la parte fraccionaria de la expresión. De la misma manera, podemos configurar la resolución, en número de bits, del ciclo de trabajo deseado para la señal PWM especificándolo en el registro LEDC_HSTIMERx_DUTY_RES. El módulo interrumpirá cuando el *prescaler* alcance el valor $2^{LEDC\_HSTIMERx\_DUTY\_RES} - 1$ y se reseteará el contador. En la Figura 12 se muestra la estructura interna de un canal *high-speed*.

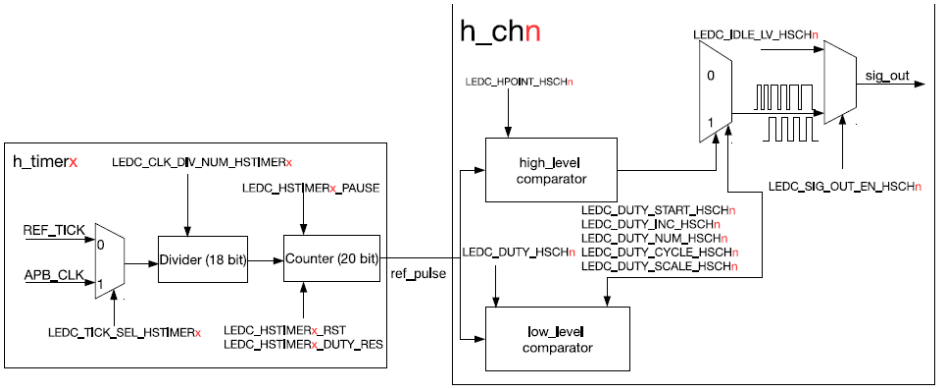


*Figura 12. Diagrama de bloques de canal high-speed [5].*

Para finalizar, tenemos dos últimos registros en los cuales indicaremos el ciclo de trabajo de la señal y el instante inicial en el cual queremos que se inicie la PWM. Esto último es muy útil a la hora de realzar, por ejemplo, señales inversas o señales que queremos que comiencen en un momento determinado de tiempo. Los registros LEDC_DUTY_HSCHn y LEDC_HPOINT_HSCHn configuran dichos aspectos respectivamente. En la Figura 13 se muestra un diagrama de la señal de salida del módulo LED_PWM en función de los valores de *duty* y *hpoint*.

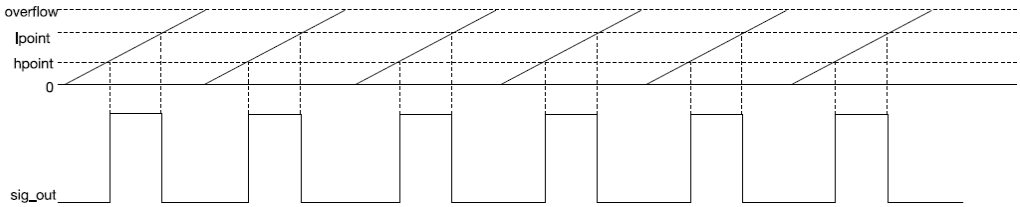


*Figura 13. Señal de salida del módulo LED_PWM [5].*

### 3.4.2. Wi-Fi

Haremos uso de estándar 802.11 b/g/n para la transmisión Wi-Fi, cuya velocidad máxima alcanza los 150 Mbit/s a una frecuencia de 2.4 GHz. Seguridad WPA o WPA2, escáner de puntos de acceso activos y pasivos y distintos modos de funcionamiento como el modo de estación (STA) o modo punto de acceso (AP).

En nuestro caso, usaremos el modo de estación. Dado que se va a generar una red con un *router*, tanto la placa como el ordenador se conectarán a éste, que funciona como punto de acceso.

El protocolo de comunicación establecido entre la placa y el ordenador será TCP, lo que quiere decir que desde el equipo (que actuará como cliente) mandaremos una trama TCP a la placa (que actuará como servidor) con la configuración que deseamos para las señales PWM de salida. En la Figura 10 podemos observar lo que comentamos.

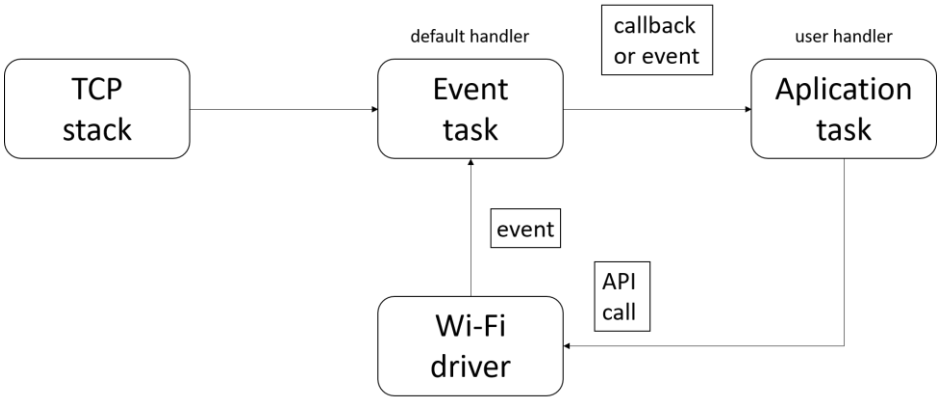


*Figura 14. Modelo de programación Wi-Fi [9].*

En la Figura 14 podemos ver cómo debemos enfocar el uso del Wi-Fi. La aplicación llama a las API del *driver* Wi-Fi para inicializar el protocolo y este las recibe y genera eventos en la aplicación. El *driver* desconoce información acerca del código.

Debemos seguir una serie de pasos a la hora de establecer y liberar una conexión Wi-Fi, que a continuación vamos a enumerar detalladamente:

1) La tarea principal llama a la función *esp_netif_init()*, la cual crea una tarea principal de LwIP e inicializa su trabajo.
2) Después llama a *esp_event_loop_init()* para crear una tarea de evento del sistema.
3) Posteriormente llama a *esp_netif_create_default_wifi_ap() para funcionamiento AP* o *esp_netif_create_default_wifi_sta()* para funcionamiento STA, en función del modelo que deseemos. En nuestro caso creará una estación de enlace de interfaz de red.
4) La tarea principal llama a *esp_wifi_init()* para crear la tarea del driver e inicializarla.
5) Ahora llamaremos a la función *esp_wifi_set_mode()* a la cual le pasamos como argumento el modo de funcionamiento (*WIFI_MODE_STA, WIFI_MODE_AP o WIFI_MODE_APSTA*).
6) Llamamos a *esp_wifi_start()* para iniciar el driver del Wi-Fi. Y posteriormente a *esp_wifi_connect()* para conectarnos al punto de acceso que previamente hemos configurado.
7) Por último, para dar por finalizada una conexión Wi-Fi, deberemos llamar a las siguientes funciones *esp_wifi_disconnect()* -> *esp_wifi_stop()* -> *esp_wifi_deinit()* para desconectar la Wi-Fi, detener el *driver* y borrar su configuración.

### 3.4.3. Código fuente

El código se encuentra implementado sobre un único fichero (*main.c*), el cual vamos a explicar dividiéndolo en funciones.

Antes de ello, es importante saber cómo funciona cada uno de los aspectos que engloba la aplicación, como es el caso del servidor TCP. Se ha elegido que haga las funciones del servidor el módulo ESP32, dado que es la parte que debe recibir la configuración, ya que desde el servidor no podemos mandar la configuración manualmente.

En la Figura 15 observaremos la estructura de una conexión TCP modelo servidor-cliente, indicando las funciones de cada uno de los extremos de la conexión.

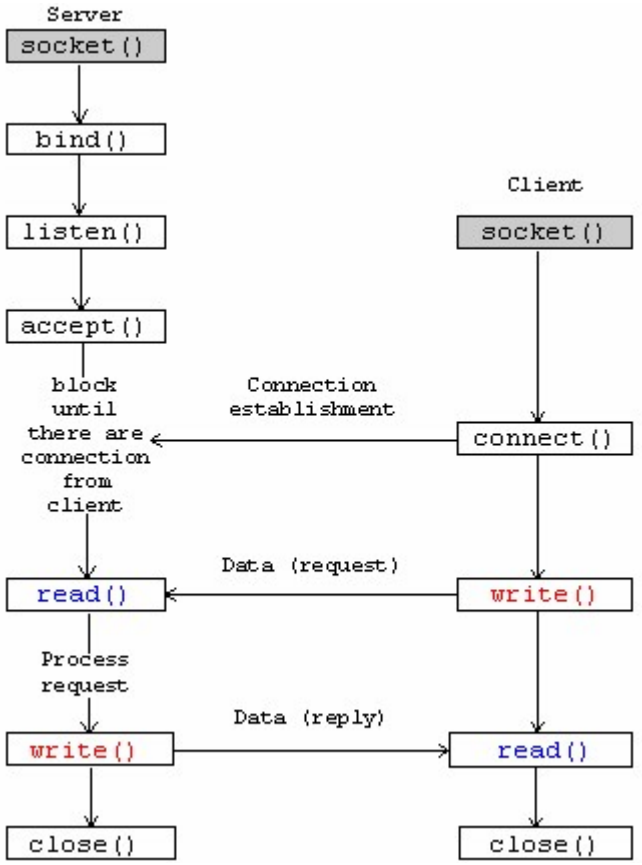


*Figura 15. Diagrama de bloques TCP modelo Servidor-Cliente [13].*

Una vez establecida la conexión, el servidor esperará a que el cliente le mande un mensaje, es decir, una trama TCP. Posteriormente el servidor comprobará que no hay errores en la trama y pasará a decodificar la información que posee. Dado que la comunicación es TCP, el servidor mandará al cliente un acuse de recibo (ACK) y, una vez recibido, reenviará los datos recibidos hacia el cliente para que podamos comprobar que la trama mandada en un momento inicial es la correcta.

Las funciones que implementan la comunicación son las que podemos observar en la Figura 15 y cuyo funcionamiento es el siguiente:

- *socket()*: Crea un nuevo *socket* para establecer la comunicación.
- *bind()*: Asigna una dirección IP al *socket*.
- *listen()*: Avisa de que el servidor está dispuesto a aceptar conexiones.

- *accept()*: Bloquea al autor de la llamada a la función hasta recibir una petición de conexión.
- *connect()*: Intento de establecer la conexión TCP.
- *read()*: Recibe datos a través de la conexión.
- *send()*: Manda datos a través de la conexión.
- *close()*: Libera la conexión entre servidor y cliente.

En la Figura 16 se muestra la composición de la trama TCP de comunicación entre el servidor y el usuario. Está compuesta por 7 campos de configuración separados por comas como medida de seguridad en la decodificación y el retorno de carro para indicar el fin de la trama.

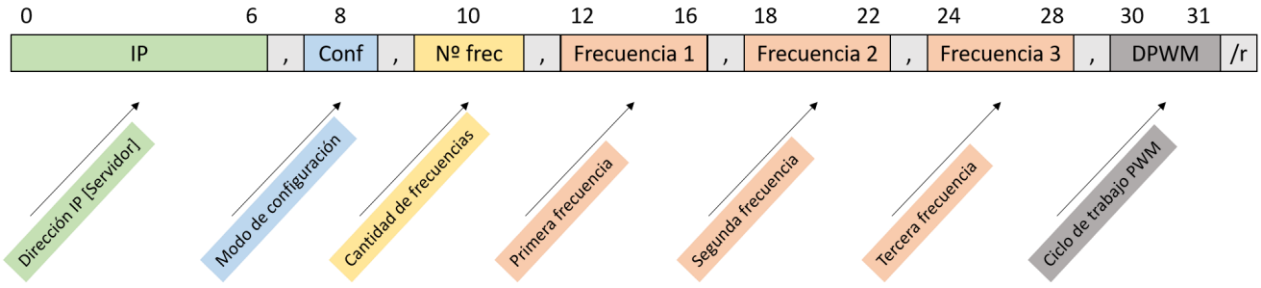


*Figura 16. Trama TCP Cliente – Servidor.*

- IP [0 - 6]: Campo que corresponde con la dirección IP de cada servidor, es decir, de cada módulo ESP32 existente. Dado que solamente la necesitaremos para corroborar que el cliente que se conecte pertenece a la red, mandaremos los dos últimos campos de la dirección del cliente, es decir, el identificativo de red y el de host/usuario.
- Modo de configuración [8]: Bit para configurar el modo de funcionamiento de la siguiente manera:
    - 0 → Modo PWM a la salida.
    - 1 → Modo senoidal a la salida.
- Cantidad de frecuencias [10]: En este campo podremos configurar el número de frecuencias con las que queremos modelar nuestro sistema. Inicialmente partiremos con dos, pero la estructura será genérica para poder ampliar en un futuro.
- Frecuencia 1 [12 - 16]: Frecuencia con la que modularemos nuestro sistema.
- Frecuencia 2 [18 - 22]: Segunda frecuencia con la que modularemos nuestro sistema.

- Frecuencia 3 [24 - 28]: Frecuencia auxiliar, la cual se leerá para una posible ampliación en un futuro (no trabajaremos actualmente con ella).
- Ciclo de trabajo PWM [30 - 31]: Cuando nos encontremos en modo PWM (ver modo de configuración), podremos configurar en este campo, el % del ciclo de trabajo que deseemos para la señal PWM.

Dado que la estructura TCP que el cliente mandará posee una estructura fija (como hemos comentado anteriormente), debemos comentar una serie de aspectos a tener presente a la hora de mandar la trama:

1) Debemos rellenar siempre todos los bits de cada campo puesto que, si no lo hacemos, la decodificación sería errónea.
2) En la dirección IP, cuando tengamos un caso en el cual la dirección del host/ordenador sea de dos dígitos (por ejemplo 192.168.1.77), debemos rellenar el último bit con un carácter o un espacio. De esta manera, podremos mantener el formato de la trama y realizar la decodificación correctamente.
3) Solo tenemos dos modos de configuración y, por tanto, todo lo que no sea un nivel bajo o nivel alto en el campo correspondiente, será interpretado como nulo y se pedirá introducir de nuevo la trama completa.
4) Al igual que en el apartado anterior, tendremos como mínimo una frecuencia y como máximo tres. En conclusión, cualquier valor que sea distinto, será interpretado como nulo y se nos pedirá volver a introducir la trama completa.
5) Cada una de las frecuencias se leerá en Hz, por tanto, se debe introducir la frecuencia en dicha medida (recordamos que hay que rellenar todos los bits, por ejemplo, si queremos introducir 10 Hz, el campo se debe configurar como “00010”).

   Cuando nos encontremos en el modo senoidal, solo haremos uso de una frecuencia para la señal (que se generará mediante PWM con su posterior filtrado) que será la Frecuencia 1.
6) El ciclo de trabajo (cuando nos encontramos en PWM) no debe exceder del 99% ni encontrarse por debajo del 1%; esto es debido a que, en tal caso, no tendríamos señal a la salida.

Pasamos a continuación a detallar las diferentes funciones implementadas en el código fuente de la aplicación.

#### *3.4.3.1. configTimer0Group0*

La función *configTimer0Group0* se encarga de configurar el *timer* 0 del grupo 0 para que interrumpa cada ms y podamos cambiar la señal PWM saliente por el puerto GPIO26.

```
/******************************************************************************
* Function Name  : configTimer0Group0
* Description    : Función de configuración del TIMER0GROUP0 para cambiar frecuencia
cada ms
* Input          : None
* Output         : None
* Return         : None
* Attention                 : None
*******************************************************************************/
void configTimer0Group0(void){

        timer_config_t timer0Config = {
                        .alarm_en = TIMER_ALARM_EN, /*Alarma habilitada*/
                        .counter_en = TIMER_PAUSE, /*Cuenta deshabilitada*/
                        .counter_dir = TIMER_COUNT_UP, /*Contador ascendente*/
                        .intr_type = TIMER_INTR_LEVEL, /*Interrupción por nivel*/
                        .auto_reload = TIMER_AUTORELOAD_EN, /*Recarga habilitada*/
                        .divider = 80, /*Prescaler 80, contador cuenta cada 1 us*/
        };

        timer_init(TIMER_GROUP_0, TIMER_0, &timer0Config); /*Inicializamos el TIMER0
del GROUP0*/
        timer_set_counter_value(TIMER_GROUP_0,        TIMER_0,        0x00000000ULL);
/*Inicializamos el contador del TIMER0 para que arranque en 0*/
        timer_set_alarm_value(TIMER_GROUP_0,                                TIMER_0,
Fpclk/timer0Config.divider/1e3); /*Alarma cada ms*/
        timer_enable_intr(TIMER_GROUP_0, TIMER_0); /*Habilitamos la interrupción*/
        timer_isr_register(TIMER_GROUP_0, TIMER_0, timer0_group0_isr,
                        (void *) NULL, 0, NULL);
        timer_start(TIMER_GROUP_0, TIMER_0); /*Arrancamos TIMER0 GROUP0*/
}
```

*Código 1. Función configTimer0Group0.*

Como podemos observar, para configurar el *timer 0* el fabricante pone a nuestra disposición una serie de funciones que se encargan de tal propósito, aunque en primer lugar debemos crear una estructura del tipo *timer_config_t* que posteriormente usaremos en las funciones.

La estructura posee 6 campos, como podemos observar en el código. Configuran si existirá alarma, si queremos que el *timer* comience encendido o apagado, la dirección de la cuenta del mismo, si queremos que interrumpa una vez alcanzado el número que posteriormente configuraremos, si queremos que reinicie una vez alcance dicho valor y, por último, si haremos uso del *prescaler* indicando el valor deseado.

- *timer_init()*: se encarga de inicializar el *timer*, pasándole como argumentos el grupo al que pertenece, el *timer* que queremos usar y la estructura previamente definida.

- *timer_set_counter_value()*: configura el valor desde el cual queremos arrancar el *timer*, al igual que en la anterior función debemos indicar el grupo al que pertenece, así como el *timer* concreto y el valor inicial. Recordamos que los registros de los *timer* del microprocesador son de 64 bits, por tanto, debemos pasarle un valor de dicho tamaño.

- *timer_set_alarm_value()*: configura el valor de interrupción del *timer*, expresado en número de cuentas. Le pasamos el grupo al que pertenece, el *timer* concreto y el número de cuentas.

- *timer_enable_intr()*: habilita la interrupción si en la estructura hemos configurado tal propósito, solo debemos indicarle el grupo y el *timer*.

- *timer_isr_register()*: registra la interrupción asociada al *timer*, debemos indicarle cómo se llamará la rutina de atención a la interrupción (ISR), así como si tiene algún parámetro de entrada, el grupo y el *timer* correspondiente.
- *timer_start()*: arranca el *timer*, indicaremos el grupo y el *timer* que queremos arrancar.

#### *3.4.3.2. configTimer1Group0*

El segundo *timer* lo utilizaremos para ir sacando las muestras de la señal senoidal (cuando este tipo de señal se requiera), siempre y cuando nos encontremos en dicho modo de configuración. En este caso, haremos uso del *timer* 1 del grupo 0.

```
/******************************************************************************
* Function Name  : configTimer1Group0
* Description    : Función de configuración del TIMER1GROUP0 para sacar muestras del
seno
* Input          : None
* Output         : None
* Return         : None
* Attention              : None
******************************************************************************/
void configTimer1Group0(void){

        timer_config_t timer1Config = {
                        .alarm_en = TIMER_ALARM_EN, /*Alarma habilitada*/
                        .counter_en = TIMER_PAUSE, /*Cuenta deshabilitada*/
                        .counter_dir = TIMER_COUNT_UP, /*Contador ascendente*/
                        .intr_type = TIMER_INTR_LEVEL, /*Interrupción por nivel*/
                        .auto_reload = TIMER_AUTORELOAD_EN, /*Recarga habilitada*/
                        .divider = 80, /*Prescaler 80, contador cuenta cada 1 us*/
        };

        timer_init(TIMER_GROUP_0, TIMER_1, &timer1Config); /*Inicializamos el TIMER1 del
GROUP0*/
        timer_set_counter_value(TIMER_GROUP_0, TIMER_1, 0x00000000ULL); /*Inicializamos
el contador del TIMER1 para que arranque en 0*/
        timer_set_alarm_value(TIMER_GROUP_0, TIMER_1, 40); /*Alarma cada 40 us (25KHz)*/
        timer_enable_intr(TIMER_GROUP_0, TIMER_1); /*Habilitamos la interrupción*/
                timer_isr_register(TIMER_GROUP_0, TIMER_1, timer1_group0_isr,
                                (void *) NULL, 0, NULL);
                timer_start(TIMER_GROUP_0, TIMER_1); /*Arrancamos TIMER1 GROUP0*/
}
```

*Código 2. Función configTimer1Group0.*

La configuración se realiza de la misma manera que en *configTimer0Group0*. Solamente debemos variar el valor en que queremos interrumpir y registrar una nueva rutina de atención a la interrupción.

#### *3.4.3.3. wifi_init_sta*

Para configurar el módulo Wi-Fi del ESP32 en modo STA deberemos seguir los pasos detallados en la sección 3.4.2. El fabricante, en unos de los ejemplos que pone a nuestra disposición, implementa una función que se encarga de todo el proceso. Inicialmente la función nos indicará si se ha completado el proceso de configuración y una vez se ejecute el *handler* (lo veremos a continuación), nos indicará si se ha conectado y a qué red.

```
void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());

    esp_netif_t *sta_netif = esp_netif_create_default_wifi_sta();
    assert(sta_netif);

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        sta_netif,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        sta_netif,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
                .sta = {
            .ssid = EXAMPLE_WIFI_SSID,
            .password = EXAMPLE_WIFI_PASS,
            /* Setting a password implies station will connect to all security modes
including WEP/WPA.
             * However these modes are deprecated and not advisable to be used. Incase
your Access point
             * doesn't support WPA2, these mode can be enabled by commenting below line
*/
          .threshold.authmode = WIFI_AUTH_WPA2_PSK,
            .pmf_cfg = {
            .capable = true,
            .required = false
                      },
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK(esp_wifi_start() );

    ESP_LOGI(TAG, "wifi_init_sta finished.");

    /* Waiting until either the connection is established (WIFI_CONNECTED_BIT) or
connection failed for the maximum
     * number of re-tries (WIFI_FAIL_BIT). The bits are set by event_handler() (see
above) */
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
            pdFALSE,
            pdFALSE,
            portMAX_DELAY);

    /* xEventGroupWaitBits() returns the bits before the call returned, hence we can
test which event actually
     * happened. */
    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
                 EXAMPLE_WIFI_SSID, EXAMPLE_WIFI_PASS);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
                 EXAMPLE_WIFI_SSID, EXAMPLE_WIFI_PASS);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }

    /* The event will not be processed after unregister */
    ESP_ERROR_CHECK(esp_event_handler_instance_unregister(IP_EVENT,
IP_EVENT_STA_GOT_IP, instance_got_ip));
```

```
    ESP_ERROR_CHECK(esp_event_handler_instance_unregister(WIFI_EVENT,
ESP_EVENT_ANY_ID, instance_any_id));
    vEventGroupDelete(s_wifi_event_group);
}
```

*Código 3. Función wifi_init_sta.*

#### 3.4.3.4. *event_handler*

Cuanto ocurre un evento se ejecutará un *handler* (o manejador). En nuestro caso se ejecutará una vez haya terminado la configuración e iniciará el proceso de conectarse a la red Wi-Fi que hayamos configurado. La función nos indicará por el terminal serie si se ha podido establecer la conexión o si ha fallado el proceso.

```
static void event_handler(void* arg, esp_event_base_t event_base,
                                int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_CONNECTED) {
        example_set_static_ip(arg);
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < EXAMPLE_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "static ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}
```

*Código 4. Función evento_handler.*

#### 3.4.3.5. *example_set_static_ip*

Para finalizar con lo que respecta a Wi-Fi, tenemos la función que establece una IP fija al módulo ESP32. La función deshabilita DHCP (protocolo de configuración dinámica de *host*), ya que no queremos que la dirección IP varíe y nos devuelve si se ha realizado con éxito junto con la IP configurada, la máscara de red y el *gateway*.

```
static void example_set_static_ip(esp_netif_t *netif)
{
    if (esp_netif_dhcpc_stop(netif) != ESP_OK) {
        ESP_LOGE(TAG, "Failed to stop dhcp client");
        return;
    }
    esp_netif_ip_info_t ip;
    memset(&ip, 0 , sizeof(esp_netif_ip_info_t));
    ip.ip.addr = ipaddr_addr(EXAMPLE_STATIC_IP_ADDR);
    ip.netmask.addr = ipaddr_addr(EXAMPLE_STATIC_NETMASK_ADDR);
    ip.gw.addr = ipaddr_addr(EXAMPLE_STATIC_GW_ADDR);
    if (esp_netif_set_ip_info(netif, &ip) != ESP_OK) {
        ESP_LOGE(TAG, "Failed to set ip info");
        return;
    }
    ESP_LOGD(TAG, "Success to set static ip: %s, netmask: %s, gw: %s",
EXAMPLE_STATIC_IP_ADDR, EXAMPLE_STATIC_NETMASK_ADDR, EXAMPLE_STATIC_GW_ADDR);
}
```

