Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Autor: Diego Sanz Martín

Tutor: Óscar García Población

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Autor: Diego Sanz Martín

Tutor: Óscar García Población

Tribunal:

Presidente: Concepción Batanero Ochaíta

Vocal 1º: María del Mar Lendinez Chica

Vocal 2º: Óscar García Población

Fecha de depósito: 12 de septiembre de 2022

"Con el fin de hacer más progresos, en particular en el campo de los rayos cósmicos, será necesario aplicar todos nuestros recursos y aparatos simultáneamente y de lado a lado; un esfuerzo que aún no se ha hecho, o al menos, a un sólo medida limitada."

Victor Francis Hess

Agradecimientos

Mi ingreso en el Grado de Telecomunicaciones fue del todo inesperado y precipitado para mi entorno. Yo, desde pequeño, disfruto de la música y tenía mi vida dedicada y encaminada a ser músico. A la edad de dieciocho años me gradué del Conservatorio Profesional Luis de Victoria de Ávila, presentándose una difícil decisión, pero que a día de hoy sigo considerando acertada. En la carrera, para mi sorpresa, encontré una pasión comparable a la que siento por la música y que me ha impulsado en estos cuatro años.

A mi familia, sobre todo a mis padres y hermana, gracias por el amor y el apoyo incondicional que siempre me dais. Por educarme y por convertirme en la persona que soy hoy en día, ya que sin vuestra ayuda no habría conseguido todo lo que me he propuesto. A mi pareja, que me ha acompañado durante estos cuatro años y con la que he trabajado codo con codo. Que me quiere y me demuestra que la vida es un poco más colorida a su lado.

También debo dar las gracias a mis compañeros y amigos las innumerables veces que me han ayudado. Por las tardes de estudio, que acababan siendo solo de risas, y por estar ahí siempre que os he necesitado.

Por último, una mención especial merece mi tutor Óscar García Población. Gracias por ofrecerme este proyecto tan fascinante. Por tu implicación y seguimiento constante, que ha hecho que este trabajo tenga el detalle y el rigor que se merece.

A todos gracias.

Resumen

El objetivo de este proyecto es construir un sistema de adquisición de datos para un monitor de neutrones basado en la plataforma ESP32 y la tecnología IoT. El sistema está compuesto por una estación base y una serie de nodos sensores distribuidos en el área que debemos monitorizar. Los nodos sensores están equipados con detectores de neutrones y transmiten los datos recopilados a la estación base a través de tecnología de comunicación inalámbrica. La estación base procesa los datos recibidos y los visualiza en tiempo real en una interfaz web. El sistema de adquisición diseñado en este documento, es la pasarela de comunicación entre los nodos sensores y la estación base. Los datos recopilados se almacenan en una base de datos, para que posteriormente puedan ser transferidos a la comunidad científica.

Palabras clave: IoT, neutrón, monitor, rayos cósmicos, ESP32, Espressif, InfluxDB, Grafana, Mosquitto, Google Cloud, Github Actions, Kicad 6.

Abstract

The objective of this project is to build a data acquisition system for a neutron monitor based on the ESP32 platform and IoT technology. The system is composed of a base station and a series of sensor nodes distributed in the area to be monitored. The sensor nodes are equipped with neutron detectors and transmit the data collected to the base station through wireless communication technology. The base station processes the received data and displays it in real time on a web interface. The acquisition system designed in this document is the communication gateway between the sensor nodes and the base station. The collected data is stored in a database, so that it can later be transferred to the scientific community.

Keywords: IoT, neutron, monitor, cosmic rays, ESP32, Espressif, InfluxDB, Grafana, Mosquitto, Google Cloud, Github Actions, Kicad 6.

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde la pasada década se ha experimentado un rápido desarrollo del ámbito de la Física, más concretamente en la investigación de la radiación ionizante de la atmósfera y su relación con los rayos cósmicos. Por lo tanto, el propósito de este capítulo se ha centrado en estudiar su origen, composición y las interacciones que producen al ingresar en la atmósfera, así como los distintos métodos que se emplean para su detección. Finalmente, hemos definido los requisitos de nuestro sistema enumerando los bloques que lo componen y sus interconexiones.

1.1. Contexto del trabajo

Este trabajo de fin de grado se ha realizado gracias a la colaboración con el grupo de investigación espacial **SRG**[1] de la universidad de Alcalá, que ha puesto a mi disposición el equipo y los conocimientos necesarios para mi formación.

Este equipo, integrado por mi tutor Óscar García Población, ha desarrollado importantes proyectos en el área de detección de rayos cósmicos y ha ubicado un nodo de observación en Guadalajara (**CaLMa**[2][3, 4]) y otro en la Isla Livingston, Antártica (**ORCA**[5, 6]), además de otro nodo móvil (**MiniCaLMa**[7]).

Esta colaboración ha permitido el desarrollo de numerosos artículos[8], publicaciones, tesis doctorales[9], trabajos de fin de carrera y contratos de investigación. Por lo tanto estoy agradecido de que mi trabajo pueda contribuir en el desarrollo de los proyectos existentes y futuros.

1.2. Presentación

La primera década del siglo XX fue un periodo revolucionario para la ciencia, en este periodo se realizaron grandes descubrimientos en muchas áreas. Uno de los más importantes, por su impacto en la física de partículas, fue el descubrimiento de los rayos cósmicos en 1912 por Victor Hess.

Inicialmente existía la convicción de que la radiación presente en la atmósfera aumentaba a medida que la distancia a la Tierra disminuía, puesto que emanaba de los elementos radioactivos naturales presentes en la corteza terrestre. Victor Hess basó su trabajo en la medición de los niveles de radiación ionizante. Sus experimentos resultaron ser muy reveladores ya que observó que en la atmósfera el nivel de radiación podría ser, en realidad, más alto que en la superficie.

En consecuencia, Hess determinó que la alta radiación que penetra en la atmósfera es procedente del espacio exterior. Años después Robert Andrews Millikan confirmó su descubrimiento nombrando a dicha radiación como rayos cósmicos.

1.3. The Neutron Monitor Database

El origen de los rayos cósmicos[10] es aún un objeto de debate en la comunidad científica, su composición trata principalmente de protones procedentes del Sol, aunque una pequeña proporción son partículas alfa de alta energía con un origen extragaláctico.

Estos rayos cósmicos de alta energía son particularmente interesantes para la astrofísica, ya que pueden proporcionar información sobre objetos y fenómenos extragalácticos. Dentro de este marco, surgió la necesidad de medir la cantidad de rayos detectados por los monitores geográficamente distribuidos, creándose así "The Neutron Monitor Database"(NMDB).

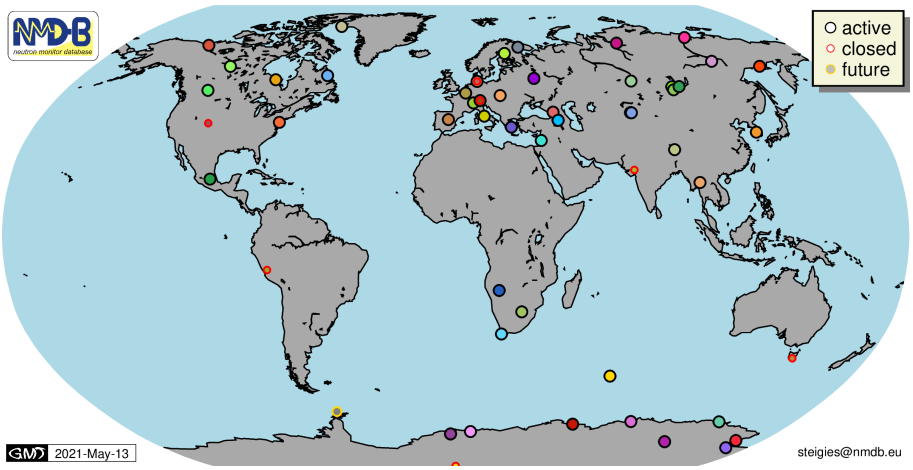


Figura 1.A: Red de estaciones NMDB

En el mapamundi de la figura 1.A observamos que las localizaciones de los detectores varían en función de los trópicos, siendo los más interesantes los ubicados en los círculos polares y el ecuador. Esto se debe a que en estas zonas la densidad atmosférica permite que más o menos rayos cósmicos viajen a través de la atmósfera sin ser absorbidos por ella como causa del campo gravitatorio terrestre.

Figura 1.A obtenida de NMDB[11]

1.4. Detección de rayos cósmicos

La detección de rayos cósmicos es una técnica utilizada para estudiar los objetos más energéticos del universo. Algunas de estas partículas pueden ser hasta millones de veces más energéticas que las generadas en el colisionador de partículas LHC. En la Tierra, la llegada de rayos cósmicos produce interacciones con la atmósfera, formando una lluvia de partículas que son detectables en la troposfera.

Existen dos métodos principales en función de la altura de la detección:

- **Detección directa**: Se produce en el espacio o en la atmósfera a gran altitud, mediante instrumentos instalados en satélites o globos aerostáticos de gran altitud.
- **Detección indirecta**: Consiste en la localización de partículas secundarias, producidas cuando los rayos cósmicos colisionan con los núcleos de nitrógeno y oxígeno del aire, originando las cascadas atmosféricas.

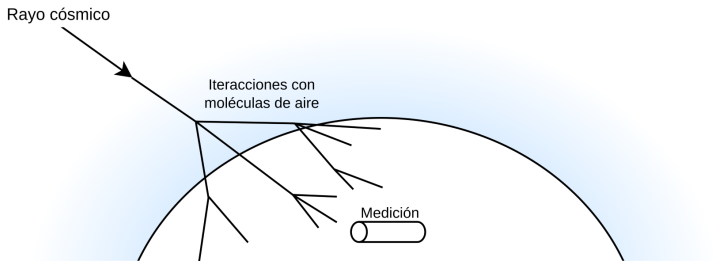


Figura 1.B: Detección de una cascada atmosférica

En la actualidad, se utilizan instrumentos de gran tamaño y complejidad para detectar estas partículas y estudiar sus propiedades, aunque existen otros más simples y portables a efectos de reducir sus prestaciones. Los detectores de partículas altamente energéticas, también pueden ser englobados en dos grandes familias de acuerdo con el método utilizado para detectar la radiación: detectores pasivos y detectores activos.

Los detectores pasivos, a diferencia de los activos, no necesitan una fuente de alimentación para detectar los neutrones, si no que presentan desventajas que dificultan un estudio eficaz de los rayos cósmicos. En primer lugar, la obtención de los datos se debe realizar mediante un microscopio óptico, de manera que toma importancia la influencia del factor humano y, en consecuencia, el error producido en sus medidas. Además, no guardan información sobre el instante de los impactos y el estudio solo se puede realizar en periodos de tiempo reducidos, impidiendo el análisis detallado del flujo de partículas.

Por este motivo se desarrollaron los detectores activos, que son los más utilizados actualmente; requieren una fuente de energía y solo registran partículas cuando están a la tensión adecuada. Una de las mayores ventajas que debemos considerar es la rapidez de detección y la continuidad en la recopilación de los datos

Existen dos tipos de modos básicos de operación: modo corriente y modo pulso. En el modo corriente se realiza la media de la corriente que se genera en un periodo de tiempo determinado y el modo pulso mide cada pulso de la señal de salida de la corriente para almacenarla. Entre los detectores activos más empleados encontramos los detectores de gas, detectores de centelleo y detectores semiconductores.

1.4.1. Detectores de gas

Normalmente estos detectores[12] están formados por tubos cilíndricos rellenos de un gas y un hilo conductor para crear en su interior dos electrodos. Cuando el detector recibe partículas incidentes que ionizan el gas a medida que lo atraviesa, se producen electrones y cargas positivas que se atraen y se repelen, formando así una diferencia de potencial que puede ser medida. La electrónica empleada en este proceso se observa de forma esquematizada en la figura 1.C.

Es fundamental destacar la importancia de las medidas del detector y su área de captura, ya que se necesita tener unas dimensiones lo suficientemente grandes para recolectar la mayor cantidad de ionización, pero lo suficientemente pequeñas para no perder precisión. Además del modelo de detector mencionado, existen detectores esféricos que son más compactos aunque más difíciles de mantener y detectores cuadrados o rectangulares que son más fáciles de fabricar, pero poco eficaces.

En nuestro caso, los monitores de neutrones con los que vamos a trabajar pertenecen a la familia de los detectores de gas, utilizando BF_3 como principio activo.

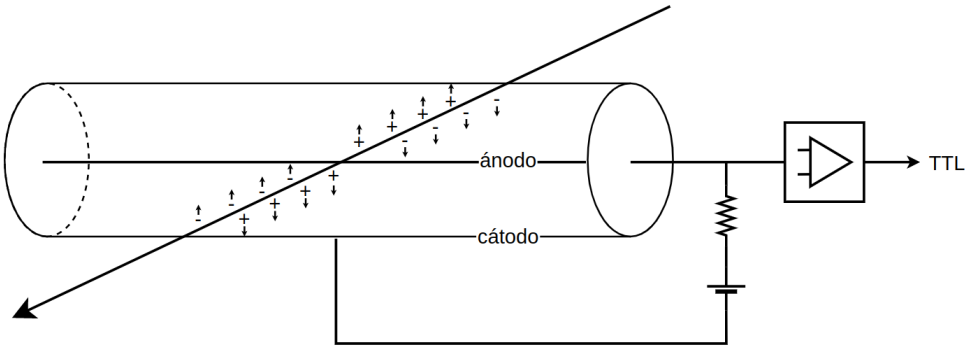


Figura 1.C: Esquema de un detector de gas

En la atmósfera existe una gran cantidad de radiación ionizante, por lo que si exponemos el detector de forma directa, no seremos capaces de diferenciar la radiación producida por los rayos cósmicos. Como podemos observar en la figura 1.D, obtenida de la Agencia para Sustancias Tóxicas y el Registro de Enfermedades (**ATSDR**[13]), tan solo un 8 % de la radiación que se encuentra en la atmósfera tiene un origen cósmico.

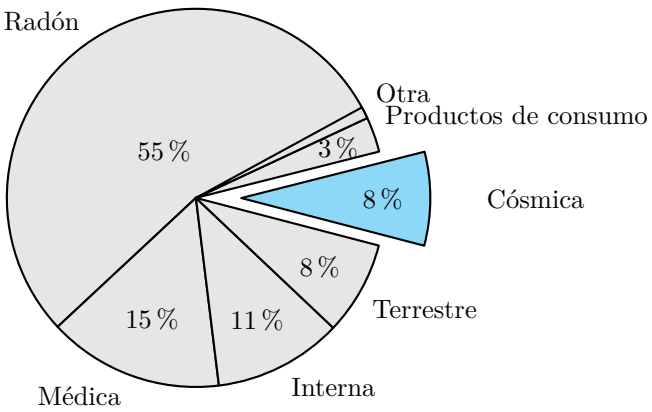


Figura 1.D: Estudio del ATSDR de la exposición a la radiación ionizante

Es por esta razón que el detector de neutrones se complementa con un esqueleto compuesto por distintos materiales que ayudan en el filtrado e incrementan la detección.

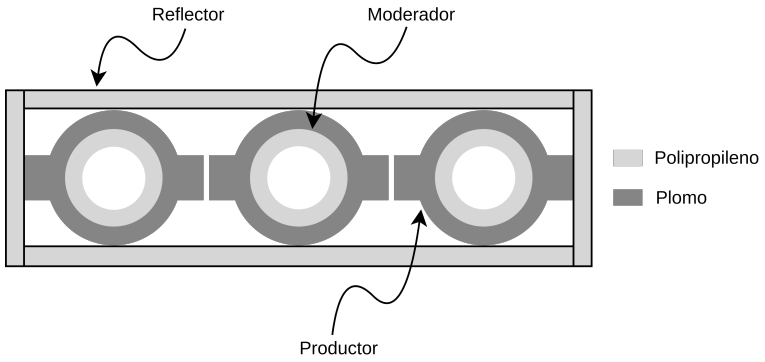


Figura 1.E: Esqueleto de un detector gaseoso de neutrones

Las partes que forma el armazón se pueden diferenciar en:

- **Reflector**: Esta capa está formada por Polipropileno, un material que actúa como barrera para los neutrones ambientales de baja energía. De esta forma se consigue filtrar eficazmente todas las partículas no procedentes de los rayos cósmicos.
- **Productor**: Formado por Plomo y como su nombre indica, es el encargado de producir en torno a 10 neutrones de energía más baja por cada neutrón de alta energía.
- **Moderador**: Compuesto por el mismo material rico en protones que el Reflector, su función es ralentizar los neutrones filtrados y generados en las dos etapas anteriores. De esta forma se incrementa la probabilidad de detección del monitor.

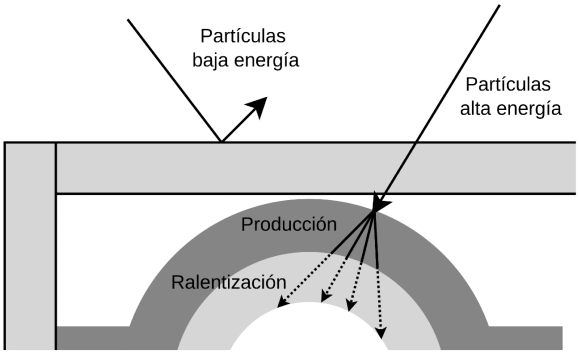


Figura 1.F: Funcionamiento del armazón de un monitor de neutrones

1.4.2. Monitor de neutrones: Chalk-River BP28

Actualmente hay dos tipos de monitores estandarizados (**IGY** y **NM64**[14]) en el funcionamiento de la red NMDB, mostrada en el mapamundi 1.A. El monitor de neutrones IGY diseñado en 1958 por Simpson fue el detector estándar para el estudio de las variaciones temporales de la intensidad de los rayos cósmicos cerca de la tierra, hasta que en 1964 se diseño el monitor de neutrones **NM64** más grande y con mayor tasa de conteo.

La elaboración de este trabajo ha contado con tres monitores de neutrones BP28 Chalk River[15] del tipo NM64 desarrollados por el Observatoire Midi Pyrénées y proporcionados por el Grupo de Investigación Espacial (SRG-UAH).

La siguiente figura muestra una foto del monitor BP28 Chalk-River en el que se señala la conexión BNC de salida en la que acoplaremos nuestro sistema de detección (**OUTPUT**) y las conexiones de alimentación del amplificador (PIN C + 13V) y de voltaje al que trabaja el monitor (-2800V).

Figura 1.G: Monitor de neutrones BP28 Chalk-River

Hemos incluido las especificaciones del monitor[14], las medidas y los materiales utilizados para su armazón, como se expone en la figura 1.E, por si resultan de utilidad para el lector.

Monitor		**Moderador**		**Productor**		**Reflector**	
Longitud activa	191 cm	Material	Polietileno	Material	Plomo	Material	Polietileno
Diámetro	14.8 cm	Grosor medio	2.0 cm	Profundidad media	156g cm-2	Grosor medio	7.5 cm
Presión	0.27 bar						

Tabla 1.1: Especificaciones del monitor BP28 y de su armazón

Figura 1.G obtenida de GEANT4 CaLMa[16]

1.5. Diseño del sistema

En vista de lo anteriormente expuesto, es necesario el diseño y construcción de un sistema capaz de acondicionar los pulsos procedentes del monitor de neutrones y detectarlos mediante un dispositivo lógico programable. En este dispositivo se implementan tanto las rutinas de detección, como las rutinas necesarias para la conexión, mediante los protocolos de comunicación Wi-Fi, SNTP y MQTT, con la estación base y demás dispositivos incluidos en el sistema. La elección de este dispositivo se reflexiona en el apartado 2.2.

La estación base contiene todos los elementos necesarios para el almacenamiento, análisis, representación y comunicación de los datos obtenidos a la comunidad científica. Del mismo modo, la elección de estos elementos se estudia en el capítulo 5.

Además, es indispensable disponer de un barómetro para establecer una correlación de los resultados con la presión atmosférica y así, realizar las correcciones que sean necesarias. En nuestro caso contaremos tanto con la estación meteorológica Vaisala, como con un módulo periférico económico de apoyo conectado a nuestro hardware.

Otros elementos necesarios son el uso de un módem inalámbrico capaz de interconectar los dispositivos de la red interna entre sí y con el ISP que suministre Internet. Adicionalmente, dado que el sistema se ubica en las instalaciones de la universidad, se ha configurado una VPN para evitar conexiones de agentes externos a la institución.

La figura representa un esquema general de los dispositivos anteriormente expuestos:

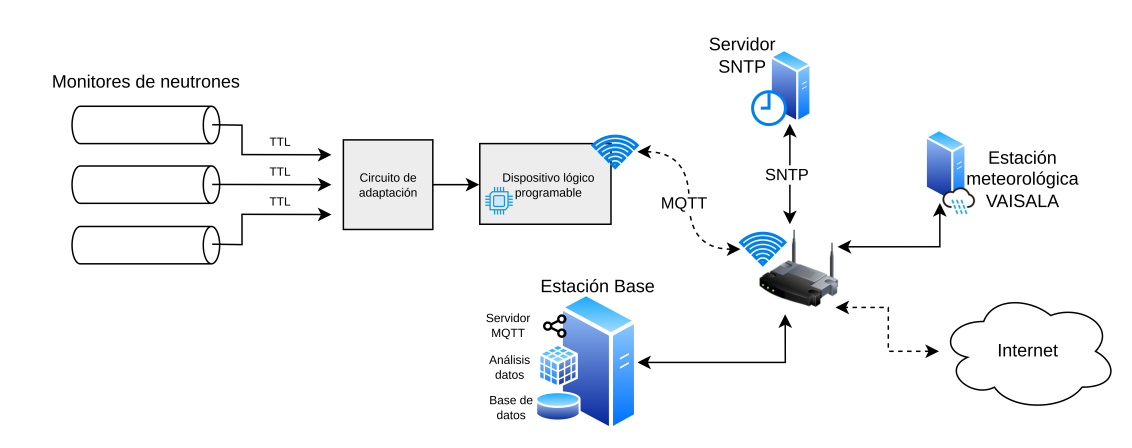


Figura 1.H: Esquema general del proyecto

Dado este esquema general, se podrán exponer múltiples soluciones con distintos componentes en función de las particularidades o necesidades de cada situación.

Capítulo 2

Hardware de adquisición

La finalidad de este capítulo es la elaboración y confección de un sistema de adquisición que reúna en un único circuito las conexiones entre el dispositivo lógico programable, la etapa de acondicionamiento, los periféricos y el sistema de alimentación.

La etapa de acondicionamiento constituye el primer bloque del sistema y tiene como objetivo amplificar y limpiar la señal de entrada, procedente de los monitores de neutrones, para el resto de bloques. Adicionalmente, hemos simulado el circuito con la herramienta LTSpice ayudándonos a ajustar los valores del bloque de amplificación.

Por otro lado, hemos dedicado la sección 2.2 al estudio de los sistemas basados en un chip o también llamados SoC, que en los últimos años han revolucionado el mercado gracias a su potencia y a la eficiencia con la que manejan sus recursos. En concreto, hablaremos del microcontrolador ESP-WROOM-32 enumerando sus características y comparándolo con otros dispositivos del mercado.

Este microcontrolador contará con un sistema de alimentación basado en la placa de prototipado LM2596 y obtendremos los datos meteorológicos gracias al módulo barómetro BMP180, en la sección 2.3 se explicará la importancia que tiene en este sistema.

Por último nos centraremos en el diseño del esquemático y el circuito impreso mediante la herramienta de automatización de diseño electrónico Kicad 6. Además, comentaremos el proceso de fabricación de estos circuitos impresos y como podemos protegerlo.

2.1. Electrónica de acondicionamiento

La conexión directa del monitor de neutrones a una de las entradas del dispositivo detector tiene una serie de importantes problemas en cuanto a eficiencia en la detección y la existencia de falsos positivos. Es por este motivo que surge la necesidad de diseñar un sistema capaz de filtrar el ruido y amplificar su salida, de tal manera que los pulsos resultantes sean próximos a un pulso cuadrado TTL.

La solución propuesta modela un circuito electrónico de alta frecuencia que utiliza un amplificador operacional comparador como se muestra en la figura 2.A:

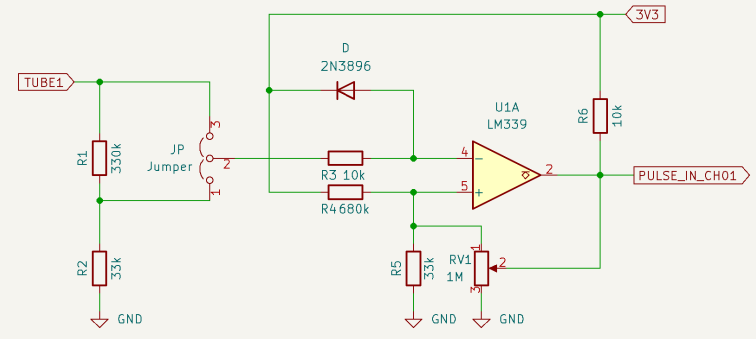


Figura 2.A: Circuito electrónico amplificador operacional comparador

2.1.1. Comparador inversor con histéresis

Este circuito se basa en la utilización del integrado LM339 de la serie de comparadores de voltaje LMx39x, capaz de trabajar con un tiempo de respuesta de 300ns, es decir, a una frecuencia de 3,3 GHz. Para la configuración y cálculo del valor de los elementos pasivos del circuito, hemos reproducido el modelo *Inverting Comparator with Hysteresis* indicado en el datasheet del componente.

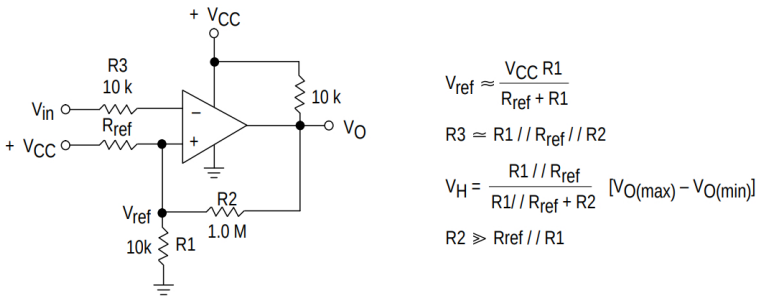


Figura 2.B: Conparador Inversor con Histeresis

Dado que el voltaje suministrado al circuito es de 3.3V y la amplitud del pulso no sobrepasa los 500mV, podemos obtener el valor aproximado de todas las resistencias del circuito. Posteriormente, se ajustará cada valor para obtener el ciclo de histéresis adecuado que regule el área de detección. Además, para proteger el integrado de sobretensiones se ha ubicado un diodo **Schottky** en cada entrada.

2.1.2. Simulación del circuito con LTSpice

Hemos realizado la simulación del circuito mostrado en la figura 2.A mediante la herramienta de simulación LTSPice.

En primer lugar, hemos simulado el comportamiento del integrado en un régimen transitorio de 100us, añadiendo ruido al pulso de entrada (verde) con la ayuda de la funcion *white()*. Como podemos notar, el amplificador operacional LM339 elimina por completo dicho ruido ya que actúa como si fuera un interruptor cuando se alcanzan los voltajes de entrada deseados. Además, como hemos configurado, la señal de entrada se encuentra invertida respecto a la salida.

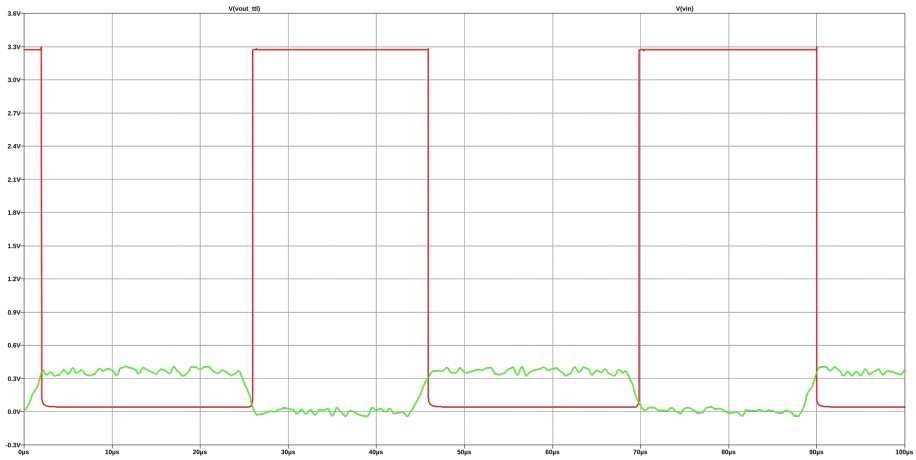


Figura 2.C: Simulación voltajes I/O del circuito

A continuación, hemos decidido representar un ciclo o curva de histéresis, que modela el comportamiento de un sistema cuya salida depende de la entrada. Como resultado, el sistema puede tomar diferentes valores incluido cuando las condiciones externas son idénticas ya que depende de su estado previo.

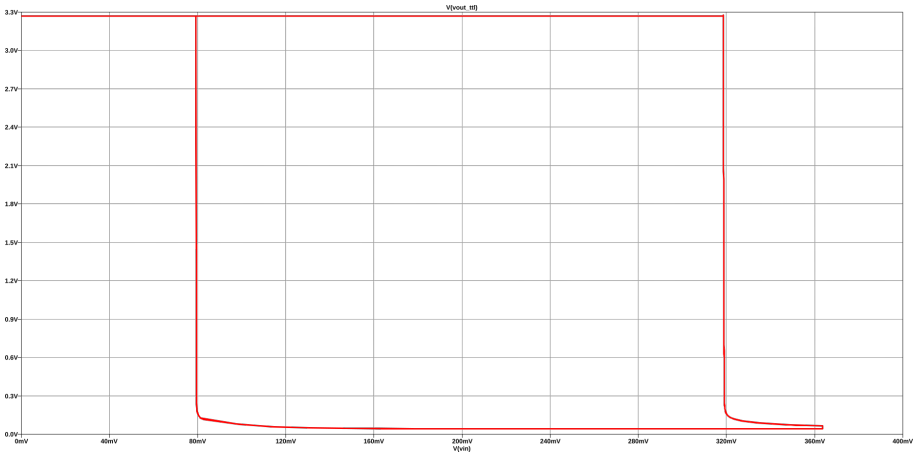


Figura 2.D: Simulación representación del ciclo de histéresis

En la figura 2.D, se indica como se producen las conmutaciones de 3.3V a 0V de la señal de salida (eje y) cuando el pulso de entrada (eje x) sobrepasa los 80mV y los 320mV.

2.2. Dispositivo lógico programable: ESP32

Este apartado está dedicado al análisis de los dispositivos que pueden realizar la tarea de detección de los pulsos acondicionados. Para ello, es necesario repasar las distintas arquitecturas que existen hoy en día a la hora de crear un dispositivo lógico programable:

- **Arquitecturas clásicas**: Las arquitecturas **Harvard** y **Von Neumann**, son estructuras de computadoras que se han utilizado desde los inicios de la computación. Aunque se basan en el mismo concepto, estas dos arquitecturas tienen algunas diferencias en la forma en que el procesador y la memoria principal se conectan. Actualmente muchos sistemas operativos implementan arquitecturas basadas o modificadas de las arquitecturas clásicas.
- **INTEL Y AMD**: Las arquitecturas computacionales de **Intel** y **AMD** de 32 y 64 bits (también conocidas como x86 o x64, respectivamente) se caracterizan por ser una arquitectura de modelo CISC, Complex Instruction Set Computer, ya que los procesadores tienen un conjunto de instrucciones extenso y complejo. Estas arquitecturas son empleadas en computadores de uso personal.
- **ARM**: Esta arquitectura, llamada *Advanced RISC Machine*, implementa el conjunto de instrucciones reducido RISC. Su ventaja frente al resto de arquitectura es el bajo consumo de sus procesadores. Es la arquitectura preferida en la tecnología movil y en los **SoC**.

2.2.1. Sistemas basados en SoC

Los SoC (System-on-Chip) son circuitos integrados en un único chip que incorporan múltiples componentes, formando así un sistema completo sin necesidad de otros chips de apoyo. Los SoC son un buen candidato para nuestro proyecto debido al poco consumo de sus procesadores, la potencia de sus sistemas y que se instalan en dispositivos económicos. Dentro de los SoC existen infinidad de dispositivos con muchas prestaciones, pero las placas más demandadas son:

- **Arduino**: es la marca más usada debido a su bajo coste y fácil programación. No obstante, por estos motivos tienen funciones limitadas y poca potencia. Pero cuenta con una gran comunidad y multitud de periféricos que ayudan a suplir dichas carencias.
- **Espressif**: se centran en crear chips de bajo consumo y con Wi-Fi y Bluetooth integrados. Son chips más potentes en comparación con los anteriores e implementan la mayoría de las librerías desarrollads para Arduino.
- **Raspberry Pi**: las tarjetas Raspberry son una buena opción en cuestión de almacenamiento de información y procesamiento de datos. Estos chips permiten utilizar sistemas operativos GNU/Linux, pero con un coste varias veces superior a los anteriores SoC.
- **BeagleBoard**: estas tarjetas son las más potentes, y por tanto, las menos económicas. Al igual que para Raspberry, permiten utilizar sistemas operativos GNU/Linux, pero requieren librerías específicas y un gran conocimiento del programador.

Por lo tanto, el dispositivo elegido ha sido un ESP32[17] de la marca Espressif, característico por poseer tecnología Wi-Fi y Bluetooth integrado y por emplear un microprocesador Tensilica Xtensa LX6 con buenas prestaciones.

2.2.2. ESP-WROOM-32

El microcontrolador ESP-WROOM-32[18] se encuentra integrado como un dispositivo SoM (System-on-Modules) caracterizado por ser un sistema de procesamiento integrado en un placa. Esta placa incluye, además de un SoC, componentes como núcleos de procesador y bloques de memoria.

Es un sistema de núcleo dual con un microprocesador Tensilica Xtensa LX6 incorporado, con posibilidad de poder controlar individualmente cada núcleo. Puede trabajar a temperaturas entre -40ºC y 85ºC y la frecuencia de reloj es ajustable (de 80 MHz a 240 MHz). Incorpora una memoria ROM de 448KB, SRAM de 520KB y SRAM en el RTC de 16KB. Además cuenta con conexión Bluetooth, Bluetooth LE y Wi-Fi, implementando el protocolo TCP/IP y el estándar 802.11 b/g/n. Para securizar todas las conexiones, el SoC incorpora un acelerador criptográfico con distintos algoritmos.

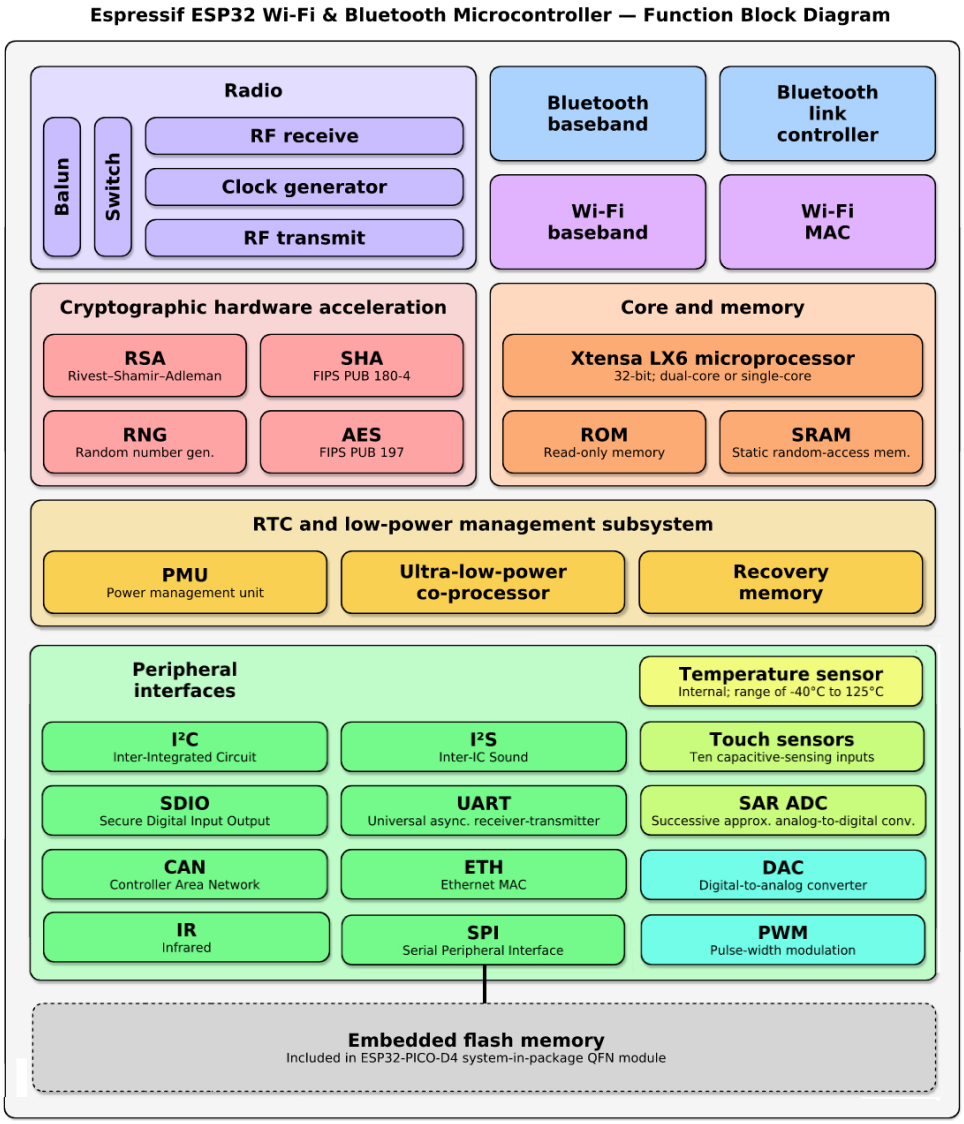


Figura 2.E: Diagrama de bloques del ESP-WROOM-32

Figura 2.E modificada de Wikipedia[19]

Los pines GPIO incluyen dos ADCs de 12 bits que se pueden configurar para 9, 10 y 12 bits de resolución con una atenuación de -0dB, -6dB y -11dB. También incluyen dos DACs de 8 bits. Inicialmente se planteó que el proceso de detección se realizara mediante esta tecnología, tomamos como referencia un estudio publicado por Alejandro Maier en el que se utiliza el ESP-WROOM-32 para crear un prototipo de osciloscopio inalámbrico, demostrando que el uso más óptimo de los ADCs era a 12 bits de resolución con una atenuación de -6dB.

En otro estudio[20] realizado por Jovan Ivkovic, se comparan varios microcontroladores del mercado realizando una serie de tests *sintéticos*, es decir, tests en los que se estudian componentes o operaciones específicos sin tener en cuenta más factores. El microprocesador del ESP-WROOM-32, obtiene unos resultados de 176 millones de operaciones de enteros por segundo y 2,805 millones de operaciones de coma flotante por segundo. Comparando estos resultados con otros microcontroladores de la misma generación, vemos datos bastante diversos, estando el ESP-WROOM-32 en la media.

Este microcontrolador es líder en la automatización del hogar, aplicaciones IoT de bajo consumo con concentración de sensores y registradores de datos, y dispositivos electrónicos portables, entre otras utilidades.

2.3. Periféricos

Aunque el proyecto cuenta con el barómetro VAISALA de altas prestaciones, hemos decidido incluir en el circuito el módulo **BMP180**. Este módulo tiene la ventaja de ser fácil de aplicar en el circuito y muy económico.

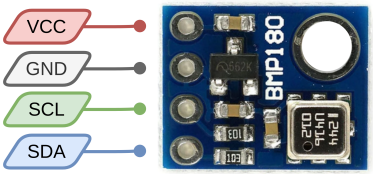


Figura 2.F: Módulo BMP180

Este módulo cuenta con el protocolo de comunicación I2C (Inter-Integrated Circuit), diseñado para transferir datos bit a bit entre dispositivos digitales. Para el correcto funcionamiento de este protocolo es necesario un dispositivo maestro, normalmente un microcontrolador, conectado a esclavos, pueden ser tanto sensores como monitores, que realizan las órdenes del maestro. Para este protocolo se implementan los puertos SDA y SCL. El puerto SDA es el encargado de enviar los mensajes entre maestro y esclavo; el puerto SCL, controlado por el maestro, transmite los pulsos de reloj.

2.4. Sistema de alimentación

Nuestro circuito tiene implementadas dos formas de alimentación dependiendo de las necesidades o del material del que dispondremos cuando se realice la instalación del sistema de adquisición.

La manera principal de suministrar energía al circuito emplea uno de los conectores **BNC** de la placa SRG (mostrada en la figura 2.G). De esta forma podemos, mediante el uso de una fuente de alimentación de laboratorio, proporcionar de forma constante la potencia requerida en nuestro circuito.

En caso de no disponer de una fuente de alimentación, se ha situado otro sistema basado en los conectores de alimentación **DC jack** de 2mm, utilizados en dispositivos con una tensión máxima de 30V. Para esta configuración necesitaremos un adaptador AC/DC que proporcione 12V y 1A.

Las entradas descritas anteriormente se acoplan al módulo regulador **LM2596** que regula el voltaje de entrada de 12V a 3.3V, que es el voltaje en el que trabajan los componentes del circuito. Pero además, obtenemos la ventaja de que mejoramos la **eficiencia**, traduciéndose en una mejora en el rendimiento del circuito. También **protegemos** el circuito ante sobrecargas y picos de tensión. Los convertidores de corriente continua (**DC-DC**), a diferencia de los transformadores tienen un tamaño y peso menor, por lo que siempre es deseable incorporar este dispositivo si necesitamos que el circuito funcione con diferentes niveles de tensión continua.

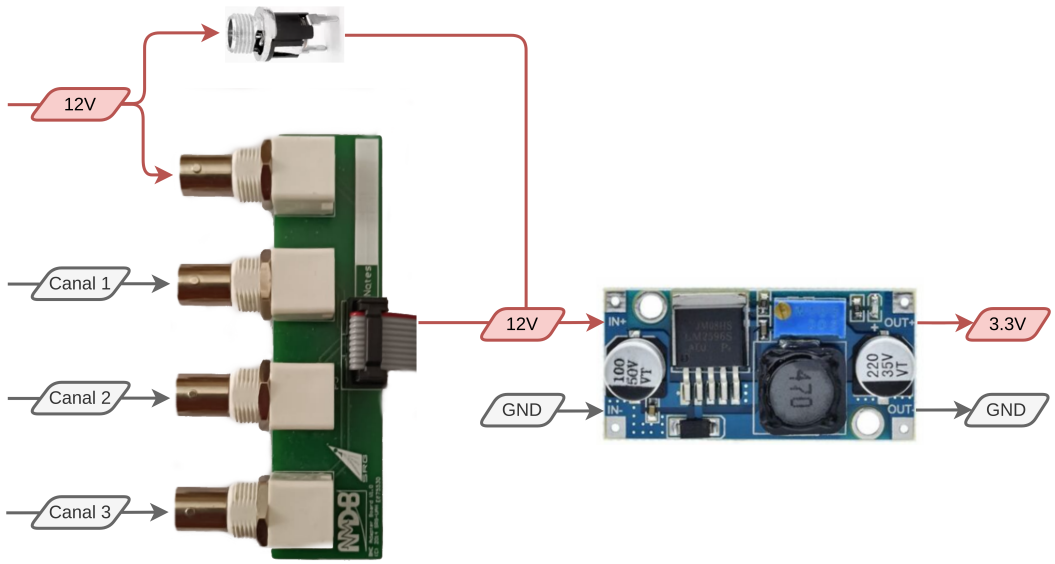


Figura 2.G: Sistema de alimentación basado en el módulo LM2596

2.5. Diseño electrónico

Kicad 6[21] es un entorno de automatización de diseño electrónico que facilita el desarrollo de esquemas para circuitos electrónicos. De sus herramientas principales nos centraremos en el *editor de esquemas*, el *editor de placas* y el *conversor de imágenes*. Aunque la utilización del *editor de símbolos y de placas* nos ayuda a diseñar nuestros propios componentes, simplificaremos el proceso descargando los diseños de páginas gratuitas como SnapEDA[22].

2.5.1. Diseño del esquemático

Un **símbolo** es un figura que representa todas las conexiones físicas de un componente electrónico, en el caso de los amplificadores operacionales se emplea un triángulo, como en la figura 2.A, aunque cuando la figura representa varios operacionales juntos se suele ilustrar como en la figura 2.H.

Una vez hemos descargado los símbolos necesarios para nuestro proyecto creamos una **librería** mediante el *explorador de librerías* de símbolos, reuniendo todos los componentes en un único archivo. De esta forma, seleccionando esta librería, de forma local o global, encontramos nuestros símbolos en el apartado de añadir símbolo o pulsando la tecla *A*.

La colocación de los componentes sobre el documento se debe realizar de forma organizada, situando las entradas del circuito separadas del resto de componentes y evitando la conexión directa de los componentes con cables. La mejor forma de señalizar e interconectar los componentes es con el uso de etiquetas globales *(Ctrl+L)*. Además, el patillaje que no se desee conectar, se debe indicar con un aspa pulsando la tecla *Q*.

La siguiente figura representa el chip LM339N con tres operacionales conectados según la solución diseñada en el apartado 2.1:

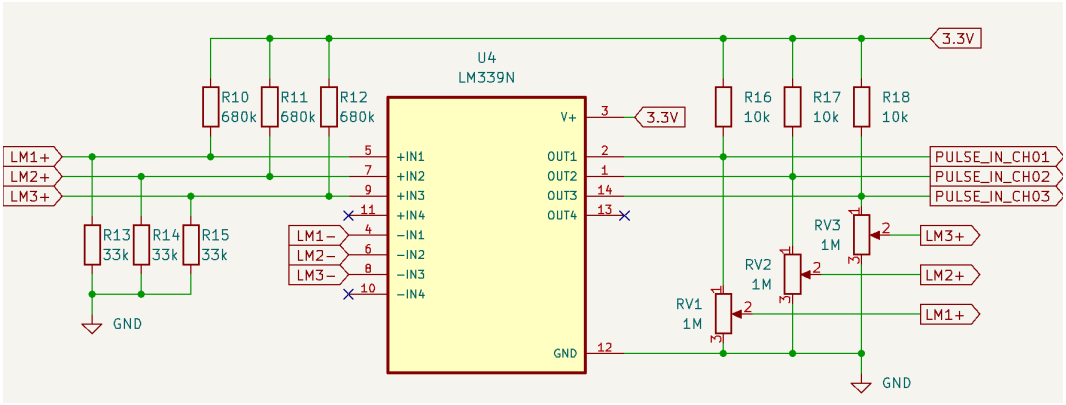


Figura 2.H: Conexión realizada entre el LM33N y el circuito mediante etiquetas globales

Por último, para comprobar que no existen errores de conexión en el esquema ejecutamos la herramienta *Control de reglas eléctricas (ERC)* desde el menú **Inspeccionar**. EL resultado de este análisis nos muestra una serie de mensajes catalogados según su riesgo. Siendo los mensajes etiquetados como **Error**, los que debemos solucionar de forma obligatoria debido a su gravedad. Si por el contrario existe algún aviso que se desee ignorar, se puede utilizar la herramienta de *Exclusión de violaciones*.

2.5.2. Diseño del PCB

Una vez que tenemos listo el esquemático, comenzamos con el diseño del PCB (*Printed Circuit Board*). En primer lugar debemos asegurarnos de comprobar que todos los footprints tienen las medidas y el patillaje correspondiente. Los **footprints** o huellas del PCB muestran las características físicas necesarias de los componentes que utilizamos en la placa, así como sus conexiones. Por este motivo, es esencial exponer con exactitud el contorno del encapsulado, la designación de pines y el patrón de tierra, entre otros. En caso de que el símbolo no contenga la huella que necesitamos, podemos descargarla de páginas gratuitas como SnapEDA[22].

Cuando hayamos asignado todas las huellas correspondientes con la herramienta de asignación de huellas, en el editor de esquemas o en el gestor del proyecto, presionamos el símbolo para abrir la placa en el editor de placas.

Para visualizar en la placa los cambios realizados en el esquema, pulsamos F8 o en su respectivo símbolo. La primera vez que se ejecute nos coloca los componentes desordenados, una de las partes más importantes a la hora de realizar nuestro PCB es la ubicación de los componentes. Cuanto más optimizada esté la placa más pequeña es y, en consecuencia, será más barata de fabricar.

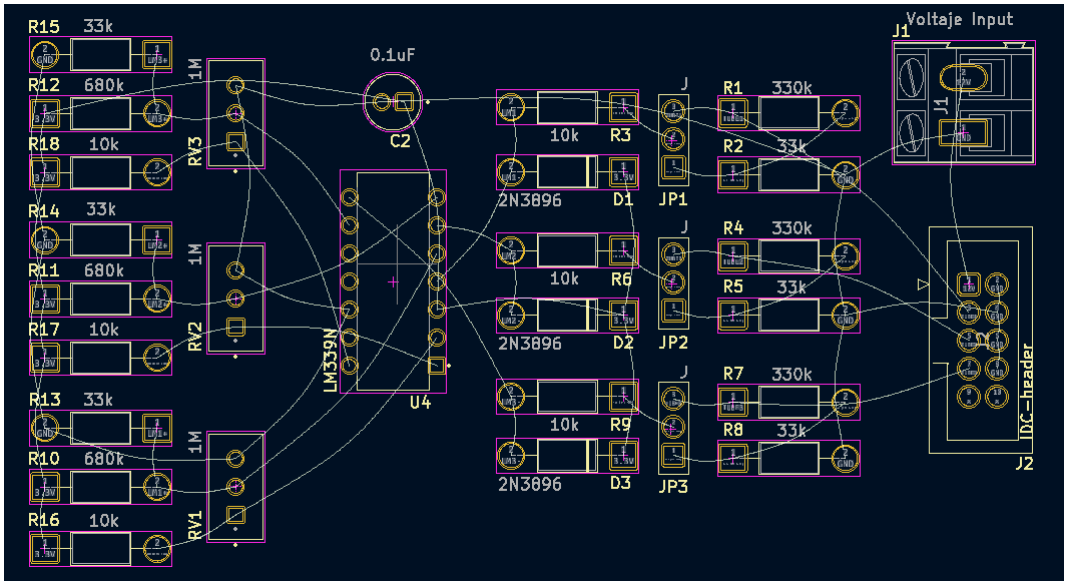


Figura 2.I: Ejemplo de ordenación de la etapa de acondicionamiento

Ahora, es el momento de **enrutar** todas las conexiones del circuito. En nuestro caso, por simplificación, hemos decidido utilizar cuatro capas de cobre, de esta forma no estaremos tan limitados a la hora de conectar las vías.

Dos de las capas contienen únicamente zonas rellenas, una para GND y otra para VCC. En las capas restantes se conecta el patillaje utilizando **pistas** de cobre. Por último, en la capa *Edge.Cuts* crearemos un rectángulo con las medidas de la placa y colocaremos los orificios que necesitemos para atornillar nuestra placa.

Kicad ofrece una herramienta para renderizar en 3D las placas que diseñemos denominada *Visor 3D*, ubicada en el menú *Ver* o mediante el comando *(Alt+3)*. Hemos obtenido los modelos 3D gratuitamente de la página oficial de GrabCad[23] y los hemos asignado a cada huella. El resultado del modelo 3D se ilustra en la siguiente figura:

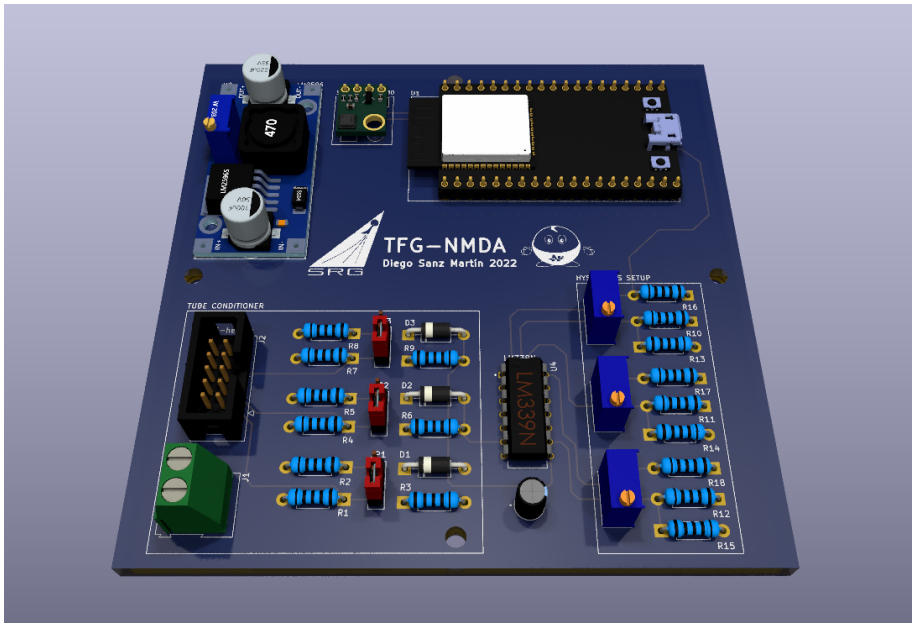


Figura 2.J: Renderización 3D del PCB

2.5.3. Fabricación del PCB

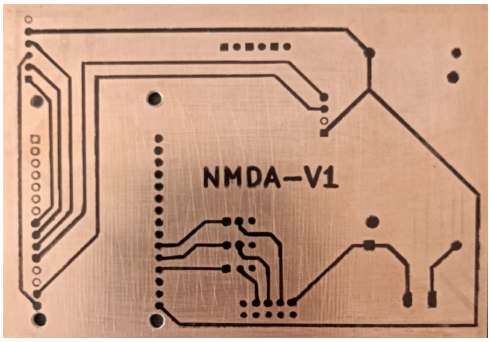
La elaboración de circuitos impresos se remonta a la década de 1930, cuando el ingeniero Paul Eisler desarrolló el primer PCB como parte de una radio. Pero no fue hasta 1940 cuando se empezó a utilizar esta tecnología a gran escala.

Están formados por una o varias capas conductoras superpuestas entre otras aislantes. Como hemos visto en el apartado de diseño de PCB 2.5.2, dichas capas conductoras están formadas generalmente por pistas o superficies de cobre. De esta forma somos capaces de interconectar todos los componentes de un circuito y al mismo tiempo sostenerlos en una base plana.

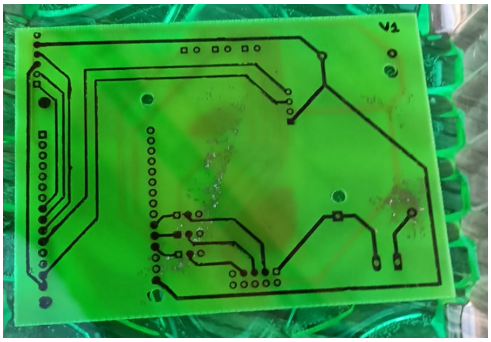
Los métodos más utilizados en la elaboración de circuitos impresos se dividen en:

- **Grabado ácido y alcalino**: Estos métodos se divide en dos pasos, la transferencia del patrón del circuito sobre la placa y el ataque por medio de un ácido o un fuerte agente oxidante respectivamente. La transferencia del circuito a la placa se puede realizar mediante serigrafía, fotograbado o transferencia térmica, todos se basan en la utilización de materiales que protejan el dibujo de las pistas sobre el ataque del ácido. Los productos químicos más utilizados para el revelado de las placas impresas son el cloruro férrico y cloruro cúprico.

  Este método lo utilizamos durante la generación de las primeras versiones del código y de la placa:



(a) Transferencia térmica del circuito



(b) Ataque químico con cloruro férrico

Figura 2.K: Proceso de fabricación de PCB

- **Grabado con plasma**: En este proceso convierte las superficies sólidas que deseamos eliminar directamente en gas. De forma selectiva se produce una reacción en la superficie de la placa a través de una mezcla de gases, para después excitar sus moléculas mediante radiofrecuencias logrando cambiar su estado a gas y produciendo finalmente la superficie grabada.
- **Grabado láser**: Este tipo de grabado en seco emplea hardware controlado por computadora, grabando el patrón del circuito gracias a un rayo láser de gran potencia. Comparado con los métodos anteriores, el grabado láser reduce el número de pasos reduciendo el tiempo de producción.

- **Fresado**: Similar al proceso anterior, pero en este caso una máquina de control numérico retira el material conductor de la placa de cobre mediante un taladro. Este método de mecanizado suele ser el más habitual en la industria, pero dependiendo del número de capas suele combinarse con otros métodos.

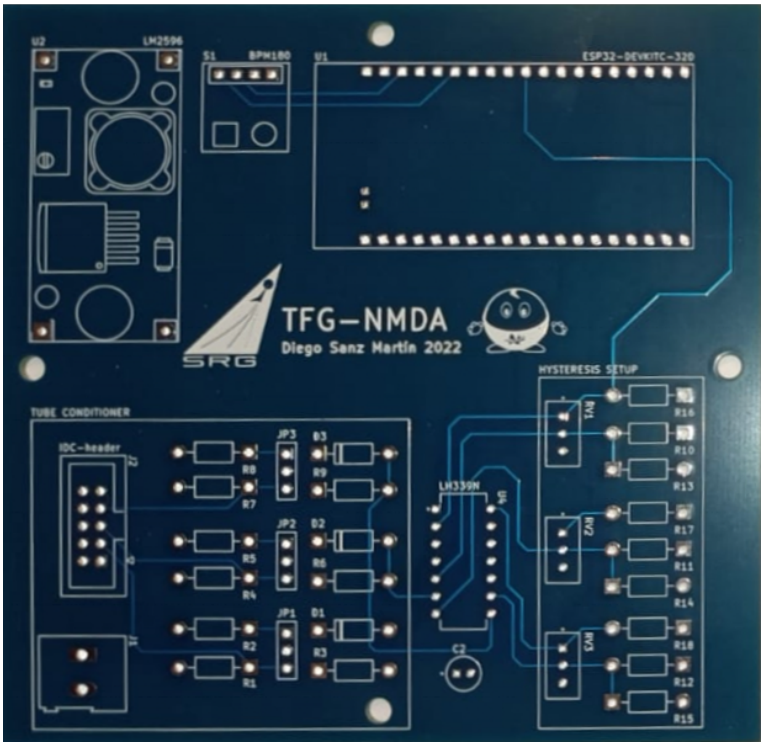


Figura 2.L: Placa de circuito impreso fabricada por JLCPCB

2.5.4. Caja estanca

Con el objetivo de unir y proteger los componentes electrónicos del sistema hemos decidido usar una caja derivación IDE-EX161 con protección IP65, por lo que el sistema es hermético al polvo y está protegido contra chorros de agua de baja potencia. Parte de esta protección se ha perdido ya que hemos perforado la caja para exponer las entradas BNC y el conector de alimentación. Sin embargo, no esperamos que tenga que resistir condiciones climáticas adversas.

(a) Frontal de la caja (b) Conectores BNC de la caja

Figura 2.M: Caja derivación IDE-EX161

Capítulo 3

Entorno de Programación del ESP32

Este capítulo esta dedicado al estudio de los bloques que componen el entorno de programación sobre el que hemos desarrollado nuestra aplicación para el microcontrolador ESP32.

Como punto de partida encontramos las herramientas desarrolladas por Espressif para el desarrollo de sus dispositivos, que son en gran medida, una de las principales razones por las que se concibió este trabajo de fin de grado. Como veremos más adelante, gracias a este set de herramientas se potencian las características mostradas en el apartado 2.2.2.

Del mismo modo, se ha realizado un estudio de los distintos sistemas operativos existentes en los sistemas en un chip (SoC), examinando sus ventajas y desventajas para finalmente explicar que hace que los sistemas en tiempo real (RTOS) sean tan interesantes.

A continuación, encontramos una sección dedicada al estudio del sistema operativo en tiempo real FreeRTOS, que se presenta como un candidato idóneo dentro de los RTOS tanto su adaptación a los sistemas Espressif como por su eficiencia en los recursos que maneja. Adicionalmente, trataremos conceptos y mecanismos indispensables en la programación de sistemas operativos en tiempo real particularizando los métodos empleados.

La última sección tiene como objetivo la breve descripción de los protocolos y servicios que, por los términos mencionados durante el desarrollo del software, resultan relevantes en este documento.

3.1. Espressif IDF

En la actualidad, podemos encontrar multitud de entornos de producción y desarrollo de software. Arduino ofrece el entorno **ArduinoIDE** que resulta perfecto para la programación de sus placas y de otras placas soportadas. Este entorno es ideal para programadores principiantes, ya que la instalación de librerías y el soporte de placas se realiza interactivamente desde sus menús.

El principal problema de usar ArduinoIDE es la gran limitación que existe a la hora de configurar tanto el hardware de nuestro microcontrolador como el uso de librerias o demás elementos software. Es por esta razón que hemos preferido instalar el framework de desarrollo **ESP-IDF**.

Este elemento, a diferencia del entorno de Arduino, cuenta únicamente con un set de herramientas para la configuración, compilación y *flasheo* del firmware a la placa. Por lo tanto, es necesario el uso de otro IDE como Vim o VS-Code.

3.1.1. Menuconfig

La configuración de nuestro microcontrolador ESP32 se realiza en un menú basado en terminal mediante el comando:

```
$ idf.py menuconfig
```

Encontrando un listado de los distintos aspectos que podemos modificar en nuestro ESP32, de los cuales son relevantes para nuestro proyecto *Serial flasher config* y *Partition table*.

Además de las opciones por defecto , podemos crear nuestro propio submenú gracias al fichero **Kconfig.projbuild**[1] ubicado en la carpeta *main* de nuestro proyecto.

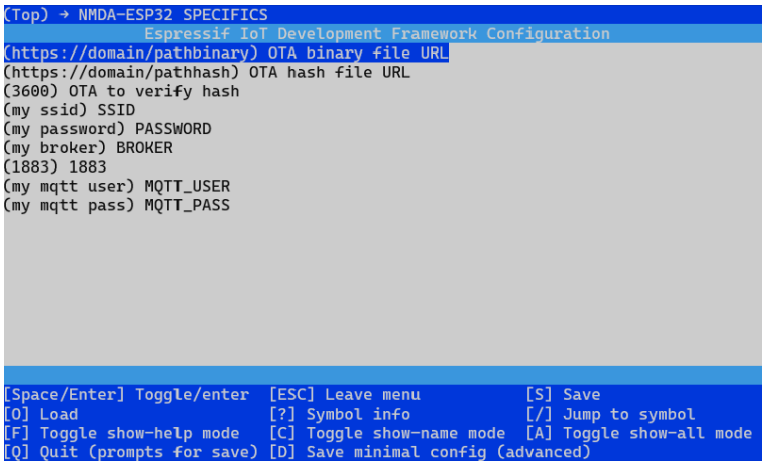


Figura 3.A: Menú específico para el proyecto

Todos estos parámetros funcionan como variables globales en nuestro proyecto y son usados durante el código mediante una abreviación. Por ejemplo, la variable que almacena la contraseña del Wi-Fi, es referenciada por **CONFIG_WIFI_PASSWORD**.

De esta forma simplificamos la distribución de nuestro software, pudiéndose implementar con una configuración distinta sin modificar ninguna línea de código.

[1] Fichero Kconfig.projbuild dentro de la carpeta `main`

3.1.2. Compilación

Después de configurar el dispositivo ESP32 y tras haber obtenido el fichero sdkconfig podremos ejecutar el comando:

```
$ idf.py build
```

El proceso de compilación se realiza a través de la herramienta de generación de código **CMake**. Los archivos de configuración utilizados se encuentran en los siguientes directorios:

- **Makefile**: Punto de entrada en la compilación, se establece el nombre del proyecto.
- **CMakeLists.txt**: En este archivo se establece el directorio del código del proyecto y se añaden componentes externos, como en nuestro caso la librería "esp-idf-lib".
- **main/CMakeLists.txt**: Registro de los de ficheros código, archivos de cabecera y archivos insertados.

La ejecución del comando build generará un directorio "**build**" con archivos provisionales de compilación, además de contener las bibliotecas y los archivos de salida **binarios** finales. En consecuencia, el directorio no suele tener un control de código fuente ni se distribuye.

Tras realizar el proceso de compilación se produce el *flasheo* del archivo binario generado mediante el comando:

```
$ idf.py flash monitor
```

Este proceso, como se ilustra en la figura 3.B, se realiza una conexión entre la placa y el ordenador generalmente mediante un cable OTG. Este cable también nos servirá como puerto de comunicaciones con el dispositivo, que al ejecutar el comando **"monitor"** nos despliega una terminal con sus mensajes de *log*.

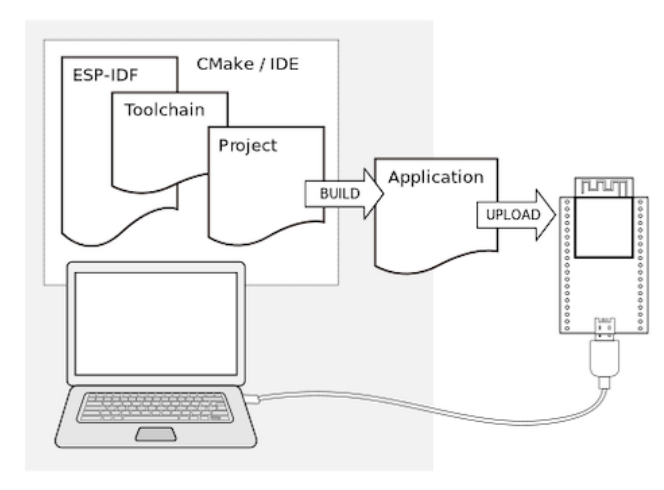


Figura 3.B: Proceso de compilación de la placa

Figura 3.B obtenida de la documentación oficial Espressif[24]

3.2. Estudio de sistemas operativos para SoC

El sistema operativo que escogemos para un microcontrolador es otro de los aspectos más importantes a la hora de elaborar un software. Considerando que la gran mayoría de estos dispositivos son open source, muchos desarrolladores elaboraron distintos SO y IDEs compatibles. El objeto de este apartado es comparar los sistemas operativos que existen para SoC y elegir el más adecuado para nuestro proyecto.

- **MicroPython**: Se trata de una implementación de Python, uno de los lenguajes de programación más usados en la actualidad. Formado por un subconjunto de la librería estándar de Python optimizado para microcontroladores. Ideal para proyectos en el que la frecuencia de ejecución y el acceso a todos los elementos del hardware no es vital.
- **Mongoose OS**: Surgió como una alternativa IoT a los sistemas. Actualmente ya no es un sistema operativo muy utilizado, pero nos sirve como antecedente de un modelo que prioriza la conectividad con distintas plataformas como Google Cloud, AWS o Azure.
- **RTOS**: Es el lenguaje para microcontroladores por excelencia, desarrollado en C, es capaz de realizar multitareas en tiempo real. Existen muchos entornos de desarrollo basados en este sistema operativo como Arduino, Zerynth o FreeRTOS[25].

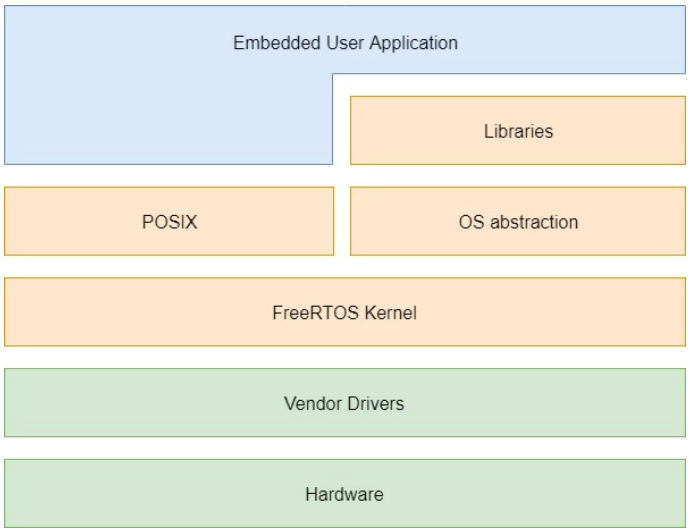


Figura 3.C: Estructura FreeRTOS + POSIX

Dado que la eficiencia de nuestro proyecto depende directamente de la capacidad que tiene el sistema de detectar los pulsos en tiempo real, el sistema operativo más adecuado es RTOS, en concreto FreeRTOS, ya que dispone de librerías y documentación disponible para los dispositivos de Espressif.

Figura 3.C obtenida de la documentación oficial de FreeRTOS[26]

3.3. Lenguaje C/C++

C es un lenguaje de programación de propósito general y actualmente es uno de los más importantes, siendo la base de otros como Java o C++. Es un lenguaje estructurado, compilado y muy eficiente ya que posee características tanto de lenguaje de alto nivel como de bajo nivel. Al ser de propósito general, es posible desarrollar cualquier tipo de programa, siendo un lenguaje versátil. Otra de las ventajas es la estandarización del lenguaje, utilizando el lenguaje ANSI C, posteriormente ratificado como estándar ISO, publicado por el Instituto Nacional Estadounidense de Estándares (ANSI).

C fue desarrollado por Dennis Ritchie a principios de los años 70 como evolución del lenguaje B. El desarrollo de C se realizó en UNIX y, en consecuencia, casi todos los programas y herramientas de UNIX, así como el compilador de C, se escribieron en C. Aunque C también se diseñó para ser capaz de migrar a otros sistemas operativos, es decir, es un lenguaje portable.

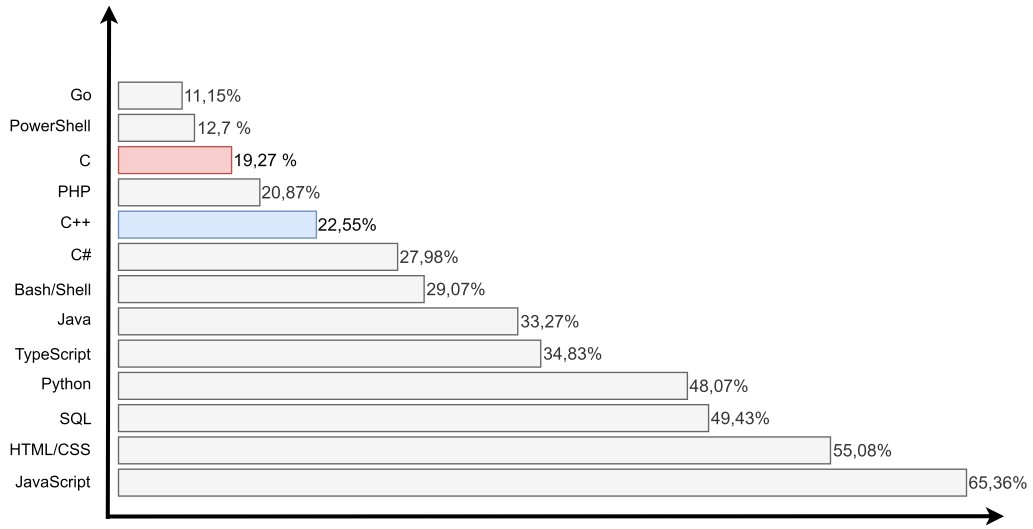


Figura 3.D: Lenguajes más utilizados según `Stack Overflow`

C++ se diseñó como una extensión de C y, por tanto, comparten características básicas como ser lenguajes estándar y compilados. Pero también existen muchas diferencias: C++ es un lenguaje de programación orientado a objetos y permite la creación de clases y objetos, el manejo de excepciones y el uso de plantillas. C++ se considera, por tanto, un lenguaje híbrido ya que tiene tanto características de la programación orientada a objetos como mejoras enfocadas a las capacidades de C.

Para este proyecto, hemos optado por emplear C, ya que aunque existe una compatibilidad entre FreeRTOS y C++, la mayoría de la documentación estaba implementada sobre C. Además, la mayoría de los microcontroladores normalmente se programan en este lenguaje y los compiladores utilizados en el proceso de *flasheo* pueden crear conflictos en el dispositivo.

Figura 3.D obtenida de la encuesta anual de `Stack Overflow`[27]

3.4. FreeRTOS

Un RTOS[28] es un sistema operativo en tiempo real que se encarga de administrar el tiempo y el espacio de los procesos de manera eficiente. Su objetivo es maximizar el tiempo de ejecución de los procesos y minimizar el tiempo de respuesta. Los RTOS se caracterizan por ser **multitarea**, por lo que pueden ejecutar varios procesos a la vez. También se caracterizan por ser preemptivos, lo que significa que un proceso puede ser interrumpido en cualquier momento y el sistema operativo asignará el tiempo de ejecución a otro proceso. Los RTOS son muy útiles en **sistemas en tiempo real**, ya que garantizan un **tiempo de respuesta** predeterminado.

En este proyecto, la eficiencia y calidad de nuestro sistema está estrechamente relacionada con la capacidad de detección. Por lo tanto, la elección de un sistema RTOS es clave para que la obtención de los datos sea lo más preciso posible. Además, los sistemas en tiempo real nos permiten gestionar de forma más eficiente los recursos de nuestro microcontrolador[29].

FreeRTOS[25] es un sistema operativo en tiempo real de código abierto, desarrollado por Real Time Engineers Ltd, líder del mercado de los microcontroladores. Proporciona los mecanismos necesarios para la planificación, dirección y control de los procesos del sistema.

3.4.1. Tareas

Una tarea o *task* es un término recurrente en la multiprogramación y hace referencia a una sección de código que se realiza de forma concurrente a otros procesos.

La **multiprogramación** es la técnica de multiplexación de varios procesos, **hilos** o **tareas** para producir un paralelismo simulado ya que, por la arquitectura de un ordenador, solo un proceso puede estar ocupando un core del CPU simultáneamente. En consecuencia, las tareas se dividen en ráfagas de CPU y se intercalan según una lógica descrita por el **planificador**, por ejemplo asignando una prioridad a cada tarea.

En FreeRTOS la creación de tareas se realiza mediante la función ***xTaskCreate()*** asignando a una función pasada como parámetro la propiedad de concurrencia. Además, dado que el planificador del sistema operativo funciona mediante prioridades, se puede fijar una prioridad más alta a una tarea más importante mediante otro parámetro del tipo entero en dicha función.

En nuestro caso, además de realizar una planificación a través de prioridades hemos optado por la implementación de la función ***xTaskCreatePinnedToCore()*** que asigna dicha tarea a un core específico, pudiendo dividir las tareas según la carga de CPU que necesiten y otorgando más recursos a tareas más pesadas.

3.4.2. Colas

En la multiprogramación a veces surge la necesidad de **comunicar tareas** entre sí, este problema es muy difícil de resolver con la variables comunes de C ya que, aunque definamos una variable global para las tareas, necesitaremos una lógica que nos indique cuando esa variable ha cambiado, además que pueden surgir excepciones al intentar acceder a una misma variable desde varios procesos.

Una **cola** es un mecanismo de comunicación de procesos que se caracteriza por ser una secuencia de elementos en la que existe operación de inserción `xQueueSend()` y una operación de extracción `xQueueReceive()`. La extracción se puede realizar por dos métodos, si la cola es **LIFO** el último elemento en entrar es el primero en salir, este modelo se suele asemejar a una pila de platos que se debe lavar. En cambio, si la cola es del tipo **FIFO** el primer elemento en llegar es el primero en salir, como por ejemplo en la cola de un establecimiento.

Ambas colas se pueden representar mediante el siguiente diagrama:

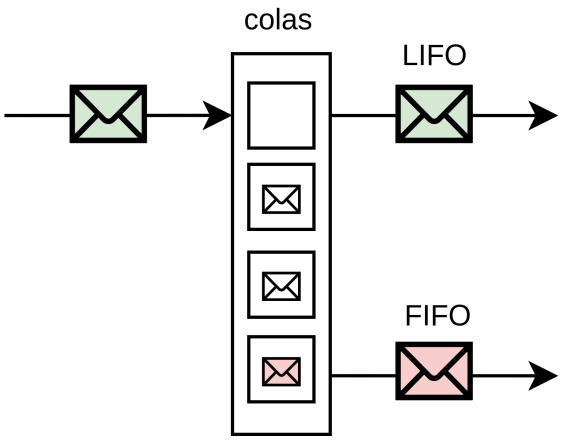


Figura 3.E: Diagrama cola FIFO y LIFO

3.4.3. Semáforos

Como se ha comentado en la sección anterior, cuando dos procesos desean acceder a una misma variable se puede producir una **excepción**, esto se debe a que aunque las operaciones de escritura y lectura parezcan instantáneas realmente no lo son y su **ejecución incontrolada** puede corromper una variable o un fichero. Solo las instrucciones denominadas como *atómicas* realizaran cambios sobre variables globales durante la ejecución de una tarea.

Un **semáforo** es un mecanismo que otorga el acceso de un recurso a una tarea mediante la función **`xSemaphoreTake()`**, impidiendo cualquier operación sobre el recurso mientras dicha tarea no haya liberado el semáforo con **`xSemaphoreGive()`**. En consecuencia, se produce un bloqueo en el resto de tareas que deseen acceder al mismo recurso.

Como veremos más adelante, en este proyecto utilizaremos los semáforos para bloquear el hilo principal mientras otras secciones de código se ejecutan para desbloquearlo cuando éstas hayan alcanzado el instante deseado, por lo que no se usarán los semáforos para acceder a recursos sensibles del código.

3.5. Protocolos utilizados

Para concluir este apartado hemos visto apropiado comentar los protocolos y tecnologías más importantes sobre las que programaremos nuestro software. Además, servirá de referencia para algunos conceptos mencionados durante las etapas de desarrollo y configuración de nuestro software de adquisición y de la estación base.

3.5.1. TCP/IP

La arquitectura TCP/IP fue desarrollado por el Departamento de Defensa de EE.UU, siendo actualmente el paquete de protocolos más utilizado. Es un conjunto de protocolos que permite la comunicación entre dispositivos dentro de una red. Provee conectividad extremo a extremo y fiabilidad. Este sistema es estandarizado y se puede utilizar tanto en redes privadas como en públicas.

El modelo TCP/IP se divide en distintas capas jerarquizadas (capa de aplicación, capa de transporte, capa de red y capa física), teniendo cada una de ellas una función distinta. El objetivo es proporcionar servicios a las capas superiores y solicitar servicios a la capa inferior.

3.5.2. Wi-Fi

El Wi-Fi[30] es una tecnología de transmisión de datos inalámbrica que se basa en el estándar 802.11. La comunicación entre dispositivos terminales, es decir, con interfaz de red, se realiza mediante puntos de acceso (AP) que son los que generan las redes WLAN.

El estándar Wi-Fi 802.11 registra las normas de funcionalidad de WLAN creadas por el IEEE y define el uso de las bandas de radio industriales, científicas y médicas (ISM) para las capas tanto física como la subcapa MAC. Entre versiones del estándar 802.11 puede haber interferencias, por esto existen métodos para controlarlo como el espectro ensanchado por secuencia directa (DSSS) y la multiplexación por división de frecuencia ortogonal (OFDM).

El ESP-WROOM-32 utiliza tres versiones compatibles entre sí, siendo la más actual la versión 802.11n. El estándar 802.11n funciona en las dos bandas de frecuencia de 2,4GHz y 5GHz, llegando a velocidades de transmisión máximas de 72,2Mbps y 150Mbps, respectivamente.

3.5.3. SNTP

Es imprescindible que los ordenadores tengan un reloj preciso y sincronizado para evitar errores a largo plazo. Por ello, se han definido varios protocolos de tiempo. Los primeros fueron el Daytime Protocol (RFC 867) y el Time Protocol (RFC 868). Como curiosidad, el Time Protocol sufre el problema del año 2036, también conocido como Y2K36, en el que en 2036 su contador llegará a su valor máximo. Este problema se solucionó con el protocolo NTP (Network Time Protocol) y sus futuras versiones, una de ellas siendo SNTP (Simple Network Time Protocol).

El protocolo SNTP[31] sincroniza el tiempo de sistema de todos los dispositivos de una misma red. Los clientes obtienen esta información de un servidor de tiempo y se conectan utilizando el protocolo de la capa de transporte UDP en el puerto 123. El cliente envía una solicitud SNTP cada periodo de tiempo y ajusta el tiempo si recibe una respuesta válida del servidor.

Este protocolo es utilizado por ordenadores poco potentes ya que necesita menos memoria y recursos CPU que NTP. SNTP no dispone de todos algoritmos que implementa NTP y, como consecuencia, SNTP tiene valores de tiempo menos precisos que NTP.

Actualmente se utiliza el protocolo SNTPv4 que permite direccionamiento IPv4 e IPv6 y el intercambio de información entre servidores tanto NTP y SNTP ya que existe compatibilidad entre ellos.

3.5.4. MQTT

La tecnología del Internet de las cosas o **IoT** se ha convertido rápidamente en un aspecto central en la comodidad de muchas personas. En consecuencia, hemos experimentado un auge sin precedentes en la automatización de los procesos logrando que hasta los electrodomésticos más simples, como una tostadora, puedan estar programados y conectados a la red del hogar.

Es por este motivo, son muchos protocolos los enfocados a la tecnología IoT, siendo los más importantes AMQP, DDS, XMPP y MQTT, entre otros.

MQTT (Message Queuing Telemetry Transport[32]) es un protocolo de comunicación que utiliza el modelo Publicación-Suscripción para permitir la comunicación M2M (Machine to Machine). La arquitectura publicación-suscripción es un diseño de patrón de mensajes en el cual el remitente envía mensajes a un mediador (broker), es decir, no se comunica directamente con el consumidor. Los clientes se conectan vía TCP/IP con el broker, que mantiene registro de los clientes conectados. Los mensajes se clasifican en *topics* organizados jerárquicamente y los consumidores son los que se suscriben a los *topics* para recibir la información deseada.

MQTT, al ser un protocolo sencillo, es ideal para dispositivos de escasa potencia. Requiere de un ancho de banda mínimo, emplea los puertos 1883 y 8883 y dispone de un mecanismo de calidad del servicio (QoS).

Capítulo 4

Software de adquisición

Una vez hemos creado un marco teórico y práctico sobre el que desarrollar nuestro software, es el momento de detallar las tareas que componen el sistema de adquisición y cuales de los requisitos mencionados hasta el momento satisfacen.

El resto de secciones del capítulo se dividen como se ha implementado cada tarea, comentando las secciones de código más relevantes. Cabe destacar que se hacen multitud de referencias a los tasks, las colas y los semáforos descritos en la sección 3.4, que se implementan según la lógica descrita en dicho capítulo, también se hace referencia a conceptos relacionados con los protocolos descritos en la sección anterior.

4.1. Descripción del software de adquisición

Resumiendo lo visto hasta el momento, disponemos de tres monitores BP28 que producen un pulso cada vez que es atravesado por un rayo cósmico, dicho pulso es amplificado y filtrado por la electrónica descrita en la sección 2.1. Por ultimo, introduciremos los pulsos adaptados en el microcontrolador ESP32 en los canales asignados en el sdkconfig, que corresponden con los pines físicos del dispositivo. Además, obtendremos los datos meteorológicos mediante el protocolo I2C.

Por lo tanto definiremos las tareas del microcontrolador así:

- **Configuración de las tecnologías Wi-Fi, SNTP y MQTT**: Implementaremos el protocolo de comunicación inalámbrica para establecer una conexión con el resto de elementos de la red interna e Internet, el protocolo SNTP para sincronizar el reloj del microcontrolador y realizar marcas de tiempo precisas y el protocolo MQTT como método de comunicación con la estación base.
- **Obtención de datos meteorológicos**: Obtendremos la presión atmosférica para establecer una correlación con los resultados y realizar las correcciones que sean necesarias
- **Muestreo y detección de los rayos cósmicos**: Mediante el uso de la herramienta hardware *pcnt* se obtendrá el numero de detecciones (cuentas) que se han producido en n segundos. Aparte, haremos uso de las interrupciones del microcontrolador para obtener el instante de detección de cada rayo cósmico interceptado por el monitor.
- **Clasificación y envío de mensajes**: Se ha diseñado una lógica que obtiene los mensajes que cada tarea introduce en la cola `telemetry_queue` y según su tipo genera el JSON correspondiente insertando los datos de dicho mensaje.

4.2. Programa principal

La estructura del programa se ha diseñado para que exista una tarea principal encargada de la creación e inicialización de los recursos y tareas secundarias. Adicionalmente, esta tarea sincronizará la inicialización de dichos recursos, para que se ejecuten a medida que se produzcan los eventos a los que la ejecución del programa debe llegar.

Para asegurarnos de que la configuración se ejecuta en orden y no se crean conflictos entre las tareas, hemos implementado un sistema de sincronización basado en semáforos.

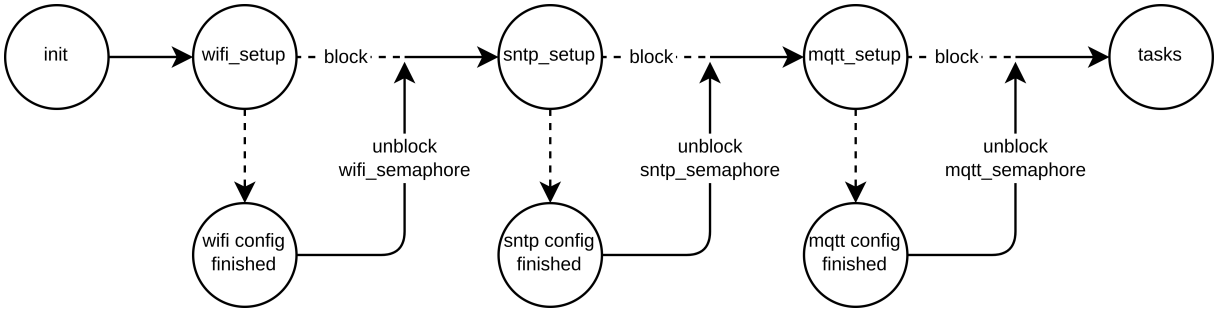


Figura 4.A: Sincronización de tareas proceso principal

Los semáforos se desbloquearán cuando hayan alcanzado los siguientes eventos respectivamente:

- **`wifi_setup()`**: Conexión con la red **Wi-Fi** y con una dirección **IP** asignada
- **`sntp_setup()`**: Sincronización del **reloj** del microcontrolador con el servidor **SNTP**
- **`mqtt_setup()`**: Conexión con el servidor **MQTT** y suscripción al **topic**

Una vez que el microcontrolador libera el hilo principal, **crea** la tarea de obtención de datos meteorológicos **`task_meteo()`**, la involucrada en el muestreo de los rayos **`task_pnct()`** y la encargada de la clasificación y envío de los mensajes a la estación base **`mss_sender()`**.

Como se explica en la sección 3.4, dado que nuestro microcontrolador dispone de dos **CPU**, vamos a reservar el core 1 para la ejecución de la tarea `task_pnct()`, ya que es más susceptible ante los desalojos que pueda realizar el planificador. El resto de tareas, por ser menos relevantes o por no requerir un tiempo de respuesta ajustado, se delegara su ejecución en el core 0. Si nuestro microcontrolador cuenta con un único core, es esencial establecer prioridades a las tareas, siendo las tareas con prioridad más alta las que deben acaparar más tiempo de CPU.

Todo el código mencionado en este capítulo se encuentra también alojado en el repositorio Github del proyecto[33]

4.3. Configuración Wi-Fi

Como explicamos en la sección 2.2.2, nuestro microcontrolador dispone de distintos protocolos de comunicaciones. Para establecer una conexión inalámbrica con nuestro módem hemos creado la función **`wifi_setup()`**[1] que contiene llamadas a métodos de la librería **`"esp_wifi.h"`**. En resumen, esta función realiza una verificación del dispositivo Wi-Fi, configura la conexión y comienza con el proceso de búsqueda y conexión al **AP**[2] mediante el uso del manejador Wi-Fi.

A medida que el manejador **`wifi_event_handler()`**[1] recibe nuevas peticiones, filtra cada tipo de mensaje según su **event_id**. De forma que cuando el evento indica que se ha iniciado (`WIFI_EVENT_STA_STAR`) realiza la conexión mediante la función `esp_wifi_connect()`[1], cuando obtiene la dirección IP (`IP_EVENT_STA_GOT_IP`) desbloquea el semáforo Wi-Fi permitiendo avanzar a la función principal. Cuando el microcontrolador desconecta del **AP**[2] se reinicia el dispositivo ESP32 mediante la función `esp_restart()`.

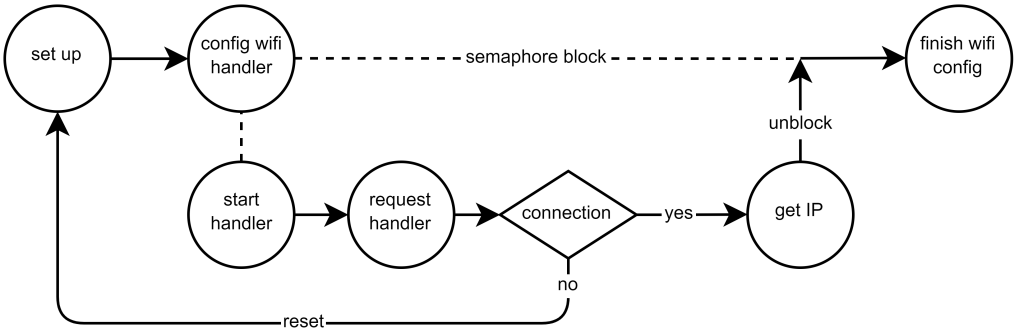


Figura 4.B: Diagrama configuración Wi-Fi

Cabe destacar que el manejador Wi-Fi seguirá recibiendo peticiones después de finalizar la configuración, por lo que si el dispositivo se desconecta de la red Wi-Fi, por ejemplo durante una caída de conexión, mientras realiza otra tarea, podremos reiniciar el proceso y asegurarnos de que se vuelve a conectar.

Del mismo modo, necesitaremos configurar las variables incluidas en el **`sdkconfig`** que hacen referencia al nombre de la red (**`CONFIG_WIFI_SSID`**) y su contraseña (**`CONFIG_WIFI_PASSWORD`**).

4.4. Configuración SNTP

La sincronización del reloj del ESP32 se realiza de forma muy sencilla gracias a la librería **`"esp_sntp.h"`**. Mediante la función `sntp_setservername()`[3] se especifica el servidor SNTP con el que estableceremos la sincronización. En principio se utilizó el dominio `pool.ntp.org` pero después hemos preferido instalar un servidor SNTP de forma local y utilizar la IP local del mismo.

La lógica utilizada para la configuración es similar a la mostrada en el apartado anterior, cuando se inicializa el servidor SNTP se produce un bloqueo del hilo principal hasta que `on_got_time()`[3] devuelva una marca de tiempo sincronizada. La configuración de la zona horaria se realiza mediante la función `setenv()` con el parámetro **`"CET-1CEST-2,M3.5.0/02:00:00,M10.5.0/03:00:00"`**

[1] Función incluida en `wifi.c`
[2] Access Point explicación en la sección 3.5.2
[3] Función incluida en `sntp.c`

4.5. Configuración MQTT

Tras comentar las implicaciones del protocolo MQTT en la sección 3.5.4 , nos centraremos en esta sección en la preparación del microcontrolador para realizar las comunicaciones con nuestra estación base a través de este protocolo.

En la función **`mqtt_setup()`**[4] nos hemos servido de los métodos definidos en la biblioteca **`"mqtt_client.h"`** que habilitan el registro de nuestro cliente en un servidor MQTT. Mediante la variable `esp_mqtt_client_config_t` podremos definir tanto el host destino de nuestros mensajes como el puerto al que van dirigidos, así como el usuario y la contraseña necesarios para establecer la conexión. Estas valores se encuentran referenciados en el **`sdkconfig`**.

Finalmente, será el manejador **`mqtt_event_handler()`**[4] el que procese las peticiones MQTT según su **event_id**, desbloqueando el `mqtt_semaphore` cuando el dispositivo se haya conectado al servidor (`MQTT_EVENT_CONNECTED`) y notificará en el log cuando los eventos indiquen una desconexión, publicación, suscripción a topic, errores, etcétera.

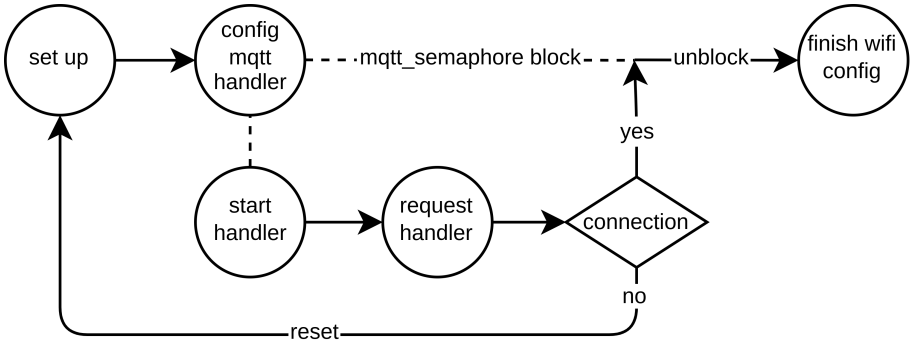


Figura 4.C: Diagrama configuración MQTT

Además, hemos redefinido la función `esp_mqtt_client_publish()` a la función **`mqtt_send_mss()`** y hemos iniciado de forma predefinida los argumentos que necesitan ser reconfigurados en otras partes del código, hemos reducido el número de argumentos de la nueva función a únicamente el mensaje y el **topic**[5] donde se quiere publicar.

Como podemos observar el proceso de sincronización del hilo principal es prácticamente idéntico al mostrado en la figura 4.B de la sección anterior.

[4] Función incluida en `mqtt.c`
[5] Definición de topic en la sección 3.5.4

4.6. Sistema de clasificación y envío de mensajes

El fichero **mss_sender.c** contiene la función encargada de la clasificación, formateo y envío de los datos obtenidos por el resto de tareas.

El sistema estará formado por una cola de mensajes del tipo LIFO que tendrá un máximo de 100 entradas, por lo que si se produce desbordamiento la cola esta configurada para descartar los mensajes más antiguos. La estructura del mensaje será del tipo **telemetry_message** y contendrá los siguientes elementos:

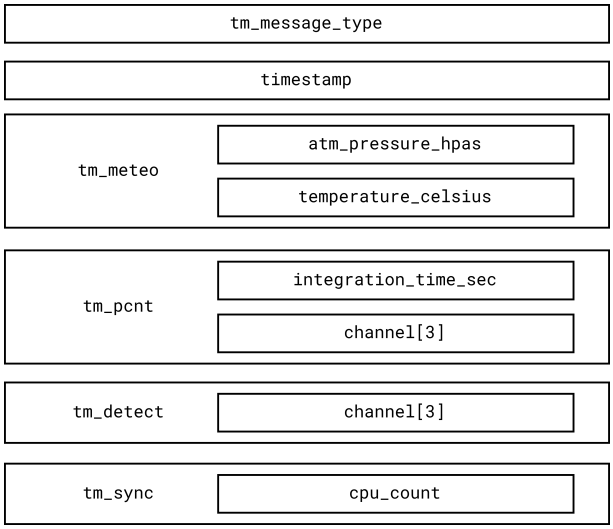


Figura 4.D: Estructura del mensaje telemetry

La función **mss_sender()**[6] comienza su ejecución inicializando el servicio MQTT descrito en el apartado anterior mediante `mqtt_setup()`[6]. A continuación, el proceso entra en estado bloqueado hasta que la cola **telemetry_queue** reciba un mensaje. Dependiendo del tipo de mensaje definido en `tm_message_type` se genera un mensaje JSON con distintos campos en el payload.

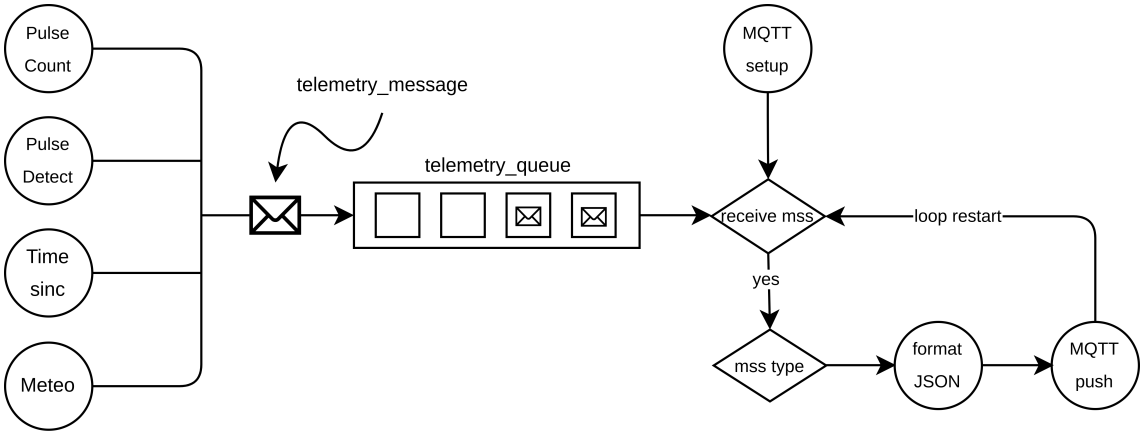


Figura 4.E: Lógica de clasificación y envió de un telemetry_message

[6]Función incluida en `mss_sender.c`

4.7. Obtención de datos meteorológicos

La obtención de los datos meteorológicos a partir del módulo barómetro instalado en la placa se ha realizado gracias a la biblioteca[7] **"bmp180"**. En primer lugar, la tarea **task_meteo()**[8] inicializa los pines **SDA** y **SCL** asociados al módulo BMP180 y crea un telemetry_message del tipo **TM_METEO**. Obtiene la temperatura en Celsius y la presión atmosférica en Pascales e identifica el mensaje mediante un timestamp. Por último envía el mensaje a la cola **telemetry_queue** mediante la función `xQueueSend()`.

4.8. Tareas de detección y monitorización de rayos cósmicos

Las tareas de detección y monitorización en principio se podrían concebir como una única función, ya que si detectamos cada pulso obtendremos el número de pulsos por minuto. Pero esto realmente presenta varios inconvenientes.

La detección, como veremos más adelante, se realiza mediante una interrupción, por lo que es más interesante dividir el trabajo en dos tareas, otorgando más importancia al proceso de detección e invocando el conteo de pulsos con una tarea cada n segundos.

4.8.1. Contador de pulsos: task_pcnt()

Esta tarea tiene como finalidad, el conteo de los rayos cósmicos detectados por unidad de tiempo. Para desarrollar esta tarea se ha utilizado el **contador hardware PCNT**, capaz de registrar el número de flancos ascendentes y/o descendentes de las señales de entrada.

Hemos creado la función **pulse_counter_init()**[9] que, mediante el uso de la librería `"driver/pcnt.h"`, asigna y configura los canales PCNT definidos en el **sdkconfig**. Además, hemos creado la función **get_and_clear()**[9] encargada de leer el canal indicado como parámetro y reinicializarlo a 0. De esta forma, mediante un bucle infinito y la función `vTaskDelay()`, somos capaces de obtener los valores cada n segundos. Una vez obtenidas las cuentas de cada canal, las identificamos mediante una marca de tiempo en μs mediante la función `gettimeofday()`.

Por último, asignamos los valores obtenidos anteriormente a un mensaje `telemetry_message` del tipo **TM_PULSE_COUNT** y lo enviamos a la cola **telemetry_queue** mediante la función `xQueueSend()`.

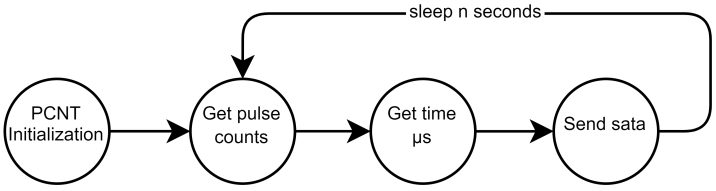


Figura 4.F: Diagrama tarea `task_pcnt()`.

[7] Librería incluida en `esp-idf-lib`
[8] Función incluida en `meteo.c`
[9] Función incluida en `pulse_monitor.c`

4.8.2. Detector de pulsos: `detection_isr_handler()`

Cuando necesitamos obtener el instante en el que un pulso es captado por nuestro microcontrolador, nos percatamos de que el procedimiento de **polling** utilizado en el apartado anterior resulta poco efectivo.

El problema surge al consultar de forma activa el estado de una entrada, ya que el tiempo entre consultas puede ser insuficiente para registrar el pulso como en el caso *a)* de la figura 4.G, o que el tiempo transcurrido desde que se produce el pulso hasta que se detecte, genere una imprecisión como en el caso *b)*. El único suceso válido sería el representado en el caso *c)*.

Pero lo más importante es que en este proceso se desperdicia mucho tiempo de CPU en operaciones prescindibles.

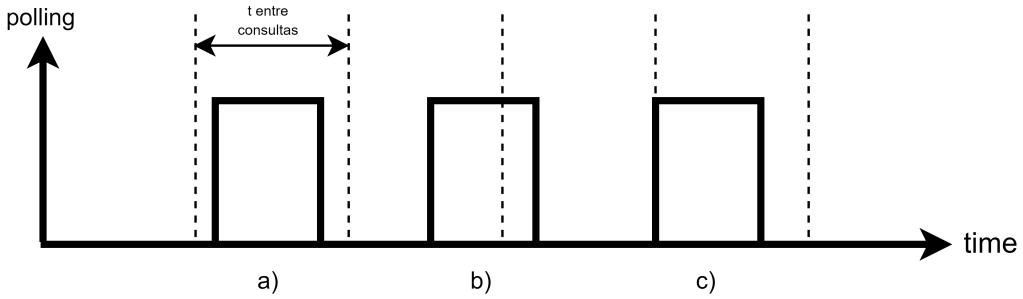


Figura 4.G: Ejemplificación del proceso de polling

En estos casos, resulta más favorable el uso de la **rutina de servicio de interrupción**. Para explicar como funciona este mecanismo, debemos centrarnos primero en definir los elementos que intervienen en el proceso de una interrupción:

- **Interrupción**: Es la acción que desencadena una suspensión temporal del sistema operativo. En nuestro caso, la interrupción se producirá cuando se detecte un flanco de subida/bajada del pulso procedente del monitor de neutrones.
- **IRQ**: Es la señal que indica a la CPU que debe suspender de forma temporal la ejecución de un proceso para pasar a ejecutar la subrutina de servicio de interrupción *ISR*.
- **ISR**: Es el código encargado de resolver la interrupción. En nuestro caso, se ejecutará el código de la función `detection_isr_handler()`[10].

Para configurar la ISR de nuestro microcontrolador, hemos definido la función `init_GPIO()`[10] con todas las llamadas necesarias para inicializar los pines de entrada definidos en el `sdkconfig` y el método de detección mediante flanco de subida y de bajada **`GPIO_INTR_ANYEDGE`**. Además, indicamos que la rutina de interrupción será la función **`detection_isr_handler()`**[10].

En esta función escribiremos el código para caracterizar el flanco detectado, obteniendo el nivel de los canales mediante la llamada **`gpio_get_level()`**. A diferencia de la tarea anterior, la obtención de la marca de tiempo se debe realizar con la función atómica `xthal_get_ccount()`, ya que la ejecución de `gettimeofday()` dentro de una ISR produce excepciones en el sistema. Por último enviaremos el mensaje telemetry_message del tipo **TM_PULSE_DETECTION** a la cola **`telemetry_queue`**.

[10]Función incluida en `pulse_detection.c`

4.9. Actualización mediante OTA

Cuando se diseña un sistema remoto surge la necesidad de distribuir actualizaciones sobre el software *flasheado* en el microcontrolador. Este proceso se realiza normalmente mediante la interfaz **UART**, ya sea mediante un conversor a USB situado en la placa o mediante los puertos TX y RX. En ambos casos, es inevitable realizar la conexión físicamente sobre la placa.

Para resolver este problema, se creó la tecnología **Over-The-Air** que permite la programación de un dispositivo a través de su conexión inalámbrica. Este mecanismo es habitual en los smartphones dada la cantidad de nuevas aplicaciones, servicios y fallos de seguridad que aparecen cada día.

En FreeRTOS, se puede realizar este proceso mediante el uso de la librería **`"esp_https_ota.h"`** que nos permite, mediante la URL en la que se ubica nuestro nuevo binario y un certificado digital, actualizar el microcontrolador de forma segura. El proceso de actualización se puede establecer por medio de una conexión con el dispositivo para indicarle que existe una nueva actualización, o bien, que sea el dispositivo el que compruebe si existe dicha actualización.

Para simplificar el código hemos desarrollado la segunda opción de forma que, para verificar si existe un nuevo binario, primeramente, debe descargar un hash y comprobar que es distinto al de su versión. En caso afirmativo, descarga el nuevo binario, *flashea* la placa y reinicia el dispositivo con el nuevo software.

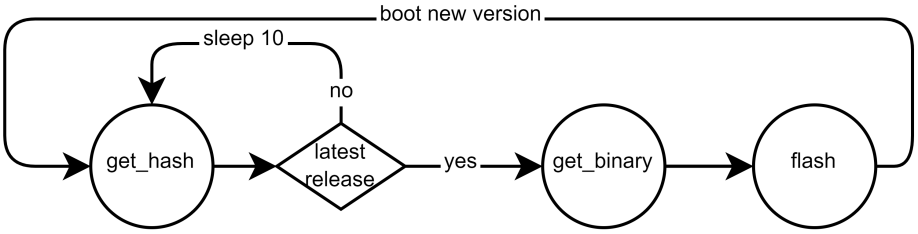


Figura 4.H: Diagrama función task_ota()

El primer *flash* de la placa se realiza de forma física, pero a partir de ese momento, ya podemos actualizar nuestro software de forma remota. El hash que identifica a cada versión se genera mediante la herramienta **sha256sum**, que explicaremos detalladamente en la sección 6.1.1.

Por último, para que se puedan realizar correctamente las peticiones HTTPS, es necesario el uso de un certificado **SSL** que identificará la pagina web a la que nos conectamos, proporcionando además, autenticación de los usuarios y privacidad en las comunicaciones.

Para obtener este certificado necesitamos un navegador web como Chrome o Firefox. En el dominio en el que se ubican el hash y el binario clickamos sobre el candado de la URL, desplegando el menú con los certificados asociados. La raíz de la ruta de certificación es el certificado más seguro, por lo que es el que seleccionaremos para nuestro dispositivo ESP32.

En el fichero **sdkconfig** se encuentran las variables que hacen referencia a las URLs en las que se encuentra el hash (**`CONFIG_OTA_HASH`**) y el binario (**`CONFIG_OTA_URL`**), así como la que hace referencia al tiempo que transcurre entre las verificaciones de los hashes (**`CONFIG_OTA_TIME`**).

Función incluida en `ota.c`

4.10. Estructura de los módulos del programa

El siguiente diagrama muestra todos los archivos involucrados en el proyecto y sus dependencias entre sí. Como podemos observar, hemos creado una librería **"common.h"** que contiene las declaraciones, variables y librerías indispensables para todos los módulos. Esta librería se incluirá posteriormente dentro de otras con preferencias más particulares, como por ejemplo en **"wifi.h"** que además de incluir la librería común se incluye **"esp_wifi.h"**, variables y definiciones específicas del módulo **wifi.c**.

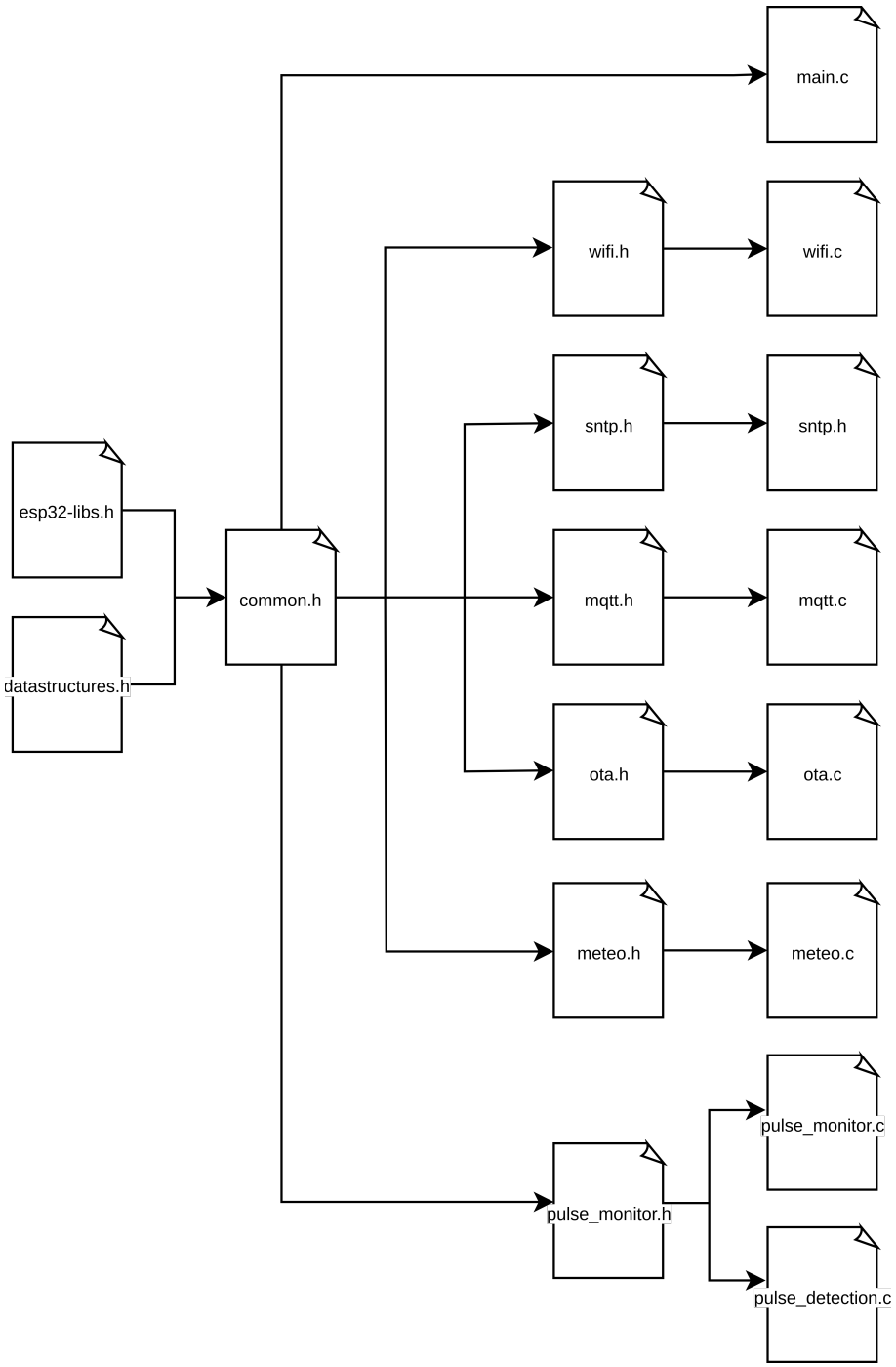


Figura 4.I: Diagrama de los archivos involucrados en el software de adquisición

Capítulo 5

Configuración estación base

La finalidad de este capítulo es la implementación de un servidor que sea capaz de manejar las peticiones proporcionadas por el ESP32 del sistema de adquisición, almacenarlas en una base de datos y generar una interfaz web desde la que se puedan visualizar los datos procesados en gráficas.

En la primera sección se ha dedicado a explicar la estructura de los servicios proporcionados por el servidor y como interactúan entre sí, proporcionando un sistema completo para la obtención de los datos y, además, para la monitorización de los recursos del servidor.

En la sección de InfluxDB se discute la elección de esta base de datos en lugar de las bases de datos clásicas como MySQL y como el agente de servidor Telegraf se relaciona con la base de datos y obtiene las métricas de los servicios que monitoriza. La configuración, como en el resto de apartados, se realizará de forma guiada aportando capturas para su entendimiento.

En la sección de Mosquitto estudiaremos el funcionamiento del servidor MQTT y como procesa los mensajes procedentes del sistema de adquisición. Realizaremos una configuración simple, aunque comentaremos la cantidad de opciones que ofrece este servicio.

Por último, explicaremos como unir Grafana con la base de datos InfluxDB y como orquestar los datos mediante el uso de gráficas, tablas u otros mecanismos de visualización de datos. Generaremos un *dashboard* en el que mostrar las gráficas organizándolas en función del dispositivo del que proceden, simplificando su monitorización y generando alertas para sistemas críticos.

5.1. Estructura

Nuestra estación base se compone de un servidor implementado sobre un **MiniPC** con un procesador Intel(R) Core(TM) i7-10510U, 8 GB de memoria RAM, puerto Ethernet Gigabit y el sistema operativo Debian 11 instalado. Este servidor, como hemos explicado anteriormente, tiene la función de recoger, almacenar y graficar los datos transmitidos por el sistema de adquisición. Los servicios se ejecutarán mediante el uso de Docker y Docker Compose, lo que nos garantiza una rápida implementación a la vez que aporta escalabilidad a nuestro sistema.

La ejecución del archivo `docker-compose.yml` genera la siguiente estructura:

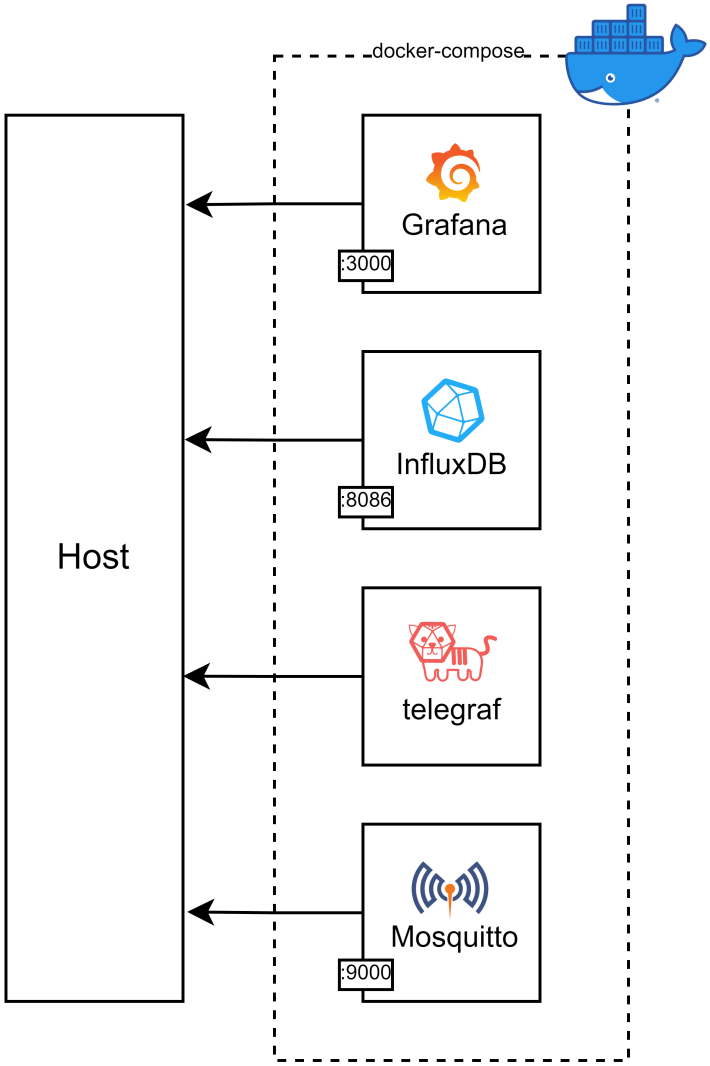


Figura 5.A: Estructura de la estación base

5.2. InfluxDB y Telegraf

InfluxDB es un sistema de gestión de bases de datos de código abierto que, a diferencia de otras clásicas como MySQL, almacena y analiza datos de series temporales. Una *serie temporal* es un conjunto de datos obtenidos en intervalos de tiempo y ordenados cronológicamente, por lo que resultan ideales para tareas de monitorización, almacenamiento de datos de sensores IoT y realizar análisis en tiempo real. Cuenta con un motor de base de datos optimizado para datos de serie de tiempo, un lenguaje de consulta simple y potente (**InfluxQL**), una **API RESTful** para acceder a los mismos desde aplicaciones y servicios y una arquitectura de **escalabilidad horizontal** para permitir el crecimiento de estos y el tráfico.

El uso de InfluxDB es sencillo gracias a que permite su instalación en cualquier sistema operativo, ser ejecutado en una amplia variedad de plataformas y por su interfaz de la línea de comandos (CLI) que facilita la administración de la base de datos. Como es **escalable**, maneja grandes cantidades de datos y tráfico sin problemas, pudiendo añadir más servidores para manejar el crecimiento de los datos y el tráfico gracias a que su arquitectura de escalabilidad es horizontal. Finalmente InfluxDB es una base de datos de código abierto que puede ser usado y modificado libremente por cualquiera. Como la comunidad de código abierto es muy activa existen una gran variedad de recursos disponibles para ayudar a los usuarios a manejar InfluxDB.

Telegraf, desarrollado por InfluxData[34], es un agente servidor de código abierto para la **monitorización** de redes y servidores, diseñado para ser eficiente y modular. Está escrito en el lenguaje de programación **Go**[35] lo que le proporciona potencia y rapidez, pudiendo ser ejecutado en Linux, MacOS y Windows.

Lo más interesante de estos sistemas es la forma en la que son capaces de recopilar y distribuir los datos, formando una simbiosis perfecta.

5.2.1. Configuración de la base de datos y el agente servidor

La configuración del Docker Compose de InfluxDB y Telegraf se resume en el código aportado a continuación extraído del `docker-compose.yml`:

```
  influxdb:
    image: influxdb
    container_name: influxdb
    ports:
      - "8086:8086"

  telegraf:
    image: telegraf
    container_name: telegraf
    user: telegraf:998
    links:
      - influxdb
    volumes:
      - ./telegraf.conf:/etc/telegraf/telegraf.conf:ro
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - INFLUX_TOKEN=jYPEWwAPHRMd1VtPDuIW2pfEBPxcl...
```

En la primera ejecución del `docker-compose.yml` el servidor descarga las imágenes de los dockers de *influxdb* y *telegraf* directamente del repositorio oficial y configura los parámetros de los dockers con las variables seleccionadas. El contenido y utilidad de estas variables se explicará a medida que vayamos configurando el entorno desde la interfaz gráfica.

Una vez hemos desplegado los servicios con Docker Compose, nos dirigimos a la IP local de nuestro servidor, accediendo a través del puerto **8086**. Como podemos observar en la sección de código anterior, en Docker se exponen los puertos de forma que asignamos un puerto del servidor al puerto interno del docker de la siguiente forma `"p-serv:p-docker"`.

Al acceder por primera vez al servidor de InfluxDB deberemos configurar los parámetros de nuestra base de datos. Desde la última versión, las bases de datos se denominan **buckets** que son accesibles mediante los permisos otorgados a la **organización** a la que pertenecen:

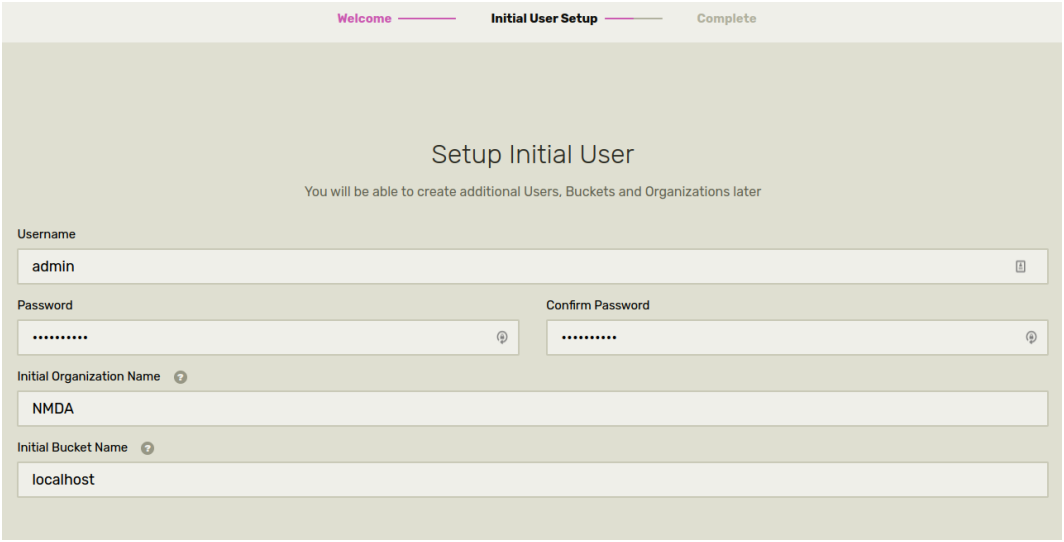


Figura 5.B: Configuración inicial InfluxDB

Desde la sección *Telegraf* de **Load Data** realizaremos la conexión entre la base de datos y el agente de servidor **telegraf**. Al crear una nueva configuración seleccionaremos *System* y *Docker* como sistemas para monitorizar, seleccionando los plugins *cpu*, *disk*, *docker*, *mem*, *net*, *processes*, *swap* y *system*. Estos plugins obtendrán parámetros de los recursos del sistema, permitiendo controlar la carga del sistema y de los servicios que se encuentran en ejecución.

Adicionalmente hemos seleccionado el plugin **mqtt-customer**[36] que nos permite obtener los datos del servidor MQTT directamente. La forma de implementar este plugin es copiar las líneas de código extraídas del repositorio de Github y sustituyendo las variables de *servers* y *topics* por los configurados en la sección 4.5.

Tras realizar la configuración guiada se genera un **token** para acceder al bucket, dicho token debe sustituir al contenido de la variable de entorno **INFLUX_TOKEN** del docker-compose.

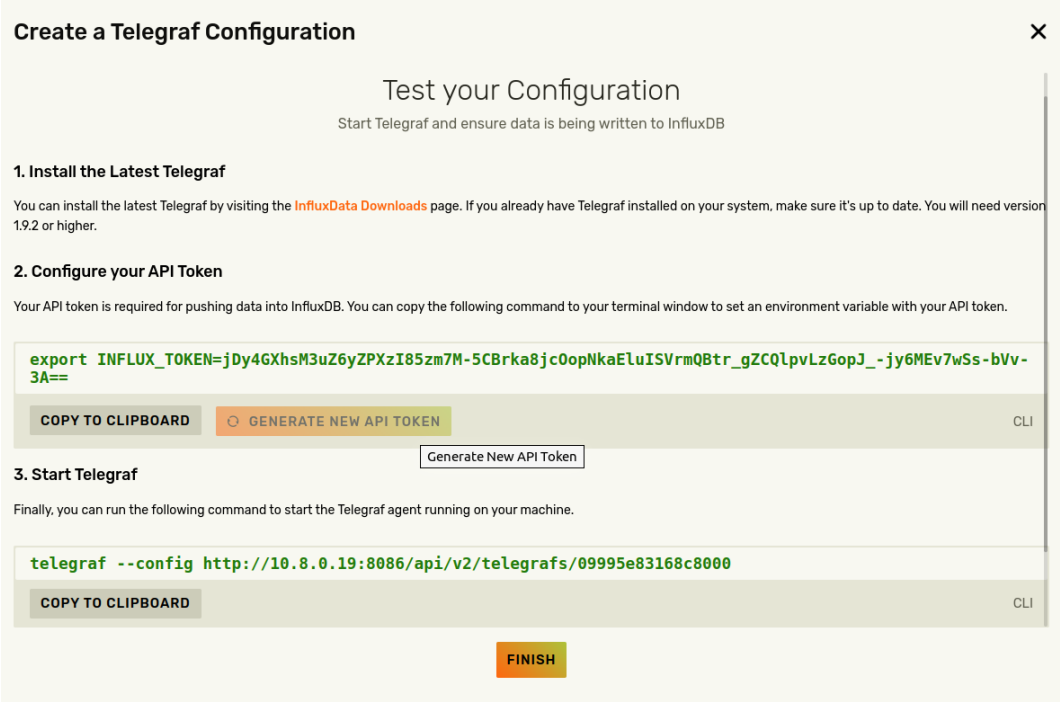


Figura 5.C: Obtención del token necesario para el archivo `docker-compose.yml`

Desde el apartado *Edit Telegraf Config* podemos exportar el archivo de configuración con la configuración por defecto de los módulos.

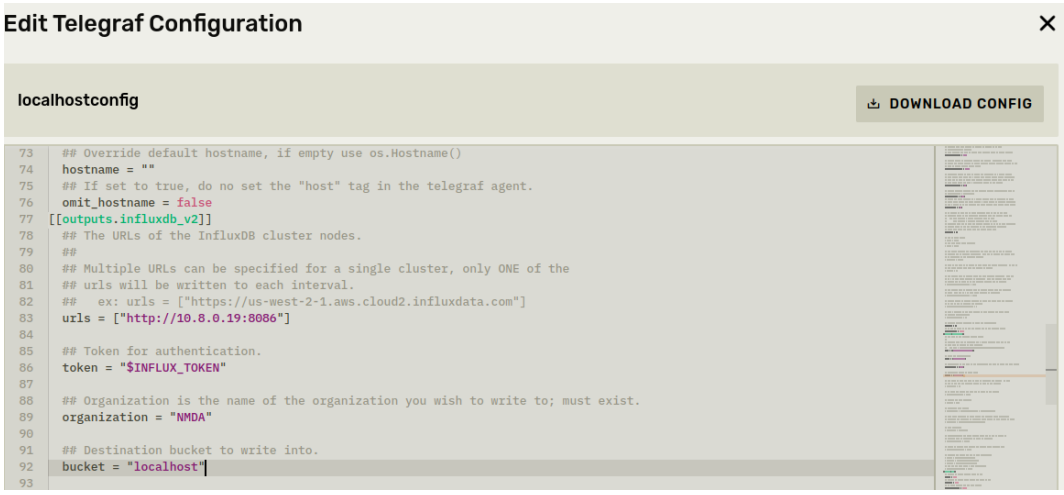


Figura 5.D: Obtención del fichero `telegraf.conf`

5.3. Mosquitto

Mosquitto[37] es un servidor de mensajería MQTT de código abierto, muy ligero y simple que permite una implementación en diferentes dispositivos y sistemas operativos. Además, su mantenimiento es simple, pero aun así logra manipular miles de dispositivos y miles de mensajes simultáneamente. Mosquitto actúa de forma que un cliente MQTT se conecta al servidor **broker** y publica un mensaje en un *topic*, cuando lo recibe el servidor Mosquitto almacena el mensaje y lo reenvía a todos los clientes que están suscritos al *topic*.

5.3.1. Configuración del broker

```
  mqtt:
    image: eclipse-mosquitto
    container_name: mqtt
    links:
      - influxdb
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
    ports:
      - "9000:9000"
```

El docker de Mosquitto obtiene la configuración de la ruta `/etc/telegraf/telegraf.conf`, por lo que mediante la sección **volumes** configurada en el docker-compose podemos copiar el siguiente archivo en dicha ruta:

```
log_type all

listener 9000
allow_anonymous true
```

En esta configuración **`log_type`** se hace referencia a los tipos de mensaje que se desean registrar como *debug*, *error*, *warning*, *notice* o *all* que incluye todos. Este fichero luego puede ser interpretado por InfluxDB o Grafana para la generación de alertas, y para simplificar la configuración.

Mediante la variable **`listener`** que especifica el puerto en el que se van a escuchar las conexiones entrantes, también podemos vincular una IP al agente mediante la opción `bind address/host` o utilizar un *websocket*.

Si deseamos establecer una contraseña para nuestro servidor MQTT, deberemos cambiar `allow_anonymous` a `False` y añadir el fichero donde se encuentre dicha contraseña con `password_file /etc/telegraf/pass` por ejemplo. Este archivo se puede incluir también en la sección **volumes** antes mencionada.

Esta resulta la configuración más básica que podemos realizar, pero gracias al manual de configuración de Mosquitto[38] podemos establecer infinidad de características desde establecer colas con calidad de servicio (QoS) hasta listas de usuarios permitidos (whitelist) o bloqueados (blacklist).

5.4. Grafana

Grafana[39] es un servidor web con software de código abierto que visualiza y analiza datos a partir de una base de datos. Sirve para monitorizar diferentes fuentes de datos, redes, servidores, bases de datos, sistemas operativos, aplicaciones y para crear dashboards personalizados (tablas y gráficos que muestran datos en tiempo real). El lenguaje de Grafana es de consulta SQL de manera que extrae datos de la base de datos y los representa en un gráfico.

Las ventajas de Grafana frente a otras herramientas de visualización de datos son que es compatible con una gran variedad de fuentes con datos distintos, así logra obtener múltiples datos de diferentes fuentes y los visualiza en un único sitio, es personalizable al permitir que se puedan crear dashboards para visualizar los datos según las preferencias o necesidades e incluso es posible incluir plugins y widgets que mejoran la funcionalidad del dashboards creado, el interfaz de Grafana es intuitiva por lo que es fácil y rápido configurar dashboards. Por último se pueden compartir, visualizar e interactuar los dashboards con otros usuarios.

5.4.1. Configuración de la interfaz web

La configuración del Docker Compose de Grafana se resume en el código aportado a continuación extraído del `docker-compose.yml`:

```
  grafana:
    image: grafana/grafana-enterprise
    container_name: grafana
    links:
      - influxdb
    ports:
      - "3000:3000"
```

La configuración se realiza desde el navegador a través del puerto **3000**. En primer lugar, accedemos al servidor de Grafana mediante el usuario y contraseña *admin*. Tras realizar el loggin será obligatorio cambiar la contraseña de nuestro usuario admin.

Desde la sección **Data Source** seleccionamos como fuente de datos InfluxDB y sustituimos los campos de *URL* por la IP de nuestro servidor InfluxDB y su puerto, *token* por el obtenido en la figura 5.C y *organization* y *bucket* por los elegidos en la configuración inicial de la base de datos.

Para comprobar que la configuración introducida funciona correctamente debemos obtener el tick verde de la figura 5.E al pulsar en el botón *Save & test*.

En Grafana la representación de los datos se organiza con la ayuda de los **dashboards**, que funcionan como una pizarra en la que puedes añadir tablas, gráficas o otras representaciones de datos.

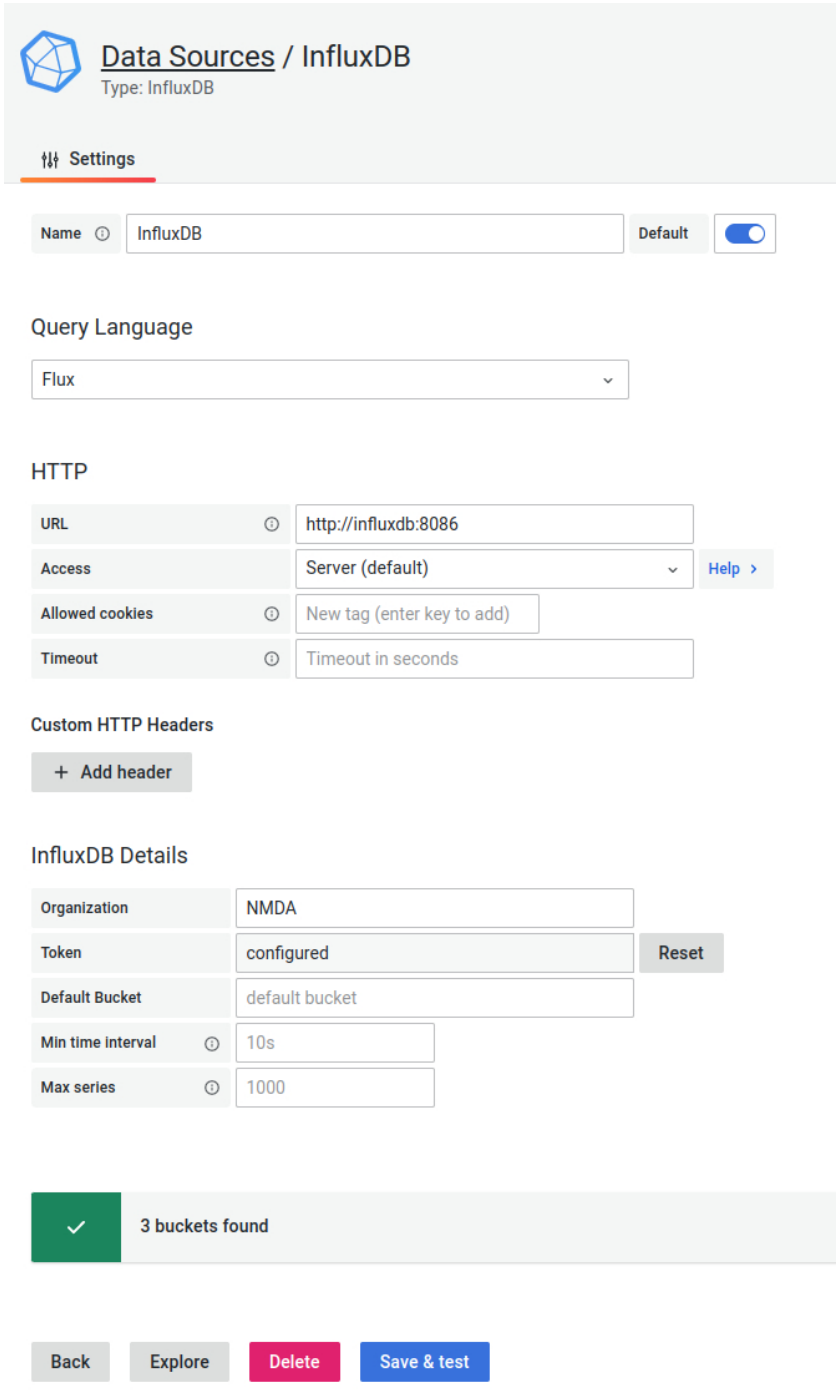


Figura 5.E: Configuración inicial InfluxDB

Por último, para la generación de tablas usaremos el lenguaje **Flux** desarrollado por InfluxData, que nos permite filtrar los datos en función del *bucket*, el tipo de medida y el tipo de dato. Además, la apariencia de la gráfica se puede modificar cambiando el tipo de representación, los colores, su leyenda, las unidades con las que trabajan los datos, etcétera. Todas estas opciones hacen de Grafana un entorno ideal para generar nuestros gráficos.

Figura 5.F: Lenguaje Flux

Capítulo 6

Componentes adicionales

6.1. Integración y distribución continua: CI/CD

CI/CD es un método de distribución de aplicaciones a clientes mediante el uso de procesos de automatización de las etapas de desarrollo de las aplicaciones. La integración continua se refiere a la incorporación de código nuevo en una aplicación. La distribución continua garantiza que la implementación del nuevo código se realice, aplicándose después de la automatización.

6.1.1. Github Actions

GitHub[40] es un servicio basado en la nube para almacenar y administrar código, así como mantener un seguimiento y compartir proyectos. Estos proyectos se generan en repositorios y los cambios realizados se guardan con *commits*. Para destacar los *commits* más importantes podemos utilizar los ***tags***, o etiquetas, para determinar las nuevas versiones del proyecto. Github, además, ofrece una plataforma de integración y despliegue continuo que permite automatizar la compilación, las prueba y el despliegue de tu software.

Mediante un flujo de trabajo diseñado con el lenguaje YAML y ubicado en la ruta `.github/workflows` de nuestro repositorio, podremos definir el entorno, las tareas y los eventos que desencadenarán su ejecución *trigger*.

Hemos establecido como evento la publicación de un tag que comience con v, por ejemplo *v1.1.0* o el tag ***latest***.

A continuación hemos seleccionado como entorno el sistema operativo *Ubuntu*, ejecutándose nuestras tareas (steps) sobre el contenedor `espressif/idf v4.4`. La ventaja de este entorno es que no es necesaria la instalación del set de herramientas de Espressif, por lo que simplificamos en gran medida las tareas y delegamos la configuración del entorno al distribuidor oficial.

```
jobs:
    builder:
        runs-on: ubuntu-latest
        container: espressif/idf:v4.4
        steps:
          - ...
```

La lógica que hemos implementado sigue el esquema de la figura 6.A, de forma que se realizará un *checkeo* del código y se compilará mediante los comandos de IDF explicados en la sección 3 del capítulo 3.1. Por último actualizará o creará un *latest release* con el binario generado en el bloque de compilación y un hash asociado a esta versión.

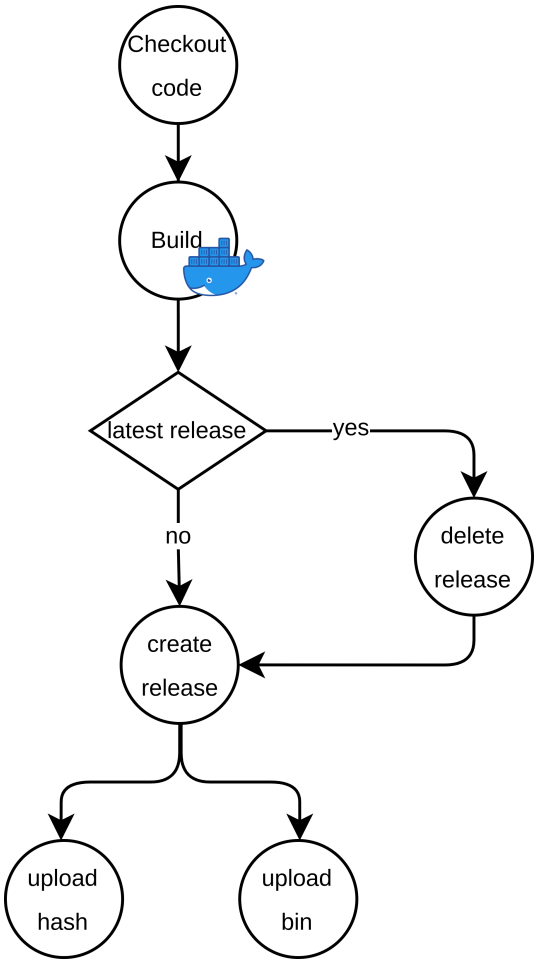


Figura 6.A: Lógica Github Actions

Estas tareas se realizarán de forma segura y sin revelar datos o información sensible gracias al uso de los secretos de Github. De esta forma, hemos configurado un **token** en Github para actualizar el *release* y subir los nuevo ficheros y lo hemos almacenando en la variable `secrets.TOKEN`.

Además, dado que el *menuconfig* genera un archivo **sdkconfig** con las contraseñas del Wi-Fi y MQTT en texto plano se ha evitado añadir este archivo al control de versiones y en su lugar hemos comprimido dicho archivo y se ha almacenado en la variable `secrets.SDKCONFIG`. En el proceso de compilación se usará esta variable descomprimida para generar los binarios configurados, pero no se mostrán nunca estas contraseñas.

Todo el código mencionado en este capítulo se encuentra también alojado en el repositorio Github del proyecto[33]

6.2. Implementación en Google Cloud

Google Cloud[41] es un servicio ofrecido por Google que permite almacenar en la nube gran variedad de herramientas sin necesidad de hardware o software adicional. Destaca por su escalabilidad y flexibilidad, adaptando las necesidades del usuario o la empresa de manera sencilla. También se caracteriza por la seguridad a la hora de proteger datos y su precio competitivo.

Hemos elegido esta plataforma frente a AWS (Amazon Web Services) por diversas razones. En primer lugar, debido a la facilidad de uso e interfaz más intuitiva. En segundo lugar, Google Cloud ofrece un mejor soporte técnico y una mayor cantidad de documentación. Por último, como hemos nombrado antes, por su escalabilidad y precio.

6.2.1. Google Compute Engine

Google Compute Engine (GCE) es una infraestructura de Google Cloud Platform que ofrece numerosos servicios para lanzar máquinas virtuales según las necesidades de cada usuario. GCE permite ejecutar gran cantidad de máquinas virtuales simultáneamente y administrar recursos de manera eficiente, para así ahorrar tanto tiempo como dinero.

Nuestra estación base se ha configurado sobre un servidor conectado de forma local con el resto de dispositivos, este modelo hace que la escalabilidad del sistema sea muy complicada, ya que no podremos situar distintos sistemas de adquisición geográficamente distribuidos y conectarlos al mismo servidor.

La solución planteada es utilizar la infraestructura de Google Cloud, alojando nuestro servidor en la nube, de forma que cada sistema de adquisición pueda conectarse de forma remota y enviar los datos recopilados.

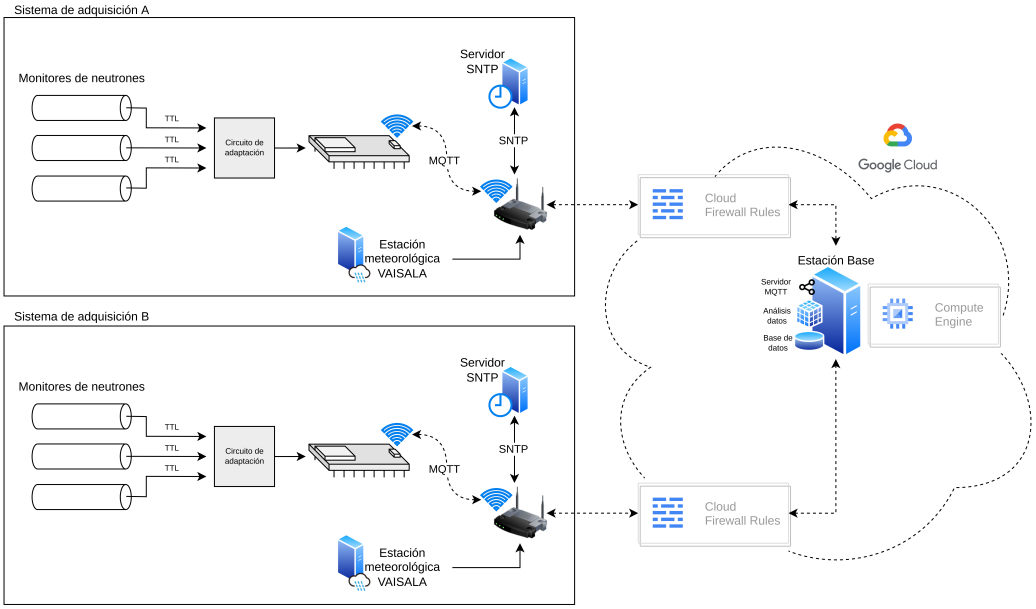


Figura 6.B: Modelo escalabilidad del sistema

Cabe mencionar que la conexión de los sistemas a la nube de Google se puede filtrar mediante el uso de su propio *firewall*, simplemente deberemos configurar una regla para el puerto 9000 del servidor de Mosquitto y el puerto http/https. Hay que tener en cuenta que desde ese momento el puerto se encontrará disponible para todo Internet, por lo que debemos securizar la conexión mediante usuario y contraseña o el uso de claves simétricas. Adicionalmente, deberemos modificar el puerto expuesto en el `docker-compose.yml` sustituyendo `"3000:3000"` por `"80:3000"`, de esta forma visualizaremos la interfaz web desde le puerto HTTP en lugar del asignado por defecto.

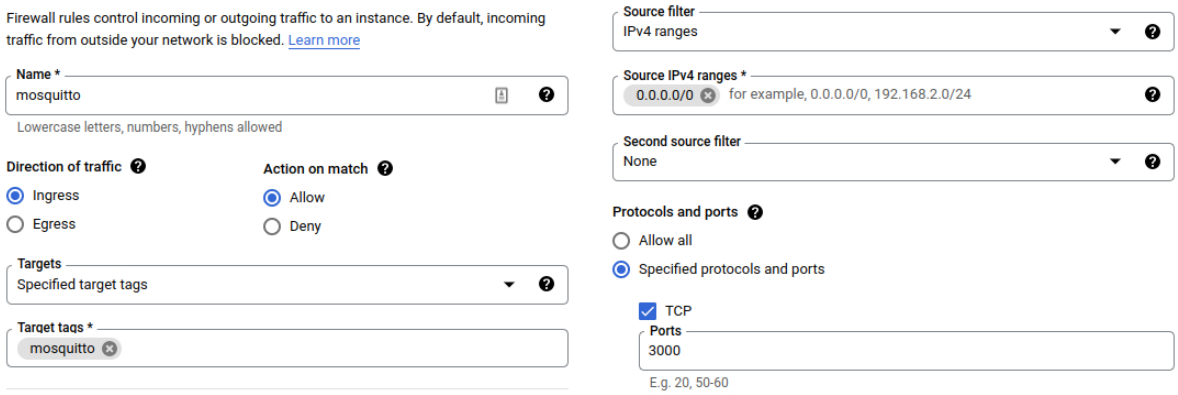


Figura 6.C: Creación de regla firewall en Google Cloud

Este servidor debe tener instalado *docker* y *docker-compose*, de forma que, levantando el servicio configurado en el `docker-compose.yml`, simplemente reproduciendo la configuración mostrada en el capítulo 5 habremos transferido el servicio a Google.

Las especificaciones de la máquina virtual de Google se muestran en la figura 6.D, además de las reglas *firewall* configuradas. La herramienta de estimación del coste de la instancia de Google nos puede ayudar a ajustar los recursos del servidor teniendo en cuenta el gasto del servicio.

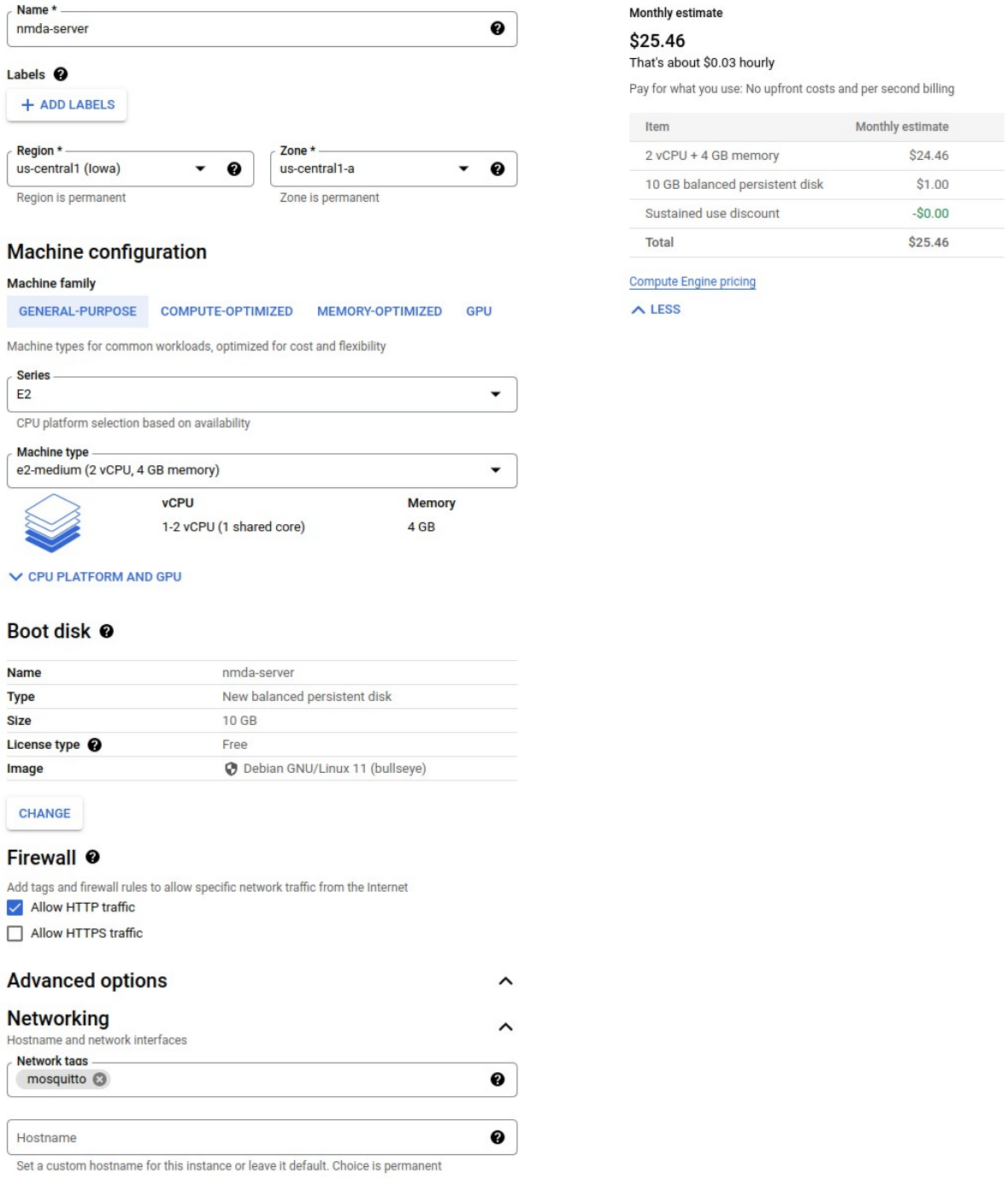


Figura 6.D: Creación de Google Cloud Compute instance

Capítulo 7

Resultados

En este capítulo se va a mostrar el resultado de los distintos bloques del sistema de adquisición, de forma que sirva tanto como un reflejo del funcionamiento final del sistema como una comprobación en caso de replicar el modelo.

En primer lugar explicaremos los pasos necesarios para realizar la compra de nuestros circuitos impresos en la empresa JLCPCB, generando los archivos necesarios en Kicad 6 y eligiendo los parámetros de configuración de la página web. Una vez hayamos obtenido la placa realizaremos el montaje físico de la placa, soldando los componentes electrónicos.

En relación con el software, primeramente comentaremos la salida obtenida al realizar un *flasheo* directamente en la placa, como se ha ilustrado en la figura 3.B del capítulo 3. Después realizaremos una actualización del firmware del dispositivo mediante OTA, visualizando además el funcionamiento de la integración continua en Github Actions.

Por último, desde Grafana visualizaremos los datos obtenidos del monitor y de la monitorización del servidor.

7.1. Hardware

JLCPCB[42] es una empresa de manufacturación de PCB fundada en 2006 en China que fabrica sistemas empotrados por encargo. Es conocida por sus productos de calidad y bajo coste. Fabrica PCB de una, dos y varias capas y cada una de ellas son sometidas a un control de calidad antes de ser enviadas. Como ya explicamos y anticipamos en el capítulo 2 la creación del circuito impreso se encargó a esta empresa, que en pocas semanas recibimos en nuestro domicilio. En Kicad 6 generamos los archivos necesarios en el proceso de fabricación desde el menú *Archivo/Salidas de fabricación/Gerbers*.

El ensamblado de la placa se realizó siguiendo el esquema del apéndice A, soldando cada componente con la ayuda de una estación de soldadura, estaño y pasta térmica. Además, para no soldar componentes sensibles, como el ESP32, los módulos o el LM339N, directamente a la placa se han soldado unos *sockets* para insertar dichos componentes, facilitando su extracción.

La siguiente figura muestra el resultado del ensamblaje:

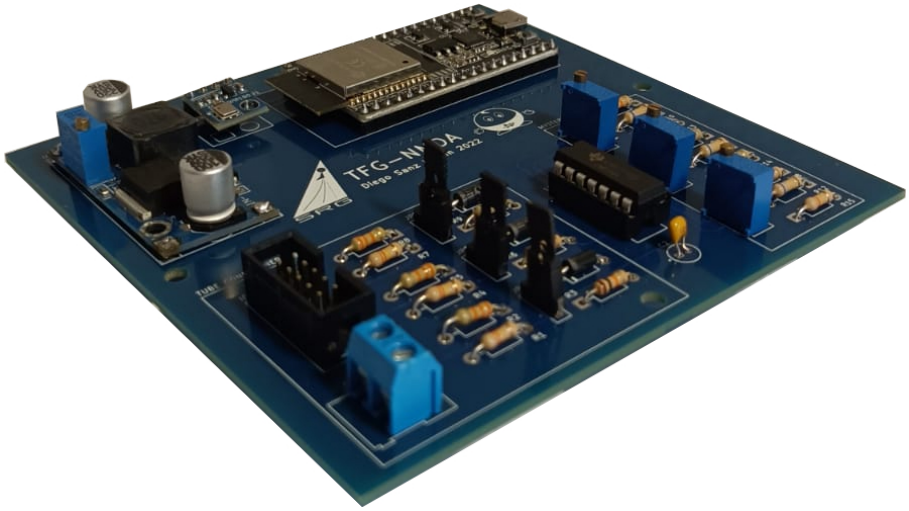


Figura 7.A: Placa PCB ensamblada

7.2. Software de Adquisición

7.2.1. Ejecución del software

Las capturas mostradas en esta sección son el resultado de la ejecución del siguiente comando, que como vimos en capítulos anteriores, *flashea* nuestro código a la placa.

```
$ idf.py flash monitor
```

La ejecución de nuestro programa comienza con la configuración de los protocolos Wi-Fi y SNTP, como se puede observar en la figura 7.B, cuando nuestro microcontrolador obtiene su dirección IP local el semáforo `wifi_semaphore` se desconecta. De igual modo ocurre con SNTP al obtener una fecha sincronizada.

Figura 7.B: Resultado de la configuración de Wi-Fi y SNTP

Después de que haya terminado la inicialización de todos los recursos el hilo principal crea las tareas asignándoles sus *cores* correspondientes, como vimos en la sección 4.2. La siguiente figura muestra el resultado de la ejecución de las tareas, mostrándose la obtención de la temperatura y la presión atmosférica, los pulsos detectados en cada canal y como se publican los mensajes mediante el protocolo MQTT.

```
I (4827) METEO: is running on 0 Core
I (4827) MSS_SEND: is running on 0 Core
I (4827) METEO: Initializing BMP180 meteo moule
I (4827) METEO: Waiting for NTP
I (4837) MQTT: INIT CLIENT
I (4847) MSS_SEND: MQTT initializing
I (4847) MONITOR: is running on 1 Core
I (10877) MONITOR: CH1: 0 pulses per 10 secs
I (10877) MONITOR: CH2: 0 pulses per 10 secs
I (10877) MONITOR: CH3: 0 pulses per 10 secs
I (10877) MONITOR: Sleeping for 10 s
I (10877) MSS_SEND: Publishing message
I (11223) METEO: BMP180 meteo module reports 24.299999 C and 95278 hPas
I (11447) METEO: NTP ready. Starting read loop
E (21377) MQTT: MQTT_EVENT_BEFORE_CONNECT
I (21377) MQTT: MQTT_EVENT_CONNECTED
I (21377) MQTT: MQTT_EVENT_PUBLISHED, msg_id=3746
I (21377) MQTT: MQTT_EVENT_PUBLISHED, msg_id=36813
I (22887) MONITOR: CH1: 151 pulses per 10 secs
I (22887) MONITOR: CH2: 128 pulses per 10 secs
I (22887) MONITOR: CH3: 136 pulses per 10 secs
I (23377) MQTT: MQTT_EVENT_PUBLISHED, msg_id=47115
I (30897) MONITOR: CH1: 132 pulses per 10 secs
I (30897) MONITOR: CH2: 143 pulses per 10 secs
I (30897) MONITOR: CH3: 129 pulses per 10 secs
I (30897) MONITOR: Sleeping for 10 s
I (30897) MSS_SEND: Publishing message
I (31507) MQTT: MQTT_EVENT_PUBLISHED, msg_id=27705
I (40907) MONITOR: CH1: 150 pulses per 10 secs
I (40907) MONITOR: CH2: 135 pulses per 10 secs
I (40907) MONITOR: CH3: 121 pulses per 10 secs
I (40907) MONITOR: Sleeping for 10 s
I (40907) MSS_SEND: Publishing message
```

Figura 7.C: Ejecución de las tareas

7.2.2. Ejecución de CI y actualización mediante OTA

Para realizar una actualización a través de internet mediante OTA, es necesario realizar un nuevo tag *latest*. El **workflow** de Github Actions se encarga de realizar la compilación en la nube y de generar el binario y el nuevo hash.

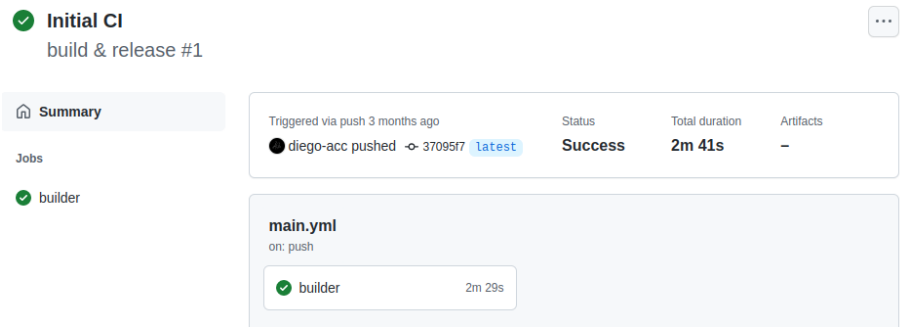


Figura 7.D: Workflow de Github

Al terminar la ejecución del **workflow** podemos visualizar el resultado de cada *step* y el tiempo que han tardado, así como los *logs* asociados a cada una de ellas. Como podemos intuir, el proceso de compilación del binario es la tarea que más tiempo requiere.

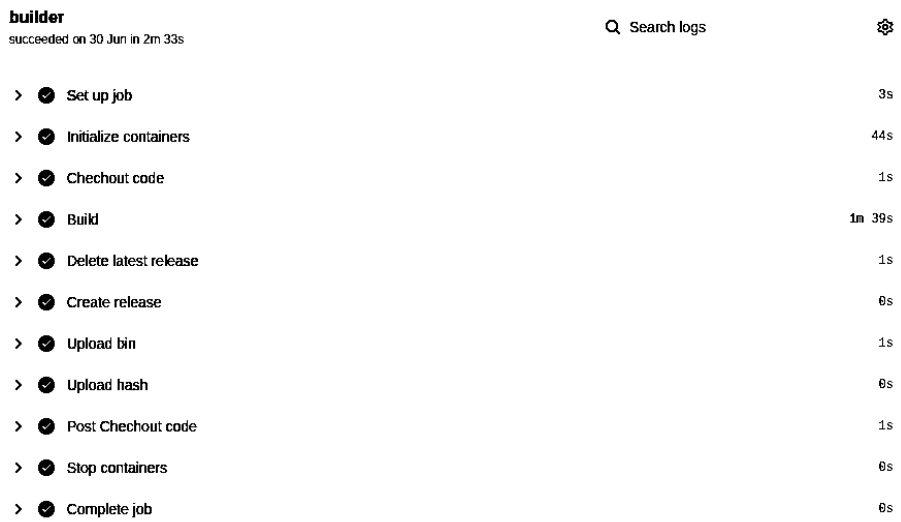


Figura 7.E: *Steps* del workflow

Desde la sección **releases** de nuestro proyecto de Github se obtiene el resultado final de la integración y distribución continua. Desde esta página descargará nuestro microcontrolador el binario y comparará los hashes. Además, se incorpora el código asociado a este tag.

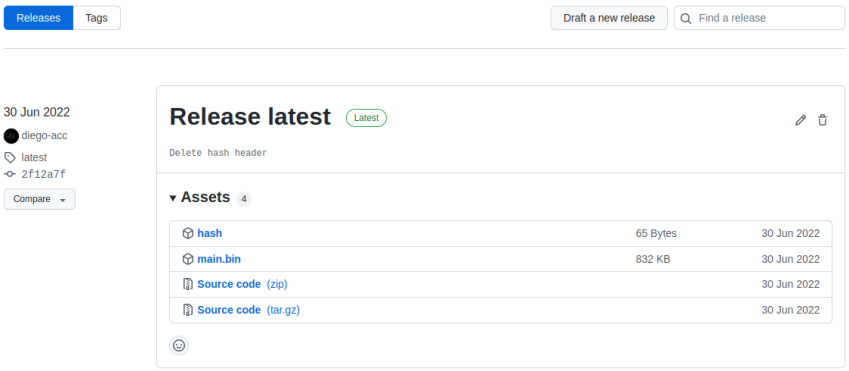


Figura 7.F: *Releases* del repositorio de Github

Ahora que Github tiene almacenada una versión de código superior a la implementada en el microcontrolador, la tarea `task_ota()` verifica que el *hash* guardado es distinto y se produce la actualización del *firmware*. El ESP32 descarga el nuevo binario (*Reciving data*), lo almacena en la memoria y reinicia el microcontrolador con la siguiente versión.

```
I (5066) OTA: is running on 0 Core
E (5096) MQTT: MQTT_EVENT_BEFORE_CONNECT
I (5106) OTA: OTA verify hash
I (5066) MONITOR: is running on 1 Core
I (5126) MQTT: MQTT_EVENT_CONNECTED
I (5116) MONITOR: CH1: 0 pulses per 10 secs
I (5126) MONITOR: CH2: 0 pulses per 10 secs
I (5126) MONITOR: CH3: 0 pulses per 10 secs
I (5136) MONITOR: Sleeping for 8 s
I (5146) METEO: BMP180 meteo module reports 27.000000 C and 95267 hPas
I (5146) METEO: Waiting for NTP
I (5146) METEO: NTP ready. Starting read loop
I (5186) MSS_SEND: Publishing PULSECOUNT
I (5226) MSS_SEND: Publishing METEO
I (8376) OTA: Receiving data
I (8646) OTA: Disconnecting
I (8656) OTA: Finished
I (9626) OTA: Receiving data
I (9866) OTA: Receiving data
I (9866) OTA: new version available
I (9866) OTA: Finished
I (9866) OTA: Disconnecting
I (13136) MONITOR: CH1: 115 pulses per 10 secs
I (13136) MONITOR: CH2: 101 pulses per 10 secs
I (13136) MONITOR: CH3: 93 pulses per 10 secs
I (13136) MONITOR: Sleeping for 10 s
I (13146) MSS_SEND: Publishing PULSECOUNT
I (23036) OTA: Receiving data
I (23146) MONITOR: CH1: 132 pulses per 10 secs
I (23146) MONITOR: CH2: 143 pulses per 10 secs
I (23146) MONITOR: CH3: 133 pulses per 10 secs
I (23146) MONITOR: Sleeping for 10 s
I (23156) MSS_SEND: Publishing PULSECOUNT
I (23276) OTA: Disconnecting
I (24176) OTA: Receiving data
I (24426) esp_https_ota: Starting OTA...
I (24426) esp_https_ota: Writing to partition subtype 16 at offset 0x110000
I (24436) OTA: Receiving data
I (24506) OTA: Receiving data
I (24506) OTA: Receiving data
I (24516) OTA: Receiving data
...
I (40916) OTA: Receiving data
I (41296) OTA: Receiving data
I (41306) OTA: Receiving data
I (41466) esp_image: segment 0: paddr=00110020 vaddr=3f400020 size=1f8d8h (129240) map
I (41506) esp_image: segment 1: paddr=0012f900 vaddr=3ffb0000 size=00718h (  1816)
I (41506) esp_image: segment 2: paddr=00130020 vaddr=400d0020 size=972f4h (619252) map
I (41756) esp_image: segment 3: paddr=001c731c vaddr=3ffb0718 size=0319ch ( 12700)
I (41766) esp_image: segment 4: paddr=001ca4c0 vaddr=40080000 size=15ac0h ( 88768)
I (41806) esp_image: segment 5: paddr=001dff88 vaddr=50000000 size=00010h (    16)
I (41806) OTA: Disconnecting
I (41816) OTA: Disconnecting
I (41816) esp_image: segment 0: paddr=00110020 vaddr=3f400020 size=1f8d8h (129240) map
I (41876) esp_image: segment 1: paddr=0012f900 vaddr=3ffb0000 size=00718h (  1816)
I (41886) esp_image: segment 2: paddr=00130020 vaddr=400d0020 size=972f4h (619252) map
I (42146) esp_image: segment 3: paddr=001c731c vaddr=3ffb0718 size=0319ch ( 12700)
I (42156) esp_image: segment 4: paddr=001ca4c0 vaddr=40080000 size=15ac0h ( 88768)
I (42196) esp_image: segment 5: paddr=001dff88 vaddr=50000000 size=00010h (    16)
I (42256) OTA: OTA flash succsessfull.
restarting in 5 seconds

entry 0x40080694
I (29) boot: ESP-IDF v4.4.1-117-g8c8f700c1d 2nd stage bootloader
I (29) boot: compile time 11:44:37
I (29) boot: chip revision: 1
I (33) boot_comm: chip revision: 1, min. bootloader chip revision: 0
I (40) boot.esp32: SPI Speed      : 40MHz
I (45) boot.esp32: SPI Mode       : DIO
I (49) boot.esp32: SPI Flash Size : 4MB
...
```

Figura 7.G: Ejecución de la tarea `task_ota()`

7.3. Interfaz Web

Finalmente, en esta sección visualizaremos mediante el uso de gráficas los datos obtenidos por el sistema de adquisición de datos conectado a dos monitores de neutrones BP28 Chalk-River.

En la figura 7.H hemos representado las siguientes gráficas:

Figura 7.H: Datos obtenidos por el sistema de adquisición

- **Monitor Counts Graph**: Esta gráfica muestra el número de cuentas que han obtenido los monitores cada 10 segundos. Los resultados revelan que el número de **cuentas por minuto** es del orden de 720-840cpm, lo que es un resultado acertado en la localización geográfica en la que se encuentra nuestro monitor. En la base Antártica, debido a la baja densidad atmosférica, las cuentas son del orden de 10000cpm.
- **Monitor Counts Pie Chart**: En este gráfico circuilar se ha representado la media de cuentas por cada canal, esto es necesario para saber si existe algún problema hardware o software que desequilibre el número de cuentas de los monitores.
- **Monitor Counts Bar Chart**: Dado que en la primera gráfica no se visualiza de forma directa el número de cuentas cada 10 segundos, resulta más útil generar un gráfico de barras que además representa su proporción, siendo el rango de 140-150cpm los valores más detectados.
- **Hut Temp Atmospheric Pressure**: En estas gráficas se representan la temperatura y la presión obtenidas del módulo BMP180, que como se puede observar cuando se supera la temperatura de 39ºC deja de funcionar. Es por esta y otras razones, que el barómetro VAISALA es imprescindible en este proyecto.

Las siguientes gráficas representan los parámetros capturados por el agente servidor Telegraf. Estos datos son clave para el buen mantenimiento del servidor, ya que si detectamos que por ejemplo la CPU está sobrecargada, tiene una temperatura crítica o falta espacio en el disco, deberemos realizar cambios hardware en la estación base.

Figura 7.I: Monitorización del MiniPC

Del mismo modo, se ha representado la carga de cada Docker, el porcentaje de espacio que utilizan y el tiempo que llevan ejecutándose de forma ininterrumpida. Todo esto es útil para verificar el correcto funcionamiento del servidor y que no se hayan producido reinicios.

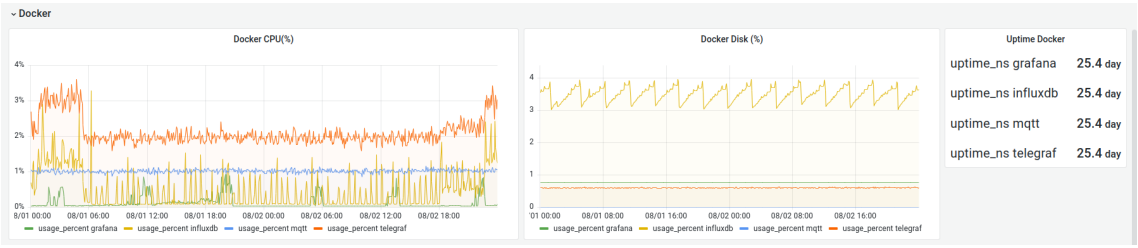


Figura 7.J: Monitorización del servidor Docker

Capítulo 8

Conclusiones y líneas futuras

En conclusión, este trabajo de fin de grado ha consistido en la elaboración de un sistema de adquisición de datos para el monitor BP28 Chalk-River. El sistema permite visualizar en tiempo real gráficas que representan el número de rayos cósmicos detectados por minuto.

Ha sido muy interesante el estudio de los rayos cósmicos. Comprender cómo Victor Hess descubrió por primera vez en 1912 la existencia de la radiación ionizante en las capas superiores de la atmósfera y como esto desembocó en el estudio de la detección de estas partículas y la creación de una base de datos mundial para almacenar información sobre multitud de detectores repartidos geográficamente en el mundo. Además,

El desarrollo del circuito de adquisición ha puesto en práctica todos los conocimientos obtenidos de las asignaturas de electrónica cursadas durante la carrera, que me han permitido crear y simular un circuito de alta frecuencia, para posteriormente generar un esquemático[A] y un PCB[B] con el software Kicad 6. En este proceso, he analizado las distintas formas de fabricación de los circuitos impresos obteniendo, finalmente, un circuito de acabado profesional manufacturado por JLCPCB, sobre el que he soldado todos los componentes.

Sobre este circuito se ha desarrollado el software de adquisición gracias a la plataforma Espressif IDF y el sistema operativo FreeRTOS. Hemos establecido los objetivos de nuestro sistema y hemos implementado en distintas funciones las tareas que debe realizar nuestro microcontrolador. El desarrollo de un sistema en tiempo real ha requerido de los conocimientos en multiprogramación obtenidos durante la asignatura de Sistemas Operativos y los adquiridos en el lenguaje C en las asignaturas de Programación(C) y Programación Avanzada(C++).

La estación base se compone de un MiniPC que obtiene datos del sistema de adquisición a través del protocolo MQTT. El diseño y construcción del servidor se ha generado mediante la tecnología Docker que además de simplificar el proceso de instalación nos aporta escalabilidad al sistema. Las técnicas de monitorización usadas en el MiniPc y el servidor Docker han sido estudiadas en la asignatura de Gestion y Administración de Redes.

Por último, el modelo propuesto de integración y distribución continua permite que el sistema sea escalable. Esta tecnología, desconocida hasta el momento para mí, me ha resultado fascinante y con gran potencial para el mundo IoT. Por otro lado, la implementación de nuestro sistema en Google Cloud se ha visto simplificada gracias a los conocimientos adquiridos durante la asignatura Sistemas Operativos Avanzados y mis prácticas externas.

Lineas futuras

Para el presente Trabajo de Fin de Grado se proponen los siguientes puntos:

- Incorporar el módulo de Arduino **SDCH** para almacenar los datos obtenidos también de forma local. Esto puede ser útil si se producen desconexiones durante periodos muy largos de tiempo.
- Establecer un mecanismo de gestión de *logs* con **logrotate** y una rutina crontab. De esta forma realizaremos una limpieza de *logs* cada cierto tiempo.
- Monitorizar el microcontrolador ESP32 mediante la implementación del protocolo SNMP o mediante un canal MQTT de notificaciones. Un parámetro interesante es **uptime**, que nos aportará el tiempo de ejecución ininterrumpida del dispositivo y con el que verificaremos que el dispositivo no ha sufrido reinicios.

Capítulo 9

Presupuesto

Concepto	Cantidad	Importe
Caja derivación EX161	1	5,67 €
Circuitos impresos	10	47,49 €
LM2596	10	13,41
BMP180	10	8,25 €
ESP-WROOM-32	10	42,20
LM339N	10	8,34 €
2N3896	30	4,52 €
10k	60	0,60 €
33k	60	0,60 €
330k	30	0,30 €
680k	30	0,30 €
Pot 1M	30	6,50 €
0.1uF	20	2,24 €
IDC header 2x5	10	12,4 €
BNC 1x4	10	25,04 €
Terminal Block	10	1,23 €
Total		354,13 €

Tabla 9.1: Componentes Electrónicos

Concepto	Cantidad	Importe
ASUS Zenbook Flip 14	1	799 €
MiniPC MSI Cubi	1	594,60 €
Dremel 4000	1	152,20 €
Estación soldadura	1	89,90 €
Router ASUS RT-AC86U	1	164,30 €
Total Parcial		1800 €

Tabla 9.2: Material

Concepto	Horas	Precio/Hora	Importe
Salario	300	50€/Hora	15000 €
Total Parcial			15000 €

Tabla 9.3: Personal

Concepto	Importe
Componentes electrónicos	354,13 €
Material	1800 €
Personal	15000€
Total	17154,13 €

Tabla 9.4: Importe final

Bibliografía

[1] "Grupo de Investigación Espacial," https://www.uah.es/es/investigacion/unidades-de-investigacion/grupos-de-investigacion/Grupo-de-Investigacion-Espacial-Space-Research-Group [último acceso 23/junio/2022].

[2] "Página de CaLMa," https://neutronmonitors-srg-uah.web.uah.es/ [último acceso 26/junio/2022].

[3] "NDB, Guadalajara Spain," https://www.nmdb.eu/station/calm/ [último acceso 24/junio/2022].

[4] E. Catalán, "Monitor de neutrones de castilla-la mancha (CaLMa)." [Online]. Available: https://docplayer.es/8219116-Monitor-de-neutrones-de-castilla-la-mancha-calma-edwjoe-yahoo-es.html

[5] "ORCA: NEMO Y MITO," https://neutronmonitors-srg-uah.web.uah.es/instruments/orca/ [último acceso 24/junio/2022].

[6] "Observatorio de Rayos Cósmicos Antártico, Antarctica / Spain," https://www.nmdb.eu/station/orca/ [último acceso 25/junio/2022].

[7] "MiniCalma," https://neutronmonitors-srg-uah.web.uah.es/instruments/minicalma/ [último acceso 25/junio/2022].

[8] O. G. Población, "The neutron monitor control panel," *Journal of Physics: Conference Series*, no. 632, 2015. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/632/1/012055/pdf

[9] O. García, "Calma station and a new data acquisition system for neutron monitors," Ph.D. dissertation, Universidad de Alcalá Departamento de Automática, 2020.

[10] V. Florio, "El origen de los rayos cósmicos : un estudio indica con mayor precisión que esas partículas ultraenergéticas provienen del exterior de la vía láctea," *Pesquisa FAPESP*, no. 260, oct 2017. [Online]. Available: https://revistapesquisa.fapesp.br/es/el-origen-de-los-rayos-cosmicos/

[11] "Figura obtenida de NMDB," https://www.nmdb.eu/maps/nmdb_map.png [último acceso 24/junio/2022].

[12] J. Medina, *Introducción al estudio de los rayos cósmicos*, U. de Alcalá, Ed., 2011.

[13] "Resúmenes de Salud Pública: radiación ionizante," https://www.atsdr.cdc.gov/es/phs/es_phs149.html [último acceso 02/junio/2022].

[14] "NDB,Neutron monitors," https://www.nmdb.eu/public_outreach/en/04_nm/ [último acceso 28/junio/2022].

[15] "Chalk River BF3 Proportional Counter (ca. 1960s, 70s)," https://www.orau.org/health-physics-museum/collection/proportional-counters/neutron-detectors/chalk-river-bf3.html [último acceso 25/junio/2022].

[16] "Imagen obtenida de la presentación de GEANT4 y CaLMa," https://upload.wikimedia.org/wikipedia/commons/thumb/9/92/Espressif_ESP32_Chip_Function_Block_Diagram.svg/1200px-Espressif_ESP32_Chip_Function_Block_Diagram.svg.png [último acceso 9/agosto/2022].

[17] "Documentación sobre ESP32," https://www.espressif.com/en/products/socs/esp32 [último acceso 23/junio/2022].

[18] "Página oficial de ESP32 Espressif," https://www.espressif.com/en/products/socs/esp32 [último acceso 3/agosto/2022].

[19] "Figura obtenida de la documentación de Wikipedia," https://upload.wikimedia.org/wikipedia/commons/thumb/9/92/Espressif_ESP32_Chip_Function_Block_Diagram.svg/1200px-Espressif_ESP32_Chip_Function_Block_Diagram.svg.png [último acceso 13/julio/2022].

[20] J. Ivkovic, "Analysis of the performance of the new generation of 32-bit microcontrollers for iot and big data application," in *7th International Conference on Information Society and Techology, Kopaonik,*, 2017. [Online]. Available: https://www.researchgate.net/publication/316173015_Analysis_of_the_performance_of_the_new_generation_of_32-bit_Microcontrollers_for_IoT_and_Big_Data_Application

[21] "Documentación oficial sobre Kicad," https://docs.kicad.org/ [último acceso 28/junio/2022].

[22] "Página oficial de SnapEDA," https://snapeda.com/ [último acceso 3/agosto/2022].

[23] "Página oficial de Grabcad," https://grabcad.com/ [último acceso 22/agosto/2022].

[24] "Figura obtenida de la documentación oficial de Espressif," https://docs.espressif.com/projects/esp-idf/en/latest/esp32/_images/what-you-need.png [último acceso 25/junio/2022].

[25] "Guía de introducción a FreeRTOS," https://www.freertos.org/\uppercase{RTOS}.html [último acceso 25/junio2022].

[26] "Figura obtenida de la documentación oficial de FreeRTOS," https://www.freertos.org/fr-content-src/uploads/2019/08/POSIX.jpg [último acceso 9/agosto/2022].

[27] "Figura obtenida de la encuesta anual de stack overflow," https://survey.stackoverflow.co/2022/ [último acceso 20/agosto/2022].

[28] D. Ibrahim, *ARM-Based Microcontroller Multitasking Projects*, Newnes, Ed., 2020.

[29] J. Arm, "Measuring the performance of FreeRTOS on ESP32 multi-core," *IFAC-PapersOnLine*, vol. 55, no. 4, pp. 292–297, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896322003639

[30] D. Stanley, "Control and provisioning of Wireless Access Points (capwap) Protocol Binding for IEEE 802.11," RFC 5416, mar 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5416

[31] D. Mills, "Simple network time protocol SNTP version 4 for IPv4, IPv6 and OSI," RFC 4330, jan 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4330

[32] "MQTT specifications," https://mqtt.org/mqtt-specification/ [último acceso 18/julio/2022].

[33] "Repositorio oficial del proyecto," https://github.com/diego-acc/Neutron-Monitor-Data-Acquisition [último acceso 10/sept/2022].

[34] "Get started with IfluxDB oss 2.2," https://docs.influxdata.com/influxdb/v2.2/ [último acceso 15/junio/2022].

[35] "Página oficial de Go," https://go.dev/ [último acceso 1/junio/2022].

[36] "MQTT consumer input plugin," https://github.com/influxdata/telegraf/blob/release-1.23/plugins/inputs/mqtt_consumer/README.md [último acceso 20/julio/2022].

[37] "Página oficial de Mosquitto," https://mosquitto.org/ [último acceso 3/julio/2022].

[38] "Mosquitto.conf man page," https://mosquitto.org/man/mosquitto-conf-5.html [último acceso 26/julio/2022].

[39] "Documentation Grafana," https://grafana.com/docs/ [último acceso 18/junio2022].

[40] "Documentación Github Actions," https://github.com/features/actions [último acceso 30/julio/2022].

[41] "Página oficial de Google Cloud," https://cloud.google.com/?hl=es-419 [último acceso 20/mayo/2022].

[42] "Página oficial de JLCPCB," https://jlcpcb.com/ [último acceso 3/agosto/2022].

[43] D. A. Zajac, "An open source iot garage real time controller (garagertc," *Jordan Journal of Electrical Engineering*, vol. 6, no. 3, pp. 179–203, 2020. [Online]. Available: https://www.academia.edu/44811944/An_Open_Source_IoT_Garage_Real_Time_Controller_GarageRTC_

[44] R. Michon, "A fauts architecture for the ESP32 microcontroller," in *Sound and Music Computing Conference (SMC-20)*. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02988312/document

[45] "Guía de introducción IDE Espressif," https://docs.espressif.com/projects/esp-idf/en/release-v4.4/esp32/index.html [último acceso 27/mayo/2022].

[46] "Página ofical The latex Project," https://www.latex-project.org/ [último acceso 27/agosto/2022].

[47] "Google Cloud Fundamentals: Core Infrastructure," https://es.coursera.org/learn/gcp-fundamentals [último acceso 10/junio/2022].

[48] "Essential Google Cloud Infrastructure: Foundation," https://www.coursera.org/learn/gcp-infrastructure-foundation [último acceso 10/junio/2022].

[49] "Curso de iniciación ESP32," https://learnesp32.com/ [último acceso 1/07/2022].

[50] "Essential Google Cloud infrastructure: Core service," https://es.coursera.org/learn/gcp-infrastructure-core-services [último acceso 20/junio/2022].

[51] "Elastic Google Cloud Infrastructure: Scaling and Automation," https://es.coursera.org/learn/gcp-infrastructure-scaling-automation [último acceso 22/junio/2022].

[52] "Reliable Google Cloud Infrastructure: Design and Process," https://es.coursera.org/learn/cloud-infrastructure-design-process [último acceso 19/junio/2022].

[53] D. Levin, "Resistor 0.25w 6.3x2.3mm." Mayo 2020, https://grabcad.com/library/resistor-0-25w-1 [último acceso 18/agosto/2022].

[54] C. Lara, "Potenciometro," Abril 2014, https://grabcad.com/library/potenciometro-1 [último acceso 18/agosto/2022].

[55] U. ELECTRONICS, "Circuito integrado LM339 comparador de voltaje," September 2019, https://grabcad.com/library/lm339n-1 [último acceso 18/agosto/2022].

[56] M. WORKSHOP, "LM2596 DC to DC Buck Converter Module," October 2020, https://grabcad.com/library/lm2596-dc-to-dc-buck-converter-module-1 [último acceso 18/agosto/2022].

[57] Singlefonts, "Pin header and jumper: Pin header, straight, 3 poles, pitch 2,54 mm with a jumper," April 2019, https://grabcad.com/library/pin-header-and-jumper-1 [último acceso 18/agosto2022].

[58] D. Levin, "Aluminium Electrolytic Capacitors R/A," june 2020, https://grabcad.com/library/aluminium-electrolytic-capacitors-r-a-1 [último acceso 18/agosto/2022].

[59] ——, "Aluminium Electrolytic Capacitors V/T," june 2020, https://grabcad.com/library/aluminium-electrolytic-capacitors-v-t-1 [último acceso 18/agosto/2022].

[60] "Single Supply Quad Comparators LM339, LM339e, LM239, LM2901, LM2901e, LM2901v, ncv2901, mc3302," https://www.onsemi.com/pdf/datasheet/lm339-d.pdf [último acceso 18/agosto/2022].

[61] A. Maier, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things," in *7th International Conference on Internet Technologies and Applications*, 2017. [Online]. Available: https://www.researchgate.net/publication/320273388_Comparative_Analysis_and_Practical_Implementation_of_the_\uppercase{ESP}32_Microcontroller_Module_for_the_Internet_of_Things

[62] "Cajas estancas de derivación IP65-IP67," https://ide.es/esp/productos/cajas-de-derivacion-y-mecanismos/cajas-estancas-de-derivacion-ip65-ip67 [último acceso 18/agosto/2022].

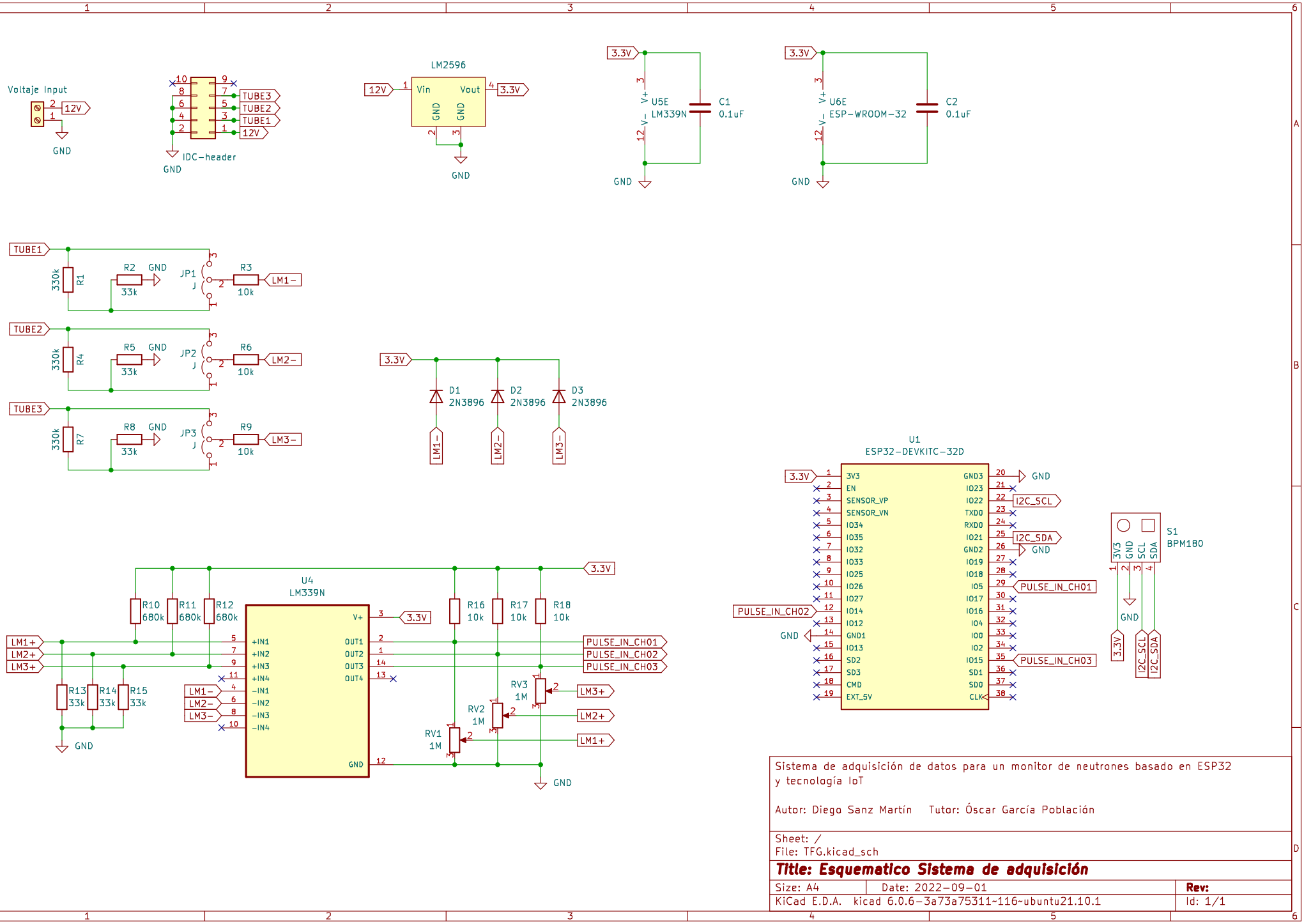
[63] "Página oficial de GNU Linux," https://www.gnu.org/home.es.html [último acceso 1/junio/2022].

[64] "Página oficial de Vim," https://www.vim.org/ [último acceso 1/junio/2022].

[65] "Página oficial de Diagrams.net," https://www.diagrams.net/ [último acceso 1/junio/2022].

Apéndice A

Esquemático



Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Autor: Diego Sanz Martín Tutor: Óscar García Población

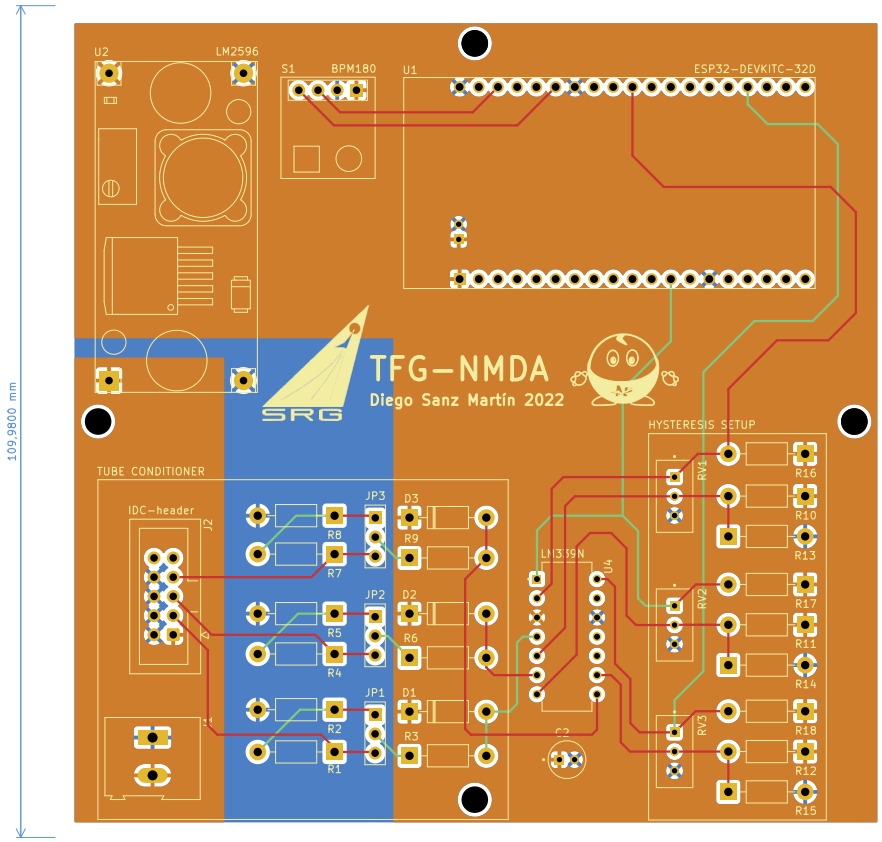
Sheet: /
File: TFG.kicad_sch

Title: Esquematico Sistema de adquisición

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A. kicad 6.0.6-3a73a75311-116~ubuntu21.10.1		Id: 1/1

Apéndice B

Esquemático PCB



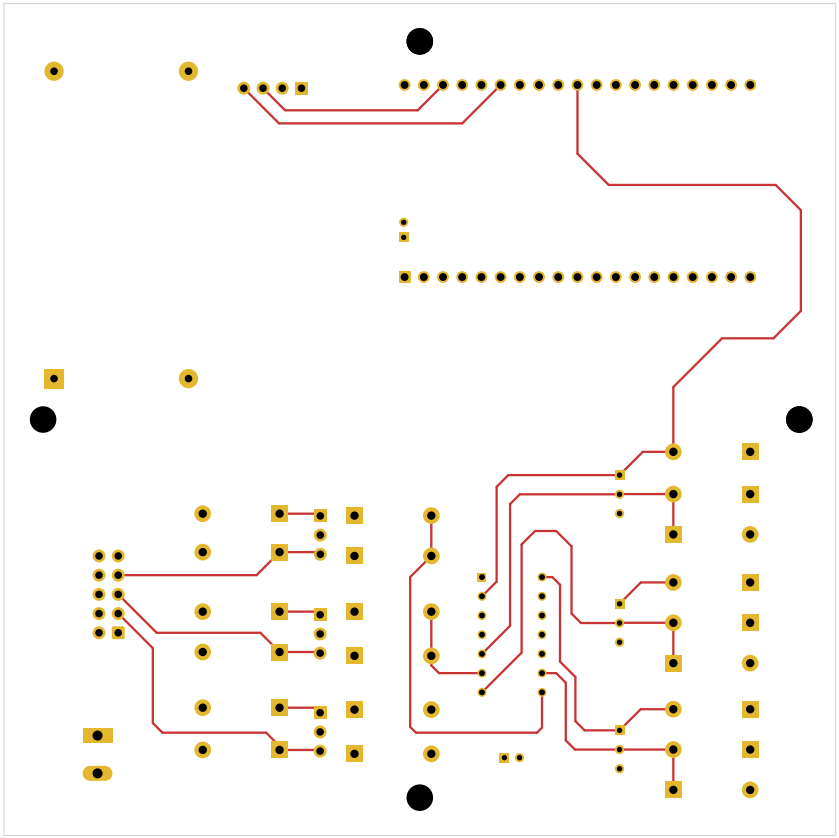
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A. kicad 6.0.6-3a73a75311~116~ubuntu21.10.1		Id: 1/1



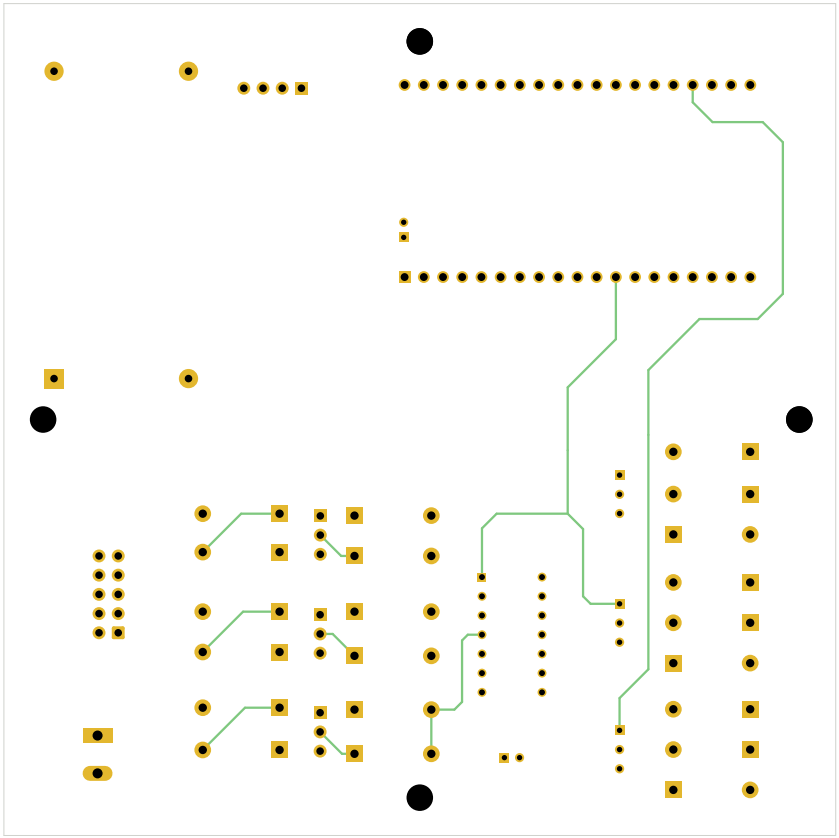
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A. kicad 6.0.6-3a73a75311~116~ubuntu21.10.1		Id: 1/1



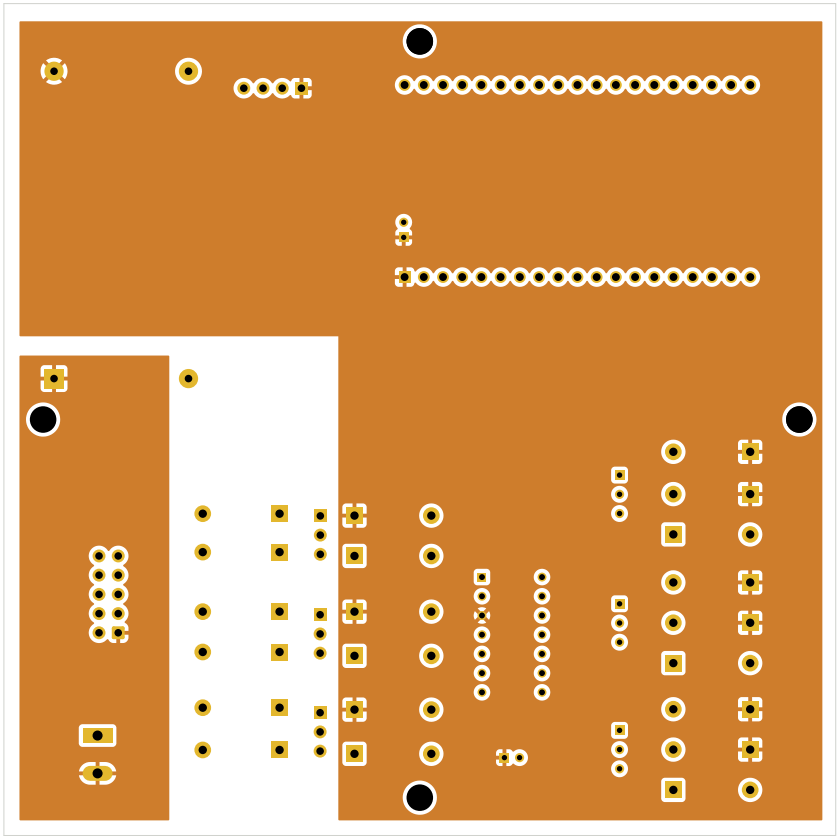
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A.	kicad 6.0.6-3a73a75311~116~ubuntu21.10.1	Id: 1/1



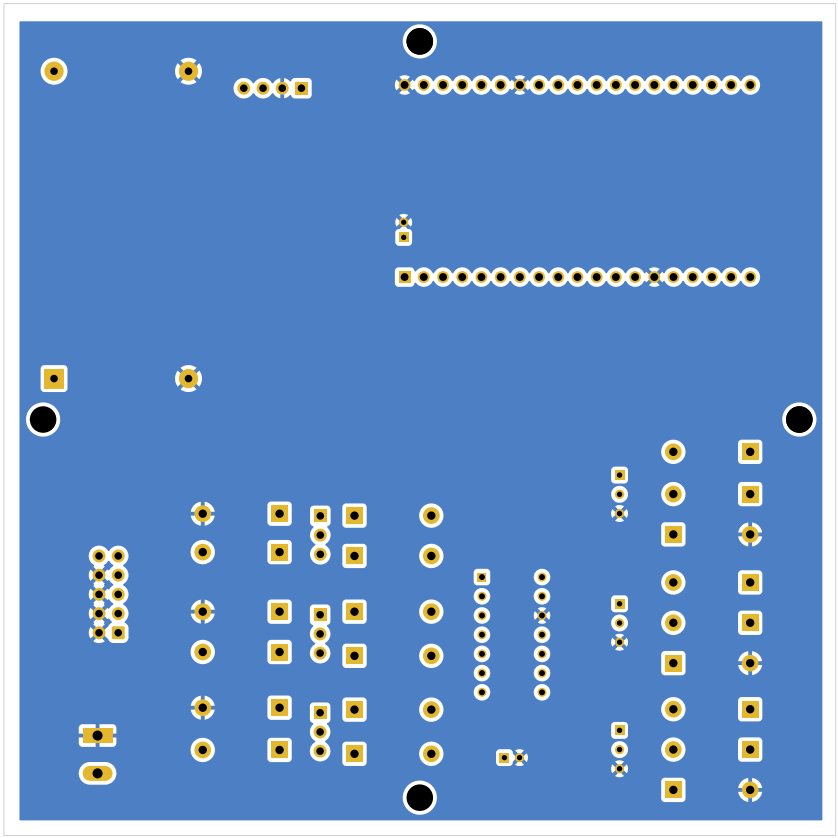
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A. kicad 6.0.6-3a73a75311~116~ubuntu21.10.1		Id: 1/1



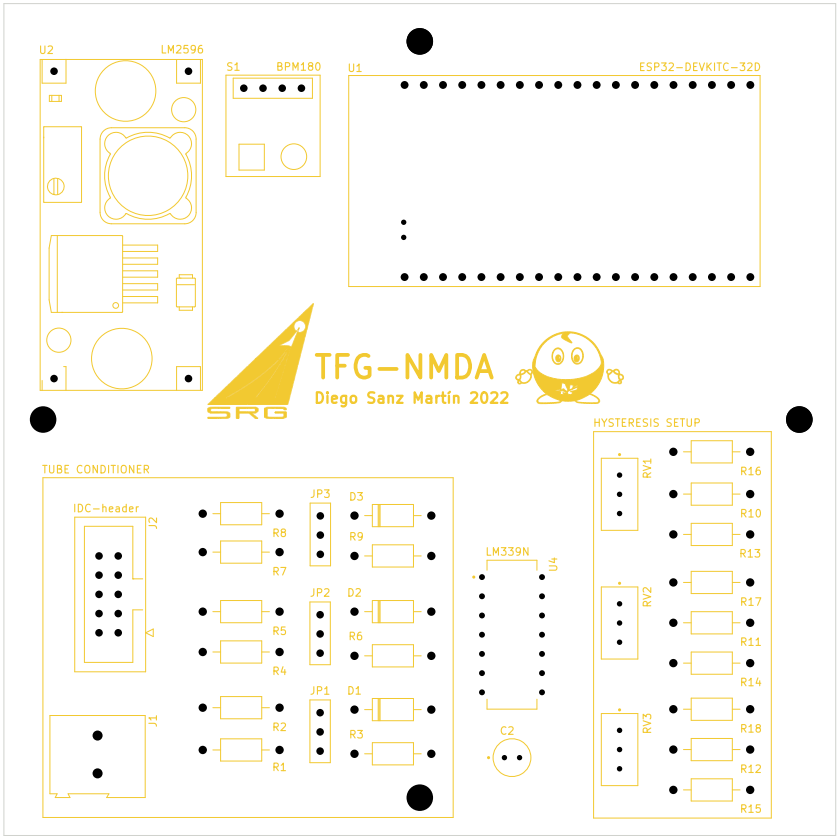
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A.	kicad 6.0.6-3a73a75311~116~ubuntu21.10.1	Id: 1/1



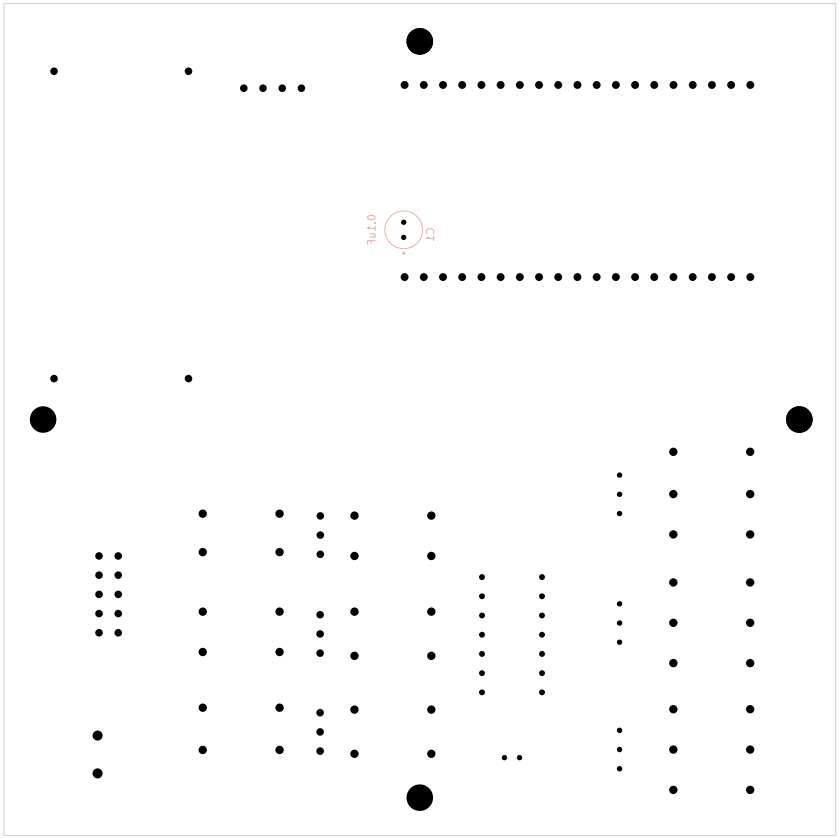
Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

Sheet:
File: TFG.kicad_pcb

Title: Trabajo Fin de Grado | Diego Sanz Martín

Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A.	kicad 6.0.6-3a73a75311~116~ubuntu21.10.1	Id: 1/1



Sistema de adquisición de datos para un monitor de neutrones basado en ESP32 y tecnología IoT

Tutor: Óscar García Población Dto. Automática

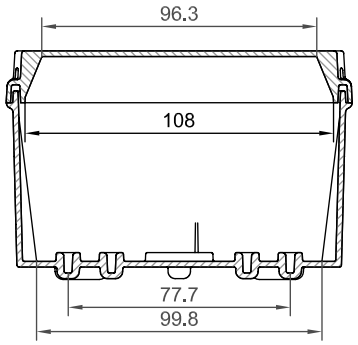
Sheet:
File: TFG.kicad_pcb

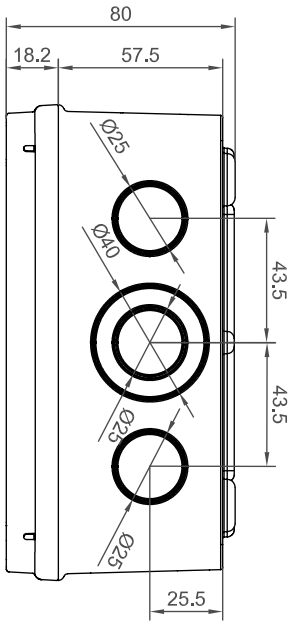
Title: Trabajo Fin de Grado | Diego Sanz Martín

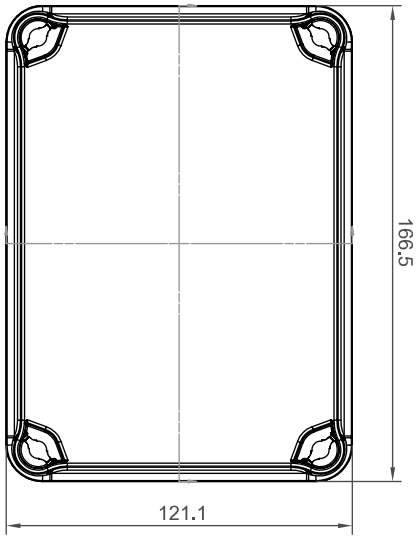
Size: A4	Date: 2022-09-01	**Rev:**
KiCad E.D.A. kicad 6.0.6-3a73a75311~116~ubuntu21.10.1		Id: 1/1

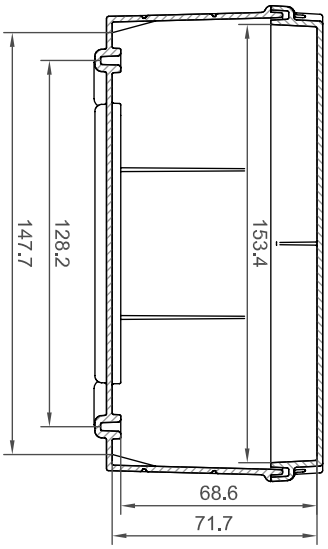
Apéndice C

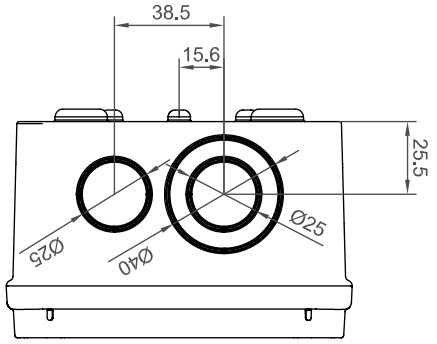
Plano caja derivación

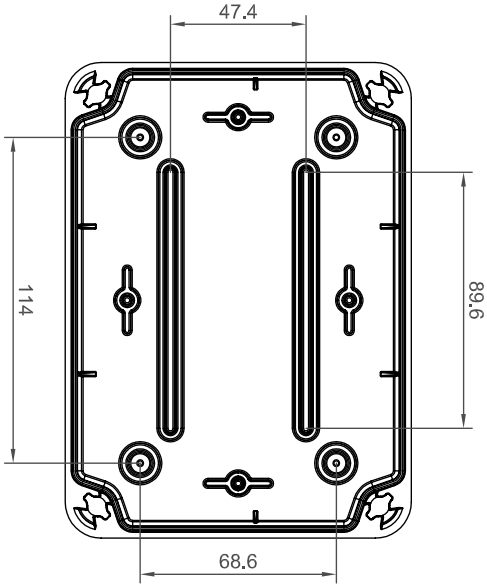


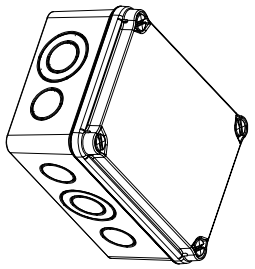












MODIFICADO / MODIFIED			
FIRMA / SIGNATURE	NOMBRE / NAME	FECHA / DATE	Nº EDIC. / NUMBER
DIBUJADO / DRAWN	J.MADRID	19/04/2021	
COMPROBADO / CHECKED	C.VICENTE	19/04/2021	
APROBADO / APPROVED			

www.ide.es

EX161

DESCRIPCION / DESCRIPTION

DESCRIPCION DEL CAMBIO / CHANGE DESCRIPTION

TAMAÑO / SIZE	HOJA 1 DE 1 / SHEET 1 OF 1	TOLERANCIAS GENERALES / GENERAL TOLERANCES	
A2	1:1	EN 22768-mK	

CODIGO DE PLANO / DRAWING CODE

220.06.007

PROYECTO / PROJECT: LIGHT

SUSTITUYE / REPLACE

EDICION / EDITION Nº 2

Apéndice D

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux [63]
- Entorno de desarrollo Vim [64]
- Creador de diagramas Diagrams [65]
- Entorno de desarrollo Espressif [45]
- Curso Espressif [49]
- Entorno de automatización de diseño electrónico Kicad [21]
- Git, Github y Github Actions [40]
- Cuenta Google Cloud [41]
- Cursos Google [47, 48, 50–52]
- Procesador de textos LaTeX [46]
- Modelos 3D obtenidos de GrabCad [53–60]

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR