industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

DISEÑO Y DESARROLLO DE NODOS SENSORES DE BAJO COSTE PARA MONITORIZAR PARÁMETROS EN AGRICULTURA DE PRECISIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **José Antonio Peña Simó**

Director: Juan Antonio López Riquelme

Codirector: Antonio Mateo Aroca

Cartagena, 10 de octubre de 2019

Universidad Politécnica de Cartagena

Quiero agradecer a Juan Antonio López Riquelme el darme la oportunidad de realizar este trabajo, así como su inestimable ayuda para realizarlo.

Necesito hacer mención especial a Manuel Forcén Muñoz, ya que ha sido piedra angular en la realización del proyecto y al cual estoy enormemente agradecido.

Por supuesto agradecer a mi familia y a todas las personas que me han apoyado y acompañado en algún momento de mi vida porque son partícipes de lo que soy.

Por último, quiero dedicárselo a esas personas que son como un faro en la vida, que te guían cuando más perdido puedes estar.

Índice

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción

1.1 Marco de trabajo.

La sociedad actual avanza a pasos agigantados hacia un mundo más globalizado, donde todo está al alcance de la mano y las barreras que antaño se podían tener como las distancias y la dificultad de hacer llegar un mensaje a otro punto del planeta, hoy en día son meras anécdotas del pasado. Al igual que dentro de unos años estaremos hablando de temas que a día de hoy son casi impensables.

Todo este desarrollo viene de la mano de Internet, que es el gran impulsor de esta globalización y desarrollo de tecnologías, gracias a que facilita la comunicación y transferencia de datos entre personas y dispositivos.

La utilización de dispositivos conectados hecha ya un hábito diario de la población media, con los Smartphone al alcance de todos, se crea una nueva necesidad, como es, darle un uso a estos dispositivos. La conexión entre estos ayuda a una vida más cómoda para sus usuarios, ya que mediante pequeños movimientos tienes infinidad de servicios.

Esta área está dentro del marco del Internet de las Cosas (*IoT*, de sus siglas en inglés, *Internet of Things*)[1], concepto que hace mención al uso de la conexión que se realiza, en este caso mediante internet, con el resto de objetos de la vida cotidiana que nos rodea, dándole un uso inteligente a los objetos utilizados.

1.2 Motivación.

Debido al auge del movimiento *maker*, campos como es el del Internet de las Cosas están creciendo cada vez más.

El movimiento *maker* está basado en la cultura del *DIY* (hazlo tú mismo, o en sus siglas en inglés, *do it yourself*), que se puede definir como la práctica de la fabricación o reparación de cosas por uno mismo, con la que se entretiene y se aprende al mismo tiempo. Los *makers* se ayudan de esto gracias a la aparición de nuevos dispositivos electrónicos y plataformas que hacen viable la realización de pequeños proyectos de una forma relativamente sencilla y a un bajo coste.

El Código Abierto, u *Open Source*, es uno de los principales culpables de esta iniciativa, ya que está basado en la colaboración de contenidos y conocimientos de forma abierta y desinteresada de software libre por parte de programadores y usuarios informativos en general.

Junto a la filosofía de software libre surge también la de hardware libre, que es la creación de dispositivos hardware con contenido accesible al público. Esto no es más que cualquier dispositivo electrónico o maquinaria que se realice bajo esta filosofía, está acompañada de sus correspondientes diagramas o esquemáticos que permiten su reproducción. Las aportaciones de estos contenidos se realizan por parte de sus creadores, o bien, por atribuciones de otros usuarios buscado siempre una mejora del producto inicial.

Dentro de todo este marco de movimientos nace el interés e inquietud de poner en práctica los conocimientos adquiridos en la titulación y se considera la realización de un proyecto que abra las puertas a este mundo, tomado como punto de partida para entrar en la comunidad que engloba todos estos términos.

1.3 Objetivos.

Este proyecto pretende dar una solución a la monitorización y adquisición de datos desde un nodo inalámbrico.

Para logar el objetivo, habrá que pasar por distintas fases como serán:

- Estudio de mercado de los dispositivos que se pueden encontrar en el momento.

- Comparativa y búsqueda de prestaciones que el proyecto requiera.
- Desarrollo de los diseños y distintas ideas.
- Realización de los componentes y dispositivos necesarios para llevar a cabo el proyecto.
- Comprobación del correcto funcionamiento y toma de datos.
- Solución de problemas y adversidades que se puedan encontrar durante el proceso, entre otras cosas…

Todas estas fases nombradas anteriormente se podrán realizar tantas veces como dispositivos o módulos se requieran para lograr el objetivo final. Ya que dependiendo de los sensores o componentes elegidos pueden tener una interfaz o protocolo determinado, por lo que habrá que adaptarse a las condiciones que se encuentren.

Una vez realizado todos estos pasos, se podrá cumplir la finalidad del trabajo realizado.

Capítulo 2

Estado del Arte

2.1 Introducción.

En el capítulo anterior se presentó como objetivo principal de este trabajo el desarrollo de un dispositivo inalámbrico que permita la comunicación de datos para una vez realizado esto poder utilizarlos para su monitorización.

Para conseguir dicho objetivo hay que seleccionar un conjunto de elementos hardware y software, con lo que lleva ello asociado, para a partir de ahí seguir diseñado el resto de componentes.

En este capítulo se describen las distintas posibilidades que se han analizado para diseñar y desarrollar cada módulo de este trabajo. Para ello se buscaran las principales opciones disponibles en el mercado actual de los distintos fabricantes o creadores de la rama tecnológica en estudio.

2.2 Protocolo de comunicación.

Para realizar la comunicación entre dispositivos se necesita un medio por el cual realizar la transmisión de información, para realizar esta función disponemos de los protocolos de comunicación.

Una definición que podemos encontrar de protocolo de comunicación, en el ámbito de la informática y telecomunicación, es la de un sistema que permite que dos o más entidades

se comuniquen entre ellas para transmitir información por medio de cualquier tipo de variación de una magnitud física.

Según el modelo *OSI* (*Open System Interconnection*, interconexión de sistemas abiertos)[2], podemos definir la comunicación de *ETD* (Equipos Terminales de Datos) en varios niveles.

Capas	Niveles	Categorías
Capa 7	Nivel de aplicación	Aplicación
Capa 6	Nivel de presentación	
Capa 5	Nivel de sesión	
Capa 4	Nivel de transporte	
Capa 3	Nivel de red	Transporte de datos
Capa 2	Nivel de enlace de datos	
Capa 1	Nivel físico	

Tabla 2.1. Modelo OSI.

Como podemos ver en la tabla anterior, se dividen en siete niveles, que a su vez pueden agruparse en dos categorías distintas. En la categoría de Aplicación se abarcan las cuatro capas superiores que trabajan con los problemas que particularizan a las aplicaciones, y en la categoría de Transporte de datos engloba las tres capas inferiores que trabajan con lo referido al trasporte de los datos.

A continuación, centrándonos en el tipo de comunicación a utilizar, hay que hacer un estudio de las distintas tecnologías que podemos encontrar en este momento.

2.2.1 WiFi

La tecnología que permite la comunicación inalámbrica más extendida hoy en día es la tecnología WiFi, gracias a ella es posible conectar dispositivos a Internet. Aunque no destaca por poseer una alta eficacia energética, es interesante tenerla en cuenta debido a su gran estandarización. Debe su nombre al término *Wireless Fidelity*. Está basada en el estándar IEEE 802.11.

Ilustración 2.1. Tecnología WiFi.

2.2.2 LiFi

Esta tecnología se basa en comunicación óptica inalámbrica de corto alcance que mediante el parpadeo que realizan los dispositivos emisores de luz, sea posible la emisión de mensajes. Se presenta como una alternativa sostenible, eficiente y muy económica a las actuales, aunque actualmente se encuentra en fase de desarrollo.

Su nombre proviene del acrónimo del término inglés *light fidelity*, término que se usa para etiquetar a los sistemas de comunicaciones inalámbricas rápidas y de bajo costo, de transmisión de datos a través de la luz.

Ilustración 2.2. Tecnología LiFi.

2.2.3 Bluetooth

Presentamos Bluetooth como una tecnología comunicación de corto alcance entre dispositivos. Su conexión física trabaja a través del estándar de radio frecuencia en la banda ISM de los 2.4 GHz. El radio de alcance entre dispositivos puede llegar hasta los 100 metros, pero a medida que esta distancia es mayor, se pierde potencia en la transferencia de datos. Su capacidad de trasferencia máxima alcanza los 24 Mbit/s.

Ilustración 2.3. Tecnología Bluetooth.

2.2.4 BLE

Una evolución del sistema Bluetooth sería BLE, *Bluetooth Low Energy*, es un estándar nuevo de la tecnología Bluetooth, que permite la transmisión de datos con un menor gasto energético. Este estándar se establece a partir de la versión 4.0 de la tecnología Bluetooth. Actualmente dispone de una capacidad de transferencia de hasta 50 Mbit/s.

Surge ligado al IoT para mejorar la conectividad entre dispositivos sin renunciar al bajo consumo. También es conocido por Bluetooth ULP (*Ultra Low Power*) y Bluetooth Smart.

Ilustración 2.4. Tecnología Bluetooth Smart.

2.2.5 Zigbee

Zigbee se define como especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radiodifusión digital de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal.

De esta tecnología se puede destacar su bajo coste y su reducido consumo energético, que son fundamentales para fijar una topología basada en una red en malla entre los dispositivos conectados.

Ilustración 2.5. Tecnología Zigbee.

2.2.6 GPRS

Se define como Servicio General de Paquetes vía Radio, en inglés General Packet Radio Service, y se basa en la trasmisión de paquetes para una comunicación de datos, especialmente a Internet. Este sistema, basado en el sistema de comunicación GSM, está enfocado a las comunicaciones móviles. Tiene una gran ventaja respecto a los otros sistemas de comunicación antes nombrados, que es que mientras dispongamos de cobertura telefónica podremos realizar la comunicación.

Ilustración 2.6. Tecnología GPRS.

2.2.7 NFC

NFC, *Near Field Communication*, es una tecnología inalámbrica de corto alcance que trabaja en la banda de los 13.56 MHz. Permite la comunicación entre dispositivos sin la necesidad de un emparejamiento previo entre ellos y con una buena velocidad de trasferencia de datos. Su rango de acción es de unos pocos centímetros, por lo que requiere que los

dispositivos se encuentren prácticamente en contacto uno con otro para realizar la comunicación.

Ilustración 2.7. Tecnología NFC.

2.2.8 IrDA

IrDA, *Infrared Data Association*, define un estándar físico en trasmisión de datos mediantes rayos infrarrojos. La tecnología IrDA está basada en rayos luminosos que se mueven en el espectro infrarrojo. La velocidad de transferencia entre dispositivos oscila entre los 9600 bit/s y los 4 Mbit/s.

Ilustración 2.8. Tecnología IrDA.

Una vez que ya hemos nombrado algunas de las tecnologías de comunicación entre dispositivos, en el siguiente paso debemos analizar los puntos positivos y negativos que nos proporcionan y estudiar cual se adapta mejor a las necesidades de este trabajo. Siendo interesante hacer hincapié en el bajo consumo y una transferencia de datos óptima.

2.3 Sistemas de procesamiento.

Antes de nada, debemos definir el concepto de procesamiento de datos.

Podemos decir que el procesamiento de datos es el manejo y almacenamiento de elementos para producir una información. Partiendo de este concepto, un sistema de procesamiento es el encargado de realizar las tareas de procesado de unos datos dedicados a un fin concreto.

Siguiendo con esto, un sistema puede estar formado por diferentes elementos. En nuestro caso, estos elementos contendrán entre ellos uno o más dispositivos, tantos como se requiera y cumpla los estándares fijados al inicio de este proyecto

Para realizar las tareas de cómputo será necesario una unidad de procesamiento, donde toman importancia los microcontroladores o los microprocesadores. En el caso de los microprocesadores el abanico de tipos es más amplio, ya que un microprocesador es un ordenador a pequeña escala, por lo que dispone de un procesador propio que realiza las funciones de cerebro del dispositivo.

El microcontrolador, según su abreviatura µC o sus siglas en inglés MCU, *Micro Controller Unit*, se define como un circuito integrado programable, el cual ejecuta una serie de órdenes grabadas en su memoria. También se puede clasificar como un autómata programable, que se encarga de trabajar con las entradas y salidas que recibe, y su función principal es dirigir las operaciones que se le haya implementado. Pueden disponer de CPU, pero siempre con mucha menos potencia que la de los microprocesadores, ya que no requieren de grandes operaciones de procesado de datos.

Para seguir conociendo más sobre los microcontroladores, habrá que centrarse en algunos modelos que nos ofrezca el mercado actualmente.

A continuación entramos en estudio de las distintas ofertas de los diferentes fabricantes que predominen en este ámbito.

2.3.1 Arduino UNO

Arduino es una compañía *open source* y *open hardware*, que se dedica al diseño y desarrollo de placas hardware. Se enfoca en acercar y facilitar el área de la electrónica y la programación a cualquier usuario. Uno de sus grandes éxitos es la gran comunidad que se ha creado de desarrolladores en esta plataforma.

La plataforma Arduino nos facilita la creación de proyectos gracias su propio entorno de desarrollo integrado, que nos permite escribir código y subirlo a la tarjeta. Este entorno es Arduino IDE y más adelante hablaremos de él.

En este caso analizamos una tarjeta Arduino, en concreto nos centraremos en el modelo Arduino UNO.

Ilustración 2.9. Arduino UNO.

La tarjeta Arduino UNO fue la primera en aparecer en la que ya es una más que conocida familia de dispositivos de Arduino, entre los que encontramos una amplia gama de modelos con distintas funcionalidades y *shield* (expansiones adaptables que se acoplan a los modelos base para completar su funcionalidad).

Cabe recordar, que como uno de los principios de este proyecto, se buscaba la utilización de dispositivos de Código abierto, y Arduino entra dentro de esa familia, al igual que el resto de dispositivos con lo que se trabajará.

Centrándonos en las características técnicas de esta tarjeta, antes de entrar más en detalle se puede destacar de ella que está basada en el microchip de Atmel (actualmente adquirida por Microchip Technology), ATmega328P, que proporciona a la tarjeta 14 pines digitales de entrada/salida, de los cuales 6 de ellos se pueden usar como salidas PWM y 6 entradas analógicas. Además conexión por USB, cristal de cuarzo de 16 MHz, conector ICSP, entre otras.

Entrando en detalles sobre las especificaciones que nos proporciona la tarjeta, se pueden destacar las siguientes:

- Microcontrolador: ATmega328P.
- Voltaje de operación: 5V.
- Voltaje de entrada recomendado: 7 ~ 12V.
- Voltaje de entrada límite: 6 ~ 20V.
- Pines de entrada/salida digital: 14 (6 de salida PWM).
- Pines de entrada analógica: 6.
- Corriente continua por pin IO: 40mA.
- Corriente continua en el pin 3.3V: 50mA.
- Memoria Flash: 32KB (0.5 utilizados por el bootloader).
- SRAM: 2KB.
- EEPROM: 1KB.
- Frecuencia de reloj: 16MHz.
- Comunicación serial, RX y TX.
- Comunicación SPI, que permite trasmitir información full dúplex.
- Comunicación mediante bus I^2C.

2.3.2 Arduino Nano 33 BLE

Una alternativa dentro de la familia Arduino sería el Arduino Nano 33 BLE, que dispone de capacidad de comunicación gracias a la tecnología *Bluetooth Low Energy*.

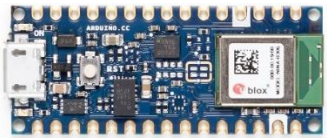

Ilustración 2.10. Arduino Nano 33 BLE.

A diferencia de la mayoría de tarjetas de la familia Arduino, este modelo se basa en el microcontrolador nRF52840 fabricado por Nordic Semiconductor. La comunicación BLE la realiza gracias al módulo NINA B306, usando la versión de BT 5.0.

Otras características que posee son las siguientes:

- Reloj de 64MHz
- Memoria Flash de 1MB
- Memoria RAM de 256KB
- Voltaje de operación es de 3.3V
- 8 pines analógicos de I/O.
- 14 pines digitales de I/O.
- Un puerto de comunicación UART.
- Un puerto de comunicación SPI.
- Un puerto de comunicación I^2C.

2.3.3 STM32

En esta ocasión hablamos de la familia STM32. Esta familia está formada por microcontroladores de 32 bits basados en el procesador ARM Cortex-M y fabricados por STMicroelectronics.

Dentro de la familia STM32 existen diferentes series de microcontroladores, donde su principal diferencia es el modelo de procesador que montan. Las series que se ofrecen son: H7, F7, F4, F3, F2, F1, F0, L4, L1, L0.

Aquí vamos a hacer mención de la serie STM32 F1, que fue el primer grupo de microcontroladores basados en el núcleo ARM Cortex-M3.

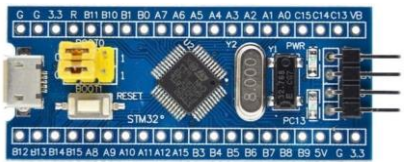

Ilustración 2.11. STM32.

La serie STM32 F1 ha sufrido cambios a lo largo del tiempo, aumentando la velocidad de la CPU, el tamaño de la memoria interna y la variedad de periféricos. Las líneas de evolución han sido:

- Conectividad, con los modelos STM32F105 y STM32F107,
- Rendimiento, con STM32F103.
- Acceso USB, con STM32F102.
- Acceso, con STM32F101.
- Valor, con STM32F100.

Ahora podemos ver un resumen de las características que nos puede ofrecer esta serie dependiendo de que dispositivo se seleccione:

- Núcleo ARM Cortex-M3 con una velocidad de reloj máxima de 24/36/48/72 MHz.
- Memoria RAM de 4/6/8/10/16/20/24/32/48/64/80/96 KB.
- Memoria Flash de 16/32/64/128/256/384/512/768/1024 KB.

2.3.4 ESP8266

El primero que encontramos es el chip ESP8266. Es un chip de comunicación WiFi creado por Espressif, que gracias a su bajo ha tenido una gran repercusión entre los desarrolladores de electrónica de open source, ya que reemplazaba al módulo de Arduino MKR100, que realizaba las mismas funciones pero a un precio bastante inferior.

Ilustración 2.12. ESP8266.

Desde su creación en 2014, este chip ha ido evolucionando desde el módulo ESP-01 fabricado por la empresa Al-Thinker, hasta la última versión en el mercado que es ESP-12. Podemos destacar que al carecer este chip de memoria Flash, está obligado a ir acompañado

de un módulo que se la proporcione. Esta carencia la solvento la empresa con la versión ESP8285, que incorporaba una memoria Flash de 1MB.

Una vez visto algunos datos por encima de este chip, vamos a centrarnos en sus especificaciones más a fondo:

- CPU Tensilica L106 de 32 bit
- Voltaje de operación: 3 ~ 3.6V
- Corriente de operación: 80 mA
- Consumo mínimo: 0.5 µA
- Temperatura de operación: -40/+125ºC
- Pines de entrada salida: 17
- Soporta protocolos IPv4/TCP/UDP/HTTP/FTP
- RTC, reloj de tiempo real
- Comunicación UART
- Comunicación SPI
- Comunicación mediante bus I2C

2.3.5 ESP32

Antes hemos hablado del chip ESP8266 y su nueva versión ESP8285 que aportaba una memoria Flash integrada, pero su fabricante, Espressif, da un salto con el modelo ESP32.

En el ESP32 nos encontramos un dispositivo mucho más potente que su predecesor. Ha sido diseñado completamente enfocado al marco del Internet de las Cosas. Entre las nuevas aportaciones que le presta el fabricante, destacamos la inclusión de conectividad por BLE, Bluetooth Low Energy, junto a la conectividad WiFi que ya disponía el modelo ESP8266, algo muy a tener en cuenta para facilitar las comunicaciones. También nos encontramos ante un procesador adicional y la aportación de pines de salida digital a analógica, DAC.

Ilustración 2.13. ESP32.

Otros datos de interés que podemos encontrar en esta versión son la incorporación de un sensor efecto Hall, un termómetro, memoria Flash integrada, posibilidad de transmisión mediante IR.

Para terminar de hablar de este dispositivo, enumeramos sus características técnicas más relevantes:

- Procesador Tensilica Xtensa Dual-Core LX de 32 bit
- RAM: 520 KB
- Flash: hasta 16 MB
- ROM: 448 KB
- Voltaje de operación: 2.2 ~ 3.6V
- Rango de Temperatura: -40/+125ºC
- Consumo mínimo: 2.5 µA
- Pines GPIO utilizables: 11
- Pines PWM: 16
- Pines ADC: 18
- Pines DAC: 2
- Comunicación UART: 2
- Comunicación SPI: 4
- Comunicación mediante bus I2C: 2
- Comunicación mediante CAN bus

2.3.6 nRF52840

En el apartado del Arduino Nano 33 BLE, citado anteriormente, hemos nombrado que estaba basado en el SoC nRF52840. Ya que resulta un componente interesante a tener en cuenta, vamos a hondar un poco más a fondo sobre sus características.

Ilustración 2.14. nRF52840.

Fabricado por Nordic Semiconductor, se presenta como un SoC multiprotocolo avanzado, con las tecnologías de Bluetooth 5, Thread y Zigbee. Es el miembro más avanzado actualmente de la familia de componentes nRF52. Este dispositivo está construido alrededor de la CPU ARM Cortex M4 de 32 bits. También dispone de la tecnología NFC, por lo que se sitúa como un componente muy completo.

- Otras de sus características que podemos ver son:
- Frecuencia de reloj de 64 MHz
- Memoria Flash de 1 MB
- Memoria RAM de 256 KB
- Rango de voltaje soportado de 1.7 hasta 5.5 V.

2.3.7 CC2652R

En este caso presentamos un dispositivo de Texas Instruments, el CC2652R. Es un componente miembro de la plataforma MCU SimpleLink que la forman los dispositivos de TI. Permite la comunicación por RF y un trabajo a baja potencia.

Ilustración 2.15. CC2652R.

Dispone de varias tecnologías como son Zigbee, Thread, Bluetooth 5 y WiFi. También le caracteriza un consumo energético muy bajo.

Otras características que podemos encontrar son las siguientes:

- Frecuencia de reloj de 48 MHz
- Memoria Flash de 352 KB
- Memoria RAM de 8KB
- Memoria ROM de 256KB
- Reloj de tiempo real.
- Dos puertos de comunicación UART.
- Dos puertos de comunicación SPI.
- Un puerto de comunicación I2C.

2.4 Entornos de desarrollo.

Comenzamos definiendo el concepto de entorno de desarrollo como un conjunto de herramientas digitales que proporciona servicios integrales para facilitar al programador o desarrollador de software su tarea. Conocidos como entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés como *Integrated Development Environment*, IDE.

A pesar de la existencia de infinitud de entornos de desarrollo, muchos van directamente relacionados con una placa de desarrollo o una familia de microcontroladores concreta, por lo que haremos énfasis primero en los entornos asociados a los dispositivos descritos en este documento, sin olvidar al resto.

2.4.1 Arduino IDE

Arduino IDE es un entorno de desarrollo en el que se realiza la programación de cada una de las placas de Arduino.

Este entorno de desarrollo ayuda a escribir código y posteriormente subirlo a la tarjeta. Está creado por y para la tarjeta Arduino, pero al ser una plataforma de software libre, la comunidad de desarrolladores ha adaptado este entorno para otras muchas tarjetas y dispositivos, por lo que resulta muy interesante ya que puede ser utilizado por un gran número de dispositivos de diferentes fabricantes.

2.4.2 Energia

Al igual que hemos comentado con Arduino IDE, Energia es el IDE de las placas de desarrollo de Texas Instruments.

La apariencia del entorno de desarrollo destaca por su gran similitud con el entorno de Arduino IDE, ya que Energia está basada en este otro. Sigue siendo de software libre, por lo que las actualizaciones y evoluciones de este entorno van a llegar también por la propia comunidad de desarrolladores. Aunque en este caso resulta más limitado teniendo en cuenta que su utilización se limita a la familia de dispositivos de TI.

2.4.3 PlatformIO

En esta ocasión nos encontramos con PlatformIO, un ecosistema de código abierto para el desarrollo del Internet de las Cosas.

Esta aplicación se presenta como un creador de código multiplataforma. Lo que aporta distinto a los casos anteriores es que desde un solo entorno de desarrollo tienes la posibilidad de trabajar con más de 200 tarjetas de desarrollo y más de 15 plataformas de desarrollo.

2.4.4 Notepad++

Otra herramienta es Notepad++, un editor de texto y de código fuente libre con soporte para varios lenguajes de programación.

Con esta herramienta tenemos la posibilidad de editar código y de forma simple. En este caso no estaríamos ante un entorno de desarrollo, ya que no tendríamos la posibilidad de cargar el código en el dispositivo, pero es una herramienta útil y sencilla para la creación de nuestro trabajo.

2.4.5 ESP-IDF

Es el marco de desarrollo oficial de Espressif, enfocado para el chip ESP32. Acompañado con la información necesaria para su programación y trabajo en el chip.

Está creado en un entorno Linux, pero puede adaptarse sin problemas para trabajar con él en otros sistemas operativos. Como se ha dicho anteriormente, el fabricante aporta guías tanto de instalación como de uso de los recursos necesarios para poder adaptarlo a las necesidades de cada usuario.

2.4.6 Android Studio

No hay que olvidar, que para la realización de este proyecto, es necesario una comunicación con un dispositivo Smartphone, por lo que para ello resulta necesario una herramienta para la creación de una aplicación capaz de interactuar con los componentes del nodo.

En nuestro caso, ya que la aplicación ha de funcionar sobre un Smartphone con sistema operativo Android, vamos a presentar la herramienta Android Studio.

Android Studio es el entorno de desarrollo oficial para la plataforma Android, reemplazando a Eclipse como IDE oficial para el desarrollo de aplicaciones para Android.

2.4.7 App Inventor

Una alternativa más sencilla a Android Studio es App Inventor. Es un entorno de desarrollo de software creado por Google donde el usuario puede crear aplicaciones de forma más visual gracias a un conjunto de herramientas que enlazan una serie de bloques.

Con esta plataforma se le facilita al usuario su uso debido a que no requiere un conocimiento alto de programación, ya que no tiene escribir ni modificar líneas de código.

2.5 Sistemas de alimentación.

Otra de las cuestiones a tener en cuenta tiene que ser el sistema de alimentación que proporcione energía al dispositivo.

2.5.1 Baterías

Para realizar una alimentación inalámbrica, la mejor opción son las baterías, ya que proporcionan la independencia de cables conectados constantemente a fuentes de alimentación y la capacidad de llegar a sitios donde no llega la instalación de la corriente eléctrica. Unos de los modelos de baterías más extendidas en el mercado para estas finalidades, son las baterías de Li – ion, ion de litio, y las baterías LiPo, Polímero de Litio. Ambas muy parecidas, ya que tienen en común su fabricación con Litio.

Ilustración 2.16. Batería Li-ion.

Dependiendo de su voltaje, podemos encontrar diferentes modelos. En el caso de un voltaje de 3.7 V, estaremos antes baterías de una celda, de 7.4 V, dos celdas, y así sucesivamente. Hablando ya de la capacidad, nos encontramos con mA/h, que es la carga eléctrica que puede almacenar la batería, que representa la cantidad de electricidad que atraviesa un conductor al que se le aplica corriente, durante una hora.. Todas estas baterías suelen ir acompañadas de un circuito de protección integrado que permita alimentarlas sin que sufran daños.

2.5.2 Convertidores CC/CC

Los convertidores cc/cc son dispositivos que transforman corriente continua de una tensión a otra. Suelen ser reguladores de tensión. Estos dispositivos simplifican la

alimentación de un sistema, ya que permiten generar las tensiones requeridas por el sistema, reduciendo en su caso la cantidad de líneas de potencia necesarias. Dentro de los convertidores de cc/cc, existen tres tipos, Reductor, *Buck*, Elevador, *Boost*, y Reductor-Elevador, *Buck-Boost*.

En el caso del convertidor reductor, *Buck*, es un convertidor de potencia que obtiene una salida de voltaje menor que a la de entrada.

El convertidor elevador, *Boost*, obtiene a la salida una tensión continua mayor que a la de su entrada.

Y por último el convertidor reductor – elevador, *Buck-Boost*, puede realizar las funciones tanto de aumentar el voltaje a la salida como de disminuirlo, respecto al de entrada.

Dentro de este tipo de dispositivos, habrá que ver los requisitos a tener en cuenta que necesite nuestro trabajo, para la utilización de alguno de ellos si fuera necesario. Si nos queremos centrar en un bajo consumo, uno de los parámetros a considerar debería ser la tensión *dropout*, que representa la mínima diferencia de tensión entre la entrada y la salida dentro de la cual el circuito es capaz de regular la salida dentro de las especificaciones, para que no se vea muy reflejado en las perdidas.

2.5.3 Carga

Una cuestión importante cuando disponemos de baterías es el sistema de carga, ya que se debe disponer de algún método capaz de proporcionar esa energía para que estas no se agoten y pueden realizar su cometido en las mejores condiciones. Una forma de realizarlo y la más extendida, es el suministro desde la red eléctrica, que mediante un transformador que se adapte a las diferentes tipos de baterías, proporcione la alimentación requerida.

Otra de las opciones a tener en cuenta pueden ser los paneles fotovoltaicos, que gracias a la radiación solar permiten crear energía eléctrica. Estos paneles están formados por numerosas celdas para la obtención de energía. Las celdas o células de energía permiten que la luz solar proporcione carga positiva y negativa a dos semiconductores que se encuentran próximos y de distinto tipo, para crear un campo eléctrico que proporcione corriente eléctrica.

Ilustración 2.17. Panel fotovoltaico.

En el caso de precisar el uso de estos paneles, habrá que tener en cuenta las especificaciones y características que describen a cada tipo para encontrar uno que se adapte a nuestras condiciones.

2.6 Conclusión.

Realizado un estudio del mercado en los distintos dispositivos y herramientas, podemos ver que existe un amplio abanico de opciones para elegir y utilizar en la creación de este proyecto. Debemos dar gracias de ello en gran parte al movimiento *maker*, ya que ha sido uno de los partícipes de que los desarrolladores se impliquen y proporcionen un soporte a todas estas tecnologías.

Ahora debemos analizar lo expuesto anteriormente y realizar la elección de los dispositivos que mejor se adaptan a nuestras necesidades y puedan satisfacer los requisitos de este proyecto.

Capítulo 3

Descripción del sistema

3.1 Introducción.

En el capítulo anterior se estudiaron las diferentes soluciones que se presentan para trabajar en este proyecto. Se trataron los distintos dispositivos que se encuentran actualmente en el mercado y las posibilidades que poseían cada uno.

Teniendo en cuenta las prestaciones que debería tener el dispositivo a desarrollar, se ha realizado una elección de los componentes que se utilizaran para llevarlo a cabo, así como los consiguientes sensores y herramientas para que cumpla su funcionalidad.

3.2 ESP32.

Para la realización del trabajo se ha elegido el SoC (System on Chip) de Espressif Systems. La elección del ESP32 se ha basado principalmente en la capacidad del dispositivo en realizar tanto las tareas de cómputo como de comunicación, ya que como se ha podido leer en el capítulo anterior, posee un microprocesador capaz de satisfacer las necesidades del proyecto, además de proporcionar distintas alternativas para realizar la comunicación.

Con este SoC se cumplen las expectativas propuestas sobre el bajo consumo y bajo coste de los dispositivos con los que trabajar. Por lo que parece un componente adecuado para cumplir con lo requerido al hacer el planteamiento de este proyecto.

Se situaba como el sucesor del ESP8266, aportando nuevas características y grandes mejoras respecto al otro modelo, pero sus fabricantes han dado un paso más. Diseñado pensando en el Internet de las Cosas (*IoT*), trabaja con un procesador Xtensa Dual-Core LX6 de 32 bits, se sitúa en una elevada posición respecto a las ofertas actuales del mercado para un uso industrial.

Actualmente se puede encontrar en el mercado como módulo de superficie o ya incorporados en una placa de desarrollo. Ambas opciones las proporciona el propio fabricante, pero también se pueden encontrar versiones de otros fabricantes con pequeñas modificaciones entre ellas. Lo que indica que es una opción bastante demandada en el mercado.

3.2.1 Características generales.

Necesitamos conocer más en profundidad las funciones que nos puede aportar este dispositivo, para ello es necesario un buen conocimiento de sus características técnicas. Con la visualización a continuación del diagrama de bloques, damos comienzo a la descripción de las especificaciones.

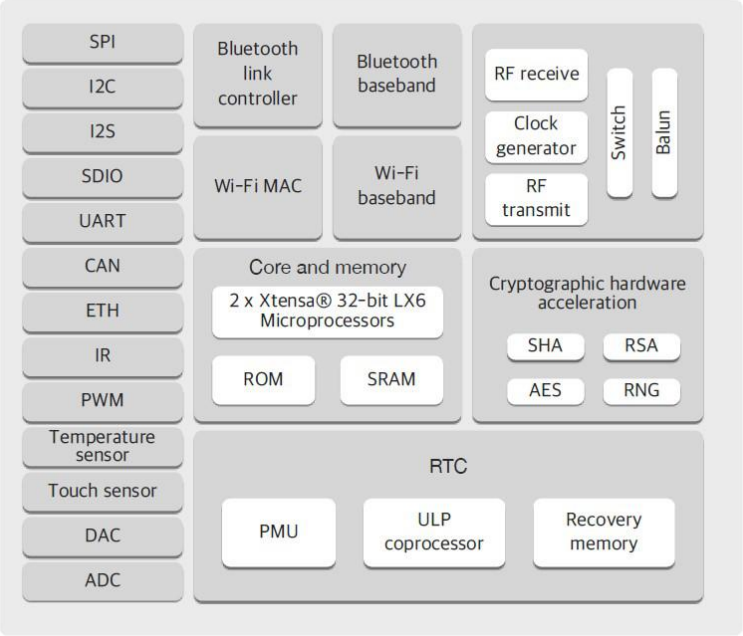

Ilustración 3.1. Diagrama de Bloques de ESP32.

3.2.1.1 Procesadores.

Empezamos por analizar su procesador principal. Un microprocesador Tensilica Xtensa LX6 de 32 bits, con dos núcleos de procesamiento, con una frecuencia de reloj de hasta 240 MHz y un rendimiento de hasta 600 DMIPS. Gracias a los dos núcleos, se resuelven grandes problemas en la gestión de los procesos.

Otro de los potenciales de este dispositivo es la disponibilidad de un procesador secundario. El coprocesador ULP (ultra baja energía, o siguiendo su terminación original, *Ultra Low Power*) es una máquina de estados simple diseñada para realizar mediciones utilizando un ADC, el sensor de temperatura y sensores externos I^2C, mientras el procesador principal se encuentra en modo de sueño profundo.

3.2.1.2 Conectividad inalámbrica.

Respecto a la conectividad inalámbrica nos encontramos con las dos principales posibilidades que aporta este SoC:

- WiFi: 802.11 b /g / n / e / i (802.11n @ 2.4 GHz hasta 150 Mbit/s).
- Bluetooth: v4.2 BR / EDR y Bluetooth Low Energy (BLE).

3.2.1.3 Memoria.

Cuando hablamos de la memoria debemos diferenciar entre las diferentes formas de memoria que podemos encontrar. En el caso de la memoria interna, destacamos los siguientes tipos:

- Memoria ROM, 448KB, destinados al arranque y a las funciones principales.
- Memoria SRAM, 520 KB, destinados a los datos e instrucciones.
- RTC rápido SRAM, 8 KB, para almacenamiento de datos y CPU principal durante el arranque RTC desde el modo de reposo profundo.
- RTC lento SRAM, 8KB, para acceder al coprocesador durante el modo de reposo profundo.
- eFuse, 1 Kbit, de los cuales 256 bits se utilizan para el sistema y los 768 bits restantes están reservados para aplicaciones de clientes.
- Flash incorporado, conectado internamente a través de los pines, variando la capacidad y el número de pin dependiendo del módulo seleccionado.

Respecto al Flash externo y SRAM, el ESP32 admite hasta cuatro flashes QSPI externos de 16 MB y SRAM con cifrado de hardware basado en AES para proteger los programas y datos de los desarrolladores.

En la siguiente imagen podemos ver la estructura de la asignación de direcciones de memoria.

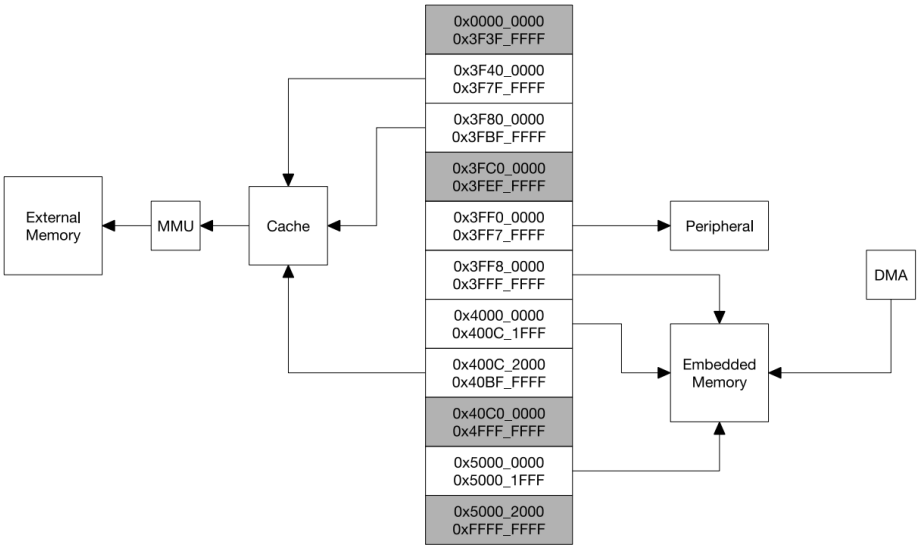

Ilustración 3.2. Estructura de mapeo de direcciones.

3.2.1.4 Seguridad.

- Todas las funciones de seguridad estándar IEEE 802.11 son compatibles, incluidas WFA, WPA / WPA2 y WAPI.
- Arranque seguro.
- Cifrado flash.
- OTP de 1024 bits, hasta 768 bits para clientes.
- Aceleración de hardware criptográfico: AES, SHA-2, RSA, criptografía de curva elíptica (ECC), generador de números aleatorios (RNG).

3.2.1.5 Gestión de baja energía y RTC.

Una de las prestaciones que proporciona este dispositivo es la gestión de un bajo consumo energético, que se realiza gracias a los distintos tipos de ahorro de energía que presenta:

- *Active mode.* El chip está activo y realiza sus tareas de transmisión y recepción de datos.
- *Moden-sleep mode.* La CPU está operativa y el reloj es configurable. La banda base y la radio de WiFi / Bluetooth están deshabilitadas.
- *Light-sleep mode.* La CPU está en pausa. Se ejecutan la memoria y los periféricos RTC, así como el coprocesador ULP. Cualquier evento de activación activará el chip.

- *Deep-sleep mode.* Solo se encienden la memoria RTC y periféricos RTC. Los datos de conexión WiFi y Bluetooth se almacenan en la memoria RTC. El coprocesador ULP es funcional.
- *Hibernation mode.* El oscilador interno de 8 MHz y el coprocesador ULP están desactivados. La memoria de recuperación de RTC está apagada. Solo un temporizador RTC en el reloj lento y ciertos GPIO-RTC están activos. El temporizador RTC o los GPIO-RTC pueden activar el chip desde el modo de Hibernación.

A continuación observamos una tabla resumen aportada por el fabricante con algunos datos de consumo de energía en función de cada modo.

Power mode	Description			Power consumption
Active (RF working)	Wi-Fi Tx packet			
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered on.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP co-processor is powered on.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA
Hibernation	RTC timer only			5 μA
Power off	CHIP_PU is set to low level, the chip is powered off.			0.1 μA

Tabla 3.1. Consumos energéticos por modos de energía de ESP32.

3.2.1.6 Características eléctricas.

Teniendo en cuenta que una parte importante a conocer antes de empezar a trabajar con cualquier dispositivo son los valores eléctricos que soporta con el fin de no estropear ni dañar ningún componente, mostramos la tabla aportada por el fabricante de las condiciones recomendadas de operación por parte del ESP32.

Symbol	Parameter	Min	Typical	Max	Unit
VDDA, VDD3P3_RTC [1] VDD3P3, VDD_SDIO (3.3 V mode) [2]	Voltage applied to power supply pins per power domain	2.3	3.3	3.6	V
VDD3P3_CPU	Voltage applied to power supply pin	1.8	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T [3]	Operating temperature	–40	-	125	°C

Tabla 3.2. Valores eléctricos recomendados para ESP32.

Con estos valores nos aseguramos un rendimiento óptimo de lo indicado por parte del fabricante.

3.2.2 Sensores y periféricos.

Un punto importante para la selección de un dispositivo son los sensores y periféricos de que dispone, ya que van a ser los encargados de permitir al dispositivo relacionarse con el medio exterior, ya sea a través de lecturas de otros dispositivos o componentes, o directamente a través de la medición de sensores. Para ello, se debe analizar qué nos ofrece ESP32.

3.2.2.1 Interfaz de entrada / salida de uso general. (GPIO).

El chip dispone de 34 pines GPIO que permiten la asignación de diversas funciones mediante la programación de los registros apropiados. Dentro del número total de pines hay de distintos tipos, digitales, analógicos, capacitivos táctil, etc. Por lo que a la hora de elegir un pin para realizar una función, es necesario consultar anteriormente su configuración inicial.

La mayoría de estos pines pueden configurarse como *pull-up* o *pull-down* internos, o en gran impedancia. También, la mayoría de los pines de I/O digitales son bidireccionales, no inversos y triestados, incluidas las memorias intermedias de entrada y salida con control triestado. Estos pines se pueden multiplexar con otras funciones como SDIO, UART, SPI, etc.

Para las operaciones de baja potencia los GPIO se pueden configurar para mantener sus estados.

3.2.2.2 Convertidor de analógico a digital. (ADC).

ESP32 integra ADC SAR de 12 bits y admite mediciones en 18 canales. El coprocesador ULP también está diseñado para medir el voltaje mientras funciona en modo inactivo, por lo que permite un bajo consumo de energía. Con la configuración adecuada los ADC se pueden configurar para medir el voltaje en 18 pines como máximo.

3.2.2.3 Convertidos de digital a analógico. (DAC).

Permite el uso de dos canales DAC de 8 bits para convertir dos señales digitales en dos salidas de señal de voltaje analógico. LA estructura de diseño se compone de cadenas de resistencia integradas y un buffer.

Este DAC dual permite la fuente de alimentación como referencia de voltaje de entrada. Los dos canales DAC también pueden admitir conversiones independientes.

3.2.2.4 Sensor Hall.

Otra curiosidad es la incorporación de un sensor efecto Hall basado en una resistencia N-carrier. Cuando el chip está en el campo magnético, el sensor Hall desarrolla un pequeño voltaje lateralmente en la resistencia, que el ADC puede medir directamente.

3.2.2.5 Sensor táctil.

El ESP32 tiene 10 GPIO de detección capacitiva, que permiten detectar variaciones inducidas al tocar o acercarse a los GPIO con un dedo u otros objetos. La naturaleza de bajo ruido del diseño y la alta sensibilidad del circuito permite utilizar almohadillas relativamente pequeñas.

También se pueden utilizar matrices de almohadillas, de modo que se pueda detectar un área más grande o más puntos. Los 10 GPIO de detección capacitiva son los siguientes:

Capacitive-sensing signal name	Pin name
T0	GPIO4
T1	GPIO0
T2	GPIO2
T3	MTDO
T4	MTCK
T5	MTDI
T6	MTMS
T7	GPIO27
T8	32K_XN
T9	32K_XP

Tabla 3.3. GPIO de detección capacitiva disponibles en ESP32.

3.2.2.6 Procesador de ultra baja potencia. (ULP).

El procesador ULP y la memoria RTC permanecen encendidos durante el modo de reposo profundo. Por lo que permite almacenar un programa para el procesador ULP en la

memoria lenta RTC para acceder a los dispositivos periféricos, temporizadores internos y sensores internos durante el modo de reposo profundo.

3.2.2.7 Controlador de Host SD/SDIO/MMC.

El controlador permite hasta 80 MHz de salida de reloj en tres modos diferentes de bus de datos: 1 bit, 4 bits y 8 bits. Admite dos tarjetas SD/SDIO/MMC4.41 en modo de bus de datos de 4 bits. También es compatible con una tarjeta SD que funciona a 1.8V.

3.2.2.8 Controlador esclavo SDIO/SPI.

ESP32 integra una interfaz de dispositivo SD que se ajusta a la versión 2.0 de la especificación de tarjeta SDIO estándar de la industria, y permite que un controlador host acceda al SoC utilizando la interfaz y protocolo del bus SDIO.

3.2.2.9 Transmisor / receptor asíncrono universal. (UART).

El chip dispone de tres interfaces UART, que son UART0, UART1 y UART2, que proporcionan comunicación asíncrona (RS232 y RS485) y soporte IrDA, que se comunican a una velocidad de hasta 5 Mbps. UART proporciona gestión de hardware de las señales CTS y RTS, y control de flujo software (XON y XOFF). Se puede acceder a todas las interfaces mediante el controlador DMA o directamente desde la CPU.

3.2.2.10 Interfaz I^2C.

En el caso del I^2C, disponemos de dos interfaces de bus I^2C que pueden servir como maestro o esclavo, según las necesidades requeridas. Las interfaces admiten un modo estándar de 100 Kbit/s, un modo rápido de 400 Kbit/s, hasta 5 MHz limitados por la fuerza de extracción de SDA, un modo de direccionamiento de 7bit/10bit y un modo de direccionamiento dual.

También se pueden programar registros de comandos para controlar las interfaces, de modo que proporcionen más flexibilidad.

3.2.2.11 Interfaz I2S.

Dos interfaces I2S estándar están disponibles en el ESP32. Pueden operar en modo maestro o esclavo, en modos de comunicación *full duplex* y *half duplex*, y se pueden configurar

para operar en una resolución de 8/16/32/48/64 bits como canales de entrada o salida. Admiten una frecuencia de reloj BCK desde 10 kHz hasta 40 MHz.

3.2.2.12Control remoto por infrarrojos.

El controlador remoto por infrarrojos admite ocho canales de transmisión y recepción remota por infrarrojos. AL programar la forma de onda de pulso, admite varios protocolos de infrarrojos. Ocho canales comparten un bloque de memoria de 512x32 bits para almacenar la forma de onda de transmisión o recepción.

3.2.2.13Control de pulsos.

El contador de pulsos captura el pulso y cuenta los bordes de pulso a través de siete modos. Tiene ocho canales, cada uno de los cuales captura cuatro señales a la vez. Las cuatro señales de entrada incluyen dos señales de pulso y dos señales de control.

3.2.2.14Modulación de ancho de pulso. (PWM).

El controlador de modulador de ancho de pulso se puede utilizar para conducir motores digitales y luces inteligentes. Consta de temporizadores PWM, operador PWM y un submódulo de captura dedicado.

3.2.2.15Interfaz periférica en serie. (SPI).

ESP32 presenta tres SPI (SPI, HSPI y VSPI) en modo esclavo y maestro en modos de comunicación *full duplex* de 1 línea y *half duplex* de ½ línea. También admiten las siguientes características:

- Cuatro modos de formato de transferencia SPI, que dependen de la polaridad (CPOL) y la fase (CPHA) del reloj SPI.
- Hasta 80 MHz.
- FIFO de hasta 64 bytes.

Todos los SPI también se pueden conectar al flash/SRAM externo y LCD. Cada SPI puede ser servido por controladores DMA.

3.2.3 PINOUT.

Una vez analizados los periféricos, se hace necesario conocer los pines de conexión de los que dispone ESP32.

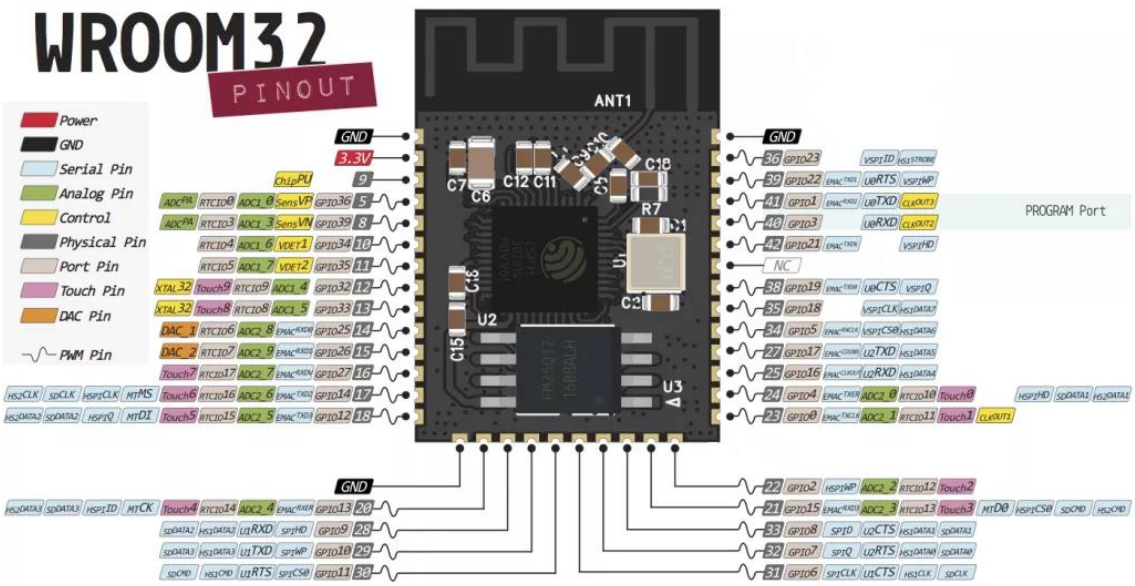

Ilustración 3.3. ESP32 SoC pinout.

En la imagen anterior podemos ver los pines de salida del, pero en este trabajo se va a utilizar una placa de desarrollo con el chip ya incorporado. En la siguiente imagen se muestra la distribución de pines en una placa de desarrollo.

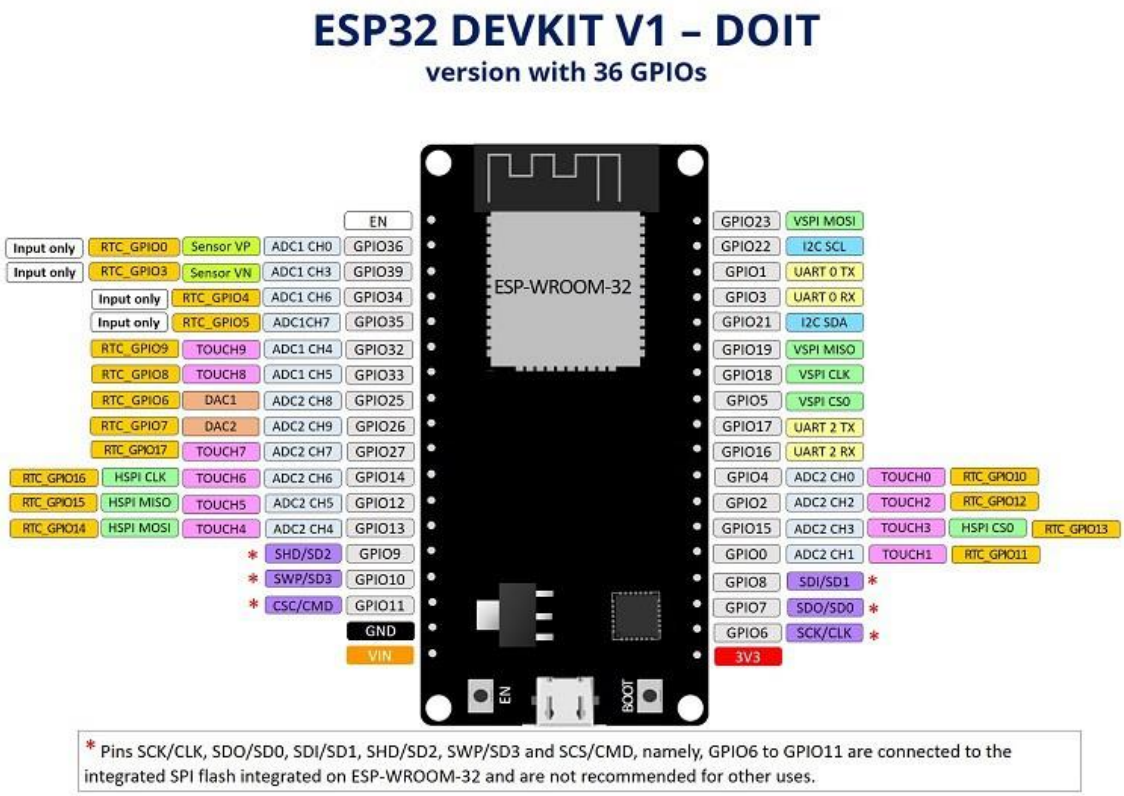

Ilustración 3.4. ESP32 Devkit pinout.

3.2.4 ESP-IDF.

Como vimos en el capítulo anterior, Espressif ofrece un entorno de desarrollo oficial para ESP32. Ya que está la posibilidad de trabajar sobre el entorno proporcionado por el fabricante, se ha valorado positivamente la opción de utilizar un soporte oficial para el desarrollo del proyecto, por lo que será la herramienta a utilizar en este proyecto.

El entorno de programación se llama ESP-IDF y está basado en el sistema Linux, pero el propio fabricante ofrece las herramientas para poder adaptarlo y utilizarlo en otros sistemas operativos.

Otra de las ventajas de este entorno, es que hay también a disposición una guía de programación oficial en la que se puede encontrar librerías, definición de funciones, ejemplos de uso e innumerables recursos que resultan imprescindibles para exigirle el máximo al dispositivo. Todo esto con el soporte del fabricante oficial y la ayuda de la comunidad de desarrolladores que aportan su contribución en la resolución de errores.

3.3 STM32.

Para darle una mayor polivalencia al proyecto y así mejorar las características del sistema final, en forma de la posibilidad de añadir un número mayor de sensores o periféricos, se ha pensado la opción de añadir un segundo microcontrolador al sistema para mejorar todo lo anterior.

De esta forma es posible la creación de un sistema con varias placas periféricas que realicen funciones diferentes entre ellas.

Para realizar lo antes expresado, se ha elegido un micro de la familia STM32, en concreto el dispositivo STM32F103C8, ya que según sus características ofrece un buen rendimiento y puede realizar las funciones deseadas de forma satisfactoria. A continuación se analizaran algunas de estas características que se han tenido en cuenta para su elección.

3.3.1 Características generales.

Para resaltar brevemente algunas de las características generales de este microcontrolador, haremos un breve resumen de lo más destacado, ya que será el ESP32 quien realice las principales tareas de gestión y procesado de datos.

- Posee como procesador, el ARM de 32 bit Cortex-M3 CPU Core, que trabaja a una frecuencia de 72 MHz (1.25 DMIPS/MHz).
- Voltaje de operación de 3.3V.
- En referencia a la memoria, tiene 64 Kbyte de memoria Flash y 20 Kbyte de SRAM.
- Dispone de un RTC integrado, con suministro de voltaje de batería para el RTC y registros de respaldo.
- Respecto al consumo de energía, tiene distintos modos de ahorro energético, como son los modos *Sleep*, *Stop* y *Standby.*
- Dos convertidores A/D de 12 bits y 1 µs, de hasta 16 canales. Con un rango de conversión de 0 a 3.6V, una capacidad de doble muestra y retención y sensor de temperatura.
- Controlador DMA de 7 canales.
- Periféricos compatibles como timers, ADC, SPIs, I^2Cs y USARTs.
- 16 pines I/O digitales, la mayoría de ellos tolerables a 5V. Interrupciones disponibles.
- Interfaz de depuración de cable serie (SWD) y JTAG.

Tras este pequeño resumen de algunas de las características más importantes que podemos encontrar en el dispositivo, a continuación tenemos una tabla aportada por el fabricante en la que se simplifica aún más lo antes comentado.

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	**General-purpose**	3		3		3		3	
	Advanced-control	1		1		1		1	
Communication	**SPI**	1		2		2		2	
	I^2C	1		2		2		2	
	USART	2		3		3		3	
	USB	1		1		1		1	
	CAN	1		1		1		1	
GPIOs		26		37		51		80	
12-bit synchronized ADC Number of channels		2 10 channels		2 10 channels		2 16 channels(1)		2 16 channels	
CPU frequency		72 MHz							
Operating voltage		2.0 to 3.6 V							
Operating temperatures		Ambient temperatures: -40 to +85 °C / -40 to +105 °C Junction temperature: -40 to + 125 °C							
Packages		VFQFPN36		LQFP48, UFQFPN48		LQFP64, TFBGA64		LQFP100, LFBGA100, UFBGA100	

Tabla 3.4. Características generales STM32.

3.3.2 PINOUT.

Para el conocer un poco más acerca de este dispositivo, se hace necesario conocer los pines de salida que nos ofrece. En la siguiente ilustración se observa de forma bastante intuitiva las disponibilidades que tiene respecto a los periféricos y pines de I/O.

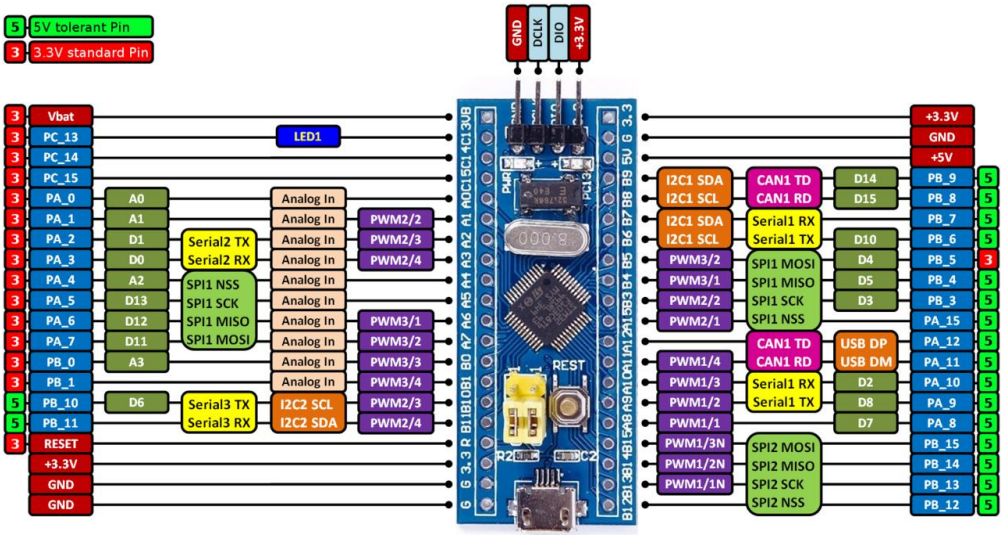

Ilustración 3.5. Pinout placa desarrollo STM32F203C8T6.

Aunque no se va a hacer uso de él de forma independiente a su placa de desarrollo, también se considera importante conocer la distribución de pines del microcontrolador.

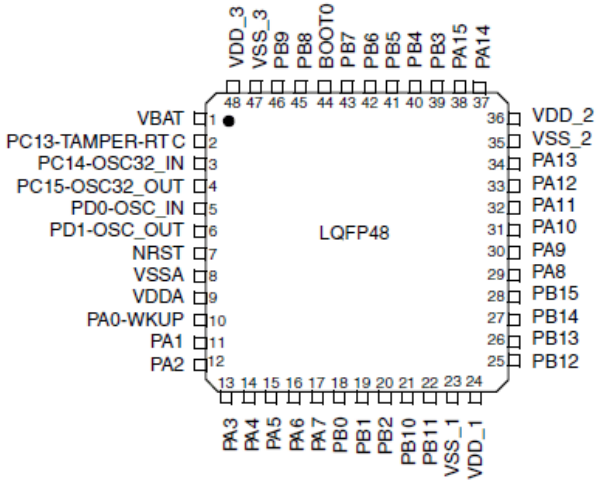

Ilustración 3.6. Pinout micro STM32F103C8T6.

3.3.3 Arduino IDE.

Para la programación del microcontrolador se ha considerado la utilización del entorno de programación de Arduino.

IDE de Arduino ofrece la posibilidad de adaptarse a la programación de esta tarjeta gracias a la contribución de desarrolladores y la política de *Open Source* de Arduino.

3.4 AppInventor.

Para realizar una comprobación exitosa del correcto funcionamiento del proyecto se hace necesaria la creación de una aplicación móvil con la cual poder interactuar con el dispositivo para realizar la lectura y extracción de datos.

Ya que para el desarrollo de esta aplicación se ha elegido el sistema operativo Android por disponibilidad del dispositivo, se ha tenido muy en cuenta la herramienta de AppInventor por encima de la otra posibilidad, que era Android Studio.

AppInventor ofrece la posibilidad de crear aplicaciones móviles sin la necesidad de un amplio conocimiento de la programación, gracias a unos sencillos conceptos de unión de bloques y lógica.

La finalidad de la creación de esta aplicación móvil es meramente la comprobación del correcto funcionamiento del sistema, lo que ha imperado la elección de esta opción debido a su sencillez.

3.5 Bluetooth Low Energy.

La comunicación entre el dispositivo móvil y el sistema electrónico se realizará por medio del Bluetooth Low Energy.

Como se ha comentado en el capítulo anterior, esta tecnología en constante evolución permite la transmisión de datos con un escaso consumo energético, por lo que es una tecnología ideal para su uso en este trabajo.

Para conocer algo sobre el funcionamiento de esta tecnología se van a dar una breve información que nos servirá para saber cómo funciona.

3.5.1 GAP.

Acrónimo de *Perfil de Acceso Genérico*, es quien controla las conexiones y la publicidad en Bluetooth. Es lo que permite que el dispositivo sea visible para el resto de dispositivos.

3.5.2 GATT.

Acrónimo de *Perfil de Atributo Genérico*, es quien define la forma en la que dos dispositivos Bluetooth Low Energy interactúan entre ellos y transfieren datos de uno a otro utilizando conceptos como servicios y características.

Utiliza un protocolo de datos genérico llamado *Attribute Protocol* (ATT), que gestiona el almacenamiento de servicios, características y datos relacionados con un ID de 16 bits.

Un concepto clave para entender las transacciones GATT es la relación servidor/cliente. En la que se realiza un intercambio de datos entre ellos.

3.5.2.1 Servicios y características.

Las transacciones GATT están basadas en objetos anidados de alto nivel llamados Perfiles, Servicios y Características.

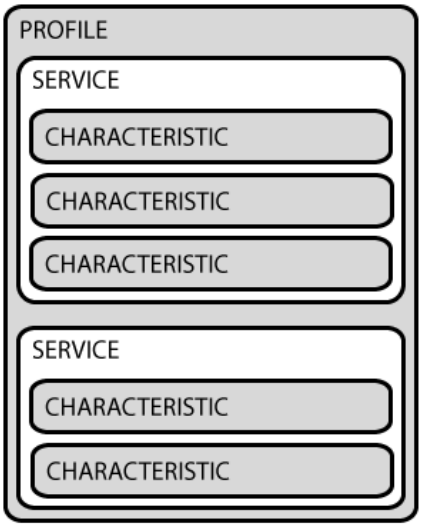

Ilustración 3.7. BLE. Perfil, Servicios y Características.

Un perfil es una colección predefinida de servicios. Existe una lista de perfiles adaptados oficialmente, al igual que en los servicios y características.

Los servicios son utilizados para separar los datos en entidades lógicas. Contienen fragmentos específicos de datos llamados características. Un servicio puede tener una o más características. Cada servicio se distingue de otro por medio de un ID numérica llamada UUID, que está formada por 16 o 128 bits.

Las características son el concepto más bajo de las transacciones GATT, en la que se encapsulan un único punto de datos. Al igual que en los anteriores, se distingue del resto con un UUID personalizado.

De esta forma se estructuran los dispositivos BLE para realizar la comunicación entre ellos.

3.6 Alimentación

Para la alimentación del sistema se estudiarán las alternativas en las fuentes de alimentación y sistemas de carga más adecuados.

Como fuente de alimentación se presenta como principal opción una batería de Li-ion, ya que son las más utilizadas y sus características son adecuadas para las especificaciones del sistema. Para permitir que la batería cumpla su función un mayor tiempo, se empleará un sistema de carga para mantener al sistema con el mejor rendimiento posible en lo referente a la energía. Siempre se tendrá en cuenta la necesidad de aportar convertidores de continua-continua para realizar amplificaciones o reducciones de voltaje según lo requieran los componentes que conformen el sistema.

Capítulo 4

Hardware

4.1 Introducción.

En este capítulo se va a describir los distintos dispositivos que formarán parte del proyecto, al igual que todos los componentes que darán cuerpo al sistema general. Como cualquier sistema electrónico, este proyecto está formado por distintos dispositivos que juntos satisfagan una necesidad, a continuación detallaremos con cuales se ha dispuesto.

4.2 Componentes del sistema.

Primero de todo se van a citar algunos de los principales componentes que darán cuerpo al sistema.

4.2.1 ESP32.

Como se ha comentado en el capítulo anterior, la pieza principal de este trabajo es el SoC de Espressif. En esta ocasión contamos con una placa de desarrollo en la que está montado este chip.

Ilustración 4.1. ESP32 devkit.

Esta tarjeta hará las funciones principales de gestión de datos y recursos, por lo que será donde se realice un desarrollo más importante.

4.2.2 STM32.

Como también se ha comentado anteriormente en otros capítulos, el microcontrolador STM32 también va a formar parte del sistema. Con este micro se pretende liberar al ESP32 de la adquisición de datos recibidos de los sensores, y tomarlo como punto de partida para la ampliación en módulos que permitan una gestión "inteligente" entre ellos.

Ilustración 4.2. STM32 devkit.

Dispone de un mayor número de ADCs por lo que facilita la conexión entre distintos dispositivos sensores. Al igual que en el caso del ESP32, se va a utilizar montado en la placa de desarrollo aportada por el fabricante.

4.2.3 Módulo µSD.

Para el almacenamiento de datos se utilizará un adaptador para una µSD en la que se realizará la escritura de los valores medidos por el sistema.

Ilustración 4.3. Módulo µSD.

4.2.4 Placa sensor.

Para la conexión con el sensor y la placa solar, se utilizará una tarjeta desarrollada anteriormente en el Departamento de Tecnología Electrónica (DTE)[3] de la Universidad Politécnica de Cartagena (UPCT)[4].

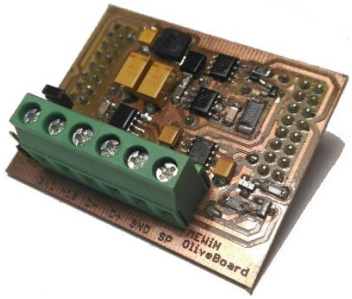

Ilustración 4.4. Placa sensor.

4.2.5 Dendrómetro.

En la realización de la medida, se utilizará un dendrómetro, en concreto el modelo D6. Un dendrómetro es un sensor de crecimiento de árboles que a través de la medición del crecimiento del grosor de su tronco o tallo principal permite realizar un seguimiento de la progresión de crecimiento.

Ilustración 4.5. Dendrómetro D6.

La medida se obtiene gracias a unas galgas extensiométricas que convierten la presión que ejerce el tronco del árbol sobre ellas, creando una extensión de las mismas, en unos valores de resistencia eléctrica.

4.2.6 Batería.

La alimentación se realizará a través de una batería Li-ion como la de la imagen siguiente, de un voltaje de 3.7v y 1800mAh de capacidad de almacenamiento eléctrico.

4.2.7 Placa solar.

El mantenimiento de energía de la batería se realizará a través de una placa solar, en el caso de este proyecto se ha elegido una placa solar de 5W, que aporta hasta 5v y 1A a condiciones más optimas de trabajo.

4.2.8 Convertidor DC/DC.

En cuanto a la obtención de un voltaje óptimo para los distintos componentes se utilizará un módulo de conversión de continua a continua. Servirá como elevador de voltaje de unos 3.7v hasta unos 5v.

El seleccionado corresponde al fabricante Pololu como se muestra en la imagen anterior.

4.3 Primera versión del sistema completo.

Tras la realización de una breve explicación de los principales componentes del trabajo, solo falta la descripción del sistema en su conjunto.

Para una primera versión se ha utilizado una placa perforada en la que se han montado y soldado todos los componentes para crear una apariencia orientativa del resultado final de la tarjeta. Con este fin se puede llegar a unos resultados bastantes certeros del

funcionamiento en el desarrollo completo, obteniendo nuevas ideas para mejorar y posibles defectos en la planificación o diseño del proyecto.

En la realización del montaje de esta placa se ha tenido en cuenta una búsqueda previa de una caja en la que realizar la instalación. La caja protectora debe tener protección total a los efectos meteorológicos, ya que se va a encontrar en la intemperie. Para ello debe seguir el estándar de protección IP, con un valor mínimo de IP66, que representa una protección completa contra el contacto, protección contra penetración de polvo y protección contra la penetración de agua en caso de inyección pasajera.

Para solventar la problemática de la estanqueidad, se ha recurrido a una caja del fabricante Fibox, que poseen una larga trayectoria en fabricación de este tipo de materiales.

Una vez solventado el tema del aislamiento, se procede a utilizar las medidas internas de capacidad y la toma de medidas para la fijación de la tarjeta dentro del recipiente, utilizando los orificios para la tornillería que aporta el fabricante.

Tras las medidas pertinentes, el siguiente paso es la colocación de los componentes sobre la placa. En esta situación cabe de gran importancia una jerarquía de posiciones que permita la mínima necesidad de cableado aproximando más entre ellos los componentes que más conexionado común necesitan. Dentro del conexionado, las situaciones más críticas, si se le puede denominar así, es el conexionado de comunicación.

Ilustración 4.6. Sistema completa primera versión.

Todos los cables por los que circula una corriente emiten ruido, y el ruido es uno de los problemas principales de las comunicaciones, por ello, se tiene que evitar en medida de lo posible los cables excesivos o que puedan interferir en las comunicaciones.

Después de realizar varias pruebas para cumplir de la mejor forma posible lo anteriormente citado, el resultado de esta primera versión realizada en una placa perforada parece que coge forma. En la imagen anterior se muestra la distribución que se ha utilizado en esta primera versión.

Como se ve en esta primera versión, se han conectado los componentes y se ha puesto la batería como sistema de alimentación. Pero para mantener un buen funcionamiento de la batería y que permita alimentar correctamente todo el sistema, se ha añadido una placa fotovoltaica. Esta placa se ha decidido colocar en el interior de la caja, optando por seleccionar una tapadera de material transparente que permita el paso de luz solar a través de ella. La solución ha quedado como se muestra en la siguiente imagen.

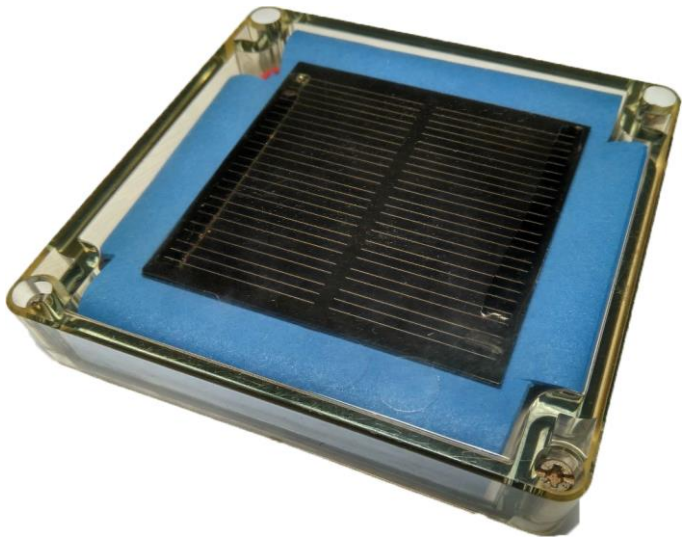

Ilustración 4.7. Placa fotovoltaica en la caja.

Viendo el resultado obtenido se considera un buen punto de partida en la resolución del proyecto, ya que cumple con satisfacción las indicaciones previas al inicio de este trabajo.

4.4 Segunda versión del sistema completo.

Con el afán de seguir continuando la línea del trabajo y aumentar los conocimientos en la materia, tras analizar los pros y los contras en la primera versión realizada, se procede a crear una segunda versión de la tarjeta de desarrollo. En esta ocasión se utiliza un software

de desarrollo de PCBs para el diseño de ello. El software utilizado para el desarrollo de esta fase ha sido KiCad[5], una aplicación para el diseño electrónico de software libre.

En esta segunda versión se han añadido a la tarjeta componentes como diodos led o pulsadores, a fin de facilitar la interactividad del usuario con el sistema y realizar mejoras en el rendimiento de los recursos.

4.4.1 Esquemático.

Para comenzar con el diseño, lo primero que se requiere es la creación de un esquemático. KiCad proporciona la herramienta de *Schematic Layout Editor*, en la que se puede diseñar un esquema eléctrico del diseño a realizar, al igual que permite la creación de componentes que se incluirán en el trabajo.

En la siguiente imagen se puede observar el esquemático del proyecto.

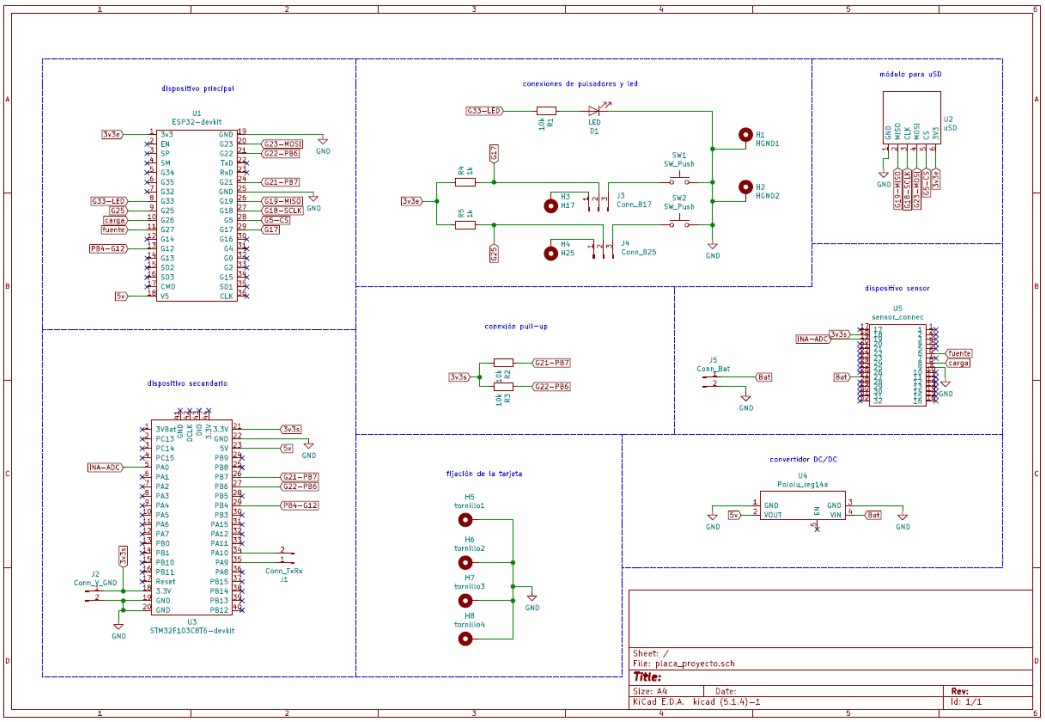

Ilustración 4.8. Esquemático del sistema.

Tal y como se muestra en la imagen anterior, se han ido organizando las conexiones de la forma más limpia en la manera de lo posible teniendo en cuenta la poca experiencia en la utilización de estos sistemas de diseño, para crear una mejor percepción de ellas, del mismo modo, en un rápido vistazo podemos localizar lo que necesitemos buscar.

Entrando a analizar los componentes, podemos ver por una parte las dos placas de desarrollo. Entre ellas se realiza una comunicación a través de I^2C, por lo que se añade un pull-up externo para alimentar el bus. El dispositivo ESP32 dispone por defecto de pull-up internos, pero tras varias pruebas se decidió a añadir de forma adicional unos de uso externo.

Otra de las conexiones que se realizan es la del ESP32 con el módulo de la μSD. En esta ocasión se realiza mediante SPI.

Del dispositivo que realizara la función de periférico con el sensor extraemos desde un amplificador de instrumentación, a través de una línea ADC del STM32, la señal recogida del sensor a utilizar, en este caso el dendrómetro. De esta misma tarjeta salen las conexiones con la placa solar y la batería, de la cual esta última se puede monitorizar el valor de su carga a través de otra línea que en este caso comunica con ESP32, ya que será quien realice las funciones principales del sistema.

La alimentación se realizará a unos 5 voltios desde la placa fotovoltaica y alrededor de unos 3.7 voltios desde la batería. Para unificar el voltaje se dispone a conectar un convertidor continua - continua para mantener la alimentación del resto de componentes en un valor constante de 5v.

Este valor de 5 voltios alimenta tanto a la placa de desarrollo del ESP32 como a la del STM32, que ambas disponen de reguladores de tensión incorporados. Y a partir de la alimentación de 3.3v que suministran cada una de ellas se alimentan el resto de componentes electrónicos. Ya que prácticamente toda la electrónica utilizada en este trabajo tiene un voltaje de operación de 3.3 voltios.

En un primer momento se pensó para esta segunda versión del sistema la reducción de parte de la electrónica que componen las placas de desarrollo para evitar consumos innecesarios y mejorar lo referente al tamaño, siendo la alimentación total de la electrónica de 3.3 voltios a partir de un único regulador de voltaje, pero debido a ciertas situaciones inesperadas y la escasez de tiempo se optó por montar el sistema con las placas de desarrollo completas.

Finalmente, como se ha comentado al principio de este apartado, se añadieron unos pulsadores para controlar la activación del Bluetooth y gestionar los tiempos de bajo consumo, al igual que un diodo led para poder observar directamente desde la tarjeta si se encuentra en funcionamiento.

4.4.2 PCB.

Tras finalizar la parte del esquemático, le llegó el turno ya al diseño de la PCB, utilizando el *PCB Layout Editor* de KiCad. Para empezar con el diseño es necesario la creación de todas las huellas de los componentes a utilizar.

Una vez se tiene todo lo anteriormente nombrado preparado, se dio comienzo al posicionamiento de los componentes y a la elección de la configuración más óptima para crear todas las pistas de la tarjeta.

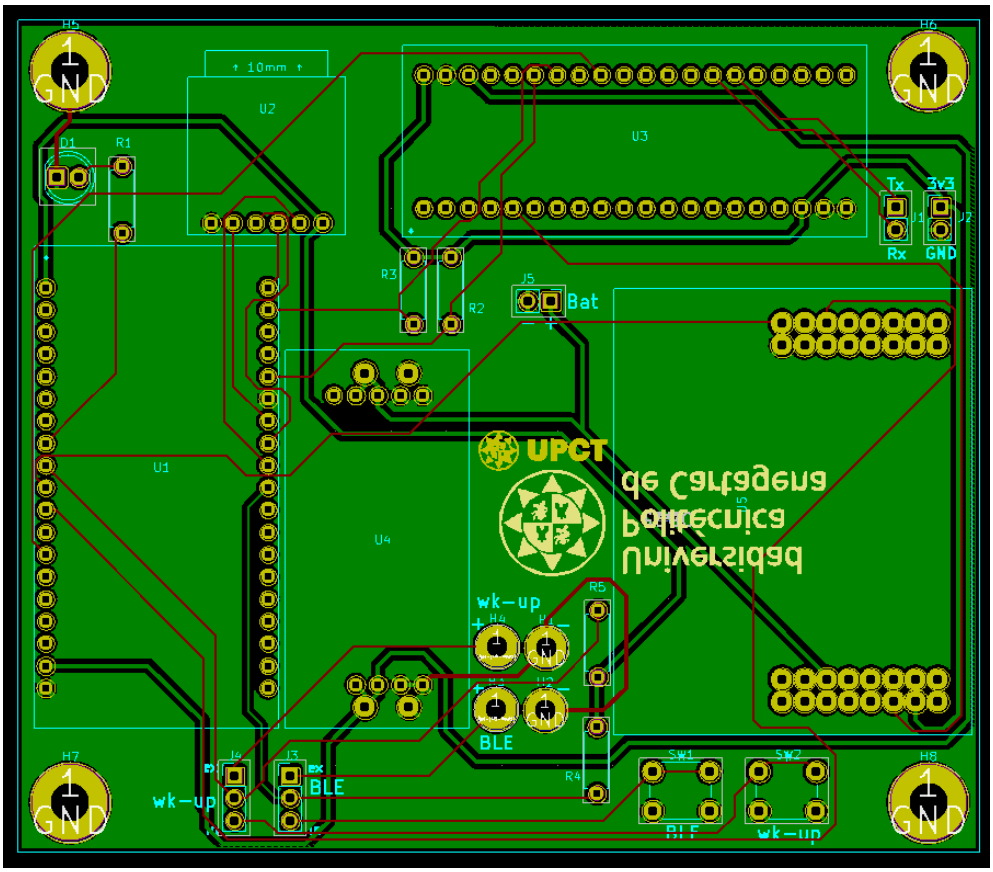

Ilustración 4.9. Diseño PCB del sistema.

Para su realización se diseñó una tarjeta a dos caras, y se tuvo en cuenta la diferenciación de pistas entre datos y alimentación, tal y como recomienda en el software y que facilita su obtención con distintas herramientas de cálculo con funciones matemáticas.

Otra de las funcionalidades que aporta este software es la de la pre visualización de un diseño 3D donde se puede apreciar una imagen aproximada del resultado final del diseño.

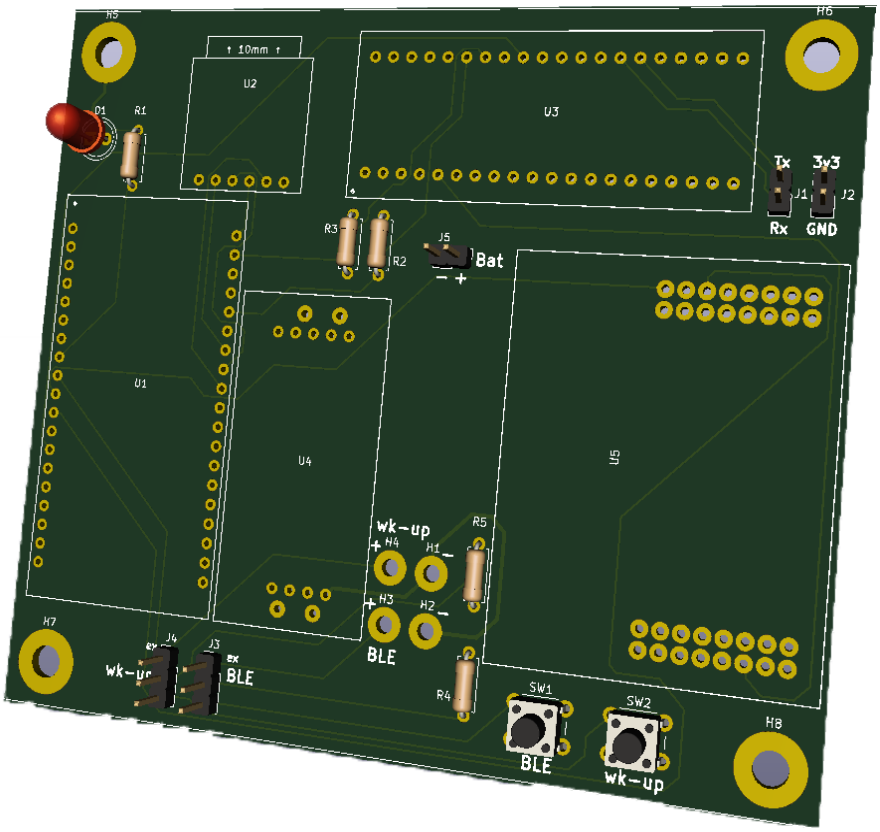

Ilustración 4.10. Diseño PCB en 3D.

Capítulo 5

Software

5.1 Introducción.

En el capítulo anterior se describió el hardware del sistema, en este capítulo de describirá la parte software que acompaña al hardware.

Cabe destacar que este capítulo de dividirá en distintas partes o apartados, ya que se ha realizado programación sobre tres componentes distintos, como son ESP32, STM32 y la aplicación Android para poder realizar un testeo del buen funcionamiento del sistema.

5.2 Programación de ESP32.

Otra vez más situamos el ESP32 como centro del sistema, a partir de él se creará el cuerpo principal del software del sistema completo.

La programación de este dispositivo se ha realizado a través de la plataforma de desarrollo oficial, ESP-IDF.

En lo referente a la programación, el dispositivo realiza una secuencia de acciones y posteriormente entra en modo de bajo consumo para no disminuir la carga de las baterías en exceso. Tras cierto tiempo en reposo o bajo consumo, el dispositivo se despierta de forma automática y vuelve a general la secuencia de acciones. También se ha implementado una

forma de despertar al dispositivo en algún momento que el usuario lo requiera para hacer alguna medida o realizar alguna comprobación mediante un pulsador conectado a un GPIO del dispositivo.

Cada ocasión que se despierta, el dispositivo inicializa todos los componentes así como la conexión de los periféricos y realiza una búsqueda para localizar los periféricos conectados. La inicialización se lleva a cabo con las siguientes funciones.

```
rtc_iniciar();
sd_iniciar();
sensor_iniciar();
```

Las funciones están organizadas en diferentes archivos para facilitar la organización y la localización en el código.

La puesta a hora del RTC se realiza a través de WiFi mediante una conexión a un servidor. La conexión WiFi se realiza únicamente al principio de la puesta en marcha del dispositivo y se desconecta automáticamente al obtener la información. Una vez que se ha realizado la actualización de la hora no será necesario volver a realizar ninguna conexión en los posteriores momentos de bajo consumo, ya que el procesador ULP del micro permite almacenar la información.

```
void rtc_iniciar(){

       time_t now;  struct tm timeinfo; time(&now);

       localtime_r(&now, &timeinfo);

       if (timeinfo.tm_year < (2016 - 1900)) {

             ESP_LOGI(TAG1, "Time is not set yet. Connecting to WiFi and
getting time over NTP.");

             obtain_time();

             time(&now);   }

       char strftime_buf[64];

       setenv("TZ", "GMT-2", 1); tzset();     localtime_r(&now, &timeinfo);

       strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);

       ESP_LOGI(TAG1, "The current date/time in Spain is: %s",
strftime_buf);

       strcpy(buf_hora,strftime_buf);

}
```

En la inicialización de la tarjeta μSD el dispositivo establece la conexión y comprueba la información accesible de las características de la tarjeta introducida en el módulo.

```
void sd_iniciar(void){

    ESP_LOGI(TAG, "Initializing SD card");

    ESP_LOGI(TAG, "Using SPI peripheral");

    sdmmc_host_t host = SDSPI_HOST_DEFAULT();

    sdspi_slot_config_t slot_config = SDSPI_SLOT_CONFIG_DEFAULT();

    slot_config.gpio_miso = PIN_NUM_MISO;

    slot_config.gpio_mosi = PIN_NUM_MOSI;

    slot_config.gpio_sck  = PIN_NUM_CLK;

    slot_config.gpio_cs   = PIN_NUM_CS;

    esp_vfs_fat_sdmmc_mount_config_t mount_config = {

        .format_if_mount_failed = false,

        .max_files = 5

    };

    sdmmc_card_t* card;

    esp_err_t ret = esp_vfs_fat_sdmmc_mount("/sdcard", &host, &slot_config,
&mount_config, &card);

    if (ret != ESP_OK) {if (ret == ESP_FAIL) {

            ESP_LOGE(TAG, "Failed to mount filesystem. "

                "If you want the card to be formatted, set format_if_mount_failed =
true.");

        } else {ESP_LOGE(TAG, "Failed to initialize the card (%d). " "Make sure SD
card lines have pull-up resistors in place.", ret);

       }return; }

    sdmmc_card_print_info(stdout, card);

}
```

El proceso de lectura y escritura se realizará posteriormente una vez que se hayan realizado las medidas a almacenar.

La función de iniciar el sensor activa los parámetros de inicialización del bus I^2C y espera a que se realice una petición.

Tras realizar estas acciones se realiza la lectura del sensor, la activación del Bluetooth por si fuese necesario una conexión, y se prepara el dispositivo para entrar en el modo de bajo consumo.

```
sensor_leer();
ble_iniciar();
rtc_dormir(suenyo);
```

Previamente, para realizar la búsqueda del sensor se realiza una búsqueda de los dispositivos conectados y se comprueban las direcciones. En la lectura, realmente solo se procesa el valor que se recibe desde el STM32, ya que es el encargado de realizarla.

```
void sensor_buscar(){
    printf("Escaneando el bus...\t");
    uint8_t devices_found = 0;
    for(uint8_t address = 1; address < 127; address++){
            i2c_cmd_handle_t cmd = i2c_cmd_link_create();
            i2c_master_start(cmd);
            i2c_master_write_byte(cmd, (address << 1) | I2C_MASTER_WRITE,
true);
            i2c_master_stop(cmd);
            if(i2c_master_cmd_begin(num_i2c, cmd, 1000 / portTICK_RATE_MS)
== ESP_OK)
            {
                printf("Dispositivo encontrado con direccion 0x%03x\t",
address);
                devices_found++;
                address_found = address;
                break;
            }
            i2c_cmd_link_delete(cmd);
    }
    if(devices_found == 0) printf("No se encuentran dispositivos\t");
    printf("...escaner completado!\n");
}
```

En la función de buscar se realiza la búsqueda del dispositivo mediante las distintas direcciones que se encuentran en el bus de I^2C y utiliza la dirección encontrada.

Una vez seleccionada la dirección que se ha encontrado en el bus, se realiza la lectura desde el dispositivo seleccionado, solicitando previamente la configuración al propio dispositivo para realizar una acción u otra en función del dispositivo del que se trate.

```
void sensor_leer(){
    printf("Se realiza lectura desde direccion %03x\n", address_found);
    i2c_master_sensor_write_conf();
    i2c_master_sensor_read();
    if(data_rd[3]==49 && data_rd[4]==48 && data_rd[5]==48){
        i2c_master_sensor_write_read();
        i2c_master_sensor_read();
    } else if(data_rd[3]==48 && data_rd[4]==48 && data_rd[5]==49){
        printf("Caso configuracion 1\n");
    } else if(data_rd[3]==48 && data_rd[4]==49 && data_rd[5]==48){
        printf("Caso configuracion 2\n");
    }
    mostrarDato();
    sd_escritura(dato_entero, buf_hora);
}
```

Una vez realizada la lectura, se muestra el dato por pantalla y se almacena en la memoria de la μSD.

Para activar el Bluetooth se ha incorporado un pulsador, que se debe accionar únicamente cuando se requiere el acceso por esta tecnología. De este modo se realiza un ahorro de energía por parte del dispositivo.

```
if(pulsador==0){
            printf("Iniciando BT...\n");
            ble_iniciar();
    }
```

Una vez encendido el Bluetooth se puede realizar la conexión a través de un dispositivo móvil.

Dentro de las funciones de BLE, se establece la inicialización de los gestores GAP y GATT para crear el anuncio y poder hacerlo visible. Tras ello, mediante un handle del GATT se activan los diferentes servicios y características propios de la tecnología BLE.

Para realizar la conexión se accede a través de la función de conexión.

```
case ESP_GATTS_CONNECT_EVT: {        esp_ble_conn_update_params_t
conn_params = {0};

        memcpy(conn_params.bda, param->connect.remote_bda,
sizeof(esp_bd_addr_t));

        conn_params.latency = 0;

        conn_params.max_int = 0x20

         conn_params.min_int = 0x10

        conn_params.timeout = 400;            ESP_LOGI(GATTS_TAG,
"ESP_GATTS_CONNECT_EVT, conn_id %d, remote %02x:%02x:%02x:%02x:%02x:%02x:",

                 param->connect.conn_id,

                 param->connect.remote_bda[0], param->connect.remote_bda[1],
param->connect.remote_bda[2],

                 param->connect.remote_bda[3], param->connect.remote_bda[4],
param->connect.remote_bda[5]);

        gl_profile_tab[PROFILE_A_APP_ID].conn_id = param->connect.conn_id;

        esp_ble_gap_update_conn_params(&conn_params);

        break;

    }
```

Tras realizar la conexión llegaría el turno de realizar la lectura. Con esta función será donde se actualicen los valores leídos desde la aplicación móvil.

```
case ESP_GATTS_READ_EVT: {

        ESP_LOGI(GATTS_TAG, "GATT_READ_EVT, conn_id %d, trans_id %d, handle
%d\n", param->read.conn_id, param->read.trans_id, param->read.handle);

        esp_gatt_rsp_t rsp;

        memset(&rsp, 0, sizeof(esp_gatt_rsp_t));

        rsp.attr_value.handle = param->read.handle;

        sensor_leer(); ///ACTUALIZA VALORES

        vTaskDelay(30 / portTICK_RATE_MS);

        rsp.attr_value.len = 1;

        rsp.attr_value.value[0] = dato_entero;

        esp_ble_gatts_send_response(gatts_if, param->read.conn_id, param-
>read.trans_id,

                                    ESP_GATT_OK, &rsp);

        break;

    }
```

Finalmente, para terminar la secuencia de acciones se entraría en la función dormir para realizar el periodo de bajo consumo.

```
void rtc_dormir(int t){
    ++boot_count;
      ESP_LOGI(TAG2, "Boot count: %d", boot_count);
      const int deep_sleep_sec = t;
      ESP_LOGI(TAG2, "Entering deep sleep for %d seconds", deep_sleep_sec);
      esp_deep_sleep(1000000LL * deep_sleep_sec);
}
```

Como se ha comentado antes, está la alternativa de despertar al dispositivo antes del tiempo establecido con una señal externa mediante la siguiente función.

```
void rtc_despertar(int gpio_wakeup){
    esp_sleep_enable_ext0_wakeup(gpio_wakeup, 0);
}
```

En la siguiente imagen se muestra la salida por el monitor de una ejecución del código cargado en el dispositivo. Se realiza una medida simulada para comprobar de forma visual el proceso citado anteriormente.

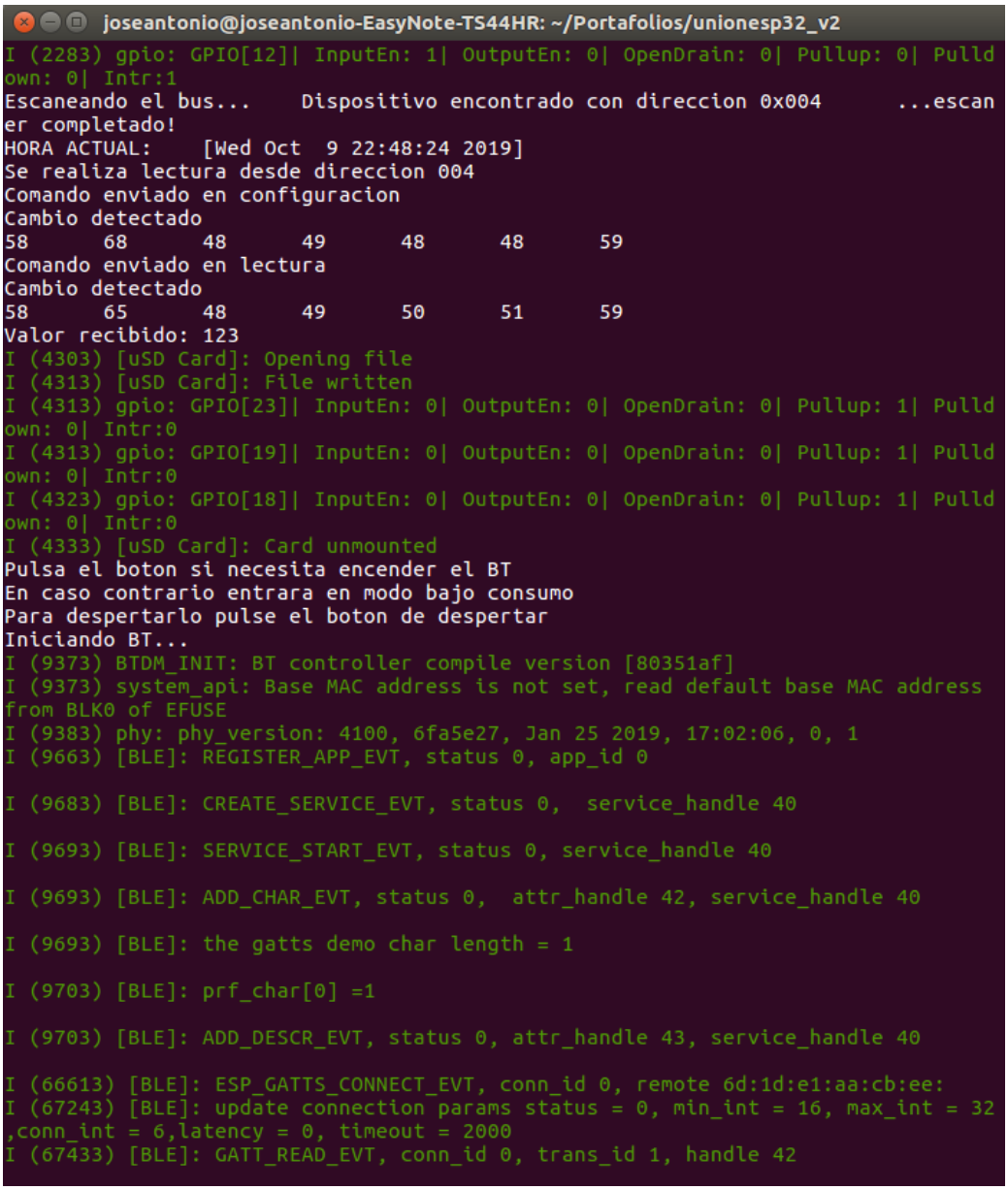

Ilustración 5.1. Salida por pantalla de la ejecución del código.

A falta de la explicación de la comunicación entre los dos dispositivos, a rangos generales estas son unas pinceladas sobre el funcionamiento del software del ESP32.

5.3 Programación de STM32.

En este apartado se va a comentar el funcionamiento del software del STM32, que se ha realizado utilizando el IDE de Arduino.

Aquí se va a comentar de forma breve la secuencia de acciones que se realizan en este dispositivo, ya que la función principal del STM32 es recibir la información del sensor y enviarla al ESP32, donde nos centraremos más adelante.

Siguiendo la estructura de programación del entorno de Arduino, el código de este dispositivo se divide en tres partes.

En esta primera parte se definen los valores iniciales y las variables. También aparece la función *setup( )*, que se ejecuta al principio de la ejecución e inicializa en el dispositivo estos valores.

```
#include <Wire_slave.h>


#define Interrupcion PB4

#define tamanoComando 6

#define tamanoPaquete 7


bool comando1Recibido,comando2Recibido,comando3Recibido = false;

char
cadenaRead[tamanoPaquete],cadenaConf[tamanoPaquete],cadenaTime[tamanoPaquet
e];

static boolean comandoCompleto, inicioComando = false;

char comando[tamanoComando];


void setup(){

  Serial.begin(115200);

  Serial.println("Iniciando");

  Wire.begin(4);

  Wire.onReceive(receiveEvent);

}
```

La segunda parte corresponde a la función *loop( )*, que es el bucle que se ejecuta cíclicamente dentro del código.

```
void loop(){
/* funciones */
if(comando1Recibido){
    comando1Recibido = false;
    Serial.println("Comando recibido");
    digitalWrite(Interrupcion, LOW);
    delay(100);
    digitalWrite(Interrupcion, HIGH);

  }else if(comando2Recibido){
    comando2Recibido = false;
    Serial.println("Comando recibido");
    digitalWrite(Interrupcion, LOW);
    delay(100);
    digitalWrite(Interrupcion, HIGH);

  }else if(comando3Recibido){
    comando3Recibido = false;
    Serial.println("Comando recibido");
    digitalWrite(Interrupcion, LOW);
    delay(100);
    digitalWrite(Interrupcion, HIGH);
  }
    delay(100);
}
```

Y en la última parte se forma de las funciones que se utilizarán en el transcurso del programa. Un ejemplo de una de las funciones que forman esta parte es la siguiente.

```
//Funcion para recibir el comando
void receiveEvent(int howMany){
  int i = 0;
  char rec[tamanoComando];
  while(Wire.available()){
      rec[i] = Wire.read();
      Serial.print(rec[i]);
      if(rec[i]==':') inicioComando=true;
      if(rec[i]==';' && inicioComando){
          comandoCompleto=true;
          strcpy(comando,rec);
          comprobarComando();
          i=0;
      }i++;
  }
}
```

En la función anterior se recibe el comando desde el otro dispositivo y se analiza para posteriormente procesar otra función.

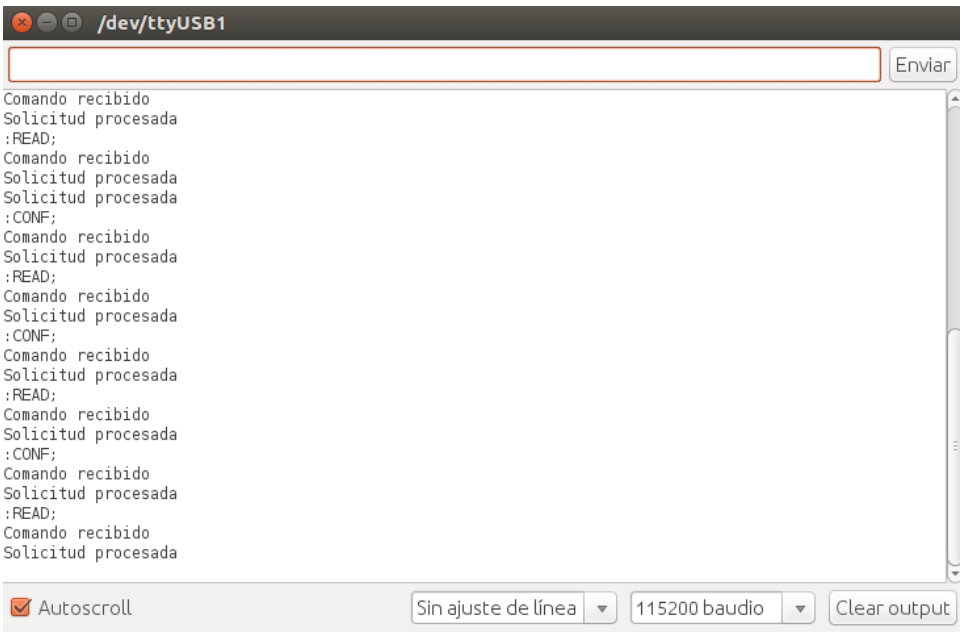

Ilustración 5.2. Salida por pantalla de la ejecución del código.

En la imagen anterior se puede observar la salida por pantalla del código explicado en esta parte.

Conociendo un poco la forma en la que se ha estructurado el código, analizamos como se realiza la comunicación entre los dispositivos.

5.4 Comunicación ESP32 – STM32.

Se va a realizar una explicación de la comunicación entre los dos dispositivos. Esta comunicación se realiza mediante I^2C como se ha comentado anteriormente. Para esta comunicación se ha establecido un protocolo a seguir para cada comunicación, con la creación de unas tramas a enviar que identificarán ambos dispositivos.

El protocolo a utilizar que se ha establecido es el siguiente, como se muestra en la imagen.

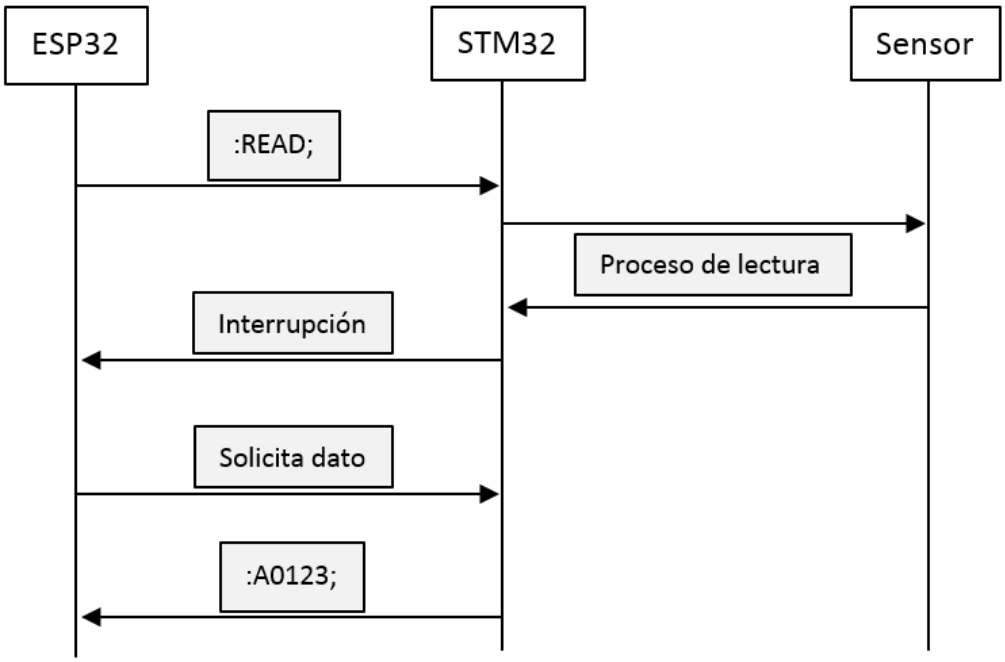

Ilustración 5.3. Protocolo de comunicación entre dispositivos.

Como se observa, la primera acción la realiza el ESP32 mandando una trama al STM32. Todas las tramas se formaran de dos puntos [:] para iniciar la trama y punto y coma [;] para finalizar.

En la primera trama enviada desde el ESP32, se forma de los dos caracteres de inicio y fin de trama comentados y de cuatro caracteres intermedios que identificarán la función a realizar. Se ha pensado tres distintas opciones de identificación.

```
:CONF; //Solicita configuración de la tarjeta
:READ; //Solicita una lectura del sensor
:TIME; //Establece el tiempo de lectura al sensor
```

Con el envío de cada una de estas funciones, el STM32 detecta la función a realizar y ejecuta una operación destinada a cada trama.

En el caso que la trama enviada sea la de lectura, como se muestra en la imagen del ejemplo, se procederá a realizar la lectura del sensor antes de seguir la comunicación entre los dispositivos.

Una vez realizado el proceso de lectura del sensor o bien de procesado de la trama solicitada, el STM32 envía una interrupción al ESP32 advirtiendo que está disponible para seguir con la comunicación.

Tras recibir la interrupción el ESP32 hace una petición al otro dispositivo solicitando el dato que previamente se ha identificado con la primera trama.

Por último, el STM32 envía una nueva trama con el valor resultado. Esta trama tiene como caracteres de inicio y final los mismos que se establecieron para todas las tramas, pero los caracteres intermedios aumentan en número siendo normalmente cinco caracteres, que se identificaran de la siguiente forma.

```
:A0123; //Valor analógico
:D0100; //Valor digital
```

En el segundo carácter más significativo se enviará una 'A' o una 'D' refiriéndose a que el valor enviado es un valor analógico o digital, respectivamente, para que el dispositivo pueda analizarlo correctamente.

De esta forma se establece el protocolo elaborado para la comunicación entre los dispositivos.

5.5 *Aplicación Smartphone.*

Por ultimo en este capítulo se explicará cómo se ha creado y funciona el software de la aplicación del Smartphone. Para su creación se ha utilizado la herramienta de Android, AppInventor, que se gestiona de forma online y permite interactuar con el dispositivo mientras se realizan los procesos de desarrollo.

Como ya se ha comentado anteriormente, el fin de la creación de esta aplicación es la comprobación del correcto funcionamiento del sistema, por lo que no se ha empleado excesivo tiempo en su creación y se ha decidido utilizar esta herramienta que permite el diseño de forma sencilla.

Para la realización del diseño, primero se seleccionan los componentes que quieres que aparezcan en pantalla y su distribución.

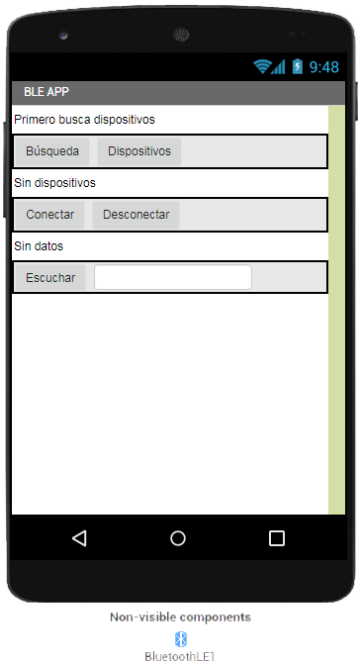

Ilustración 5.4. Distribución de pantalla AppInventor.

Y una vez realizado esto, la forma de funcionar es a través de la conexión entre bloques. El esquema utilizado para realizar la aplicación es el siguiente.

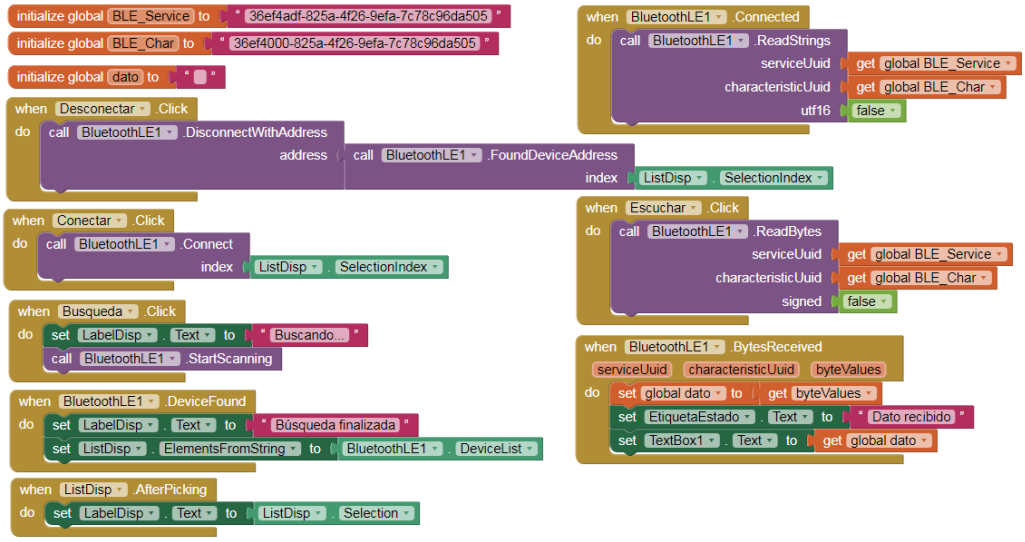

Ilustración 5.5. Esquema de bloques AppInventor.

En él se ha añadido distintos bloques que dan forma y funcionalidad a los componentes añadidos en pantalla, que permitirán conectar el dispositivo móvil con el sistema a través de Bluetooth Low Energy.

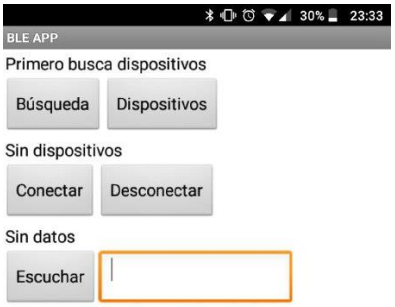

Ilustración 5.6. Apariencia aplicación móvil.

Esta es la apariencia que obtenemos ya en nuestro dispositivo. Por lo que se aprecia es similar a lo mostrado en la herramienta.

Analizando la funcionalidad de la aplicación, encontramos dos botones en la parte superior. En el botón de 'Búsqueda' se realiza una búsqueda de los dispositivos en el alcance de nuestro terminal. Una vez realizada la búsqueda nos cambiará la segunda etiqueta mostrando "Búsqueda finalizada".

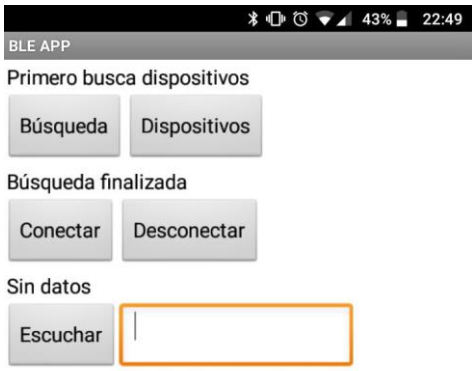

Ilustración 5.7. Apariencia aplicación. Búsqueda finalizada.

Al pinchar en el botón 'Dispositivos' nos aparecerá un listado de los dispositivos encontrados.

Ilustración 5.8. Apariencia aplicación. Lista dispositivos.

Seleccionamos el dispositivo requerido y nos aparecerá el nombre en la etiqueta que se muestra como "Sin dispositivos" cambiándose este valor por el nombre del dispositivo seleccionado.

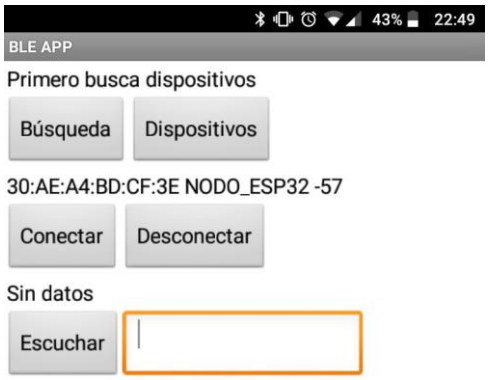

Ilustración 5.9. Apariencia aplicación. Dispositivo seleccionado.

A continuación pulsamos el botón 'Conectar' y la vinculación con el dispositivo quedará realizada. Tras establecer conexión, si se pulsa el botón 'Escuchar' se cambiará el valor de la etiqueta "Sin datos" por "Dato recibido" y nos aparecerá el valor del dato recibido en el cuadro de texto inferior.

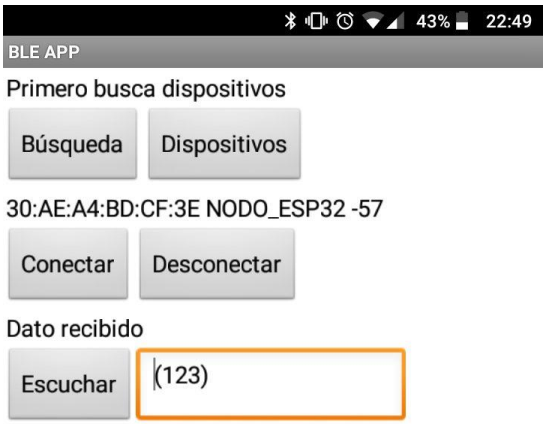

Ilustración 5.10. Apariencia aplicación. Dato recibido.

Capítulo 6

Conclusiones y trabajos futuros

Durante el desarrollo de este proyecto se ha realizado un aprendizaje bastante satisfactorio en el ámbito de la electrónica, más concretamente en el manejo y trabajo con microcontroladores. Es un punto a valorar de forma positiva ya que una de las principales motivaciones de realizar esta elección fue la de encontrarse en un caso real de diseño de un sistema electrónico de estas características.

Pero las inquietudes por seguir en esta línea de trabajo no acaban aquí, ya que su ejecución ha potenciado el interés por lo realizado y la ambición de seguir absorbiendo más conocimientos sobre la materia.

En el transcurso de este trabajo se han observado posibles carencias o mejoras a realizar, así mismo como proyectos que pueden complementarse para mejorar la funcionalidad de ambos.

Uno de los posibles proyectos a realizar como mejora sería la reducción de electrónica en la placa del sistema, a fin de mejorar el consumo y reducir el tamaño. Continuando con la línea de trabajo ya establecida, la realización de periféricos adaptables que permitan trabajar distintos sensores. De forma paralela, el diseño de una aplicación móvil más completa en la que se pueda realizar un uso total de los recursos que ofrece el sistema. Ampliando lo ya realizado, establecer distintos métodos de comunicación con el sistema a fin de facilitar al usuario la recogida de información.

Capítulo 7

Referencias bibliográficas

[1] L. Atzori, A. Iera, y G. Morabito, «The Internet of Things: A survey», *Computer Networks*, vol. 54, n.º 15, pp. 2787-2805, oct. 2010.

[2] A. C. Estrada, *Seguridad por niveles: Seguridad de acuerdo al modelo de capas TCP/IP*. Alejandro Corletti Estrada, 2011.

[3] «Departamento de Tecnología Electrónica», *Departamento de Tecnología Electrónica*. [En línea]. Disponible en: https://www.dte.upct.es/. [Accedido: 06-oct-2019].

[4] «Universidad Politécnica de Cartagena». [En línea]. Disponible en: https://www.upct.es/. [Accedido: 06-oct-2019].

[5] «KiCad EDA». [En línea]. Disponible en: http://kicad-pcb.org/. [Accedido: 08-oct-2019].

[6] «STM32duino», *GitHub*. [En línea]. Disponible en: https://github.com/stm32duino. [Accedido: 23-sep-2019].

[7] «STM32 32-bit Arm Cortex MCUs», *STMicroelectronics*. [En línea]. Disponible en: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html. [Accedido: 10-jul-2019].

[8] «Arduino - Home». [En línea]. Disponible en: https://www.arduino.cc/. [Accedido: 10-jul-2019].

[9] «Espressif Systems - Wi-Fi and Bluetooth chipsets and solutions». [En línea]. Disponible en: https://www.espressif.com/. [Accedido: 06-sep-2019].

[10] «The Internet of Things with ESP32». [En línea]. Disponible en: http://esp32.net/. [Accedido: 06-sep-2019].

[11] «ESP32 Forum - Index page». [En línea]. Disponible en: https://www.esp32.com/. [Accedido: 06-sep-2019].

[12] «ESP-IDF Programming Guide — ESP-IDF Programming Guide v4.1-dev-279-g96b96ae24 documentation». [En línea]. Disponible en: https://docs.espressif.com/projects/esp-idf/en/latest/. [Accedido: 19-sep-2019].

[13] «Getting Started with Bluetooth Low Energy [Book]». [En línea]. Disponible en: https://www.oreilly.com/library/view/getting-started-with/9781491900550/. [Accedido: 23-sep-2019].

[14] «Bluetooth Technology Website | The official website of Bluetooth technology.», *Bluetooth Technology Website*. [En línea]. Disponible en: https://www.bluetooth.com/. [Accedido: 23-sep-2019].
[15] «Introduction to Bluetooth Low Energy», *Adafruit Learning System*. [En línea]. Disponible en: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction. [Accedido: 23-sep-2019].
[16] B. Phillips, K. Marsicano, y C. Stewart, *Android Programming: The Big Nerd Ranch Guide*, Edición: 003. Atlanta, Georgia: Big Nerd Ranch Guides, 2017.
[17] «Bluetooth low energy overview | Desarrolladores de Android», *Android Developers*. [En línea]. Disponible en: https://developer.android.com/guide/topics/connectivity/bluetooth-le. [Accedido: 25-sep-2019].
[18] D. Anurag, S. Roy, y S. Bandyopadhyay, «Agro-sense: Precision agriculture using sensor-based wireless mesh networks», en *2008 First ITU-T Kaleidoscope Academic Conference - Innovations in NGN: Future Network and Services*, 2008, pp. 383-388.
[19] «Avances recientes en la programación de los riegos». [En línea]. Disponible en: https://riunet.upv.es/handle/10251/119227. [Accedido: 25-sep-2019].
[20] E. Fereres Castiel y D. A. Goldhamer, «Avances recientes en la programación de los riegos», *ing.agua*, vol. 7, n.º 1, p. 47, mar. 2000.
[21] «Precision Farming Profitability.pdf». .
[22] «MIT App Inventor | Explore MIT App Inventor». [En línea]. Disponible en: https://appinventor.mit.edu/. [Accedido: 25-sep-2019].
[23] «Nordic Semiconductor - Home - nordicsemi.com». [En línea]. Disponible en: https://www.nordicsemi.com/. [Accedido: 25-sep-2019].
[24] «Android Developers». [En línea]. Disponible en: https://developer.android.com/. [Accedido: 25-sep-2019].
[25] N. Kolban, *Kolban's book on ESP32*. Leanpub, 2016.
[26] «nkolban - Overview», *GitHub*. [En línea]. Disponible en: https://github.com/nkolban. [Accedido: 06-oct-2019].
[27] «Andreas Spiess», *YouTube*. [En línea]. Disponible en: https://www.youtube.com/channel/UCu7_D0o48KbfhpEohoP7YSQ. [Accedido: 06-oct-2019].

Anexo I

Preparación ESP-IDF

I. Introducción.

Gracias a los recursos que proporciona Espressif para ayudar al desarrollo de aplicaciones para sus dispositivos, tenemos una guía de iniciación para la instalación del entorno de desarrollo. En esta ocasión se realizará en el sistema operativo Ubuntu, que es una distribución de Linux basada en la arquitectura Debian, ya que resulta más accesible que el resto de sistemas operativos.

A continuación se hará una breve explicación de los distintos pasos a seguir para la instalación del entorno de ESP-IDF para la programación en ESP32.

II. Instalación.

A. Primer paso. Requisitos previos.

Cada SO requiere una preparación y la instalación de requisitos previos. Para el caso de Ubuntu/Debian se requiere la instalación de los siguientes paquetes:

```
sudo apt-get install git wget libncurses-dev flex bison gperf python python-pip python-setuptools python-serial python-click python-cryptography python-future python-pyparsing python-pyelftools cmake ninja-build ccache
```

B. Segundo paso. Obtener ESP-IDF.

Para la creación de aplicaciones para ESP32, es necesario el uso de bibliotecas del software proporcionado por Espressif desde su repositorio. Para ello, será necesario clonar el repositorio para su instalación. Por defecto se utiliza el directorio **~/esp** para la instalación.

Para realizar este paso, abra la Terminal y ejecute el siguiente comando:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

C. Tercer paso. Configuración de herramientas.

Para su correcto funcionamiento es necesario la instalación de herramientas como el compilador, el depurador, los paquetes de Python, etc.

```
cd ~/esp/esp-idf
./install.sh
```

D. Cuarto paso. Configuración de variables de entorno.

Para que las herramientas puedan ser usadas desde la línea de comandos es necesario establecer las variables de entorno.

```
. $HOME/esp/esp-idf/export.sh
```

E. Quinto paso. Inicio de proyecto.

Para realizar una primera prueba, se puede utilizar el proyecto *hello_world* situado en el directorio *get-started*, que se encuentra dentro de los ejemplos que se ofrecen.

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

F. Sexto paso. Conectar el dispositivo.

Conecte el dispositivo al computador y compruebe el puerto serie que utilizará la placa. En el caso de Linux comenzará con **/dev/tty.**

G. Séptimo paso. Configuración.

Una vez que conozcamos el puerto serie utilizado, accedemos al directorio del proyecto y accedemos a **menuconfig** para preparar la configuración.

```
cd ~/esp/hello_world
idf.py menuconfig
```

Tras introducir los comandos anteriores en la Terminal, nos debe aparecer la siguiente imagen:

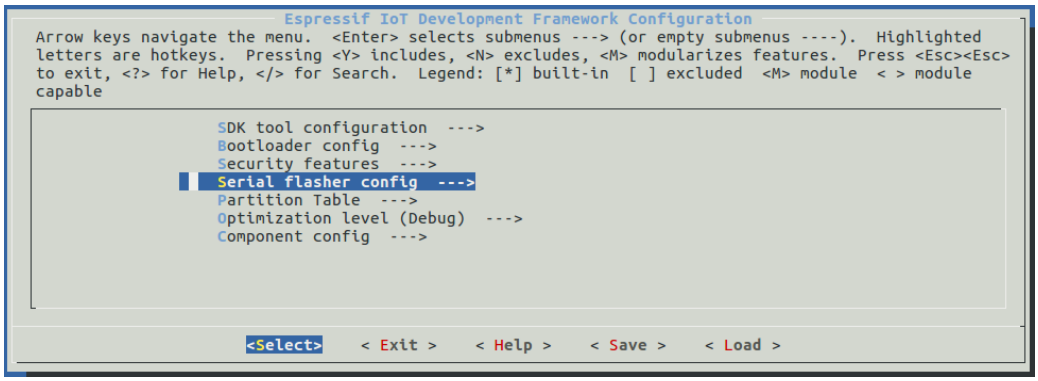

Ilustración I.0.1. Herramienta menuconfig.

En el menú mostrado podremos navegar y acceder a los diferentes submenús para configurar las características de nuestro proyecto.

Accediendo a **Serial flasher config** deberíamos ver que el puerto serie es el correcto. Por defecto en Linux aparecerá el correcto, no haría falta modificar nada.

H. Octavo paso. Construcción del proyecto.

Para la construcción del proyecto deberá ejecutar en la Terminal el siguiente comando:

```
idf.py build
```

Una vez realizado esto, se compilará la aplicación y todos los componentes de ESP-IDF, se generará el gestor de arranque, la tabla de particiones y los binarios de la aplicación.

I. Noveno paso. Flash del dispositivo.

Para realizar la actualización de la construcción en el dispositivo ESP32, ejecute el siguiente comando, pero antes cambie **PORT** por el número de puerto serie de su dispositivo, al igual que se puede cambiar **BAUD** por la velocidad en baudios que se necesite. La velocidad predeterminada es 460800.

```
idf.py -p PORT [-b BAUD] flash
```

De forma similar, si ya se ha configurado los parámetros anteriores desde **menuconfig**, debería poder realizarse la actualización de flash así:

```
idf.py flash
```

Con la opción de flash, se crea y actualiza automáticamente el proyecto. Por lo que no será necesario ejecutar **build**.

J. Décimo paso. Monitor.

Una vez realizado todos los pasos anteriores, nuestro proyecto debe estar ejecutado de forma correcta en el dispositivo. Para poder comprobar que esto es así y poder ver los comandos que nuestro proyecto muestra por pantalla, podemos acceder al monitor.

```
idf.py -p PORT monitor
```

Recuerde cambiar PORT por el puerto serie correspondiente.

Una vez realizado, nuestro dispositivo mostrará por pantalla el siguiente contenido:

```
...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

Desde el monitor puede acceder al menú pulsando **Ctrl+T** o directamente salir del monitor pulsando **Ctrl+AltGr+}**.

Para más información o para cualquier modificación de esta, deberíamos acceder a ESP-IDF Programming Guide/Get Started.

Anexo II

Preparación IDE Arduino para STM32

I. Introducción.

Para poder realizar la programación del STM32 con el entorno de Arduino es necesario realizar unos breves y sencillos pasos previos que se explicarán a continuación.

Antes de empezar con la preparación debemos saber que esta placa permite tres formas de programar el STM32 a través del entorno de Arduino.

- Con conversor USB-TTL.
- Con programador ST-Link.
- A través del puerto USB-micro.

En esta ocasión se ha elegido realizarlo mediante un conversor USB-TTL, por lo que será este el caso explicado.

II. Instalación de Arduino IDE.

Para poder configurar el entorno de Arduino previamente debemos tenerlo instalado en nuestra computadora. De no ser así debemos entrar en la página web oficial de Arduino y en la sección de descargas, seleccionar el paquete de instalación que corresponda con las características de nuestro equipo.

Aquí se añade un enlace para acceder al sitio web.

Una vez realizada la descarga, ejecutar el instalador y seguir los pasos mostrados por el fabricante. Tras ello, tendremos disponible el IDE de Arduino para poder trabajar con él.

III. Configuración de Arduino IDE.

El IDE de Arduino no dispone de un soporte oficial para STM32. Gracias a la comunidad de desarrolladores, en especial a STM32duino[6], ha creado la posibilidad mediante la realización de unos pequeños cambios, de la utilización de este entorno para la programación con la familia STM32.

Para la configuración, abrimos el IDE de Arduino y debemos dirigirnos al *gestor de tarjetas*.

```
Herramientas/Placa:/Gestor de tarjetas…
```

Una vez tengamos abierto el cuadro de gestor de placas, buscamos en la barra de búsqueda el soporte para placas zero.

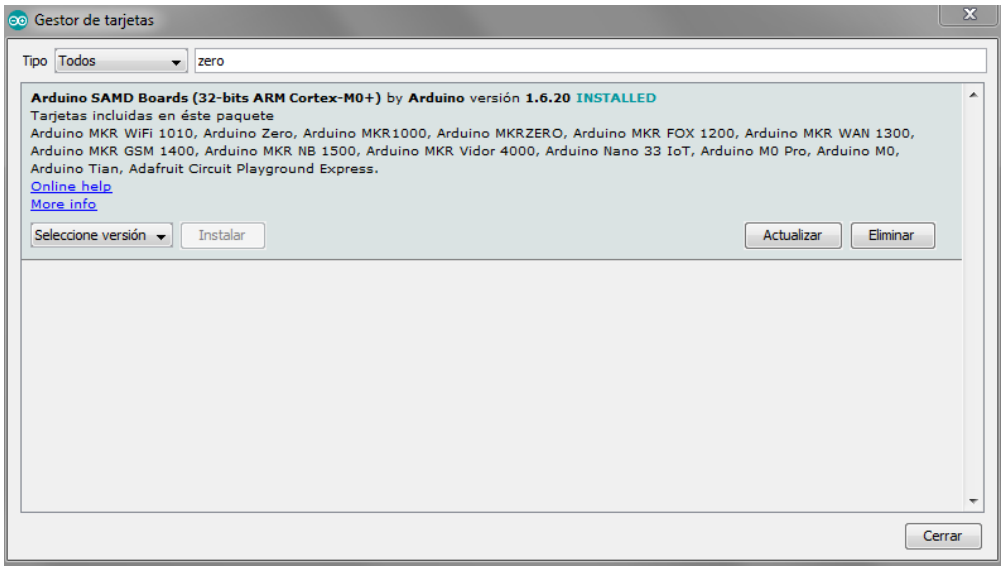

Ilustración II.0.1. Gestor de tarjetas zero.

Instalamos el paquete ya que será necesario para la compilación de los ficheros.

Una vez instalado, debemos acceder al gestor de URLs para añadir la ruta para la gestión de las placas de la familia STM32.

```
Archivo/Preferencias/Gestor de URLs Adicionales de Tarjetas:
```

Tras acceder a la ventana de *Preferencias* pinchamos en el botón que se encuentra a la derecha del campo de texto para abrir la ventana del *Gestor* y copiamos la siguiente ruta en el ella.

```
http://dan.drown.org/stm32duino/package_STM32duino_index.json
```

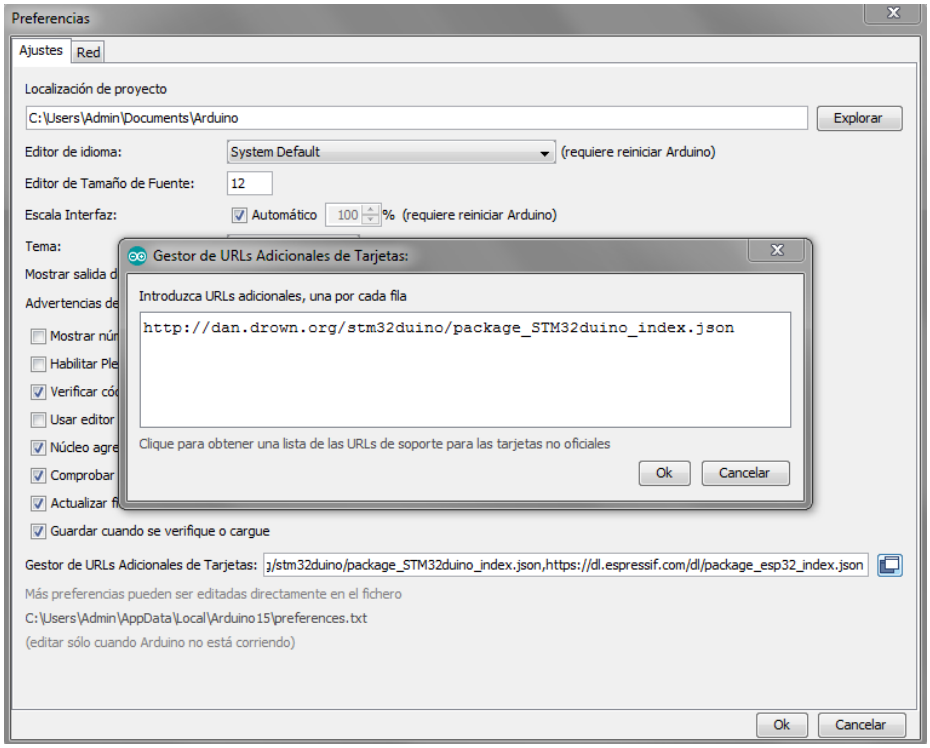

Ilustración II.0.2. Gestor de URLs Adicionales.

Una vez realizado esto, salimos y nos dirigimos de nuevo al *Gestor de tarjetas*, donde debería aparecernos el paquete para la instalación de la familia STM32.

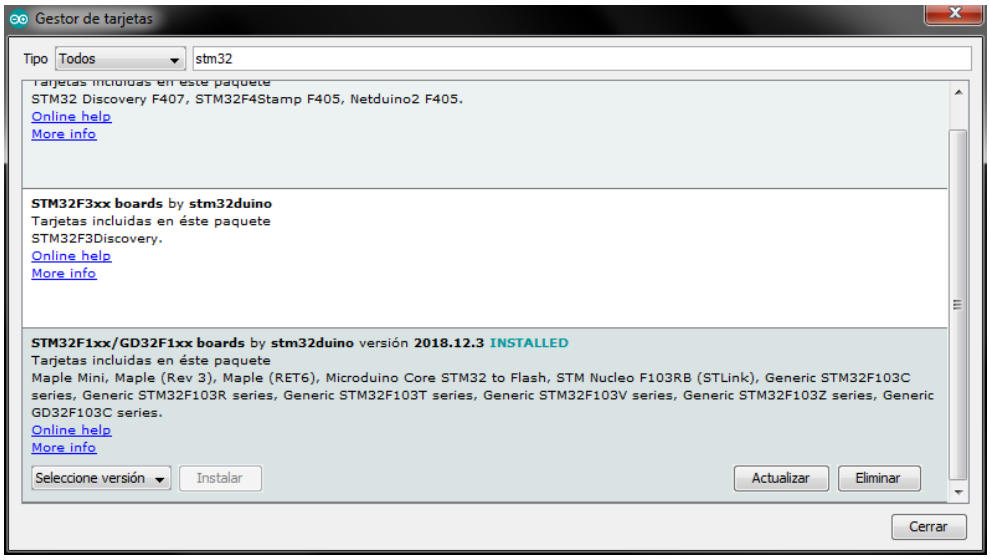

Ilustración II.0.3. Gestión de tarjetas STM32.

En nuestro caso seleccionamos e instalamos el paquete correspondiente a la familia STM32F1xx, ya que es el que se corresponde con nuestro dispositivo.

Una vez realizado todo, ya deberíamos poder programar el STM32 en el IDE de Arduino.

IV. Conexión de STM32.

Para realizar la programación del STM32 se va a necesitar un conversor USB-TTL que nos permitirá la programación del microcontrolador a través de UART. Conectamos con USB-TTL por serial al dispositivo STM32.

Antes de realizar la conexión hay que tener en cuenta que el micro del STM32 opera a 3.3V, por lo que habrá que configurar el conversor USB-TTL a dicho voltaje para no dañar ningún dispositivo.

STM32	USB-TTL
Vcc	3V3
PA9 (Tx1)	Rx
PA10 (Rx1)	Tx
GND	GND

Tabla II.1. Conexión STM32 con USB-TTL.

En la tabla anterior podemos ver las conexiones con los pines de los dos dispositivos. Para que quede más visual, en la siguiente imagen se muestra cómo será la distribución de las conexiones.

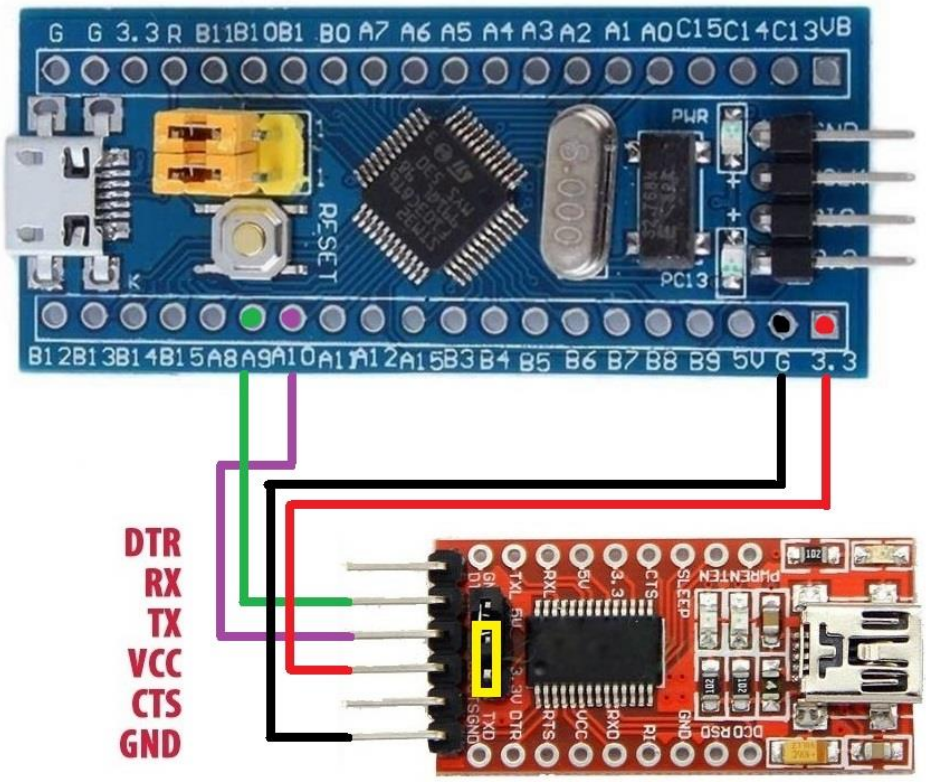

Ilustración II.0.4. Conexión STM32 con USB-TTL.

V. Programación STM32 con Arduino IDE.

Para poder realizar la programación del STM32, debemos conocer la posición del pin **BOOT0**. Este pin situación en la parte superior de la placa nos permite elegir entre el modo de programación o de uso normal de la tarjeta.

Para cambiar el valor de este pin la placa dispone de un jumper que se puede mover según la función a elegir.

- En la posición LOW o 0, la placa funciona de forma normal con el último programa en su flash.
- En la posición HIGH o 1, la placa está en modo programación, por lo que espera una actualización de la flash a través de UART.

En la siguiente imagen se muestra una marca sobre el BOOT0 en la posición en la que debe estar el jumper para su programación.

Ilustración II.0.5. Posición BOOT0 para programación.

Teniendo en cuenta lo anterior, ya está la tarjeta preparada para su programación. A continuación solo falta la selección de las características correspondientes en el IDE de Arduino.

Accedemos a la selección de la placa y seleccionamos la tarjeta correspondiente.

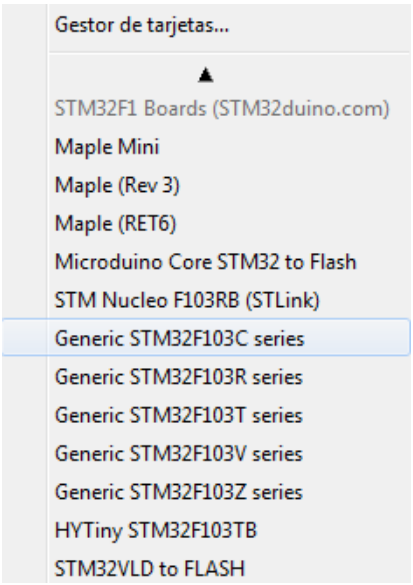

Ilustración II.0.6. Tarjetas STM32.

En este caso seleccionamos "*Generic STM32F103C series*" y configuramos los siguientes parámetros. Entre ellos seleccionamos el método de subida como "*Serial*", ya que vamos a realizarla a través de la UART, como se ve en la siguiente ilustración.

Ilustración II.0.7. Parámetros STM32.

De esta forma tendríamos ya todo configurado y preparado para poder utilizar nuestro dispositivo en el entorno de Arduino.