*Código 5. Función example_set_static_ip.*

#### 3.4.3.6. *tcp_server_task*

Anteriormente hemos detallado como funciona un servidor TCP, así como su diagrama de bloques en la Figura 15. La tarea *tcp_server_task* se encarga de implementar la funcionalidad de servidor hasta el punto de establecer la conexión entre ambas partes. Una vez se establezca la conexión, pasaremos a leer datos que nos mande el cliente.

```
static void tcp_server_task(void *pvParameters)
{
    char addr_str[128];
    int addr_family = (int)pvParameters;
    int ip_protocol = 0;
    struct sockaddr_in6 dest_addr;

    if (addr_family == AF_INET) {
        struct sockaddr_in *dest_addr_ip4 = (struct sockaddr_in *)&dest_addr;
        dest_addr_ip4->sin_addr.s_addr = inet_addr(EXAMPLE_STATIC_IP_ADDR);
        dest_addr_ip4->sin_family = AF_INET;
        dest_addr_ip4->sin_port = htons(PORT);
        ip_protocol = IPPROTO_IP;
    } else if (addr_family == AF_INET6) {
        bzero(&dest_addr.sin6_addr.un, sizeof(dest_addr.sin6_addr.un));
        dest_addr.sin6_family = AF_INET6;
        dest_addr.sin6_port = htons(PORT);
        ip_protocol = IPPROTO_IPV6;
    }

    int listen_sock = socket(addr_family, SOCK_STREAM, ip_protocol);
    if (listen_sock < 0) {
        ESP_LOGE(TAG, "Unable to create socket: errno %d", errno);
        vTaskDelete(NULL);
        return;
    }

    ESP_LOGI(TAG, "Socket created");

    int err = bind(listen_sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    if (err != 0) {
        ESP_LOGE(TAG, "Socket unable to bind: errno %d", errno);
        ESP_LOGE(TAG, "IPPROTO: %d", addr_family);
        goto CLEAN_UP;
    }
    ESP_LOGI(TAG, "Socket bound, port %d", PORT);

    err = listen(listen_sock, 1);
    if (err != 0) {
        ESP_LOGE(TAG, "Error occurred during listen: errno %d", errno);
        goto CLEAN_UP;
    }

    while (1) {

        ESP_LOGI(TAG, "Socket listening");

        struct sockaddr_in6 source_addr; // Large enough for both IPv4 or IPv6
        uint addr_len = sizeof(source_addr);
        int sock = accept(listen_sock, (struct sockaddr *)&source_addr, &addr_len);
        if (sock < 0) {
            ESP_LOGE(TAG, "Unable to accept connection: errno %d", errno);
            break;
        }

        // Convert ip address to string
        if (source_addr.sin6_family == PF_INET) {
            inet_ntoa_r(((struct     sockaddr_in     *)&source_addr)->sin_addr.s_addr,
addr_str, sizeof(addr_str) - 1);
        } else if (source_addr.sin6_family == PF_INET6) {
            inet6_ntoa_r(source_addr.sin6_addr, addr_str, sizeof(addr_str) - 1);
        }
        ESP_LOGI(TAG, "Socket accepted ip address: %s", addr_str);

        do_retransmit(sock);
```

```
        shutdown(sock, 0);
        close(sock);
    }

CLEAN_UP:
    close(listen_sock);
    vTaskDelete(NULL);
}
```

*Código 6. Función tcp_server_task.*

#### 3.4.3.7. *do_retransmit*

Como hemos comentado, la función anterior implementa la funcionalidad de servidor hasta establecer la conexión. Una vez establecida llamaremos a la función *do_retransmit*, cuya funcionalidad es la de recibir los datos enviados por el cliente (Figura 16) y reenviarlos, para que se pueda corroborar desde el cliente que la trama es correcta.

```
static void do_retransmit(const int sock)
{
    int len;
    char rx_buffer[128];

    do {
        len = recv(sock, rx_buffer, sizeof(rx_buffer) - 1, 0);
        if (len < 0) {
            ESP_LOGE(TAG, "Error occurred during receiving: errno %d", errno);
        } else if (len == 0) {
            ESP_LOGW(TAG, "Connection closed");
        } else {
            rx_buffer[len] = 0; // Null-terminate whatever is received and treat it
like a string
            ESP_LOGI(TAG, "Received %d bytes: %s", len, rx_buffer);
            if (len != 33)
               ESP_LOGE(TAG, "Datos erróneos, por favor vuelva a introducir");
                       else
               decodifica(rx_buffer);

            // send() can return less bytes than supplied length.
            // Walk-around for robust implementation.
            int to_write = len;
            while (to_write > 0) {
                int written = send(sock, rx_buffer + (len - to_write), to_write, 0);
                if (written < 0) {
                    ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
                }
                to_write -= written;
            }
        }
    } while (len > 0);
}
```

*Código 7. Función do_retransmit.*

#### 3.4.3.8. *decodifica*

Desde la función *do_retransmit* llamaremos a esta función para decodificar la trama TCP que hemos recibido. La función se encarga de recorrer el *buffer* de recepción buscando los separadores (“,”) y cuando encuentra uno, almacena el dato que se encuentra entre el nuevo separador encontrado y el anterior. Como la trama es de longitud fija (Figura 16), las posiciones de lectura de las variables serán siempre las mismas.

```
/*****************************************************************************
* Function Name  : decodifica
* Description    : Función de decodificación de la trama TCP
* Input          : rx_buffer
* Output         : None
* Return         : None
* Attention              : None
*****************************************************************************/
void decodifica(char rx_buffer[128])
{
  for (int i=0; i<33; i++){
    if (rx_buffer[i] == ','){
      if (i == 7){
        for (int j=0; j<7; j++){
          ip[j]= rx_buffer[j];
        }
        ESP_LOGI(TAG, "Ip del cliente: 192.168.%s", ip);
      }
      else if(i == 9){
        for (int j=0; j<1; j++){
          senoPWM[j]= rx_buffer[8+j];
        }
        if (atoi(senoPWM) != 0 && atoi(senoPWM) != 1) /*Si no hemos metido valor
correcto*/
          ESP_LOGE(TAG, "Modo configuración erróneo, por favor vuelva a introducir");
        else /*Resto*/
          ESP_LOGI(TAG, "%s", (atoi(senoPWM) == 0) ? "Modo PWM" : "Modo Seno");
      }
      else if(i == 11){
        for (int j=0; j<1; j++){
          numFrec[j]= rx_buffer[10+j];
        }
        if (atoi(senoPWM) == 0){
          if (atoi(numFrec) != 1 && atoi(numFrec) != 2 && atoi(numFrec) != 3) /*Si no
hemos metido ningún número correcto de frecuencias*/
            ESP_LOGE(TAG, "Número de frecuencias erróneos, por favor vuelva a
introducir");
          else /*Resto*/
            ESP_LOGI(TAG, "Número de frecuencias: %d", atoi(numFrec));
        }
        else if(atoi(senoPWM) == 1)
          ESP_LOGI(TAG, "Número de frecuencias = 1");
      }
      else if(i == 17){
        for (int j=0; j<5; j++){
          frec1[j]= rx_buffer[12+j];
        }
        if (atoi(senoPWM) == 0){ /*Si es modo PWM*/
          if(atoi(numFrec) != 1 && atoi(numFrec) != 2 && atoi(numFrec) != 3) /*Si no
hemos metido ningún número correcto de frecuencias*/
            ESP_LOGW(TAG, "Frecuencia 1: 0 Hz");
          else{
            ESP_LOGI(TAG, "Frecuencia 1: %d Hz", atoi(frec1));
          }
        }
        else if (atoi(senoPWM) == 1){ /*Si es modo seno*/
          ESP_LOGI(TAG, "Frecuencia Seno: %d Hz", atoi(frec1));
        }
      }
      else if(i == 23){
        for (int j=0; j<5; j++){
          frec2[j]= rx_buffer[18+j];
        }
        if(atoi(senoPWM) == 0){ /*Si es modo PWM*/
          if (atoi(numFrec) != 1 && atoi(numFrec) != 2 && atoi(numFrec) != 3) /*Si no
hemos metido ningún número correcto de frecuencias*/
            ESP_LOGW(TAG, "Frecuencia 2: 0 Hz");
          else if(atoi(numFrec) == 2 || atoi(numFrec) == 3){ /*Si hay dos o tres
frecuencias, actualizamos*/
            ESP_LOGI(TAG, "Frecuencia 2: %d Hz", atoi(frec2));          }
        }
      }
      else if(i == 29){
        for (int j=0; j<5; j++){
          frec3[j]= rx_buffer[24+j];
        }
        if(atoi(senoPWM) == 0){ /*Si es modo PWM*/
```

```
          if (atoi(numFrec) != 1 && atoi(numFrec) != 2 && atoi(numFrec) != 3) /*Si no
hemos metido ningún número correcto de frecuencias*/
            ESP_LOGW(TAG, "Frecuencia 3: 0 Hz");
          else if(atoi(numFrec) == 3) /*Si hay 3 frecuencias, mostramos*/
            ESP_LOGI(TAG, "Frecuencia 3: %d Hz", atoi(frec3));
        }
      }
    }
    else if (rx_buffer[i] == '\r'){
      for (int j=0; j<2; j++){
        DPWM[j]= rx_buffer[30+j];
      }
      if(atoi(senoPWM) == 0){
        if(atoi(DPWM) < 1 || atoi(DPWM) > 99) /*Si el ciclo de trabajo no es válido*/
          ESP_LOGE(TAG, "Ciclo de trabajo erróneo, por favor vuelva a introducir");
        else{
          ESP LOGI(TAG, "Ciclo de trabajo PWM: %d", atoi(DPWM));
        }
      }
    }
  }
  wifi = 1; /*Flag de decodificación completada*/
}
```

*Código 8. Función decodifica.*

### 3.4.3.9. *timer0_group0_isr*

Esta función implementa la rutina de atención a la interrupción del *timer* 0 del grupo 0. Como explicamos en la configuración (sección 3.4.3.1), el *timer* 0 se encargará de conmutar entre las PWM generadas de diferente frecuencia y encaminarla a la de salida por el GPIO26.

Aunque su funcionamiento no se reduce solamente a tal propósito, también se encargará de actualizar la configuración de las señales recibidas por Wi-Fi. Esto es debido a que si las actualizamos en la interrupción siguiente a la recepción por Wi-Fi, favorecemos la sincronización entre señales.

```
/*****************************************************************************
* Function Name  : timer0_group0_isr
* Description    : Función de atención a la interrupción del TIMER0GROUP0
* Input          : arg - NULL
* Output         : None
* Return         : None
* Attention      : None
*****************************************************************************/
void timer0_group0_isr(void *arg){

    timer_group_clr_intr_status_in_isr(TIMER_GROUP_0, TIMER_0); /*Borramos flag de
interrupción*/

    if (wifi == 1){ /*Si hemos decodificado la trama, actualizamos*/
      if (atoi(senoPWM) == 0){ /*Si es modo PWM*/
        if(atoi(numFrec) == 1 || atoi(numFrec) == 2 || atoi(numFrec) == 3){ /*Si hemos
metido un número correcto de frecuencias*/
          frecuencia1 = atoi(frec1);
          pwmAConfig.freq hz = frecuencia1;
          ledc_timer_config(&pwmAConfig); /*Configuramos TIMER0 para PWM-A*/
        }
        if(atoi(numFrec) == 2 || atoi(numFrec) == 3){ /*Si hay dos o tres frecuencias,
actualizamos*/
          frecuencia2 = atoi(frec2);
          pwmBConfig.freq_hz = frecuencia2;
          ledc_timer_config(&pwmBConfig); /*Configuramos TIMER1 para PWM-B*/
        }
        if(atoi(DPWM) > 1 && atoi(DPWM) < 99){ /*Si el ciclo de trabajo es válido*/
          pwmACanal0.duty = atoi(DPWM)*pow(2, 7)/100;
          pwmBCanal1.duty = atoi(DPWM)*pow(2, 7)/100;
```

```
          pwmInversaCanal2_GPIO25.duty = pow(2, 7) - atoi(DPWM)*pow(2, 7)/100;
          pwmInversaCanal2_GPIO25.hpoint = atoi(DPWM)*pow(2, 7)/100;
          ledc_channel_config(&pwmACanal0); /*Configuramos canal 0 para PWM-A*/
          ledc_channel_config(&pwmBCanal1); /*Configuramos canal 1 para PWM-B*/
          ledc_channel_config(&pwmInversaCanal2_GPIO25); /*Configuramos canal 2 para
PWM-Inversa*/
        }
      }
      else if (atoi(senoPWM) == 1){ /*Si es modo seno*/
        frecuenciaSeno = atoi(frec1);
        frecuenciaPWM = Nmuestras*frecuenciaSeno;
        timer_set_counter_value(TIMER_GROUP_0,        TIMER_1,        0x00000000ULL);
/*Inicializamos el contador del TIMER1 para que arranque en 0*/
        timer_set_alarm_value(TIMER_GROUP_0,                                    TIMER_1,
10e7/(frecuenciaSeno*Nmuestras)); /*Alarma cada 1/frecuenciaSeno*/
        pwmConfigSeno.freq_hz = frecuenciaPWM;
        ledc timer config(&pwmConfigSeno); /*Configuramos TIMER1 para PWM*/
      }
      wifi = 0;
    }
      /*Cada ms*/
      if(atoi(senoPWM) == 0){ /*Si estamos en modo PWM*/
          bit = bit^1; /*Alternamos valor*/
          if(bit){
            pwmACanal0.gpio_num = 27; /*Asignamos señal de frecuencia F1 a GPIO 27*/
            pwmBCanal1.gpio_num = 26; /*Asignamos señal de frecuencia F2 a GPIO 26*/
            pwmInversaCanal2_GPIO25.timer_sel = LEDC_TIMER_1; /*Cambiamos el timer de
la inversa, para que muestre la inversa de PWM-B*/
            ledc channel config(&pwmACanal0); /*Actualizamos cambios en PWM-a*/
            ledc_channel_config(&pwmInversaCanal2_GPIO25); /*Configuramos TIMER1 para
PWM-Inversa*/
            ledc channel config(&pwmBCanal1); /*Actualizamos cambios en PWM-B*/
          }
          else{
            pwmACanal0.gpio num = 26; /*Asignamos señal de frecuencia F1 a GPIO 26*/
            pwmBCanal1.gpio_num = 27; /*Asignamos señal de frecuencia F2 a GPIO 27*/
            pwmInversaCanal2_GPIO25.timer_sel = LEDC_TIMER_0; /*Cambiamos el timer de
la inversa, para que muestre la inversa de PWM-A*/
            ledc_channel_config(&pwmACanal0); /*Actualizamos cambios en PWM-A*/
            ledc_channel_config(&pwmInversaCanal2_GPIO25); /*Configuramos TIMER0 para
PWM-Inversa*/
            ledc_channel_config(&pwmBCanal1); /*Actualizamos cambios en PWM-B*/
          }
      }

    timer_group_enable_alarm_in_isr(TIMER_GROUP_0,  TIMER_0);  /*Volvemos  a  habilitar
alarma, se desactiva al interrumpir*/
}
```

*Código 9. Función timer0_group0_isr.*

En la parte de conmutación del puerto lo que debemos hacer es reasignar el puerto de cada una de las PWM con frecuencias distintas. También deberemos cambiar el valor del *timer* interno asociado a la PWM invertida, para que cuando conmutemos el puerto cambie también la señal invertida correspondiente.

Una vez realizados los cambios en las estructuras correspondientes, deberemos actualizar los valores llamando a las funciones *ledc_channel_config()* y *ledc_timer_config()*, cuya misión es actualizar los valores de las estructuras de tipo *ledc_channel_config_t* y *ledc_timer_config_t* respectivamente.

También debemos borrar el *flag* de interrupción del *timer* y volver a activar la alarma, puesto que cuando se produce una interrupción se desactiva por defecto. Para ello usaremos las funciones *timer_group_clr_intr_status_in_isr()* para borrar el *flag* y *timer_group_enable_alarm_in_isr()* para reactivar la alarma. Ambas tienen como parámetros el grupo y el *timer* concreto.

#### 3.4.3.10. *timer1_group0_isr*

Implementa la rutina de atención a la interrupción del *timer* 1 del grupo 0. Este segundo *timer*, tal y como explicamos cuando se configuró (sección 3.4.3.2) se encargará de sacar las muestras del seno cuando esa sea la salida configurada.

```
/******************************************************************************
* Function Name  : timer1_group0_isr
* Description    : Función de atención a la interrupción del TIMER1GROUP0
* Input          : arg - NULL
* Output         : None
* Return         : None
* Attention      : None
******************************************************************************/
void timer1_group0_isr(void *arg){

    timer_group_clr_intr_status_in_isr(TIMER_GROUP_0,  TIMER_1);  /*Borramos  flag  de
interrupción*/

    if(atoi(senoPWM) == 1){ /*Si estamos en modo seno*/
      pwmSenoCanal3_GPIO26.duty = tablaD[cuenta++]; /*Nuevo ciclo de trabajo*/
      ledc_channel_config(&pwmSenoCanal3_GPIO26); /*Configuramos canal 3 para PWM*/
        if (cuenta >= Nmuestras) /*Si hemos llegado a la última muestra reiniciamos*/
          cuenta = 0;
    }

    timer_group_enable_alarm_in_isr(TIMER_GROUP_0,  TIMER_1);  /*Volvemos  a  habilitar
alarma, se desactiva al interrumpir*/
}
```

*Código 10. Función timer1_group0_isr.*

#### 3.4.3.11. *app_main*

Es la función principal de nuestra aplicación. Se encargará de la configuración inicial de las señales PWM y de los *timer*, así como de llamar a la función de configuración del módulo Wi-Fi. Por último, crearemos la tarea del servidor TCP que simulará nuestro ESP32.

```
void app_main(void){

  if (init == 0){
    configTimer0Group0(); /*Configuramos TIMER0*/
    configTimer1Group0(); /*Configuramos TIMER1*/
    ledc_timer_config(&pwmAConfig); /*Configuramos TIMER0 para PWM*/
    ledc_timer_config(&pwmBConfig); /*Configuramos TIMER1 para PWM*/
    ledc_channel_config(&pwmInversaCanal2_GPIO25); /*Configuramos canal 2 para inversa
PWM*/
    ledc_channel_config(&pwmACanal0); /*Configuramos canal 0 para PWM-A*/
    ledc_channel_config(&pwmBCanal1); /*Configuramos canal 0 para PWM-B*/
    ledc_timer_config(&pwmConfigSeno); /*Configuramos TIMER2 para PWM*/
    init = 1;
  }

  //Initialize NVS
  esp_err_t ret = nvs_flash_init();
  if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    ret = nvs_flash_init();
  }
  ESP_ERROR_CHECK(ret);
  ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
  wifi_init_sta();
  xTaskCreate(tcp_server_task, "tcp_server", 4096, (void*)AF_INET, 5, NULL);
}
```

*Código 11. Función app_main.*

## 3.5. Resultados teóricos

Vamos a explicar teóricamente los resultados que se esperan de la aplicación para poder conocer realmente qué se está configurando y realizando con todo el código desarrollado. Como comentamos en la descripción, el objetivo de la aplicación es la de generar las señales PWM (bien para generar PWM puras o para emular un seno) que excitarán al driver de la luminaria y será capaz de recibir por Wi-Fi la configuración de dichas señales y actualizarlas. En la Figura 17 se muestra la señal que excitará al driver de los LED compuesta por dos señales cuadradas consecutivas con un intervalo de separación de 1 ms.

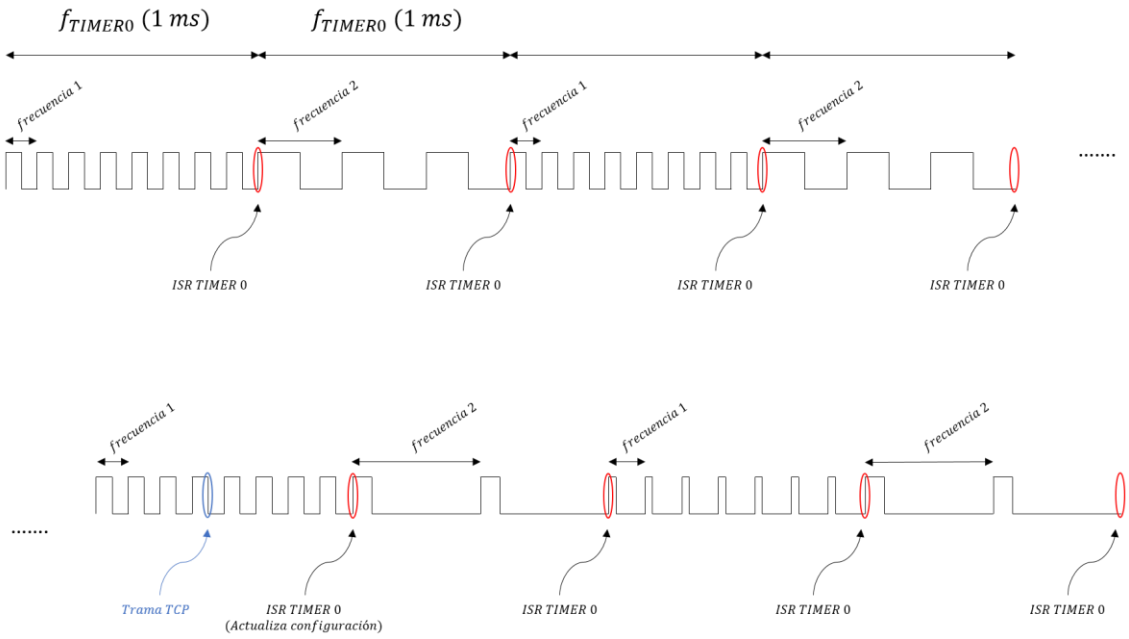


*Figura 17. Resultados teóricos modo PWM – GPIO26.*

En la Figura 17 podemos observar cómo sería el funcionamiento del sistema en modo PWM (con 2 señales emitidas secuenciales, de diferente frecuencia) y cómo será la salida por los pines del puerto GPIO 25 y 26. Una señal será la invertida de la otra.

En cambio, si nos encontramos en el modo de funcionamiento senoidal, la señal PWM que deberemos generar será diferente. Y con un filtro paso bajo posteriormente se obtendrá la senoidal asociada. En la Figura 18 podemos observar la salida en este otro modo de funcionamiento.

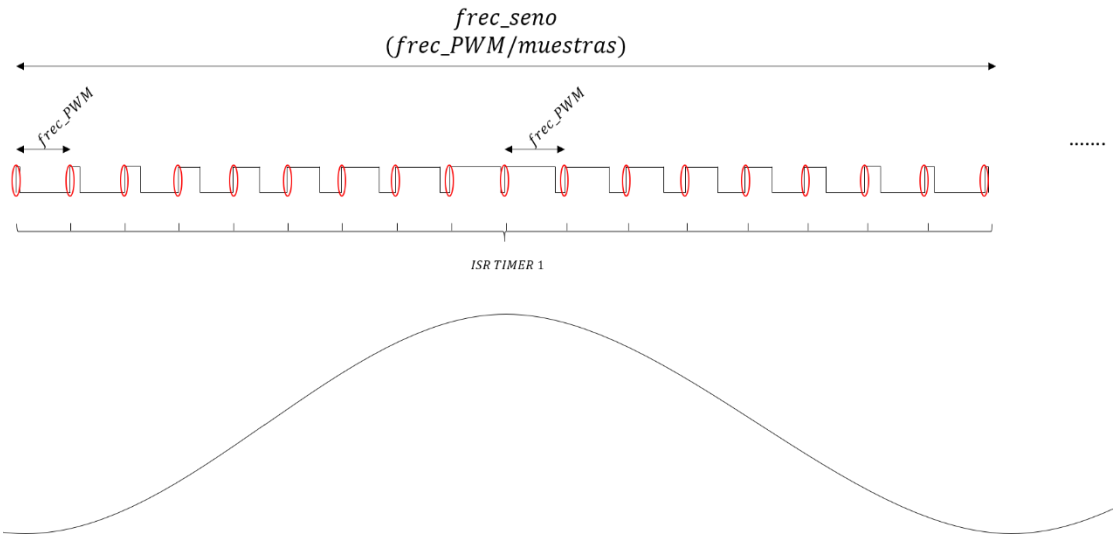


*Figura 18. Resultados teóricos modo seno – GPIO26 y seno.*

## 3.6. Resultados prácticos

Ahora contrastaremos los resultados teóricos con los que se han obtenido en el laboratorio una vez finalizada la aplicación. En las siguientes figuras mostraremos distintas señales de salida variando frecuencia y ciclo de trabajo a través del Wi-Fi gracias a la trama TCP de la Figura 16.

En la Figura 19 podemos observar la señal inicial de salida de la aplicación, por defecto obtenemos una señal de un ciclo de trabajo del 50% y dos frecuencias de 3 y 6 KHz. También vemos la señal invertida.


*Figura 19. Resultados prácticos modo PWM – GPIO26 y GPIO25 (3KHz, 6KHz, 50%).*

Haciendo uso de la conexión Wi-Fi hemos variado las frecuencias y el ciclo de trabajo; en la Figura 20 tenemos una señal de salida del microcontrolador de un ciclo de trabajo del 70% y frecuencias de 2 y 8 KHz. Es de vital importancia comprobar las conmutaciones de la señal original y la invertida dado que posteriormente controlarán los transistores, no podemos permitir que ambas se encuentren a nivel alto al mismo tiempo, ya que conducirían los dos MOSFET y se cortocircuitaría la alimentación de los LED.

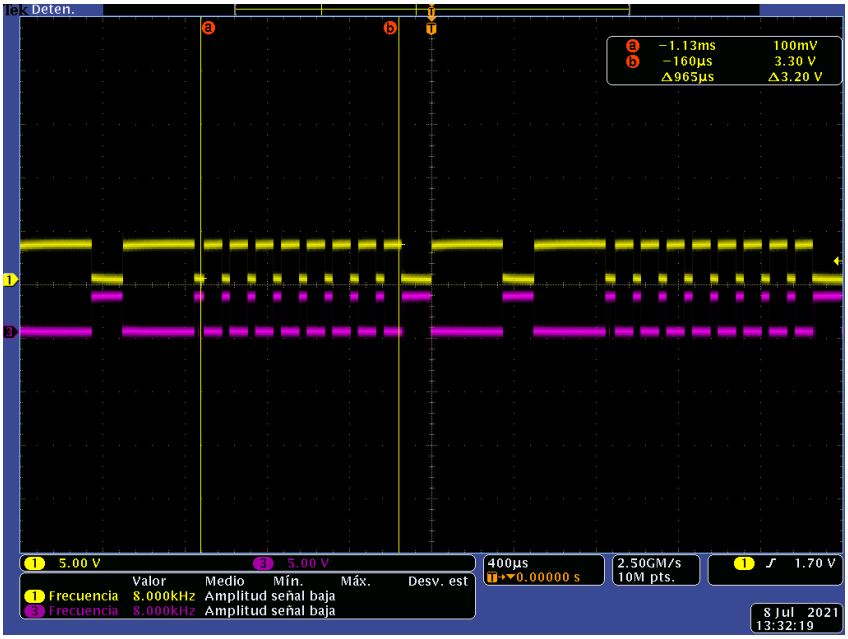


*Figura 20. Resultados prácticos modo PWM – GPIO26 y GPIO25 (2KHz, 8KHz, 70%).*

Finalizando con la etapa de los resultados prácticos, hemos variado el ciclo de trabajo para que sea inferior además de seleccionar otras dos frecuencias distintas. En la Figura 21 observamos las señales para un ciclo de trabajo del 30% y frecuencias de 4 y 9 KHz.

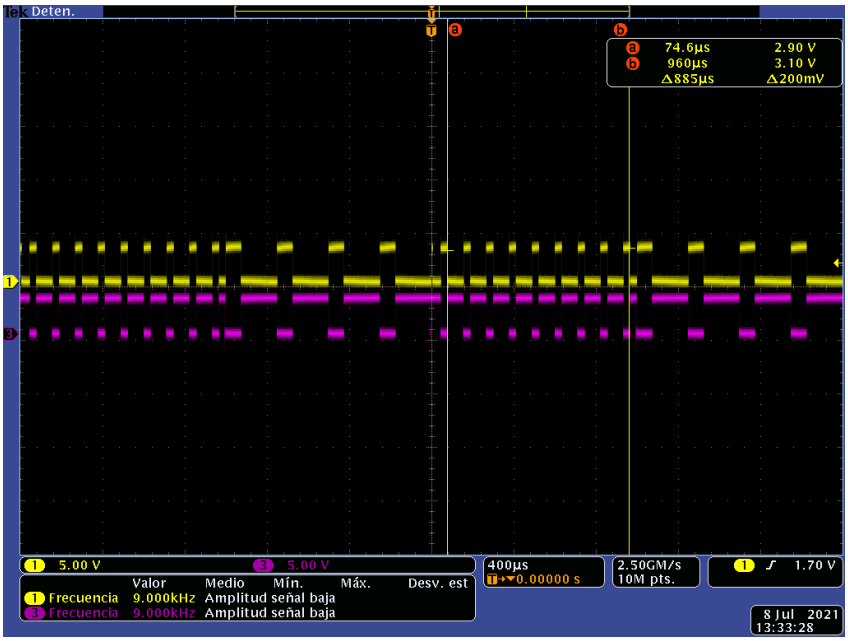


*Figura 21. Resultados prácticos modo PWM – GPIO26 y GPIO25 (4KHz, 9KHz, 30%).*

# 4. Lógica de conmutación

Este apartado lo vamos a dedicar a describir los circuitos electrónicos diseñados para desarrollar el driver de excitación de las luminarias LED, de manera que no emitan de forma continua si no que estén modulados por las señales de salidas del microcontrolador que contienen la información.

## 4.1. Descripción general

Se trata de otra etapa de nuestro proyecto, en este caso se encargará de la conmutación de los transistores MOSFET que posteriormente modularán la señal en la carga, es decir, en las luminarias led.

La etapa en sí es sencilla, se encuentra formada por un controlador de puerta MOSFET aislado y dos transistores MOSFET, uno de ellos será controlado por la primera señal PWM generada por el microcontrolador y el otro por la señal inversa de esta misma.

El controlador de puerta se encuentra aislado entre entrada y salida, y es ideal para conmutar MOSFET de altos y medios voltajes reduciendo al máximo la potencia disipada durante la conmutación.

En la Figura 22 observamos el diagrama de bloques completo, incluyendo microcontrolador, driver, conmutadores (MOSFET) y fuente de alimentación.

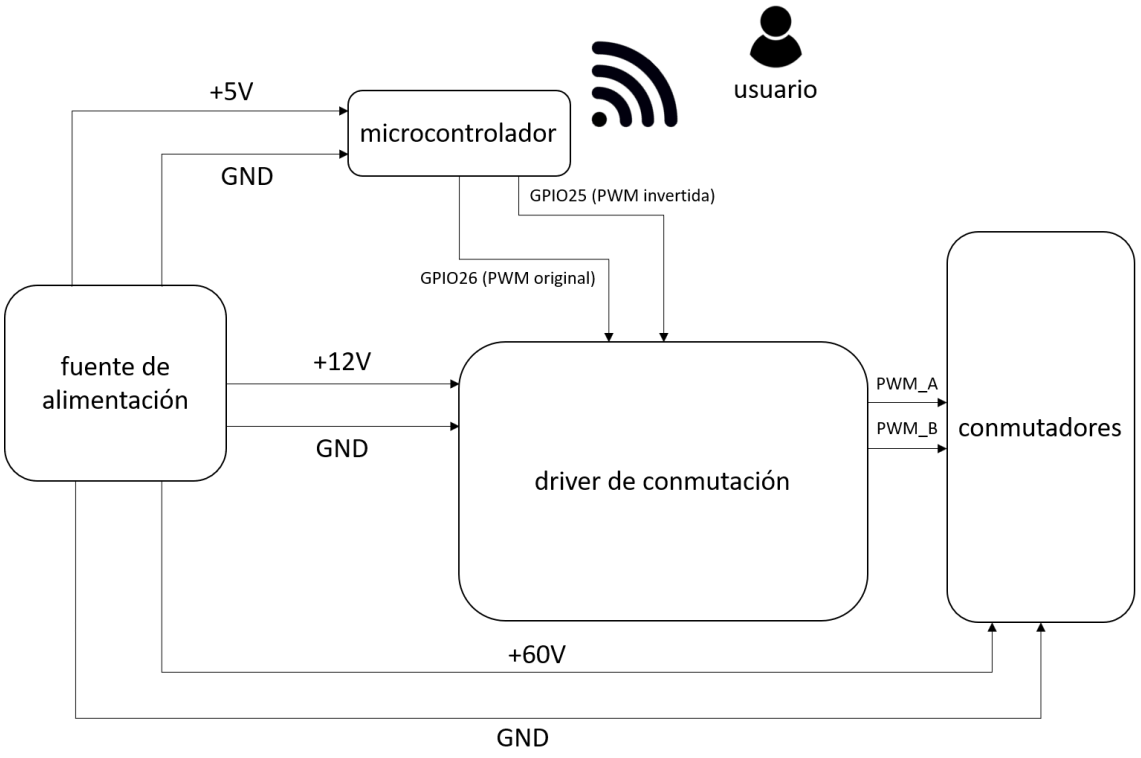


*Figura 22. Diagrama de bloques del proyecto.*

La lógica de conmutación abarca los bloques del driver y conmutadores visibles en la Figura 22. Si englobamos estas dos partes como un único bloque obtendríamos el diagrama visible en la Figura 23.

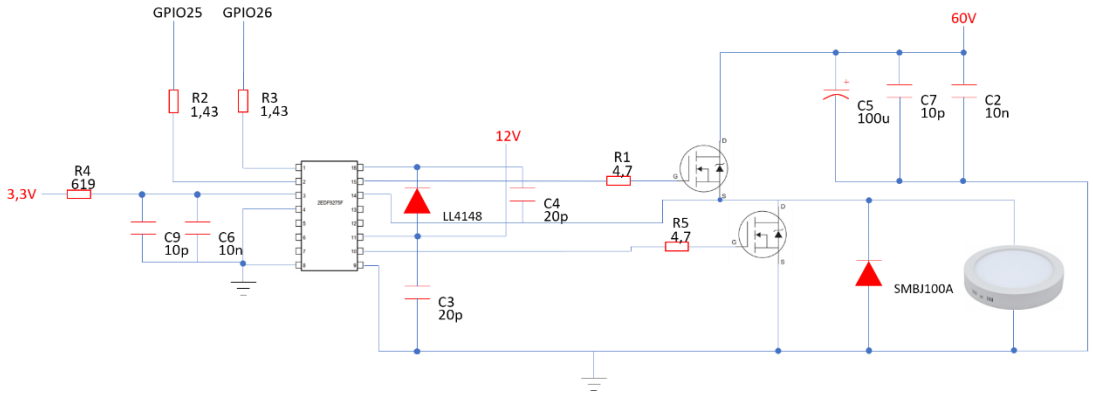


*Figura 23. Diagrama de bloques de la lógica conmutacional*

## 4.2. Material utilizado

Para implementar el circuito diseñado se han utilizado los elementos que se citan a continuación

### 4.2.1. 2EDF9275FXUMA1

Se trata del controlador de puerta para conmutar transistores MOSFET de gran voltaje a altas frecuencias, dado que el circuito integrado permite señales PWM de entrada de hasta 10 MHz (es importante observar también la frecuencia máxima de conmutación del transistor, puesto que debe ser capaz de soportar la misma frecuencia si queremos que ambos trabajen a mismo rendimiento).

En este caso se ha elegido este modelo de la familia Infineon debido a su aislamiento y a su alta velocidad de conmutación. En nuestro caso no llegaremos a rangos de frecuencia tan altos porque no son necesarias para nuestra aplicación actual, pero para posibles mejoras en un futuro podríamos aumentar dicho rango [14]. En la Figura 25 podemos observar el diagrama de bloques interno del chip.

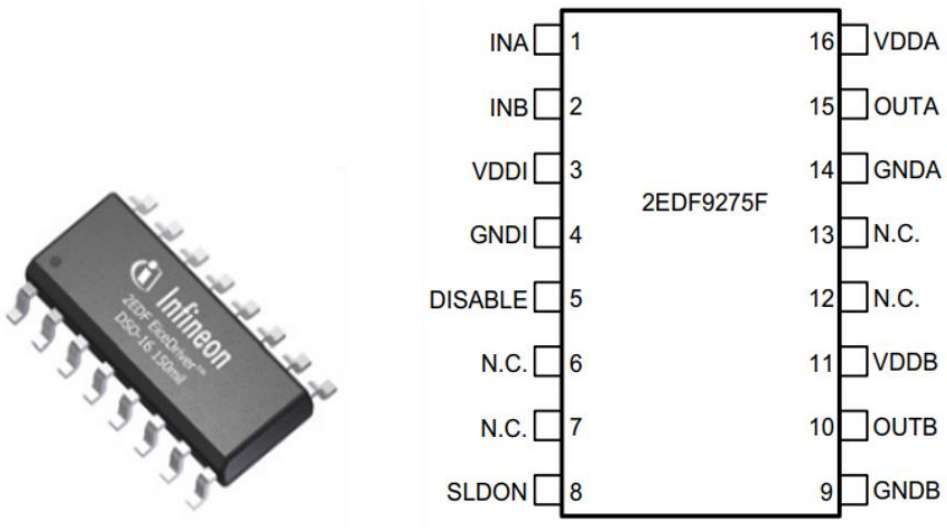


*Figura 24. 2EDF9275FXUMA1 – CI & pinout [14].*

Sus características más destacadas son:

- Corrientes de salida elevadas.
- Aislamiento entre entrada y salida.
- Conmutación de PWM de hasta 10 MHz.
- Retardo de propagación de señal PWM bajo.
- Tiempo muerto controlable a través de resistencia interna (15 ns a 250 ns).
- Tensión de salida en el rango de 4,5V a 20V.
- Rango de temperaturas permitida elevado, desde -40 ºC a +150 ºC.
- Empaquetado DSO16.

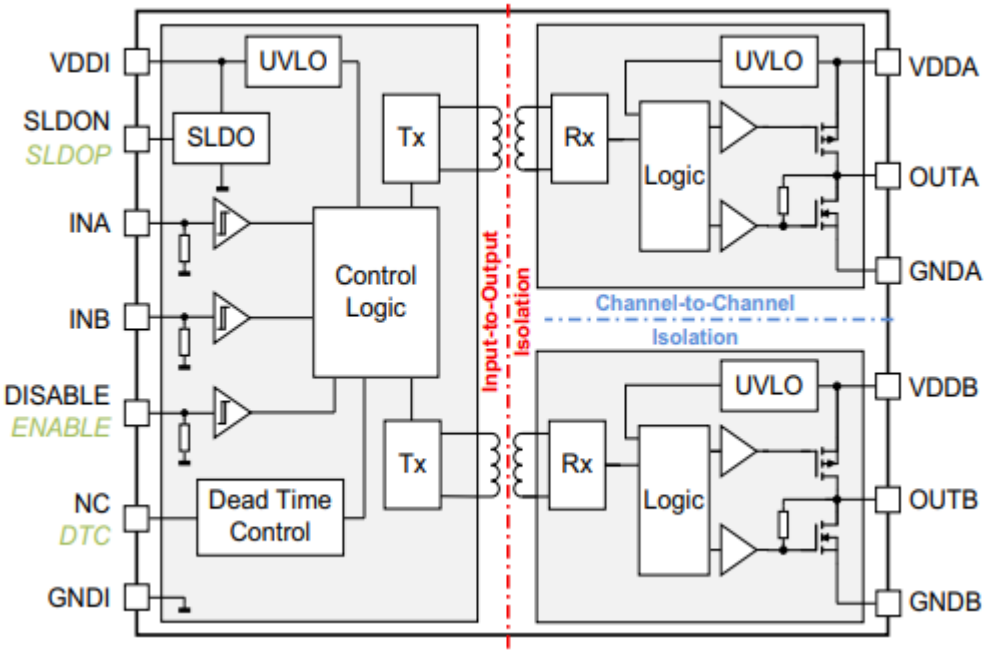


*Figura 25. Diagrama de bloques interno 2EDF9275FXUMA1 [14].*

A continuación, vamos a detallar en la Tabla 2 los pines del circuito integrado del controlador de puerta, así como la función de cada uno de ellos:

| Pin | Símbolo | Función |
|---|---|---|
| 1 | INA | Señal de entrada digital CMOS/TTL para canal A con resistencia *pull-down* interna.<br>Si no se usa el canal se recomienda conectar el pin a GNDI (no dejar al aire). |
| 2 | INB | Señal de entrada digital CMOS/TTL para canal B con resistencia *pull-down* interna.<br>Si no se usa el canal se recomienda conectar el pin a GNDI (no dejar al aire). |
| 3 | VDDI | Tensión de alimentación en la entrada (3,3V).<br>Se recomienda la colocación de un condensador de *bypass* desde VDDI a GNDI. |
| 4 | GNDI | Toma de tierra en la entrada.<br>Todas las señales en el lado de entrada se encuentran referidas a dicho pin. |
| 5 | DISABLE | Señal de entrada digital CMOS/TTL para ambos canales con resistencia de *pull-down* interna.<br>Un nivel alto deshabilita ambos canales a la salida. |
| 6 | N.C | No conectado, dejar al aire. |
| 7 | N.C | No conectado, dejar al aire. |
| 8 | SLDON | Tensión de 3,3V si lo dejamos al aire o conectamos directamente con VDDI.<br>Si lo conectamos a GNDI, se activa SLDO y se puede usar una tensión mayor de 3,5V.<br>Resistencia de *pull-up* interna a VDDI. |
| 9 | GNDB | Toma de tierra para el canal B de salida. |
| 10 | OUTB | Controlador de puerta en la salida para el canal B. |
| 11 | VDDB | Tensión de alimentación en la salida para el canal B.<br>Se recomienda la colocación de un condensador de bypass desde VDDB a GNDB. |
| 12 | N.C | No conectado, dejar al aire para aislamiento. |
| 13 | N.C | No conectado, dejar al aire para aislamiento. |
| 14 | GNDA | Toma de tierra para el canal A de salida. |
| 15 | OUTA | Controlador de puerta en la salida para el canal A. |
| 16 | VDDA | Tensión de alimentación en la salida para el canal A.<br>Se recomienda la colocación de un condensador de bypass desde VDDA a GNDA. |

*Tabla 2.Pines 2EDF9275FXUMA1 (DSO16).*

### 4.2.2. IRF7465PBF

El MOSFET de potencia utilizado es también de la familia de Infineon. Este modelo en concreto se puede usar para una gran cantidad de aplicaciones, posee una carga

puerta-drenador baja para reducir las pérdidas de potencia en el proceso de conmutación del transistor, así como un voltaje y una corriente de avalancha completamente caracterizados [15].

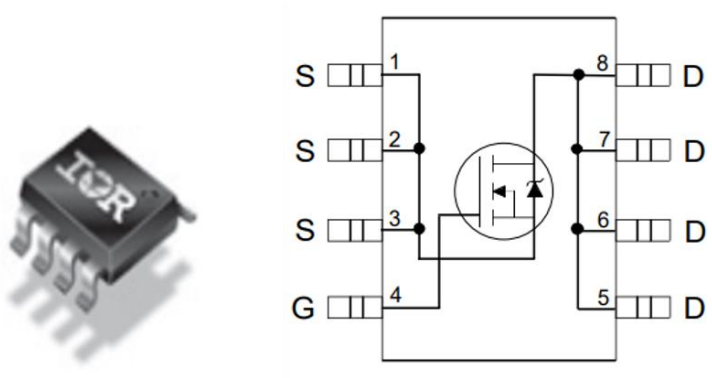


*Figura 26. IRF7465PBF – CI & pinout [15].*

Sus características más destacadas son:

- Mínimo consumo de potencia durante la conmutación.
- Tensión y voltaje de avalancha completamente caracterizados.
- $R_{DS(on)}$ máxima de 0,28Ω con una tensión entre puerta y surtidor de 10V.
- Apto para conmutaciones elevadas.
- Empaquetado SO-8.

En la Tabla 3 podemos observar la asignación de pines del transistor de conmutación con su función indicada:

| Pin | Símbolo | Función |
|---|---|---|
| 1 | S | Surtidor |
| 2 | S | Surtidor |
| 3 | S | Surtidor |
| 4 | G | Puerta |
| 5 | D | Drenador |
| 6 | D | Drenador |
| 7 | D | Drenador |
| 8 | D | Drenador |

*Tabla 3. IRF7465PBF (SO-8).*

## 4.3. Montaje PCB del driver

En la hoja de características de nuestro controlador de puerta MOSFET [14] podemos ver diversas aplicaciones del circuito integrado, basándonos en ello hemos desarrollado nuestro propio circuito de conmutación con un diagrama de bloques sencillo

pero eficaz (Figura 23). Se ha diseñado la placa de circuito impreso para el driver, de manera que junto con el microcontrolador conformen todos los bloques de la Figura 22 a excepción de la fuente de alimentación conmutada. En la Figura 27 podemos ver la PCB completa con todos los componentes de montaje superficial solados.

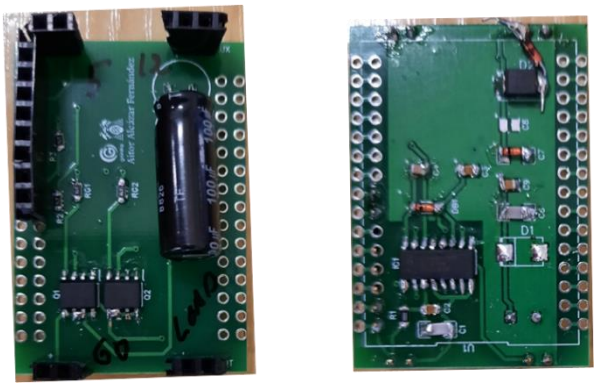

*Figura 27. PCB de la lógica conmutacional construido.*

# 5. Fuente conmutada de tensión

Este apartado está dedicado al diseño y la implementación de una fuente conmutada de tensión, la cual alimente a todos los componentes del proyecto y nos asegure una tensión estable para garantizar la integridad de cada uno de ellos.

Se ha elegido una configuración en tensión en lugar de corriente, puesto que necesitamos poder cortar a la salida de la fuente los 60V necesarios en las luminarias y con una configuración en corriente no sería posible, puesto que los picos de tensión y corriente que se generaban eran muy elevados.

## 5.1. Descripción general

La última etapa de nuestro proyecto es una fuente conmutada de tensión continua a su salida, que a la vez permita conmutar la misma al llevarla a los led de iluminación soportando dichas conmutaciones y sin presentar grandes variaciones en la tensión y corriente.

Este punto se introduce porque inicialmente se intentó conmutar la fuente de corriente que suelen incorporar las luminarias led, pero no soportan las conmutaciones necesarias y presentan grandes picos de tensión y corriente.

A diferencia de un regulador de tensión, que utilizan transistores trabajando en su zona activa, las fuentes conmutadas conmutan a altas frecuencias los transistores entre sus zonas de corte y saturación funcionando como un interruptor que hace que se incremente la eficiencia de una fuente frente a una lineal, puesto que disipa mucha menos energía en los transistores y resistencias de polarización.

Son muchas las ventajas que nos aporta una fuente conmutada frente a una fuente de alimentación lineal:

- Trabaja a frecuencias del orden de KHz frente a los 50 Hz de una fuente lineal. Esto es una gran ventaja, puesto que podremos reducir el tamaño de los componentes (resistencias, condensadores, bobinados, etc.).
- Las fuentes lineales trabajan siempre con voltajes más altos en la entrada que a la salida, mientras que las conmutadas pueden aumentar o disminuir el voltaje respecto a la entrada o incluso invertirlo.
- En las fuentes conmutadas los transistores están en corte o saturación, por tanto, las pérdidas de energía se reducen considerablemente frente a la fuente lineal, en la cual el transistor se encuentra en su zona activa.
- Se pueden generar múltiples voltajes a partir del mismo transformador.

Existen diversas topologías para configurar una fuente conmutada, las más conocidas son la Buck, Boost, Buck-Boost y Flyback. La eficiencia de cada una de ellas es prácticamente similar, destaca la Buck por poder llegar a obtener salidas de incluso 1000V. La topología más demandada es la Flyback, debido a su bajo coste y a su sencillez respecto al resto.

## 5.2. Estudio de la topología a seleccionar

La topología seleccionada para nuestra fuente ha sido Flyback puesto que tendremos dos salidas (para alimentar los led y el controlador de puerta MOSFET) y esta configuración lo permite.

Podemos identificar dentro de una fuente diversas etapas claramente diferenciadas. Al fin y al cabo, una fuente conmutada se puede catalogar como un convertidor DC-DC, esto quiere decir que la red debe ser previamente rectificada y filtrada para pasar de alterna a continua, por tanto, la primera etapa de una fuente será un rectificador y un filtro.

Una segunda etapa es de control; esta etapa está formada normalmente por un circuito integrado que controla la conmutación de un MOSFET a través del ancho del pulso de una señal PWM. Posteriormente tendríamos el transformador y un diodo rectificador en el bobinado secundario junto con el filtro de salida de la fuente.

Muchos elementos pueden alterar el rendimiento de una fuente; el acoplamiento entre los bobinados del transformador no es perfecto y se pueden generaran inductancias de dispersión (inductancia generada por el flujo de dispersión) que deben ser disipadas, las capacidades parásitas generadas por el MOSFET, etc.

A continuación, en la Figura 29 y en la Figura 30 se muestran tanto la topología de una fuente de tipo Flyback como un circuito de su implementación.

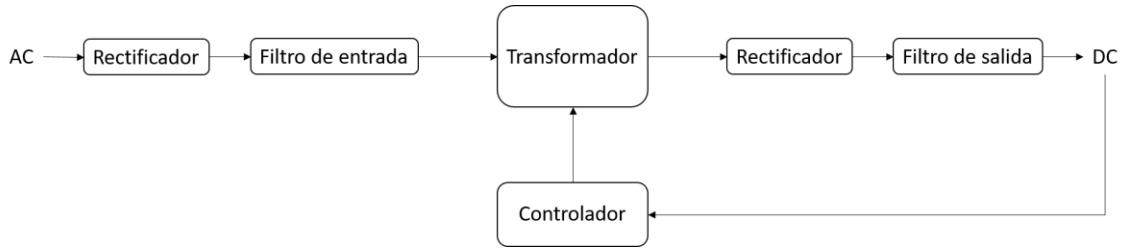


*Figura 28. Diagrama de bloques de fuente Flyback.*

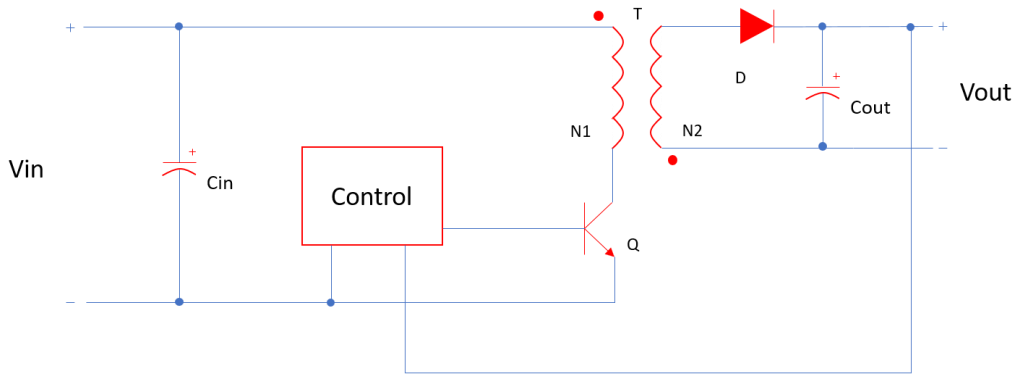


*Figura 29. Topología del convertidor Flyback [21].*

En la Figura 30 podemos observar el funcionamiento de una fuente con topología Flyback en función del estado del MOSFET.

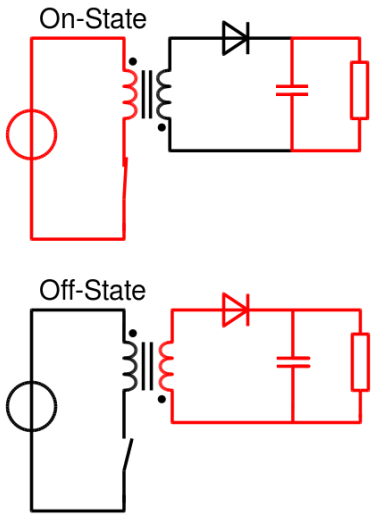


*Figura 30. Funcionamiento transformador Flyback [20].*

En la Figura 29 podemos observar un convertidor Flyback básico con todas las partes descritas anteriormente, a excepción del rectificador y filtro de entrada que se encargan de transformar la alterna de la red en continua.

El rectificador y filtro en la entrada son de vital importancia, el primero de ellos porque se encargará de convertir 220Vac en aproximadamente 311Vcc (220 * √2) y el filtro se encargará de establecer el menor rizado posible

El dispositivo de conmutación debe poder abarcar la tensión de entrada máxima además de la tensión inducida en el transformador. Se podrían utilizar transistores BJT, aunque es recomendable utilizar MOSFET puesto que llegan a frecuencias más altas y eso implica una reducción de tamaño nada despreciable. Además, es conveniente que las pérdidas durante la conmutación sean mínimas, para ello necesitamos que la capacidad entre drenador y fuente sea baja.

A continuación, nos centrarnos en el transformador. Mientras que el transistor (MOSFET preferiblemente) se encuentra en saturación (ON) circula por el bobinado primario corriente que almacena energía en forma de campo magnético en el núcleo, pero sin transmitirla al secundario, la tensión del secundario es negativa y por tanto el diodo rectificador está en inversa y solamente la energía almacenada en el condensador se transfiere a la carga. Cuando conmuta a corte (OFF) se induce en el bobinado secundario la tensión necesaria para que la energía previamente almacenada se transfiera a la carga y al condensador.

Cuando el MOSFET pasa desde saturación a corte la energía del primario se libera a través del secundario, pero debemos tener cuidado con las inductancias de dispersión que hemos comentado, puesto que podrían provocar sobretensiones en el MOSFET. Para evitar este hecho se suele proteger el conmutador con una red *snubber*.

Finalizamos con el rectificador y filtrado de salida, reciben la energía almacenada en el transformador en la etapa de conmutación de saturación a corte. Esta energía provocará que el diodo rectificador conduzca hasta que se agote completamente, por tanto, el diodo solo conducirá durante un tiempo mientras en MOSFET se encuentra en corte. El condensador es de vital importancia, puesto que debe proporcionar a la carga la energía hasta la próxima apertura del interruptor en el siguiente ciclo.

A continuación, se muestra una figura explicativa de las señales típicas de un convertidor Flyback básico:

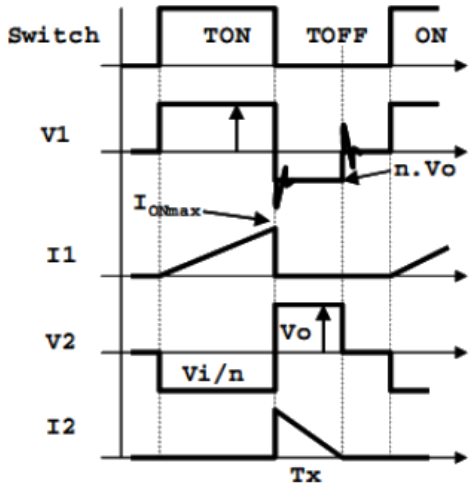


*Figura 31. Señales del convertidor Flyback [17].*

## 5.3. Diseño de la fuente

Para el diseño de la fuente nos hemos apoyado en el simulador de Texas Instruments WEBENCH [22] puesto que, una vez estudiado el funcionamiento, las etapas y los componentes necesarios, nos ha orientado de una manera excepcional dándonos muchísimas posibilidades para cada uno de los componentes, así como distintas topologías que se adecuaran a nuestras necesidades.

Como nuestra fuente debe generar 60V a la salida y ser capaz de suministrar una corriente de 500mA, ya que el foco que queremos utilizar como luminaria y transmisión de información requiere 300mA en conducción y 60V hemos elegido el modelo más simple basado en el circuito integrado UCC28704 [23]. En la Figura 32 podemos observar el diagrama de bloques interno del controlador.

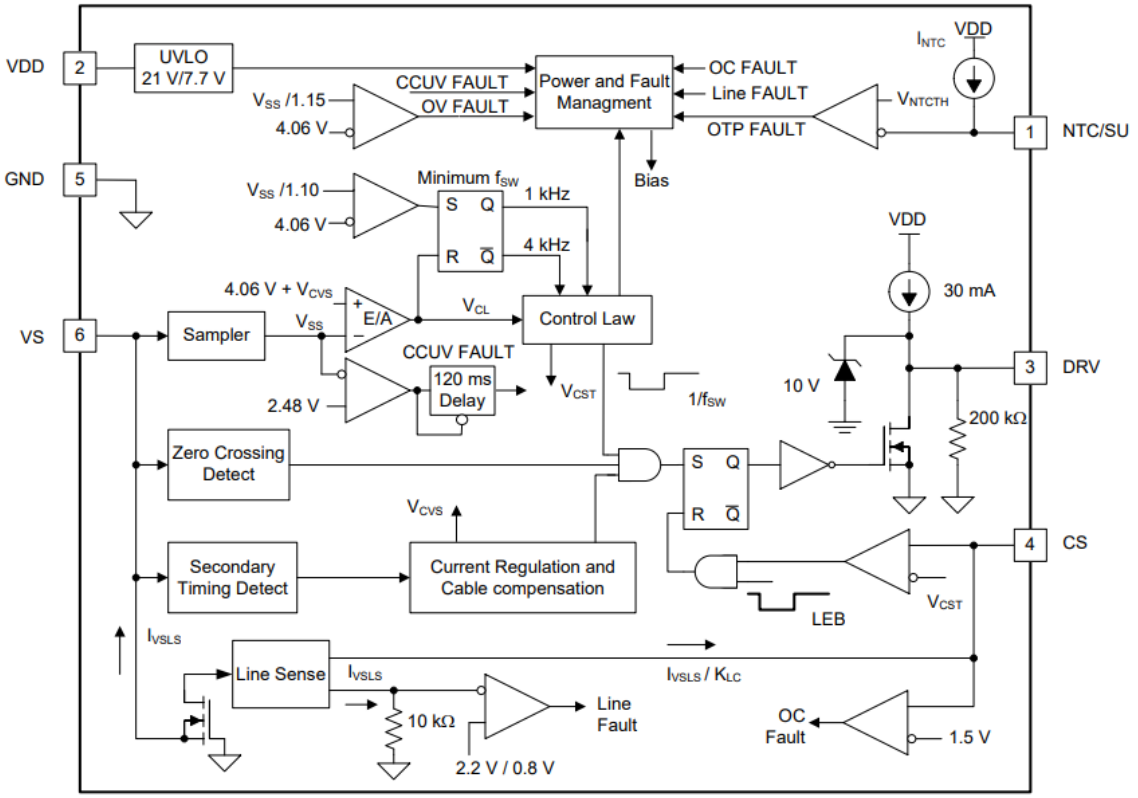


*Figura 32. Diagrama de bloques interno UCC28704.*

Se trata de un controlador Flyback regulado por PWM ideal para fuentes AC-DC como la nuestra, de alta eficiencia y bajo consumo de energía. Su consumo de corriente durante el inicio es extremadamente bajo y permite el control de la corriente de salida sin la necesidad de un optoacoplador o circuitos de realimentación desde el circuito secundario.

### 5.3.1. Material utilizado

Los componentes más importantes son los que componen la etapa de control y el transformador, puesto que conforman la base de la fuente, aunque no debemos perder de vista los restantes.

#### *5.3.1.1.* UCC28704

Como hemos introducido, es un controlador Flyback regulado, por modulación PWM, de alta eficiencia y bajo consumo de energía. No necesita un optoacoplador ni realimentación ya que el circuito primario está regulado, permite una potencia de entrada sin carga inferior a los 30mW y su frecuencia de conmutación interna máxima para obtener la tensión continua de salida es de 85 KHz. Incluso tiene un puerto de interfaz NTC para protección contra las sobre temperaturas elevadas de la placa o de los componentes [23].

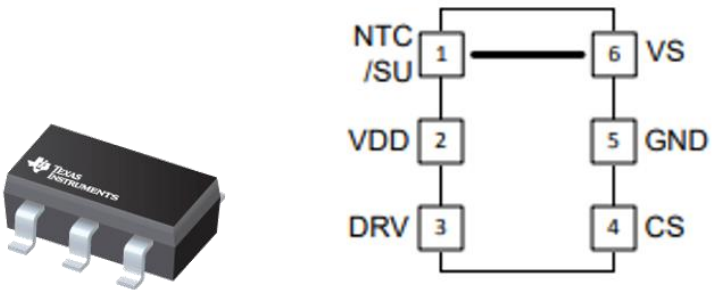


*Figura 33. UCC28704 – CI & pinout [23].*

Sus características más destacadas son:

- Regulación del circuito primario, sin necesidad de usar realimentación desde el secundario.
- No es necesario el uso de optoacoplador.
- Compatible con rectificador síncrono.
- Permite potencia de entrada sin carga inferior a los 30mW.
- Compensación de cables.
- Frecuencia máxima de 85 KHz.
- Interfaz NTC.
- Empaquetado SOT23-6.

La asignación de pines del control con su función lo podemos observar en la Tabla 4:

| Pin | Símbolo | Función |
|---|---|---|
| 1 | NTC/SU | Pin multifunción. En primer lugar, proporciona una interfaz para resistencia externa NTC para la detección de temperatura.<br>Si se pone el pin a nivel bajo, se apaga la acción PWM.<br>Si no se va a hacer uso, dejar al aire. |
| 2 | VDD | Pin de entrada de alimentación de polarización. |
| 3 | DRV | Pin de salida que acciona sobre la puerta de un MOSFET. |
| 4 | CS | La entrada se debe conectar a una resistencia de detección de corriente, referenciada a masa con el conmutador. La tensión que resulta se utiliza para supervisar la corriente de pico del circuito primario.<br>Se añade otra resistencia en serie para compensar los niveles de corriente del circuito primario cuando varía la entrada de la red CA. |
| 5 | GND | Es el pin de referencia para el controlador. |
| 6 | VS | Pin para proporcionar una tensión y retroalimentación al controlador. Se conecta a un divisor de tensión entre un devanado auxiliar y toma de tierra.<br>La resistencia superior de utiliza para controlar los umbrales de marcha-paro de la red AC. |

*Tabla 4. UCC28704 (SOT23-6)*

#### *5.3.1.2.* STD12N65M2

Como conmutador de la fuente usaremos un MOSFET de potencia de canal N de la familia de STMicroelectronics con tecnología MDmesh™ M2. Posee una resistencia de encendido baja y la conmutación de saturación a corte se encuentra optimizada, por tanto, es ideal para conmutaciones a altas frecuencias [24].

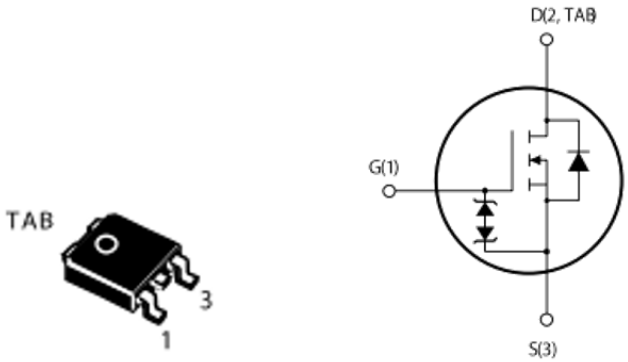


*Figura 34. STD12N65M2 - CI & pinout [24].*

Sus características más destacadas son:

- Carga de puerta baja.
- Protección con diodo Zener.
- Capacidad de trabajar a frecuencias elevadas.
- Empaquetado TO-252.

En la Tabla 5 tenemos indicada la asignación de los pines:

| Pin | Símbolo | Función |
|---|---|---|
| 1 | G | Puerta |
| 2 | D, TAB | Drenador |
| 3 | S | Fuente |

*Tabla 5. STD12N65M2 (TO-252).*

#### *5.3.1.3.* Transformador Würth Elektronik (750841291)

El transformador que utilizaremos para nuestra fuente es de la familia Würth Elektronik y se trata de un transformador Flyback para convertidores DC-DC. Se trata de un transformador estándar con un bobinado principal, un bobinado auxiliar y dos secundarios. En uno de ellos obtendremos una salida de 60V a 400mA, que utilizaremos para alimentar a los led y en el otro tenemos una tensión de 15V a 100mA que se utilizará para alimentar el controlador de puerta MOSFET.

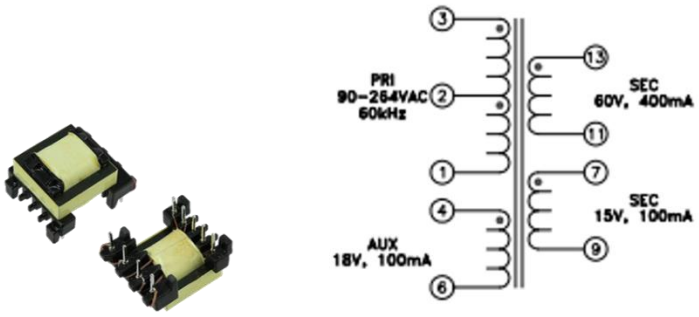


*Figura 35. Transformador (750841291) - CI & pinout [25].*

## 5.4. Diagrama de bloques

Basándonos en las diversas aplicaciones que tiene la unidad de control seleccionada para la fuente, así como en las características del resto de componentes, hemos diseñado la fuente completa con cada una de las etapas descritas en la sección 5.2.

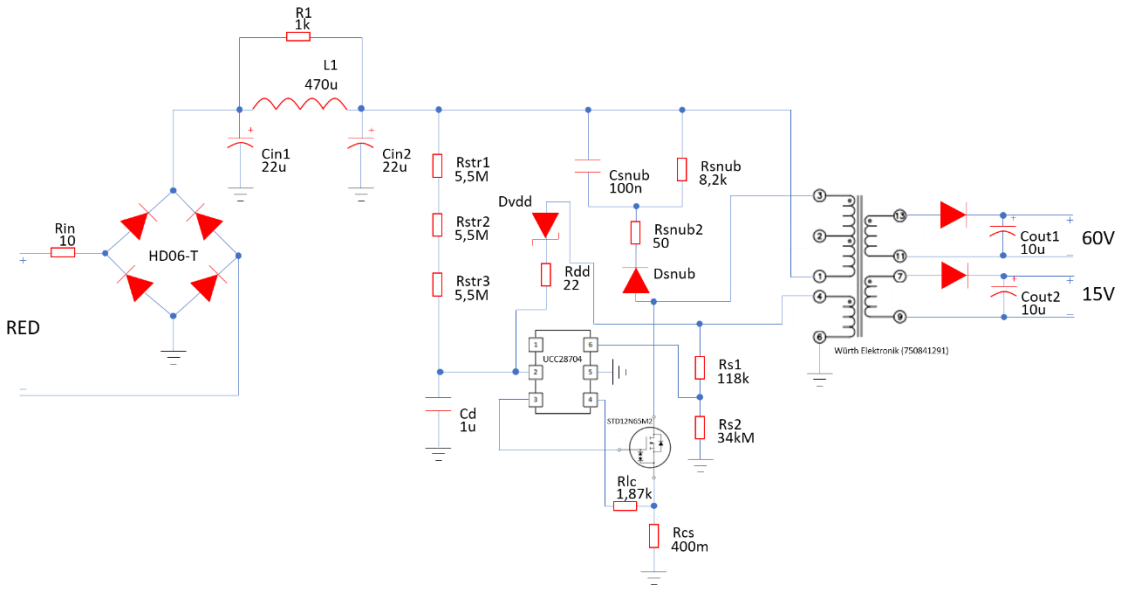


*Figura 36. Diagrama de bloques de la fuente conmutada.*

# 6. Resultados generales

A continuación, vamos a mostrar una serie de fotos dónde podremos observar el resultado final de cada una de las partes que hemos descrito, así como los resultados del proyecto final.

Durante los resultados podremos observar la señal que conmutará las luminarias con un ciclo de trabajo y unas frecuencias determinadas a través del osciloscopio. También captaremos las frecuencias emitidas por el foco a través de una cámara móvil debido al efecto *rolling shuter* de los sensores CMOS que incorporan estos dispositivos.

En la Figura 37 y en la Figura 38 podemos observar el controlador y ambas caras de la PCB de la lógica conmutacional. Ambas se unirían a modo de sándwich para intentar abarcar el menor espacio posible, como podemos observar en la Figura 39.

Finalmente usaríamos una placa para unir todos los bloques como podemos observar en la Figura 40, donde mostramos el microcontrolador unido a la PCB junto con la fuente de alimentación.

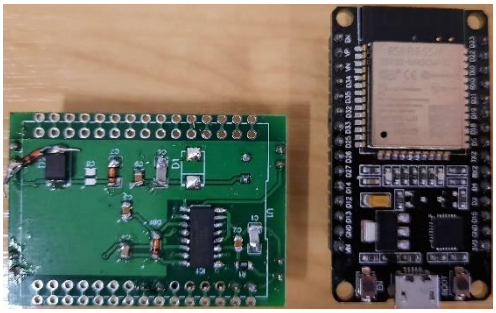

*Figura 37. PCB & ESP32 anverso.*

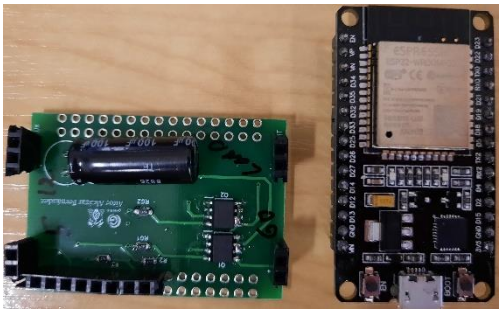

*Figura 38. PCB & ESP32 reverso.*


*Figura 39. Montaje PCB & ESP32.*

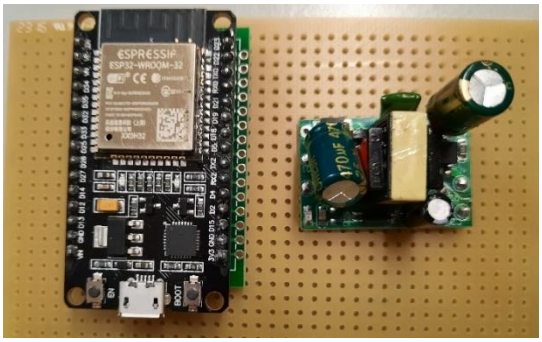


*Figura 40. Montaje PCB & ESP32 con fuente.*

En la Figura 41 observamos el foco junto al osciloscopio. Inicialmente tenemos una señal PWM de un 50% de ciclo de trabajo y frecuencias de 2 y 6 KHz, como podemos observar, de manera más detallada, en la captura del osciloscopio de la Figura 42, la conmutación de las luminarias es muy buena junto con la resolución de la frecuencia.

Observamos también que el tiempo de conmutación de las frecuencias se ajusta al ms que hemos reflejado en la Figura 17.

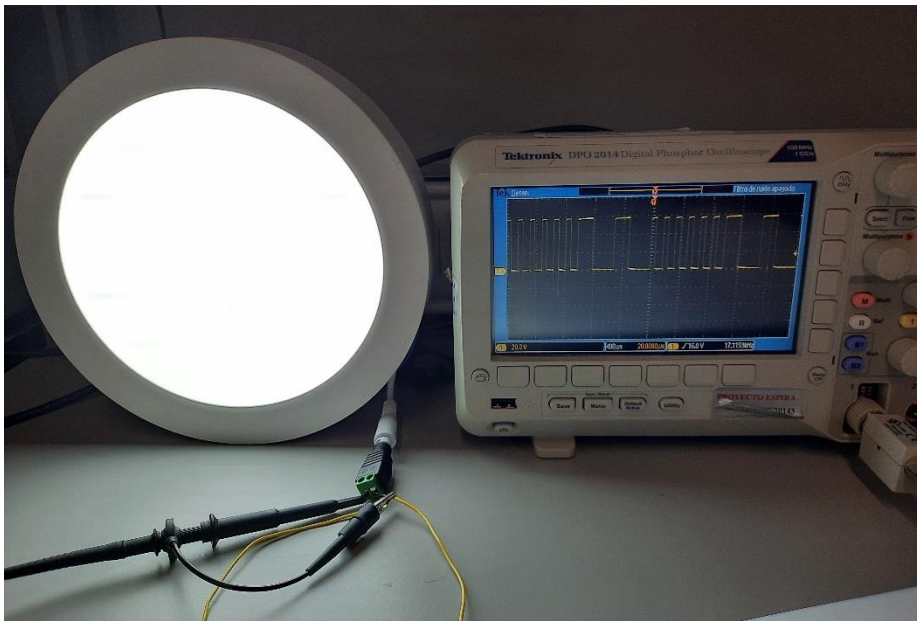


*Figura 41. Resultado final – salida del led (2KHz, 6KHz, 50%) - Montaje.*

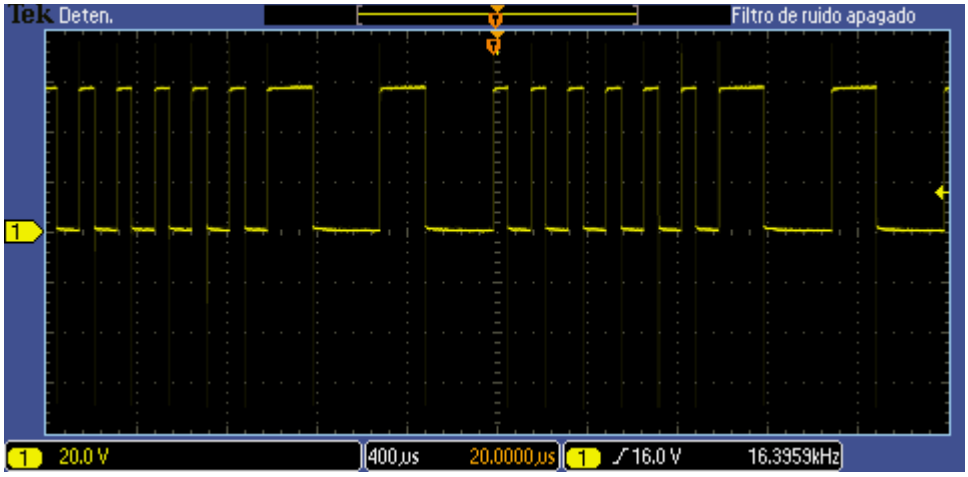


*Figura 42. Resultado final – salida del led (2KHz, 6KHz, 50%).*

Haciendo uso de la tecnología Wi-Fi vamos a ir variando la configuración de las señales y observaremos los cambios que se presentan.

En la Figura 43 podemos observar la trama TCP mandada desde un smartphone conectado a la misma red que el módulo ESP32. Una vez enviada la trama, obtenemos de manera visual como la configuración se ha actualizado, reflejado en la Figura 44.

La Figura 45 ilustra la captura del osciloscopio de la señal con frecuencias de 1 y 5KHz y ciclo de trabajo del 30%.

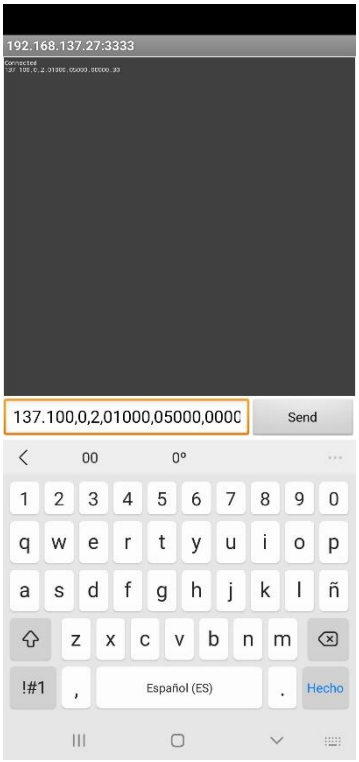


*Figura 43. Trama TCP mandada desde la aplicación TCP Client.*

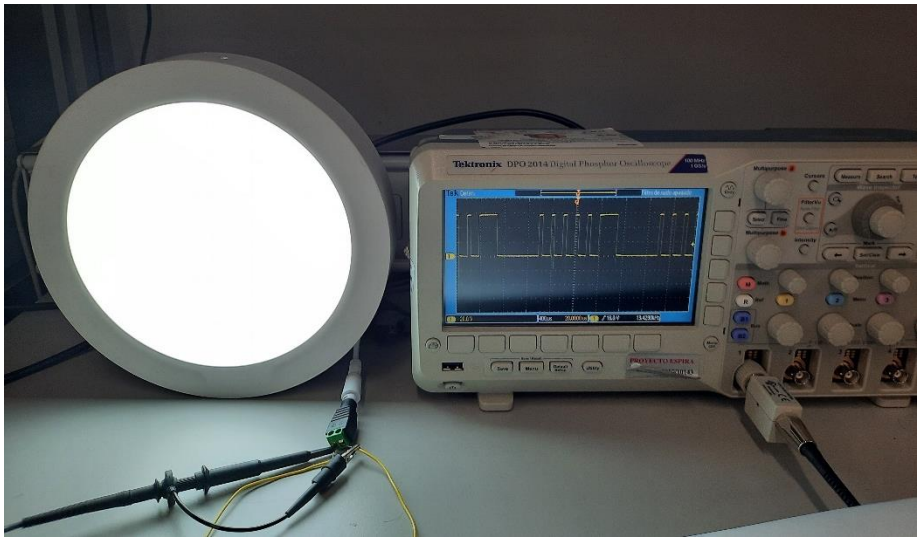


*Figura 44. Resultado final – salida del led (1KHz, 5KHz, 30%) – Montaje.*

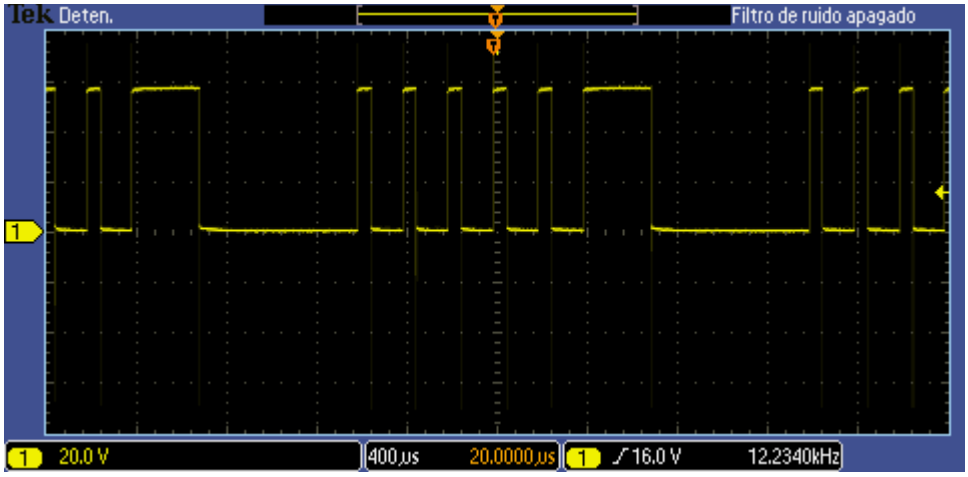


*Figura 45. Resultado final – salida del led (1KHz, 5KHz, 30%).*

A continuación vamos a volver a variar las señales, en éste caso se ha configurado con un ciclo de trabajo del 10% como podemos observar en la trama TCP de la Figura 46.

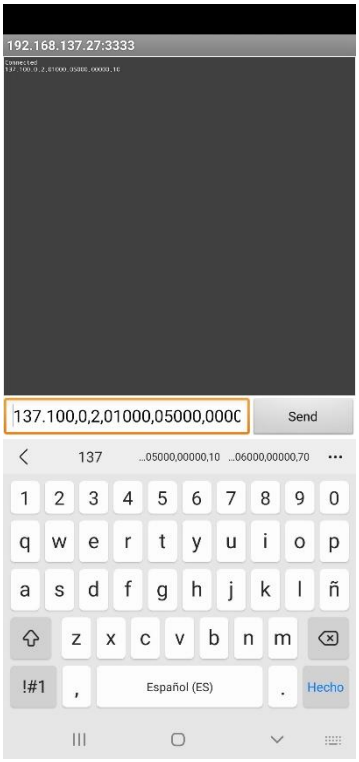


*Figura 46. Trama TCP mandada desde la aplicación TCP Client.*

En la Figura 47 podemos observar claramente como la intensidad del foco ha disminuido considerablemente debido a que el voltaje medio suministrado es muy inferior a los mostrados anteriormente. En la Figura 48 tenemos la captura del osciloscopio que lo ilustra.

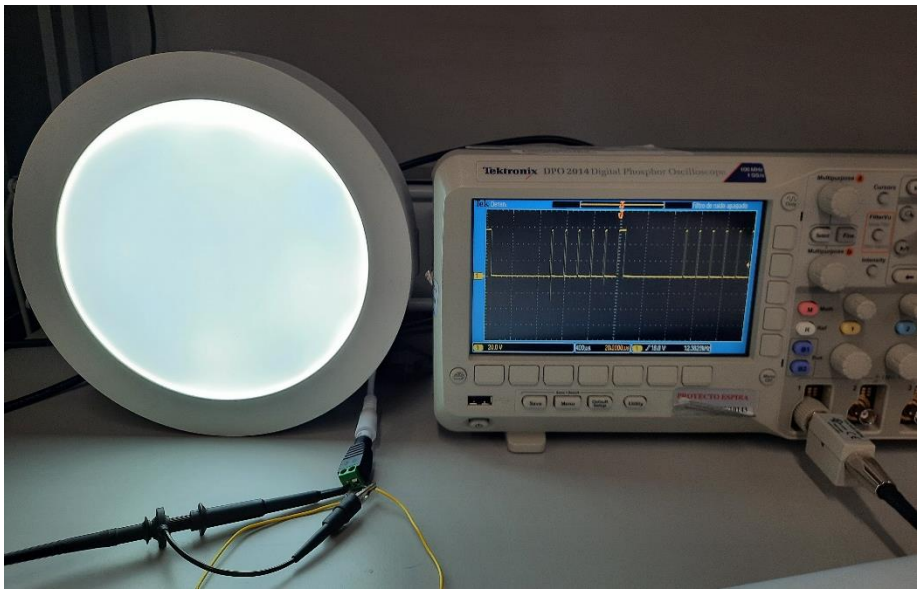


*Figura 47. Resultado final – salida del led (1KHz, 5KHz, 10%) – Montaje.*

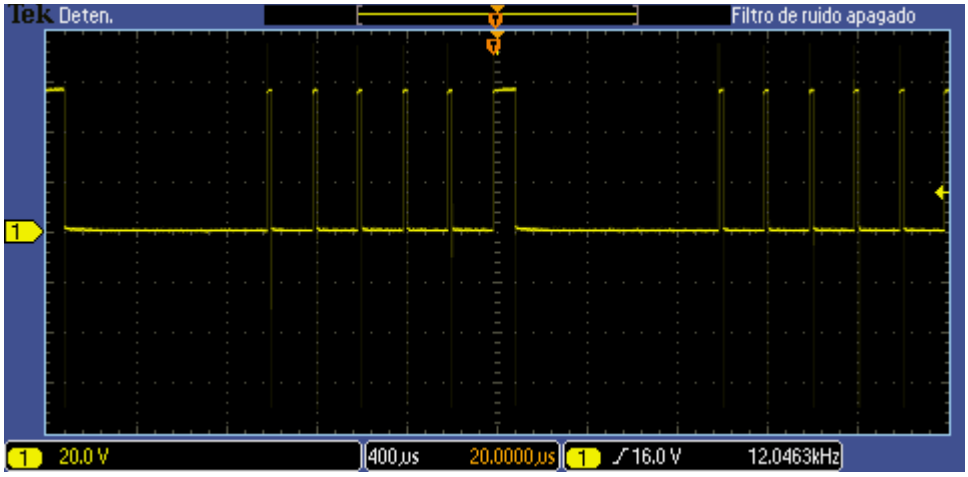


*Figura 48. Resultado final – salida del led (1KHz, 5KHz, 10%).*

En la Figura 49 podemos observar las frecuencias captadas por una cámara móvil y su correspondencia con las obtenidas en el osciloscopio en la Figura 41 y Figura 42.

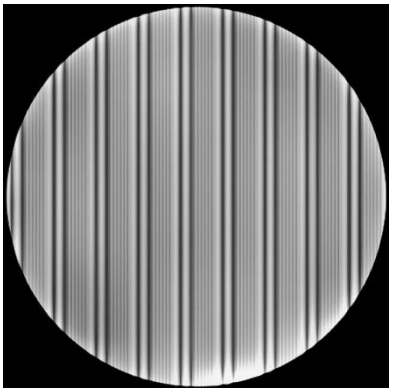

*Figura 49. Detección de frecuencias en el foco con un dispositivo móvil haciendo uso de las características de rolling shuter de su cámara.*

# 7. Conclusiones y líneas de trabajo futuras

Gracias al proyecto desarrollado, no solamente hemos tratado materias que se ha impartido a lo largo del grado, sino que hemos aprendido nuevos aspectos y nuevas tecnologías.

Ha sido un proyecto muy enriquecedor y muy útil para un ingeniero, puesto que cada una de las partes que lo componen se ha desarrollado partiendo de un estudio inicial de los requisitos que se necesitan, una búsqueda de componentes que se ajustaran a nuestras necesidades, diseñando cada una de las partes acorde a las especificaciones requeridas, análisis teóricos de los resultados deseados y banco de pruebas en el laboratorio para corroborar el funcionamiento completo del sistema. Estas etapas son las que realiza cualquier ingeniero para poder llevar a cabo un trabajo productivo, limpio y eficaz.

Una de las cosas que hemos aprendido es trabajar con sistema operativo en tiempo real, el cual nos ha aportado una ejecutabilidad buena del sistema. Aunque no todo han sido puntos positivos, puesto que nos hemos encontrado varios problemas a lo largo del proyecto relacionados con el microcontrolador.

Gracias al microcontrolador hemos conseguido poder modular la luz emitida por las luminarias manteniendo una iluminación constante como podemos observar en las fotos y resultados generales de la sección 6.

De la misma manera, al trabajar con dos frecuencias que van variando, seremos capaces de detectar una mayor cantidad de focos, puesto que, si usamos una única frecuencia, en un rango de 1-10 kHz y con incrementos de 1kHz, solamente seremos capaces de detectar 10 focos. Mientas que al usar 2 frecuencias en el mismo rango, podremos detectar hasta 45 focos distintos.

Hemos conseguido poder configurar las señales de forma remota, es decir, sin contacto directo. Esto es un gran avance, puesto que cualquier usuario que tenga acceso podría modificarlas en tiempo real.

La familia de Espressif Systems proporciona un entorno propio para trabajar con sus microcontroladores y diversos *softwares* de terceros para usarlos como editores de código fuente, pero con este entorno (ESP-IDF [36]) no hemos podido acceder a los registros de los periféricos y cuando intentábamos acceder la placa se nos reiniciaba, el sistema operativo debe tener protegida la escritura en los registros. ¿Esto que implica?, que no podemos saber a ciencia cierta que está realizando el microcontrolador internamente, puesto que nos encontramos trabajando a alto nivel.

El repositorio oficial posee una gran lista de funciones con las que podemos configurar todos y cada uno de los módulos internos del ESP32, pero debido al hecho de usarlo a alto nivel no podemos explotar al 100% todas sus capacidades.

No en todos los entornos esta abstracción de lo que realiza internamente el microcontrolador es necesariamente "mala" puesto que, al trabajar con programación de alto nivel las configuraciones, interrupciones, protocolos de comunicación, etc., se hacen mucho más sencillos, ya que tenemos funciones oficiales que hacen todo el trabajo interno en cada periférico por nosotros.

Este aspecto en un simple temporizador nos limita mucho las capacidades que pueda tener, porque si queremos implementar algo que no se pueda realizar con las funciones del entorno de Espressif Systems tendremos que buscar una manera alternativa de desarrollarlo.

En cambio, para una comunicación inalámbrica, ya sea Wi-Fi o bluetooth, nos simplifica muchísimo, puesto que, si solo queremos mandar y recibir por dichas tecnologías, tendremos funciones ya implementadas que realizan dichos procesos.

En nuestro proyecto nos hemos encontrado con limitaciones en los *timer* y en la generación de las señales PWM. Nuestra primera idea era utilizar el propio contador interno del módulo PWM_LED (Figura 12) para conmutar la frecuencia cada ms, esto no ha sido posible puesto que no hemos conseguido modificar el valor de tiempo de la interrupción de dicho módulo (solamente modificable a través de registro).

Este mismo problema nos surgió a la hora de sacar las muestras senoidales, puesto que lo más productivo sería que se interrumpiera la PWM en cada muestra y se modificase el propio ciclo de trabajo una vez concluido el anterior. Tuvimos la necesidad de usar un segundo *timer* (*timer 1*) configurado con el mismo periodo que la señal PWM (sección 3.4.3.2).

A su vez, con el propio *timer* 1 nos encontramos un nuevo inconveniente. En este caso al configurarlo con tiempos de interrupción pequeños (del orden de micro segundos) el FreeRTOS no nos dejaba operar, ya que nos indicaba que la aplicación no era ejecutable. Esto es debido a que si tenemos un tiempo de cuenta de us y queremos que se interrumpa cada 4 us, al microcontrolador no le da tiempo a ejecutar el código de la función de atención a la interrupción (Código 10) antes de que llegue la siguiente (lo más seguro es que el FreeRTOS se encuentre ejecutando tareas propias que desalojen a la interrupción por tener una mayor prioridad).

Para solucionar el inconveniente del *timer* 1 se modificó el *tick rate* del sistema operativo llevándolo al máximo permitido (1 KHz), pero los resultados fueron similares. De manera que solamente hemos sido capaces de generar una señal senoidal de 2,5 KHz y 10 muestras por cada periodo.

A través del *driver* de conmutación hemos sido capaces de conmutar las luminarias con las señales que hemos generado con el microcontrolador de manera eficiente, es decir, con conmutaciones rápidas, pocas pérdidas y sobre todo sin que las luminarias se vean afectadas. Como podemos observar en la Figura 41 y en la Figura 42 la conmutación es bastante aceptable, apenas se producen picos durante el proceso, gracias a los diodos rectificadores de picos que se añadieron para tal propósito.

Es un diseño sencillo, amigable e ideal para aplicaciones de este tipo, ya que el controlador de puerta se encuentra aislado entre la entrada y la salida, esto nos proporciona una mayor seguridad ante alteraciones que puedan perjudicar el funcionamiento.

Por último, gracias a la fuente de alimentación conmutada podemos alimentar cada uno de los componentes de nuestro circuito. Con el diseño hemos conseguido salidas de tensión constantes, sin fluctuaciones y completamente limpias.

Como líneas de trabajo futuro podemos destacar la detección de un mayor número de frecuencias, actualmente en el proyecto hemos trabajado con dos frecuencias, que nos permiten detectar una gran cantidad de luminarias led sin repetir códigos. Si aumentamos el número de frecuencias, aumentaríamos la cantidad de led que podemos detectar, puesto que cada uno emite con una frecuencia distinta. Además, este proceso sería muy sencillo, puesto que solamente deberíamos añadir un canal con una nueva señal PWM emitiendo a una frecuencia distinta de las anteriores.

# 8. Bibliografía


[1] Osram, "Decostar 35 20W 12V 44890 WFL", Osram, Múnich, *Product*. [En línea]. *Available*: https://www.google.es/shopping/product/16698849529729858296?sxsrf=ALeKk00U1TtmNY4A-7FC1l3oRxFpA7edgg:1609361230033&q=precio+bombilla+halogena+25W&biw=683&bih=656&prds=epd:4822037698740429703,prmr:1&sa=X&ved=0ahUKEwiIiI3nyfbtAhUQlhQKHVxnAM8Q8wIItwQ.

[2] Osram, "LED Star GU5,3 2,6W 2.700K", Osram, Múnich, *Product*. [En línea]. *Available*: https://www.google.es/shopping/product/3297995055631683640?sxsrf=ALeKk01lx1Q6IWRwdkatxbROKHPwhuK6ig:1609361515429&q=precio+bombilla+halogena+25W&biw=683&bih=656&prds=epd:10562121828051150760,prmr:1&sa=X&ved=0ahUKEwi50_bsyvbtAhUq8uAKHcTtD8MQ8wIIkgQ.

[3] Espressif Systems, "ESP32 Series Datasheet", Espressif Systems, Shangai, *Tech*, v3.6, 2021. [En línea]. *Available*: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[4] Espressif Systems, "ESP32-WROOM-32 Series Datasheet", Espressif Systems, Shangai, *Tech*, v3.1, 2021. [En línea]. *Available*: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[5] Espressif Systems, "ESP32 Tecnical Reference Manual", Espressif Systems, Shangai, *Tech*, v4.4, 2021. [En línea]. *Available*: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[6] Espressif Systems, "*Get Started*", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/.

[7] Espressif Systems, "ESP-MESH", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/mesh.html.

[8] Espressif Systems, "Wi-Fi", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html.

[9] Espressif Systems, "Wi-Fi *Driver*", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/release-v4.2/esp32/api-guides/wifi.html.

[10] Espressif Systems, "*ESP32-DevKitC V4 Getting Started Guide*", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html.

[11] Dynabook, "Satellite C660-10H", Dynabook, *Product*. [En línea]. *Available*: https://uk.dynabook.com/discontinued-products/satellite-c660-10h/.

[12] Electronics Hub, "ESP32 Pinout | ESP-WROOM-32 Pinout", Electronics Hub, *Tech*. [En línea]. *Available*: https://www.electronicshub.org/esp32-pinout/.

[13] Embedded diaries, “ESP32 with TCP/IP protocol”, Embedded diaries, *Tech*. [En línea]. *Available*: https://embeddeddiaries.com/esp32-with-tcp-ip-protocol/.

[14] Infineon, “*EiceDRIVER™ 2EDi product family*”, Infineon, Múnich, *Tech*, rev.2.6, 2021. [En línea]. *Available*: https://www.infineon.com/dgdl/Infineon-2EDS9265H-DataSheet-v02_06-EN.pdf?fileId=5546d462712ef9b70171832eb2cf1f37.

[15] Infineon, “*Infineon IRF7465 DataSheet*”, Infineon, Múnich, *Tech*, v.1.1, 2021. [En línea]. *Available*: https://www.infineon.com/dgdl/irf7465pbf.pdf?fileId=5546d462533600a4015355fef2681c08.

[16] ON Semiconductor, “LL4118 *Small Signal Diode*”, ON Semiconductor, Phoenix, *Tech*, rev.1.8, 2005. [En línea]. *Available*: https://www.onsemi.com/pdf/datasheet/ll4148-d.pdf.

[17] Myelectronic, “Las fuentes de alimentación conmutadas (*switching*)”, Myelectronic, *Tech*. [En línea]. *Available*: http://www.myelectronic.mipropia.com/Fuentes%20de%20Alimentacion/Las%20Fuentes%20de%20Alimentacion%20Conmutadas%20(Switching).pdf?i=1.

[18] Electronicafacil, “Fuentes conmutadas”, Electronicafacil, *Tech*. [En línea]. *Available*: https://www.electronicafacil.net/tutoriales/Fuentes-conmutadas.html.

[19] Universidad Técnica Federico Santa María, “Introducción a las fuentes conmutadas. Topologías básicas”, Universidad Técnica Federico Santa María, *Tech*. [En línea]. *Available*: http://www.elo.jmc.utfsm.cl/sriquelme/el/apuntes/p1/no%20lineales/lntroduccion%20a%20las%20fuentes%20Conmutadas.pdf.

[20] Wikipedia, “Flyback operating.svg”, Wikipedia, *Image*. [En línea]. *Available*: https://commons.wikimedia.org/wiki/File:Flyback_operating.svg.

[21] 330ohms, “Eliminadores y fuentes conmutadas”, 330ohms, *Image*. [En línea]. *Available*: https://i0.wp.com/cdn.shopify.com/s/files/1/1040/8806/files/conmutador_2.png?w=696&ssl=1.

[22] Texas Instruments, “WEBENCH® Power Designer”, Texas Instruments, Dallas, *Software*. [En línea]. *Available*: https://www.ti.com/design-resources/design-tools-simulation/webench-power-designer.html.

[23] Texas Instruments, “UCC28704 High-Efficiency Off-Line CV and CC Flyback Controller with Primary-Side Regulation”, Texas Instruments, Dallas, *Tech*, 2016. [En línea]. *Available*: https://www.ti.com/lit/ds/symlink/ucc28704.pdf?ts=1626173268391&ref_url=https%253A%252F%252Fwww.google.com%252F.

[24] STMicroelectronics, “STD12N65M2”, STMicroelectronics, Ginebra, *Tech*, rev2, 2015. [En línea]. *Available*: https://www.st.com/en/power-transistors/std12n65m2.html#documentation.

[25] Würth Elektronik, “Transformer 750841291”, Würth Elektronik, Alemania, *Tech*, rev6A, 2016. [En línea]. *Available*: https://www.we-online.com/catalog/datasheet/750841291.pdf.

[26] ESP32 *Forum*, "*USB-to-UART bridge chips*", ESP32 *Forum, Tech*. [En línea]. *Available*: http://esp32.net/usb-uart/.

[27] FreeRTOS, "*The FreeRTOS™ Kernel*", FreeRTOS, *Tech*. [En línea]. *Available*: https://freertos.org/RTOS.html.

[28] GitHub, "*idf-eclipse-plugin*", GitHub, *Tech*. [En línea]. *Available*: https://github.com/espressif/idf-eclipse-plugin.

[29] GitHub, "*vscode-esp-idf-extension*", GitHub, *Tech*. [En línea]. *Available*: https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md.

[30] Espressif Systems, "*Standard Setup of Toolchain for Windows*", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/windows-setup.html.

[31] Mclibre, "Visual Studio Code. Instalación", Mclibre, *Tech*, 2019. [En línea]. *Available*: https://www.mclibre.org/consultar/informatica/lecciones/vsc-instalacion.html.

[32] Arduino, "*Getting Started*", Arduino, *Tech*. [En línea]. *Available*: https://www.arduino.cc/en/Guide.

[33] PlatformIO. "*Professional collaborative platform for embedded development*", PlatformIO, *Tech*. [En línea]. *Available*: https://docs.platformio.org/en/latest/.

[34] Programarfacil, "Arduino IDE entorno de desarrollo oficial", Programarfacil, *Tech*. [En línea]. *Available*: https://programarfacil.com/blog/arduino-blog/arduino-ide/.

[35] PlatformIO, "VSCode", PlatformIO, *Tech*. [En línea]. *Available*: https://docs.platformio.org/en/latest/integration/ide/vscode.html#ide-vscode.

[36] Espressif Systems, "*ESP-IDF Programming Guide*", Espressif Systems, Shangai, *Tech*, 2021. [En línea]. *Available*: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/.

# 9. Anexos

## 9.1. Instalación ESP-IDF con Eclipse

Lo primero, antes de instalar el repositorio y el *plug-in* del *framework* oficial de Espressif Systems, son los requisitos previos de Java, Python y Git.

### 9.1.1. Requisitos previos

Tal y como comentamos en la sección 2.3.2.1 podemos seguir dos procedimientos para su instalación, a través de la herramienta que nos facilita el fabricante [30].

El segundo sería ir instalando manualmente cada uno de los requisitos previos a través de las páginas *web* oficiales.

#### *9.1.1.1.* Java 11 o superior

Para instalar Java es necesario acceder a la página *web* de Oracle y dentro del menú desplegable productos acceder a Java, para posteriormente descargar el ejecutable para Windows (recomendamos siempre usar versiones que hayan sido aprobadas por el fabricante y sean estables, no versiones de prueba).

En la Figura 50 observamos la página web oficial de Oracle indicando dónde deberemos acceder, posteriormente accederemos al menú de descarga de Java, visible en la Figura 51. Una vez seleccionado solamente deberemos elegir la versión a descargar como podemos observar en la Figura 52.

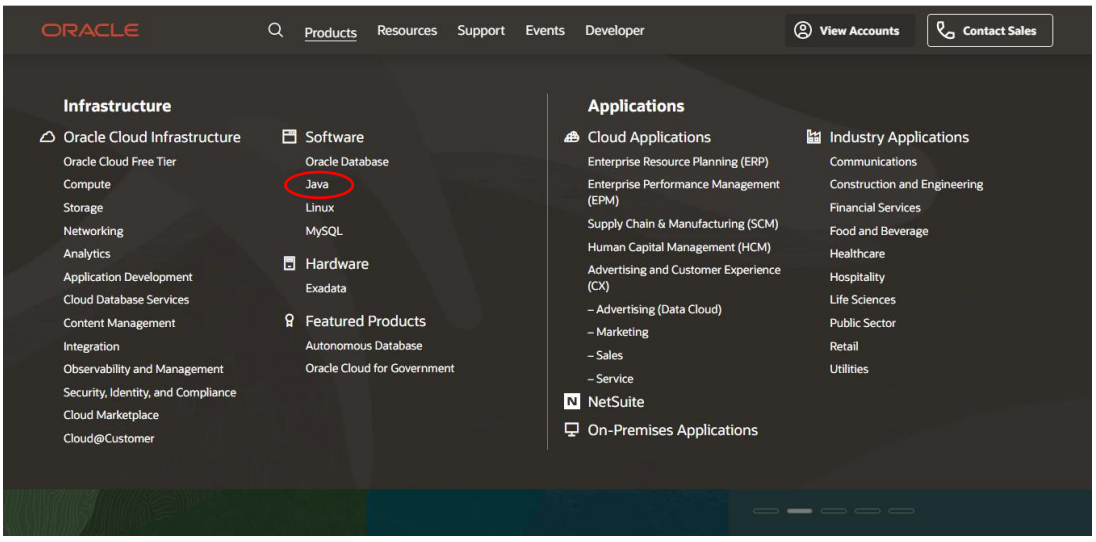


*Figura 50. Página web oficial de Oracle.*

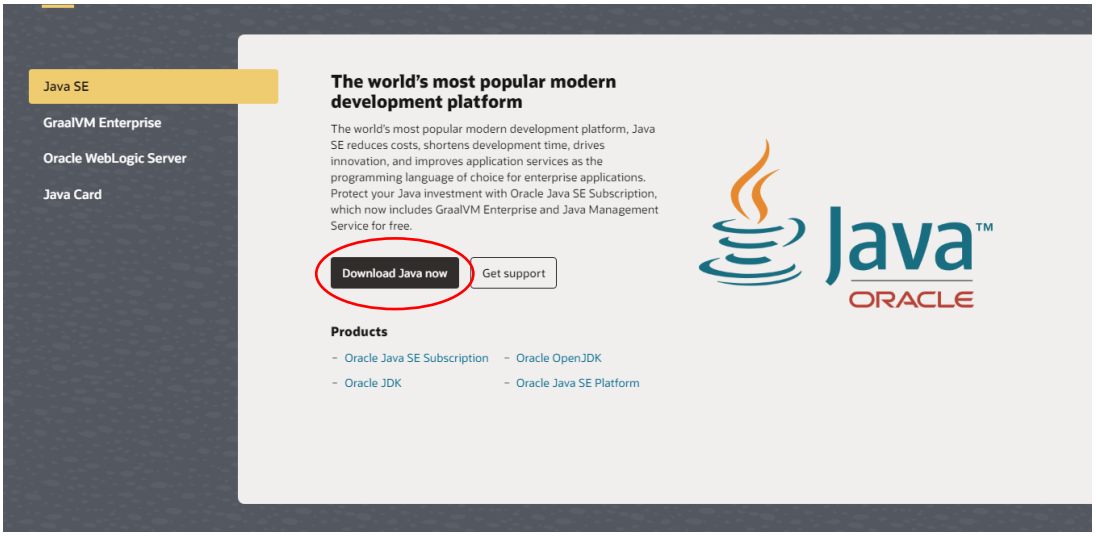


*Figura 51. Hipervínculo para acceder a las descargas de Java.*

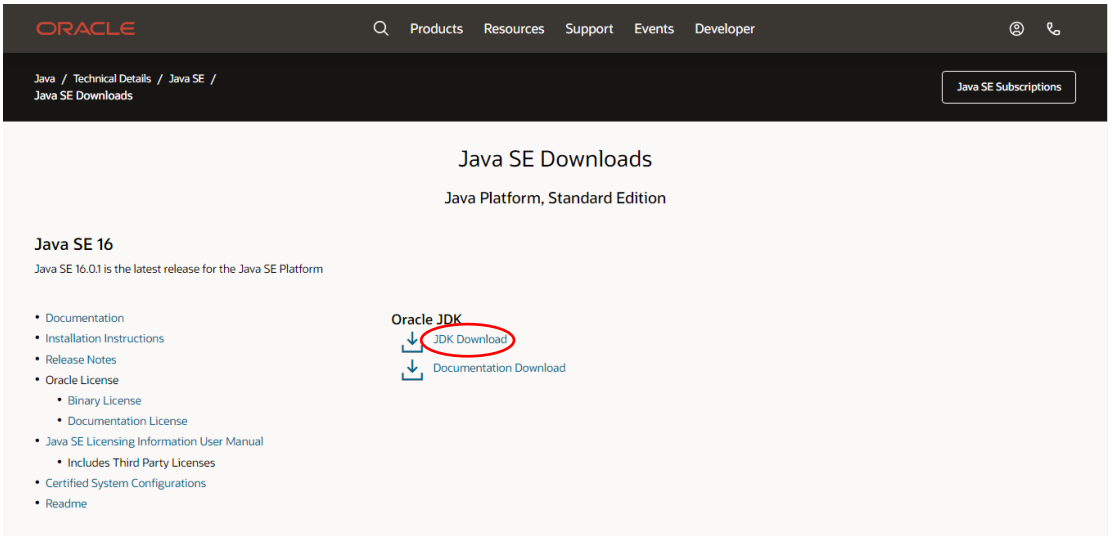


*Figura 52.Enlace de descarga del ejecutable.*

A continuación, ejecutamos el asistente de instalación y seguimos los pasos descritos posteriormente, es un proceso intuitivo y sencillo. La Figura 53, la Figura 54 y la Figura 55 nos muestra paso a paso el procedimiento a seguir para instalar Java.

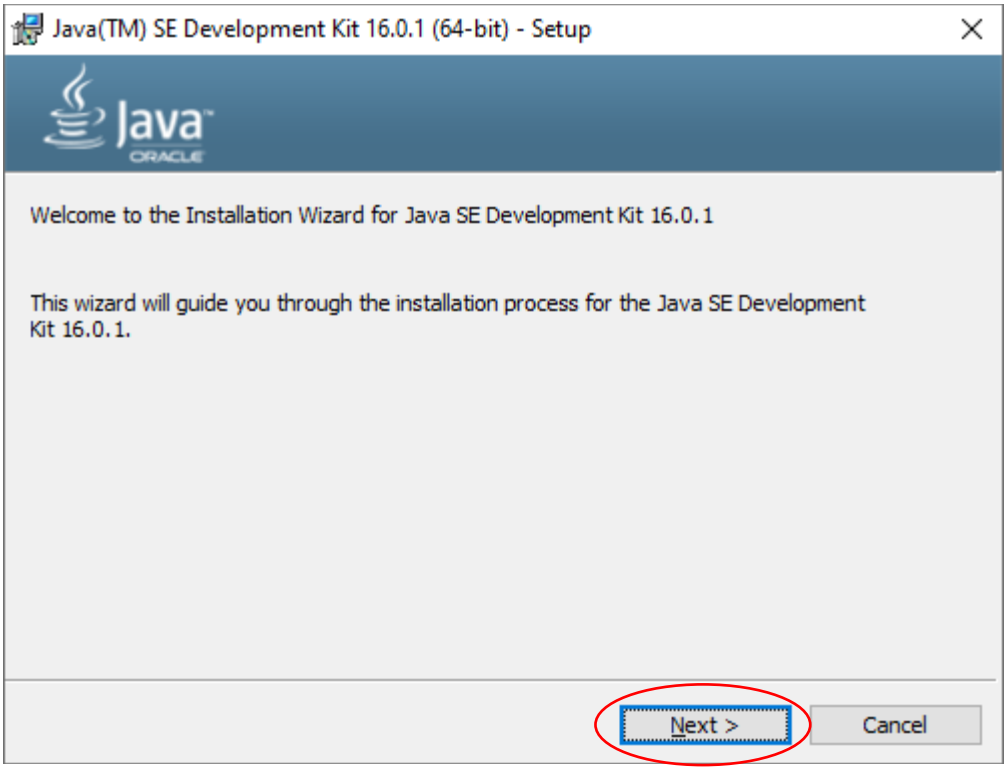


*Figura 53. Guía de instalación Java - 1.*

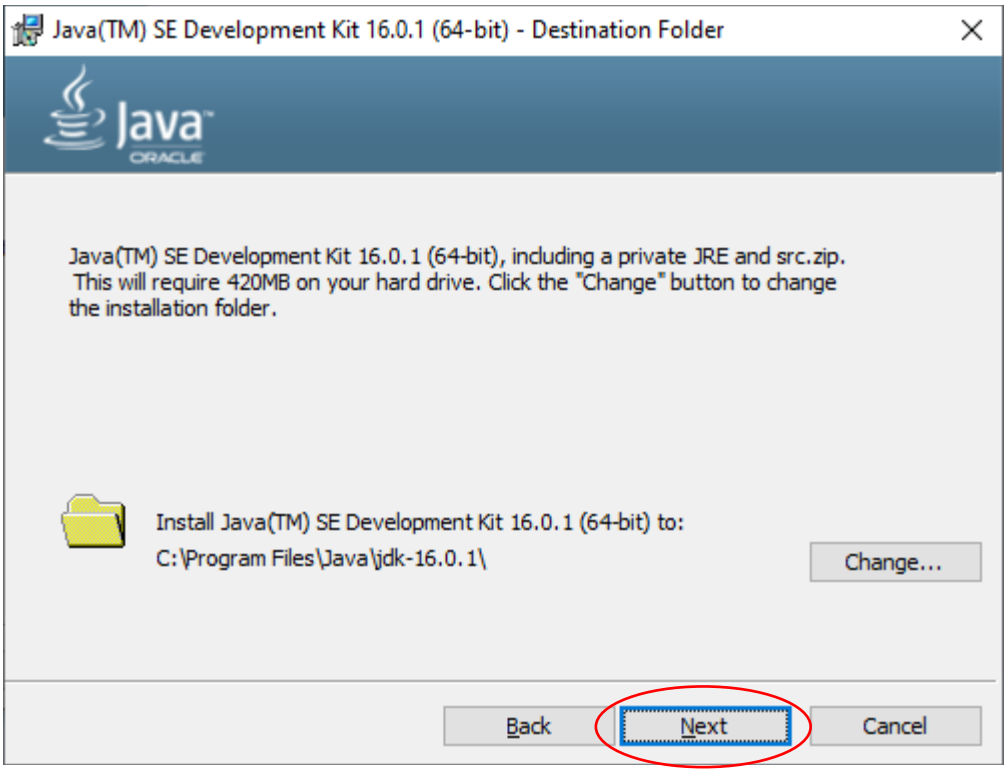


*Figura 54. Guía de instalación Java - 2.*


*Figura 55. Guía de instalación Java - 3.*

#### *9.1.1.2.* Python 3.5 o superior

Al igual que con Java, accederemos al menú de descargas de la página *web* oficial de Python y descargaremos la versión estable más reciente. En la Figura 56 observamos la página oficial de Python y en la Figura 57 el enlace de descarga del ejecutable.

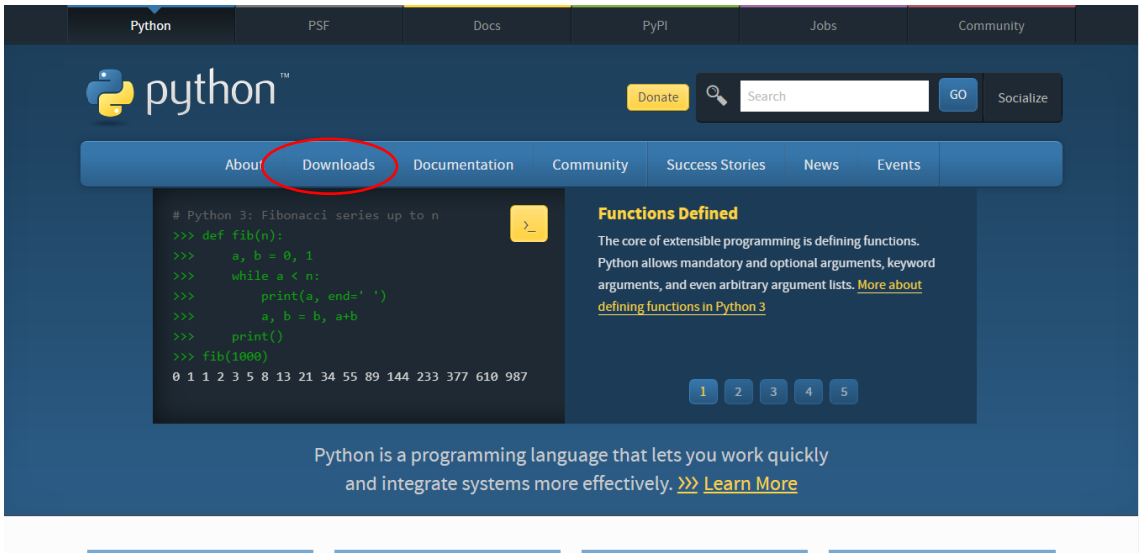


*Figura 56. Página web oficial de Python.*

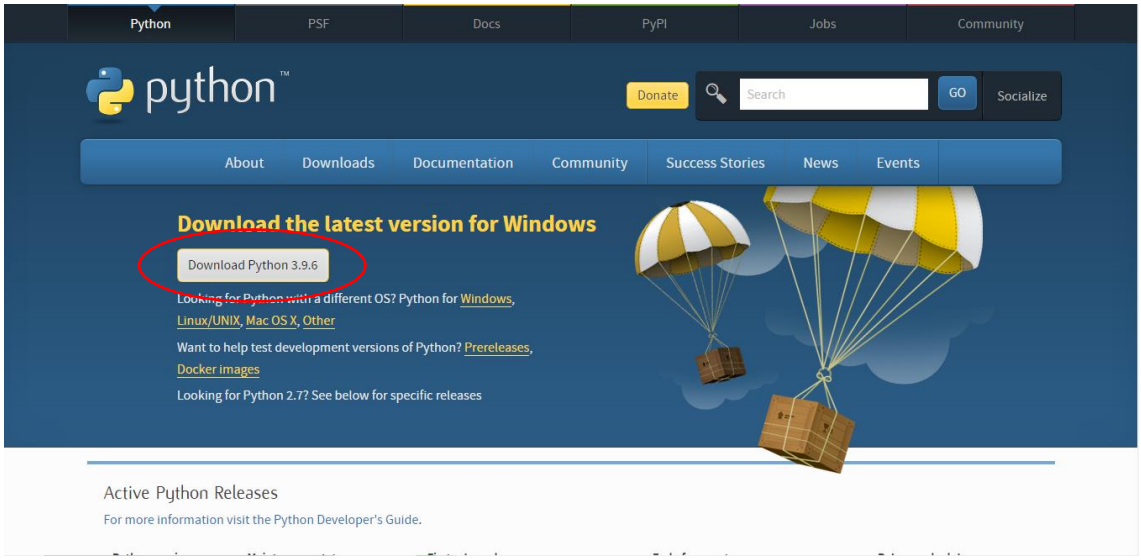


*Figura 57. Hipervínculo para acceder a la descarga de Python.*

Una vez descargado, ejecutamos el asistente de instalación. Es muy importante seleccionar la casilla "*Add Python x.x to PATH*" para que el complemento ESP-IDF en *eclipse* funcione. En la Figura 58 podemos observar la pestaña de instalación en la cual deberemos seleccionar la casilla indicada previamente y en la Figura 59 vemos como el proceso de instalación ha finalizado correctamente.

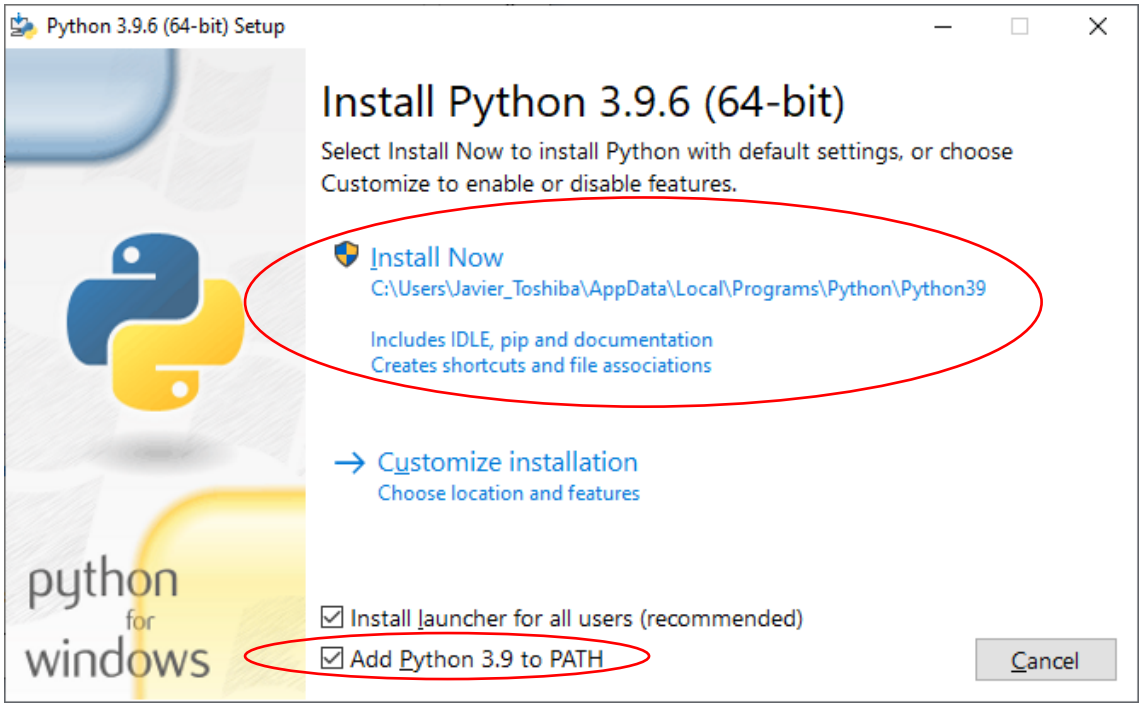


*Figura 58. Guía de instalación Python - 1.*

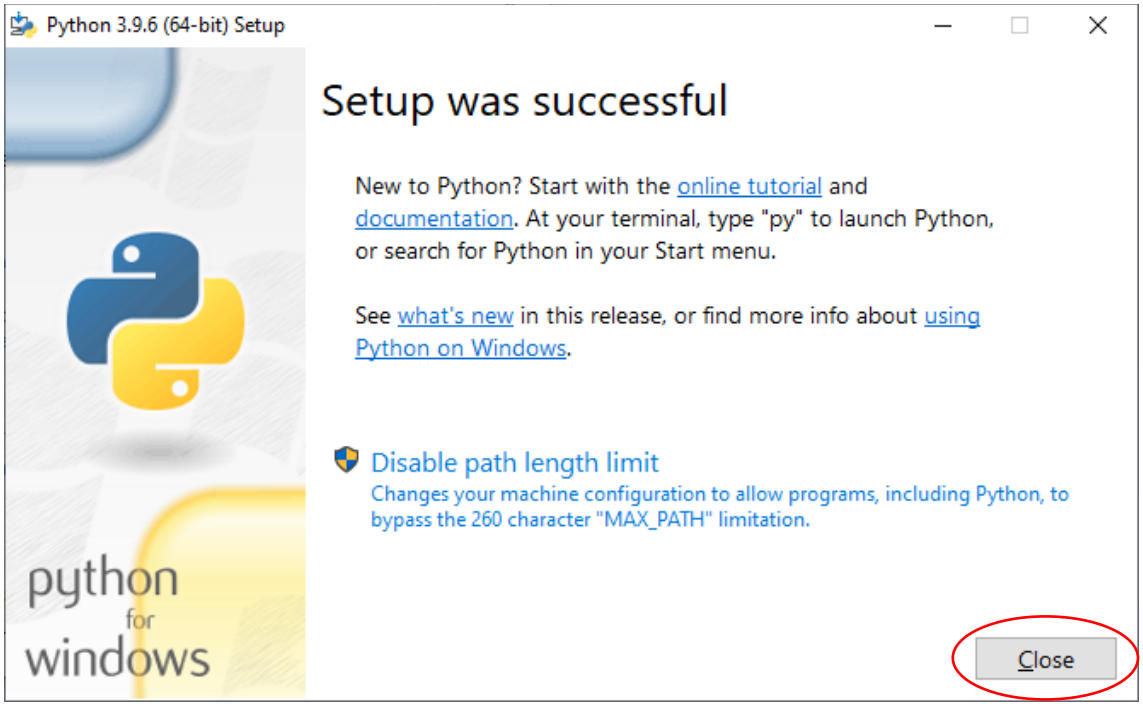


*Figura 59. Guía de instalación Python - 2.*

### *9.1.1.3.* GIT

Siguiendo el mismo método que en los dos requisitos anteriores, accedemos a la página *web* oficial para descargar la versión estable. En la Figura 60 vemos el aspecto de la página web, junto con el hipervínculo de descarga y en la Figura 61 el enlace para descargar el ejecutable.

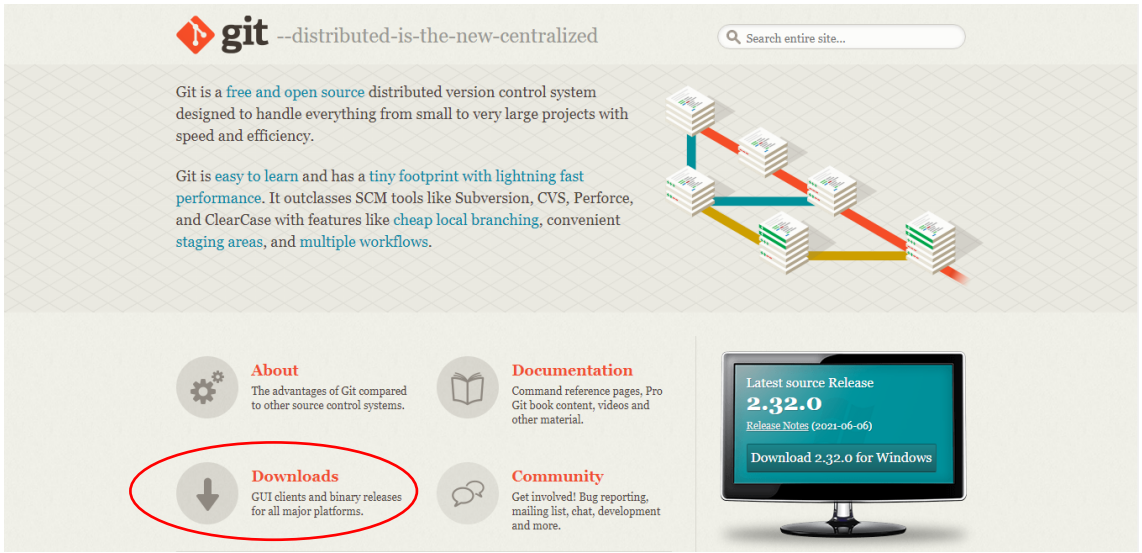


*Figura 60. Página web oficial de GIT.*

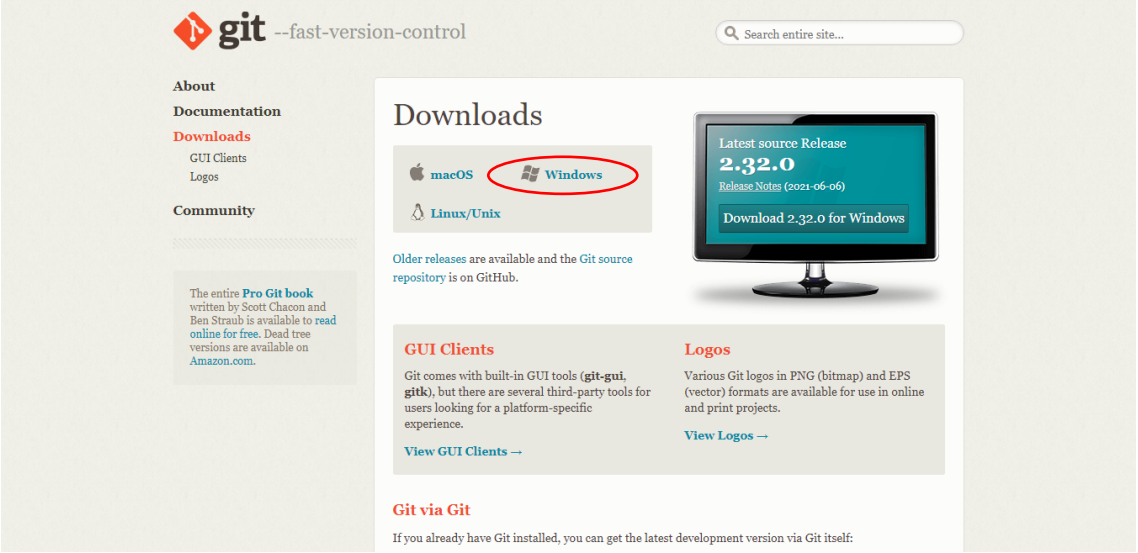


*Figura 61. Hipervínculo para acceder a la descarga de GIT.*

Una vez se ha finalizado la descarga iniciaremos el ejecutable. Deberemos seguir las indicaciones de la Figura 62 donde debemos aceptar los términos y condiciones de la licencia pública. En la Figura 63 observamos el listado de los componentes que podemos instalar (por defecto se selecciona todo).

En la Figura 64 y en la Figura 65 vemos el final del proceso de instalación.


*Figura 62. Guía de instalación GIT - 1.*

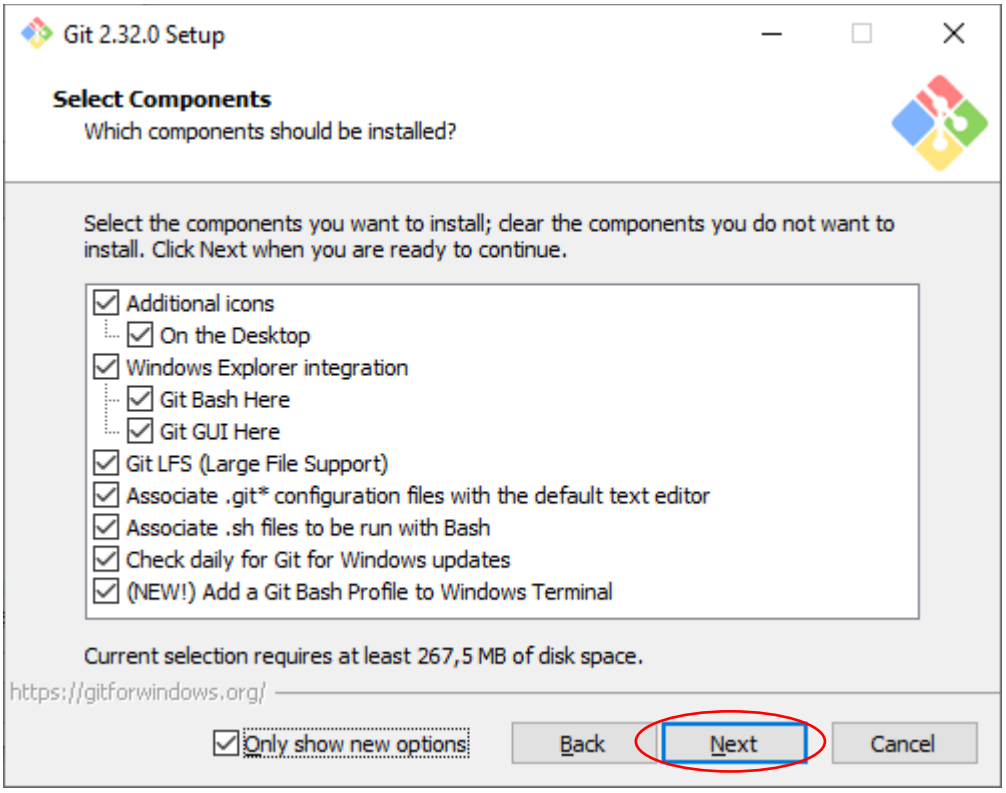


*Figura 63. Guía de instalación GIT - 2.*

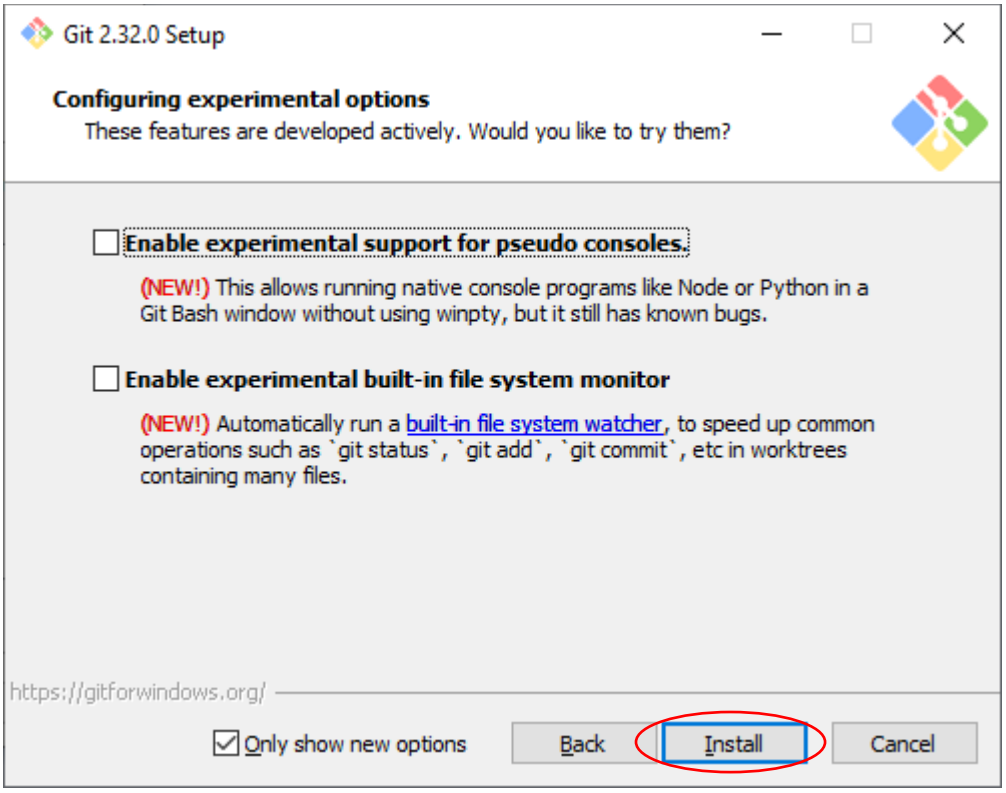


*Figura 64. Guía de instalación GIT - 3.*

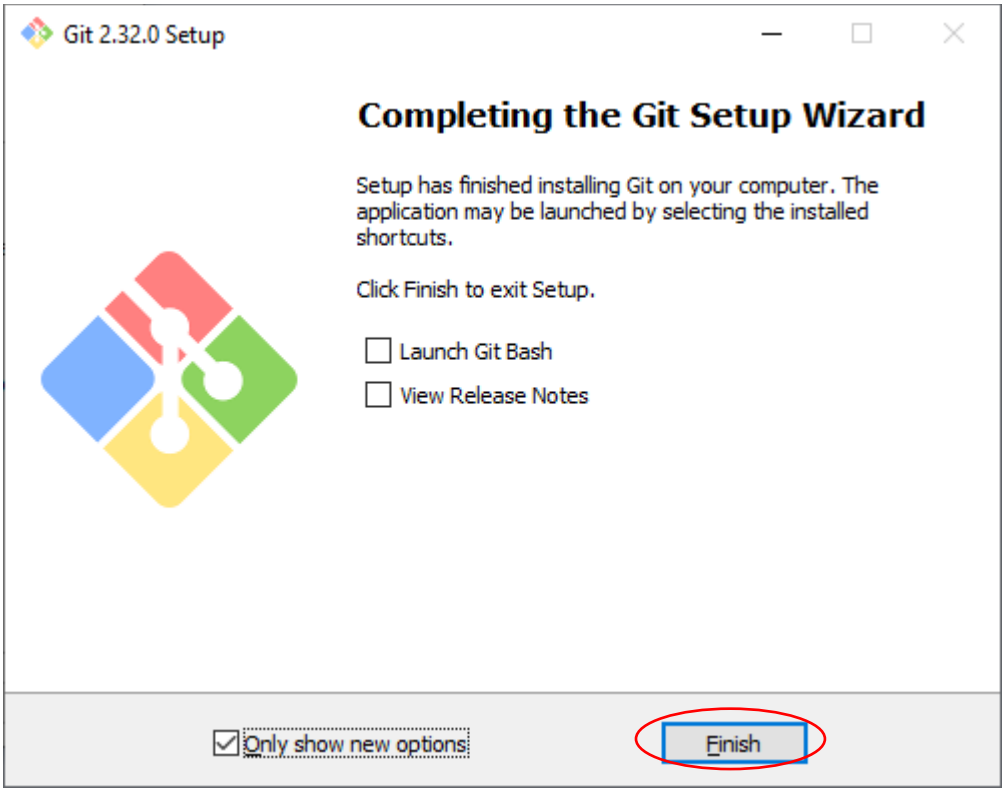


*Figura 65. Guía de instalación GIT - 4.*

#### *9.1.1.4.* Eclipse 2020-12 CDT o superior

Para finalizar con los requisitos previos, necesitamos *eclipse* como *software* de terceros para trabajar con nuestro módulo ESP32. En nuestro caso se ha usado la versión 2020-12, aunque recomendamos instalar la última disponible en la página *web* oficial.

En la Figura 66 observamos el enlace para acceder a la página de descarga del entorno de *eclipse*. Posteriormente en la Figura 67 podremos descargar el ejecutable en su versión más reciente.

Si queremos instalar otra versión, simplemente deberemos clicar en "*Download Package*", justo debajo del enlace para descargar la última versión, y seleccionar la deseada. Ver subrayado en Figura 67.


*Figura 66. Página web oficial de eclipse.*

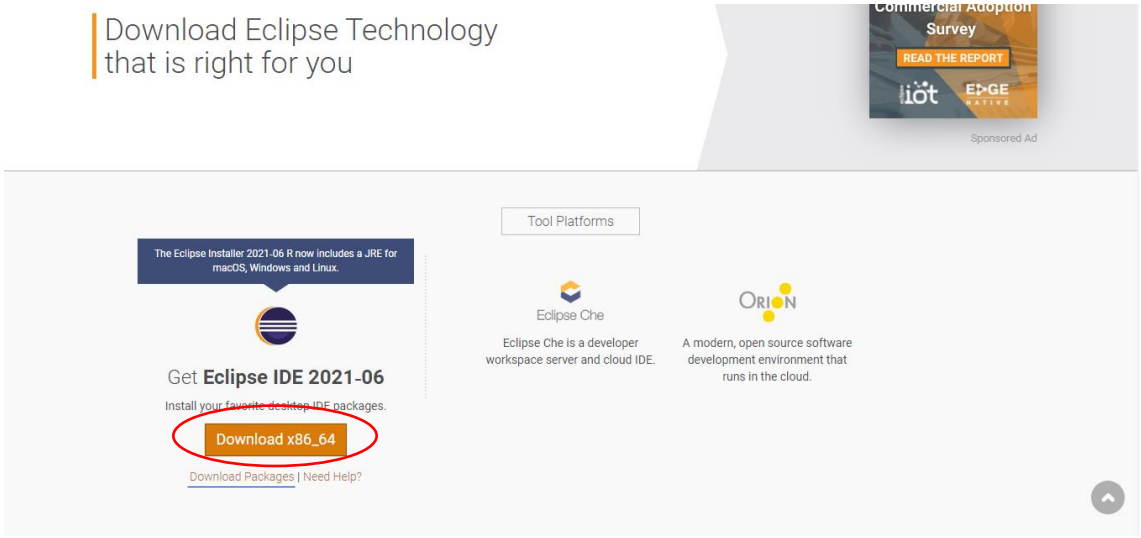


*Figura 67. Hipervínculo para acceder a la descarga de eclipse.*

A la hora de instalar el entorno deberemos tener en cuenta la herramienta que necesitamos de todas las que pone a nuestra disposición *eclipse*. En nuestro caso deberemos instalar el entorno para desarrolladores C/C++. La Figura 68 ilustra esto.

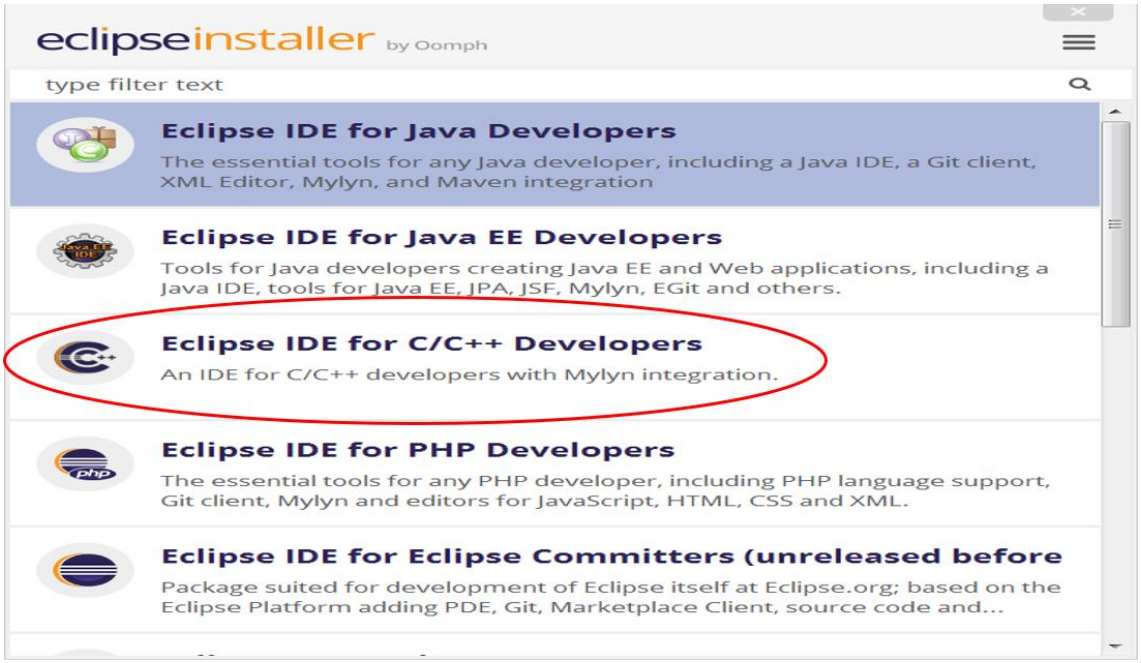


*Figura 68. Guía de instalación eclipse.*

### 9.1.2. Plug-in ESP-IDF en Eclipse

Una vez realizados todos los requisitos, solamente nos falta configurar nuestro entorno para poder trabajar. Deberemos instalar el complemento del *framework* oficial dentro de *eclipse*, para ello deberemos agregar la URL del repositorio oficial de versiones. En la Figura 69 podemos ver reflejada la ventana de instalación del *plug-in*.

1) *Help -> Install New Software*.
2) Hacer clic y en la ventana *Add*.
    a. *Name: Espressif IDF Plugin for Eclipse.*
    b. *Location: https://dl.espressif.com/dl/idf-eclipse-plugin/updates/latest/*
3) Clic en *Add*.
4) Seleccionamos todos los componentes Espressif IDF y hacemos clic en *Next*.

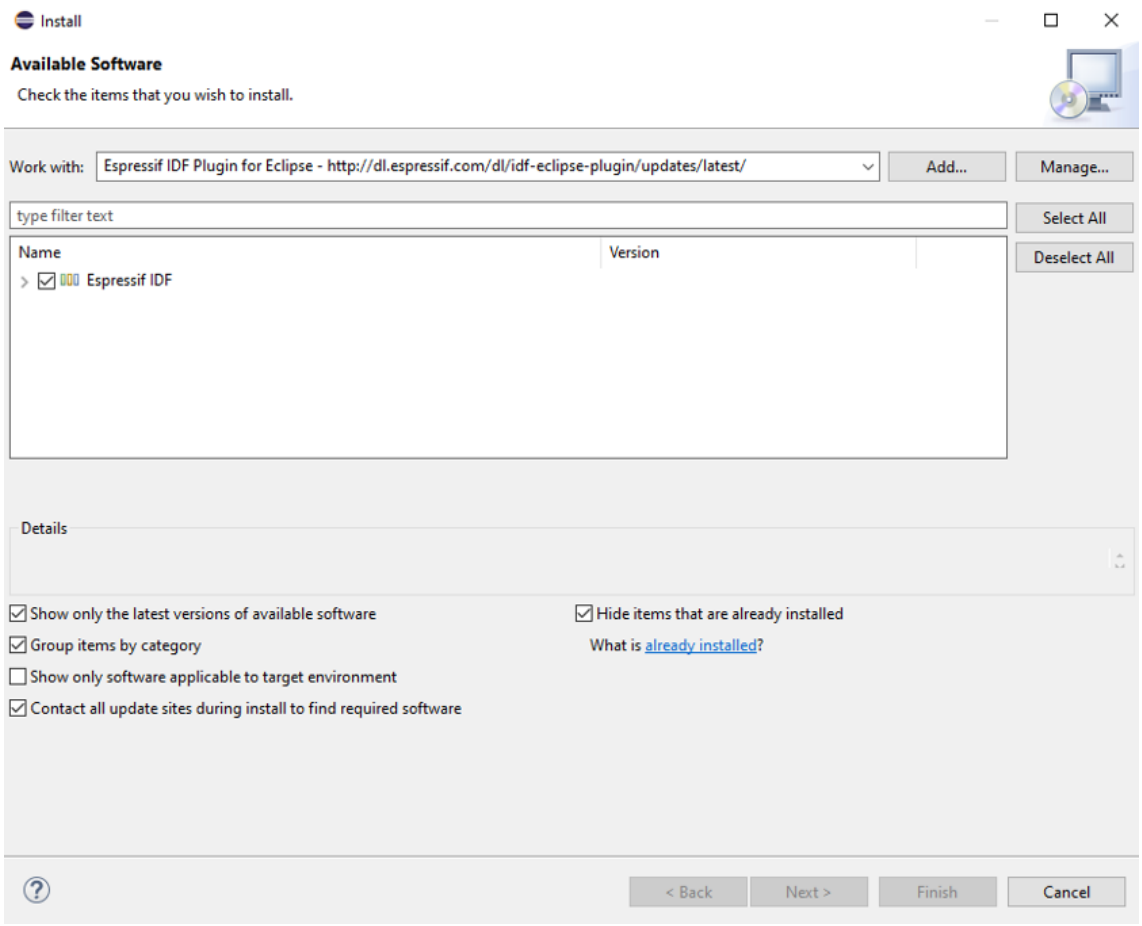


*Figura 69. Guía de instalación del plug-in.*

### 9.1.3. Repositorio ESP-IDF en Eclipse

A continuación, deberemos descargar e instalar el repositorio oficial del fabricante, lo que podemos hacer directamente desde *eclipse* siguiendo unos determinados pasos. En la Figura 70 podemos ver reflejada la ventana de instalación del repositorio.

1) *Help -> Download and Configure ESP-IDF*.
2) Seleccionamos la versión a descargar o hacemos uso de una descargada previamente y elegimos el directorio de trabajo donde se descargará e instalará (recomendamos descargar el repositorio en la misma carpeta de instalación de *eclipse*, ya que es dónde se instalarán los compiladores cruzados necesarios y, por tanto, serán más accesibles). Por defecto la ruta de instalación de *eclipse* será: C:\Users\Usuario
3) Clic en *Finish*.

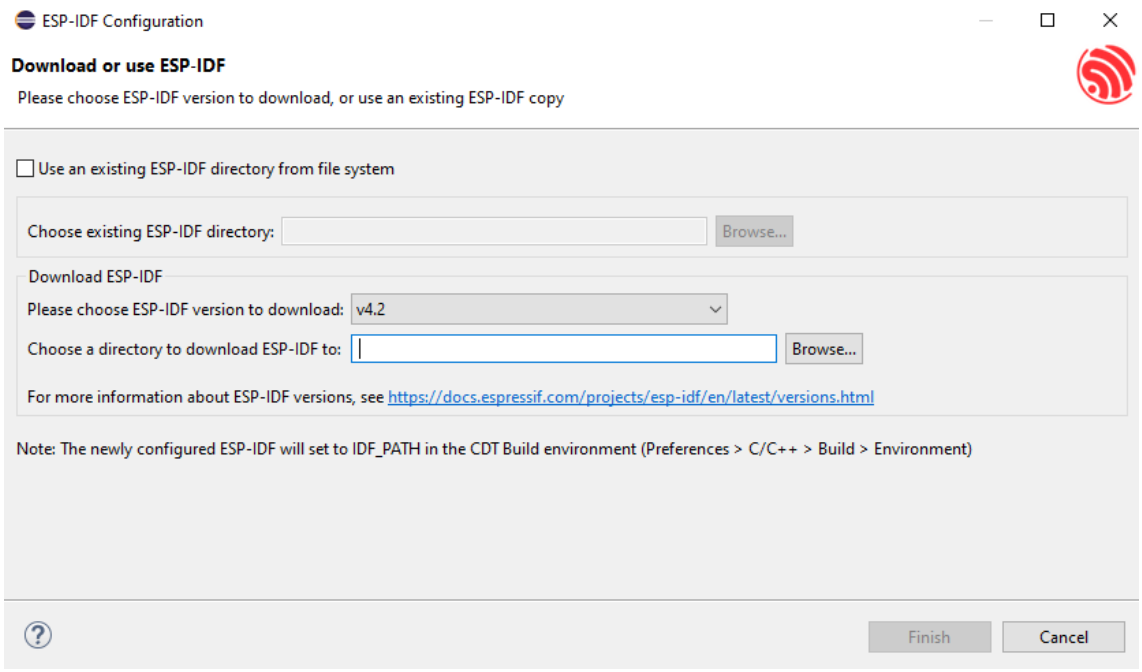


*Figura 70. Guía de instalación del repositorio.*

### 9.1.4. Herramientas ESP-IDF en Eclipse

Para finalizar la configuración, debemos instalar algunas herramientas de los requisitos previos explicados anteriormente. La función de dichas herramientas sirve para construir el *firmware* para el módulo ESP32.

Entre ellas, se incluyen Python, GIT, compiladores cruzados, herramienta *menuconfig* y las herramientas de compilación CMake y Ninja. En la Figura 71 podemos ver reflejada la ventana correspondiente a la instalación de las herramientas necesarias.

1) *Help -> ESP-IDF Tools Manager -> Install Tools.*
2) Introducir la ruta donde hemos descargado el repositorio ESP-IDF.
3) Introducir la ruta de los ejecutables de GIT y Python (si no se detectan automáticamente).
4) Clic en *Install Tools*.

La instalación puede demorar varios minutos, debido a que se tienen que descargar e instalar todas las herramientas de compilación.

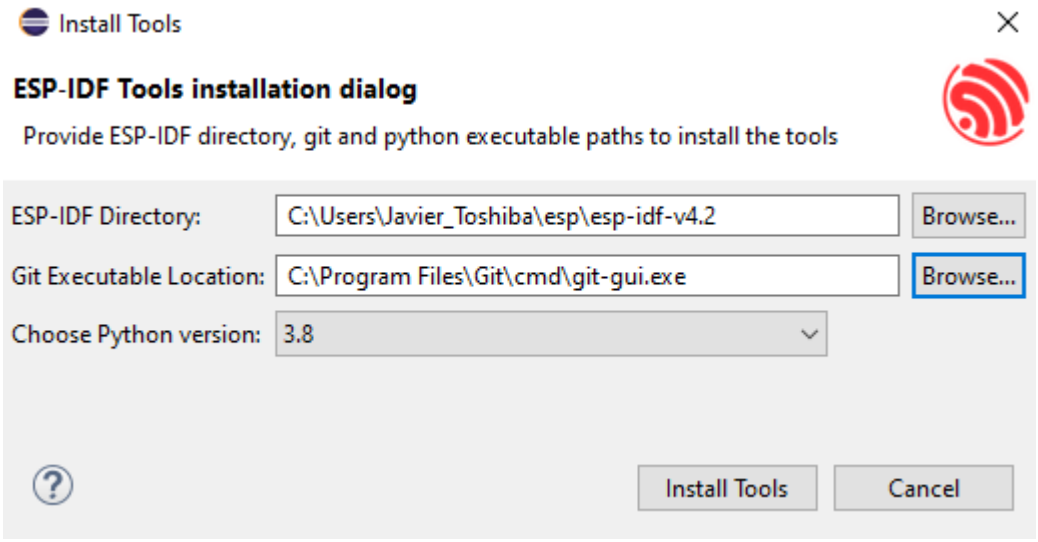


*Figura 71.Guía de instalación de las herramientas.*

## 9.2. Manual de usuario

El uso de *eclipse* es un proceso sencillo e intuitivo, deberemos tener una serie de aspectos en cuenta a la hora de crear un nuevo proyecto o como compilarlo y descargarlo en la propia placa.

Tiene muchas funcionalidades, entre ellas destaca un terminal serie propio, así como una consola que nos indica el estado de cualquier proceso de compilación o depuración que estemos realizando en cada instante.

Lo primero será crear un nuevo proyecto para nuestro módulo ESP32. Para ello, deberemos acceder a *File ->New -> Espressif IDF Project.* Si no aparece dicha opción en el menú desplegable, deberemos acceder a *File -> New -> Others -> Espressif IDF Project.* La Figura 73 y la Figura 74 ilustran el procedimiento descrito.

La familia de Espressif Systems pone a nuestra disposición una gran cantidad de plantillas (*templates*) de diversos proyectos, en los cuales nos enseñan como configurar los distintos periféricos que posee. Esto es una gran ventaja, puesto que podemos usarlas para nuestro propio proyecto, además de descargarlas en nuestra placa para comprender mejor el repositorio y el propio funcionamiento de esta misma.

Cuando creemos un nuevo proyecto, tenemos la opción de crearlo sobre una de estas plantillas. En la Figura 74, podemos observar en la parte superior de la imagen, subrayado en azul, la opción para usar una de las plantillas.

A continuación, debemos darle un nombre al proyecto, además de elegir una ubicación. Es muy importante tener en cuenta dónde vamos a alojar nuestro proyecto, como hemos comentado anteriormente, las herramientas del ESP-IDF incluyen los compiladores cruzados que se usaran. Por tanto, para favorecer la productividad de la herramienta, recomendamos que utilicemos como directorio de trabajo de *eclipse* (ruta dónde almacenaremos nuestro proyecto) la misma ruta donde se instalaron los compiladores.

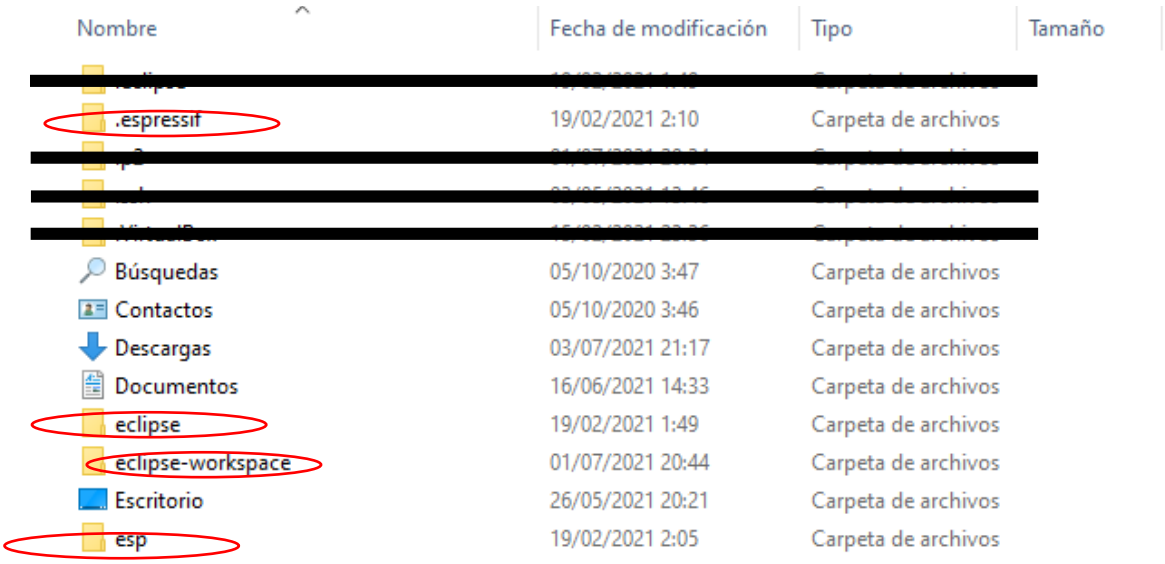


*Figura 72. Rutas del programa.*

En la Figura 72 podemos observar cómo deberíamos tener estructurado nuestro entorno, junto con el repositorio y las herramientas. La carpeta ".espressif" contiene todas las herramientas que hemos descargado e instalado en la sección 9.1.4. La carpeta "eclipse" alberga el entorno de *eclipse*, mientas que "eclipse-workspace" y "esp" serían los directorios de trabajo y del repositorio, respectivamente.

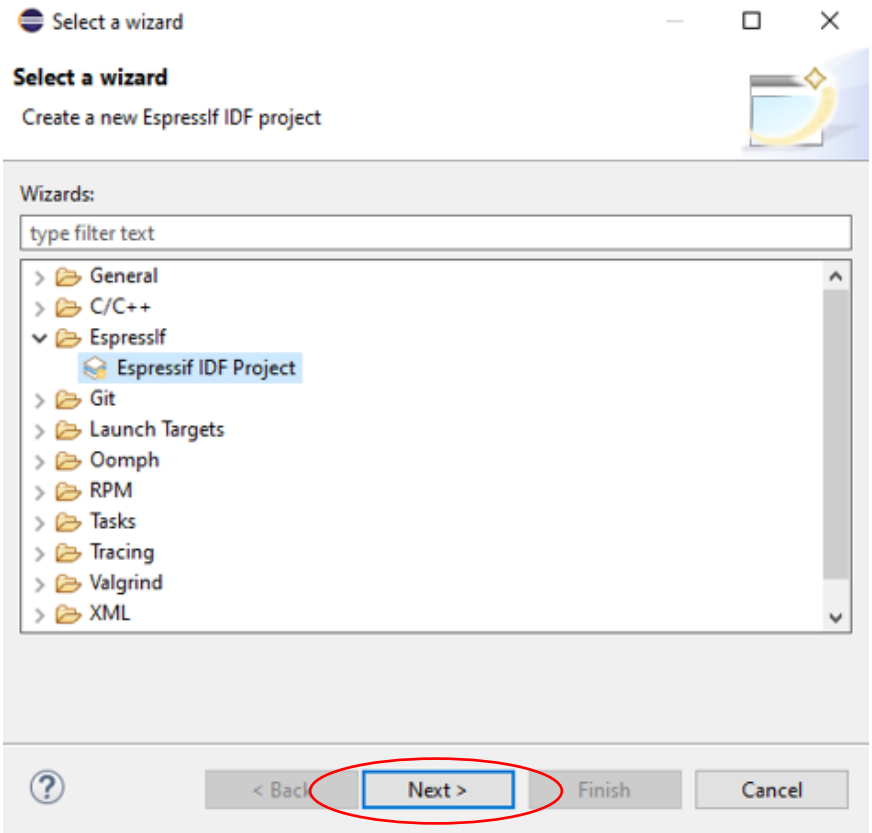


*Figura 73. Guía de creación nuevo proyecto - Creación.*

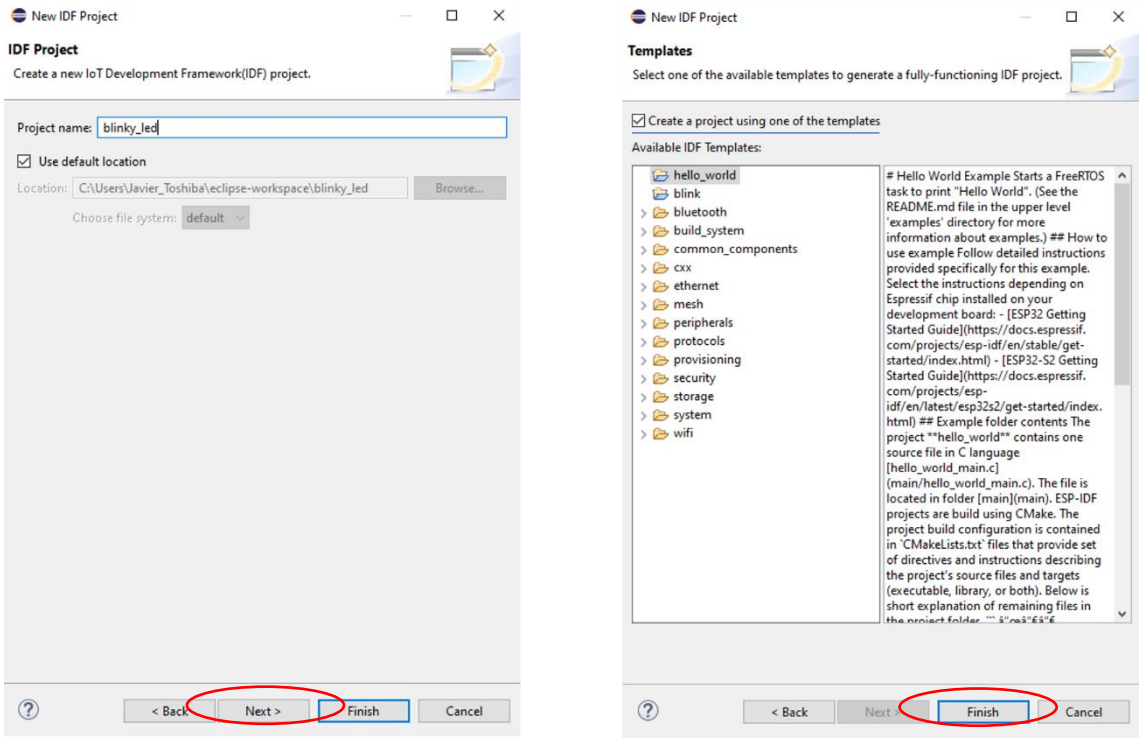


*Figura 74. Guía de creación nuevo proyecto - Creación - 1.*

Una vez creado nuestro proyecto, pasaremos a compilarlo y flashearlo en la placa. Lo primero que debemos hacer es indicarle al entorno que tipo de placa vamos a usar. En nuestro caso, como hemos instalado el *plug-in* oficial de ESP-IDF y es un proyecto de la familia de Espressif Systems como podemos ver en la Figura 73, solamente tendremos que crear una nueva.

Para ello, accederemos al menú desplegable *New Launch Target* y seleccionaremos la opción de *ESP Target*. Una vez seleccionada, podremos darle un nombre en concreto y seleccionar si el modelo de placa es un ESP32 o ESP32s2, además del puerto serie que se ha asignado a la placa (aunque el puerto serie se rellena automáticamente, recomendamos su comprobación en el administrador de dispositivos de su equipo). En la Figura 75 observamos el proceso descrito.

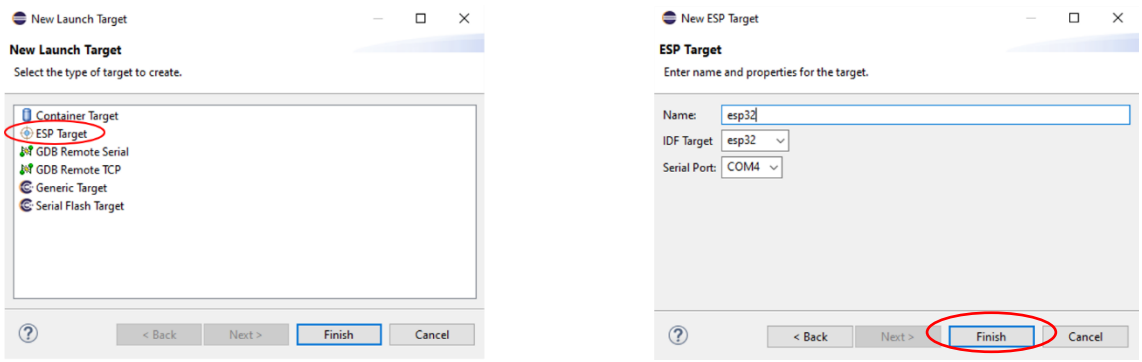


*Figura 75. Guía de creación nuevo proyecto – Configurar placa.*

El proceso en sí de compilación es muy sencillo, bastaría con hacer clic en el icono del martillo que podemos encontrar en la esquina superior izquierda del entorno, para posteriormente descargarlo en placa a través del botón que se encuentra a la derecha del primero mencionado.

Es importante ir chequeando la consola para comprobar todo el proceso que se realiza durante la compilación, así como los posibles errores que hemos cometido en la generación del código fuente de nuestra aplicación. También debemos comprobarlo cuando descargamos en placa.

El icono de compilación los podemos ver reflejados en la Figura 76 y en la Figura 77 observamos la consola durante el proceso de compilación del código fuente. De la misma manera observamos en la Figura 78 el icono para descargar en placa y en la Figura 79 la consola durante este proceso.

Si todo el proceso se ha realizado correctamente y el código fuente no tiene errores, deberemos obtener los mensajes visibles en la Figura 77 y en la Figura 79 indicando que se han realizado con éxito ambos procesos.

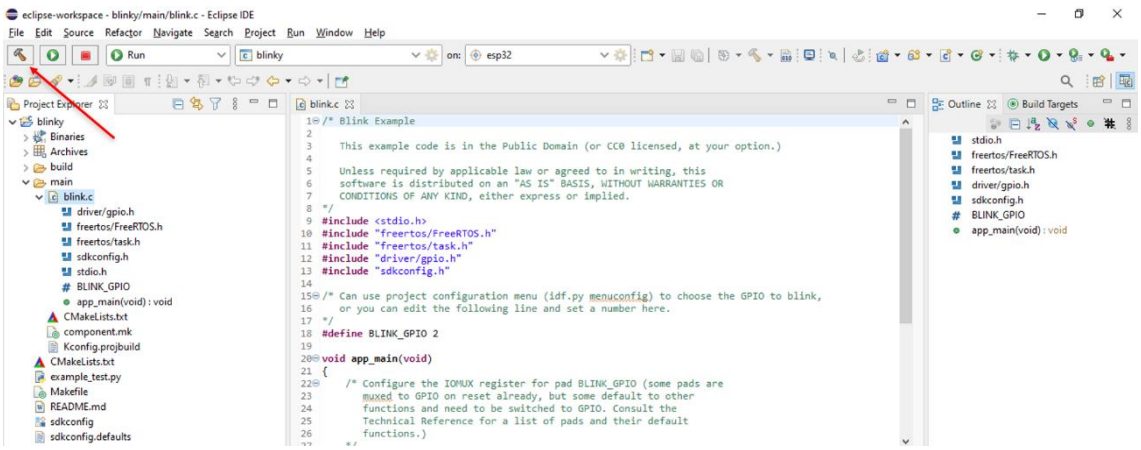


*Figura 76. Guía de creación nuevo proyecto - Compilación.*

```
Building in: C:\Users\Javier_Toshiba\eclipse-workspace\blinky\build
cmake --build . -- -v
[1/3] cmd.exe /C "cd /D C:\Users\Javier_Toshiba\eclipse-workspace\blinky\build\bootloader && C:\Users\Javier_Toshiba\.espressif\tools\cmake\3.1
ninja: no work to do.
Build complete (0 errors, 0 warnings): C:\Users\Javier_Toshiba\eclipse-workspace\blinky\build
```

*Figura 77. Guía de creación nuevo proyecto – Consola durante compilación.*

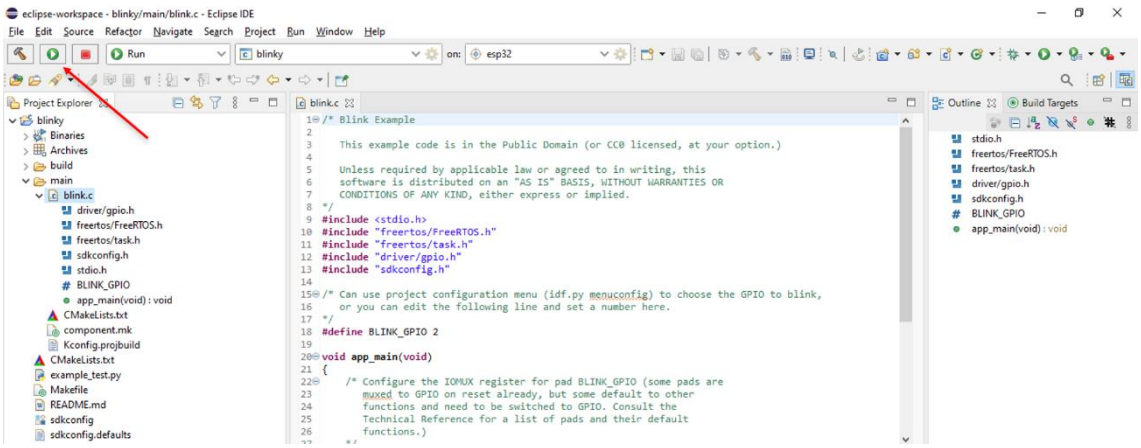


*Figura 78. Guía de creación nuevo proyecto – Descarga en placa.*

```
Hard resetting via RTS pin...
Executing action: flash
Running ninja in directory c:\users\javier_toshiba\eclipse-workspace\blinky\build
Executing "ninja flash"...
Done
```

*Figura 79. Guía de creación nuevo proyecto – Consola durante descarga en placa.*

Por último, comprobaríamos el funcionamiento de la aplicación en nuestra placa. Otra manera que tenemos de comprobar el funcionamiento y ver posibles errores en nuestro proyecto es a partir del terminal serie del entorno.

Para abrir dicho terminal, deberemos hacer clic en el icono de la pantalla que se encuentra situado en la parte central derecha del entorno. Solamente deberemos indicarle el puerto serie, que previamente se ha definido en la Figura 75, el *baudrate* y elegir la opción de "*ESP-IDF Serial Monitor*", que es el monitor que pone a nuestra disposición el *plug-in* del fabricante.

La Figura 80 muestra el icono del terminal serie del entorno, posteriormente en la Figura 81 lo configuramos con los parámetros descritos y en la Figura 82 observamos la salida de un proyecto a través del terminal serie.


*Figura 80. Guía de creación nuevo proyecto – Terminal serie.*

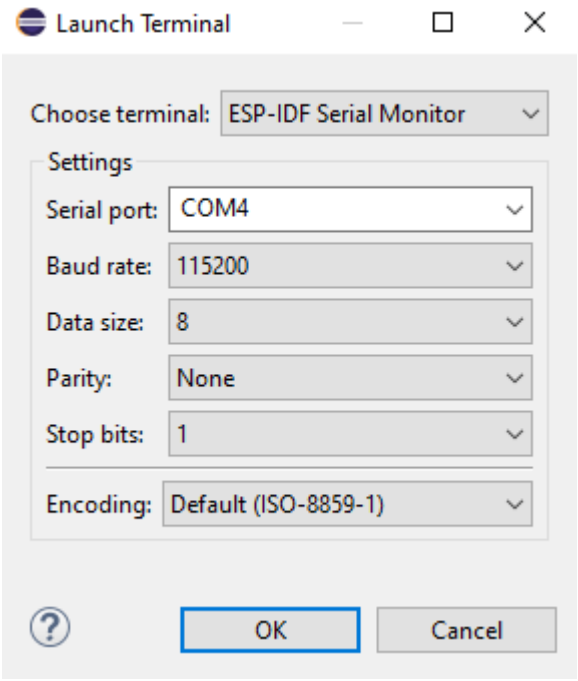


*Figura 81. Guía de creación nuevo proyecto – Configuración del terminal.*

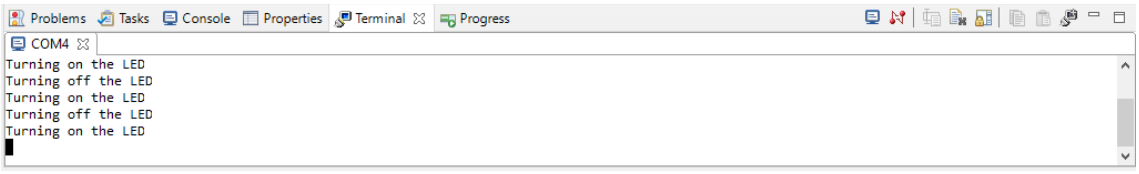


*Figura 82. Guía de creación nuevo proyecto – Terminal serie del proyecto.*

El *plug-in* de ESP_IDF en *eclipse* nos permite modificar una gran cantidad de aspectos de nuestro microcontrolador. Dentro del entorno podemos observar un fichero de configuración con extensión sdk, el cual accediendo a él podremos observar las características que se pueden modificar.

Dicho archivo le tenemos accesible en el espacio de trabajo, dentro de la carpeta del proyecto en el que nos encontremos. La Figura 83 ilustra el archivo y la ubicación de este.

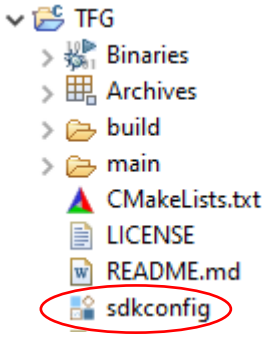


*Figura 83. Ruta del fichero de configuración.*

Una vez accedemos al fichero se nos abrirá una nueva ventana dentro del entorno, podemos observar el aspecto en la Figura 84. Dentro de esta tenemos un menú desplegable en el lateral izquierdo dónde se nos muestran todos los aspectos que podemos configurar.

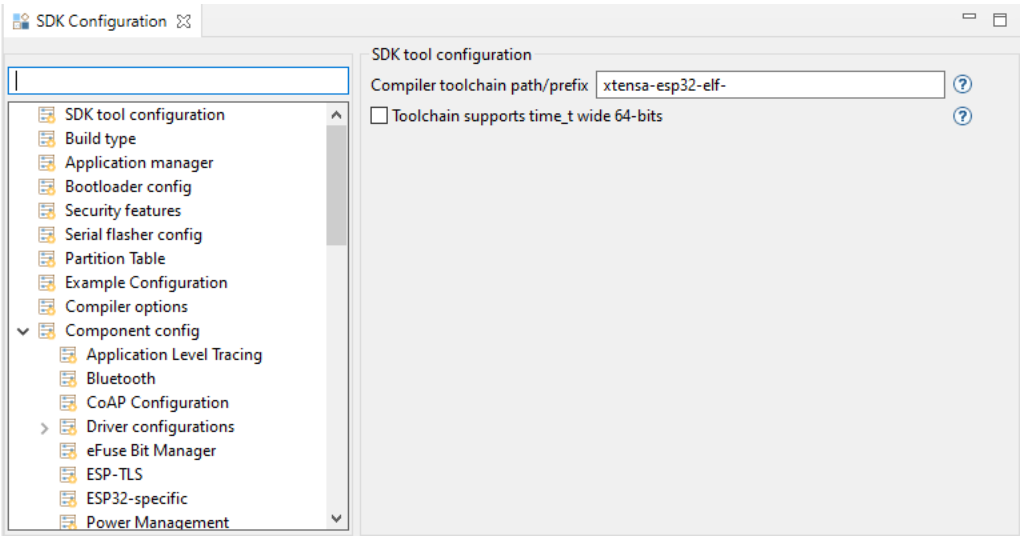


*Figura 84. Ventana del archivo de configuración.*

Gracias a que se nos permite modificar diversas configuraciones podemos adaptar nuestro microcontrolador a las necesidades que requiera el proyecto, como por ejemplo configurando para que el FreeRTOS se ejecute solamente en uno de los dos *cores* o modificar el *tick rate* (Figura 85), configurar un determinado tiempo de *watchdog* tanto para interrupciones como para tareas o incluso deshabilitarlo (Figura 86), configurar si queremos un *buffer* de transmisión dinámico o estático cuando usemos el Wi-Fi (Figura 87), etc.

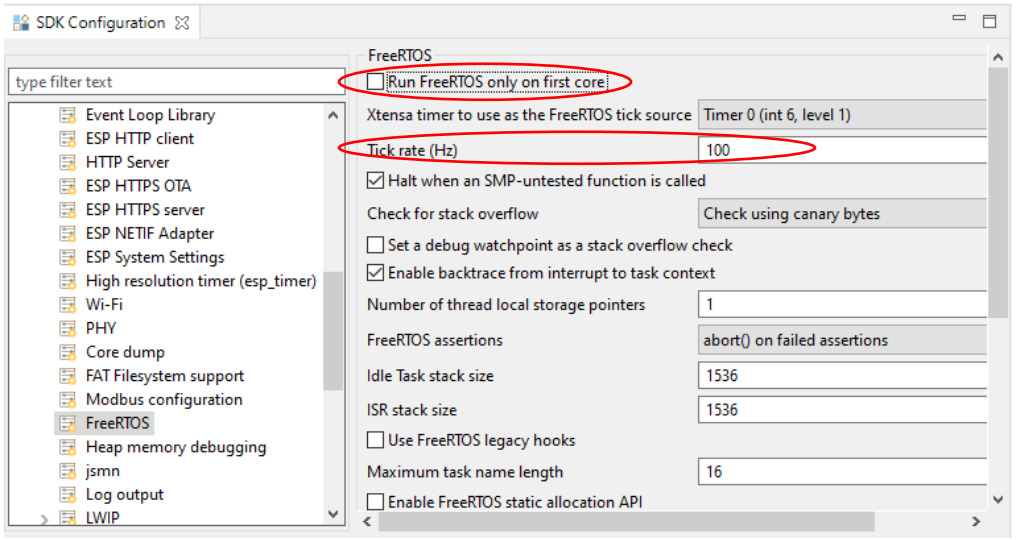


*Figura 85. Ventana del archivo de configuración – FreeRTOS.*

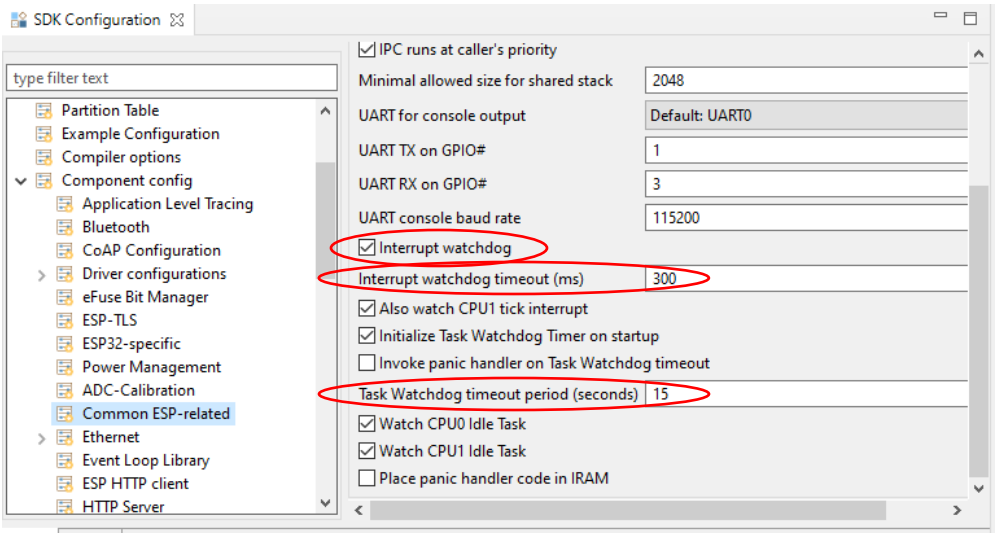


*Figura 86. Ventana del archivo de configuración – WDT.*

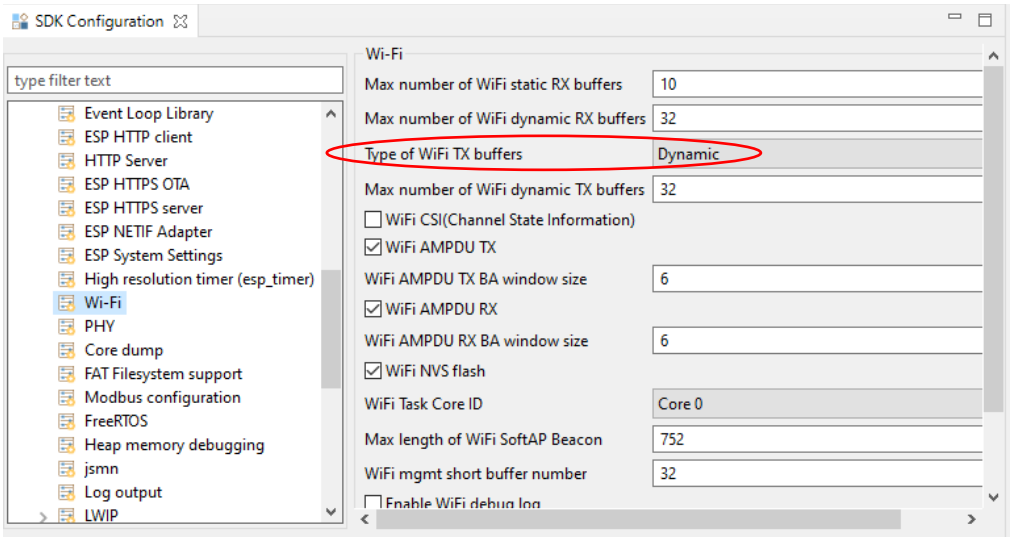


*Figura 87. Ventana del archivo de configuración – Wi-Fi.*

Es importante conocer qué estamos modificando realmente al sobrescribir la información, puesto que si modificamos algo fuera de los valores permitidos por el microcontrolador nuestro proyecto no funcionará y desconoceremos el motivo, ya que nuestra primera idea es que el error se encontrará en el código.

Por este motivo es muy recomendable que antes de modificar aspectos importantes, acudamos al manual de referencia [5] para conocer realmente que queremos modificar y cómo hacerlo. En internet también podemos encontrar una gran cantidad de material donde se nos explica toda esta información.

Otro aspecto que es importante configurar es la memoria flash de nuestro dispositivo, indicándole al entorno cual es la capacidad de dicha memoria en nuestro microcontrolador. Para ello lo configuramos dentro del mismo archivo de configuración como podemos ver en la Figura 88.

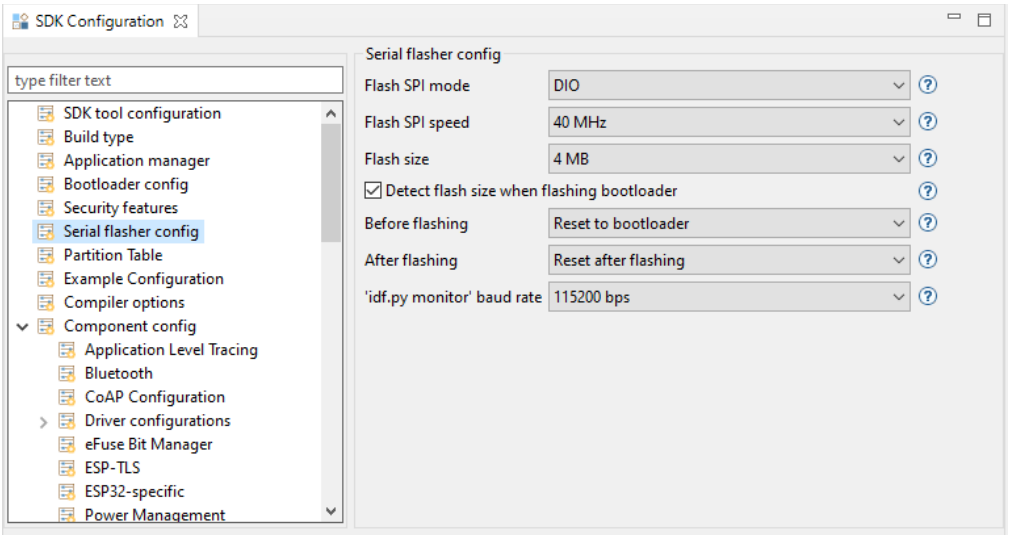


*Figura 88. Ventana del archivo de configuración – Serial flasher.*

Una vez que tengamos el fichero configurado y lo guardemos, a la hora de compilar se generará una librería de del cual el compilador obtendrá los valores que se han configurado para determinadas características.

El fichero podemos encontrarlo dentro de la carpeta "build", visible en la Figura 83, en la ruta *build->bootloader->config->sdkconfig.h*. En la Figura 89 podemos observar la librería.

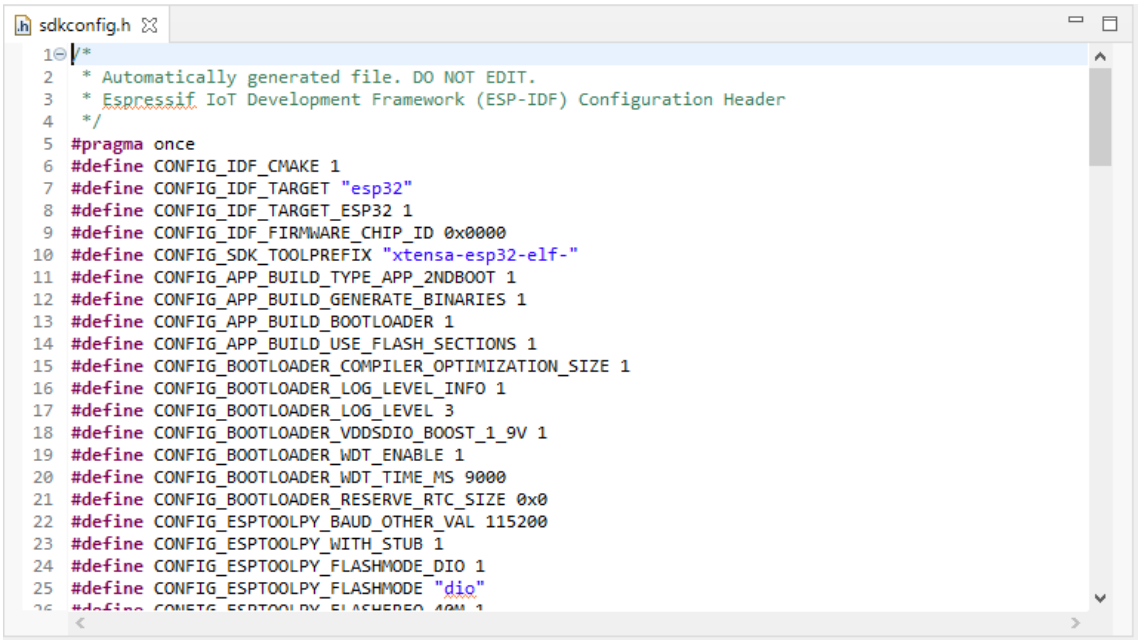


*Figura 89. Librería generada sdkconfig.*

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá