Universidad de Valladolid

TRABAJO FIN DE GRADO

Monitorización remota (vía smartphone) de la concentración de oxígeno en una planta de fabricación de vino

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECIFICAS DE TELECOMUNICACIÓN:
MENCIÓN EN SISTEMAS ELECTRÓNICOS

Autor:
Marcos Martín Gutiérrez

Tutor:
Jesús M. Hernández Mangas

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Valladolid, Junio de 2019

"But the Hebrew word, the word timshel 'Thou mayest' that gives a choice. It might be the most important word in the world. That says the way is open. That throws it right back on a man. For if 'Thou mayest' it is also true that 'Thou mayest not.'"

— John Steinbeck, East of Eden

Agradecimientos

Después de un intenso período de tiempo, hoy es el día: escribo este apartado de agradecimientos para finalizar mi trabajo de fin de grado. Este período ha supuesto para mí un aprendizaje profundo, tanto a nivel técnico como personal. Desarrollar este trabajo ha tenido un gran impacto en mi y por ello, me gustaría agradecer a todas aquellas personas que me han ayudado y apoyado durante este proceso:

A mi padre, por ser padre y amigo, por creer y confiar en mí. Por ser ejemplo de lucha, de trabajo y de esfuerzo. Por enseñarme a no rendirme nunca. Porque gracias a él decidí ser ingeniero.

A mi madre, por ser mi apoyo y mi alegría, por su generosidad y su amor, por ser una ejemplo de superación.

A mi hermana, por ser el mayor ejemplo de constancia y sacrificio. Por enseñarme que todo es posible con esfuerzo. Ojalá cumplas tus sueños.

A mi tutor Jesús M. Hernández Mangas, por todas sus enseñanzas, su clases y su paciencia. Por su ayuda en el desarrollo de este Trabajo Fin de Grado. Porque gracias a él se me han abierto puertas tanto personales como profesionales que nunca hubiera imaginado.

A mi amigo Luis Miguel, por su ayuda desinteresada y por ser un espejo en el que mirarse. Por enseñarme el valor de la amistad.

A mi primo Diego, por enseñarme a escuchar y reflexionar. Por ser primo, compañero de piso y sobre todo amigo.

A mi amiga Fátima, por su cariño, su ayuda y su generosidad. Por esas largas noches de café y apuntes en el Aulario.

A mis abuelos, por su amor incondicional, por enseñarme lo que es el amor, el respeto y la humildad.

A mis compañeros de clase, por todo el tiempo y vivencias compartidas juntos. Por luchar codo con codo, por pasar buenos y malos momentos juntos. Por su compañerismo y amistad.

A mis amigos Los Quintos, por su apoyo incondicional y por ser los mejores amigos que una persona puede tener.

Resumen

Cada vez es más habitual encontrarse con dispositivos y objetos cotidianos que son capaces de conectarse a Internet, permitiendo al usuario su control de forma remota a través de una aplicación móvil. Esta es la definición del concepto de *Internet of Things (IoT)*, una tecnología en pleno auge y evolución. Se prevé que en 2020 habrá 20.400 millones de dispositivos *IoT* en funcionamiento. El objetivo de este avance tecnológico es hacer más cómodas nuestras vidas así como proporcionar una mayor seguridad en diversos ámbitos.

Con este Trabajo Fin de Grado, se pretende mostrar el conjunto de etapas para el desarrollo de un proyecto de ingeniería de sistemas electrónicos en el campo de *Internet of Things (IoT)*. Desde el análisis del problema, hasta el montaje final del prototipo, pasando por todo su desarrollo hardware así como software, herramientas utilizadas y problemas planteados.

El objetivo de este Trabajo Fin de Grado consiste en el desarrollo de un sistema electrónico de reducido tamaño que funciona de forma autónoma como un registrador de datos. Este dispositivo se interconecta vía *USB* con el sensor de oxígeno *FireStingGO2* y registra continuamente y en tiempo real los datos que toma. Estos datos son enviados a una aplicación móvil *Android* que se conectará con el dispositivo mediante *WiFi* para su posterior manejo y análisis de datos.

Abstract

Nowadays, it is more frequent to find devices and everyday objects that can connect to the Internet, allowing the user to control them remotely through a mobile application. This is the definition of the concept of the *Internet of Things (IoT)*, technology in full development and evolution. It is expected that in 2020 there will be 20,400 million *IoT* devices in operation. The goal of this technological advance is to make our lives more comfortable and provide greater security in various areas.

With this final thesis, I want to show the set of stages for the development of an electronic systems engineering project in the field of *Internet of Things (IoT)*. From the analysis of the problem, to the final assembly of the prototype, through its hardware and software development, tools used and problems raised.

The thesis's goal is the development of a small electronic system that works autonomously as a datalogger. *USB* connect this device to the *FireStingGO2* oxygen sensor and it records the data collected by the sensor in real time. This data is sent to an *Android* mobile application that will connect to the device through *WiFi* for its management and data analysis.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción y conocimientos previos

Capítulo 1

Introducción, motivación y objetivos

Internet de las Cosas (IoT) es una de las principales tendencias que están transformando la industria, un verdadero impulsor en la transformación digital dentro de las empresas. El valor estratégico de los datos generados por los dispositivos suponen un activo único para las compañías, que extraen su máximo valor para mantener y ampliar su ventaja competitiva.

1.1. Introducción

Como podemos ver en nuestro día a día, cada vez son más las aplicaciones que utilizan la tecnología *IoT*, desde relojes inteligentes, sensores de aparcamiento o la domótica. Pero donde podemos ver la verdadera evolución de esta tecnología es en el paso del uso doméstico al industrial. Es en el ámbito empresarial donde realmente se está llevando a cabo un mayor número de iniciativas gracias a su combinación con otras tecnologías como la analítica avanzada, la movilidad, la nube y los sensores.

Durante el desarrollo de los procesos industriales, los sensores son los equivalentes a los sentidos humanos, que son los encargados de identificar estímulos del entorno exterior y transmitirlos al cerebro. De esta manera, en un sistema automatizado, esto se lleva a cabo mediante sensores, que serían los encargados de captar la información del entorno y transmitirla a la computadora para que se tome la decisión correcta en cada caso.

Cuando hablamos de automatización se está haciendo referencia al proceso productivo en el que no es necesaria la participación humana, ya que esta se limita a dar las instrucciones iniciales o, en caso de que sea necesario, modificarlas con otras nuevas. Por ello, hablar de automatización es hacerlo de un concepto en el que se ejecuta una operación controlado de forma autónoma que estará constituida por cuatro acciones concretas llevadas a cabo por los sensores industriales y la computadora: observar, analizar, tomar una decisión y ejecutarla. Estas cuatro acciones concretas sirven para mejorar la calidad y la eficiencia de los productos en la industria.

En el sector vinícola, concretamente, los sensores más utilizados son los sensores de oxígeno. Estos permiten determinar el nivel de oxígeno en el vino así como su temperatura, humedad, niveles de presión, etc. Posteriormente se analizan todos estos datos recogidos

lo que permite mejorar la eficiencia de los procesos de producción así como mejorar la calidad del producto.

El proyecto desarrollado consiste un sistema electrónico al que se conecta un sensor de oxígeno concreto *(FireStingGO2)* y registra continuamente y en tiempo real los datos que toma el sensor y mediante tecnología*WiFi* envía esos datos a una aplicación móvil (que forma parte del proyecto) para su posterior análisis.

1.2. Motivación y elección del Trabajo Fin de Grado

La principal motivación para realizar este Trabajo Fin de Grado es el desarrollo de un proyecto de Ingeniería Electrónica. Partiendo del análisis del problema hasta la creación real del dispositivo en completo funcionamiento, pasando por todas sus etapas de desarrollo. Conocer y aprender de primera mano el proceso de producción de un dispositivo electrónico en ingeniería. Enfocar todo lo aprendido durante estos años de estudio en las asignaturas del Grado, especialmente las orientadas a la electrónica y más concretamente en asignaturas tales como Ingeniería de Sistemas Electrónicos e Interconexión de Sistemas Electrónicos.

Otra motivación a la hora de elegir este proyecto fue su enfoque al mundo de Internet de las Cosas. Una tecnología en pleno auge y que, con el desarrollo de este proyecto me daría la oportunidad de aprender más acerca de este sector tecnológico desde un punto de vista más técnico.

Este proyecto está orientado al sector vinícola. Esto era una gran motivación para mí, ya que me atraía la idea de realizar un Trabajo Fin de Grado relacionado con un sector con mucha presencia en Castilla y León. Más concretamente en Palencia, ciudad en la que nací, donde es un sector en pleno crecimiento.

1.3. Descripción del problema

Se dispone de un aparato que mide la concentración de oxígeno y otras magnitudes físicas en ciertos lugares dentro de una planta de fabricación de vino. Este dispositivo: *Pyroscience FireStingGO2* dispone de una salida *USB device* con un protocolo propietario.

Se requiere diseñar un sistema electrónico autónomo (con batería) que permita conectar el aparato de medición *(USB host)* y que pueda controlar de forma inalámbrica el sistema citado a través de un *Smartphone*. El sistema sensor permite hacer y almacenar ciertas medidas que deberán ser extraídas desde el *Smartphone* a través del sistema electrónico diseñado y almacenadas en una base de datos (tipo *sqlite*) para su posterior estudio desde un PC.

1.4. Objetivos del Trabajo Fin de Grado

La realización de este Trabajo Fin de Grado tiene como objeto desarrollar de un sistema electrónico siguiendo los pasos que se han de seguir para la creación final de un prototipo que satisfaga los requerimientos propuestos. En este documento se detalla en profundidad el diseño y la construcción de este prototipo durante todas sus fases.

Los principales objetivos del trabajo:

1. Diseñar y fabricar el sistema electrónico: Analizar el problema para el diseño en *Proteus 8* el sistema electrónico que cumpla con los requerimientos. Realizar la captura esquemática y diseñar la placa de circuito impreso (PCB). También, ordenar la fabricación de la PCB para su posterior montaje y soldadura de los componentes.

2. Programar el firmware: A través del entorno de desarrollo de *Arduino (Arduino IDE)* desarrollar el firmware para el microcontrolador *ESP-12F*.

3. Aplicación para el teléfono móvil: Crear una aplicación para teléfono móvil del sistema operativo *Android* capaz de mostrar los datos recibidos del sistema electrónico.

Capítulo 2

Planificación

En este capítulo se realizará un estudio de las fases del proyecto así cómo de su estimación temporal mediante un Diagrama de Grannt.

2.1. Fases del proyecto

Para el correcto desarrollo de un proyecto de ingeniería es necesario, una vez planteado el problema, dividir este en diferentes fases o etapas. Esto permite estructurar el problema planteado en tareas más simples y elaborar un mapa temporal de trabajo.

Las fases del proyecto deben estar correctamente distribuidas y organizadas en el tiempo, lo que nos hará trabajar de manera más eficiente aprovechando los recursos y el tiempo dedicado. Concretamente para este TFG, se ha dividido el problema en siete fases desde su inicio hasta el final. Cada fase contiene una serie de subtareas específicas. La estimación temporal de cada fase será la suma de las estimaciones temporales de sus subtareas. De esta manera se ha conseguido un orden y unos rangos temporales determinados para el desarrollo del prototipo.

A continuación, se detallan en la Tabla 2.1 todas las etapas que requiere un proyecto de estas características:

Fase 1: Especificación y planificación	1.1 Estimación temporal inicial
	1.2 Estudio y selección de componentes
Fase 2: Captura esquemática	2.1 Creación de componentes nuevos
	2.2 Posicionado de componentes (en plano eléctrico)
	2.3 Rutado de conexiones
	2.4 Análisis de consumo energético
	2.5 Generación del B.O.M. (bill of materials)
	2.6 Esquemático
Fase 3: Diseño placa de circuito impreso	3.1 Creación de nuevos componentes
	3.2 Definición de Borde de placa y zona de conectores
	3.3 Posicionado de componentes (en software de diseño PCB)
	3.4 Rutado de conexiones
	3.5 Definición de planos de Masa y otros planos
	3.6 Generación de ficheros
Fase 4: Fabricación PCB	4.1 Envío al fabricante PCB
	4.2 Recepción del pedido
	4.3 Montaje de componentes
Fase 5: Desarrollo firmware y software	5.1 Especificaciones del software necesario
	5.2 Control de los elementos Hardware: Drivers
	5.3 Búsqueda de los entornos de desarrollo (IDEs) necesarios
	5.4 Programación del microprocesador del módulo WiFi
	5.5 Programación de la aplicación de control en Android
	5.6 Programación del funcionamiento en Servidores externos
	5.7 Documentación y control de cambios de firmware
Fase 6: Verificación del hardware	6.1 Verificación inicial de la placa
	6.2 Verificación del firmware on-board
	6.3 Resolución de problemas
	6.4 Análisis de prestaciones
	6.5 Montaje y pruebas finales
Fase 7: Documentación	7.1 Realización de la documentación
	7.2 Memoria Final y presentación

Cuadro 2.1: Fases del proyecto

2.2. Estimación temporal

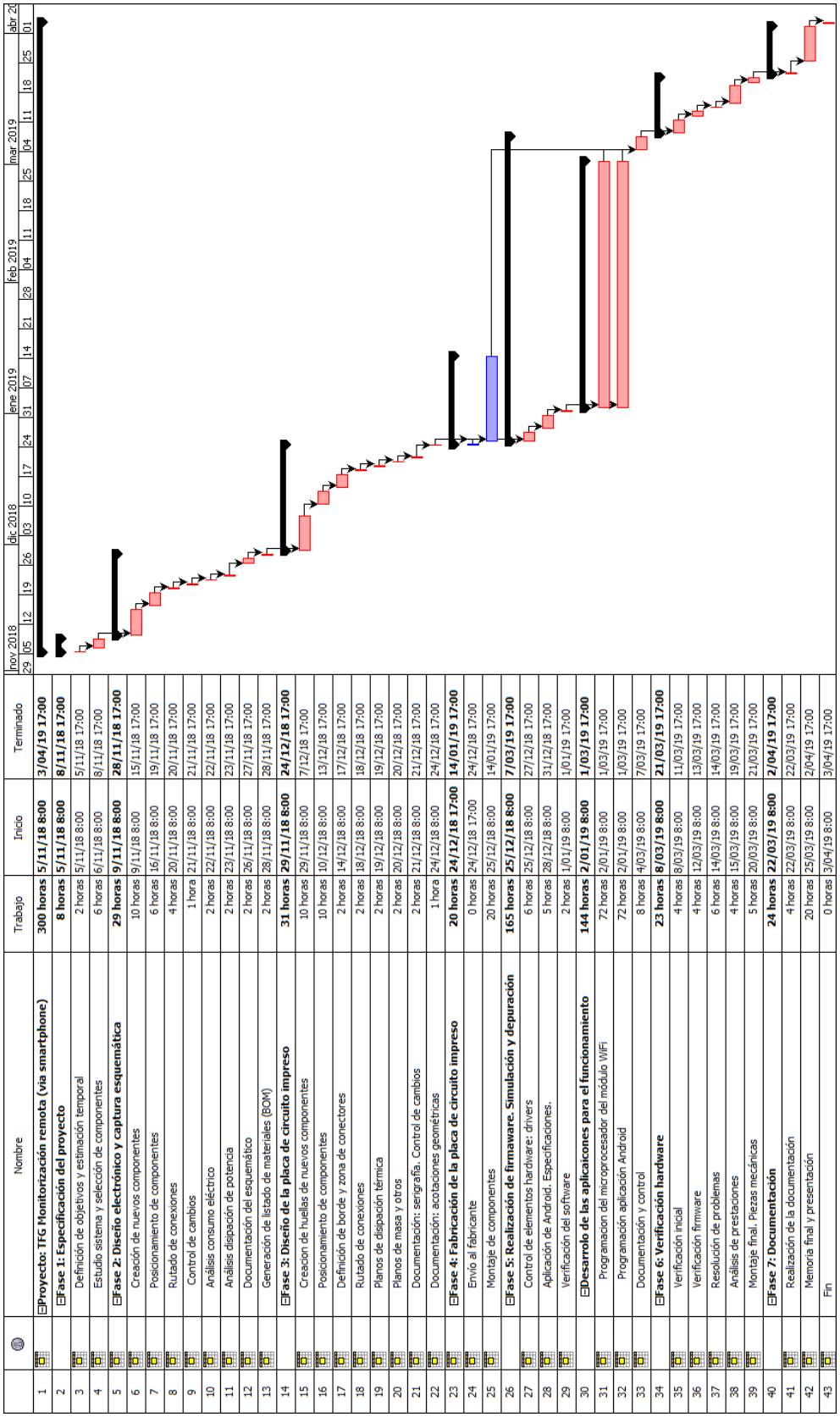
Es importante realizar una estimación temporal detallada para el desarrollo de un prototipo. Esta estimación dependerá de la dificultad de resolución de la tarea y nos permitirá conocer el orden concreto de resolución de tareas en relación con el tiempo de desarrollo de las mismas.

Para ello se ha utilizado un diagrama de Gantt, que es una herramienta gráfica que define el tiempo dedicado previsto para cada una de las fases y subtareas anteriormente detalladas (Figura 2.1) a lo largo de un tiempo total determinado. Proporciona una vista general sencilla y clara de las tareas programadas, lo que permite una mejor gestión del tiempo y mayor flexibilidad. Los proyectos suelen sufrir modificaciones. Al tener una vista general de los cambios inesperados dentro de los objetivos o los plazos de tiempo de un proyecto, puedes ajustar las tareas y recursos como corresponde.

Un diagrama de Gantt te muestra:

- La fecha de inicio y finalización de un proyecto.
- Las tareas que hay en del proyecto.
- La fecha programada de inicio y finalización de las tareas.
- Una estimación temporal de cuánto llevará cada tarea.
- Cómo se superponen las tareas y/o si hay una relación entre ellas.

En la Figura 2.1 se aprecia el Diagrama de Gantt para este Trabajo Fin de Grado:



	Nombre	Trabajo	Inicio	Terminado
1	**Proyecto: TFG Monitorización remota (vía smartphone)**	**300 horas**	**5/11/18 8:00**	**3/04/19 17:00**
2	**Fase 1: Especificación del proyecto**	**8 horas**	**5/11/18 8:00**	**8/11/18 17:00**
3	Definición de objetivos y estimación temporal	2 horas	5/11/18 8:00	5/11/18 17:00
4	Estudio sistema y selección de componentes	6 horas	6/11/18 8:00	8/11/18 17:00
5	**Fase 2: Diseño electrónico y captura esquemática**	**29 horas**	**9/11/18 8:00**	**28/11/18 17:00**
6	Creación de nuevos componentes	10 horas	9/11/18 8:00	15/11/18 17:00
7	Posicionamiento de componentes	6 horas	16/11/18 8:00	19/11/18 17:00
8	Rutado de conexiones	4 horas	20/11/18 8:00	20/11/18 17:00
9	Control de cambios	1 hora	21/11/18 8:00	21/11/18 17:00
10	Análisis consumo eléctrico	2 horas	22/11/18 8:00	22/11/18 17:00
11	Análisis disipación de potencia	2 horas	23/11/18 8:00	23/11/18 17:00
12	Documentación del esquemático	2 horas	26/11/18 8:00	27/11/18 17:00
13	Generación de listado de materiales (BOM)	2 horas	28/11/18 8:00	28/11/18 17:00
14	**Fase 3: Diseño de la placa de circuito impreso**	**31 horas**	**29/11/18 8:00**	**24/12/18 17:00**
15	Creacion de huellas de nuevos componentes	10 horas	29/11/18 8:00	7/12/18 17:00
16	Posicionamiento de componentes	10 horas	10/12/18 8:00	13/12/18 17:00
17	Definición de borde y zona de conectores	2 horas	14/12/18 8:00	17/12/18 17:00
18	Rutado de conexiones	2 horas	18/12/18 8:00	18/12/18 17:00
19	Planos de disipación térmica	2 horas	19/12/18 8:00	19/12/18 17:00
20	Planos de masa y otros	2 horas	20/12/18 8:00	20/12/18 17:00
21	Documentación: serigrafía. Control de cambios	2 horas	21/12/18 8:00	21/12/18 17:00
22	Documentación: acotaciones geométricas	1 hora	24/12/18 8:00	24/12/18 17:00
23	**Fase 4: Fabricación de la placa de circuito impreso**	**20 horas**	**24/12/18 17:00**	**14/01/19 17:00**
24	Envío al fabricante	0 horas	24/12/18 17:00	24/12/18 17:00
25	Montaje de componentes	20 horas	25/12/18 8:00	14/01/19 17:00
26	**Fase 5: Realización de firmaware. Simulación y depuración**	**165 horas**	**25/12/18 8:00**	**7/03/19 17:00**
27	Control de elementos hardware: drivers	6 horas	25/12/18 8:00	27/12/18 17:00
28	Aplicación de Android. Especificaciones.	5 horas	28/12/18 8:00	31/12/18 17:00
29	Verificación del software	2 horas	1/01/19 8:00	1/01/19 17:00
30	**Desarrolo de las aplicaciones para el funcionamiento**	**144 horas**	**2/01/19 8:00**	**1/03/19 17:00**
31	Programacion del microprocesador del módulo WiFi	72 horas	2/01/19 8:00	1/03/19 17:00
32	Programación aplicación Android	72 horas	2/01/19 8:00	1/03/19 17:00
33	Documentación y control	8 horas	4/03/19 8:00	7/03/19 17:00
34	**Fase 6: Verificación hardware**	**23 horas**	**8/03/19 8:00**	**21/03/19 17:00**
35	Verificación inicial	4 horas	8/03/19 8:00	11/03/19 17:00
36	Verificación firmware	4 horas	12/03/19 8:00	13/03/19 17:00
37	Resolución de problemas	6 horas	14/03/19 8:00	14/03/19 17:00
38	Análisis de prestaciones	4 horas	15/03/19 8:00	19/03/19 17:00
39	Montaje final. Piezas mecánicas	5 horas	20/03/19 8:00	21/03/19 17:00
40	**Fase 7: Documentación**	**24 horas**	**22/03/19 8:00**	**2/04/19 17:00**
41	Realización de la documentación	4 horas	22/03/19 8:00	22/03/19 17:00
42	Memoria final y presentación	20 horas	25/03/19 8:00	2/04/19 17:00
43	Fin	0 horas	3/04/19 8:00	3/04/19 17:00

Figura 2.1: Diagrama de Gantt inicial

Parte II

Trabajo desarrollado

Capítulo 3

Estudio de las especificaciones y selección de componentes

A la hora de desarrollar un prototipo es muy importante el estudio previo de las especificaciones del problema planteado. Es fundamental conocer perfectamente lo que se requiere para poder escoger los componentes que mejor se adapten a la elaboración del prototipo. Este paso es la base sobre la que se sustentan las siguientes fases de la elaboración del prototipo.

El sistema electrónico debe tener conectividad inalámbrica con la aplicación móvil, así como una interfaz USB Host para conectarse al sensor y ser un sistema autónomo con batería. Por ello, el estudio de las especificaciones de este proyecto se divide en cinco:

- Microcontrolador con conectividad inalámbrica: Se requiere un microcontrolador capaz de recibir y transmitir de manera inalámbrica. También es necesario que se pueda programar para controlar un USB Host. Esto se consigue seleccionando un microcontrolador capaz de soportar buses de comunicación tales como SPI, UART e I2C. Al ser un sistema autónomo se requiere un consumo bajo. Este módulo hará de puente de comunicación entre el sensor y la aplicación móvil.

  Se ha elegido un chip comercial de bajo coste, el ESP-12F (fabricado por Espressif), que implementa el protocolo TCP/IP. Incluye un microcontrolador (Tensilica Xtensa LX106) para manejar dicho protocolo y el software necesario para la conexión 802.11 b/g/n, que permite la conexión WiFi Direct. Además dispone de entradas/salidas digitales de propósito general (GPIO), así como una entrada analógica (ADC de 10bit) y una memoria Flash de 4MB. Por otro lado, soporta SPI, I2C y UART, lo que nos permite flexibilidad a la hora de buscar un periférico USB Host. Tiene una tensión de alimentación de 3,3V y un bajo precio de aproximadamente 3 euros.

- Periférico que conecte vía USB Host con el sensor: Es necesario encontrar un pe-

Figura 3.1: Microcontrolador ESP-12F

riférico capaz de abrir una sesión, controlar la conexión y acceder como maestro a otros dispositivos. Se ha seleccionado el controlador USB Host MAX3421E. Este se conecta usando el bus SPI con el microcontrolador ESP-12F y permite interactuar y controlar dispositivos USB.

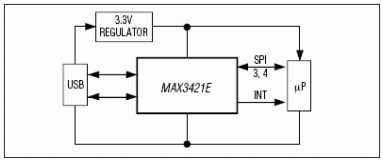


Figura 3.2: Periférico MAX3421E

- Periférico con interfaz USB para programar el microcontrolador: Se necesita una interfaz para conectar el ESP-12F al PC para poder cargar programas en el microcontrolador. El adaptador USB serial FT232RL permite la comunicación a través del puerto USB de tu PC con la UART del microcontrolador. Además, con la electrónica necesaria (MCP73831), se cargará la batería.

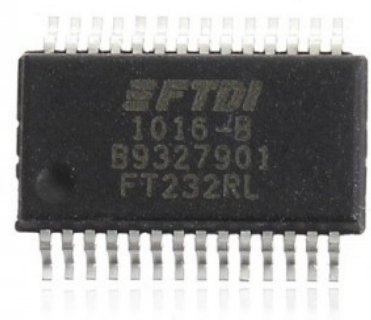


Figura 3.3: Periférico FT232RL

- Protocolo Sensor FireStingGO2 V3.33: Para poder solicitar datos al sensor y establecer una comunicación correcta es necesario conocer su protocolo de comunicación interno. Por ello se ha hecho un estudio de este protocolo mediante de su datasheet,

este protocolo es información confidencial. Debido al alto coste del sensor, ha resultado imposible disponer de él, por ello se ha realizado un simulador. El desarrollo del simulador del sensor se ha hecho a través del módulo educativo LPC-2103 facilitado por el tutor para emular el comportamiento del protocolo del sensor.

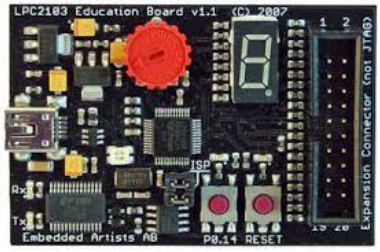


Figura 3.4: Education Board LPC2103

Figura 3.5: Sensor FireStingGO2

- Aplicación Android que muestre los datos recibidos: La aplicación Android es necesaria para controlar la solicitud de datos al sensor a través del sistema electrónico. Esta se conectará vía WiFi Direct con el módulo ESP-12F y mediante su interfaz se seleccionará que datos queremos visualizar. Una vez seleccionado mandará una petición al sistema electrónico que, a su vez, solicitará datos al sensor. Este responderá al ESP-12F con los datos, que serán enviados a la aplicación móvil. Por último, los datos recibidos serán mostrados en una gráfica interactiva que se actualiza en intervalos regulares de tiempo. Una funcionalidad añadida es que, a través de la aplicación móvil, los datos pueden ser enviados a una base de datos en la nube vía WiFi.

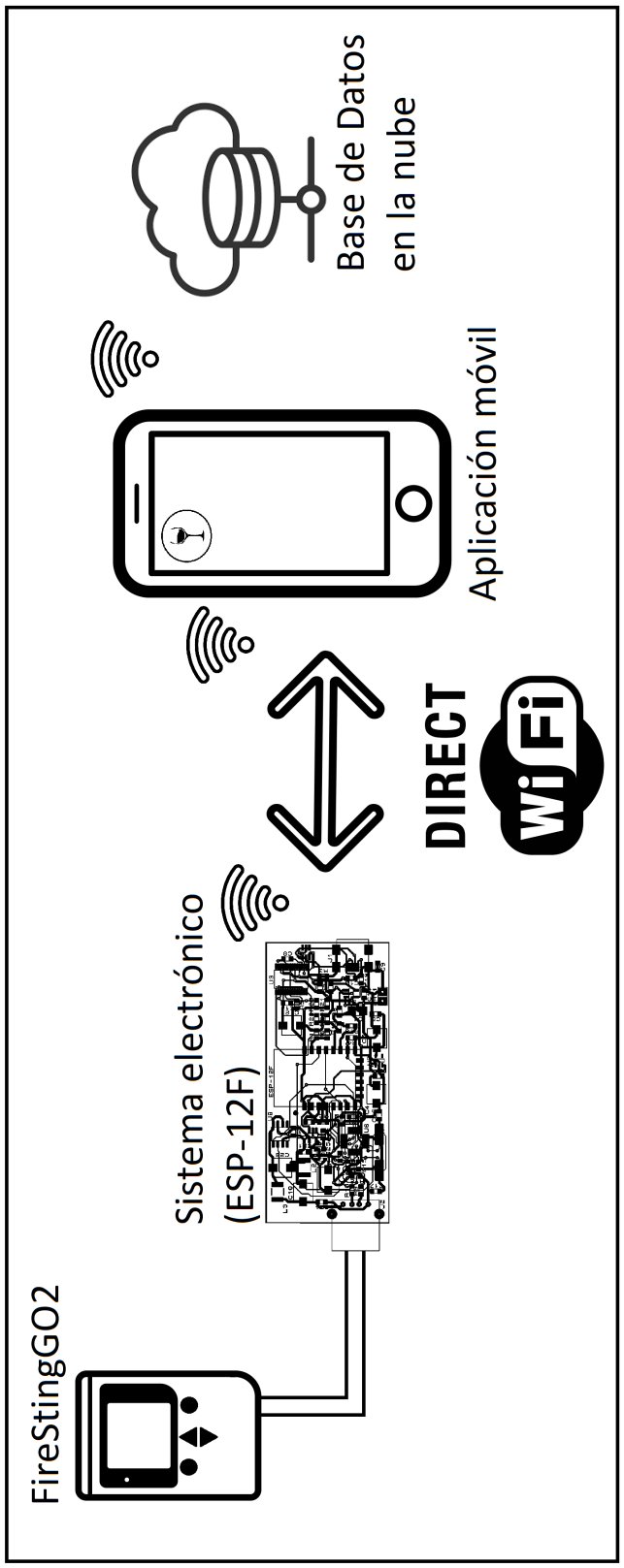


Figura 3.6: Infraestructura de comunicación

Capítulo 4

Desarrollo hardware

4.1. Captura esquemática

Para el diseño del hardware es necesario realizar un diagrama de bloques que muestre la comunicación entre los módulos internos, tanto a nivel de voltajes como a nivel de líneas de datos. Esto permite tener una visión global del hardware del proyecto. También nos permite realizar un análisis previo más exhaustivo de los componentes necesarios.

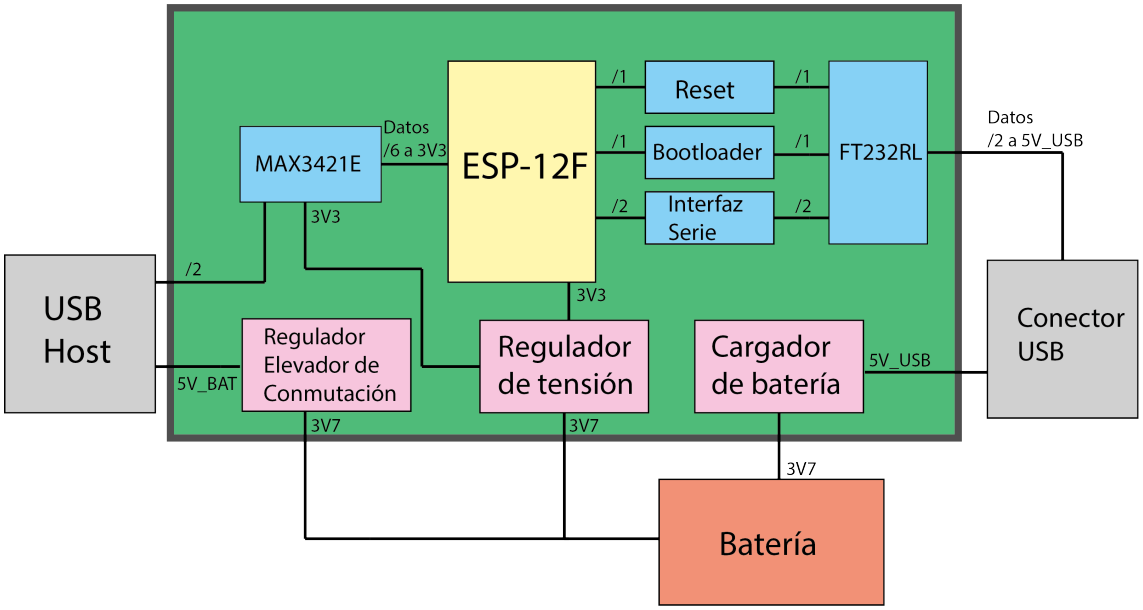


Figura 4.1: Diagrama de comunicación intramodular

Este diseño hardware parte de la alimentación, es decir, tendremos que seleccionar un voltaje de alimentación adecuado para nuestro circuito. Examinando las hojas de los componentes, comprobamos que el ESP-12F está alimentado a 3V3, así como el MAX3421E, mientras que el FT232RL tiene una alimentación de 5V. Por otro lado, necesitamos una tensión de 5V para alimentar el USB Host, necesarios para poder conectar el sensor a nuestro módulo. Además, debido a que uno de los requerimientos es que el sistema sea

autónomo se ha seleccionado una batería Li-Ion de 3V7. Por ello es necesario usar en el circuito una serie de elevadores y reguladores de tensión para tener el voltaje correcto en cada momento.

La alimentación del sistema variará de dos formas:

- Cuando el módulo opera de forma autónoma alimentado por la batería: La tensión de entrada del circuito son los 3V7 que provee la batería de Li-Ion. Para alimentar el ESP-12F y el MAX3421E son necesarios 3V3, por ello será necesario un regulador de tensión de 3V7 a 3V3. Por otro lado, el USB Host tiene que tener una tensión constante de 5V, lo que hace necesario un elevador de tensión de 3V7 a 5V. Es por esto que podemos decir que los cinco voltios con los que se alimenta el circuito en este estado provienen de la batería (5V_BAT).

- Cuando el módulo sea programado o se esté cargando la batería: El sistema tendrá cinco voltios de entrada a través del conector USB de alimentación y que alimentarán el FT232RL. La comunicación de datos entre este y el microcontrolador ESP-12F se hará, por lo tanto, mediante divisores de tensión (resistencias). Como la batería es de 3V7, también será necesario un regulador de tensión que pase de 5V a 3V7. Estos cinco voltios no son los mismos que utilizaremos de forma autónoma, ya que este solo estará conectado cuando necesitemos programar el ESP-12F y/o alimentar la batería. Se puede decir que estos 5V son del conector USB (5V_USB).

En este punto ya podemos empezar a realizar la captura esquemática del prototipo. Partiremos de la base del diagrama de comunicación modular (Figura 4.1) y con los componentes ya seleccionados se procede a la creación del esquemático a través de la herramienta Proteus 8. Este es un software informático para la ejecución de proyectos de construcción de equipos electrónicos en todas sus etapas: diseño del esquema electrónico, programación del software, construcción de la placa de circuito impreso, simulación de todo el conjunto, depuración de errores, documentación y construcción.

En este esquemático aparecerán todos los componentes necesarios para el correcto diseño hardware y su interconexionado. Se utilizarán los componentes pertenecientes a la librería de Proteus y en el caso de que algún componente no esté en las propias librerías se creará de cero, como se mostrará posteriormente. Cada componente deberá tener una serie de parámetros que los definan, como por ejemplo su precio o su código de stock. Todos los parámetros permitirán generar el BOM (Bill Of Materials, Figura 8.1), que es una lista de todos los componentes utilizados en la captura esquemática. A partir de este documento se puede hacer un pedido de los componentes necesarios de una forma más eficiente.

Se ha dividido la captura esquemática en cuatro hojas que agrupan los componentes y las interconexiones. Posteriormente se explicará cada uno de los principales circuitos que forman cada hoja. A continuación se mostrarán todas las hojas de la captura esquemática:

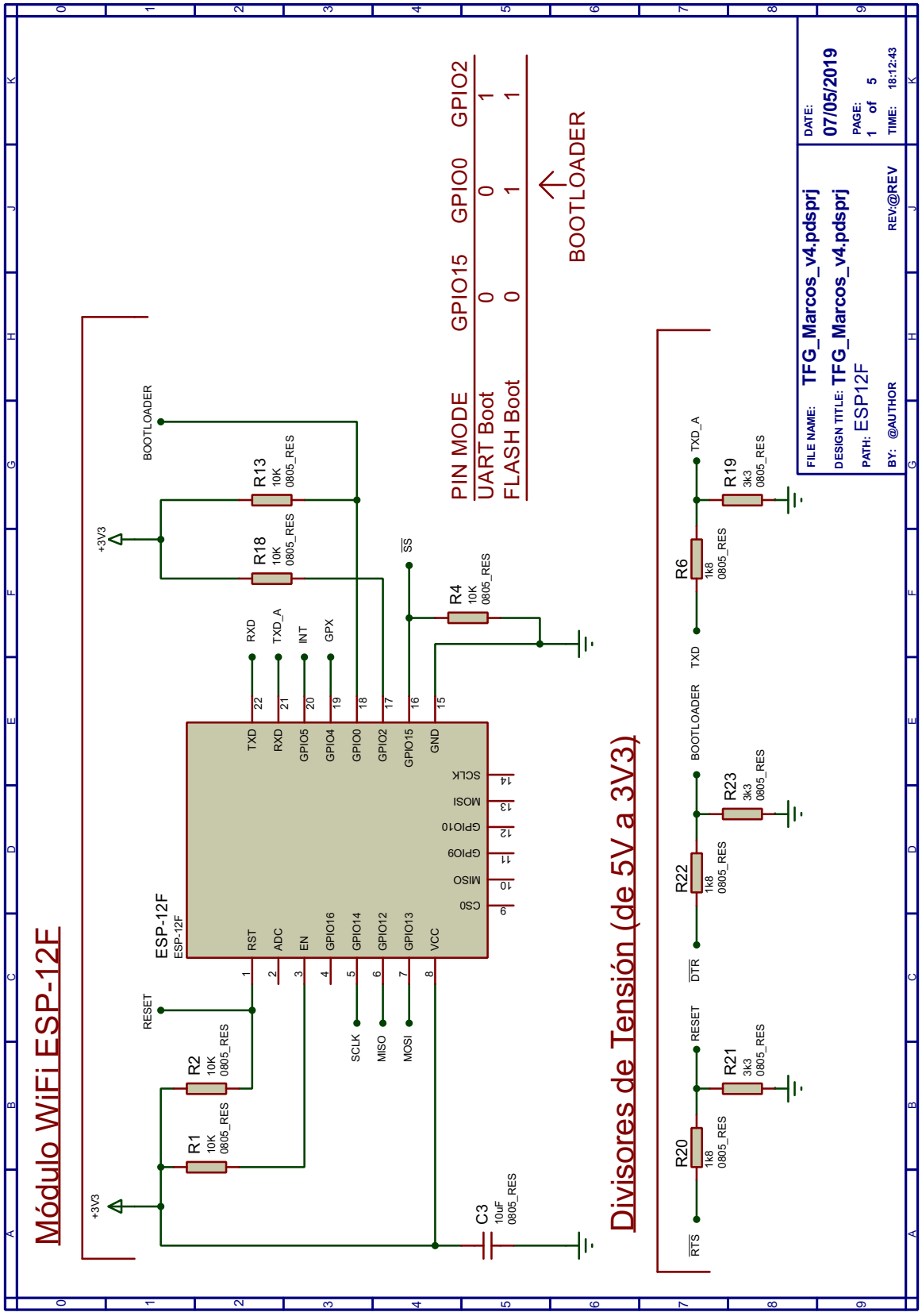


Figura 4.2: Esquemático circuito ESP-12F

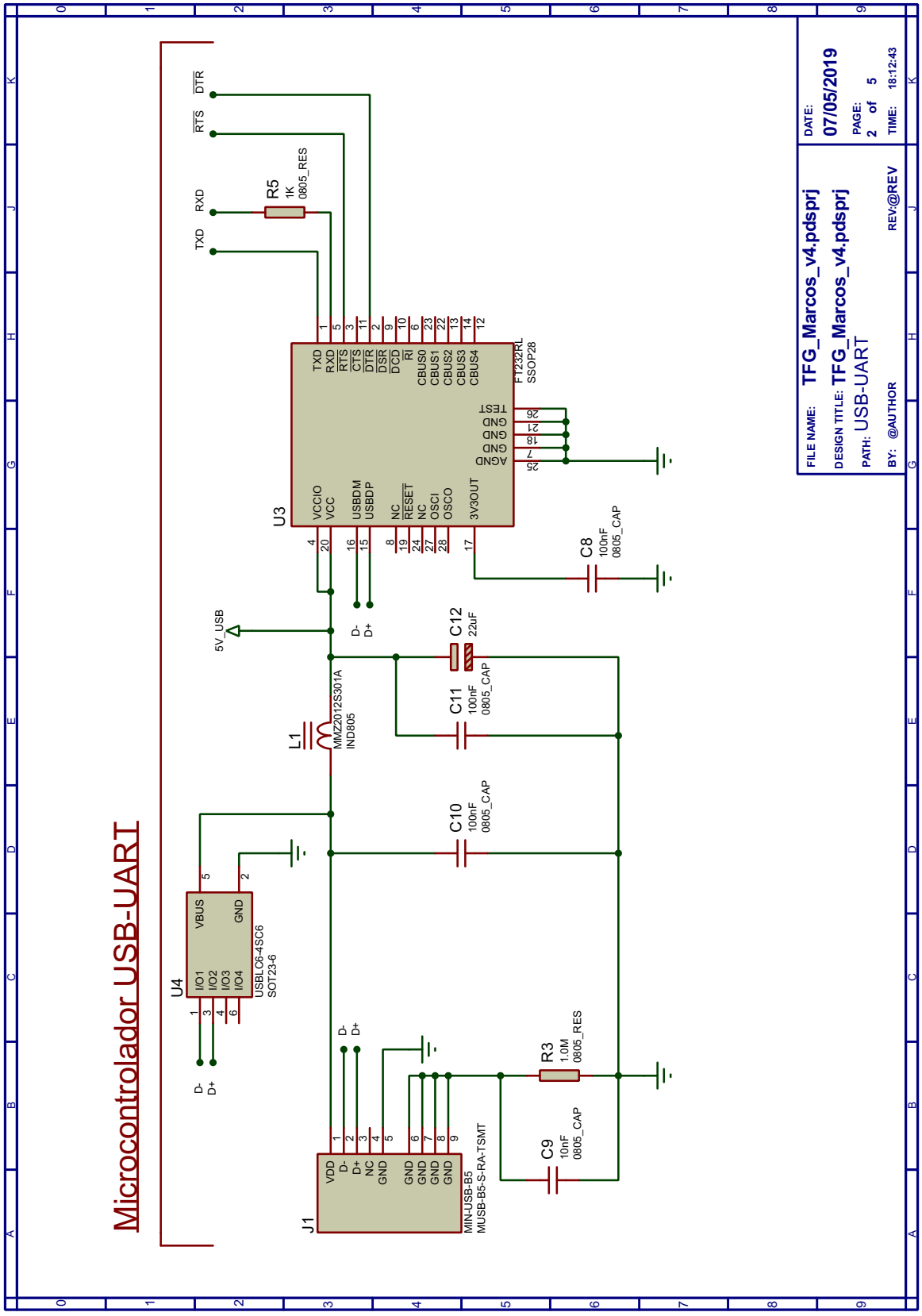


Figura 4.3: Esquemático circuito FT232RL

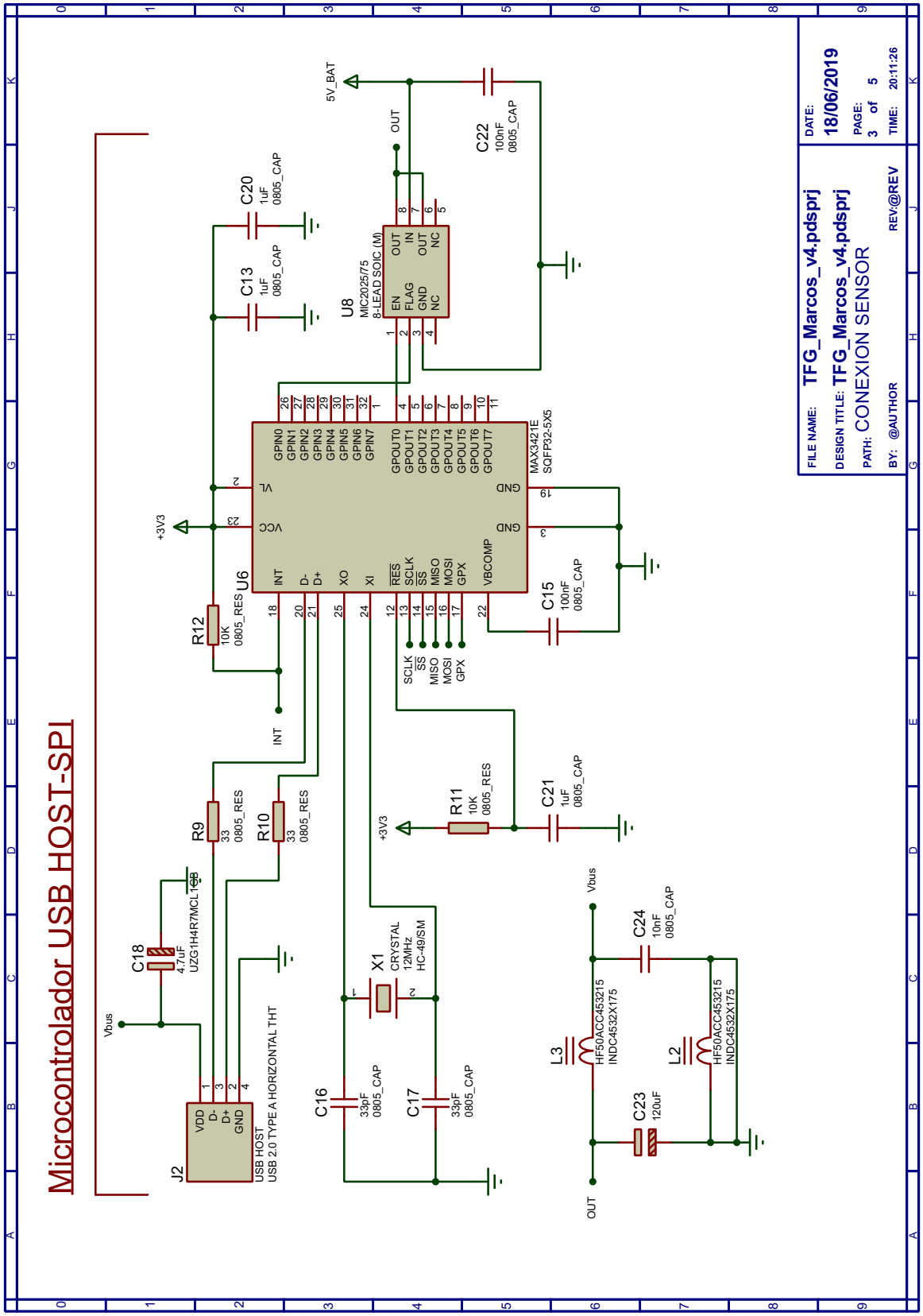


Figura 4.4: Esquemático circuito MAX3421E

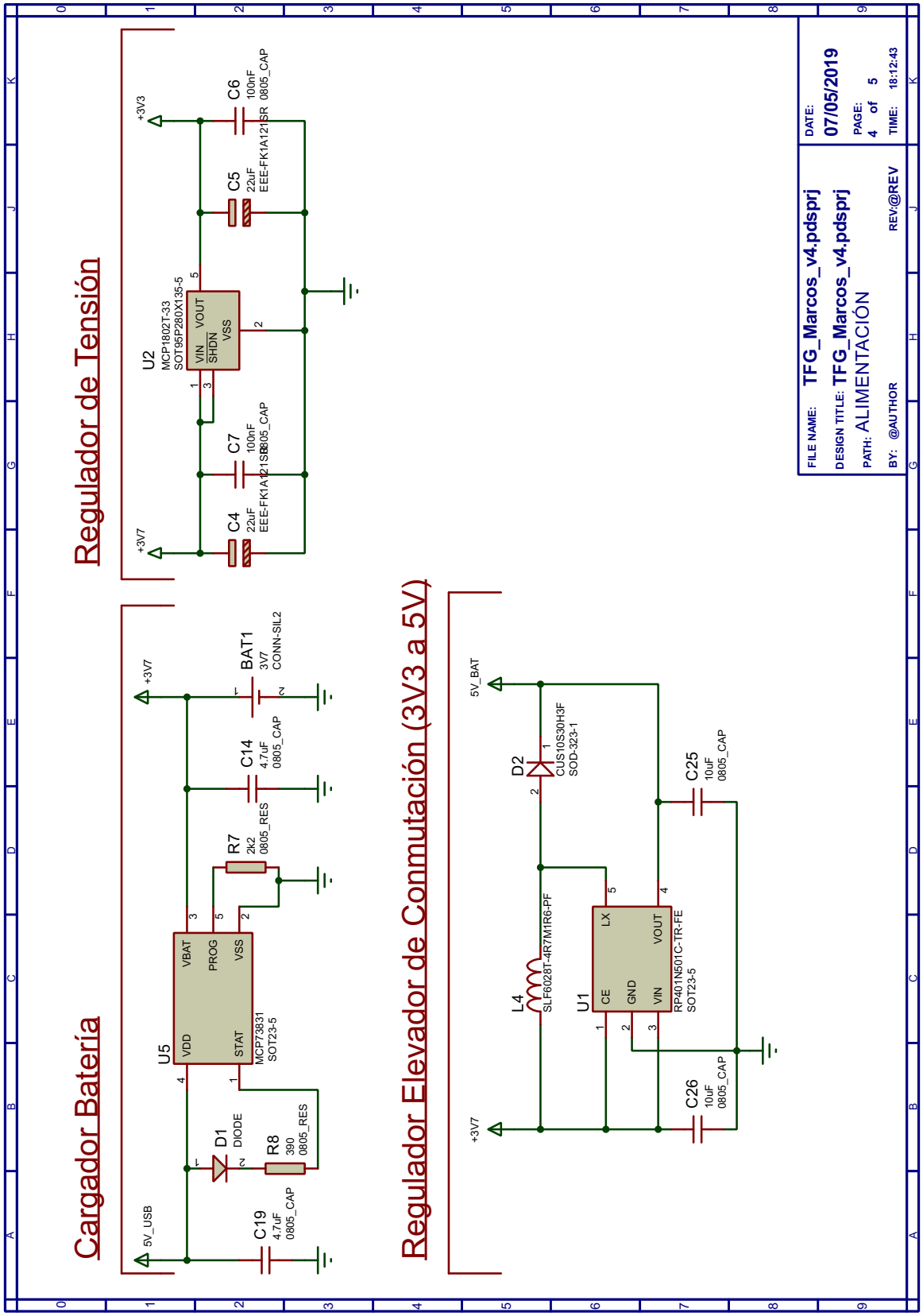


Figura 4.5: Esquemático circuito alimentación

4.1.1. Circuito del microcontrolador ESP-12F

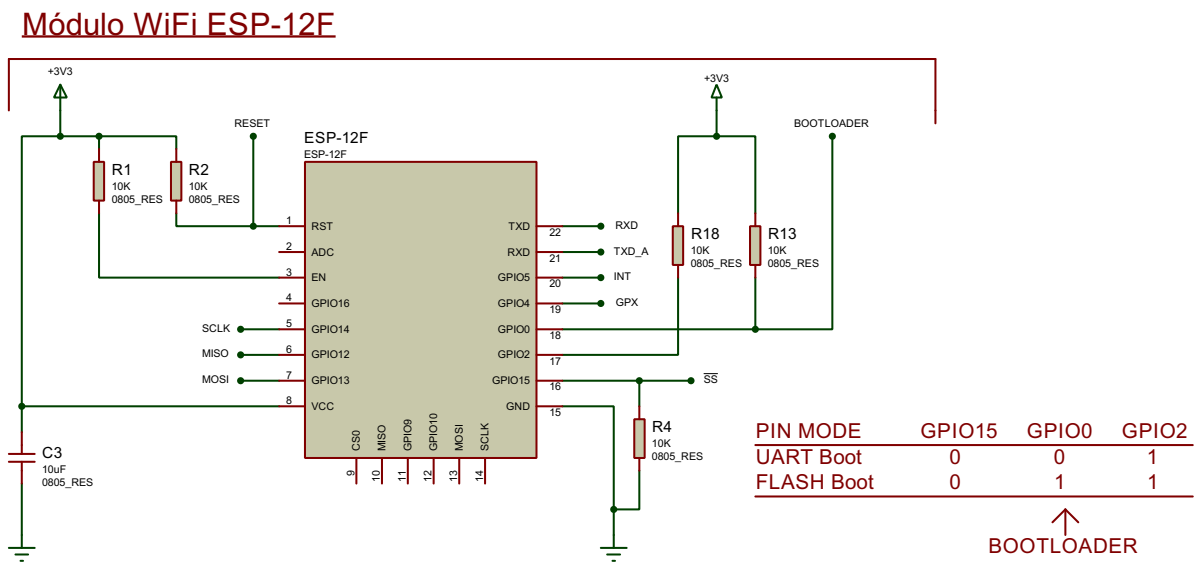


Figura 4.6: Circuito del microcontrolador ESP-12F

El ESP-12F no posee por si solo, la electrónica necesaria para su funcionamiento: insuficientes condensadores de desacoplo, sin regulador de voltaje, sin circuito de reset, sin adaptador USB-Serie, etc. Para el uso de este microcontrolador es importante:

- Proveer energía suficiente al módulo. Para un uso estable del ESP8266 se requiere una fuente de alimentación con 3.3V y >= 250mA. No se recomienda utilizar la alimentación del adaptador USB-Serie. Estos adaptadores normalmente no suministran corriente suficiente para que el ESP8266 funcione de forma segura en todas las situaciones. Es preferible un suministro externo o regulador junto con condensadores de filtrado. En nuestro caso, se utilizará la alimentación de la batería.
- Conectar resistencias de arranque a GPIO0 (HIGH), GPIO2(HIGH), GPIO15 (LOW).
- Poner el ESP8266 en modo bootloader antes de subir código.

Con estas consideraciones se ha desarrollado el circuito necesario para su correcto funcionamiento como se aprecia en la Figura 4.6. En el esquemático se ha incluido la electrónica necesaria:

- Resistencia R2: Se añade una resistencia de Pull-Up al reset del microcontrolador (PIN1-RST) que es activo en bajo. De esta manera se suministra una tensión constante de 3V3, en cambio cuando RESET se activa en bajo se deriva toda la corriente a masa y la caída de tensión es 0V. RESET es un reset (valga la redundancia) controlado mediante software a través del FT232RL (conversor USB-serie) sustituyendo a un interruptor convencional.

- Resistencia R1: Según las especificaciones EN siempre en HIGH mediante resistencias de 10k a 3,3V (pull-up).
- Resistencia R4: Resistencia Pull-Down. Necesaria ya que GPIO15 debe estar activo en baja durante el arranque.
- Resistencia R18: Resistencia Pull-Up. Necesaria ya que GPIO2 debe estar activo en alta durante el arranque.
- Resistencia R13: Resistencia Pull-Up. Necesaria ya que GPIO0 debe estar activo en alta durante el arranque.
- Condensador C3: Condensador de desacoplo en el pin de la alimentación del microcontrolador.

Los pines TXD0 y RXD0 se utilizarán para la comunicación serie (UART). En la tabla se aprecia como para tener el ESP-12F en modo BOOTLOADER es necesario tener el GPIO0 a 1. Los pines GPIO12, GPIO13, GPIO14, GPIO15, GPIO4 y GPIO5 se utilizarán para la comunicación SPI (Serial Peripheral Interface).

4.1.2. Circuito microcontrolador para interfaz USB-Serie

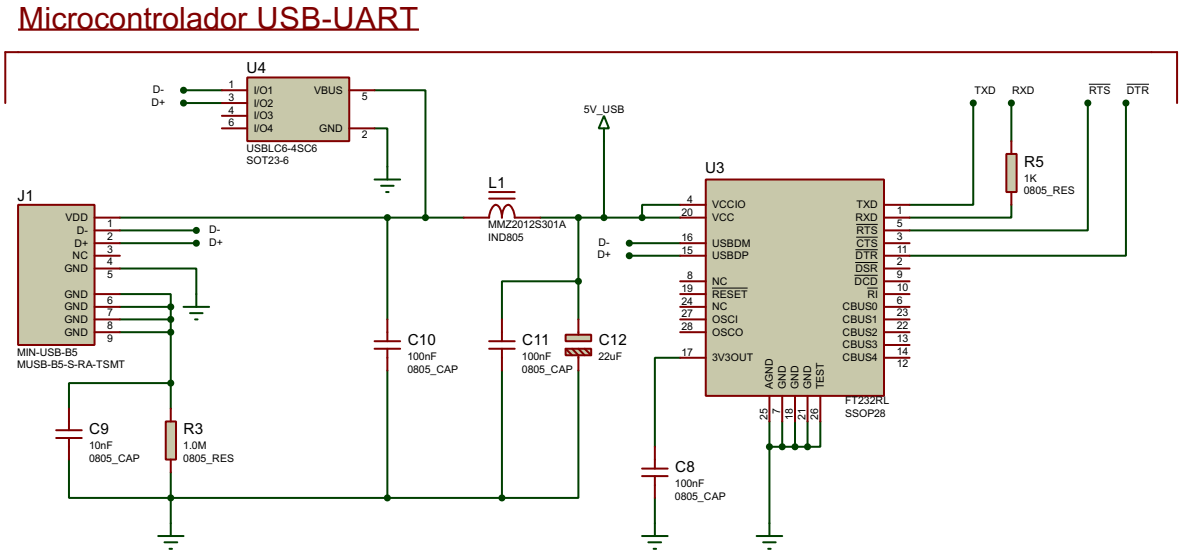


Figura 4.7: Circuito microcontrolador para interfaz USB-Serie

Como se ha visto en la subsección anterior, el ESP-12F posee una interfaz puerto serie (UART) un ya que es la base de casi cualquiera de las comunicaciones de los microcontroladores. El adaptador interfaz USB serial FT232RL permite comunicarte a través del

puerto USB de tu PC con la UART del microcontrolador. La interfaz UART se emplea para la descarga de programas (mediante bootloader). El circuito utilizado en este proyecto está disponible en el propio datasheet del FT232RL. También proveerá los 5V necesarios al cargador de batería.

Las principales características del FT232RL son:

- El protocolo es manejado entero en el chip, no se requiere programación de firmware específica para USB.
- Salidas para LED's indicadores de las señales de transmisión y recepción.
- La interfaz UART soporta 7 u 8 bits de datos, 1 ó 2 bits de parada, y paridad par/impar/marca/espacio/sin paridad.
- USB 2.0 Full speed.
- Salidas LEDs indicador de señal de transmisión y recepción.
- Buffer de recepción de 128 Bytes y de transmisión de 256 bytes.
- Voltaje de Operación: 5 volts.
- EEPROM de 1024 bits para almacenar descriptores del dispositivo USB y configuraciones I/O CBUS
- Pines I/O CBUS configurables

Se ha añadido el supresor de descargas electroestáticas (ESD) USBLC6-4SC6 a la alimentación de entrada. Como su propio nombre indica es un circuito de protección contra ESD.

4.1.3. Divisores de tensión para interfaz USB-Serie

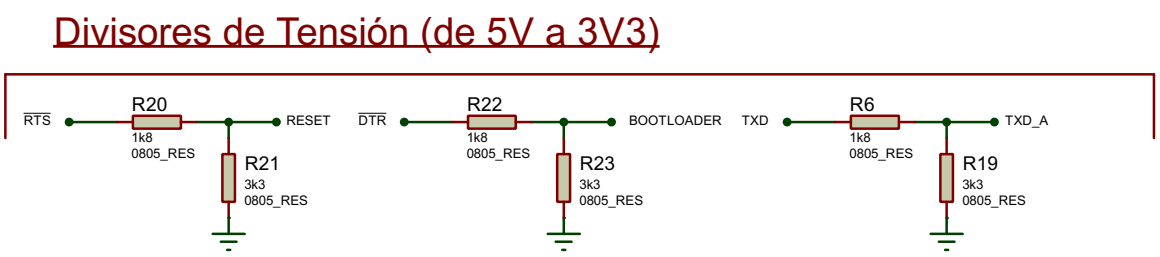


Figura 4.8: Divisores de tensión para interfaz USB-Serie

Un divisor de tensión es una configuración de circuito eléctrico que reparte la tensión eléctrica de una fuente entre una o más impedancias conectadas en serie. Por lo que a la

salida del divisor de tensión tendremos un voltaje menor que el de entrada, pero estos dos voltajes estarán relacionados por el valor de las resistencias.

Los divisores de tensión mostrados en la Figura 4.8 son utilizados para establecer la comunicación serie entre el FT232RL y el ESP-12F. Esto es así porque el voltaje de entrada que soporta el ESP-12F es de 3V3 mientras que el voltaje de salida del FT232RL es de 5V. Los valores de las resistencias utilizadas son valores típicos para divisores de tensión. De esta manera se consigue una correcta conversión de los niveles de voltaje de los datos para la comunicación serie.

4.1.4. Circuito microcontrolador para interfaz USB Host-SPI

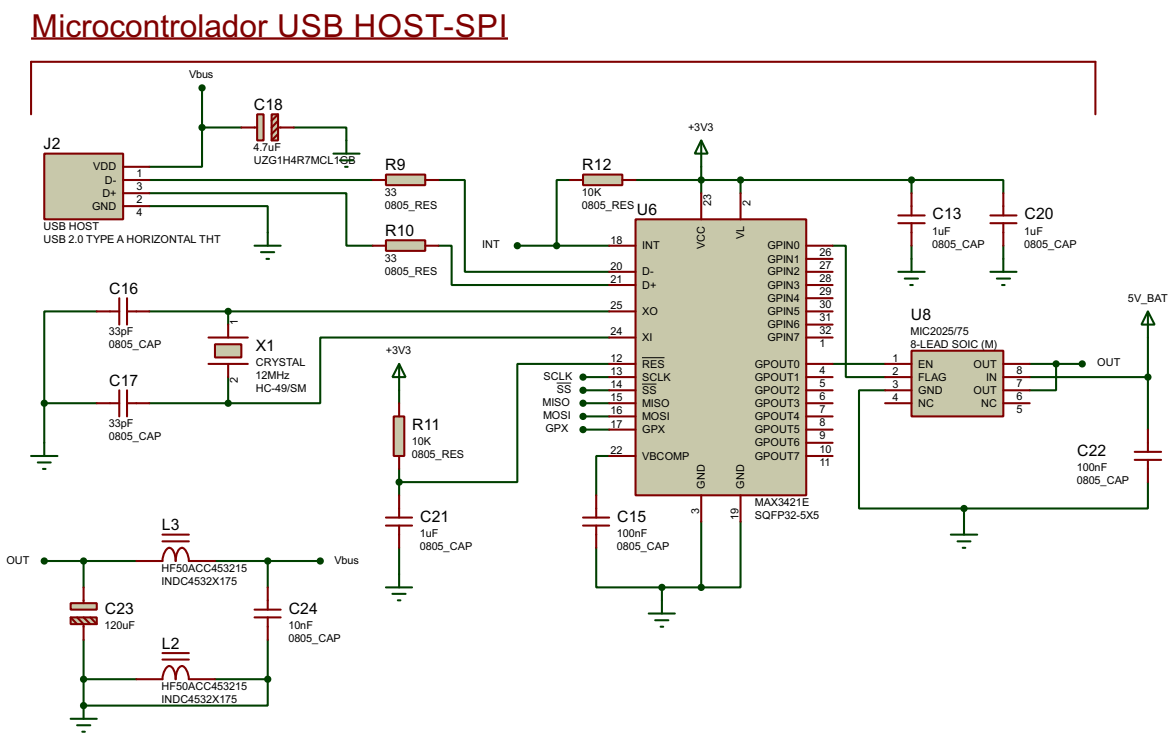


Figura 4.9: Circuito microcontrolador para interfaz USB Host-SPI

USB host es una extensión de la norma USB 2.0 que permite a los dispositivos USB tener mayor flexibilidad en la gestión de la interconexión. Esto hace posible que ciertos dispositivos actúen como host, por lo que se les puede conectar otros dispositivos USB y controlarlos.

El estándar USB utiliza una arquitectura maestro/esclavo: un hub USB (concentrador USB) actúa como USB maestro, mientras el dispositivo USB conectado actúa como esclavo. Solo el hub USB pueden gestionar la configuración y la transferencia de datos durante la conexión. Un controlador USB Host cambia esta situación ya que es capaz de abrir una

sesión, controlar la conexión e intercambiar las funciones maestro/dispositivo. Siguiendo las especificaciones para el desarrollo del prototipo, necesitamos un controlador USB Host para poder solicitar datos al sensor y de esta manera poder enviarlos a la aplicación móvil. Aquí surge la necesidad de buscar un controlador USB Host que se pueda comunicar con el microcontrolador ESP-12F mediante algún protocolo compatible.

El controlador USB Host MAX3421E contiene la lógica digital y los circuitos analógicos necesarios para implementar un periférico USB full-speed o un host full-/low-speed que cumpla con la especificación USB 2.0. El MAX3421E funciona usando un conjunto de registros al que se accede mediante una interfaz SPI que funciona hasta 26MHz. Cualquier maestro SPI (microprocesador) puede añadir periféricos USB o la funcionalidad de host mediante una interfaz sencilla SPI de 3 o 4 cables. Estas son algunas de sus características:

- Solución USB independiente del microprocesador.
- La interfaz SPI opera a un voltaje del sistema entre 1.4V y 3.6V.
- Incluye ocho entradas y salidas de propósito general.
- Cumple con la revisión de la especificación USB 2.0 (Full-Speed 12Mbps Peripheral, Full-/Low-Speed 12Mbps/1.5Mbps Host).
- Interfaz SPI de 26MHz.
- Pin de interrupción de salida.
- Voltaje de alimentación de 3V3.

El circuito de la Figura 4.9 sigue la configuración mostrada en el datasheet para *USB Peripheral/Host Controller with SPI Interface*. Además, se ha añadido el limitador de corriente MIC2025/75 que es un interruptor MOSFET optimizado para aplicaciones de distribución de alimentación de uso general que requieren protección en un circuito. Tiene internamente la corriente limitada y desconexión térmica para proteger el dispositivo y la carga.

4.1.5. Circuito de cargador de batería

Una de las principales características de este prototipo es que debe ser autónomo, esto es, debe ser capaz de funcionar sin estar conectado constantemente a la red eléctrica. Mediante el uso de una batería se podrá dejar conectado nuestro dispositivo al sensor en cualquier lugar, sin la necesidad de tener cerca la red eléctrica. Se ha optado por el uso de una batería Li-Ion. Estas, como cualquier batería, son dispositivos que están diseñados para almacenar la energía, la peculiaridad de este tipo de baterías es que emplea como electrolito una sal de litio que consigue los iones necesarios para la reacción electroquímica reversible que tiene lugar entre el cátodo y el ánodo. Sus ventajas son las siguientes:

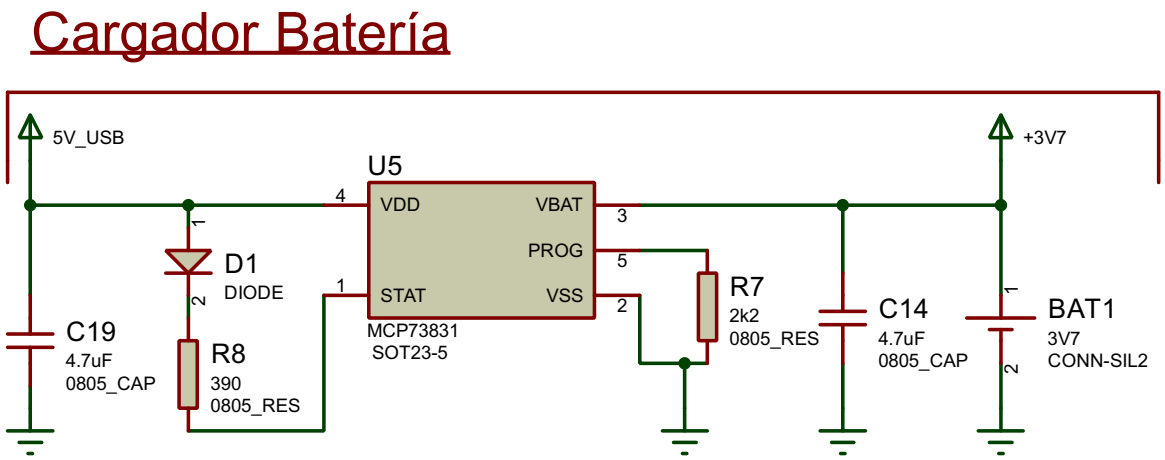


Figura 4.10: Circuito de cargador de batería

- Ligereza de sus componentes.
- Una elevada densidad de energía: acumulan mucha mayor carga por unidad de peso y volumen.
- Resistencia a la descarga. Muy baja tasa de autodescarga.
- Poco efecto memoria.
- Capacidad para funcionar con un elevado número de ciclos de regeneración.
- Facilidad para saber la carga que almacenan.

Estas propiedades permiten una gran flexibilidad, ya que en el mercado podemos encontrar una gran variedad de baterías Li-Ion tanto en su forma, tamaño y capacidad de almacenamiento.

Para poder cargar la batería a través de nuestro dispositivo es necesario un circuito cargador de batería. Se ha optado por el controlador de gestión de carga lineal MCP73831, ya que se adapta a aplicaciones de bajo coste y con espacio limitado, gracias a su pequeño tamaño físico y la baja cantidad de componentes externos. Este dispositivo incluye un transistor de paso y una detección de corriente integrado que emplean un algoritmo de carga de voltaje constante/corriente constante con terminación seleccionable de preacondicionamiento y carga. La Figura 4.10 muestra el circuito empleado en este proyecto, para ello se han seguido las indicaciones del datasheet del MCP73831, en el que aparece el esquemático con cada componente externo necesario y así como sus valores.

Con este circuito conseguimos pasar de una tensión de 5V a 3V7, que es la tensión necesaria para cargar la batería. Será el micro-USB (cuando se conecte a un PC o red eléctrica) el que se encargue de proveer los 5V al cargador de batería a través del FT232RL.

4.1.6. Circuito de regulador de tensión

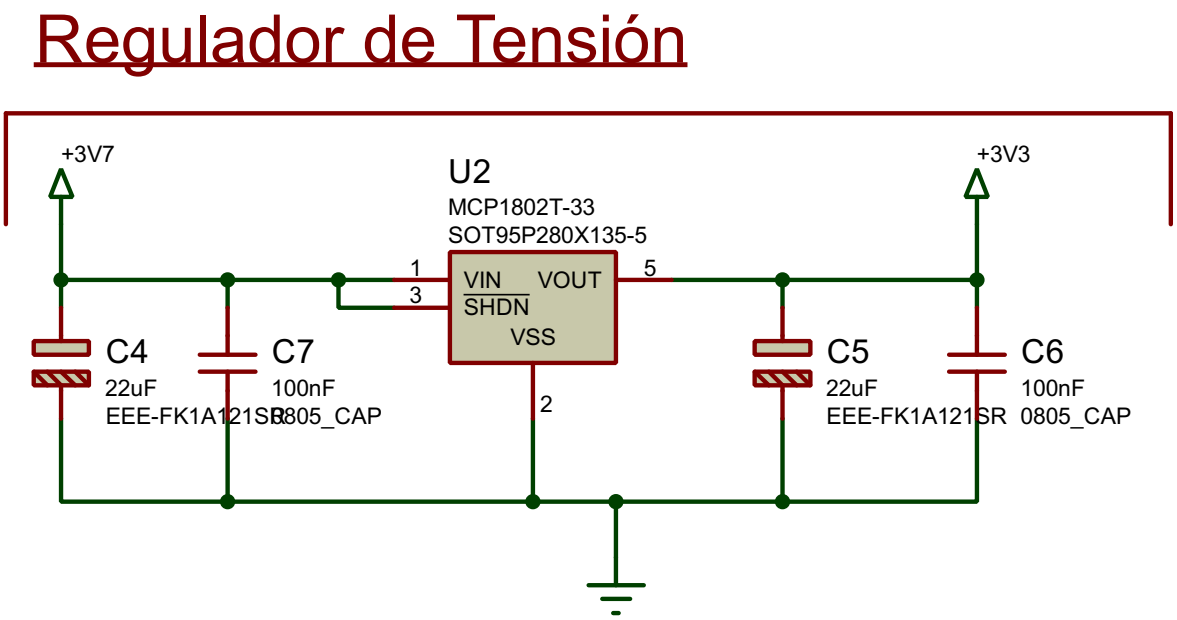


Figura 4.11: Circuito de regulador de tensión

La batería provee una tensión de 3V7, mientras que el microcontrolador ESP-12F y el controlador USB Host MAX3421E tienen una tensión de alimentación de 3V3, por ello es necesario regular esa tensión. Se ha utilizado el regulador de tensión MCP1802T que regula una tensión de entrada de 3V7, proveyendo una tensión de salida de 3V3. El MCP1802 es un regulador de voltaje de baja caída (LDO) de CMOS que puede administrar hasta 300mA de corriente mientras solamente consume 25uA de corriente de reposo (típica). El rango de operación de entrada se especifica de 2V a 10V. En el propio datasheet encontramos un circuito típico, con los componentes necesarios para que a la salida tengamos 3V3. Se han añadido condensadores electrolíticos a la entrada y la salida de este circuito.

4.1.7. Circuito de regulador elevador de conmutación (3v3 a 5v)

Anteriormente se ha señalado la necesidad de tener dos líneas diferentes para 5V. Esto es debido a que los 5V_USB proporcionados por el micro USB, son necesarios para alimentar el FT232RL, así como cargar la batería. Mientras que los 5V_BAT, son necesarios para alimentar el circuito cuando esté funcionando de forma autónoma, es decir, con batería. Se ha utilizado el convertidor Step-Up DC/DC RP401N501C que eleva una tensión de 3V7 a 5V. Para el desarrollo de este circuito se han seguido las indicaciones del datasheet, ya que los componentes varían según la tensión de entrada y salida.

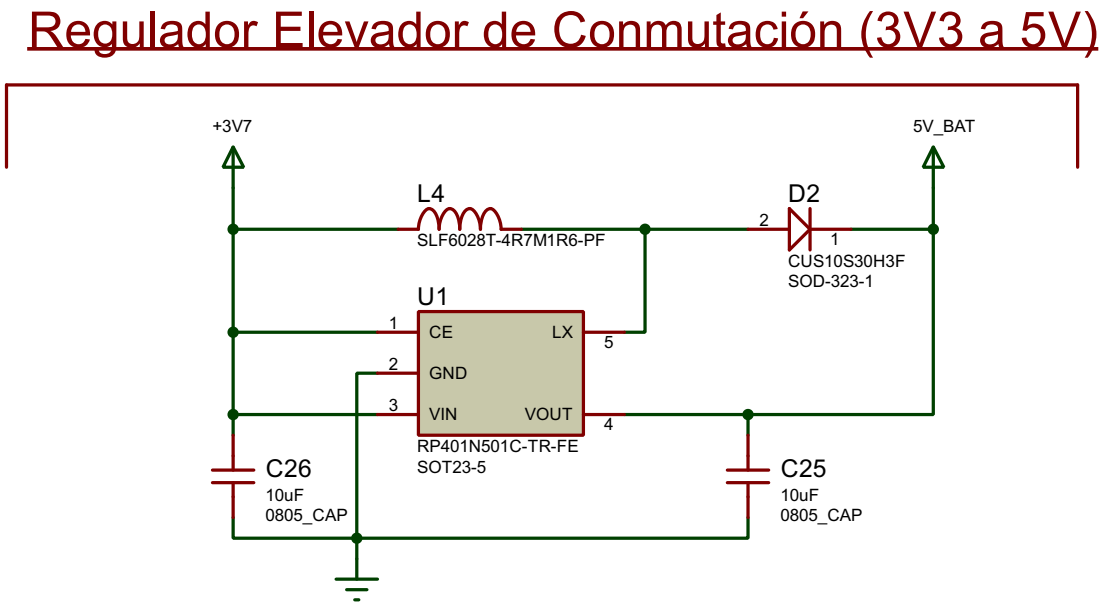


Figura 4.12: Circuito de regulador elevador de conmutación (3v3 a 5v)

4.2. Diseño de la placa de circuito impreso (PCB)

Una vez tenemos el esquemático correctamente diseñado, procedemos a realizar el diseño de la placa de circuito impreso. Para ello accederemos a la pestaña diseño PCB disponible en el propio Proteus. Destacar que en el proceso de diseño del circuito electrónico de este proyecto se ha realizado con Proteus 8, que a diferencia de su predecesor Proteus 7, tiene integrado en el mismo software tanto el diseño del esquemático como el diseño de la PCB, así como otras funcionalidades. Mientras que en la versión 7 el software se dividía en dos, ISIS para el diseño esquemático y ARES para el diseño PCB.

A medida que hemos ido desarrollando el diseño esquemático, Proteus va generando automáticamente la lista de nodos (NETLIST). Un nodo es un grupo de pines interconectados entre sí y la lista de nodos es una lista con todos los nodos que forman nuestro diseño. La pestaña de diseño PCB es capaz de recibir esta lista de nodos para diseñar, a partir de ella, nuestra placa de circuito impreso. De esta forma nos aseguramos que nuestra placa tendrá unidos entre sí los pines de forma idéntica a como los hemos definido en nuestro esquema electrónico. Podemos consultar la lista de nodos desde la pestaña "explorador del diseño" seleccionando la vista específica. Cualquier modificación que realicemos en el diseño electrónico, se refleja automáticamente en el resto de las pestañas de Proteus. De esta forma la modificación y rediseño de nuestra placa se realizará de forma mucho más simple y segura. En primer lugar, es necesario revisar todas las huellas de los componentes. La huella (o footprint) del componente electrónico es el espacio físico que van a ocupar

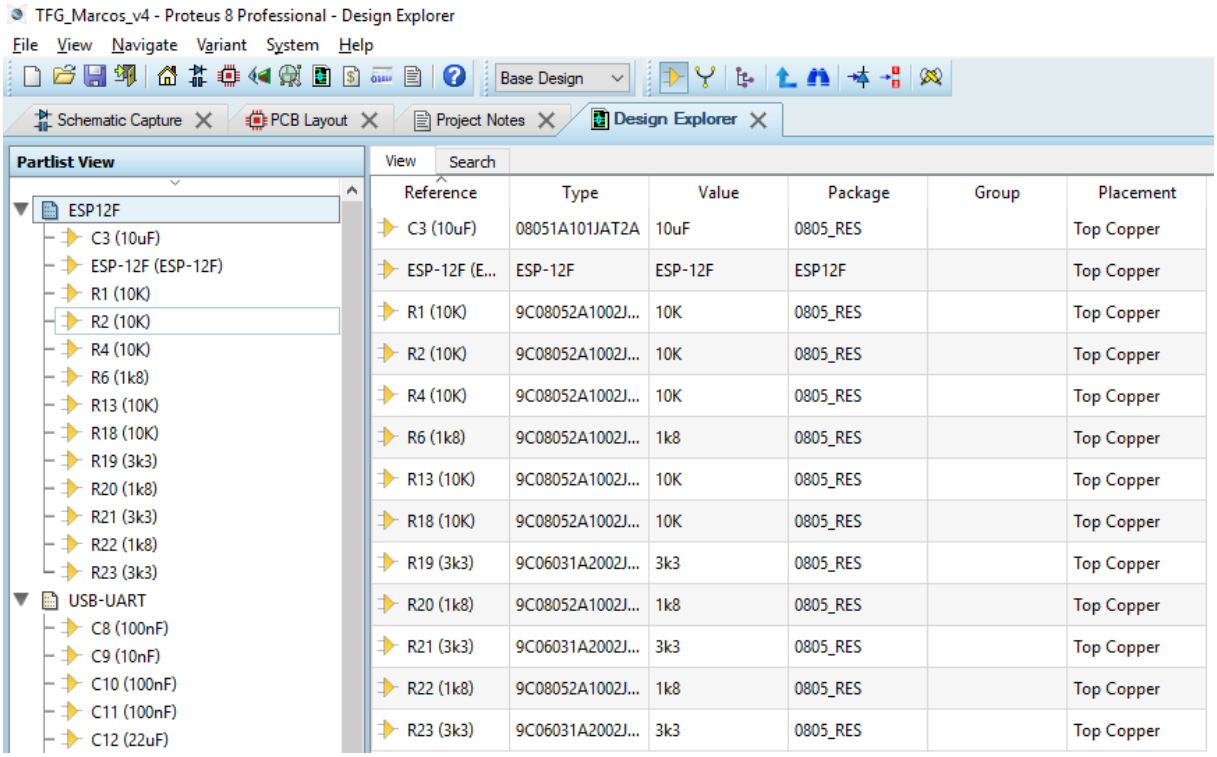
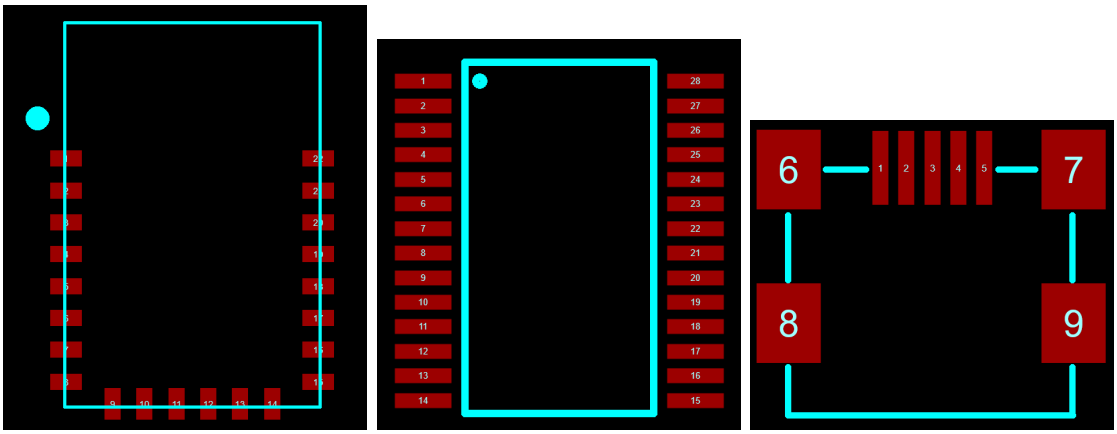


Figura 4.13: NETLIST del diseño electrónico en Proteus 8

nuestros componentes en la placa de circuito impreso, por lo tanto esta huella ha de tener las dimensiones adecuadas para permitir la soldadura del componentes electrónico a la PCB.

Esta huella debe ser consecuente con las medidas reales del componente, ya que cuando se fabrique la PCB la huella generará un conjunto de marcas o dibujos sobre ella y un conjunto de pads, que serán las superficies metálicas sobre los que soldaremos nuestros componentes y a través de los cuales se interconectarán las pistas del circuito. Por ello es importante revisar cada huella con las medidas indicadas en el datasheet y modificar y/o crear las huellas que sean necesarias.

Una gran parte de los componentes, sobre todo los más genéricos como resistencias, condensadores, etc. tienen huellas estándar (métrica 0805), es por esto que ya están incluidos en las propias librerías de Proteus. Suministra una extensa librería compuesta por más de 35.000 dispositivos. Esta librería incluye elementos estándar como resistencias, condensadores, transistores, diodos, válvulas, TTL, CMOS, ECL, microprocesadores, memorias, PLDs, ICs analógicos y amplificadores operacionales. En otros componentes, sin embargo, es necesario diseñar sus huellas siguiendo sus propios datasheet. Para este proyecto se han diseñado varias huellas como la del ESP-12F, mini-USB, FT232RL, etc.



(a) Huella ESP-12F (b) Huella FT232RL (c) Huella mini-USB

Figura 4.14: Huellas diseñadas

4.2.1. Módulo completo

Una vez se han diseñado todas las huellas de los componentes de manera correcta, procedemos a colocar los componentes dentro de un espacio delimitado previamente, siguiendo una serie de reglas. Al ser una empresa externa la que fabricará nuestra PCB, este área tendrá unas dimensiones estándar de 10cm x 10cm. Las medidas finales de la placa son de 7,8cm x 3,4cm, por lo que se tendremos la posibilidad de panelizar nuestro diseño a la hora de mandarlo fabricar. Para conseguir un buen diseño de la PCB se ha seguido una serie de directrices:

- Pistas lo más cortas posibles, minimizando así efectos parásitos resistivos y capacitivos.
- Las pistas de datos (D+ y D-, por ejemplo) deben tener la misma longitud entre ellas.
- Los dos puertos USB (componentes de entrada/salida) estarán próximos al borde de la PCB. Se ha optado por colocar uno a cada lado de la placa. Esto facilita su accesibilidad.
- Para el resto de componentes, se debe seguir el criterio de agruparlos según bloques lógicos. De esta manera se minimizaran las trazas de conexión.
- Los condensadores de desacoplo deben situarse cerca de los circuitos integrados que deben proteger, conectado a las pistas de alimentación o masa.
- Es conveniente evitar el trazado de ángulos de 90º.
- Pistas lo más anchas posibles, priorizando las de alimentación y datos. El ancho de la pista debe ser suficiente para permitir el paso de la corriente máxima que vaya a circular por la pista.

- Añadir planos de masa, pues el nivel de ruidos se reducirá considerablemente al blindar toda la placa. Mejorará la CEM (Compatibilidad Electromagnética).
- Respetar la distancia mínima entre pistas y los bordes de la placa
- Se debe prever la sujeción de la placa a un chasis o caja, para ello se dispondrá un agujero de inserción en cada esquina. de la placa.

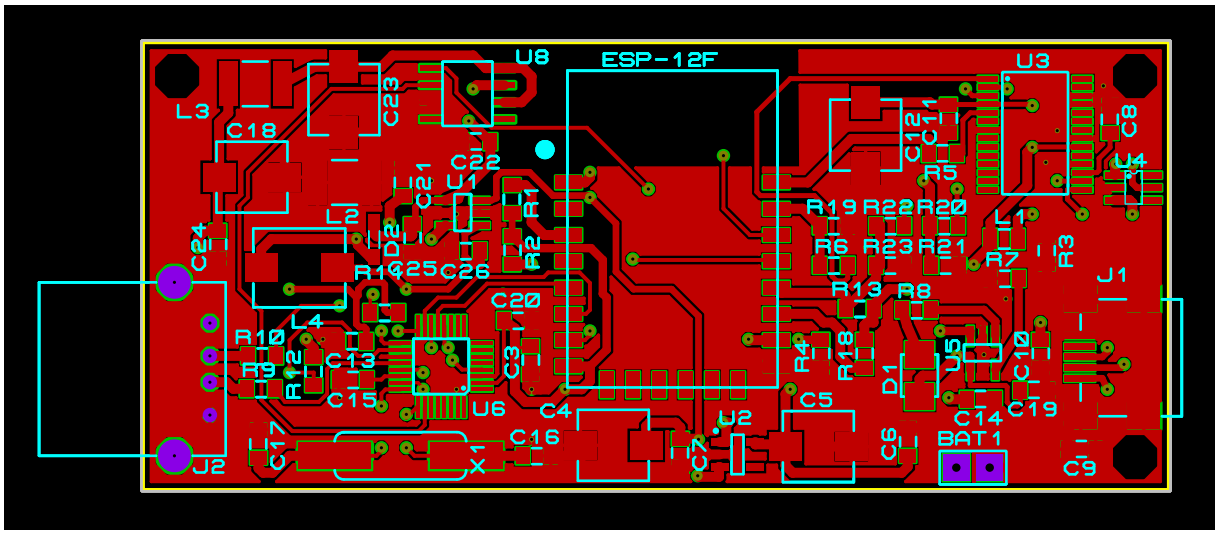


Figura 4.15: Layout módulo (cara frontal)

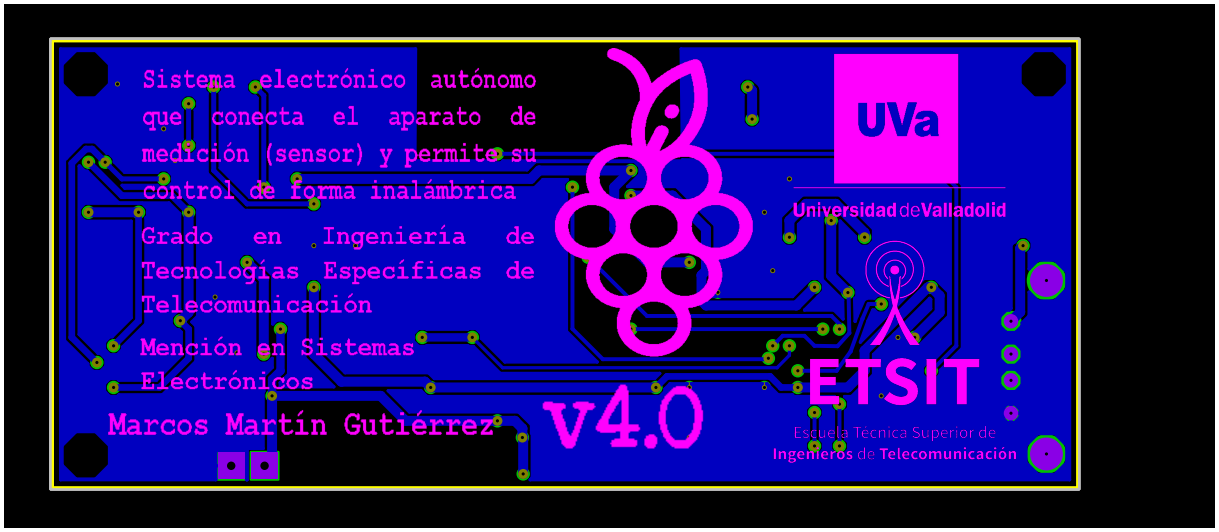


Figura 4.16: Layout módulo (cara trasera)

Partiendo de las reglas anteriores y colocados todos los componentes de forma óptima en el área delimitada, se debe enrutar la placa. Para ello hacemos uso de la opción "Autoroute" de Proteus, que ruta automáticamente toda la PCB. Posteriormente, arreglaremos

manualmente todos los CRC (Cyclic Redundancy Check) y DRC (Design Rule Check) errors.

El diseño PCB de este proyecto parte de la idea de que desde un lado debemos acceder al sensor y desde otro debemos programar el microcontrolador y cargar la batería. De esta manera, se ha colocado cada dispositivo de entrada salida en cada lado de la placa. Así tenemos que el lado izquierdo dispone del conector USB Host, que será donde irá conectado el sensor. Mientras que en el lado derecho tenemos el mini-USB que será por donde cargaremos la batería y programaremos el ESP-12F. Este último se ha colocado en el centro de la placa, ya que hace de punto intermedio. En la parte derecha, entre el USB Host y el ESP-12F se ha colocado el bloque lógico del MAX3421E, mientras que en la parte derecha (entre el ESP-12F y el mini-USB) se ha colocado el bloque lógico del FT232RL. El resto de bloques lógicos (cargador de batería, regulador de tensión) excepto el elevador de tensión, se han situado en la parte inferior derecha, cerca de la entrada mini-USB. Y por último, el bloque lógico del elevador de tensión se ha colocado en la parte superior izquierda, cerca del USB Host.

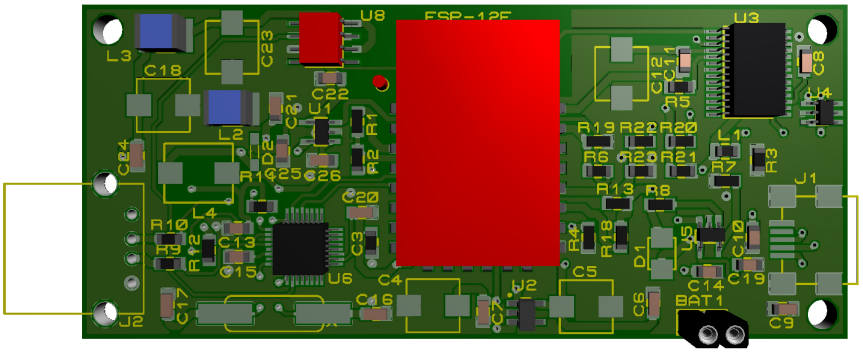


Figura 4.17: Vista 3D módulo (cara frontal)

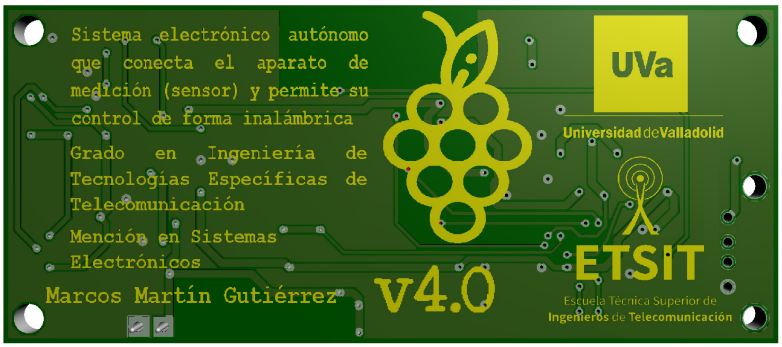


Figura 4.18: Vista 3D módulo (cara trasera)

Proteus 8 permite la visualización 3D de nuesrta PCB. Una vez tenemos diseñada nuestra PCB podemos acceder a la pestaña "3D Visualizer", en ella podemos realizar una visualización 3D de la PCB, en la que podremos encontrar componentes, placa y serigrafía. También se permite exportar el fichero 3D en formato 3DS. En las siguientes figuras vemos la parte frontal y trasera del circuito.

4.2.2. Consumo eléctrico

Es importante conocer el consumo eléctrico de nuestro dispositivo, ya que al ser un dispositivo autónomo conviene conocer el tiempo de funcionamiento. Esto nos permite seleccionar entre una gama de baterías con distintas capacidades y elegir entre ellas la que más se ajuste a nuestras necesidades. Se ha realizado un análisis del consumo eléctrico, seleccionando las corrientes máximas de cada dispositivo activo de nuestro sistema. En la Figura 4.19 se aprecia cada uno de estos componentes activos, con sus voltajes de alimentación y sus corrientes máximas. Estas se han recopilado de cada uno de los datasheets correspondientes. Como se aprecia, para una batería de 3V7 con una capacidad de 1500 mAh, el tiempo estimado de funcionamiento (para el peor caso) será de aproximadamente 7 horas. Esto se consigue dividiendo la capacidad por hora de la batería entre la suma de las corrientes máximas de los componentes activos con una alimentación igual o inferior a 3V7. Anteriormente se ha comentado la flexibilidad de las baterías de Li-Ion, esto nos permite elegir la más conveniente según la capacidad que posea.

CONSUMO DE POTENCIA

COMPONENT	VOLTAGE	CURRENT
ESP-12F	3.3V	170mA
MAX3421E	3.3V	45mA
TOTAL	3.3V	215mA
RP401N501C-TR-FE	3.7V	500µA
MCP1802T	3.7V	25µA
TOTAL	3.7V	215.525mA
MIC2025/75	5V	5µA
diode	5V	9mA
FT232rl	5V	15mA
MCP73831	5V	1.5mA
TOTAL	3.7V	241.3mA
USB	500mA máx	

FINAL CALCULATIONS:

LIPO battery 3.7V 1500mAh

(1500mAh/215.5mA) = 6.96h

Figura 4.19: Consumo eléctrico

4.2.3. Fabricación PCB

En la fabricación de la PCB de este proyecto se ha optado por una fabricación externa, mas concretamente mediante la empresa JLCPCB (Shenzhen JIALICHUANG Electronic Technology Development Co.,Ltd.), que es una empresa que fabrica PCB a precios muy bajos (10 PCB por 2 dólares), con una alta calidad y en tiempo bastante corto, según su página web ellos son los mayores productores de PCB en china, teniendo más de 200.000 clientes en todo el mundo y más de 8000 ordenes al día.

A través de las herramientas Proteus se ha generado el Gerber necesario para la fabricación de la PCB. Gerber es el formato de fichero estándar en la industria electrónica para comunicar la información del diseño de una PCB a un fabricante. La información de una PCB completa no va en un único fichero gerber sino en varios:

- Fichero de las trazas en el cobre de la cara TOP.
- Fichero de las trazas en el cobre de la cara BOTTOM.
- Fichero de la solder mask en cara TOP.
- Fichero de la solder mask en cara BOTTOM.
- Fichero de la silkscreen en cara TOP.
- Fichero de la silkscreen en cara BOTTOM.
- Fichero con posición de los agujeros (drills).
- Fichero con información del tamaño de agujeros.
- Fichero con información del contorno de la PCB.
- Fichero tipo README donde se indica que es cada uno de los ficheros gerbers.

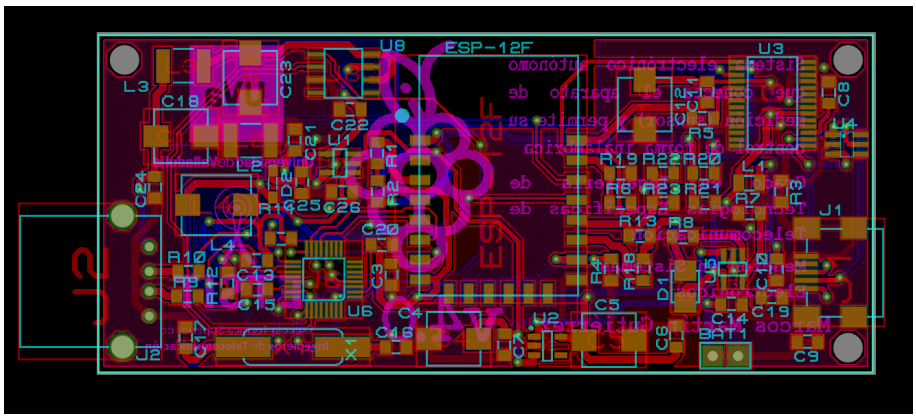


Figura 4.20: Gerber PCB

Ya generado este fichero, desde la propia web de JLCPCB se adjunta al pedido y en un periodo de tiempo entre 10-15 días podremos disponer de nuestras PCB. Destacar que para este proyecto existe la posibilidad de panelizar el Gerber para generar dos placas en un solo fichero Gerber. Esto es porque el ancho de las placas es de 3,4cm, mientras que el máximo permitido es de 10cm. Por lo que se podrán panelizar concretamente dos placas en un mismo Gerber.

4.3. Montaje de los componentes

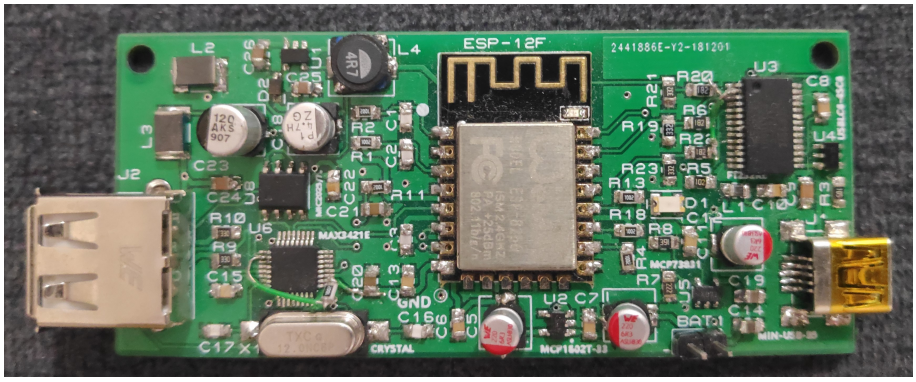


Figura 4.21: Módulo cara frontla

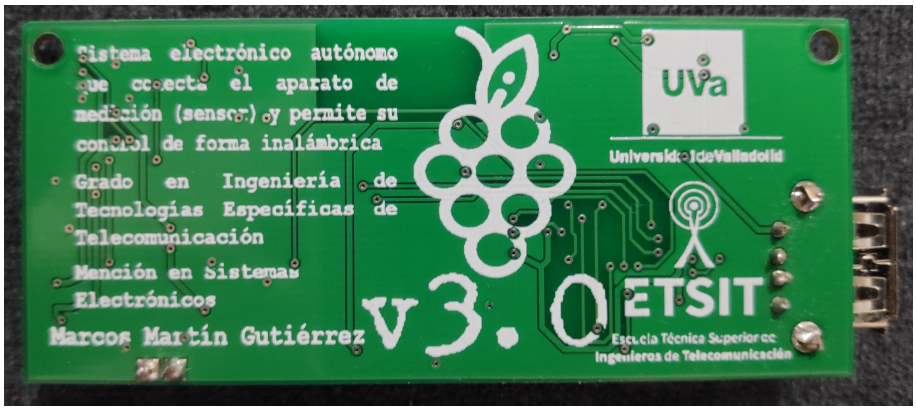


Figura 4.22: Módulo cara trasera

Una vez que hemos recibido las PCBs ya podemos realizar el montaje y soldadura de los componentes. Para la óptima realización de este proceso se ha hecho uso del Laboratorio de proyectos, disponible en la Escuela Técnica Superior de Ingenieros de Telecomunicación. Este laboratorio posee todas las herramientas necesarias para el montaje de placas de circuito impreso: soldadores de precisión, microscopios digitales, osciloscopios, multímetros, etc.

Es importante antes de empezar a soldar, disponer de una lista de los componentes que vamos a soldar así como los respectivos esquemáticos y diseño PCB en formato impreso.

Esto nos permitirá que la soldadura y montaje se haga de la manera más correcta y precisa, ya que podremos comprobar en todo momento la ubicación de los componentes.

En primer lugar soldaremos los componentes más pequeños. En este caso se empezó soldando los siguientes componentes: MAX3421E, MCP73831, MCP1802T-33 y FT232RL, ya que son los componentes con las patillas más pequeñas. Posteriormente se procedió a soldar los condensadores y resistencias más pequeños y el diodo. Después se pasó a soldar los condensadores más grandes, así como inductancias y el oscilador. Más tarde, se soldó el ESP12-F, mini-USB ya son los componentes con los pines más grandes. Y por último, se sueldan los componentes que atraviesan la placa, como son: USB Host y el conector de la batería.

En este momento ya tenemos el montaje completo de nuestra placa. Se procede a conectarla al ordenador y vemos como el ordenador reconoce nuestra placa. Además, vemos como el LED que hemos añadido junto con el conjunto lógico del FT232RL luce. Por otro lado, también comprobamos mediante el multímetro que los voltajes en cada zona del circuito son correctos, tanto con el sistema funcionando con la batería, como con el sistema conectado al PC.

Capítulo 5

Desarrollo firmware

5.1. Introducción

Una de las partes fundamentales para el desarrollo de un prototipo electrónico es el desarrollo software. También es una de las partes que mayor tiempo requieren dada su complejidad. Debido a esto es una buena práctica realizar en paralelo el desarrollo del firmware y hardware, mientras se fabrica nuestra PCB. En esta memoria de Trabajo Fin de Grado se ha detallado en profundidad todo el desarrollo hardware, pero una vez que el prototipo está creado y es reconocido por cualquier PC, es el momento de la elaboración del firmware. Según la definición de firmware: "Se conoce como firmware al conjunto de instrucciones de un programa informático que se encuentra registrado en una memoria ROM, flash o similar. Estas instrucciones fijan la lógica primaria que ejerce el control de los circuitos de alguna clase de artefacto. Una de las funciones más importantes de todo firmware es controlar y gestionar tanto lo que es el arranque del sistema del dispositivo como la correspondiente iniciación". Por ello, su desarrollo es tan importante como el desarrollo del hardware, ya que nuestro sistema no tendría ninguna utilidad sin él.

El primer paso es conocer las especificaciones del software necesario, es decir, saber que entorno de desarrollo debemos usar, que lenguaje de programación y las librerías necesarias. En nuestro caso, al ser un microcontrolador ESP-12F, el entorno de desarrollo será Arduino IDE, el cual se detallará más adelante. También es importante conocer a fondo los componentes empleados en nuestro prototipo, esto nos permitirá saber que drivers necesitamos en nuestro PC para que este los reconozca adecuadamente, así evitarnos problemas de compatibilidad. Una vez que ya disponemos de las herramientas necesarias diseñaremos el software necesario que nos permita cumplir las especificaciones del proyecto. En este punto se irá desarrollando de forma paralela el firmware del microcontrolador con el software de la aplicación Android. Más adelante se detallarán ambos softwares. Se realizarán diferentes pruebas para comprobar el correcto funcionamiento de cada software por separado. Por último, integraremos ambas partes, comprobando que el conjunto funciona de manera adecuada y solucionando errores si fuera necesario.

5.2. Entorno de desarrollo

En la elaboración y compilación del firmware se ha empleado el entorno de desarrollo Arduino IDE. Para hablar de este entorno de desarrollo es necesario conocer Arduino, ya que este ha desempeñado un papel muy importante para los aficionados del mundo de la electrónica. Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo (software), diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. Arduino es una plataforma abierta que facilita la programación de un microcontrolador. Las principales características de Arduino es que es un hardware de código abierto, tiene una baja complejidad, lo que permite desarrollar fácil y rápido. Arduino IDE es el entorno de desarrollo gratuito para Arduino que proporciona un editor de lenguaje C más las herramientas necesarias para compilar e implementar, así como bibliotecas de rutinas C necesarias para programar diferentes placas. Un ejemplo claro es la programación UART, la cual requeriría establecer registros, manejar interrupciones, etc. Para evitar todo ese trabajo Arduino IDE proporciona bibliotecas de alto nivel.

Arduino IDE dispone las librerías necesarias para ESP12-F. Esto se debe gracias a que una serie de talentosas personas han construido un proyecto Open Source Github que proporciona un "plug-in" o "extensión" a la herramienta IDE Arduino. Lo que hace esta extensión es permitir escribir código en el IDE de Arduino que aprovechan las interfaces de la biblioteca Arduino que, en tiempo de compilación y despliegue, generan código que se ejecutará en el ESP8266. De esta manera, se puede usar el IDE de Arduino para generar firmware como si de un Arduino se tratara. Concretamente para ESP8266 se han añadido bibliotecas para comunicar con WiFi usando TCP e IP, crear servidores, usar un sistema de archivos en la memoria flash, trabajar con tarjetas SD, servidores y periféricos.

5.3. Desarrollo firmware ESP-12F

Los dispositivos que se conectan a la red WiFi se llaman estaciones (STA). La conexión a WiFi es proporcionada por un punto de acceso (AP), que actúa como un centro para una o más estaciones. El ESP-12F puede operar en dos modos de conexión WiFi:

- Modo Soft ACCESS POINT (Soft AP), punto de acceso habilitado por software, para establecer su propia red WiFi. Por lo tanto, podemos conectar otras estaciones al módulo ESP8266.
- Modo STATION, para conectarla a una red WiFi a través de un punto de acceso.

Figura 5.1: Soft ACCESS POINT

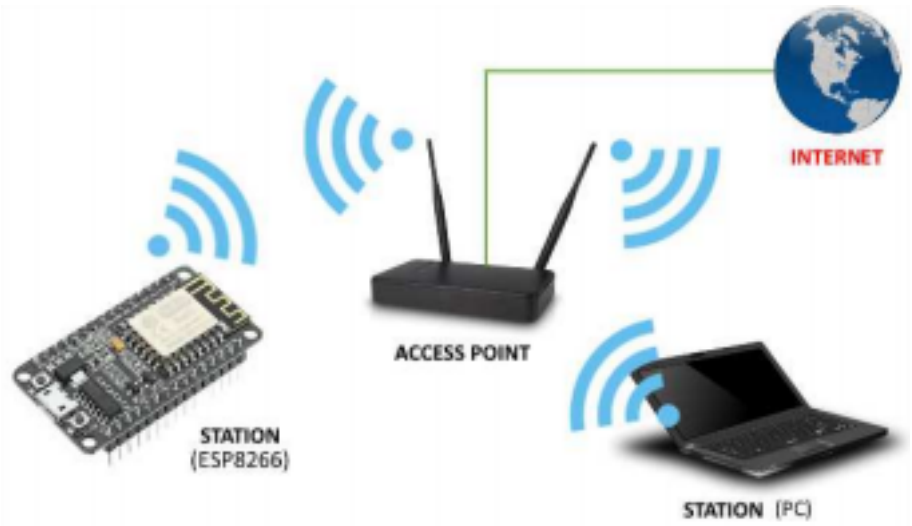


Figura 5.2: STATION

Un punto de acceso se reconoce por su SSID (Service Set Identifier), que será el nombre identificador de la red WiFi a la que se conecta cada estación. Cuando el chip ESP8266 funciona como punto de acceso deberemos elegir el SSID con el que queremos identificar la red así como la contraseña. Sin embargo, si el chip funciona como estación deberemos proporcionar a este los datos de conexión del punto de acceso (router) de la red local al que se conectará. En nuestro caso solamente necesitaremos usar el modo Access Point, ya que nuestro módulo será utilizado de pasarela entre en sensor y la aplicación Android. Para ello se han usado las siguientes librerías:

- ESP8266WiFi.h: Esta librería proporciona las rutinas específicas WiFi de ESP8266 a las que llamamos para conectarse a la red. Es la que nos permitirá seleccionar el modo de funcionamiento soft AP para nuestro ESP-12F.
- ESP8266WebServer.h: Esta librería permite disponer de un servidor HTTP que atiende las peticiones que se realicen. Esto nos será de utilidad a la hora de comunicarnos con la aplicación Android, ya que será esta la que nos haga la petición de un tipo de dato. También permite pasar parámetros. El método hanldeClient() será el que escuche las peticiones del servidor.

La estructura de comunicación para este proyecto será la siguiente: El módulo ESP-12F actúa como punto de acceso para la aplicación móvil. Esta se conectará por WiFi directo al ESP-12F mediante un SSID y una contraseña. Una vez conectada la aplicación móvil mandará una petición de datos, mientras que el ESP-12F estará escuchando. La aplicación mandará el tipo de datos que solicita, cuando el ESP-12F conoce el tipo de dato enviará como parámetro a la función que gestiona la comunicación con el sensor mediante el protocolo SPI. Esta función devolverá el dato solicitado y será el ESP-12F el que enviará este dato a la aplicación móvil.

Hasta el momento se ha explicado cómo es la comunicación ESP-12F<->APP, por ello se va a proceder a explicar cómo es la comunicación SENSOR<->ESP-12F.

Para la comunicación entre el sensor y el microcontrolador se ha utilizado el controlador host MAX3421E. La comunicación entre estos dos dispositivos es necesario el protocolo SPI (Serial Peripheral Interface). Este estándar de comunicaciones permite controlar el flujo de datos en serie regulado por un reloj, es decir, de comunicación síncrona. Incluye una línea de reloj, dato entrante, dato saliente y un pin de selección de chip, que permite conectar o desconectar el dispositivo con el que uno desea comunicarse, permitiendo multiplexar las líneas de reloj. La sincronización y la transmisión de datos se realiza por medio de 4 señales:

- SCLK (Clock): Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit.
- MOSI (Master Output Slave Input): Salida de datos del maestro y entrada de datos al esclavo.
- MISO (Master Input Slave Output): Salida de datos del esclavo y entrada al maestro.

Figura 5.3: Comunicación SPI

- SS/Select: Para seleccionar un esclavo, o para que el maestro le diga al esclavo que se active.

El bus SPI tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro (master) puede iniciar la comunicación con uno o varios dispositivos esclavos (slave), y enviar o recibir datos de ellos. Los dispositivos esclavos no pueden iniciar la comunicación, ni intercambiar datos entre ellos directamente.
El funcionamiento del bus SPI es el siguiente: Es el maestro el que mantiene todas las

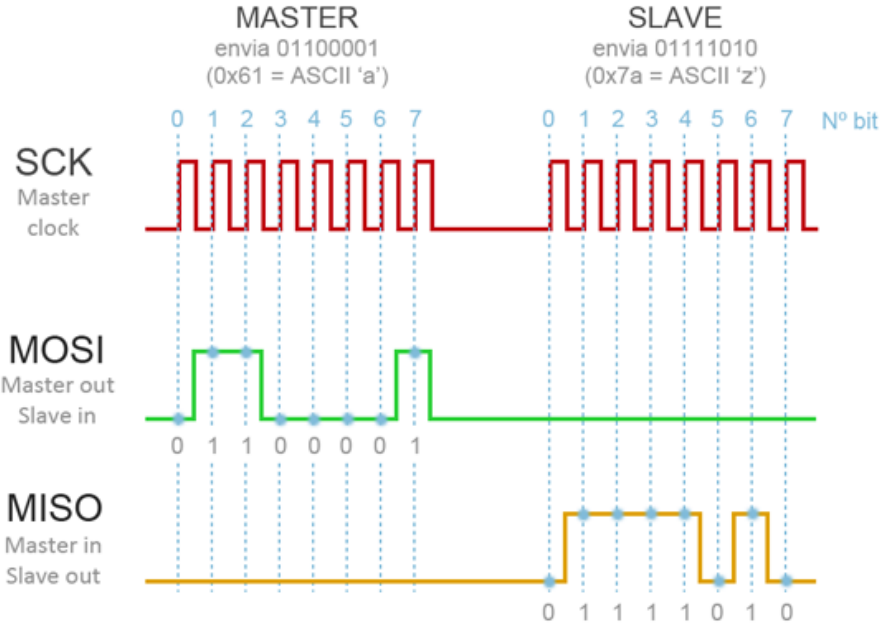


Figura 5.4: Comunicación SPI

líneas SS en estado ALTO. Si el maestro quiere comunicarse con un esclavo pone en BAJO la línea SS correspondiente al esclavo. Esto hace que el esclavo sepa que debe iniciar la comunicación. Una vez realizada la selección del esclavo, en cada pulso de la señal de reloj (por lo general flanco de subida) el maestro enviará un bit al esclavo y al mismo tiempo recibirá un bit del esclavo. La trama de datos no sigue ninguna regla concreta, por ello es necesario que tanto maestro como esclavo conozcan la longitud y el significado de los datos que van a enviar y recibir. Las principales ventajas del bus SPI son: Alta velocidad

de trasmisión y Full Duplex, dispositivos necesarios sencillos y baratos, puede mandar secuencias de bit de cualquier tamaño, sin dividir y sin interrupciones.

En nuestro firmware se ha utilizado la librería SPI.h que es la que nos permite establecer la comunicación ESP-12F<->MAX3421E. También se ha utilizado la librería usbhub.h que es la que permite controlar el MAX3421E como USB Host. Dentro de nuestro firmware se ha utilizado la función getSensorData(). Esta es una adaptación de una función de la librería para Arduino USB_Host_Shield_2.0 desarrollada por Oleg Mazurov. El principio de esta función es ciertamente sencillo: anteriormente hemos mandado el tipo de dato solicitado por la aplicación Android como parámetro a esta función. La función manda este parámetro al sensor a través de otra función como es Ftdi.SndData().

Dependiendo del tipo de dato el sensor devolverá un dato u otro. Posteriormente se le solicita al sensor la respuesta a esa petición mediante la función Ftdi.RcvData(), la cual rellenará un buffer con los datos enviados por el sensor. Los datos de este buffer serán devueltos como retorno de la función getSensorData() para que puedan ser enviados a la aplicación móvil.

Para la ayuda al entendimiento del flujo del firmware se ha creado el diagrama de flujo de la Figura 5.4. Este firmware se ha desarrollado para que sea ampliamente escalable y tenga poca dependencia del sensor que se conecte, bastaría con conocer que tipo de dato solicita el sensor.

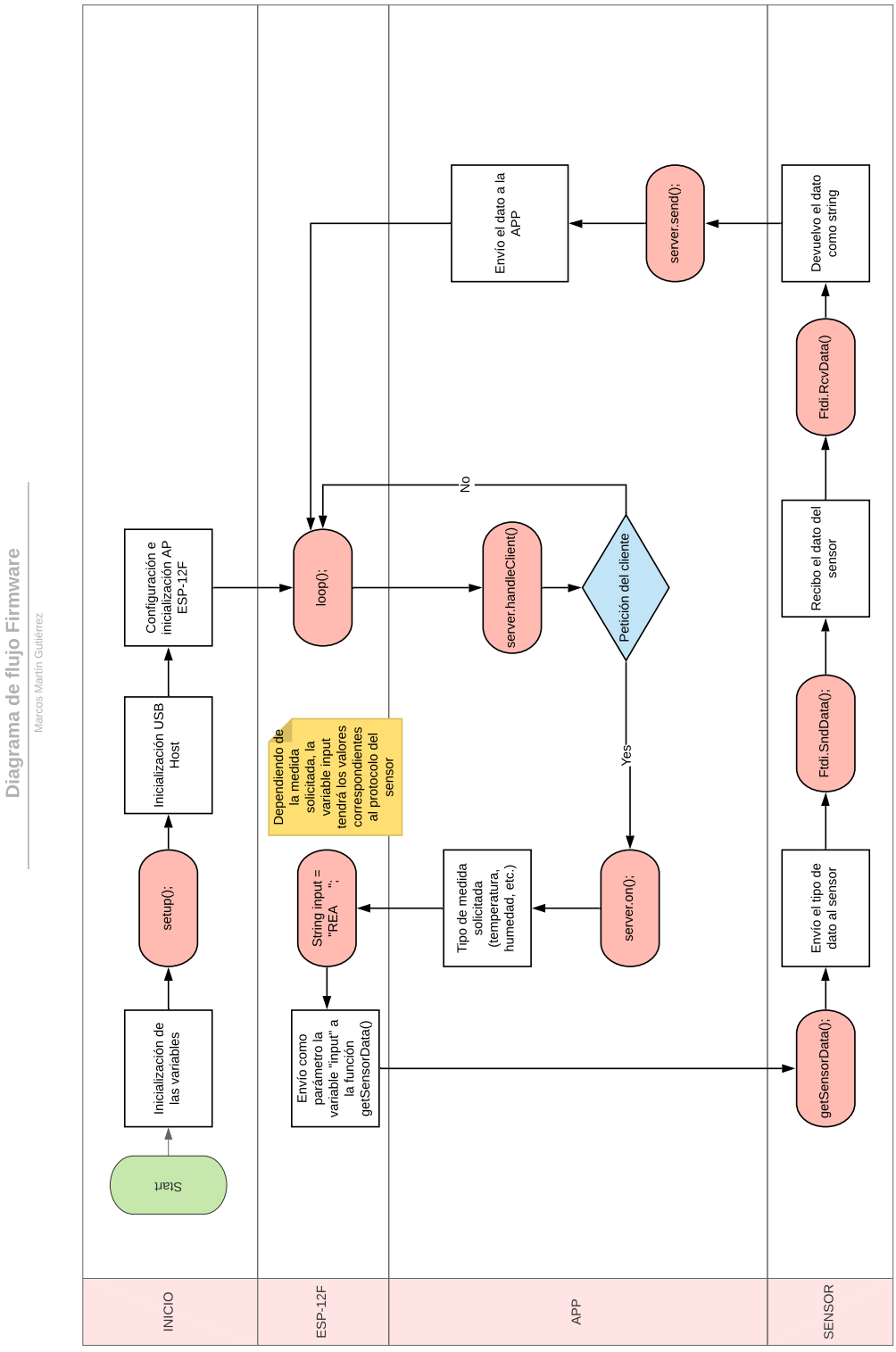


Figura 5.5: Consumo eléctrico

Capítulo 6

Desarrollo software aplicación móvil

6.1. Introducción

Para el desarrollo de un proyecto IoT es importante, además de disponer de un sistema electrónico que se conecte a Internet, que este sistema pueda ser controlado desde un dispositivo móvil. Es por esto que se ha creado una aplicación móvil simple que solicite datos, muestre los datos recibidos y envíe estos datos a una base de datos:

- La aplicación móvil debe ser capaz de pedir un tipo de datos al sistema, para que este a su vez solicite estos datos al sensor. Esta aplicación móvil desarrollada solicita los datos de Temperatura y Humedad de forma constante. Esto es ampliamente escalable permitiendo introducir nuevos tipos de datos de forma sencilla como línea futura.
- Se ha añadido una interfaz gráfica que se actualiza en tiempo constante, cuya misión es mostrar el dato requerido al sensor. Esta gráfica recibe el dato medido por el sensor cada dos segundos y dibuja sobre una gráfica temporal la variación de ese dato.
- Por último la aplicación debe ser capaz de enviar estos datos a una base de datos para su posterior análisis.

El realizar el control del sistema electrónico desde una aplicación móvil permite una cierta movilidad y flexibilidad ya que cualquiera con un smartphone y la aplicación Android instalada puede controlar recoger los datos del sensor y almacenarlos en una base de datos.

La aplicación se ha desarrollado para el sistema operativo Android debido a las facilidades para el desarrollo de aplicaciones para esta plataforma, ya que es un software de código abierto es gratis y accesible a todo el mundo. El lenguaje de programación para crear la aplicación Android utilizado ha sido Java.

6.2. Entorno de desarrollo

El entorno de desarrollo utilizado para la realización de esta aplicación Android ha sido Android Studio que es el entorno de desarrollo integrado oficial para la plataforma

Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

Algunas de sus características principales son las siguientes:

- Integración de ProGuard y funciones de firma de aplicaciones.
- Renderizado en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Un sistema de compilación basado en Gradle flexible.
- Refactorización específica de Android y arreglos rápidos.
- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Soporte integrado para Google Cloud Platform, que permite la integración con Google Cloud Messaging y App Engine.
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK.
- Gran cantidad de herramientas y frameworks de prueba.
- Compatibilidad con C++ y NDK.
- Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.

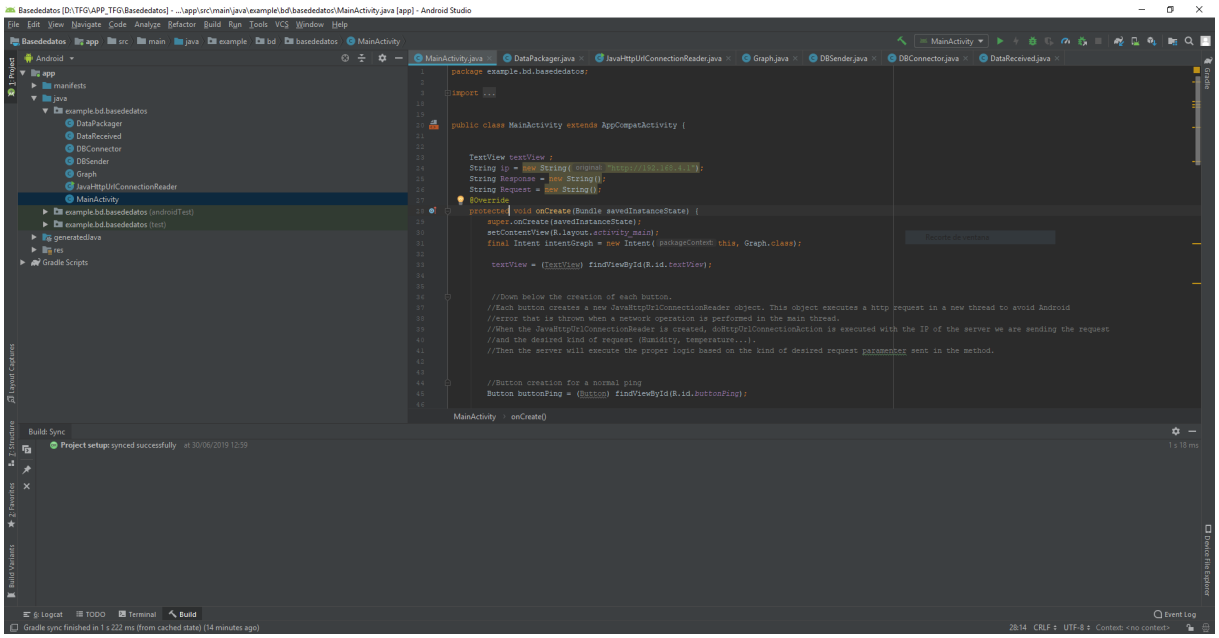


Figura 6.1: Entorno de desarrollo Android Studio

6.3. Interfaz gráfica

La aplicación desarrollada funciona en líneas generales de la siguiente manera: En primer lugar el usuario debe conectarse a la red WiFi creada por el sistema electrónico, una vez que el smartphone esté conectado se accede a la aplicación Android. En el menú inicial podremos seleccionar entre Humedad y Temperatura, como datos a solicitar al sensor, mediante unos botones. Pulsando en cualquiera de los dos accedemos a otra interfaz donde se mostrará una gráfica vacía en la que el eje X será el intervalo de tiempo, mientras que el eje Y será el rango de unidad del dato solicitado en un intervalo determinado, que será lo más próximo a la realidad. Automáticamente esta gráfica se irá actualizando y dibujando con los datos recibidos vía WiFi Direct. Como se aprecia en la imágen hay un Switch denominado “Wifi”, este sirve para apagar el WiFi del teléfono móvil, con lo que este quedará conectado mediante la tecnología móvil correspondiente (3G, 4G, etc.). A la derecha de este Switch hay un botón “ENVIAR A BD” que, como su propio nombre indica sirve para enviar estos datos a la base de datos.

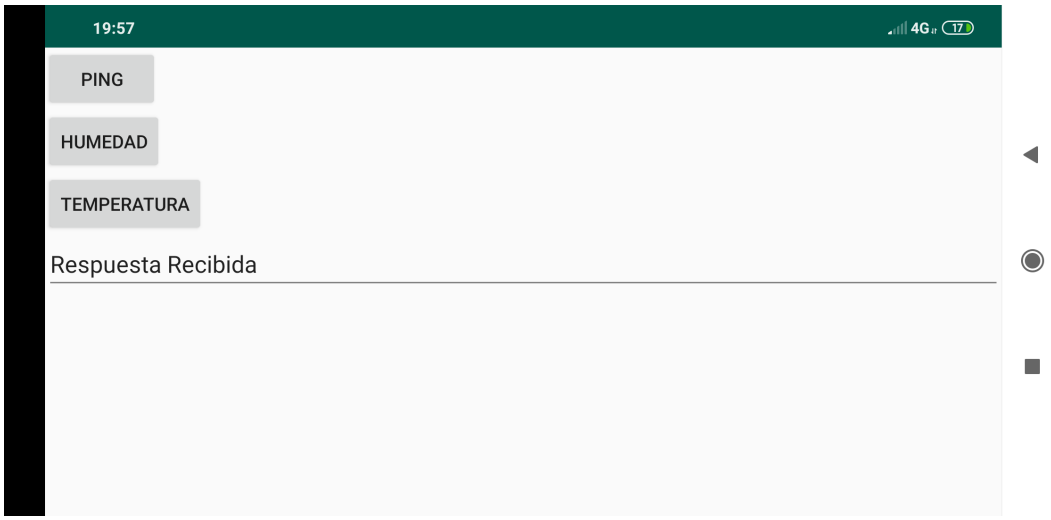


Figura 6.2: Menú inicial App

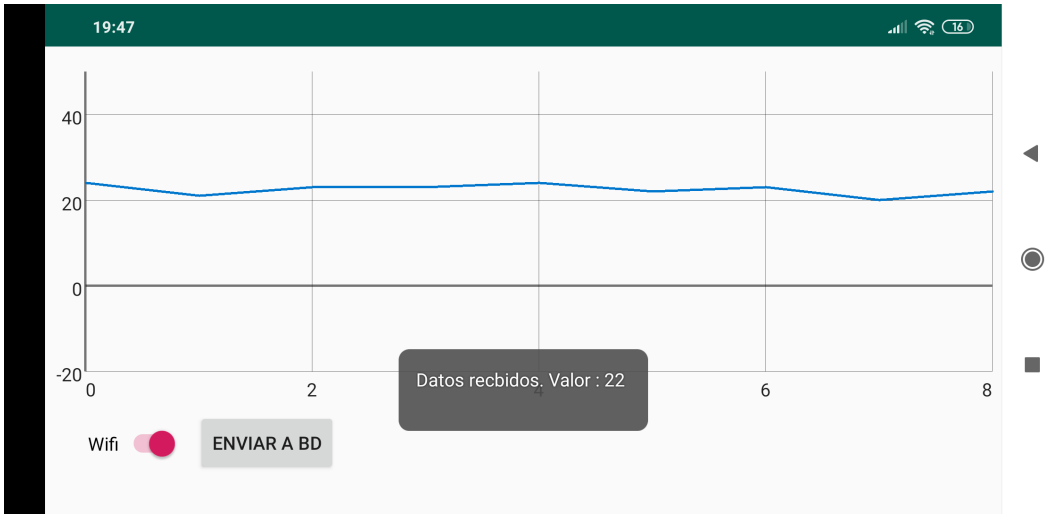


Figura 6.3: Gráfica temperatura App

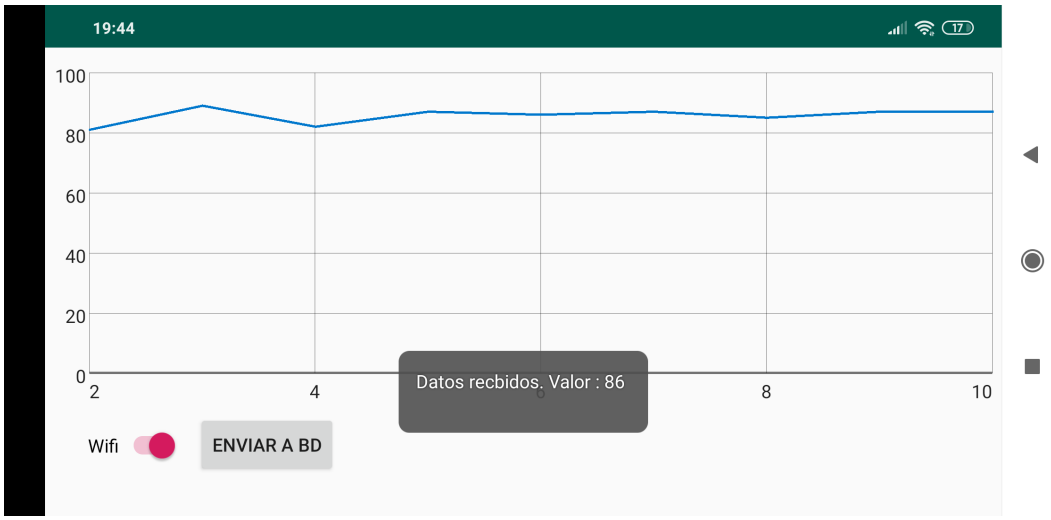


Figura 6.4: Gráfica humedad App

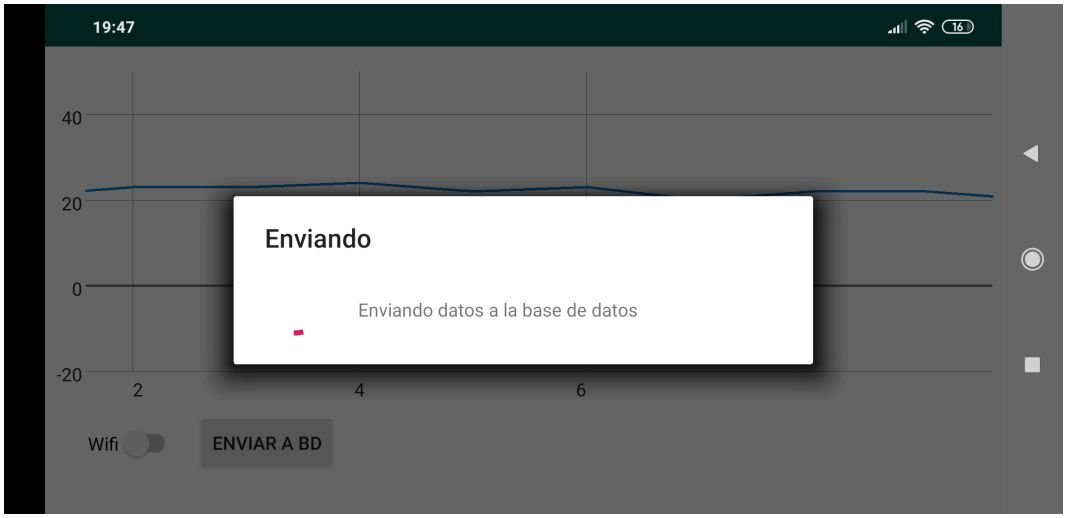


Figura 6.5: Envío Base de Datos App

6.4. Software de la aplicación

En este apartado se va a explicar el software de la aplicación desarrollado así como la base de datos creada.

6.4.1. Software APP

Se ha desarrollado una aplicación móvil sencilla y escalable, ya que se puede ampliar a cualquier tipo de datos. Este software funciona de la siguiente manera:

En el main crearemos un botón por cada tipo de dato, en este caso se han creado dos, uno para la humedad y otro para la temperatura. Cada botón crea un nuevo objeto JavaHttpUrlConnectionReader. Este objeto ejecuta una solicitud http en un nuevo hilo para evitar el error de Android que se produce cuando se realiza una operación de red en el hilo principal. Cuando se crea el JavaHttpUrlConnectionReader, doHttpUrlConnectionAction se ejecuta con la IP del servidor al que enviamos la solicitud y el tipo de solicitud deseado (Humedad, temperatura, etc.).

Luego, el servidor ejecutará la lógica adecuada en función del tipo de parámetro de solicitud deseado enviado en el método. Básicamente se manda una petición mediante la IP y el tipo de dato al ESP-12F y obtenemos una respuesta. Es la clase Graph.java la que dibujará la gráfica con los datos recibidos. El eje Y dependerá del tipo de dato solicitado. A través de doHttpUrlConnectionAction, recibe el dato requerido y lo imprimirá en una ventana emergente (Toast). La respuesta solicitada será del tipo REA 1 3 1/25, se dividirá este string a en dos: antes del la barra y después de la barra. De esta manera obtenemos el la entrada (REA 1 3 1) que será el tipo de dato solicitado y la salida (25) que será el dato devuelto. Estos dos datos, junto con la fecha y la hora se envían a la clase DataReceived.java. Posteriormente, se irá dibujando sobre la gráfica el dato de salida y se incrementará en uno la posición del eje X.

```
1  while (stopThread== false) {
2
3  try {
4      JavaHttpUrlConnectionReader Connection = new
       JavaHttpUrlConnectionReader();
5      Response = Connection.doHttpUrlConnectionAction(ip,Request);
6      System.out.println(Response);
7      final String[] responseSplitted = Response.split("/");
8      runOnUiThread(new Runnable() {
9          @Override
10         public void run() {
11
12             DataReceived d = new DataReceived(Calendar.getInstance().
       getTime().toString(),responseSplitted[0] , responseSplitted[1]);
13             Data.add(d);
14             series.appendData(new DataPoint(lastX++, Double.valueOf(
       responseSplitted[1])), false, 10);
```

```
15
16              Toast toast = Toast.makeText(getApplicationContext(),"Datos
    recbidos. Valor : "+responseSplitted[1],Toast.LENGTH_SHORT);
17              toast.show();
18          }
19
20
21      });
22
23
24      //Your code goes here
25  } catch (Exception e) {
26      e.printStackTrace();
27  }
28      try {
29
30          Thread.sleep(2000);
31      } catch (InterruptedException e) {
32          // manage error ...
33      }
34
35  }
```

También podemos encontrar en la clase Graph.java el switch que apaga o enciende el WiFi de nuestro smartphone para posteriormente enviar los datos a la base de datos.

De la misma manera tenemos en esta clase el botón que envía los datos a la base de datos. A través de un bucle for se irán pasando los datos a la clase DBSender.java que es la encargada de enviar estos datos a la base de datos. A DBSender se le pasa la dirección URL donde esta alojado db_post.php (que es el encargado de poner una nueva línea en la base de datos), así como la fecha, tipo de dato solcitado y dato recibido. Todos estos datos serán empaquetados en la clase DataPackager.java desde DBSender.java, antes de enviarlos mediante esta última clase. La clase DBConnector.java es la encargada de establecer la conexión con la base de datos.

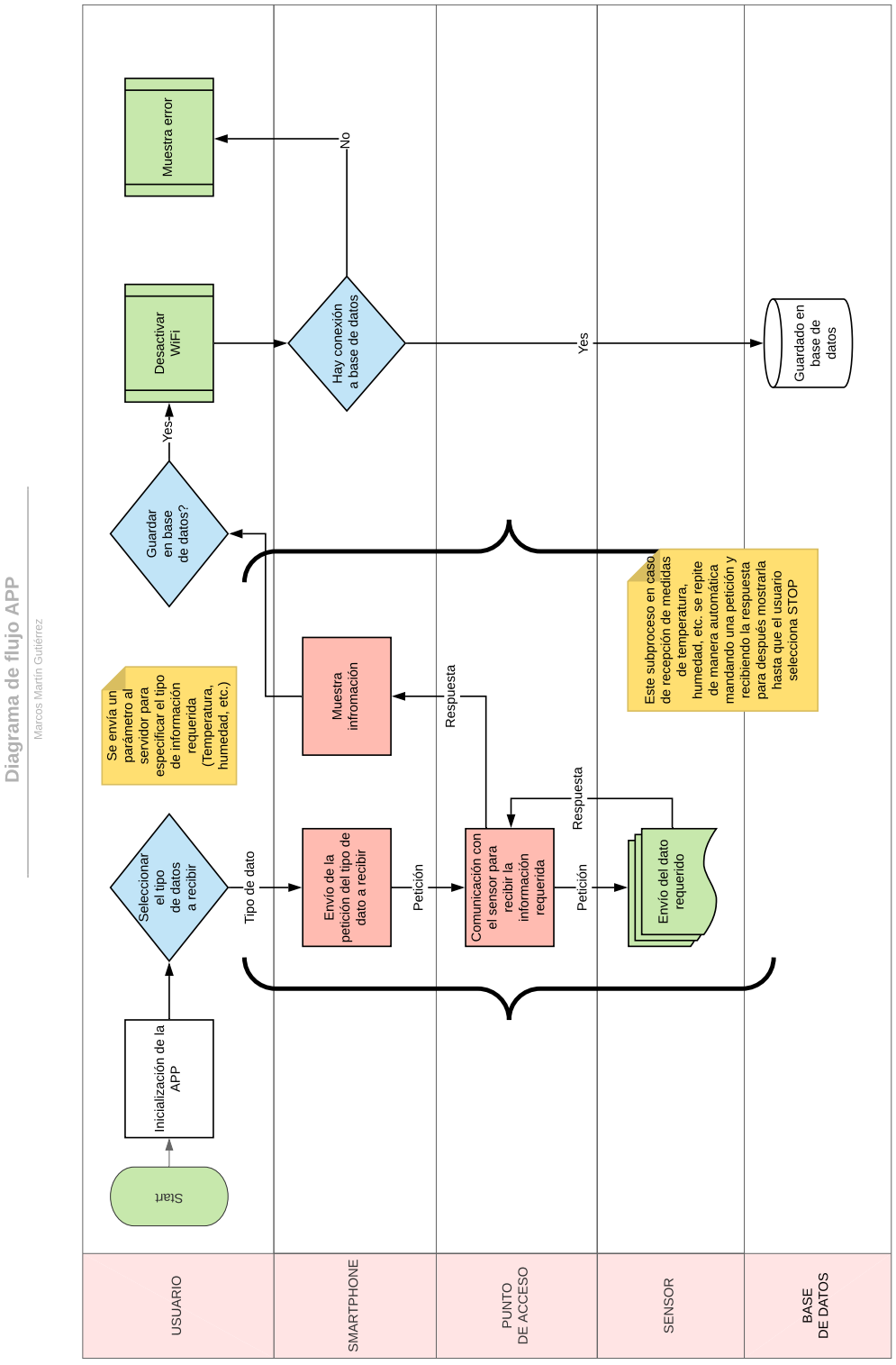


Figura 6.6: Consumo eléctrico

6.4.2. Base de Datos

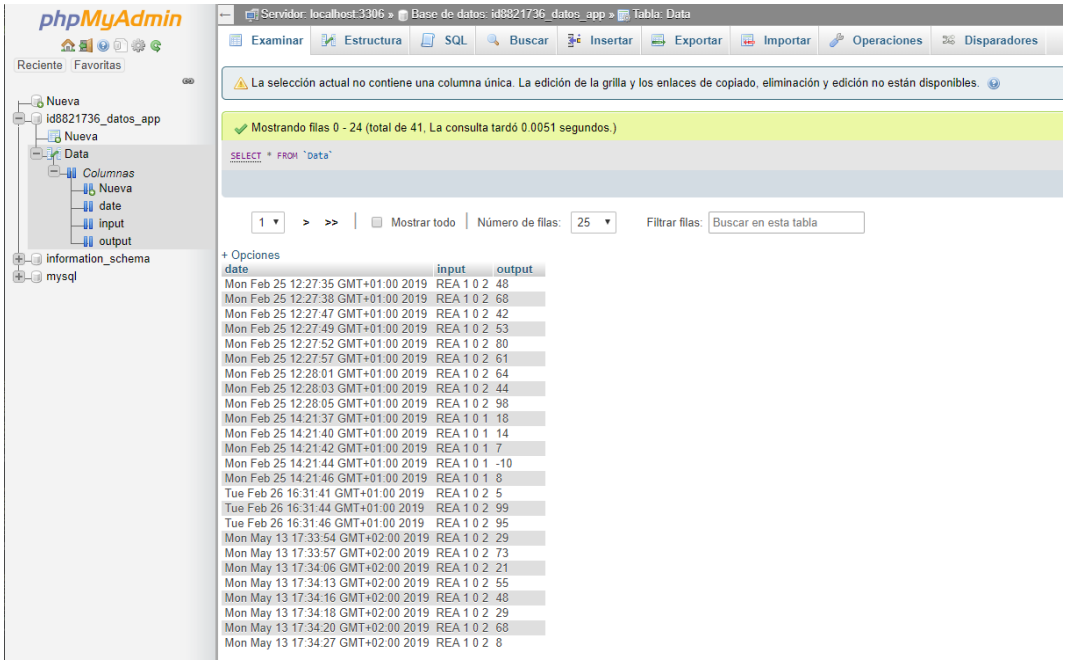


Figura 6.7: Datos en Base de Datos

Hosting Web "000Webhost"

El hosting es un servicio en línea que te permite publicar un sitio o aplicación web en Internet. Cuando te registras en un servicio de hosting, básicamente alquilas un espacio en un servidor donde puedes almacenar todos los archivos y datos necesarios para que tu sitio web funcione correctamente. Para este proyecto se ha utilizado "000Webhost" ya que es uno de los hosting web más grandes de la red, especialmente en su modalidad gratuita. Esta plataforma ofrece a los usuarios 1.5GB de almacenamiento junto con 100 GB de tráfico al mes para alojar sus páginas web de forma totalmente gratuita. También ofrece una modalidad de pago con numerosas características adicionales. Posee un panel de control muy sencillo de utilizar, y cuenta con opciones para actualizar nuestra cuenta de hosting grautito a hosting Premium, con lo cual si tenemos un proyecto que crece mucho en tráfico podemos mejorar la cuenta y obtener un mayor rendimiento de nuestro hosting.

Las características que nos ofrece 000webhost en su Free Hosting son:

- 1.5 GB de almacenamiento.
- 100 GB de Ancho de banda.
- Hosting para 2 sitios web.
- Sin publicidad.

- Soporte de PHP y MySQL.
- Autoinstalador de aplicaciones como WordPress.

phpMyAdmin

phpMyAdmin es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando un navegador web. Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios y exportar datos en varios formatos.

A la hora de tratar con bases de datos, crear, borrar, modificar, ejecutar sentencias SQL, etc... es algo que puede resultar complicado, pero gracias a phpMyAdmin, cualquier usuario con unos pocos conocimientos, es capaz de realizar la tareas más cotidianas con bases de datos SQL.

Hoy en día, la gran mayoría de las páginas web que visitamos diariamente hacen uso de una base de datos, por lo que poder gestionar una base de datos es algo prácticamente imprescindible.

Gracias a herramientas como phpMyAdmin esta gestión se puede desarrollar de una manera visual y muy intuitiva.

El acceso a phpMyAdmin es vía web, esto quiere decir que se aloja en nuestro servidor y podemos acceder desde cualquier dispositivo con conexión a Internet en lugar de usar un único ordenador.

Una ventaja no escrita de phpMyAdmin es que está instalado en la inmensa mayoría de los alojamientos web que puedas encontrar en todo el mundo. Si aprendes a usar phpMyAdmin, aunque sea de manera básica, podrás hacerlo en cualquier hosting que te encuentres.

Cuenta con las siguientes especificacionesn:

- Interfaz Web para la gestión gráfica.
- Administrador de base de datos MySQL, MariaDB y Drizzle.
- Importación de datos desde CSV y SQL.
- Exporta datos a varios formatos: CSV, SQL, XML, PDF (vía la biblioteca TCPDF), ISO/IEC 26300 - OpenDocument Text y Spreadsheet, Word, Excel, LaTeX y otros.
- Administración de múltiples servidores.
- Crea gráficos PDF del diseño de la base de datos.
- Crea consultas complejas usando Query-by-Example (QBE).

- Búsqueda global en una base de datos o un subconjunto de esta.
- Live charts para monitoriar las actividades del servidor MySQL tales como conexiones, procesos, uso de CPU y/o memoria, etc.

InnoDB

El mecanismo de almacenamiento de datos utilizado ha sido InnoDB. Este mecanismo es de código abierto para la base de datos MySQL, incluido como formato de tabla estándar en todas las distribuciones de MySQL AB a partir de las versiones 4.0. Su característica principal es que soporta transacciones de tipo ACID y bloqueo de registros e integridad referencial. InnoDB ofrece una fiabilidad y consistencia muy superior a MyISAM, la anterior tecnología de tablas de MySQL, si bien el mejor rendimiento de uno u otro formato dependerá de la aplicación específica.

Se han generado dos documentos PHP, *db_post.php* y *db_print.php*. El primero sirve para introducir datos en la base de datos, es decir, agrega una línea a la base de datos. Por otro lado, *db_print.php* es para imprimir la base de datos.

En la Figura 6.7 se aprecia como se han recibido los datos desde la aplicación móvil y se han almacenado en la base de datos. Esta base de datos consta de tres columnas: date, input y output.

- date: consiste en la fecha y la hora en la que se toma el dato del sensor. Esta fecha y hora es en la que se ha representado el dato en la gráfica de la aplicación Android.
- input: es el dato solicitado por el teléfono móvil al sensor. Este dato está en el formato de la trama del protocolo del sensor.
- output: es el dato concreto devuelto por el sensor, que es el que ha sido representado en la gráfica de la aplicación móvil.

A esta base de datos se puede acceder desde cualquier dispositivo con acceso a Internet. Esto permite una gran flexibilidad para este proyecto, ya que se puede recoger los datos mediante un smartphone y acceder a estos datos desde cualquier lugar.

Capítulo 7

Desarrollo simulador

7.1. Introducción

El alto coste del sensor FireStingGO2 supuso una dificultad para disponer del sensor para la realización de pruebas. Es por ello que, conocedores del protocolo interno del sensor, se optó por desarrollar un simulador de este sensor, es decir, un simulador del protocolo de comunicación de este sensor. Para ello, mi tutor del Trabajo Fin de Grado me facilitó un placa educativa LPC2103 que anteriormente ya habíamos utilizado en la asignatura de Interconexión de Sistemas Electrónicos. Mediante esta placa se consiguió simular la respuesta del sensor ante un trama recibida. Es decir, se consiguió que el LPC2103 simulara el comportamiento del sensor a la hora de recibir y transmitir datos. Para ello hubo que crear un firmware determinado, así como el desarrollo de un makefile para grabar el firmware en la memoria flash. Con ello se conseguía que interconectando nuestra placa con la placa LPC2103 vía USB, realizar pruebas enviando y recibiendo datos, para así poder desarrollar el firmware de este Trabajo Fin de Grado de manera correcta. Algunas características de LPC2103 Education Board son:

- Microcontrolador NXP ARM7TDMI LPC2103 con memoria Flash de 32 KByte y SRAM de 8 KByte.
- Cristal de 14.7456 MHz.
- Interfaz de puente UART a USB en UART0.
- 2 Kbit I2C E2PROM.
- 7-segment LED display.
- Reset button.
- Pulsador en P0.14 (entrada de interrupción).
- Entrada analógica a través de un potenciómetro trimmer.
- Conector de expansión de 20 posiciones.

- Genera +3.3V y 1.95V para alimentar el LPC2103.
- +3.3V disponible para circuitos externos, hasta 300 mA.
- Alimentado directamente desde el conector USB mini-B.
- Descarga de programas simple y automática (ISP) a través del canal puente UART-to-serial. Circuito que controla automáticamente el gestor de arranque desde el canal del puente UART-to-serial.
- PCB de cuatro capas (material FR-4) para una mejor inmunidad al ruido

En las secciones siguientes se detallan tanto el desarrollo hardware como firmware.

7.2. Desarrollo hardware LPC2103

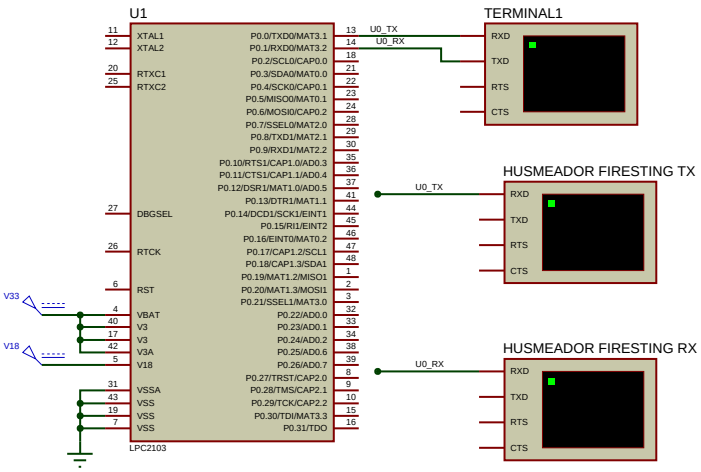


Figura 7.1: Esquemático simulador de sensor

Como ya se disponía de la placa LPC2103, se desarrolló un esquemático en ISIS (Proteus 7) junto con el código en VSM Studio con la finalidad de simular el comportamiento del sensor para realizar emulaciones sin necesidad de cargar el programa a la placa. Este esquemático se utilizó única y exclusivamente para emular el comportamiento del LPC2103 y así depurar el código antes de cargarlo a la propia placa. El desarrollo del esquemático para la simulación fue mínimo ya que se partió del ejemplo de la práctica 0 de la asignatura de Interconexión de Sistemas Electrónicos. El esquemático está formado por el LPC2103 de las propias librerías de Proteus, a este se le alimenta con las tensiones correspondientes y se le añade un terminal de entrada salida en la UART0. Mediante este terminal simularemos el comportamiento de entrada/salida del LPC2103, es decir, introduciremos el dato que debe recibir el LPC2103 y veremos la respuesta de este. Por

otro lado, se han añadido dos husmeadores, uno para TXD0 y otro para RXD0. Con ellos veremos en formato hexadecimal que transmitimos y que recibimos en todo momento.

7.3. Desarrollo firmware LPC2103

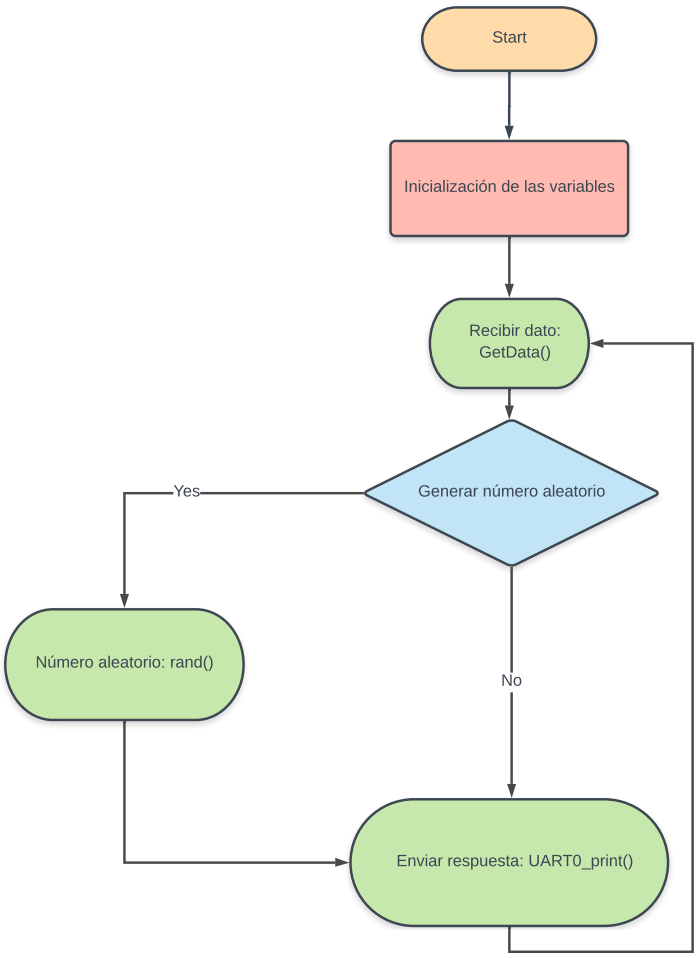


Figura 7.2: Diagrama de flujo del simulador de sensor

Para el desarrollo del código que simula el comportamiento interno del sensor se ha partido del código de la Práctica 0 de la asignatura de Interconexión de Sistemas Electrónicos. Primero se inicializa la UART0 de manera correcta para establecer la comunicación, ya que es la que se va a utilizar para establecer la comunicación con nuestro dispositivo. Y posteriormente se han reciclado las funciones para leer y enviar datos a través de UART0. Dentro del main() se ha introducido todas las posibles tramas del protocolo del sensor LPC2103, mediante condicionales. Para algunos casos, como pueden ser datos concretos de oxígeno, temperatura, humedad, presión, etc. se han generado números aleatorios consecuentes con la medida en cuestión. Una vez que ya se ha desarrollado el código y

se ha depurado de manera emulada ejecutándolo mediante Proteus, se procede a cargar este código en la memoria Flash del LPC2103, para que al conectarlo a nuestro proyecto simule el comportamiento del sensor programado anteriormente. Para ello se ha adaptado un Makefile ya que es necesario cargar el programa en la memoria Flash de la placa.

Capítulo 8

Fabricación carcasa 3D

8.1. Introducción

Debido a la disponibilidad en el laboratorio de proyectos de una impresora 3D se creyó conveniente desarrollar una carcasa 3D para nuestro proyecto. Esto supuso algunas ventajas como son dotar de protección a la placa, dar al proyecto un aspecto visual más técnico y el aprendizaje por mi parte al desarrollo de elementos 3D para su impresión por medio de la aplicación web Tinkercad.

La impresión 3D es un grupo de tecnologías de fabricación por adición donde un objeto tridimensional es creado mediante la superposición de capas sucesivas de material. Las impresoras 3D ofrecen a los desarrolladores del producto la capacidad para imprimir partes y montajes hechos de diferentes materiales con diferentes propiedades físicas y mecánicas, a menudo con un simple proceso de ensamble.

Es una tendencia actualmente debido a las infinitas aplicaciones que posee, ya que nos permite crear cualquier elemento físico que imaginemos. Podemos encontrar implementaciones de la tecnología de impresión 3D en ámbitos como el automotor, en la construcción de prototipos, lo que significa un ahorro importante debido a la reducción en los tiempos de modelado y en materiales, en la industria de la joyería, el calzado, el diseño industrial. También en la ingeniería y el sector aeroespacial las impresoras 3D encuentran un lugar en la elaboración de piezas, ya que también podrían usarse para la elaboración de elementos en el espacio.

Una impresora 3D dispositivo que es capaz de generar cuerpos físicos sólidos tridimensionales mediante la adición capa a capa de un material, generalmente plástico ABS, pero no es el único material que se utiliza. Es decir que permite crear, sin la necesidad de utilizar cualquier tipo de molde, un objeto que luego podremos tener en nuestras manos.

La más importante diferencia que podemos encontrar entre las formas tradicionales de construcción de objetos y la impresión 3D es que mientras en el primer método el objeto a modelar se obtiene quitando el material sobrante, en la impresión 3D sólo se utiliza estrictamente el material a utilizar, lo que produce importantes ahorros, además de menor contaminación.

8.2. Herramienta CAD

Para realizar una impresión 3D, primero necesitaremos un archivo creado con algún software de modelado 3D, en nuestro caso se utilizó la aplicación web Tinkercad. Con este archivo comunicamos a la impresora 3D qué es lo que debe modelar. Este archivo generado contienen precisas instrucciones acerca de las coordenadas que se deben seguir para crear el objeto y generalmente son introducidos a la impresora a través de USB mediante un pendrive o memoria, pero también existen modelos de impresoras 3D que permiten su conexión directa a la PC o vía WiFi.

Tinkercad es un software gratuito online creado por la empresa Autodesk, una de las empresas punteras en el software de diseño 3D de la mano de su programa estrella para tal fin, Inventor. Se optó por esta aplicación por ser una primera inmersión en el mundo del diseño 3D de una manera sencilla y atractiva, ya que la interfaz de trabajo es simple y muy atractiva inicialmente, si bien una vez dominados los conceptos básicos carece de herramientas para llegar a diseños complejos. Sus ventajas son claras: es sencillo de usar, intuitiva ,su aspecto es atractivo y con unas pocas horas de entrenamiento podemos adquirir mucha destreza en su uso. Para la realización de nuestra carcasa solo se ha requerido paciencia, dedicación e imaginación.

8.3. Diseño 3D

El diseño 3D se ha realizado por partes, utilizando diferentes figuras geométricas para adaptar nuestra carcasa al diseño ideado. Para el realizar un diseño adecuado de la carcasa, también se recreo en 3D la batería y la placa, lo cual permitió realizar una carcasa con las medidas exactas requeridas. En las siguientes imágenes se aprecia cada una de las partes utilizadas así como el montaje completo:

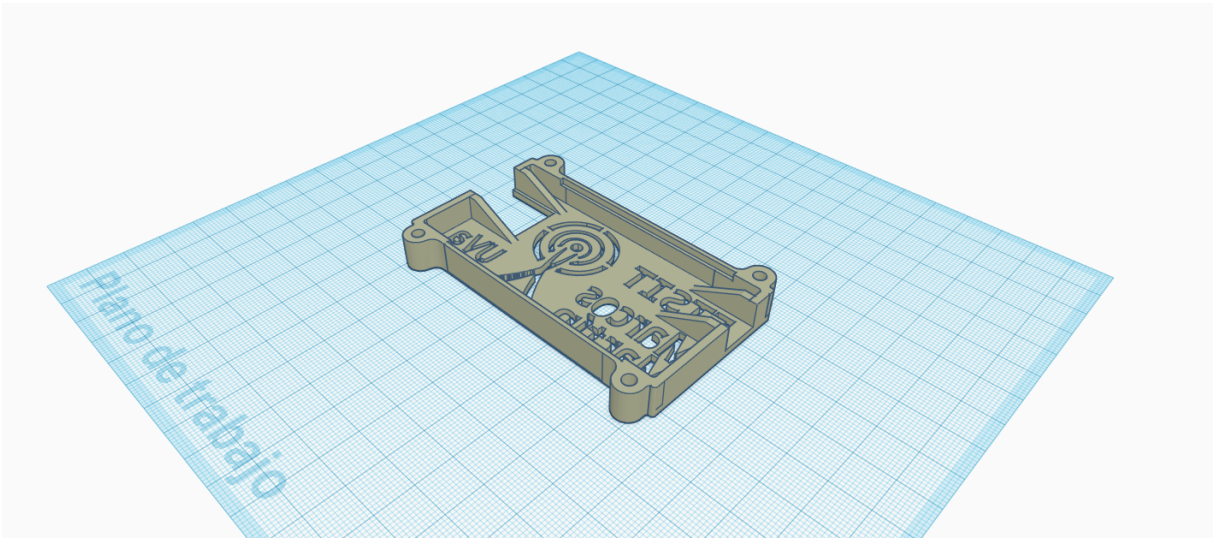


Figura 8.1: Carcasa superior 3D

Figura 8.1.: Como se puede apreciar en la imagen, la carcasa superior consta del símbolo de la escuela, así como el nombre de la universidad, nombre de la escuela y mi nombre. También posee dos aberturas en los laterales, esto es porque ahí será donde irán encajados los puertos USB. Los agujeros en las esquinas sirven para atornillarla a la carcasa inferior. La carcasa superior se diseña al revés para que a la hora de su impresión tenga un base más sólida.

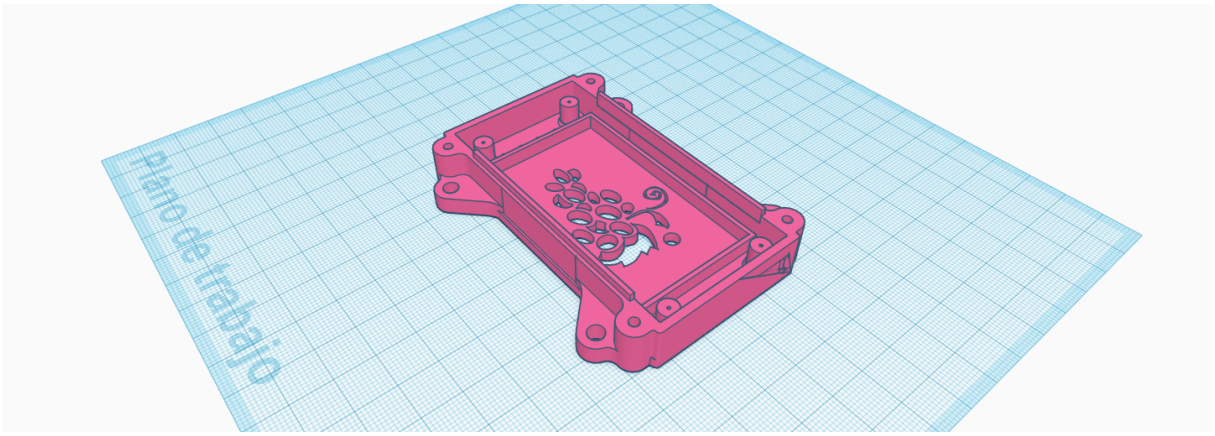


Figura 8.2: Carcasa inferior 3D

Figura 8.2.: La carcasa inferior tiene un racimo de uvas para mostrar que esté prototipo está orientado al sector vinícola. También posee un recinto rectangular que será donde irá encajada la batería. Por otro lado, posee cuatro columnas con agujeros, esto es para sustentar y atornillar nuestra placa. Los agujeros de las esquinas exteriores se utilizarán para atornillar ambas partes.

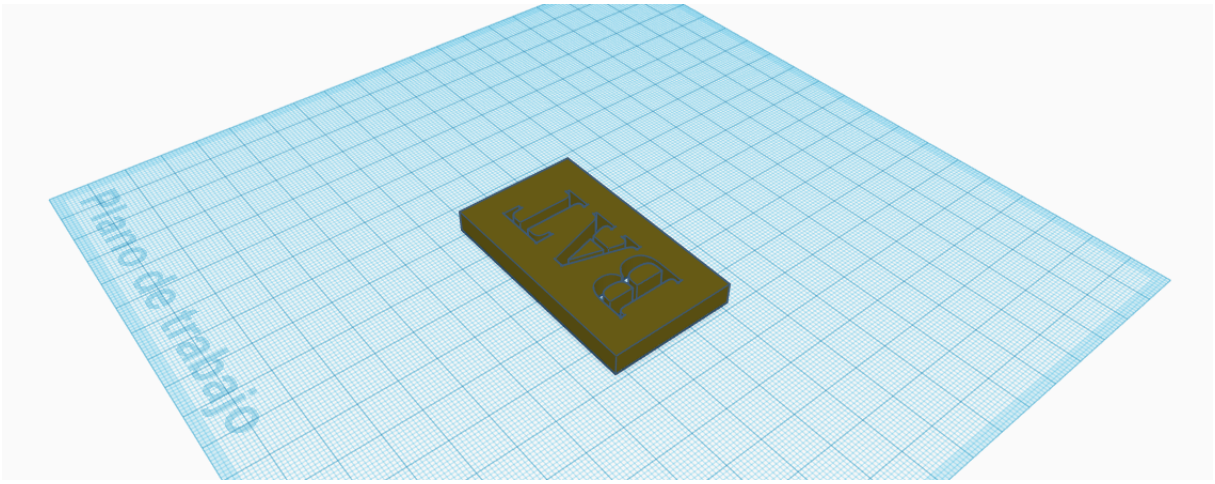


Figura 8.3: Batería 3D

Figura 8.3.: Esta batería 3D se usó para realizar pruebas. Tiene las dimensiones reales de la batería utilizada para este proyecto. Esta batería irá en la carcasa interior, dentro de un recinto diseñado especialmente para mejorar su sujeción.

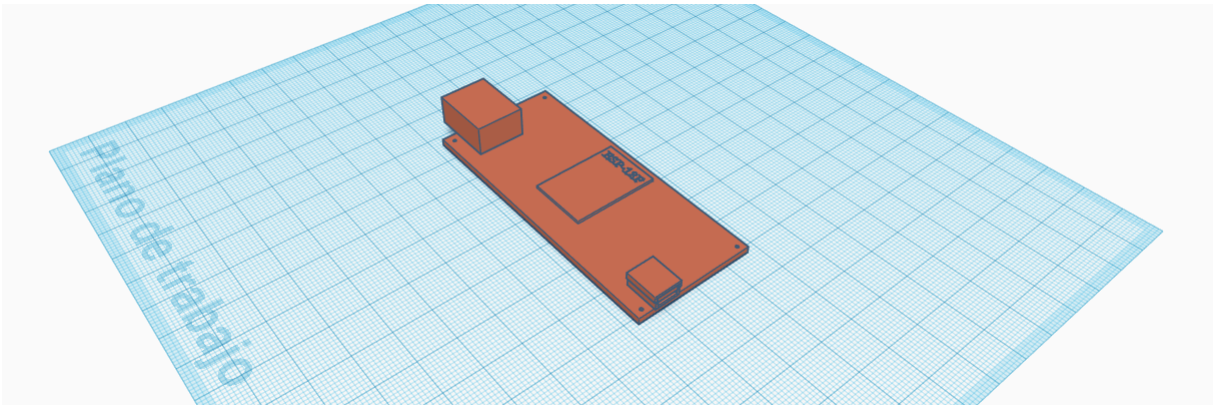


Figura 8.4: Placa 3D

Figura 8.4.: Al igual que la Batería 3D, se creó esta placa 3D con las medidas exactas de la placa del proyecto. Se utilizó para comprobar como encajaba en la carcasa. Esta placa irá colocada encima de la batería.

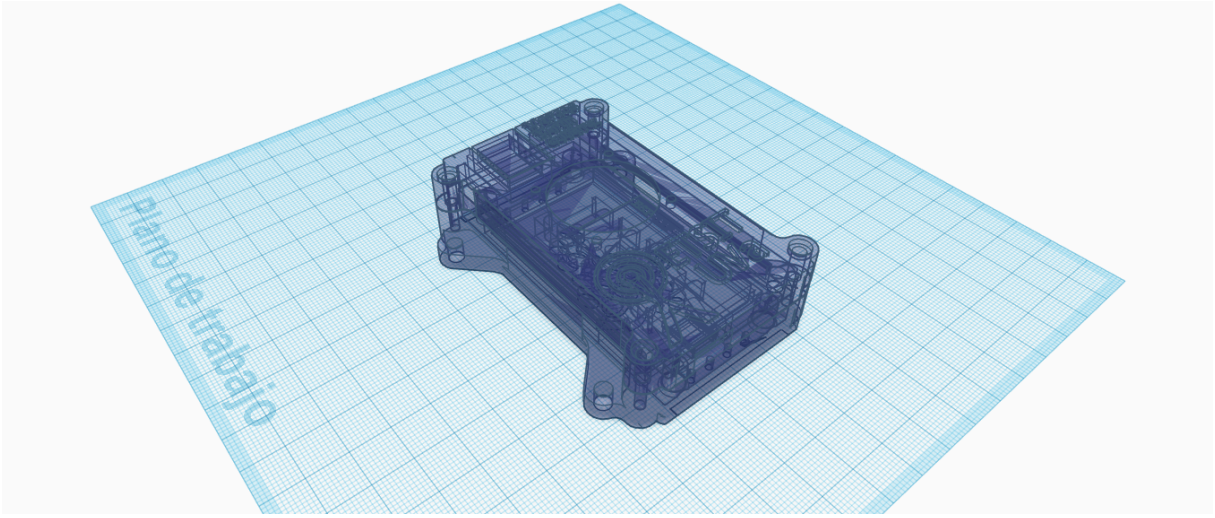


Figura 8.5: Diseño completo 3D

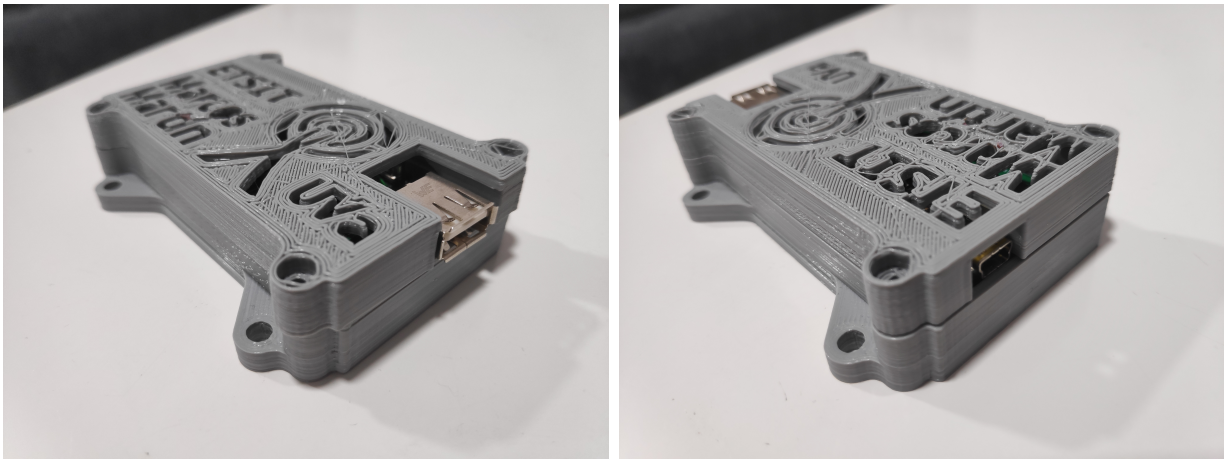
Figura 8.5.: Este es el montaje completo en modo transparente para que se aprecie de manera genérica cada parte interna del desarrollo de una figura 3D.

8.4. Montaje final carcasa 3D

Una vez realizado el diseño de la carcasa, a través de la propia aplicación Tinkercad se generará un archivo que introduciremos en la impresora 3D. Este fichero tiene un formato específico (STL) que con el software de la impresora 3D se convertirá en "gcode" y se imprimirá. El resultado final es el siguiente:

(a) Carcasa real exterior (b) Carcasa real interior

Figura 8.6: Huellas diseñadas



(a) Carcasa real con placa lado USB Host (b) Carcasa real con placa lado mini-USB

Figura 8.7: Huellas diseñadas

Parte III

Finalización

Capítulo 9

Montaje final

LLegado a este punto, ya disponemos de todos los componentes hardware y software. Por ello, realizamos el montaje completo para comprobar que se cumplen las especificaciones iniciales. El montaje consta del simulador conectado a nuestro prototipo y al mismo tiempo un smartphone con nuestra aplicación Android solicitando un tipo datos y recibiendo el dato concreto, mostrándolo en una gráfica.

En las siguientes imágenes podemos ver el montaje completo del sistema. Se puede apreciar el pequeño tamaño del sistema electrónico con respecto al smartphone o la batería. Las dimensiones de la batería son similares a las del proyecto, esto está premeditado para su correcto posicionamiento en la carcasa. También vemos el smarphone con el menú inicial de la aplicación Android creada, así como la placa LPC2103 y como se interconecta con nuestro sistema. Este sería, en definitiva, el montaje completo. En la Figura 9.3 vemos el montaje del sistema electrónico final dentro de la carcasa creada.

Figura 9.1: Montaje completo 1

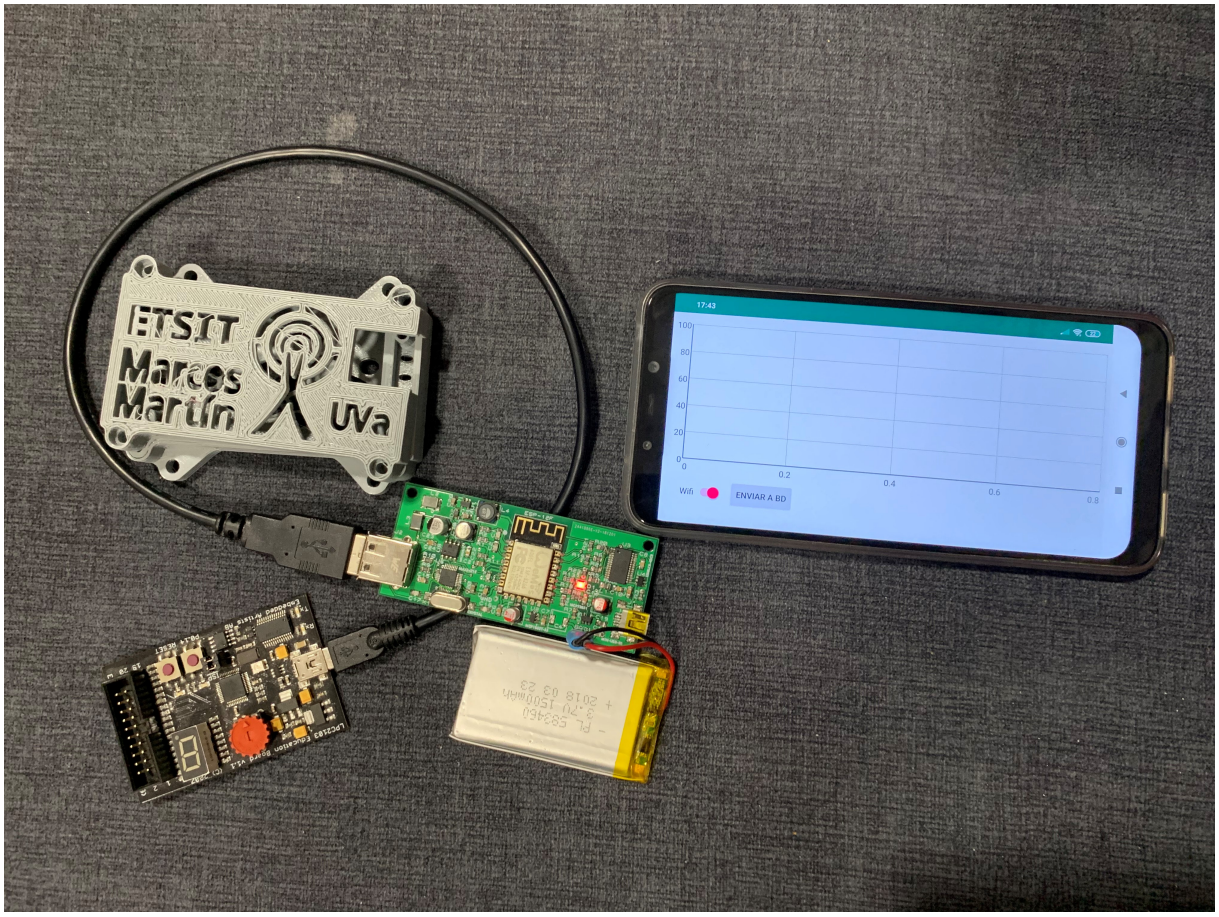


Figura 9.2: Montaje completo 2

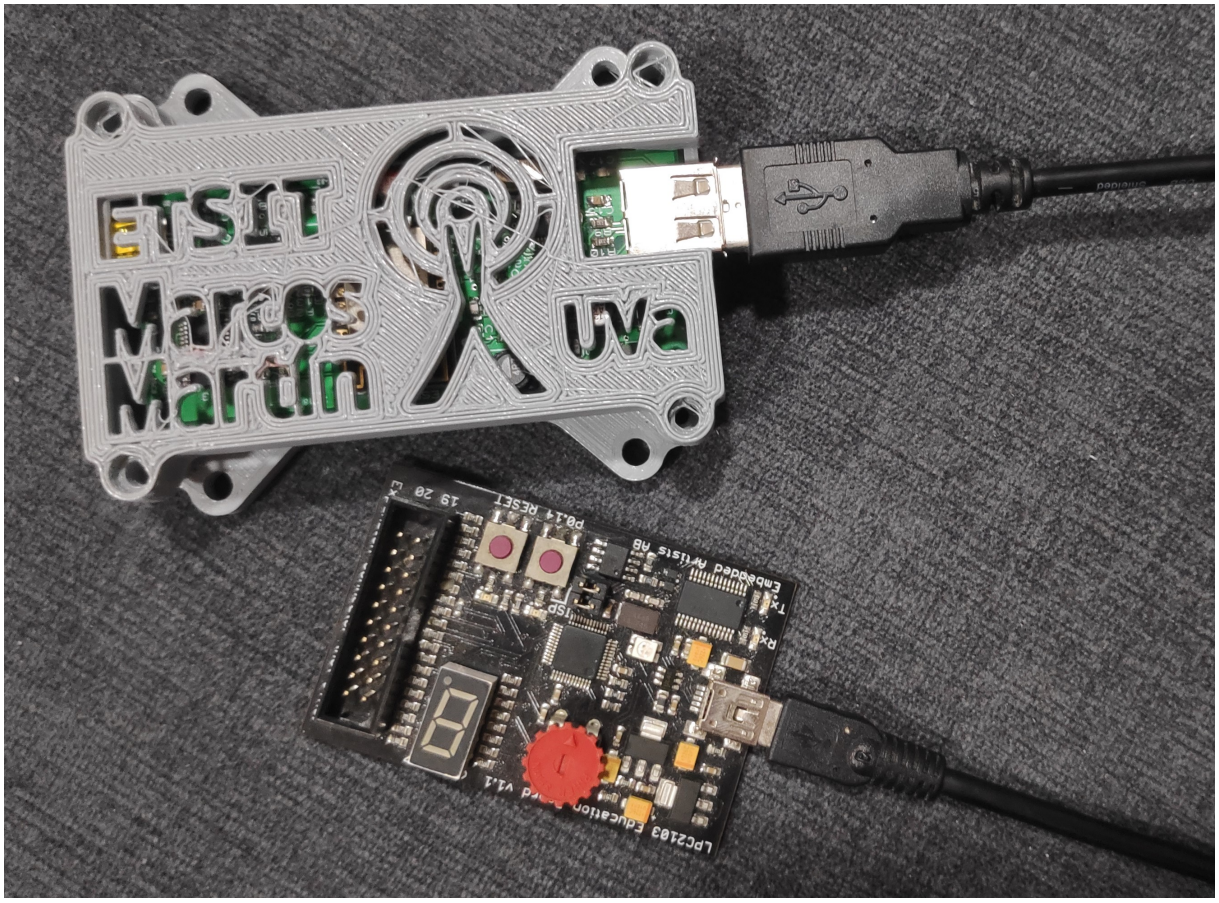


Figura 9.3: Montaje completo 3

Capítulo 10

Limitaciones encontradas

La principal limitación encontrada es la disponibilidad del sensor, ya que ha supuesto realizar un simulador del prototipo, lo cual requería tiempo. Además de no poder probarlo en el sensor real y realizar las pruebas pertinentes. Por contra, desarrollar un simulador supuso un mayor entendimiento del protocolo del sensor, lo cual facilitó la programación de la solicitud de datos.

Otra limitación encontrada es la disponibilidad de librerías para el USB Host (MAX3421E). Se usaron unas librerías "Open Source" creadas por Oleg Mazurov disponibles GitHub. Estas librerías están pensadas para Arduino, y aunque tienen una adaptación para ESP8266, ha sido necesario una adaptación por nuestra parte.

Por último, destacar que todas las limitaciones encontradas no han sido definitivas y han servido para aprender y mejorar el producto, por lo tanto ha sido algo positivo.

Capítulo 11

Estado del arte

Actualmente existen multitud de dispositivos registradores de datos encargados de registrar medidas de diferentes tipos de sensores, como por ejemplo medidas de temperatura y de humedad, y mostrarlas en una pantalla. Los dispositivos registradores de datos se implementan normalmente en un sistema electrónico formado por uno o varios sensores, una memoria donde guardar los datos y, opcionalmente, una pantalla de visualización de los mismos. Como existe una gran variedad de dispositivos registradores, los sensores varían según el tipo de datos a registrar.

Los dispositivos registradores de datos convencionales presentan varios inconvenientes. En primer lugar, los datos registrados están disponibles solamente en los propios registradores (en su memoria), teniendo que visualizar los datos en ellos a través de una pantalla integrada o recoger los datos mediante un ordenador. Además, como los datos se guardan en una memoria interna del dispositivo, la cantidad de datos a guardar está limitada al tamaño de dicha memoria. Como consecuencia del uso de una memoria limitada, para liberar esa memoria es necesario acceder al sistema y recoger los datos, almacenándolos en otro dispositivo. Cada vez que se necesite tomar datos del sensor y manipularlos hay que acceder físicamente, mediante un cable especifico, al sistema.

Otra desventaja de los registradores de datos actuales es que están sujetos al tipo de sensor de entrada, con lo que el registrador de datos se fabrica de una manera determinada dependiendo del tipo de datos a registrar. Por ejemplo, los registradores de datos de temperatura están formados por un sensor de temperatura conectado de una manera determinada al sistema electrónico que registra y guarda los datos. Por tanto, el sistema electrónico del registrador de datos se debe diseñar en función del sensor a utilizar.

Productos competidores

Existen multitud de registradores de datos en el mercado y estos toman diferentes tipos de datos. Debido a las características especiales del dispositivo aquí descrito se procede a realizar una clasificación de los diferentes competidores dependiendo de diferentes características.

Principales competidores con entrada USB: Existen muy pocos en el mercado ya que la mayoría de los registradores de datos viene junto con el sensor del que toman los

datos.

- USB PT-104: Registrador de datos con resistencia de platino PT-104 con interfaz USB y Ethernet. El PT-104 es un registrador de datos para medición de temperatura con 4 canales. Mide temperatura, resistencia y tensión. Opera con sensores PT100 y PT1000. Precio 559 euros.

  Este registrador de datos no mide el nivel de oxígeno, aunque si mide la temperatura. Tampoco posee conexión WiFi y almacena los datos en una memoria propia.

- Registrador de datos HOBO UX120-006M: registrador de datos para sensores externos / 4 canales externos para un gran número de sensores / amplia memoria para 1,9 millones de valores de / gran pantalla LCD / Interfaz USB / resolución de 16 Bits / vida de las pilas de 1 año / software opcional. Precio: 139 euros.

  Se aprecia en la descripción que tiene cuatro entradas USB para sensores externos, pero por el contrario guarda los valores tomados en una memoria propia. No posee conexión WiFi.

Principales competidores con sensor de oxigeno: No existen multitud de ellos en el mercado ya que la medida de oxigeno tiene aplicaciones muy específicas, es decir, la medida de oxigeno del medio está orientado a sectores determinados, como por ejemplo el sector vinícola.

- U26-001 Registrador de Datos HOBO para Oxígeno Disuelto: Mide las concentraciones de oxígeno en lagos, arroyos, ríos, estuarios y aguas costeras. Este registrador de datos preciso se recomienda para proyectos de investigación en biología acuática e hidrología. El HOBO U26 utiliza la tecnología óptica del sensor de DO RDO® Basic (Oxígeno disuelto resistente) y es fácil de mantener. Precio 1250 euros.

  Este registrador de datos incluye el sensor en él, es decir, es un sensor registrador de datos. Esto hace que, a diferencia del propuesto en este estudio, el U26-001 este atado a un sensor concreto. Por otro lado, para recoger los datos en el U26-001 es necesario acceder físicamente a él mediante un ordenador y un software propio para recoger los datos.

- SDL150: Medidor de Oxígeno Disuelto/Registrador de datos: Registra datos en una tarjeta SD y en formato Excel. Mide el oxígeno disuelto. Almacena manualmente 99 lecturas y lecturas de 20 M con la tarjeta de memoria SD de 2 G.

  Presenta las mismas desventajas que el registrador de datos U26-001 respecto al dispositivo propuesto. Almacena los datos en una memoria interna, no se conecta vía WiFi, lo que hace que haya que conectarse físicamente a él para extraer los datos.

- Registrador de datos PCE-PHD 1: El registrador de datos PCE-PHD 1 es un aparato de múltiples capacidades para la inspección de la calidad del agua. El registrador

de datos portátil sirve para el control en el agua de los valores del pH, conductividad, oxígeno y es así también muy apropiado para la medición de la temperatura. Posee un software propio con un cable de datos RS-232. Todos los valores pueden ser almacenados directamente en una tarjeta SD (hasta 16 GB) en formato Excel, o ser traspasados mediante la interfaz RS-232 en tiempo real a un PC.

En este también presenta las mismas desventajas que los dos anteriores con respecto al registrador de datos aquí propuesto.

Principales competidores con salida de datos vía WiFi: Existen multitud de ellos en el mercado, la mayoría orientado a sensores de temperatura y humedad.

- Data logger Wi-Fi de temperatura testo Saveris 2-T1: Las temperaturas medidas con el sensor integrado se registran también de forma fiable por períodos prolongados y enviadas directamente al testo Cloud (almacenamiento de datos en línea) a través de su WLAN. Navegando en Internet desde su smartphone, tablet o PC, usted puede acceder a dichas lecturas en todo momento y desde cualquier parte. Precio 120 euros.

  Las principales diferencias de este registrador de datos es que no posee entrada USB y solo mide temperatura.

- EL-WIFI-TH - Registrador de Datos, WiFi Temperatura y Humedad: El sensor EL-WiFi-TH mide la temperatura y la humedad del entorno en el que está situado. Los datos se transmiten de forma inalámbrica mediante una red WiFi a un PC y se visualizan mediante un paquete de software gratuito. Durante la configuración, el sensor busca redes inalámbricas disponibles mientras se conecta físicamente al PC. Precio 123 euros.

  Al igual que el caso anterior este registrador de datos no posee entrada USB, no es independiente del sensor y no mide niveles de oxígeno.

Como se puede comprobar no existe en el mercado un registrador de datos orientado al sensor de oxigeno que cumpla los tres requisitos anteriores conjuntamente: entrada USB a la que se conecta un sensor cualquiera, envío de datos vía WiFi evitando de esta manera la interconexión física para la recogida de los datos y por último que tome datos de oxígeno, temperatura, etc.

Por lo general, los registradores de datos están unidos al sensor del que toman los datos y forman uno entre ambos. Esta es otra de las desventajas, ya que el registrador de datos no puede actuar de forma independiente e interactuar con otros sensores. Actualmente no existen registradores de datos de oxigeno con entrada USB y envío de datos vía WiFi.

Este dispositivo electrónico viene a sustituir los registradores de datos convencionales sin WiFi. Cualquier empresa que utilice sensores utiliza por lo tanto registradores de datos, estos registradores de datos convencionales tienen el perjuicio de no estar conectados a

una red WiFi y por lo tanto no poder usarse en remoto. Debido a esto, el tiempo de producción se ve incrementado respecto al dispositivo planteado ya que este dispositivo adquiere los datos de manera instantánea y desde una distancia razonable. Se elimina la necesidad de acceder físicamente a los registradores de datos, con todo lo que esto conlleva. Cualquier industria actual invierte en tecnología para mejorar la eficiencia a la hora de la producción, intentando reducir tiempos e intentando aumentar la producción por hora. Por ello, el dispositivo planteado es una herramienta muy útil y necesaria, reduciendo tiempos de toma de datos y realizando una toma de datos constante e instantánea.

Se adquiere la capacidad de tener los datos en tiempo real, lo que hace que el análisis de estos datos sea más exacto, exhaustivo y preciso. Al tomar los datos de manera constante y sin necesidad de acceder el dispositivo, se pasa de analizar los datos pasados a analizar los datos presentes. Esto ayuda a mejorar la calidad de la producción.

Otros sistemas que puede sustituir son los registradores de datos asociados a sensores concretos. Al conectarse el sensor vía USB y ser un dispositivo reprogramable, este registrador de datos puede adaptarse a cualquier ámbito. Con ello se consigue que, para una misma industria con diferentes tipos de sensores, utilizar el mismo registrador de datos. Esto ayuda a la persona que analice los datos a ser más eficiente, ya que, aprendiendo a utilizar un registrador de datos, puedes manejar la toma de datos de todos los sensores. Por otro lado, se reducen los costes debido a la compra de registradores de datos diferentes, sustituyéndose por registradores de datos de ámbito general.

Capítulo 12

Presupuesto económico

Para realizar un presupuesto económico se ha utilizado la aplicación Ipscore de la OEPM, esta realiza una estimación de valor de una patente y genera el valor actual neto esperado (VAN). Además, esta aplicación genera una serie de gráficas con información comercial. IPscore es un programa informático gratuito de la OEPM para la valoración de patentes que puede ayudar a los proveedores de servicios de PI, como por ejemplo, pequeñas y medianas empresas, a evaluar patrones y carteras de patentes.

IPscore es:

- Una herramienta basada en Microsoft Access para la evaluación y gestión de patentes.
- Una guía para identificar posibles ventajas y oportunidades para reducir costes.

IPscore puede usarse para evaluar:

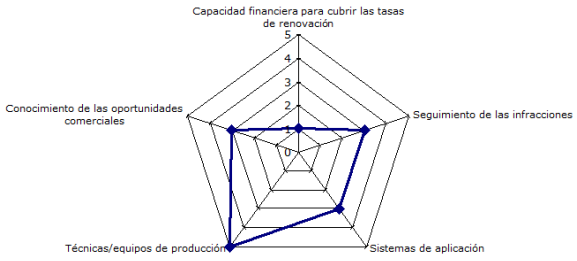
- Patentes concretas y tecnologías patentadas.
- Proyectos de I+D, incluso cuando aún no exista una patente.
- Ideas y propuestas de proyectos.

IPscore se compone de:

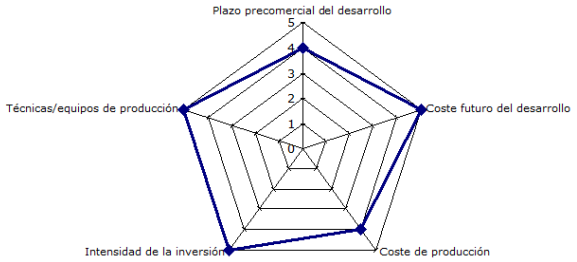
- 40 factores de valoración.
- Escalas predefinidas que guían las respuestas.
- Una estimación rápida del valor monetario de una patente (valor actual neto).
- Informes sobre una única patente o sobre toda una cartera de patentes.
- Una base de datos que ofrece actualizaciones de documentación y evaluación.

En primer lugar deberemos responder una serie de cuestiones relacionadas con nuestro prototipo y posteriormente se generará el informe complementario. Este informe esta dividido en cuatro partes:

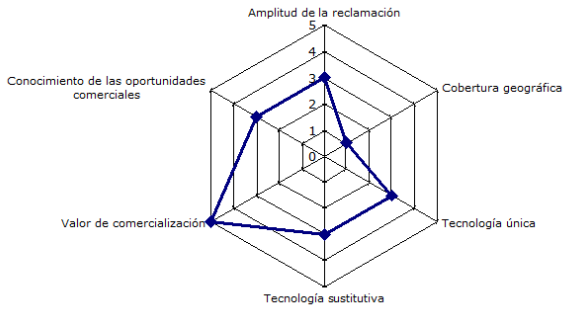
- Competencia organizativa (Figura 12.1 (a)): es el potencial que tiene una organización para actuar y cambiar en busca de ventajas competitivas.
- Factores de coste (Figura 12.1 (b)): valores del consumo de los recursos que han sido necesarios para poder producir productos o prestar servicios.
- Oportunidades (Figura 12.1 (c)): engloba todos los factores positivos que se crean en el entorno y que una vez identificados pueden aprovecharse para mejorar la situación de la compañía.
- Situación competitiva (Figura 12.1 (d)): Son características de una empresa que la diferencia de otras colocándole en una posición relativa superior para competir. Es decir, cualquier atributo que la haga más competitiva que las demás.



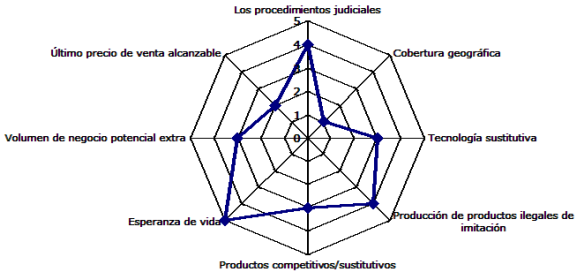
(a) Competencia organizativa



(b) Factores de coste



(c) Oportunidades



(d) Situación competitiva

Figura 12.1: Informe complementario

El valor actual neto (VAN) es un criterio de inversión que consiste en actualizar los cobros y pagos de un proyecto o inversión para conocer cuanto se va a ganar o perder con esa inversión.

Net present value for Registrador de datos

The net present value of the patented technology is: **131**

(Discount factor = 10%)

Calculations for net present value are based on the following assumptions:

Factor		Score		Financial assumptions
B5	Plazo precomercial del desarrollo	4	½ año [0,5]	0,5
C2	Tasa de crecimiento del mercado	5	Muy alto (15%) [0,15]	0,15
C3	Esperanza de vida	5	8 años [8]	8
C6	Volumen de negocio potencial extra	3	Medio (4%) [0,04]	0,04
D1	Mantenimiento del rendimiento del negocio	3	50% [0,5]	0,5
D2	Coste futuro del desarrollo	5	Bajo (½%) [0,005]	0,005
D3	Coste de producción	4	Descenso del 15% debido al uso de la tecnología patentada [0,85]	0,85
D4	Intensidad de la inversión	5	50% de la actual intensidad de la inversión [0,5]	0,5

Financial results from the company accounts	
Business turnover	150
Direct costs	36
Indirect costs	80
Provision for depreciation	20
Net result	14
Depreciation period (yrs)	1
Definition of business area	
Share of current company turnover	30%
Parameters used in the calculations	
Discount factor	10,0%
Total growth in general company market	4,0%

Figura 12.2: Valor Actual Neto (VAN)

En la siguiente gráfica se muestra el valor actual neto frente la tasa de descuento, es decir, el coste de capital que se aplica para determinar el valor actual de un pago. La tasa de descuento se utiliza para descontar el dinero futuro.

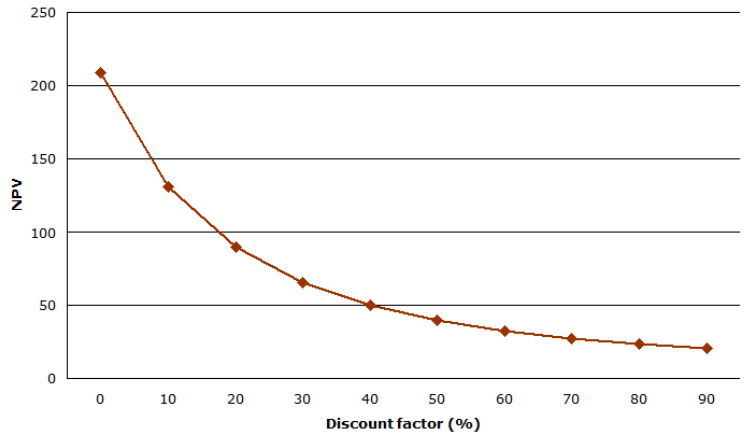


Figura 12.3: Valor Actual Neto (VAN) vs Factor de Descuento (FD)

La liquidez es la capacidad que tiene una empresa para hacer frente a sus obligaciones financieras. En la imagen siguiente vemos la liquidez anual de nuestro proyecto, así como la liquidez acumulada.

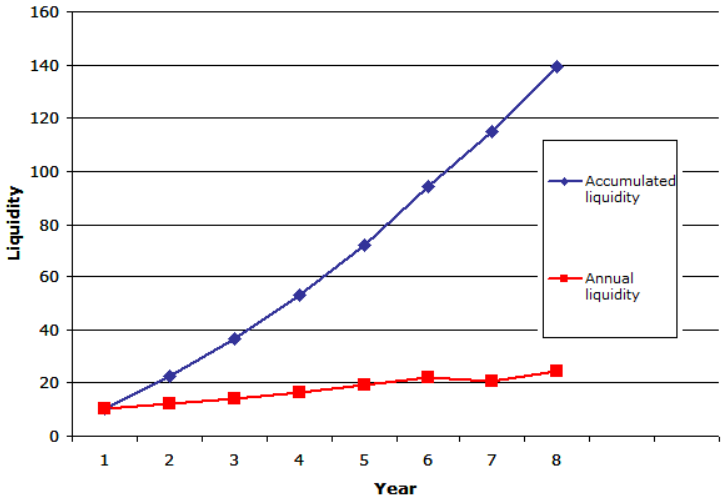


Figura 12.4: Liquidez

Los costes directos que se han introducido en el IPscore son los coste del diseño hardware (costes directos) que podemos extraer del Bill Of Materials (BOM) generado durante el desarrollo hardware, teniendo un resultado de 36,11 euros.

Bill Of Materials for TFG_Marcos_v4

Design Title TFG_Marcos_v4
Author
Document Number
Revision
Design Created sábado, 4 de mayo de 2019
Design Last Modified martes, 7 de mayo de 2019
Total Parts In Design 61

0 Modules

Quantity	References	Value	Stock Code	Unit Cost
Sub-totals:				€0,00

24 Capacitors

Quantity	References	Value	Stock Code	Unit Cost
1	C3	10uF	KEMET C322C101G1G5TA	€0,61
3	C4-C5,C12	22uF	WURTH ELEKTRONIK 865080542006	€0,26
7	C6-C8,C10-C11,C15, C22	100nF	MULTICOMP MC0805B104K500A5.08MM	€0,10
2	C9,C24	10nF	VISHAY D103K69Y5PL63L0R	€0,27
1	C13	1uF	MULTICOMP MC0805B104K500A5.08MM	€0,10
2	C14,C19	4.7uF	VISHAY D472K33Y5PH63J5R	€0,15
2	C16-C17	33pF	KEMET C317C220K2G5TA	€0,26
1	C18	4.7uF	Nichicon UZG1H4R7MCL1GB	€0,62
2	C20-C21	1uF	MULTICOMP MCMLR50V105MY5V	€0,22
1	C23	120uF	PANASONIC ELECTRONIC COMPONENTS EEE-FK1A121SR	€0,38
2	C25-C26	10uF	TDK C2012X7S0J106M085AC	€0,98
Sub-totals:				€6,95

19 Resistors

Quantity	References	Value	Stock Code	Unit Cost
7	R1-R2,R4,R11-R13, R18	10K	VISHAY MRS25000C1002FRP00	€0,03
1	R3	1.0M	VISHAY MRS25000C1004FCT00	€0,09
1	R5	1K	VISHAY MRS25000C1001FCT00	€0,05
3	R6,R20,R22	1k8	VISHAY MRS25000C1801FCT00	€0,08
1	R7	2k2	VISHAY MRS25000C2201FCT00	€0,09
1	R8	390	VISHAY MRS25000C3900FCT00	€0,09
2	R9-R10	33	VISHAY MRS25000C3309FCT00	€0,05
3	R19,R21,R23	3k3	VISHAY MRS25000C3301FCT00	€0,08
Sub-totals:				€1,13

7 Integrated Circuits

Quantity	References	Value	Stock Code	Unit Cost
1	U1	RP401N501C-TR-FE	Ricoh Electronic Devices Company RP401N501C-TR-FE	€0,98
1	U2	MCP1802T-33	MICROCHIP MCP1802T-3302I/OT	€0,43
1	U3	FT232RL	FTDI FT232RL-REEL	€3,76
1	U4	USBLC6-4SC6	STMICROELECTRONICS USBLC6-4SC6	€0,37
1	U5	MCP73831	MICROCHIP MCP73831T-2ACI/OT	€0,45
1	U6	MAX3421E	MAXIM INTEGRATED PRODUCTS MAX3421EEHJ+	€7,14
1	U8	MIC2025/75	MICREL SEMICONDUCTOR MIC2025-2YM	€0,72
Sub-totals:				€13,85

0 Transistors

Figura 12.5: Bill Of Materials 1

Quantity	References	Value	Stock Code	Unit Cost
Sub-totals:				€0,00
2 Diodes				
Quantity	References	Value	Stock Code	Unit Cost
1	D1	DIODE	BROADCOM LIMITED HSMG-A100-K42J2	€0,20
1	D2	CUS10S30H3F	Toshiba CUS10S30H3F	€0,36
Sub-totals:				€0,56
9 Miscellaneous				
Quantity	References	Value	Stock Code	Unit Cost
1	BAT1	3V7	BAK YOBLP9225431S3	€5,80
1	ESP-12F	ESP-12F		€2,87
1	J1	MIN-USB-B5	AMP - TE CONNECTIVITY 1734035-1.	€0,98
1	J2	USB HOST	WURTH ELEKTRONIK 614004190021	€1,67
1	L1	MMZ2012S301A	TDK MMZ2012S301A	€0,10
2	L2-L3	HF50ACC453215	TDK HF50ACC453215T	€0,21
1	L4	SLF6028T-4R7M1R6-PF	TDK SLF6028T-4R7M1R6-PF	€0,85
1	X1	CRYSTAL	MULTICOMP HC49SM-12-30-50-60-16-ATF	€0,94
Sub-totals:				€13,63
Totals:				€36,11

martes, 18 de junio de 2019 17:19:55

Figura 12.6: Bill Of Materials2

Capítulo 13

Consideraciones finales, conclusiones y líneas futuras

La idea final de este proyecto es mostrar el desarrollo de un prototipo de un producto IoT. En conclusión podemos decir que se han mostrado todos los pasos para llegar al producto final, partiendo de una idea o un problema inicial. En primer lugar, se ha explicado como analizar el problema y detallar cada una de sus fases, para posteriormente realizar un estudio exhaustivo de las especificaciones y componentes necesarios que nos sirvieran para empezar a desarrollar el producto. Se ha intentado mostrar en esta memoria todos los pasos de la manera más profunda y técnica posible. Por otro lado, se ha explicado cada aplicación utilizada para la creación de cada elemento, tanto de software como de hardware.

Para el futuro se puede retomar este proyecto para modificar el firmware del sistema electrónico con el fin de que pueda servir para otros sensores o sistemas. Bastaría con conocer el protocolo de comunicación del sensor o sistema a conectar y que este tuviera entrada USB. En líneas generales también se puede mejorar el sistema electrónico con mejores componentes para desarrollar un sistema comercial mas robusto y potente si fuera necesario. Diremos que el producto desarrollado no es un producto final, sino un prototipo.

Otra línea futura es el desarrollo de una aplicación móvil mas completa. Se ha creado una primera aplicación móvil sencilla pero ampliamente escalable, ya que se puede añadir la solicitud de cualquier tipo de dato al sensor, simplemente imitando el comportamiento de la solicitud de datos de humedad o temperatura y conociendo la trama que se debe enviar al sensor.

Las conclusiones personales son que el desarrollo de este Trabajo Fin de Grado me ha supuesto un aprendizaje profundo de cada uno de los procesos para generar un sistema electrónico utilizando las últimas tecnologías. Afianzando conceptos y prácticas adquiridas en las asignaturas del grado, así como aprender y ampliar mis conocimientos en diferentes ramas de las telecomunicaciones.

Parte IV

Apéndices

Apéndice A

Código en Arduino firmware C/C++

```
#include <cdcftdi.h>
#include <usbhub.h>
//Required for doing all WiFi related functionalities
//such as connection, AP, etc.
#include "ESP8266WiFi.h"
//Handles all HTTP protocols
#include "ESP8266WebServer.h"

#include "pgmstrings.h"

// Satisfy the IDE, which needs to see the include statment in the ino too.
#ifdef dobogusinclude
#include <spi4teensy3.h>
#endif
#include <SPI.h>

class FTDIAsync : public FTDIAsyncOper
{
public:
    uint8_t OnInit(FTDI *pftdi);
};

uint8_t FTDIAsync::OnInit(FTDI *pftdi)
{
    uint8_t rcode = 0;

    rcode = pftdi->SetBaudRate(115200);

    if (rcode)
    {
        ErrorMessage<uint8_t>(PSTR("SetBaudRate"), rcode);
        return rcode;
    }
    rcode = pftdi->SetFlowControl(FTDI_SIO_DISABLE_FLOW_CTRL);

    if (rcode)
        ErrorMessage<uint8_t>(PSTR("SetFlowControl"), rcode);
```

```
38
39      return rcode;
40  }
41
42  USB             Usb;
43  FTDIAsync       FtdiAsync;
44  FTDI            Ftdi(&Usb, &FtdiAsync);
45
46  //Generated WiFi Acces Point connection
47  const char *ssid = "ESPTFG";
48  const char *password = "password123";
49
50  //Server on port 80
51  ESP8266WebServer server(80);
52
53  String sensorValue;
54
55  //Handler for the rooth path
56  void handleRoot() {
57    server.send(200, "text/plain", "App TFG Sensor");
58    Serial.println("I have received a request in root: ");
59  }
60
61  void setup() {
62
63    //Baudrate
64    Serial.begin(115200);
65
66    #if !defined(__MIPSEL__)
67    //Wait for serial port to connect
68    while (!Serial);
69    #endif
70    Serial.println("Start");
71
72    if (Usb.Init() == -1)
73      Serial.println("OSCOKIRQ failed to assert");
74
75    delay( 200 );
76
77    WiFi.softAP(ssid, password);
78
79    Serial.println();
80    Serial.print("Server IP address: ");
81    //Returns ESP8266 IP address
82    Serial.println(WiFi.softAPIP());
83
84    Serial.print("Server MAC address: ");
85    //Returns ESP8266 MAC address
86    Serial.println(WiFi.softAPmacAddress());
87
88    //Which routine to handle at root location
89    server.on("/", handleRoot);
```

```
90
91    //Start server
92    server.begin();
93    Serial.println("Server listening");
94  }
95
96  void loop() {
97
98    //Handling of incoming requests
99    server.handleClient();
100
101   Usb.Task();
102
103   //Send the measurement to the sensor and return the data
104   String getSensorData (string measure){
105         if( Usb.getUsbTaskState() == USB_STATE_RUNNING ) {
106     uint8_t rcode;
107
108       if(Serial.available()) {
109         uint8_t data = measure;
110         //sending to the sensor
111         rcode = Ftdi.SndData(1, &data);
112         if (rcode)
113           ErrorMessage<uint8_t>(PSTR("SndData"), rcode);
114       }//if(Serial.available()...
115
116       delay(50);
117
118       //reading the sensor
119       //buffer size must be greater or equal to max.packet size
120       //it it set to 64 (largest possible max.packet size)
121       //here, can be tuned down for particular endpoint
122       char  buf[64];
123       uint16_t rcvd = 64;
124       rcode = Ftdi.RcvData(&rcvd, buf);
125        if (rcode && rcode != hrNAK)
126         ErrorMessage<uint8_t>(PSTR("Ret"), rcode);
127
128         if( rcvd ) { //more than zero bytes received
129           for(uint16_t i=0; i < rcvd; i++ ) {
130             if (buf[i]!=0x60 )
131             {
132                if (buf[i]!=0x01 )
133               {
134                  sensorValue = buf[i];
135               }
136             }
137           }
138           return sensorValue;
139         }
140       delay(10);
141     }//if( Usb.getUsbTaskState() == USB_STATE_RUNNING..
```

```
142   }
143
144   //Temperature
145     server.on("/temp", []() {
146
147   //Depending on measurement
148     String input = "REA 1 3 5";
149   //Send the measurement(input) to the sensor
150   //and receive the data(sensorValue)
151   sensorValue = getSensorData (input);
152     String separator ="/" ;
153     String response = input+separator+sensorValue;
154
155     Serial.print("I have received a request in Temperature "+response);
156   //Send data to app
157     server.send(200, "text / plain", response);
158     });
159
160   //Humidity
161     server.on("/humidity", []() {
162   //Depending on measurement
163     String input = "REA 1 3 10";
164   //Send the measurement(input) to the sensor
165   //and receive the data(sensorValue)
166   sensorValue = getSensorData (input);
167     String separator ="/" ;
168     String response = input+separator+sensorValue;
169
170     Serial.println("I have received a request in Humidity "+response);
171   //Send data to app
172     server.send(200, "text / plain", response);
173     });
174
175   //Pressure
176     server.on("/pressure", []() {
177   //Depending on measurement
178     String input = "REA 1 3 9";
179   //Send the measurement(input) to the sensor
180   //and receive the data(sensorValue)
181   sensorValue = getSensorData (input);
182     String separator ="/" ;
183     String response = input+separator+sensorValue;
184
185     Serial.println("I have received a request in Pressure " + response);
186   //Send data to app
187     server.send(200, "text / plain", response);
188     });
189
190     //Oxygen
191     server.on("/oxygen", []() {
192   //Depending on measurement
193     String input = "REA 1 3 12";
```

```
194   //Send the measurement(input) to the sensor
195   //and receive the data(sensorValue)
196   sensorValue = getSensorData (input);
197     String separator ="/" ;
198     String response = input+separator+sensorValue;
199
200     Serial.println("I have received a request in Oxygen "+response);
201   //Send data to app
202     server.send(200, "text / plain", response);
203     });
204 }
```

Apéndice B

Código aplicación sensor para Android

B.1. Código del MainActivity.java

```
1  package example.bd.basededatos;
2
3  import android.os.Bundle;
4  import android.support.design.widget.FloatingActionButton;
5  import android.support.design.widget.Snackbar;
6  import android.support.v7.app.AppCompatActivity;
7  import android.support.v7.widget.Toolbar;
8  import android.view.View;
9  import android.view.Menu;
10 import android.view.MenuItem;
11 import  android.widget.Button;
12 import  android.widget.TextView;
13 import android.content.Intent;
14 import android.support.v7.app.AppCompatActivity;
15 import android.os.Bundle;
16 import android.view.View;
17 import android.widget.EditText;
18
19
20 public class MainActivity extends AppCompatActivity {
21
22
23     TextView textView ;
24     String ip = new String("http://192.168.4.1");
25     String Response = new String();
26     String Request = new String();
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.activity_main);
31         final Intent intentGraph = new Intent(this, Graph.class);
32
33          textView = (TextView) findViewById(R.id.textView);
34
35
```

```
36          //Button creation for a normal ping
37          Button buttonPing = (Button) findViewById(R.id.buttonPing);
38
39          buttonPing.setOnClickListener(new View.OnClickListener() {
40              public void onClick(View v) {
41                  Thread thread = new Thread(new Runnable(){
42                      @Override
43
44                      public void run() {
45
46                          try {
47                              JavaHttpUrlConnectionReader Connection = new
    JavaHttpUrlConnectionReader();
48                              Response = Connection.doHttpUrlConnectionAction
    (ip,Request);
49                              System.out.println(Response);
50                              runOnUiThread(new Runnable() {
51                                  @Override
52                                  public void run() {
53
54                                      textView.setText(Response);
55                                  }
56                              });
57                              //Your code goes here
58                          } catch (Exception e) {
59                              e.printStackTrace();
60                          }
61
62
63                      }
64                  });
65                  thread.start();
66
67
68
69
70              }
71          });
72
73
74          //Button creation for a humidity check
75           Button buttonHumidity = (Button) findViewById(R.id.buttonHumidity)
    ;
76
77                  buttonHumidity.setOnClickListener(new View.OnClickListener
    () {
78                      public void onClick(View v) {
79                          // your handler code here
80                          intentGraph.putExtra("request", "humidity");
81                          intentGraph.putExtra("ip", ip);
82                          startActivity(intentGraph);
83                          // your handler code here
```

```
84
85
86
87                      }
88          });
89
90          //Button creation for a temperature check
91           Button buttonTemp = (Button) findViewById(R.id.buttonTemp);
92          buttonTemp.setOnClickListener(new View.OnClickListener() {
93              public void onClick(View v) {
94                  // your handler code here
95                  intentGraph.putExtra("request", "temp");
96                  intentGraph.putExtra("ip", ip);
97                  startActivity(intentGraph);
98
99
100             }
101         });
102
103
104
105     }
106
107     @Override
108     public boolean onCreateOptionsMenu(Menu menu) {
109         // Inflate the menu; this adds items to the action bar if it is
    present.
110         getMenuInflater().inflate(R.menu.menu_main, menu);
111         return true;
112     }
113
114     @Override
115     public boolean onOptionsItemSelected(MenuItem item) {
116         // Handle action bar item clicks here. The action bar will
117         // automatically handle clicks on the Home/Up button, so long
118         // as you specify a parent activity in AndroidManifest.xml.
119         int id = item.getItemId();
120
121         //noinspection SimplifiableIfStatement
122         if (id == R.id.action_settings) {
123             return true;
124         }
125
126         return super.onOptionsItemSelected(item);
127     }
128
129
130 }
```

B.2. Código del JavaHttpUrlConnectionReader.java

```
1  package example.bd.basededatos;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8  import java.net.URLEncoder;
9
10
11
12 //A complete Java class that shows how to open a URL, then read data (text)
       from that URL,
13 //HttpURLConnection class (in combination with an InputStreamReader and
      BufferedReader).
14
15 public class JavaHttpUrlConnectionReader
16 {
17     public static void main(String[] args)
18             throws Exception
19     {
20         new JavaHttpUrlConnectionReader();
21     }
22
23
24
25
26     //Returns the output from the given URL.
27
28     //I tried to hide some of the ugliness of the exception-handling
29     //in this method, and just return a high level Exception from here.
30     //Modify this behavior as desired.
31
32     //@param desiredUrl
33     //@return
34   //@throws Exception
35
36     public String doHttpUrlConnectionAction(String desiredUrl,String
      desiredRequest)
37             throws Exception
38     {
39         URL url = null;
40         BufferedReader reader = null;
41         StringBuilder stringBuilder;
42
43         try
44         {
45             // create the HttpURLConnection
46             if (desiredRequest.isEmpty())
```

```
47                  url = new URL(desiredUrl);
48                  else
49                 url = new URL(desiredUrl+"/"+desiredRequest);
50
51              HttpURLConnection connection = (HttpURLConnection) url.
    openConnection();
52
53              // just want to do an HTTP GET here
54              connection.setRequestMethod("GET");
55
56              // uncomment this if you want to write output to this url
57              //connection.setDoOutput(true);
58
59              // give it 15 seconds to respond
60              connection.setReadTimeout(15*1000);
61              connection.connect();
62
63              // read the output from the server
64              reader = new BufferedReader(new InputStreamReader(connection.
    getInputStream()));
65              stringBuilder = new StringBuilder();
66
67              String line = null;
68              while ((line = reader.readLine()) != null)
69              {
70                  stringBuilder.append(line + "\n");
71              }
72              return stringBuilder.toString();
73          }
74          catch (Exception e)
75          {
76              e.printStackTrace();
77              throw e;
78          }
79          finally
80          {
81              // close the reader; this can throw an exception too, so
82              // wrap it in another try/catch block.
83              if (reader != null)
84              {
85                  try
86                  {
87                      reader.close();
88                  }
89                  catch (IOException ioe)
90                  {
91                      ioe.printStackTrace();
92                  }
93              }
94          }
95      }
96  }
```

B.3. Código del Graph.java

```
1  package example.bd.basededatos;
2  import java.util.Random;
3
4  import android.app.Activity;
5  import android.net.ConnectivityManager;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.widget.Button;
9  import android.widget.CompoundButton;
10 import android.widget.Toast;
11 import android.view.Gravity;
12 import  android.net.NetworkInfo;
13 import com.jjoe64.graphview.GraphView;
14 import com.jjoe64.graphview.Viewport;
15 import com.jjoe64.graphview.series.DataPoint;
16 import com.jjoe64.graphview.series.LineGraphSeries;
17 import java.util.Calendar;
18 import java.util.*;
19 import android.net.wifi.WifiManager;
20 import android.content.Context;
21 import android.widget.Switch;
22
23 public class Graph extends Activity {
24
25     private static final Random RANDOM = new Random();
26     private LineGraphSeries<DataPoint> series;
27     private int lastX = 0;
28     String ip = new String();
29     String Response = new String();
30     String Request = new String();
31     boolean stopThread = false;
32     GraphView graph;
33     public Vector<DataReceived> Data = new Vector<DataReceived>();
34     // here, we choose to display max 10 points on the viewport and we
       scroll to end
35
36
37     @Override
38     protected void onCreate(Bundle savedInstanceState) {
39         super.onCreate(savedInstanceState);
40         setContentView(R.layout.activity_graph);
41         // we get graph view instance
42
43         // data
```

```
44          graph = (GraphView) findViewById(R.id.graph);
45          series = new LineGraphSeries<DataPoint>();
46          graph.addSeries(series);
47          // customize a little bit viewport
48          Viewport viewport = graph.getViewport();
49          viewport.setYAxisBoundsManual(true);
50          Request = getIntent().getExtras().getString("request");
51          ip = getIntent().getExtras().getString("ip");
52
53          if (Request.equals("temp"))
54          {
55              viewport.setMinY(-20);
56              viewport.setMaxY(50);
57
58          }
59          else if (Request.equals("humidity"))
60          {
61              viewport.setMinY(0);
62              viewport.setMaxY(100);
63          }
64          viewport.setScrollable(true);
65
66
67
68
69          Button buttonDB = (Button) findViewById(R.id.databaseButton);
70
71          buttonDB.setOnClickListener(new View.OnClickListener() {
72              public void onClick(View v) {
73
74
75                  ConnectivityManager connectivityManager = (
    ConnectivityManager)getApplicationContext().getSystemService(Context.
    CONNECTIVITY_SERVICE);
76
77
78                  if(connectivityManager.getNetworkInfo(ConnectivityManager.
    TYPE_MOBILE).getState() == NetworkInfo.State.CONNECTED ||
79                          connectivityManager.getNetworkInfo(
    ConnectivityManager.TYPE_WIFI).getState() == NetworkInfo.State.CONNECTED
    )
80                  {
81
82                      for(int i=0; i<Data.size(); i++){
83                       DBSender s=new DBSender(Graph.this,"http://
    unchaperoned-helmet.000webhostapp.com/db_post.php",Data.get(i).date,Data
    .get(i).input, Data.get(i).output);
84
85                          s.execute();
86
87                          }
88
```

```
89
90                  }
91                  else
92                      Toast.makeText(getApplicationContext(),"No hay conexion
    a internet",Toast.LENGTH_LONG).show();
93              }
94          });
95
96          // initiate a Switch
97          Switch wifiSwitch = (Switch) findViewById(R.id.wifiSwitch);
98
99
100 // check current state of a Switch (true or false).
101
102
103         wifiSwitch.setChecked(true);
104
105         wifiSwitch.setOnCheckedChangeListener(new CompoundButton.
    OnCheckedChangeListener() {
106             @Override
107             public void onCheckedChanged(CompoundButton buttonView, boolean
    isChecked) {
108                 WifiManager wifiManager = (WifiManager)
    getApplicationContext().getSystemService(Context.WIFI_SERVICE);
109                 if (isChecked)
110                 {
111
112
113                     wifiManager.setWifiEnabled(true);
114             }
115             else
116                 {
117                     stopReceiving();
118                     wifiManager.setWifiEnabled(false);
119                 }
120             }
121         });
122
123
124
125
126
127     }
128
129
130
131     private void stopReceiving()
132     {
133         stopThread = true;
134         graph.getViewport().setYAxisBoundsManual(true);
135         graph.getViewport().setXAxisBoundsManual(true);
136         graph.getViewport().setScalable(true);
```

```
137         graph.getViewport().setScalableY(true);
138     }
139     @Override
140     protected void onResume() {
141
142         super.onResume();
143         // we're going to simulate real time with thread that append data
    to the graph
144
145         Thread thread = new Thread(new Runnable(){
146             @Override
147
148             public void run() {
149
150                 while (stopThread== false) {
151
152                 try {
153                     JavaHttpUrlConnectionReader Connection = new
    JavaHttpUrlConnectionReader();
154                     Response = Connection.doHttpUrlConnectionAction(ip,
    Request);
155                     System.out.println(Response);
156                     final String[] responseSplitted = Response.split("/");
157                     runOnUiThread(new Runnable() {
158                         @Override
159                         public void run() {
160
161                             DataReceived d = new DataReceived(Calendar.
    getInstance().getTime().toString(),responseSplitted[0] ,
    responseSplitted[1]);
162                             Data.add(d);
163                             series.appendData(new DataPoint(lastX++, Double
    .valueOf(responseSplitted[1])), false, 10);
164
165                             Toast toast = Toast.makeText(
    getApplicationContext(),"Datos recbidos. Valor : "+responseSplitted[1],
    Toast.LENGTH_SHORT);
166                             toast.show();
167                         }
168
169
170                     });
171
172
173                     //Your code goes here
174                 } catch (Exception e) {
175                     e.printStackTrace();
176                 }
177                     try {
178
179                         Thread.sleep(2000);
180                     } catch (InterruptedException e) {
```

```
181                         // manage error ...
182                     }
183
184                 }
185
186                 }
187             });
188             thread.start();
189
190 }
191
192     // add random data to graph
193
194     @Override
195     public void onBackPressed() {
196         super.onBackPressed();
197         stopThread = true;
198
199     }
200
201
202 }
```

B.4. Código del DBSender.java

```
1  package example.bd.basededatos;
2
3  import android.app.ProgressDialog;
4  import android.content.Context;
5  import android.os.AsyncTask;
6  import android.widget.EditText;
7  import android.widget.Toast;
8
9  import java.io.BufferedReader;
10 import java.io.BufferedWriter;
11 import java.io.IOException;
12 import java.io.InputStreamReader;
13 import java.io.OutputStream;
14 import java.io.OutputStreamWriter;
15 import java.net.HttpURLConnection;
16
17
18  //1.SEND DATA FROM EDITTEXT OVER THE NETWORK
19  //2.DO IT IN BACKGROUND THREAD
20  //3.READ RESPONSE FROM A SERVER
21
22 public class DBSender extends AsyncTask<Void,Void,String> {
23
```

```
24      Context c;
25      String urlAddress;
26      String nameTxt,posTxt,teamTxt;
27
28      String date,input,output;
29
30      ProgressDialog pd;
31      public String responseBack;
32
33      //1.OUR CONSTRUCTOR
34      //2.RECEIVE CONTEXT,URL ADDRESS AND EDITTEXTS FROM OUR MAINACTIVITY
35
36      public DBSender(Context c, String urlAddress,String...values) {
37          this.c = c;
38          this.urlAddress = urlAddress;
39
40          //INPUT EDITTEXTS
41          this.date=values[0];
42          this.input=values[1];
43          this.output=values[2];
44
45          //GET TEXTS FROM EDITEXTS
46      }
47
48     //1.SHOW PROGRESS DIALOG WHILE DOWNLOADING DATA
49
50      @Override
51      protected void onPreExecute() {
52          super.onPreExecute();
53
54          pd=new ProgressDialog(c);
55          pd.setTitle("Enviando");
56          pd.setMessage("Enviando datos a la base de datos");
57          pd.show();
58      }
59
60      //1.WHERE WE SEND DATA TO NETWORK
61      //2.RETURNS FOR US A STRING
62
63      @Override
64      protected String doInBackground(Void... params) {
65          return this.send();
66      }
67
68
69    //1. CALLED WHEN JOB IS OVER
70    //2. WE DISMISS OUR PD
71    //3.RECEIVE A STRING FROM DOINBACKGROUND
72
73      @Override
74      protected void onPostExecute(String response) {
75          super.onPostExecute(response);
```

```
76
77          pd.dismiss();
78
79          if(response == null)
80
81              //FAILURE
82              Toast.makeText(c,"No se ha podido actualizar la base de datos"+
    response,Toast.LENGTH_LONG).show();
83      }
84
85      //SEND DATA OVER THE NETWORK
86      //RECEIVE AND RETURN A RESPONSE
87
88      private String send()
89      {
90          //CONNECT
91          HttpURLConnection con=DBConnector.connect(urlAddress);
92
93          if(con==null)
94          {
95              return null;
96          }
97          try
98          {
99              OutputStream os=con.getOutputStream();
100
101             //WRITE
102             BufferedWriter bw=new BufferedWriter(new OutputStreamWriter(os,
    "UTF-8"));
103
104             bw.write(new DataPackager(date,input,output).packData());
105
106             bw.flush();
107
108             //RELEASE RES
109             bw.close();
110             os.close();
111
112             //HAS IT BEEN SUCCESSFUL?
113             int responseCode=con.getResponseCode();
114
115             if(responseCode==con.HTTP_OK)
116             {
117                 //GET EXACT RESPONSE
118                 BufferedReader br=new BufferedReader(new InputStreamReader(
    con.getInputStream()));
119                 StringBuffer response=new StringBuffer();
120
121                 String line;
122
123                 //READ LINE BY LINE
124                 while ((line=br.readLine()) != null)
```

```
125                 {
126                     response.append(line);
127                 }
128 
129                 //RELEASE RES
130                 br.close();
131 
132                 return response.toString();
133 
134             }else
135             {
136             }
137         } catch (IOException e) {
138             e.printStackTrace();
139         }
140         return null;
141     }
142 }
```

B.5. Código del DBConnector.java

```
1  package example.bd.basededatos;
2 
3  import java.io.IOException;
4  import java.net.HttpURLConnection;
5  import java.net.MalformedURLException;
6  import java.net.URL;
7 
8  public class DBConnector {
9 
10 
11     //1.SHALL HELP US ESTABLISH A CONNECTION TO THE NETWORK
12     //1. WE ARE MAKING A POST REQUEST
13 
14     public static HttpURLConnection connect(String urlAddress) {
15 
16         try
17         {
18             URL url=new URL(urlAddress);
19             HttpURLConnection con= (HttpURLConnection) url.openConnection()
       ;
20 
21             //SET PROPERTIES
22             con.setRequestMethod("POST");
23             con.setConnectTimeout(20000);
24             con.setReadTimeout(20000);
25             con.setDoInput(true);
26             con.setDoOutput(true);
```

```
27
28              //RETURN
29              return con;
30
31          } catch (MalformedURLException e) {
32              e.printStackTrace();
33          } catch (IOException e) {
34              e.printStackTrace();
35          }
36
37          return null;
38
39      }
40
41  }
```

B.6. Código del DataReceived.java

```
1  package example.bd.basededatos;
2
3  public class DataReceived {
4
5      String date, input, output;
6
7      public  DataReceived (String date,String input, String output)
8      {
9          this.date= date;
10         this.input = input;
11         this.output = output;
12
13     }
14 }
```

B.7. Código del DataPackager.java

```
1  package example.bd.basededatos;
2
3  import org.json.JSONException;
4  import org.json.JSONObject;
5
6  import java.io.UnsupportedEncodingException;
7  import java.net.URLEncoder;
8  import java.util.Iterator;
9
```

```
10
11  // 1.BASICALLY PACKS DATA WE WANNA SEND
12
13 public class DataPackager {
14
15     String date,input,output;
16
17
18     //SECTION 1.RECEIVE ALL DATA WE WANNA SEND
19
20     public DataPackager(String date, String input, String output) {
21         this.date = date;
22         this.input = input;
23         this.output = output;
24     }
25
26
27    //SECTION 2
28    //1.PACK THEM INTO A JSON OBJECT
29    //1. READ ALL THIS DATA AND ENCODE IT INTO A FROMAT THAT CAN BE SENT VIA
       NETWORK
30
31     public String packData()
32     {
33         JSONObject jo=new JSONObject();
34         StringBuffer packedData=new StringBuffer();
35
36         try
37         {
38             jo.put("date",date);
39             jo.put("input",input);
40             jo.put("output",output);
41
42             Boolean firstValue=true;
43
44             Iterator it=jo.keys();
45
46             do {
47                 String key=it.next().toString();
48                 String value=jo.get(key).toString();
49
50                 if(firstValue)
51                 {
52                     firstValue=false;
53                 }else
54                 {
55                     packedData.append("&");
56                 }
57
58                 packedData.append(URLEncoder.encode(key,"UTF-8"));
59                 packedData.append("=");
60                 packedData.append(URLEncoder.encode(value,"UTF-8"));
```

```
61
62              }while (it.hasNext());
63
64              return packedData.toString();
65
66          } catch (JSONException e) {
67              e.printStackTrace();
68          } catch (UnsupportedEncodingException e) {
69              e.printStackTrace();
70          }
71
72          return null;
73      }
74
75  }
```

B.8. Código del db_post.php

```
1  <?php
2
3  $host="localhost";
4  $username='id8821736_marcos';
5  $pwd='password123';
6  $db="id8821736_datos_app";
7
8  $con=mysqli_connect($host,$username,$pwd,$db) or die('Unable to connect');
9
10 if(mysqli_connect_error($con))
11 {
12     echo "Failed to Connect to Database ".mysqli_connect_error();
13 }
14
15 $date=$_POST['date'];
16 $input=$_POST['input'];
17 $output=$_POST['output'];
18
19 $sql="INSERT INTO Data(DATE,INPUT,OUTPUT) VALUES('$date','$input','$output
       ')";
20
21 $result=mysqli_query($con,$sql);
22
23 if($result)
24 {
25     echo ('Successfully Saved');
26 }else
27 {
28     echo('Not saved Successfully');
29 }
```

```
30 mysqli_close($con);
31
32 ?>
33
```

B.9. Código del db_post.php

```
1  <?php
2  $host="localhost";
3  $username='id8821736_marcos';
4  $pwd='password123';
5  $db="id8821736_datos_app";
6
7  $con=mysqli_connect($host,$username,$pwd,$db);
8  // Check connection
9  if (mysqli_connect_errno())
10 {
11 echo "Failed to connect to MySQL: " . mysqli_connect_error();
12 }
13
14 $result = mysqli_query($con,"SELECT * FROM Data");
15
16 echo "<table border='1'>
17 <tr>
18 <th>Date</th>
19 <th>Input</th>
20 <th>Output</th>
21 </tr>";
22
23 while($row = mysqli_fetch_array($result))
24 {
25 echo "<tr>";
26 echo "<td>" . $row['DATE'] . "</td>";
27 echo "<td>" . $row['INPUT'] . "</td>";
28 echo "<td>" . $row['OUTPUT'] . "</td>";
29 echo "</tr>";
30 }
31 echo "</table>";
32
33 mysqli_close($con);
34 ?>
35
```

Apéndice C

Código simulador sensor C para VSM Studio

```
1  // ===========================================================
2  // Simulador Sensor
3  // Autor: Marcos Martin Gutierrez
4  // Fecha: Enero 2018
5  // Asignatura: Trabajo Fin de Grado
6  // ===========================================================
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 #include <time.h>
12 #include "lpc2103.h"
13 // Function prototypes
14 void _U0prtnum(unsigned int);
15 void _U0prthex(unsigned int);
16 void _U0prthex8(unsigned char);
17 void _U0puts(unsigned char *);
18 void _U0putch(unsigned char);
19 unsigned char _U0getch();
20 void _delay_loop(unsigned int);
21
22 void _puts(char *);
23 void _printf(char *,...);
24 extern void (*_vputch)(int);
25 extern int (*_vgetch)();
26 void _putch(char);
27 char _getch();
28 int  _gets(unsigned char *,int);
29 #define UART0_printf _printf
30
31 #define FOSC     14745600          // Crystal frequency
32 #define MSEL     4                 // PLL multiplier
33 #define CCLK     (FOSC*MSEL)       // CPU clock
34 #define PCKDIV   1                 // APB divider
35 #define PCLK     (CCLK/PCKDIV)     // Peripheral clock
36 // delays
37 #define _delay_us(n) _delay_loop(CCLK/400*n/10000-1)
```

```
38 #define _delay_ms(n) _delay_loop(CCLK/4000*n-1)
39 // ==========================================================
40 // Register UxLSR (Line Status Register): Bits
41 #define DR          (1<<0)
42 #define OE          (1<<1)
43 #define PE          (1<<2)
44 #define FE          (1<<3)
45 #define BI          (1<<4)
46 #define THR_E       (1<<5)
47 #define TX_E        (1<<6)
48 #define RX_ERROR    (1<<7)
49 // ----------------------------------------------------------
50 void UART0_Init()
51 {
52  unsigned char lcr;
53  lcr = 0x03;     // 00000011 Line Control Register
54                  //       .. Character length 11 = 8 bits
55                  //      .   Stop bits         0 = 1 bit
56                  //     .    Parity enable     0 = No parity
57                  //    .     Even parity       non sense
58                  //   .      Stick parity      non sense
59                  //  .       Break control     0 = No break
60                  // .        DLAB              0 = No Access to DLM,DLL
61  U0LCR = 0x80 | lcr; // DLAB = 1, Access to DLM,DLL
62  U0DLL = 32;
63  U0DLM =  0;     // Baud = 4*14745600/(16* 32) = 115200
64  U0LCR = lcr;    // Same a above, but DLAB = 0 = Normal access to RBR/THR &
      IER
65  U0FCR = 0x07;   // 00000111 FIFO Control Register
66                  //        . FIFO enable       1 = TX&RX FIFOs enabled
67                  //       .  RX FIFO Reset     1 = Reset RX FIFO
68                  //      .   TX FIFO Reset     1 = Reset TX FIFO
69                  // ..       RX interrupt trigger level 00 = 1 byte
70 };
71 // ----------------------------------------------------------
72 void UART0_putch(char c)
73 {
74  while(!(U0LSR & THR_E));
75  U0THR = c;
76 }
77 // ----------------------------------------------------------
78 char UART0_getch()
79 {
80  while(!(U0LSR & DR));
81  return U0RBR;
82 }
83 // ----------------------------------------------------------
84 void UART0_print(char * s)
85 {
86  int i;
87  i= 0;
88  while(s[i]) UART0_putch(s[i++]);
```

```
89 }
90 #define MAX_CHAR 90
91 #define MAX_RETURN_CHAR 900
92 void GetData(char *T)
93 {
94  int i;
95  unsigned char c;
96
97  i=0;
98  do
99  {
100   c=UART0_getch();
101   UART0_putch(c);
102   T[i] = c;
103   i++;
104  } while( (c!='\r') && (c!='\n') && (i<(MAX_CHAR)) );
105  T[i-1]=0;
106 }
107
108 // ------------------------------------------------------------------------
109 int main (void)
110 {
111  char T[MAX_CHAR];
112
113
114  volatile unsigned char r;
115
116  PINSEL0 = 0x00000005; // RXD0,TXD0,RXD1,TXD1
117  PINSEL1 = 0x00000000; // P0.21 is GPIO
118
119  IOPIN   = 0xFFFFFFFF; // All outputs high
120  // P0.4 is input
121
122  UART0_Init();
123
124
125
126   while(1)
127   {
128
129   GetData( T );
130   if (strcmp(T, "#VERS")==0)
131     UART0_print("#VERS 7 1 105 7\r\n");
132   else if (strcmp(T, "#INFO ")==0)
133     UART0_print("#INFO8 1 0 1 1 3700 1 1 2\r\n");
134 //Level0Registers
135   //[0]Settings
136   else if (strcmp(T, "REA 1 0 0")==0)
137     UART0_print("REA 1 0 0 20000\r\n");
138   else if (strcmp(T, "REA 1 0 1")==0)
139     UART0_print("REA 1 0 1 -1\r\n");
140   else if (strcmp(T, "REA 1 0 2")==0)
```

```
141     UART0_print("REA 1 0 2 0\r\n");
142   else if (strcmp(T, "REA 1 0 3")==0)
143     UART0_print("REA 1 0 3 5\r\n");
144   else if (strcmp(T, "REA 1 0 4")==0)
145     UART0_print("REA 1 0 4 1\r\n");
146   else if (strcmp(T, "REA 1 0 5")==0)
147     UART0_print("REA 1 0 5 5\r\n");
148   else if (strcmp(T, "REA 1 0 6")==0)
149     UART0_print("REA 1 0 6 4000\r\n");
150   else if (strcmp(T, "REA 1 0 7")==0)
151     UART0_print("REA 1 0 7 0\r\n");
152   else if (strcmp(T, "REA 1 0 8")==0)
153     UART0_print("REA 1 0 8 0\r\n");
154 //Level0Registers
155   //[1]Calibration
156   else if (strcmp(T, "REA 1 1 0")==0)
157     UART0_print("REA 1 1 0 53000\r\n");
158   else if (strcmp(T, "REA 1 1 1")==0)
159     UART0_print("REA 1 1 1 20000\r\n");
160   else if (strcmp(T, "REA 1 1 2")==0)
161     UART0_print("REA 1 1 2 20000\r\n");
162   else if (strcmp(T, "REA 1 1 3")==0)
163     UART0_print("REA 1 1 3 20000\r\n");
164   else if (strcmp(T, "REA 1 1 4")==0)
165     UART0_print("REA 1 1 4 1013000\r\n");
166   else if (strcmp(T, "REA 1 1 5")==0)
167     UART0_print("REA 1 1 5 100000\r\n");
168   else if (strcmp(T, "REA 1 1 6")==0)
169     UART0_print("REA 1 1 6 850\r\n");
170   else if (strcmp(T, "REA 1 1 7")==0)
171     UART0_print("REA 1 1 7 81\r\n");
172   else if (strcmp(T, "REA 1 1 8")==0)
173     UART0_print("REA 1 1 8 4000\r\n");
174   else if (strcmp(T, "REA 1 1 9")==0)
175     UART0_print("REA 1 1 9 -70\r\n");
176   else if (strcmp(T, "REA 1 1 10")==0)
177     UART0_print("REA 1 1 10 1184\r\n");
178   else if (strcmp(T, "REA 1 1 11")==0)
179     UART0_print("REA 1 1 11 0\r\n");
180   else if (strcmp(T, "REA 1 1 12")==0)
181     UART0_print("REA 1 1 12 0\r\n");
182   else if (strcmp(T, "REA 1 1 13")==0)
183     UART0_print("REA 1 1 13 0\r\n");
184   else if (strcmp(T, "REA 1 1 14")==0)
185     UART0_print("REA 1 1 14 0\r\n");
186   else if (strcmp(T, "REA 1 1 15")==0)
187     UART0_print("REA 1 1 15 0\r\n");
188   else if (strcmp(T, "REA 1 1 16")==0)
189     UART0_print("REA 1 1 16 -370\r\n");
190   else if (strcmp(T, "REA 1 1 17")==0)
191     UART0_print("REA 1 1 17 0\r\n");
192   else if (strcmp(T, "REA 1 1 18")==0)
```

```
193     UART0_print("REA 1 1 18 20950\r\n");
194 //Level0Registers
195   //[3]Results
196   else if (strcmp(T, "REA 1 3 0")==0)
197     UART0_print("REA 1 3 0 0\r\n");
198   else if (strcmp(T, "REA 1 3 1")==0)
199     UART0_print("REA 1 3 1 1\r\n");
200   else if (strcmp(T, "REA 1 3 2")==0)
201     UART0_print("REA 1 3 2 2\r\n");
202   else if (strcmp(T, "REA 1 3 3")==0)
203     UART0_print("REA 1 3 3 3\r\n");
204   else if (strcmp(T, "REA 1 3 4")==0)
205     UART0_print("REA 1 3 4 4\r\n");
206   else if (strcmp(T, "REA 1 3 5")==0)
207   {
208     char c='1' + rand() %(('2' - '1') + 1);
209     char d='0' + rand() %(('9' - '0') + 1);
210     char e='0' + rand() %(('9' - '0') + 1);
211     UART0_print("REA 1 3 5 ");
212     UART0_putch(c);
213     UART0_putch(d);
214     UART0_putch('.');
215     UART0_putch(e);
216     UART0_print("\r\n");
217   }
218   else if (strcmp(T, "REA 1 3 6")==0)
219   {
220     char c='1' + rand() %(('2' - '1') + 1);
221     char d='0' + rand() %(('9' - '0') + 1);
222     char e='0' + rand() %(('9' - '0') + 1);
223     UART0_print("REA 1 3 6 ");
224     UART0_putch(c);
225     UART0_putch(d);
226     UART0_putch('.');
227     UART0_putch(e);
228     UART0_print("\r\n");
229   }
230   else if (strcmp(T, "REA 1 3 7")==0)
231     UART0_print("REA 1 3 7 90\r\n");
232   else if (strcmp(T, "REA 1 3 8")==0)
233     UART0_print("REA 1 3 8 70\r\n");
234   else if (strcmp(T, "REA 1 3 9")==0)
235   {
236     char c='0' + rand() %(('9' - '0') + 1);
237     char d='0' + rand() %(('9' - '0') + 1);
238     char e='0' + rand() %(('9' - '0') + 1);
239     UART0_print("REA 1 3 9 ");
240     UART0_putch('9');
241     UART0_putch(c);
242     UART0_putch(d);
243     UART0_putch('.');
244     UART0_putch(e);
```

```
245       UART0_print("\r\n");
246     }
247     else if (strcmp(T, "REA 1 3 10")==0)
248     {
249       char c='0' + rand() %(('9' - '0') + 1);
250       char d='0' + rand() %(('9' - '0') + 1);
251       char e='0' + rand() %(('9' - '0') + 1);
252       UART0_print("REA 1 3 10 ");
253       UART0_putch(c);
254       UART0_putch(d);
255       UART0_putch('.');
256       UART0_putch(e);
257       UART0_print("\r\n");
258     }
259     else if (strcmp(T, "REA 1 3 11")==0)
260       UART0_print("REA 1 3 11 22.3\r\n");
261     else if (strcmp(T, "REA 1 3 12")==0)
262     {
263       char c='0' + rand() %(('9' - '0') + 1);
264       char d='0' + rand() %(('9' - '0') + 1);
265       char e='0' + rand() %(('9' - '0') + 1);
266       UART0_print("REA 1 3 10 ");
267       UART0_putch(c);
268       UART0_putch(d);
269       UART0_putch('.');
270       UART0_putch(e);
271       UART0_print("\r\n");
272     }
273   //Level0Registers
274     //[20]TemperatureSettings
275     else if (strcmp(T, "REA 1 20 0")==0)
276       UART0_print("REA 1 20 0 0\r\n");
277     else if (strcmp(T, "REA 1 20 1")==0)
278       UART0_print("REA 1 20 1 240\r\n");
279     else if (strcmp(T, "REA 1 20 2")==0)
280       UART0_print("REA 1 20 2 200000\r\n");
281     else if (strcmp(T, "REA 1 20 3")==0)
282       UART0_print("REA 1 20 4700000\r\n");
283     else if (strcmp(T, "REA 1 20 4")==0)
284       UART0_print("REA 1 20 25000\r\n");
285     else if (strcmp(T, "REA 1 20 5")==0)
286       UART0_print("REA 1 20 5 400000\r\n");
287     else if (strcmp(T, "REA 1 20 6")==0)
288       UART0_print("REA 1 20 6 0\r\n");
289   //Level0Registers
290     //[30]SettingsStatus
291     else if (strcmp(T, "REA 1 30 0")==0)
292       UART0_print("REA 1 30 0 X\r\n");
293     else if (strcmp(T, "REA 1 30 1")==0)
294       UART0_print("REA 1 30 1 0\r\n");
295     else if (strcmp(T, "REA 1 30 2")==0)
296       UART0_print("REA 1 30 2 53000\r\n");
```

```
297   else if (strcmp(T, "REA 1 30 3")==0)
298     UART0_print("REA 1 30 3 20000\r\n");
299   else if (strcmp(T, "REA 1 30 4")==0)
300     UART0_print("REA 1 30 4 0\r\n");
301   else if (strcmp(T, "REA 1 30 5")==0)
302     UART0_print("REA 1 30 5 0\r\n");
303   else if (strcmp(T, "REA 1 30 6")==0)
304     UART0_print("REA 1 30 6 0\r\n");
305   else if (strcmp(T, "REA 1 30 7")==0)
306     UART0_print("REA 1 30 7 0\r\n");
307   else if (strcmp(T, "REA 1 30 8")==0)
308     UART0_print("REA 1 30 8 0\r\n");
309   else if (strcmp(T, "REA 1 30 9")==0)
310     UART0_print("REA 1 30 9 0\r\n");
311   else if (strcmp(T, "REA 1 30 10")==0)
312     UART0_print("REA 1 30 10 0\r\n");
313 //Level0Registers
314   //[31]CalibrationStatus
315   else if (strcmp(T, "REA 1 31 0")==0)
316     UART0_print("REA 1 31 0 2019\r\n");
317   else if (strcmp(T, "REA 1 31 1")==0)
318     UART0_print("REA 1 31 1 2\r\n");
319   else if (strcmp(T, "REA 1 31 2")==0)
320     UART0_print("REA 1 31 2 4\r\n");
321   else if (strcmp(T, "REA 1 31 3")==0)
322     UART0_print("REA 1 31 3 2019\r\n");
323   else if (strcmp(T, "REA 1 31 4")==0)
324     UART0_print("REA 1 31 4 2\r\n");
325   else if (strcmp(T, "REA 1 31 5")==0)
326     UART0_print("REA 1 31 5 4\r\n");
327   else if (strcmp(T, "REA 1 31 6")==0)
328     UART0_print("REA 1 31 6 0\r\n");
329 //Level0Registers
330   //[32]LoggingStatus
331   else if (strcmp(T, "REA 1 32 0")==0)
332     UART0_print("REA 1 32 0 0\r\n");
333   else if (strcmp(T, "REA 1 32 3")==0)
334     UART0_print("REA 1 32 3 1\r\n");
335   else if (strcmp(T, "REA 1 32 4")==0)
336     UART0_print("REA 1 32 4 3\r\n");
337   else if (strcmp(T, "REA 1 32 5")==0)
338     UART0_print("REA 1 32 5 18\r\n");
339   else if (strcmp(T, "REA 1 32 6")==0)
340     UART0_print("REA 1 32 6 0\r\n");
341 //Level0Registers
342   //[33]OptionsStatus
343   else if (strcmp(T, "REA 1 33 0")==0)
344     UART0_print("REA 1 32 0 2\r\n");
345   else if (strcmp(T, "REA 1 33 1")==0)
346     UART0_print("REA 1 33 2 300\r\n");
347 //Level0Registers
348   //[100-125]SensorConstants
```

```
349    else if (strcmp(T, "REA 1 123 0")==0)
350      UART0_print("REA 1 123 0 850\r\n");
351    else if (strcmp(T, "REA 1 123 1")==0)
352      UART0_print("REA 1 123 1 81\r\n");
353    else if (strcmp(T, "REA 1 123 2")==0)
354      UART0_print("REA 1 123 2 4000\r\n");
355    else if (strcmp(T, "REA 1 123 3")==0)
356      UART0_print("REA 1 123 3 -70\r\n");
357    else if (strcmp(T, "REA 1 123 4")==0)
358      UART0_print("REA 1 123 4 1184\r\n");
359    else if (strcmp(T, "REA 1 123 5")==0)
360      UART0_print("REA 1 123 5 0\r\n");
361    else if (strcmp(T, "REA 1 123 6")==0)
362      UART0_print("REA 1 123 6 0\r\n");
363    else if (strcmp(T, "REA 1 123 7")==0)
364      UART0_print("REA 1 123 7 0\r\n");
365    else if (strcmp(T, "REA 1 123 8")==0)
366      UART0_print("REA 1 123 8 0\r\n");
367    else if (strcmp(T, "REA 1 123 9")==0)
368      UART0_print("REA 1 123 9 0\r\n");
369    else if (strcmp(T, "REA 1 123 10")==0)
370      UART0_print("REA 1 123 10 -370\r\n");
371    else if (strcmp(T, "REA 1 123 11")==0)
372      UART0_print("REA 1 123 11 20950\r\n");
373    else if (strcmp(T, "REA 1 123 12")==0)
374      UART0_print("REA 1 123 12 46000\r\n");
375    else if (strcmp(T, "REA 1 123 13")==0)
376      UART0_print("REA 1 123 13 58000\r\n");
377    else if (strcmp(T, "REA 1 123 14")==0)
378      UART0_print("REA 1 123 14 15000\r\n");
379    else if (strcmp(T, "REA 1 123 15")==0)
380      UART0_print("REA 1 123 15 25000\r\n");
381    else if (strcmp(T, "REA 1 123 16")==0)
382      UART0_print("REA 1 123 16 5\r\n");
383    else if (strcmp(T, "REA 1 123 17")==0)
384      UART0_print("REA 1 123 17 0\r\n");
385
386     while((U1LSR & DR)) r = U1RBR; // Read data sent because RS485 half
        duplex
387
388
389  }
390 }
391 // ---------------------------------------------------------------------
```

Apéndice D

Makefile

```
# tools
TOOL  = arm-none-eabi
#TOOL = arm-linux
BT2 = ../bootloader2103/bt2.exe -c
#BT2  = ../bootloader2103/bt2

CC      = $(TOOL)-gcc
LD      = $(TOOL)-ld -v
AR      = $(TOOL)-ar
AS      = $(TOOL)-as
CP      = $(TOOL)-objcopy
OD  = $(TOOL)-objdump

# select the proper linker script
LNKSC = linker_flash.ld
#LNKSC  = linker_ram.ld
#LNKSC  = linker_dram.ld

CFLAGS  = -I./ -fno-common -O2 -g -mtune=arm7tdmi -nostartfiles -static -
    DVERSION=$(VCODE)
AFLAGS  = -ahls -mapcs-32
CPFLAGS = -O ihex
ODFLAGS = -x --syms

all:  code.bin code.hex
  $(BT2) -l code.bin

clean:
  -rm -f crt.lst main.lst crt.o main.o code.elf code.bin code.hex

code.bin: code.elf
  $(CP) -O binary $< $@

code.hex: code.elf
  $(CP) -O ihex $< $@

#code.elf: crt.S main.S linker_ram.ld Makefile
```

```
37 code.elf: crt.S main.c linker_ram.ld Makefile
38   $(CC) $(CFLAGS) -Wl,-T$(LNKSC) -o $@ crt.S main.c
39
40 ISP      = ../bootloader2103/lpc21isp.exe
41 USB_COM = $(shell .\detect_USB_COM.exe)
42 #USB_COM =\\.\COM12
43
44 burn: code.hex
45   $(ISP) -control -term code.hex $(USB_COM) 115200 14746
46 terminal:
47   $(ISP) -termonly code.hex $(USB_COM) 115200 14746
```

Bibliografía

[1] Arduino core for ESP8266 WiFi chip *https://github.com/esp8266/Arduino*

[2] USB Host Library Rev.2.0 *https://github.com/felis/USB_Host_Shield_2.0*

[3] Datasheet: ESP-12F WiFi Module. Version 1.0 *https://wiki.ai-thinker.com/_media/esp8266/a014ps01.pdf*

[4] Datasheet: USB Peripheral/Host Controller with SPI Interface *https://www.sparkfun.com/datasheets/DevTools/Arduino/MAX3421E.pdf*

[5] LPC2103 Education Board Users Guide *https://www.ele.uva.es/ jesus/hardware_empotrado/LPC2103_Education_Board_Users_Guide.pdf*

[6] Documentación de phpMyAdmin *https://docs.phpmyadmin.net/es/latest/*

[7] Conoce Android Studio *https://developer.android.com/studio/intro/index.html?hl=es-419*

[8] ESP8266, LA ALTERNATIVA A ARDUINO CON WIFI *https://www.luisllamas.es/esp8266/*

[9] Aprendiendo a diseñar en Tinkercad *http://diwo.bq.com/aprendiendo-a-disenar-en-tinkercad/*

[10] Pocket Oxygen Meter FireStingGO2 *https://www.pyroscience.com/media/files/FSGO2/Manual20FireStingGO2.pdf*