ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

MÓDULO DE COMUNICACIONES INALÁMBRICO PARA EL TEST TÉRMICO DE SISTEMAS ELECTRÓNICOS

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Sistemas Electrónicos

Autor/a: Irene Merino Fernández

Tutor/a: Aurelio Vega Martínez

Fecha: Noviembre 2020

AGRADECIMIENTOS

Primeramente, quiero hacer una mención especial a mi tutor, Aurelio Vega Martínez, sin el cual todo esto no habría sido posible, más con las circunstancias del confinamiento que han ocurrido durante la realización de este proyecto, finalizando de manera satisfactoria gracias a su dedicación y esfuerzo.

Continuando, me gustaría agradecer al soporte del Instituto de Microelectrónica Aplicada, que han estado pendientes del correcto funcionamiento de los equipos durante el desarrollo del proyecto, además de mis compañeros de laboratorio, los cuales en algún punto del desarrollo del proyecto han colaborado en el éxito de este.

Para terminar, pero no menos importante, acentuar el apoyo más importante que he disfrutado durante estos años, mi familia, sobre todo mi madre. Quiero agradecerle el esfuerzo y la paciencia constante para que yo haya podido finalizar esta etapa de mis estudios.

RESUMEN

El objeto de este trabajo es la realización de un sistema de test térmico de sistemas electrónicos que haga uso de la comunicación Bluetooth. La plataforma elegida va a implementar la comunicación desde un PC hacia el sistema de medida de temperatura. El objetivo es que laboratorios, o talleres de reparación, puedan obtener la imagen térmica de los sistemas electrónicos que se encuentren bajo observación, utilizando dispositivos de muy bajo coste. La comunicación inalámbrica nos facilitará la colocación de los dispositivos de lectura en el cabezal de cualquier máquina tipo plotter xy sin necesidad de instalar cableado.

Inicialmente, se ha realizado un estudio de los sensores de temperatura que podríamos utilizar para este proyecto, dando datos de precisión y precio, además del funcionamiento interno. También se ha estudiado las plataformas propuestas, STM32WB55 y ESP32, y los programas más recomendados, que nos ayudarán a desarrollar el software.

Posteriormente, se ha dado una explicación del diseño de las diferentes PCB necesarias, para trabajar con los sensores y con las placas, haciendo uso de programas como Altium Designer o CircuitCam 5.0. Se ha abordado también su fabricación. Para la implementación del software hemos hecho pruebas sobre las dos plataformas. Hemos usado, para el núcleo de STM32WB55, el entorno STM32Cube, que incluye tres programas: STM32CubeMX, para asignar los pines, STM32CubeIDE, para desarrollar el código, y STM32CubeProgrammer, para cargar el programa a la placa. En el caso del dispositivo ESP32, hemos usado el entorno de Arduino.

Para finalizar, se ha realizado un banco de pruebas para verificar el funcionamiento del sistema electrónico, primero sobre los dispositivos, y posteriormente sobre la máquina CNC que realizará la imagen térmica. Asimismo, se han dado las pertinentes conclusiones sobre la realización de este proyecto.

ABSTRACT

The purpose of this work is to carry out a thermal test system for electronic systems that makes use of Bluetooth communication. The chosen platform will implement communication from a PC to the temperature measurement system. The objective is that laboratories, or repair workshops, can obtain the thermal image of the electronic systems that are under observation, using very low-cost devices. Wireless communication will facilitate the placement of reading devices on the head of any xy plotter type machine without installing wiring.

Initially, we made a study of temperature sensors that we can use for this project, assessing information about precision, price and how they work. Moreover, we studied the platforms STM32WB55 and ESP32, as well as the most recommended programs, which enables us to develop software.

Subsequently, we explained the design of the different PCB that we need, to work with the sensors and both platforms, using programs such as Altium Designer and CircuitCam 5.0. Additionally, we dealt with their production.

For implementing software, we tested both platforms. With STM32WB55 platform we used STM32Cube environment, which includes three programs: STM32CubeMX, to assign microprocessor pins, STM32CubeIDE, to develop code and STM32CubeProgrammer, to upload our program to the board. In case of ESP32, we used Arduino.

As a final stage, a testbench was carried out in order to check the proper functioning of the electronic system, as well as the CNC machine which makes the thermal imaging. This information will be analysed in order to draw the necessary conclusions to the project.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

3D	*Three-dimensional*
ADC	*Analogic/Digital Converter*
AFR	*Adaptative Frequency Hopping*
AHB	*Advanced High-performance Bus*
AMBA	*Advanced Microcontroller Bus Architecture*
APB	*Advanced Peripheral Bus.*
ARM	*Advanced RISC Machine*
ATT	*Attribute Protocol*
AXI	*Advanced Extensible Interface*
BLE	*Bluetooth Low Energy*
BR	*Basic Rate*
CAD	*Computer Aided Design*
CAM	*Computer Aided Manufacturing*
CAN	*Controller Area Network*
CNC	*Control numérico por computadora*
CPU	*Control Processing Unit*
COM	*Communication Port*
CVSD	*Continuously Variable Shape Delta*
DAC	*Digital/Analogic Converter*
DFU	*Device Firmware Update*
DMA	*Direct Memory Access*
DSP	*Digital Signal Processor*
EDR	*Enhanced Data Rate*
EEPROM	*Electrically Erasable Programable Read-Only Memory*
GAP	*Generic Access Profile*
GATT	*General Attribute Profile*
GPIO	*General-Purpose Inputs/Outputs*
HAL	*Hardware Abstraction Layer*
HF	*High Frequency*
I^2C	*Inter Integrated Circuit*
IC	*Integrated Circuit*

IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IF	*Intermediate Frequency*
IoT	*Internet of Things*
IR	*Infrarred Remote Control*
I^2S	*Inter Integrated Sound*
IUMA	*Instituto Universitario de Microelectrónica Aplicada*
JTAG	*Join Test Action Group*
LCD	*Liquid Crystal Display*
LED	*Light Emitting Diode*
LPUART	*Low Power Universal Asynchronous Receiver-Transmitter*
LSE	*Low Surface Energy*
MAC	*Media Access Control*
MSL	*Moisture Sensitivity Level*
NZIF	*Near Zero Intermediate Frequency*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PCROP	*Proprietary Code Read Out Protection*
PEC	*Packet Error Checking*
PWM	*Pulse Width Modulated*
RAM	*Random Access Memory*
RF	*Radio Frequency*
ROM	*Read Only Memory*
RTD	*Resistance Temperature Detector*
SBC	*Low Complexity Subband Control*
SCL	*Serial Clock Input*
SD	*Secure Digital*
SDA	*Serial Data*
SDIO	*Secure Digital Input/Output*
SIG	*Special Interest Group*
SMA	*Sub Miniature version A*
SMD	*Surface-Mount Device*
SMBus	*System Management Bus*

SMPS	*Switched-Mode Power Supply*
SPI	*Serial Peripheral Interface*
SWD	*Serial Write Debug*
SWV	*Serial Wire Viewer*
TSMC	*Taiwan Semiconductor Manufacturing Company*
UART	*Universal Asynchronous Receiver-Transmitter*
UUID	*Universally Unique Identifier*
USART	*Universal Synchronous/Asynchronous Receiver-Transmitter*
USB	*Universal Serial Bus*

Capítulo 1. Introducción

En este primer capítulo, vamos a presentar los motivos y antecedentes que generan la necesidad del desarrollo de este proyecto. También expondremos nuestros objetivos, y definiremos la estructura de nuestra memoria.

1.1 Antecedentes

Con el crecimiento actual de la tecnología, cada vez se hace más difícil las tareas de mantenimiento y detección de fallos de funcionamiento en los sistemas electrónicos, pues los circuitos integrados gradualmente van siendo más complejos.

Con este crecimiento, el método tradicional de medición por contacto para realizar el diagnóstico es complicado de adaptar al alto nivel de densidad de integración actual [1], pudiendo provocar errores en el diagnóstico de fallas y la identificación de su localización, lo que podría derivar en la desviación de los resultados esperados. Los errores más comunes suelen ocurrir cuando el dispositivo está en modo de operación. Las causas más comunes son conexiones de altas resistencias, cables rotos o desconectados o partes defectuosas, cuyo resultado la mayoría de las veces resulta en un calentamiento de los circuitos integrados, provocando que la electrónica falle. Por ello, debemos usar un método que sea preciso, rápido y no destructivo para detectar si hay algún tipo de fallo [2]. Se han desarrollado diferentes técnicas con estas características, como la realización de la imagen óptica, la imagen de rayos X, el mapeo de campo magnético, y la imagen térmica. Entre todos estos métodos, esta última puede de manera significativa detectar rupturas en conexiones internas, cortocircuitos, circuitos abiertos, fallos en la alimentación o en la interacción de señales, o un mal funcionamiento de algún componente [3].

Además, es una solución que presenta las ventajas de ser una solución sin contacto, de rápida adquisición de la imagen, de sencilla operación y fácil reconfiguración del patrón de test [3]. Por lo tanto, se introduce la cámara de infrarrojos térmica como una solución de adquisición de imagen térmica, la cual nos provee una idea muy intuitiva de los resultados [4].

La imagen térmica resulta muy útil en el proceso de diseño y testeo de una PCB. La imagen térmica muestra la disipación de potencia en una PCB. El consumo de energía se asocia con el paso de los electrones a través de los dispositivos semiconductores provocando el aumento de las características térmicas de los circuitos integrados con respecto al patrón de test. Esta característica puede ser capturada por una cámara infrarroja, formando así una imagen térmica [5].

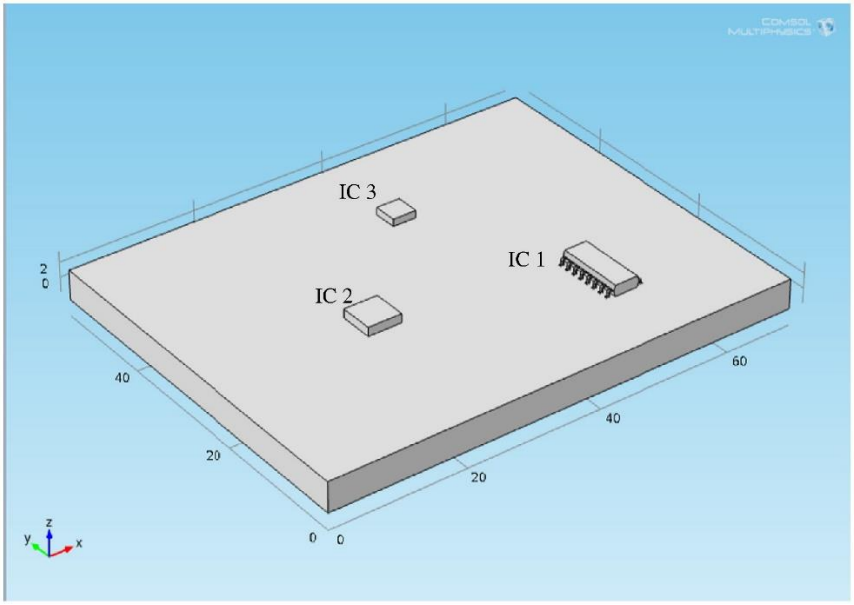


Figura 1. Ejemplo PCB con tres circuitos integrados [6].

En la Figura 1 tenemos un ejemplo de PCB, sobre la cual se realizará una simulación de imagen térmica en diferentes condiciones de potencia y temperatura, en función de cada circuito integrado. La disipación de potencia en el modelo IC1 fue cambiada entre los valores de 0,6 y 1,14 W, con el fin de obtener un rango de temperatura de 34,05ºC y 43,26ºC. Sobre el modelo IC2 se altera la disipación de potencia entre 1 y 2,33 W, logrando un rango de temperatura de entre 29,73ºC y 43,25ºC. Por último, la variación realizada en IC3 es de entre 0,6 y 1,5 W, consiguiendo temperaturas de entre 30,33ºC y 43,7ºC. Así, obtenemos la Figura 2, en la que presentamos el gradiente de las superficies de cada circuito integrado. Las regiones más calientes se marcan en rojo, mientras que otras regiones de la PCB, marcadas en azul, son más frías que los circuitos integrados [6].

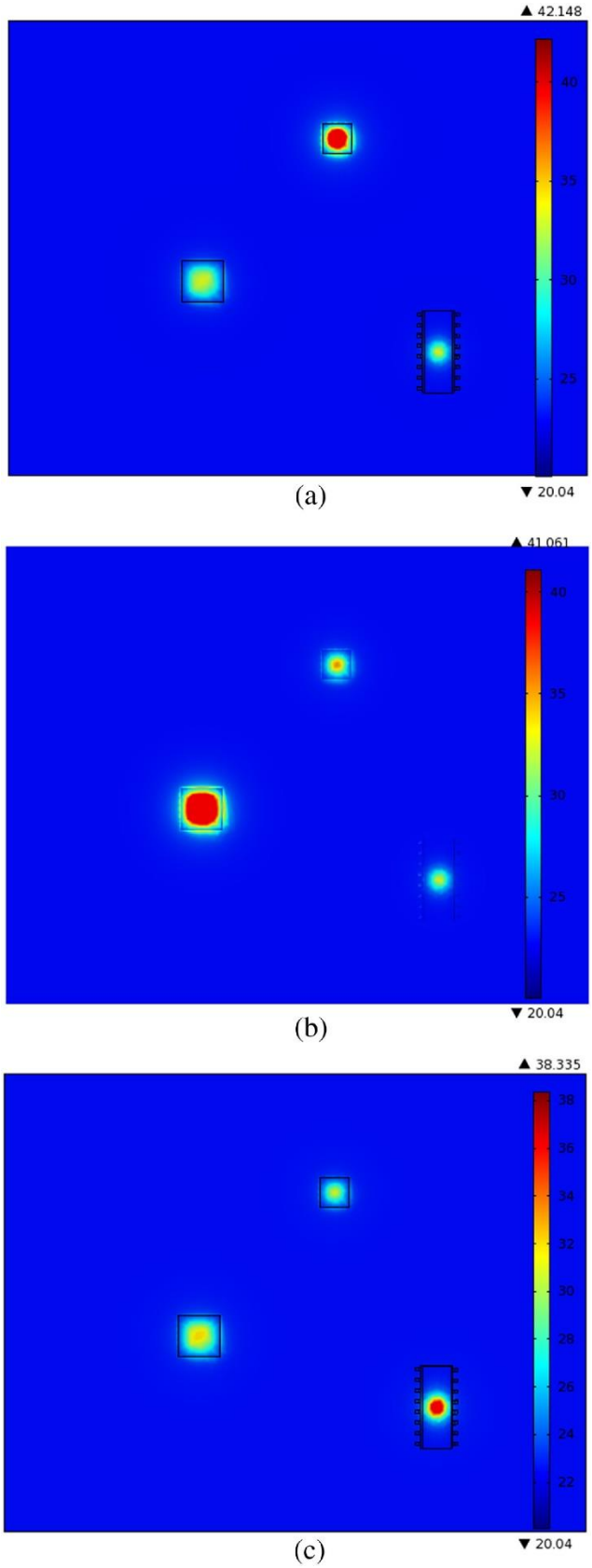


Figura 2. Ejemplos de distribución de temperatura en la capa superior de una PCB [6].

El método de detección de fallos basado en la imagen térmica cuenta con tres etapas básicas [7]:

1. Identificación de la fuente de calor.
2. Extracción de sus características.
3. Reconocimiento del patrón térmico.

Es importante abordar los tres puntos con claridad, pues afectará directamente a la precisión en la detección de errores.

Actualmente, las cámaras térmicas de infrarrojos de cierta calidad tienen un precio muy elevado [8]. Si consideráramos la tecnología de cámaras termográficas FLIR, veremos que las cámaras de montaje fijo oscilan entre 1.000 a 10.000€ [8], precio lo suficientemente alto como para buscar una alternativa. En la Figura 3 vemos la cámara térmica ETS320 de FLIR, de precio intermedio [9], usada para comprobar componentes electrónicos, algo similar a lo que realizaremos a lo largo de este proyecto.

Figura 3. Cámara termográfica FLIR [9].

De esta forma nace la necesidad de explorar nuevas soluciones que nos permitan detectar los mismos errores, pero a un menor costo. Muchas veces no es necesaria una gran resolución a la hora de la realización de la imagen, pues si podemos identificar la fuente de calor con exactitud, sabremos donde está fallando el circuito.

1.2 Objetivos

Este trabajo de fin de título consiste en generar una solución de bajo coste que se pueda acoplar a cualquier máquina CNC para realizar lecturas térmicas puntuales. Para ello, haremos uso de sensores térmicos para la construcción de una imagen en baja resolución de la PCB, centrándonos sobre todo en la temperatura de los componentes principales, lo que lo convierte en una solución más accesible.

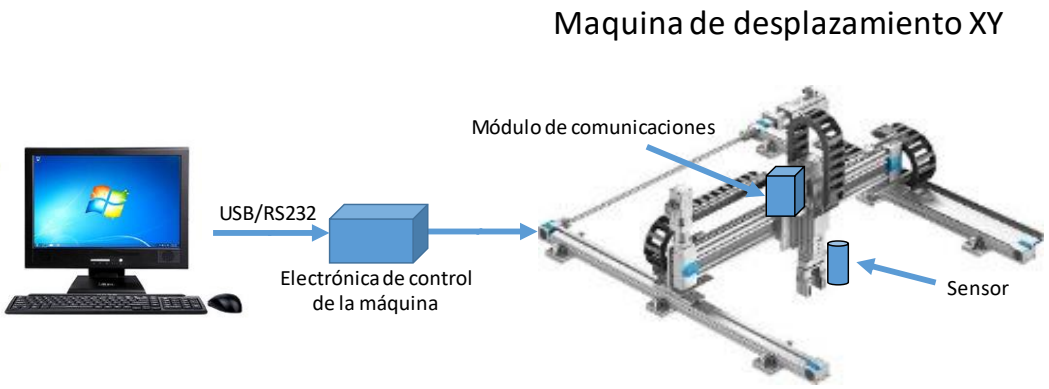


Figura 4. Sistema de movimiento XY.

En la Figura 4, podemos ver cómo sería el sistema de desplazamiento XY propuesto para el TFT. El ordenador establecerá comunicaciones seriales USB (Universal Serial Bus) o RS232 con el sistema de control de la máquina, que se encargará de mover de forma adecuada los motores. Para la gestión del movimiento del cabezal de la máquina se utilizará el software habitual de la máquina, por lo que no será objeto de nuestro TFT. Nuestro módulo de comunicaciones se colocará en cualquier posición disponible del cabezal de la máquina, de forma que se moverá de forma solidaria con él. La ubicación de los

sensores será también solidaria con el cabezal, pero en la parte inferior, de forma que estén próximos a la electrónica bajo test que se encontrará debajo.

Principalmente, se ha realizado un estudio de dos plataformas. Por un lado, el pack STM32WB55, que consta de una placa núcleo y un USB, que se pueden comunicar de manera inalámbrica. Por otro lado, tenemos el ESP32, que también cuenta con transmisión sin cableado.

Los objetivos propuestos para el desarrollo del TFT son los siguientes:

1. Diseño e implementación de un módulo inalámbrico de medida de temperatura con sensores IR. El diseño incluye el desarrollo del hardware y firmware necesario para la lectura de sensores IR, y las comunicaciones con un ordenador que actúe como host del sistema.
2. Pruebas de funcionamiento sobre el cabezal de una máquina fresadora comercial que permita que el conjunto funcione como un sistema de inspección térmica automática.
3. Evaluar y documentar el trabajo realizado.

1.3 Peticionario

La realización de este trabajo se realiza para el Laboratorio de Fabricación de Prototipos y Sistemas Electrónicos perteneciente al Instituto Universitario de Microelectrónica Aplicada (IUMA), instituto de investigación referente a la Universidad de Las Palmas de Gran Canaria.

Desarrollaremos este sistema electrónico con el fin de incluirlo en la sección de diagnóstico en las PCB, pues en el laboratorio se realizan reparaciones para diferentes empresas, así mejorando la eficiencia y la velocidad en la detección de errores en las mismas.

Por otro lado, hay que añadir como solicitante del proyecto a la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE), para así cubrir los créditos y requerimientos de la asignatura Trabajo de Fin de Grado, contemplada en el plan de estudios del Grado en Ingenierías en Tecnologías de la Telecomunicación.

1.4 Estructura del documento

En este primer capítulo hemos realizado un recorrido por los antecedentes que preceden la realización de este trabajo, enumerando los objetivos que se pretenden alcanzar, los solicitantes de este, y la propia estructura. En el capítulo 2 realizaremos un estudio sobre los posibles sensores a utilizar. En el capítulo 3 haremos un estudio completo de la plataforma STM32WB55, mientras que en el capítulo 4 lo haremos del ESP32. El flujo de desarrollo de ambas se trabajará en el capítulo 5.

En el capítulo 6 hablaremos del diseño y fabricación de la PCB para los sensores y el Shield con su circuitería. En el capítulo 7 trataremos la integración en nuestra plataforma.

En el capítulo 8 describiremos los resultados obtenidos de las pruebas realizadas. Para finalizar, en el capítulo 9 trataremos las conclusiones obtenidas tras la realización de este trabajo.

Capítulo 2. Tecnologías del proyecto

2.1 Introducción

Los sensores de temperatura son dispositivos que registran cambios de temperatura, transformándolos en señales eléctricas, posteriormente procesadas por equipo electrónico. Hay varias formas de obtener esta temperatura. A grandes rasgos, tenemos los siguientes tipos de sensores:

Por un lado, tenemos los sensores RTD. La resistencia del metal aumenta con el incremento de la temperatura. Si medimos la resistencia de un cable con la longitud, diámetro y composición conocidos podremos determinar su temperatura. Por otro lado, tenemos los termistores, cuya composición se basa en semiconductores, y su funcionamiento lo marca el comportamiento frente a cambios de temperatura de estos. La resistencia de un termistor disminuye con la temperatura, y este cambio es relativamente menor que con el anterior método, que lo hace más apropiado para cambios menores. Además, tenemos los termopares, basados en el efecto termoeléctrico. Se basa en dos nodos de diferentes metales sometidos a dos temperaturas diferentes. El nodo frío se sitúa en la temperatura referencia, como podría ser la temperatura del instrumento de medida. El nodo caliente se coloca en el objeto de medida de temperatura. Bajo estas condiciones se genera una diferencia de potencial, el cual mediremos para dar una lectura, proporcional a la diferencia de temperatura. Por último, los sensores infrarrojos, los cuales tienen ventajas sobre los anteriores, como no necesitar contacto para las medidas ya que se basa en la emisión infrarroja de los objetos para dar un valor de la temperatura de estos [14].

Hemos realizado un estudio sobre diferentes sensores de temperatura, los cuales describiremos en los siguientes apartados.

Muchos sensores utilizan el protocolo de I^2C, así que también daremos una descripción del protocolo. Además, explicaremos cómo funcionan las comunicaciones BLE, protocolo que utilizaremos en el presente proyecto.

2.2 Sensores gama Melexis MLX90614.

Sensores térmicos con tecnología infrarroja para medidas sin contacto [15]. Este sensor se forma de dos chips desarrollados y fabricados por Melexis:

- Chip detector de pila termoeléctrica infrarrojo MLX81101. Es un dispositivo con varios termopares conectados en serie, los cuales tienen en común tanto los pines que miden la temperatura de referencia, como los pines que miden la temperatura en ese momento. Esta forma de conexión ayuda a que haya más corriente que en un solo termopar, por lo que la resolución será mayor [14].
- Chip integrado de procesamiento de la señal MLX90302. Este chip se forma de un amplificador de bajo ruido, un conversor analógico digital de 17 bits de alta resolución, y una unidad DSP. Las temperaturas del objeto y ambiente están disponibles en la RAM de este chip con una resolución de 0,01ºC. Podemos acceder a ellos con la interfaz SMBus, o a través de la salida PWM de 10 bit.

Figura 5. Sensores gama MLX81101 con diferentes encapsulados.

En la Figura 5 podemos distinguir diferentes sensores de la gama, primero el designado con ACC, seguido del sensor llamado con BCF, y por último el sensor AAA, cuyas características se tratarán a lo largo de este capítulo, ya que serán objeto de nuestro estudio.

En fábrica se realiza una calibración con los siguientes rangos de temperatura:

- [-40ºC,125ºC] para la temperatura ambiente.
- [-70ºC, 380ºC] para la temperatura del objeto.

Una característica común a todos los sensores es la resolución con la que se nos presentarán los datos, que será de 0,14ºC.

En la Tabla I se reflejan los sensores de esta gama que vamos a estudiar en nuestra aplicación. Los datos dados son los siguientes: la referencia del fabricante, el campo de visión en grados, el área de visión en centímetros, la precisión en grados centígrados y el precio en euros.

El campo de visión se refiere al ángulo que incidirá sobre el objeto donde realizaremos la medida. Por otro lado, el área de visión corresponde al cálculo de la distancia que abarcará el sensor por cada centímetro alejado. Para este cálculo utilizaremos el ángulo de incidencia anterior.

En la precisión mostramos el error en el que trabaja el sensor, en función de dos intervalos, la temperatura ambiente y la temperatura del objeto respectivamente, en grados centígrados.

El precio de la tabla se ha obtenido de la web de DigiKey (www.digikey.com).

Tabla I. Resumen sensores Melexis.

<table>
<tr><th colspan="8">SENSORES IR</th></tr>
<tr><th colspan="8">FAMILIA MELEXIS</th></tr>
<tr><th rowspan="2">REFERENCIA</th><th rowspan="2">CAMPO DE VISIÓN</th><th rowspan="2">ÁREA VISIÓN</th><th colspan="4">PRECISIÓN</th><th rowspan="2">PRECIO</th></tr>
<tr><th>0-50, 0-60</th><th>50-100, 0-60</th><th>0-50, 60-120</th><th>50-100, 60-120</th></tr>
<tr><td>MLX90614ESF-AAA-000-TU</td><td>90</td><td>1:2</td><td>±0,75</td><td>±2</td><td>±2</td><td>±2</td><td>12,94</td></tr>
<tr><td>MLX90614ESF-ACC-000-TU</td><td>35</td><td>1:0.6306</td><td>±0,75</td><td>±2</td><td>±2</td><td>±2</td><td>18,11</td></tr>
<tr><td>MLX90614ESF-BCF-000-TU</td><td>10</td><td>1:0.175</td><td>±0,75</td><td>±2</td><td>±2</td><td>±2</td><td>24,39</td></tr>
<tr><td rowspan="3">MLX90614ESF-DCC-000-TU</td><td rowspan="3">35</td><td rowspan="3">1:0.6306</td><td>±0,75*</td><td>±2</td><td>±2</td><td>±2</td><td rowspan="3">18,2</td></tr>
<tr><td rowspan="2">*Los siguientes rangos tienen una mejor precisión:</td><td>16-40, 22-36</td><td>16-40, 36-38</td><td>16-40, 38-40</td></tr>
<tr><td>±0,3</td><td>±0,2</td><td>±0,3</td></tr>
</table>

En la Figura 6 vemos la asignación de los pines, que será la misma en todos los sensores de la familia. En la Tabla II daremos una definición para estos pines.

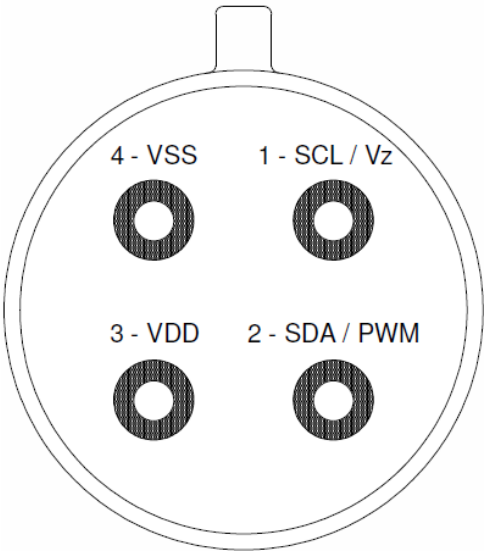


Figura 6. Descripción de los pines MLX90614 [15].

Tabla II. Definición de los pines MLX90614 [15].

Numero	Nombre del pin	Descripción
1	SCL/Vz	Entrada serial para el reloj, para interfaz SMBus. Tiene una conexión Zener de 5,7 V disponible para conectar un transistor bipolar externo para suministro del dispositivo para el modo de operación de 12V.
2	SDA/PWM	Es la entrada/salida digital. En el modo normal de operación la temperatura medida del objeto está disponible en este pin en forma de PWM. Si usamos el modo SMBus este pin será configurado automáticamente como un NMOS con drenaje abierto.
3	VDD	Alimentación
4	VSS	Tierra. El metal está conectado a este pin.

De los modos de operación que se nos ofrecen, el que escogeremos es SMBus, pues nuestra plataforma de desarrollo nos permite realizar comunicaciones a través de este protocolo, por lo que usaremos el pin 1 como SCL y el pin 2 como SDA.

La interfaz SMBus es un protocolo de dos cables, que permite la comunicación entre los dispositivos maestro y esclavo. En el sistema debe haber solo un maestro en cualquier tiempo. Además, el dispositivo MLX90614 solo podrá ser usado como esclavo.

Generalmente, el esquema de funcionamiento es el siguiente. El maestro comienza la transferencia de datos seleccionando un esclavo a través de la línea de direcciones de los esclavos. El maestro tiene permisos de lectura a la RAM y a la EEPROM, y además tiene permisos de escritura en 9 direcciones de la EEPROM (0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x0E, 0x0F, 0x19), teniendo en cuenta que la dirección 0x05 es de lectura y escritura. No debemos modificar su tercer bit, cancelaría la calibración de fábrica. En una operación de lectura se responderá con 16 bits de datos y 8 bits de PEC cuando la dirección del esclavo es la misma que la dirección puesta en la línea por el maestro.

La línea de direcciones de los esclavos permite conectar hasta 127 dispositivos, ocupando la memoria desde 0x00 hasta 0x07F en la EEPROM. No debemos conectar dos dispositivos iguales de esta gama en la misma dirección de esclavo.

Los comandos que soporta de la especificación estándar SMBus son: *Read Word* y *Write Word*.

A nivel de formación de trama, tenemos la típica propia del protocolo de comunicación I^2C. En la Figura 7 la mostraremos.

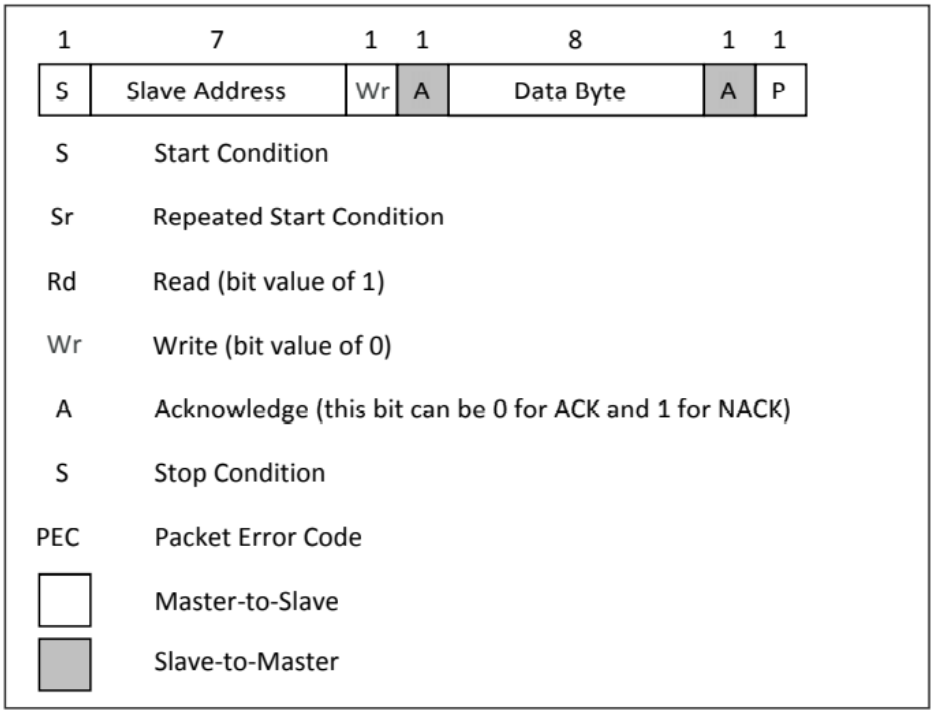


Figura 7. Trama del sensor [15].

Después de recibir 8 bits el esclavo debe realizar un reconocimiento a través de un bit ACK o NACK. Cuando el maestro comienza a comunicar, primero envía la dirección el esclavo, y únicamente el que la reconozca enviará un bit de reconocimiento, el resto permanecerá en silencio. En el caso de que el esclavo no reconozca uno de os bytes, el máster terminará la comunicación y repetirá el mensaje. Después del PEC podrá ser recibido un bit de NACK, que significará que hay un error en el mensaje recibido y el maestro deberá intentar enviar el mensaje de nuevo. El cálculo del PEC incluye todos los bits excepto los bits de START, REPEATED START, STOP, ACK y NACK. El bit más significativo de cada byte será transmitido primero.

En las Figura 8 yFigura 9 mostraremos los dos comandos posibles, de lectura y escritura de este sensor. Dependiendo del comando que introduzcamos, accederemos a la RAM o a la EEPROM, descritos en la Tabla III. Además, podemos acceder a la lectura de *flags*, o entrar en el modo de suspensión.

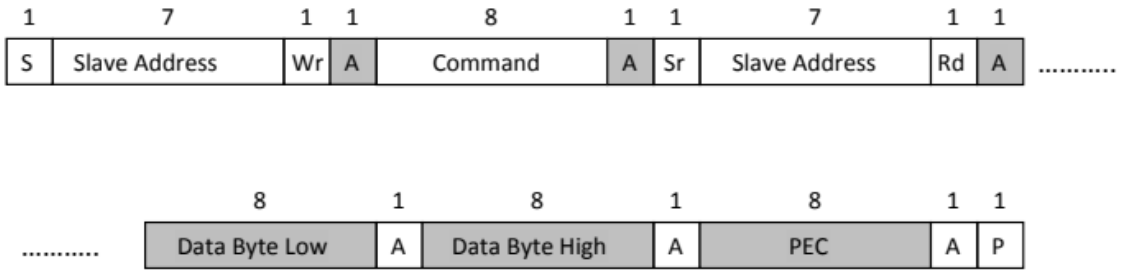


Figura 8. Formato de lectura para SMBus [15].

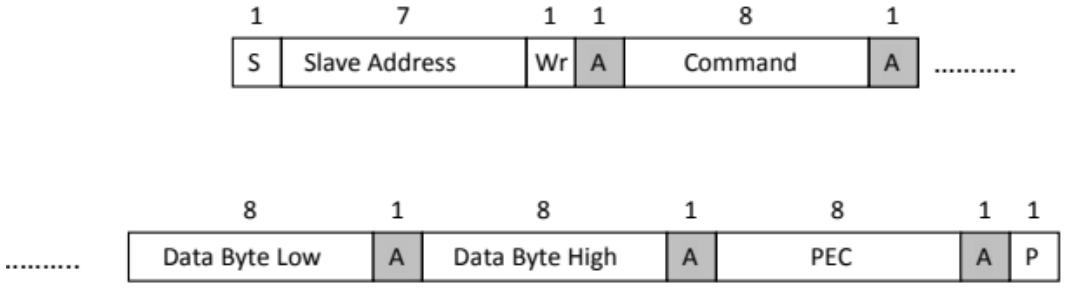


Figura 9. Formato de escritura para SMBus [15].

Tabla III. Comandos SMBus [15].

Opcode	Command
000x xxxx*	RAM Access
001x xxxx*	EEPROM Access
1111_0000**	Read Flags
1111_1111	Enter SLEEP mode

2.3 Matriz de sensores Adafruit AMG8833

Array de sensores térmicos de infrarrojos. Cuando está conectado a un microcontrolador nos devolverá un *array* de 64 lecturas de temperatura infrarrojas en el protocolo I^2C [16].

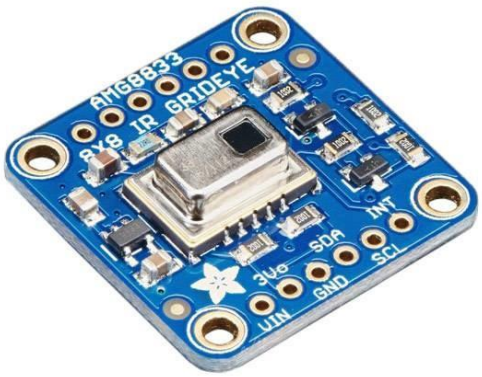


Figura 10. Matriz de sensores Adafruit AMG8833 [16].

En la Figura 10 vemos el PCB que contiene la matriz de sensores, además de su electrónica y circuitería adicional para su correcto funcionamiento.

Sus características son las siguientes:

- Mide en el rango de 0ºC a 80ºC con precisión de ±2,5ºC.
- Su resolución es de 0,25ºC
- Detecta objetos a una distancia menor a 7 metros.
- Tiene un ángulo de visión de 60º, lo que nos da un área de visión por centímetro alejado de aproximadamente 1,15 cm.

El chip está reflejado en la Figura 11, en la cual hemos señalado los pines, que definiremos en la Tabla IV.

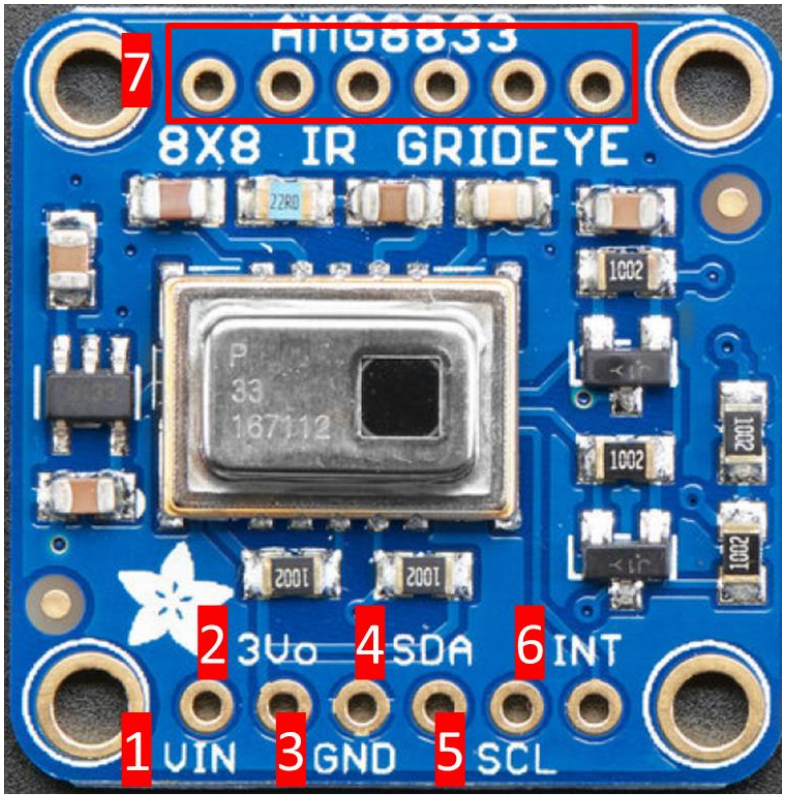


Figura 11. Descripción de pines chip Adafruit [16].

Tabla IV. Definición de los pines chip Adafruit [16].

Número	Nombre del pin	Definición
1	VIN	Pin de alimentación. El valor de la alimentación vendrá definido por el microcontrolador.
2	3Vo	Pin de salida de 3,3V, podemos tomar hasta 100mA.
3	GND	Tierra común, para lógica y alimentación.
4	SDA	Pin para conectar la línea de reloj de I^2C.
5	SCL	Pin para conectar la línea de datos de I^2C.
6	INT	Pin de interrupciones. Permite detectar cuando algo se mueve o cambia en el rango del sensor. Activa a bajo nivel.
7	-	Pines de estabilidad. No se conectan a nada.

En nuestro caso hemos usado la versión de SparkFun, basada en el mismo *array* de sensores, por lo que las características de temperatura y distancia de detección son las mismas que hemos mencionado para el chip de Adafruit. En esta versión, la temperatura tolerada es de 3,3V, el fabricante nos recomienda no intentar operar con 5V [17].

Figura 12. Chip SparkFun [17].

En la Figura 12 mostramos el chip, similar al chip de Adafruit, pero con la entrada de alimentación fijada a 3,3 V. Nos indica también en la parte posterior la dirección del I^2C, por defecto 0x69. Tiene también un jumper que podemos soldar justo debajo, en el caso de cerrarlo la dirección será 0x68 [18]. Esto es útil en el caso de que solo tengamos una línea de conexionado para I^2C, podremos usar dos *array* de este tipo, pues no habrá conflicto con las direcciones. En la Tabla V describimos cada uno de los pines del chip.

Tabla V. Pines chip SparkFun [18].

Nombre del pin	Definición
GND	Tierra común, para lógica y alimentación.
3,3V	Alimentación del chip. Debería trabajar en el intervalo 1,95 – 3,6 V
SDA	Pin para conectar la línea de reloj de I^2C.
SCL	Pin para conectar la línea de datos de I^2C.
INT	Pin de interrupciones, salida digital.

Este módulo también incluye dos conectores Qwiic, ecosistema de inter conexionado de I^2C para diversos productos, como sensores y actuadores. Este sistema hace el prototipado más rápido, disminuyendo también la probabilidad de error [19]. En la Figura 13 vemos cómo se conectaría el *array* a una placa usando este tipo de conexión.

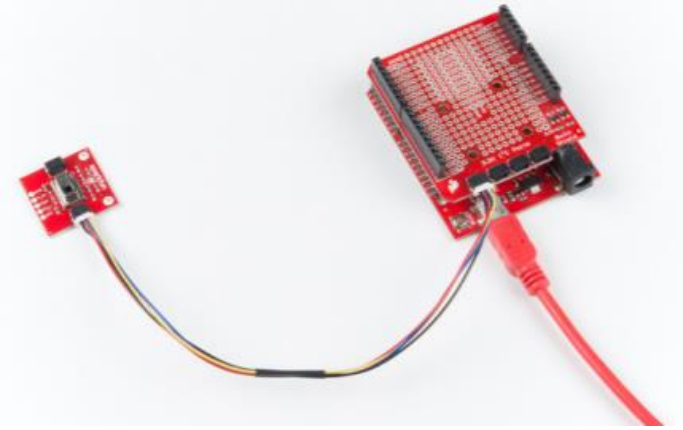


Figura 13. Ejemplo conexiones Qwiic [18].

Durante el desarrollo del proyecto hemos comprado la cámara térmica infrarroja KKMOON, basada en este *array* de sensores. Combina las funciones de medición de temperatura superficial y la imagen térmica en tiempo real [20]. En la Figura 14 vemos este dispositivo.

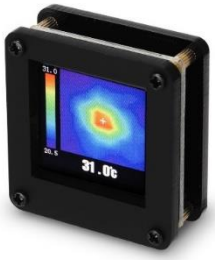


Figura 14. Cámara térmica KKMOON [20].

Esta electrónica es útil a la hora de trabajar a mano con ella, pues podemos ir viendo en tiempo real la temperatura del objeto al que apuntamos. Sin embargo, no podemos sacar datos de ellas, por lo que precisamos de un diseño a medida en la que podamos obtener la información de las imágenes para el software de control.

2.4 Bus I^2C

Bus desarrollado por Philips a finales de los años 70, principalmente pensado para productos de consumo. En la actualidad es un estándar usado por todos los fabricantes de circuitos integrados por las ventajas que presenta, pues elimina un gran número de conexiones en la PCB y reduce el número de pines en los chips. Ahorra conexionado entre los circuitos integrados, y tiene un diseño modular [10].

Este bus es de dos hilos, SDA y SCL (no consideramos GND como línea de protocolo). La comunicación es bidireccional entre maestro y esclavo, y podemos tener múltiples de ambos.

En cuanto al protocolo, el direccionamiento se realiza por software, teniendo dos formatos, el de 7 bits, que puede direccionar hasta 111 dispositivos, teniendo reservadas algunas direcciones. El fabricante fija 4 bits y el usuario programa por hardware los tres restantes, correspondientes a los primeros bits. El otro formato disponible es el de 10 bits, que puede direccionar hasta 1024 dispositivos. Consiste en un comando fijo que consta de 4 bits, seguido de una dirección de 10 bits, que puede ser fija o seleccionable por hardware [10].

Las comunicaciones de este protocolo comienzan con un bit de START, seguido por la dirección del esclavo. Posteriormente indicamos en un bit la lectura o escritura, que será validado con un bit de ACK, como cada uno de los datos transmitidos o recibidos. Al finalizar la confirmación de la última trama, podemos finalizar con un bit de STOP, o podemos seguir la transmisión con un RE-START, idéntica a la condición de START, pero sin STOP previo. Es usada para comunicar en otro modo de funcionamiento, tanto con el mismo esclavo o con otro esclavo.

En la transferencia de datos SDA debe estar estable mientras SCL está en HIGH, pues corresponde al momento de transferencia del dato. En el noveno pulso de SCL señalizamos la confirmación.

El SMBus es igualmente una interfaz a dos hilos, definida por Intel y Duracell en 1994, basándose en los principios de funcionamiento del I^2C, pero con mayores restricciones temporales y una frecuencia limitada de 10KHz hasta 100

KHz, hasta la cual podemos compartir bus para ambos. Los nombres de las líneas estándar cambian, SCL pasa a ser SMBCLK, mientras que SDA pasa a SMBDAT. También cambian los niveles lógicos. Mientras que en SMBus el nivel alto se sitúa entre 1.35 V y V_{DD}, y el nivel bajo en 0.8V, mientras que en I^2C se establece una relación con el voltaje, siendo $0.3V_{DD}$ para el nivel bajo y $0.7V_{DD}$ para el nivel alto [11].

Al poder compartir bus hasta 100KHz, trabajaremos con el bus I^2C limitado a esa frecuencia, y teniendo en cuenta las limitaciones dadas por el sensor para las instrucciones, solo podemos comandos de lectura y escritura.

2.5 Comunicaciones Bluetooth

Bluetooth Low Energy, o BLE, se introdujo como parte de la especificación de Bluetooth 4.0, proveniente de un proyecto inicialmente desarrollado por Nokia y conocido como Wibree, antes de ser adoptado por Bluetooth SIG [12], organización dedicada al desarrollo y estandarización de las comunicaciones Bluetooth. Comparado con las comunicaciones clásicas, BLE requiere menos potencia, tiempo y esfuerzo para emparejar dispositivos, y tiene velocidades de transmisión menores.

Bluetooth 4.0 y BLE son soportados en la mayoría de las plataformas: Android, a partir de la versión 4.3, iOS, a partir de su versión 5, los sistemas operativos de Apple (a partir de 10.6), de Windows 8 en adelante y GNU/Linux Vanilla BlueZ, a partir de la versión 4.93.

La pila del protocolo está dividida en dos categorías: el cliente y el servidor, los cuales tienen subcategorías que desempeñan roles específicos. Estas subcategorías son el GAP y el GATT. El primero define la topología general de la pila, mientras que el GATT describe en detalle como los datos son transferidos una vez los dispositivos tienen la conexión.

GAP es lo que permite que el dispositivo sea público hacia el exterior y determina como dos dispositivos pueden interactuar entre ellos. Distinguiremos entre dos tipos de dispositivos [12] [13]:

- **Periféricos.** Dispositivos pequeños, de baja potencia y recursos que pueden conectarse a dispositivos centrales más potentes. Posterior a la conexión, los periféricos se mantienen conectados al mismo dispositivo central. Un ejemplo podría ser un glucómetro, un medidor de pulsaciones, una baliza…
- **Dispositivo central.** Se corresponde al dispositivo de capacidad de procesamiento mayor, como podría ser un móvil o un ordenador.

Ahora vamos a definir los roles propios del GAP. Por un lado, tenemos broadcast, en el cual no precisamos una conexión explícita para transferir datos. En este caso una o varias centrales escuchan continuamente la información proveniente de uno de los periféricos. Por otro lado, tenemos el rol de conexión, en la que ambos dispositivos se interconectan para enviar información. Este último modo es el más utilizado [12] [13].

Pasamos a hablar del GATT, que define la manera en que dos dispositivos BLE se comunican usando servicios y características. El protocolo que usaremos en la comunicación es conocido como ATT, que almacena servicios, características y datos relacionados en una tabla, usando identificadores de 16 bit para cada una de las entradas. Si entra en juego el GATT significa que ya hay conexión, por lo que ya habremos pasado por el GATT, haciendo así las comunicaciones exclusivas y previniendo la comunicación bidireccional.

Los roles que se pueden adaptar en este tipo de comunicación son los siguientes [13]:

- **Cliente.** Envía una petición al servidor GATT. Puede leer y escribir atributos encontrados en el servidor.
- **Servidor.** Uno de los roles principales es almacenar atributos. Una vez el cliente hace una petición, hace que estos estén disponibles.

Cuando establecemos una conexión, por ejemplo, una baliza en la que pulsamos un botón y el ordenador debe leer esa orden, el periférico, en este caso el ordenador, sugerirá un intervalo de conexión al dispositivo central, la baliza, la cual intentará la conexión en ese intervalo para ver si hay nuevos datos disponibles. Puede no cumplirse esta solicitud por el dispositivo central, podría

estar comunicando con otros periféricos o los recursos del sistema no están disponibles. En la Figura 15 se ilustra el proceso de intercambio de datos, iniciando cada una de las transacciones [12] [13].

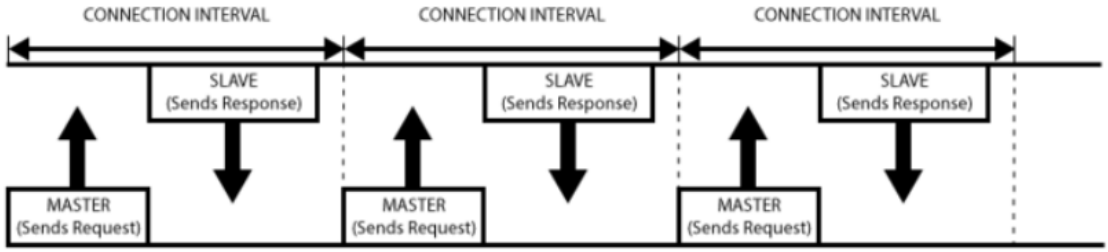


Figura 15. Transacciones entre periférico y dispositivo central [12].

Las transacciones GATT se basan en objetos anidados de alto nivel denominados perfiles, servicios y características, descritas en la Figura 16, y definidas posterior a ella.

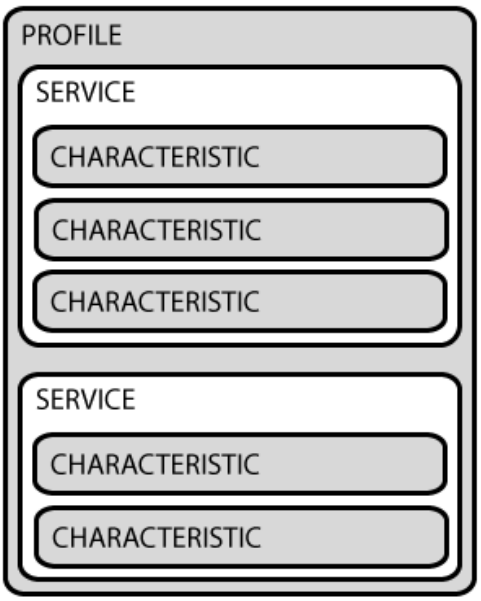


Figura 16. Objetos transacciones GATT [13].

El perfil no existe propio en el propio periférico de BLE, es una colección predefinida de servicios especificada por el Bluetooth SIG o por el propio fabricante.

Sobre los servicios, son usados para dividir datos en entidades lógicas y contienen trozos específicos de datos llamados características, pudiendo tener una o más características, y cada uno se identifica por un número único llamado UUID, de 16 bit si es un servicio oficial, o 128 bit para servicios personalizados.

El nivel inferior en las transacciones GATT son las características, que contienen un único tipo de dato, identificado de igual manera con un UUID, pudiendo usar las definidas de manera estándar, que nos asegurarán la interoperabilidad a través de hardware o software habilitada para BLE, o podemos definir características propias personalizadas, que solo serán entendidas por los periféricos y aplicaciones de uno. Estos pueden ser de lectura o escritura.

Los roles de GAP y GATT son totalmente independientes. Los periféricos y los dispositivos centrales pueden actuar ambos como servidor o cliente, dependiendo de cómo sea el flujo de datos [13].

2.6 Conclusiones

En este capítulo se ha realizado un estudio teórico sobre diferentes sensores de temperatura que hay en el mercado, examinando una familia de sensores térmicos, y una matriz de sensores, ambos de tecnología infrarroja.

Una vez realizado el estudio, optaremos por los sensores de la gama Melexis, en concreto los sensores AAA y ACC, por sus características, que se adaptan a las necesidades del sistema sin tener un precio muy elevado. Además, en vista de la gran utilidad de la cámara térmica, decidimos añadir el *array* de sensores que lleva en el sistema electrónico.

La decisión de añadir los dos sensores viene motivada por el hecho de que el *array* de sensores no mide por encima de los 80ºC, por lo que no podemos hacer uso de ellos para medir procesos de soldadura, que están sobre los 200ºC.

Capítulo 3. Estudio de la plataforma STM32WB55

3.1 Introducción

Se forma por un adaptador USB y el Núcleo-68. Este pack se forma de dispositivos de diferentes protocolos inalámbricos, incluyendo BLE 5.0 [21]. En rango del Bluetooth para estos módulos es de 10 metros en condiciones normales de trabajo, y 100 metros en campo abierto. Soporta un número máximo de conexiones simultáneas, que es 8 [22].

Figura 17. Pack STM32WB55 [21].

En la Figura 17 vemos los dos dispositivos del pack, formado por el Núcleo-68, que es la placa de desarrollo, y el adaptador USB. En este capítulo describiremos el microcontrolador STM32WB55, ambos dispositivos, y daremos una conclusión global sobre los mismos.

3.2 Microcontrolador STM32WB55

Este microcontrolador de dos núcleos y multiprotocolo inalámbrico se encuentra en ambos dispositivos incluidos en el pack, por lo que vamos a dar una explicación global del mismo.

El procesador de aplicación se basa en la arquitectura ARM Cortex-M4 a 64MHz, mientras que el procesador de la red está basado en ARM Cortex M0+. El uso de una arquitectura conformada por dos microcontroladores está optimizado para la ejecución en tiempo real, permitiendo flexibilidad en los recursos. Posee una memoria flash de hasta 1Mbyte, y una memoria RAM de hasta 256 Kbyte.

Soporta Bluetooth 5.0 y estándares inalámbricos IEEE 802.15.4, como ZigBee y Openthread, sin necesidad de módulos adicionales. Esto último lo hace adecuado para nuestra aplicación, pues nos evitamos así usar electrónica adicional, facilitando así hacer un módulo inalámbrico. Además, se nos permite ejecutar concurrentemente protocolos inalámbricos 802.15.4 y BLE [23].

En la Figura 18 vemos el diagrama de bloques del sistema RF, basado en modulación directa de la portadora en el transmisor, usando arquitectura de bajo IF en el receptor.

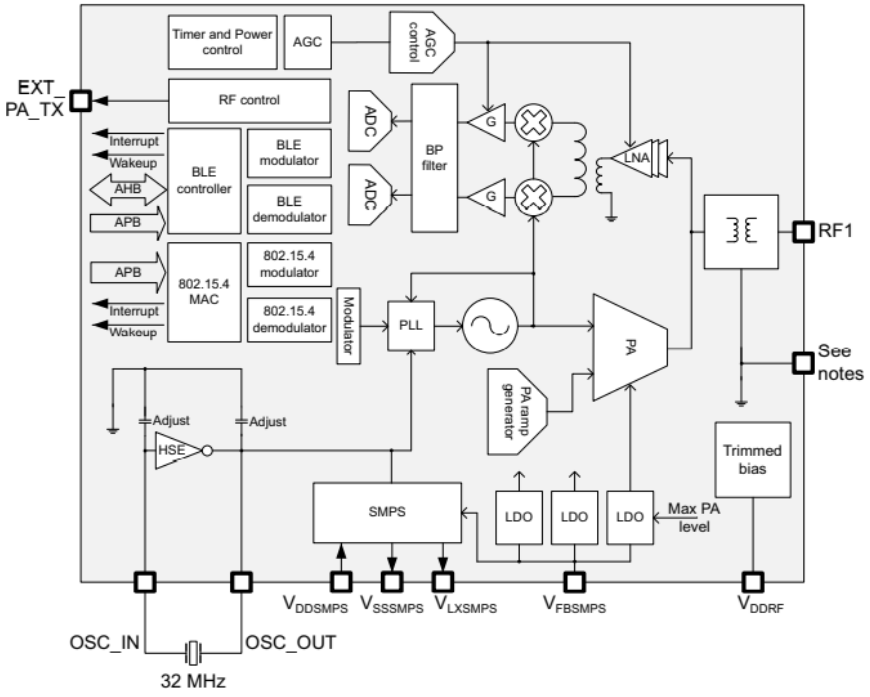


Figura 18. Diagrama de bloques del sistema RF [24].

Incluye una gran variedad de características de comunicación, como un USB de alta velocidad, soporte para audio, drivers para LCD, sensibilidad táctil, hasta

72 GPIOs, un SMPS integrado para la optimización del consumo de potencia y varios modos de baja potencia para maximizar la vida de la batería. Asimismo, añade una conexión USART y LPUART, y dos conexiones I^2C y SPI respectivamente.

Hay que mencionar que, adicionalmente incluye funciones de seguridad hardware empotradas, como encriptación hardware AES de 256 bit, protección de lectura y escritura PCROP, fusible JTAG y criptografía de clave pública, aunque en este punto no nos centraremos.

En la Figura 19 mostramos el diagrama de bloques del microcontrolador, en la que podemos ver muchas de las características mencionadas en este apartado, y otras como los temporizadores. Las interconexiones entre bloques se basan en el estándar AMBA AXI de ARM, mediante buses APB y AHB Lite, incluidas en este protocolo, el cual se orienta a comunicaciones con altas tasas de transferencia de datos y baja latencia temporal.

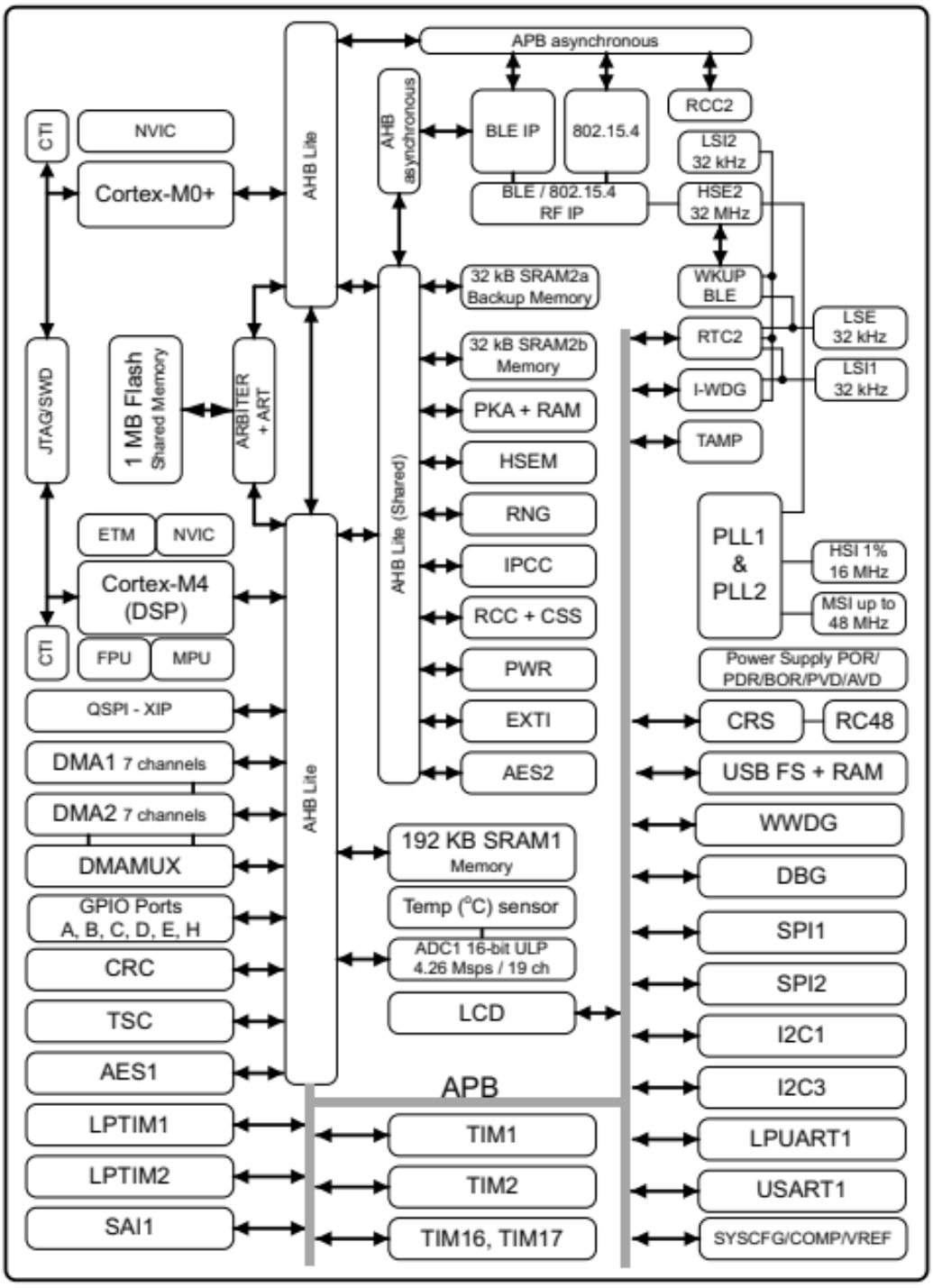


Figura 19. Diagrama de bloques STM32WB55xx [24].

En cuanto a la gestión de relojes, nuestro microcontrolador integra algunas fuentes de reloj. La Figura 20 muestra la distribución de reloj proveniente de los diferentes osciladores al núcleo y periféricos, incluyendo el subsistema RF.

Adicionalmente, maneja la apertura y cierre de relojes para los modos de bajo consumo, asegurando la robustez del reloj.

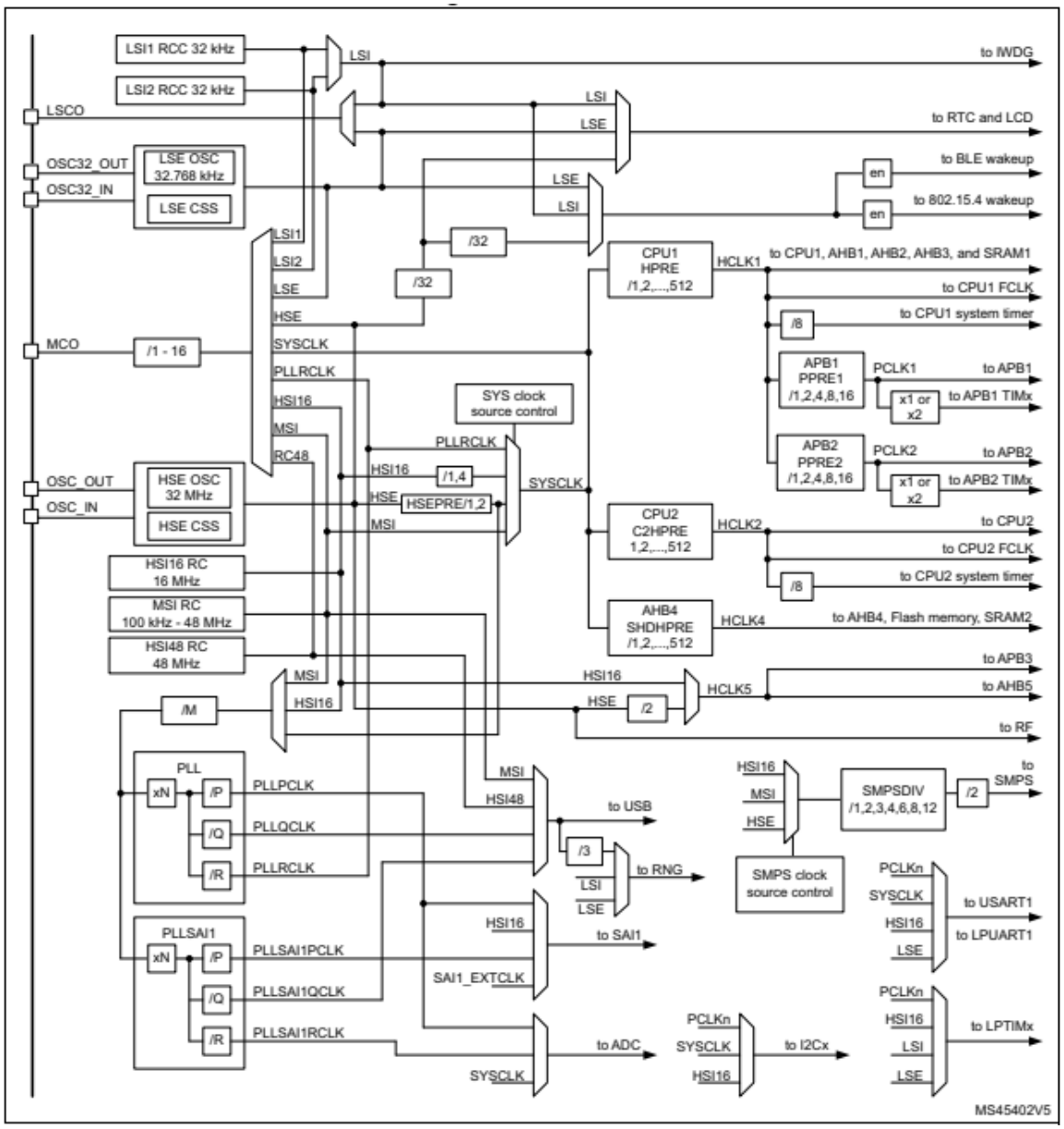


Figura 20. Distribución señales de reloj STM32WB55xx [24].

3.3 Núcleo-68

Este paquete usa microcontroladores de 32 bit de STM32WB, basados en los procesadores ARM Cortex, explicados en el apartado anterior.

Las características de este núcleo son las siguientes [22]:

- Microcontrolador STM32WB en paquete VFQFNP68.
- Transceptor RF a 2.4 GHz, soportando Bluetooth 5.0 y IEEE 802.15.4-2011 PHY.
- CPU ARM Cortex M0+ de 32 bits dedicado para la capa de radio en tiempo real.
- El SMPS reduce significativamente el consumo de potencia en el modo Run.
- Tres LED de usuario compartidos con Arduino.
- Cuatro botones.
- Oscilador de cristal LSE a 32.768 KHz.
- Oscilador de cristal con capacitores trimmer integrados a 32MHz.
- Conectores de expansión:
    - Arduino Uno V3
    - ST Morpho
- Alimentación de la placa flexible: ST-LINK/V2-1, USB, VBUS y fuentes externas.
- Depurador/programador a través de ST-LINK/V2-1 con funcionalidades USB: Almacenamiento masivo, puerto COM virtual y puerto de depurado.
- Librerías de software libres y varios ejemplos disponibles.
- Soporta diferentes entornos de desarrollos, incluyendo IAR, Keil y Arm Mbed.

En la Figura 21 vemos con más detalle algunas de las características físicas mencionadas situadas en el núcleo.

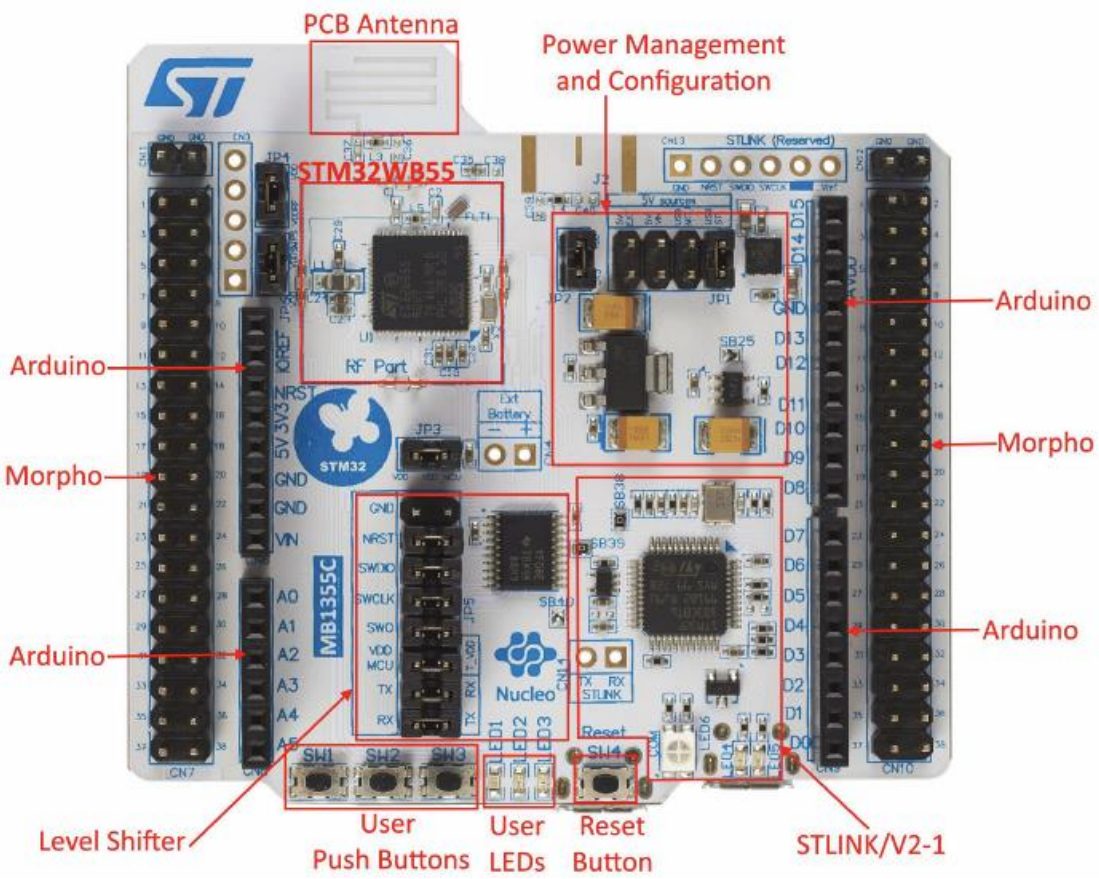


Figura 21. Detalles núcleo-68 [22].

Además, vemos en la Figura 21 la existencia de unos pines y unos jumpers con la denominación gestión de la alimentación y configuración. Dependiendo de la posición de estos pines podemos seleccionar de dónde se alimenta. La primera posición corresponde a 5 V externos, la segunda posición corresponde a 5 V internos, la tercera posición es la alimentación a través del USB, y la cuarta posición es la correspondiente a la conexión ST-LINK/V2-1, desde la cual adicionalmente se realizará la programación de nuestro microcontrolador.

En la Figura 22 vemos con más detalle el diagrama de bloques hardware para nuestro núcleo, que ilustra las conexiones entre el microcontrolador y los diferentes periféricos disponibles en rasgos generales.

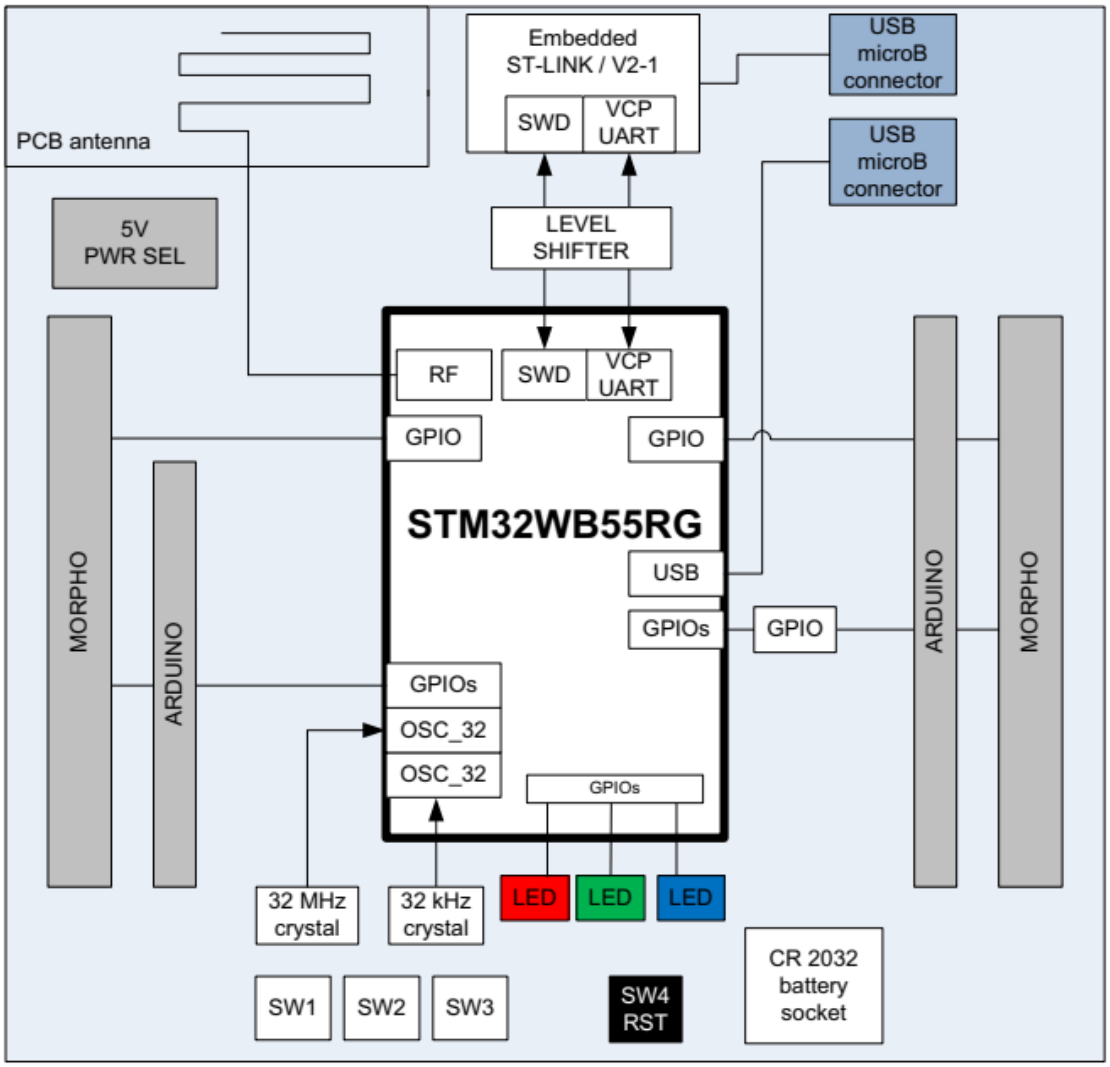


Figura 22. Diagrama de bloques hardware del núcleo-68 [22].

Figura 23. Distribución de pines núcleo-68 [22].

La distribución de pines para nuestro núcleo la tenemos detallada en la Figura 23. Los conectores ST-Morpho CN7 y CN10 son pines macho accesibles en los dos lados de la placa. Todas las señales y pines de alimentación del microcontrolador están disponibles en los conectores Morpho. También incluye los conectores compatibles con Arduino, que son los correspondientes a CN5, CN6, CN8 y CN9, pines hembra accesibles en la parte superior del microcontrolador.

La programación de este dispositivo la realizaremos a través del mismo IDE propio del fabricante, que envía datos al núcleo a través de la entrada ST-LINK.

3.4 Adaptador USB.

Las características generales asociadas a este adaptador son las siguientes [22]:

- Microcontrolador STM32WB en paquete VFQFNP68.
- Transceptor RF a 2.4 GHz, soportando Bluetooth 5.0 y IEEE 802.15.4-2011 PHY.
- CPU ARM Cortex M0+ de 32 bits dedicado para la capa de radio en tiempo real.
- El SMPS reduce significativamente el consumo de potencia en el modo Run.
- Oscilador de cristal LSE a 32.768 KHz.
- Oscilador de cristal con capacitores trimmer integrados a 32MHz.
- Solución Bluetooth completa con una antena integrada para conexiones rápidas.
- Interruptor para gestión del arranque.
- Botones de usuario.
- Tres LED de usuario.

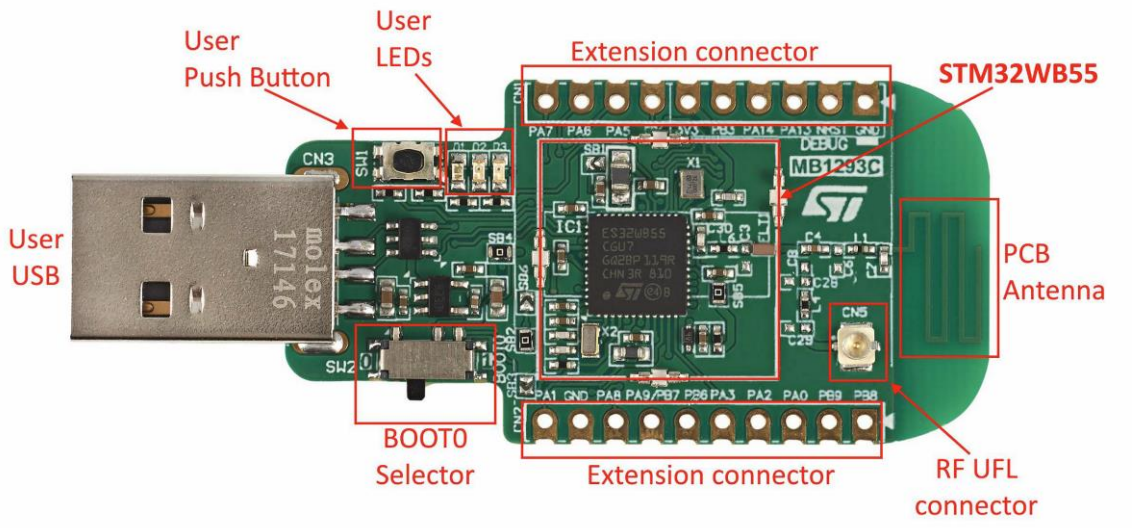


Figura 24. Detalles adaptador USB [22].

En la Figura 24 observamos las características físicas de nuestro adaptador, contemplando algunas de las mencionadas. En la Figura 25 le daremos una definición a los conectores para extensión.

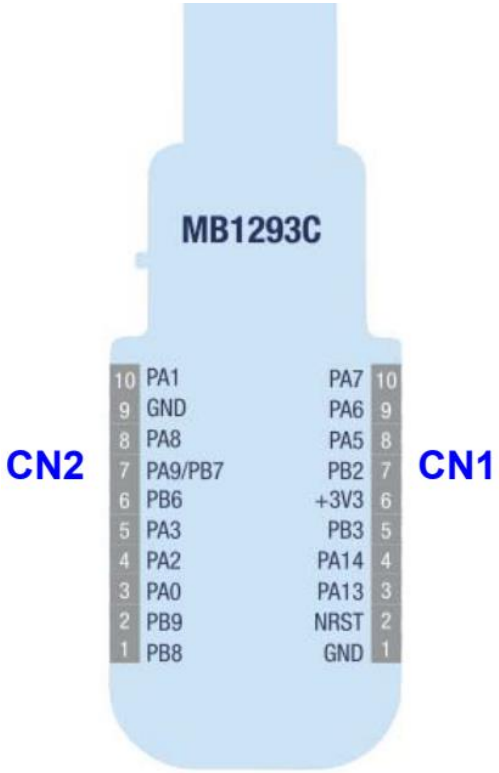


Figura 25. Distribución pines de extensión adaptador USB [22].

Para la programación de este dispositivo podremos hacerlo a través del mismo USB, usando el modo BOOT, el cual será accesible a través del BOOT0 selector, el cual podemos ver en la Figura 24. De esta manera haremos uso del modo DFU.

La programación también se podrá realizar a través de ST-LINK y los pines de expansión, representada en la Figura 26. Previamente para ello habrá que preparar una placa de adaptación para estos conectores, de los que sacaremos las señales y aprovecharemos el ST-LINK del núcleo a través de los elevadores de nivel, pines remarcados en la Figura 21. Estos están puenteados porque de ahí hacemos la programación del núcleo, pero si retiramos los jumpers y realizamos la conexión a los pines de adaptación podremos programar y depurar nuestro adaptador.

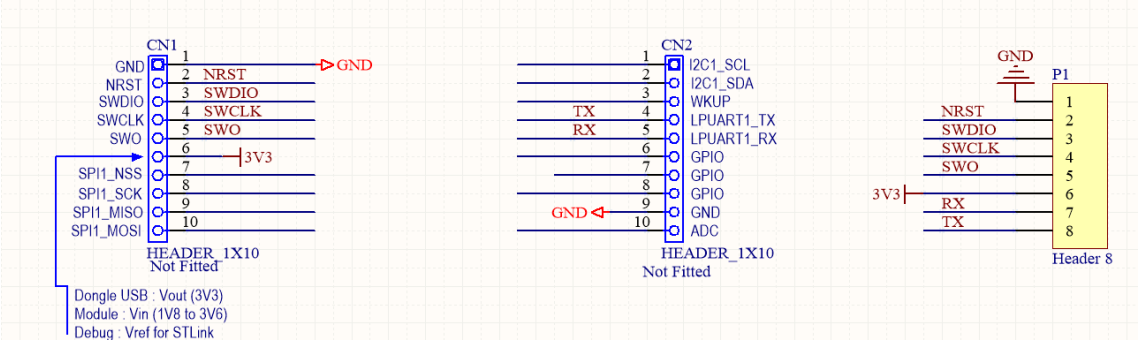


Figura 26. Conexiones para programación ST-LINK.

3.5 Conclusiones

A lo largo de este capítulo hemos hecho un análisis del microcontrolador STM32WB5, sobre el cual haremos pruebas en este trabajo, describiendo sus funcionalidades Wireless y Bluetooth, características que precisamos para este proyecto.

Además, hemos realizado una descripción de los dispositivos que conforman el pack, el núcleo-68 y el adaptador USB.

Capítulo 4. Estudio de la plataforma ESP32.

4.1 Introducción

Una vez realizado el estudio de la plataforma STM32WB55 en el capítulo 3, en este capítulo 4 vamos a ver la otra plataforma propuesta, ESP32.

La plataforma ESP32-WROOM-32D, que integra el microcontrolador ESP32-D0WD, chip de muy bajo consumo, está preparada para ser usado en dispositivos móviles, electrónica de consumo y aplicaciones de IoT. Además, tiene conectividad Bluetooth y Wifi, por lo que puede trabajar con estas comunicaciones. La plataforma puede comportarse tanto de manera independiente, como funcionar de maestro o esclavo [25]. Estas cualidades hacen que sea interesantes para considerarlo para este trabajo. En la Figura 27 vemos el microprocesador integrado en una placa de desarrollo.

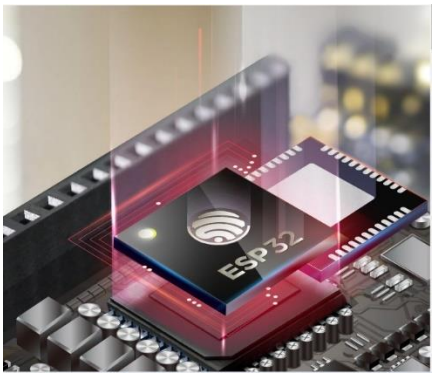


Figura 27. ESP32 montado sobre placa [25].

En este capítulo vamos a describir tanto el microprocesador como la placa de desarrollo que vamos a utilizar.

4.2 Microcontrolador ESP32-D0WD

El microcontrolador ESP32 es un chip mononúcleo de 2,4GHz con Wifi y Bluetooth integrados, diseñado con la tecnología de baja de ultra baja potencia de 40nm de TSMC. Está diseñado para conseguir la mejor potencia y actuación de radiofrecuencia, mostrando robustez, versatilidad y confianza en una gran variedad de aplicaciones y escenarios [26].

Las características genéricas con las que cuenta son las siguientes [26]:

- Wifi 802.11 b/g/n (2.4GHz), hasta 150Mbps.
- Bluetooth v4.4 y especificaciones BLE.
- Microprocesador Xtensa mononúcleo 32 bit LX6.
- Memoria de 448 ROM y 520 SRAM.
- Oscilador interno de 8MHz con calibrado.
- Oscilador RC con calibrado.
- Temporizador RTC.
- Oscilador de cristal externo de 2MHz ~ 60MHz (40MHz para aplicaciones Bluetooth o Wi-Fi).
- Oscilador de cristal externo de 32khz para RTC con calibrado.

En este caso, la memoria flash no es empotrada.

También cuenta con sistema de seguridad. Tiene encriptado de la memoria flash, contraseñas de un solo uso de 1024 bits, arranque de seguridad y aceleración hardware criptográfico [26].

Cuenta con diversas interfaces avanzadas de periféricos, que incluye 34 GPIO programables, conversores analógico-digital de 12 bit, conversores digital-analógico de 8 bits, 4 SPI, 2 I^2C, 3 UART, 1 maestro, 1 esclavo, interfaz MAC Ethernet con DMA dedicado y soporte a IEEE 1588, CAN 2.0, control remoto infrarrojo, tanto de transmisión como de recepción, PWM y sensor Hall [26]. En la Figura 28 vemos el diagrama de bloques detallado de este microprocesador [26].

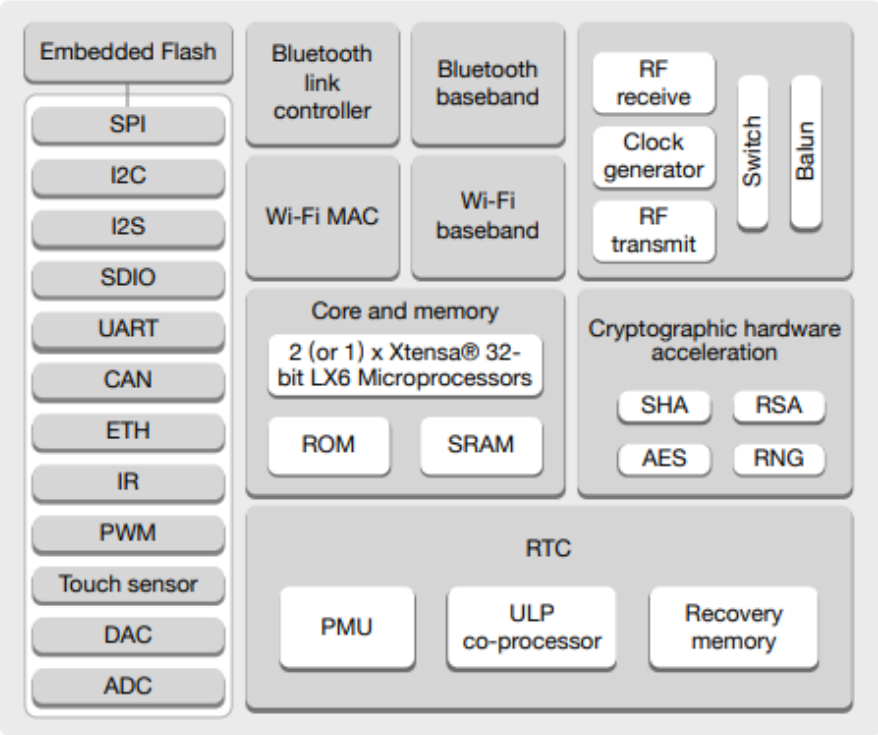


Figura 28. Diagrama de bloques ESP32-D0WD [26].

4.3 Placa de desarrollo ESP32-WROOM-32D

El ESP32-WROOM-32D es un potente módulo con Wifi, Bluetooth y BLE que tiene como objetivos varias aplicaciones, desde redes de sensores de baja potencia hasta las tareas que más demandan, como codificación de voz [27]. En la Tabla VI recogemos las especificaciones de este.

Tabla VI. Especificaciones ESP32-WROOM-32U [27].

Categoría	Ítem	Especificaciones
Wifi	Protocolos	802.11 b/g/n (802.11n up to 150 Mbps)
		Intervalo de guarda de 0.4 µs
	Rango de frecuencia	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocolos	Bluetooth v4.2 BR/EDR y BLE
	Radio	Receptor NZIF con sensibilidad de -97dBm
		Transmisor de clase 1, clase 2 y clase 3
		AFH
	Audio	CVSD y SBC
Hardware	Interfaces del módulo	Tarjeta SD, UART, SPI, SDIO, I2C, PWM, I2S, IR, contador de pulsos, GPIO, ADC, DAC, sensor táctil capacitivo
	Sensores en chip	Sensor Hall
	Cristal integrado	40 MHz
	Flash SPI integrada	4 MB
	Voltaje de operación/suministro de potencia	3.0 V ~ 3.6 V
	Corriente de operación	80 mA
	Corriente mínima del suministro de potencia	500 mA
	Rango de temperatura de operación recomendada	–40 °C ~ +85 °C
	MSL	Nivel 3

En este caso, usaremos una placa de desarrollo de 30 pines. En la Figura 29 vemos en detalle el *pinout*, marcando qué pines son recomendados para cada uno de los usos que requiramos.

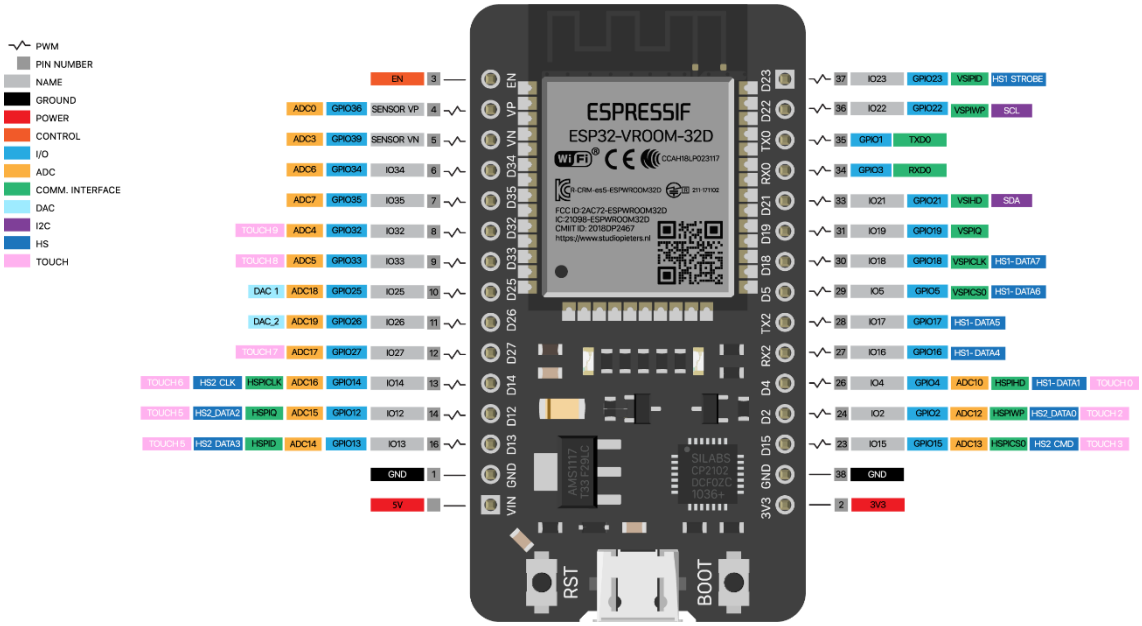


Figura 29. Pinout de la placa ESP32-VROOM-32D [28].

En concreto, los pines que van del 34 al 39 son pines solo de entrada, no tienen resistencias *de pull-up/pull-down*, no pueden ser usados como salidas. La mayoría de los pines pueden ser usados para cualquiera de los fines, pero debemos prestarles atención debido a que pueden tener un comportamiento inesperado en el arranque. Ejemplos de esto son el pin 0, el pin 1 (de entrada es recepción), el pin 3 (de salida es transmisión) o el pin 12. Por último, los pines que van desde el 6 hasta el 11 no pueden ser usados como entrada/salida, debido a que están conectadas a la memoria flash SPI integrada [28]. En la Figura 30 disponemos del diagrama de bloques de esta plataforma

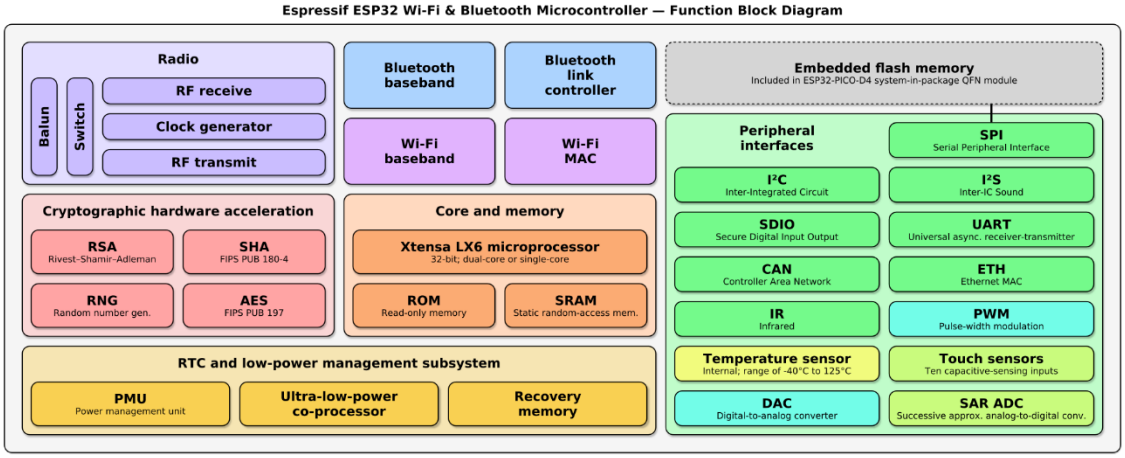


Figura 30. Diagrama de bloques del ESP32 [29].

Finalmente, en la Figura 31 tenemos con detalle el microcontrolador de esta placa, remarcando de qué pin sale cada señal.

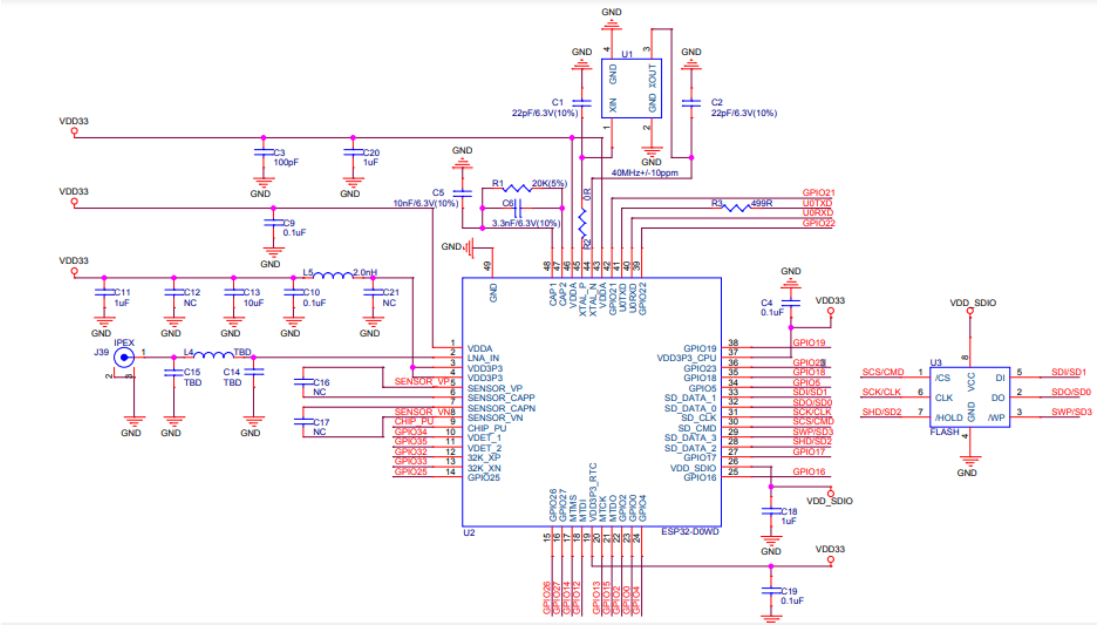


Figura 31. Microcontrolador empotrado en ESP32-WROOM-32U [27].

4.4 Conclusiones

Esta placa de desarrollo es una alternativa muy interesante, pues tiene comunicaciones inalámbricas, con una amplia selección de interfaces. Además, su tamaño la hace atractiva, pues será de más fácil acoplamiento como sistema electrónico.

Una vez dadas las visiones de cada una de las placas, vamos a desarrollar cómo sería el flujo de diseño en el capítulo 5.

Capítulo 5. Flujo de trabajo sobre las diferentes plataformas.

5.1 Introducción

El firmware para STM32WB55 se realiza en lenguaje C, por lo que podría ser desarrollado en plataformas propias del mismo, como IAR, Keil o Arm Mbed, además de su propio entorno de desarrollo, STM32CubeIDE [22]. Este último, perteneciente al ecosistema STM32Cube, es el que estudiaremos principalmente para el desarrollo de las aplicaciones.

STM32Cube es una solución completa para los microcontroladores y microprocesadores de la familia STM32. Está pensado para usuarios que precisan de un entorno gratuito y completo para STM32, además de usuarios que ya tienen un IDE alternativo, los cuales podrán integrar algunos de los módulos del ecosistema [30]. En la Figura 32 vemos las herramientas software a utilizar en el desarrollo del firmware.

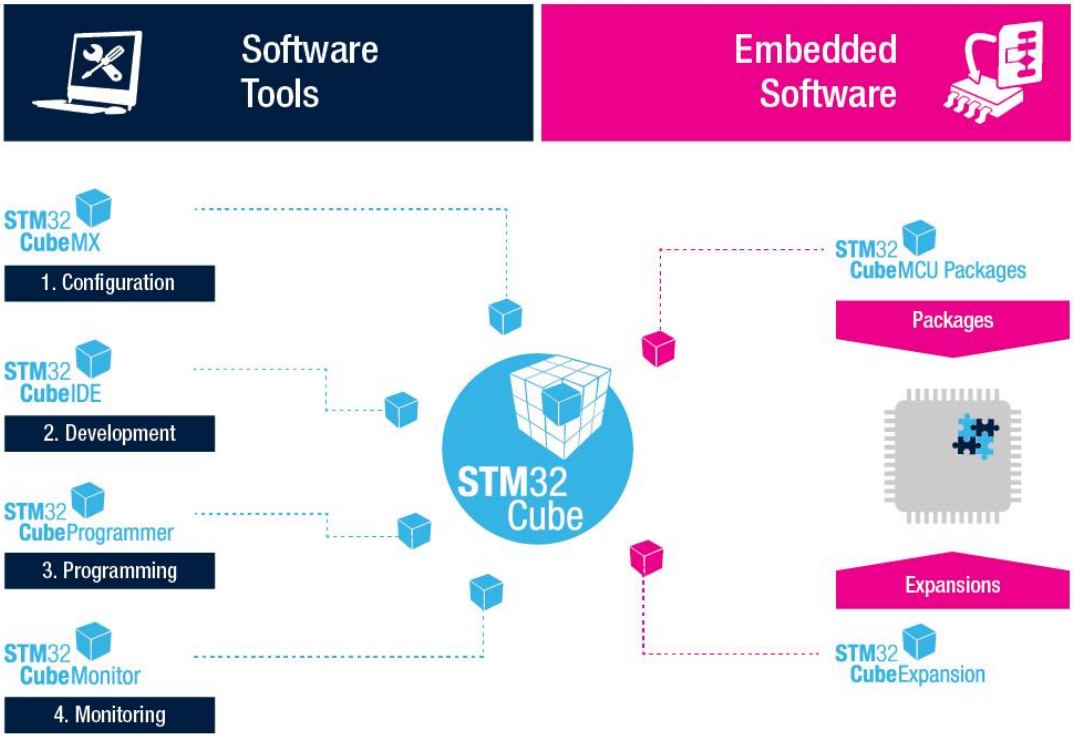


Figura 32. Flujo de desarrollo STM32 Cube Ecosystem [30].

Por otro lado, el ESP32 lo vamos a programar en el entorno Arduino, por la facilidad de integración de las diferentes librerías de los sensores y de las comunicaciones BLE.

En este capítulo vamos a detallar algunas de las aplicaciones que utilizaremos en nuestro TFT, dando una visión general de cuál será nuestro flujo de trabajo.

5.2 Ecosistema STM32Cube.

5.2.1 STM32CubeMX

Herramienta de configuración para cualquier dispositivo STM32. Esta interfaz gráfica de fácil uso genera la inicialización del código C para los núcleos Cortex-M [31].

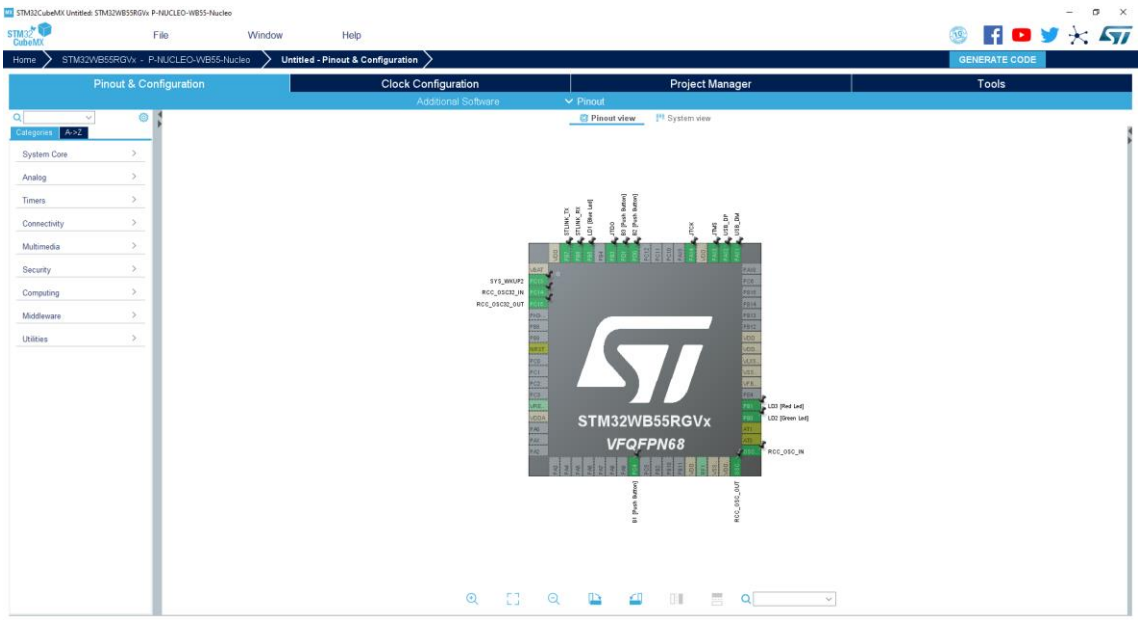


Figura 33. Ventana del entorno STM32CubeMX.

En la Figura 33 podemos ver la interfaz de esta aplicación. En el centro tenemos nuestro microprocesador, con cada uno de los pines a configurar en gris. A color vemos los pines que ya tienen una definición dada. Hay algunos que no podremos modificar, ya que son necesarios para su correcto funcionamiento.

También nos permite trabajar con diferentes interfaces de entrada – salida. No todos los pines son compatibles con todas las configuraciones posibles, entre las que se encuentran I^2C, GPIO, SPI, RF, USART y LPUART, a los cuales se nos permite dar configuraciones de temporización.

Por otro lado, podemos configurar pines para generar temporizadores, como temporizadores de baja potencia o temporizadores de propósito general, además de tener un temporizador de tiempo real. Incluye más configuraciones, como analógicas, de seguridad o multimedia, las cuales no trataremos en este documento ya que no se considera su uso en nuestro proyecto.

Adicionalmente, se nos permite calcular el consumo de potencia de las aplicaciones. Incluye el árbol del reloj, con comprobación dinámica de la configuración.

Esta herramienta es muy útil, pues la generación del código C la realiza en tiempo real, es decir, se refresca automáticamente en el código fuente que estemos desarrollando.

5.2.2 STM32CubeIDE

IDE basado en soluciones de código abierto, como Eclipse o la cadena de herramientas GNU C/C++. El entorno de desarrollo integrado Eclipse tiene como uso principal dar una plataforma de programación, compilación y depurado para muchos lenguajes de programación [32]. El lenguaje más utilizado para esta plataforma es Java, pero su modularidad nos permite usarla para otros, como C.

Este entorno incluye herramientas de compilación y de depurado avanzado. Incluye además herramientas adicionales como STM32CubeMX integradas con todas las funcionalidades incluidas, como creación del proyecto, generación del código de inicialización, el cual podremos modificar en cualquier momento durante el desarrollo sin ningún tipo de impacto sobre nuestro código, selección del microprocesador y configuración de las salidas de los pines, relojes, periféricos y middleware [33].

Las herramientas avanzadas de depurado incluyen algunas características como registros de la CPU del núcleo, registros de los periféricos y visualización

de la memoria, además de poder ver en tiempo real las variables. Incluye la interfaz SWV y analizador de errores.

Los proyectos en este IDE podrán ser en C/C++, o propios del compilador, basados en C también. Se nos permite la portabilidad de un proyecto propio a C/C++, en el caso de que quisiéramos trabajar con diversos entornos simultáneamente.

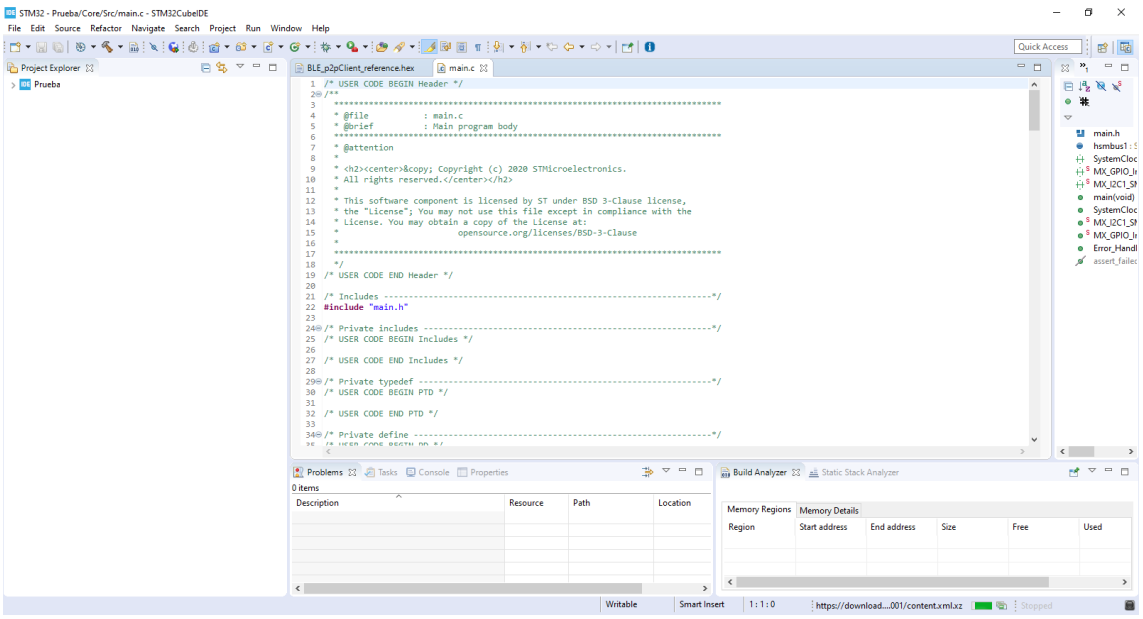


Figura 34. Ventana del entorno STM32IDE.

En la Figura 34 podemos ver la interfaz del entorno, similar a Eclipse. A la derecha tenemos el gestor de proyectos y en el centro el editor de código, en este caso C. En esta aplicación adicionalmente se nos permite compilar y depurar el firmware, además de cargarlo a través de ST-LINK.

5.2.3 STM32CubeProgrammer

Herramienta software para programar los productos de STM32. Proporciona un entorno de fácil uso y eficiente para la lectura, escritura y verificación de dispositivos a través de la interfaz de depurado (JTAG y SWD) u la interfaz de gestión de arranque (UART, USB DFU, I^2C, SPI y CAN).

Ofrece un gran rango de herramientas para programar memorias internas de dispositivos STM32, como Flash, RAM y OTP, además de memorias externas.

También permite opciones de carga, verificación de contenido y automatización a través de secuencias de comandos de los programas [34].

En la Figura 35 tenemos la interfaz del programa. En ella principalmente se nos permite conectar y cargar archivos compilados previamente en nuestro microprocesador, el cual el programa detecta de qué modo lo queremos programar (ST-LINK, USB DFU, OTA, etc.) y nos muestra diferentes parámetros que podemos configurar de cada uno de ellos, como la tasa de baudios.

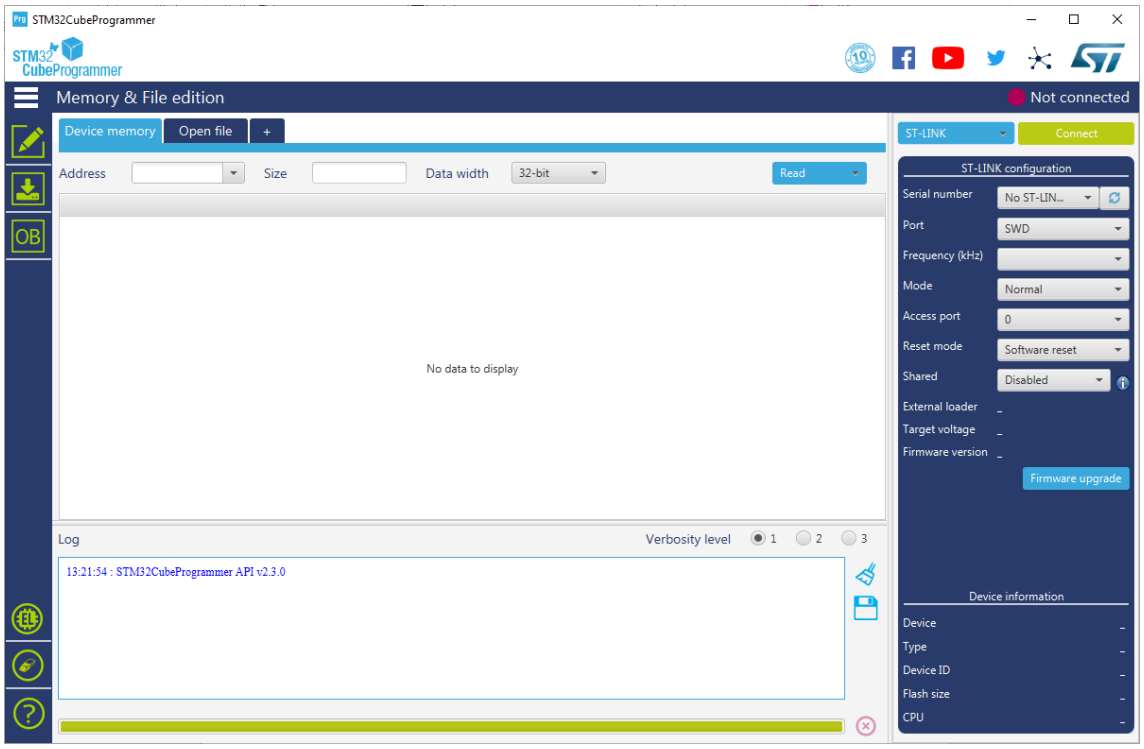


Figura 35. Ventana del entorno STM32CubeProgrammer.

La interfaz también nos muestra la memoria interna del dispositivo, las direcciones y el contenido de ellas, pudiendo ser organizada en 8, 16 o 32 bit. Este contenido también puede ser guardado en un archivo en la memoria del ordenador.

Si bien este programa es el último escalón de nuestro flujo de diseño, siendo prescindible en el caso de que estemos programando a través de ST-LINK, cuya programación podremos realizar desde el IDE.

5.3 Entorno de desarrollo Arduino

Arduino IDE es un entorno multiplataforma de código abierto basado en la facilidad de uso, tanto del hardware como del software [35]. En este entorno se usa lenguaje Arduino, también publicado como herramienta de código abierto, y disponible para su extensión por programadores avanzados. Este lenguaje, basado en AVR-C, puede ser extendido a través de librerías de C++. También permite a los diseñadores de circuitos la posibilidad de poder hacer versiones propias del módulo, así mejorándolo y adaptándolo a sus propias necesidades [35].

Este entorno de desarrollo es sencillo de uso para los principiantes, pero también da flexibilidad a los programadores avanzados para aprovechar todas sus ventajas [35].

Arduino, nacido en *Ivrea Interaction Design Institute*, surge como una herramienta sencilla para el rápido prototipado, ayudando a los estudiantes sin conocimientos previos de electrónica y programación. Tan pronto como creció entre la comunidad, la placa de Arduino comenzó a cambiar para adaptarse a las nuevas necesidades, desde los dispositivos originales de 8 bits, a los productos existentes actualmente para aplicaciones de IOT, impresión 3D o sistemas empotrados [35].

Principalmente se usa para programar plataformas Arduino, basadas en el microcontrolador Atmel AVR, pero también nos permite programar otros dispositivos, como Parallax Basic Stamp, Netmedia’s BX-24 [35], o el ESP32. Este último podemos instalarlo simplemente añadiendo al gestor de placas adicionales la instrucción del ESP32 mostrado en la Figura 36.

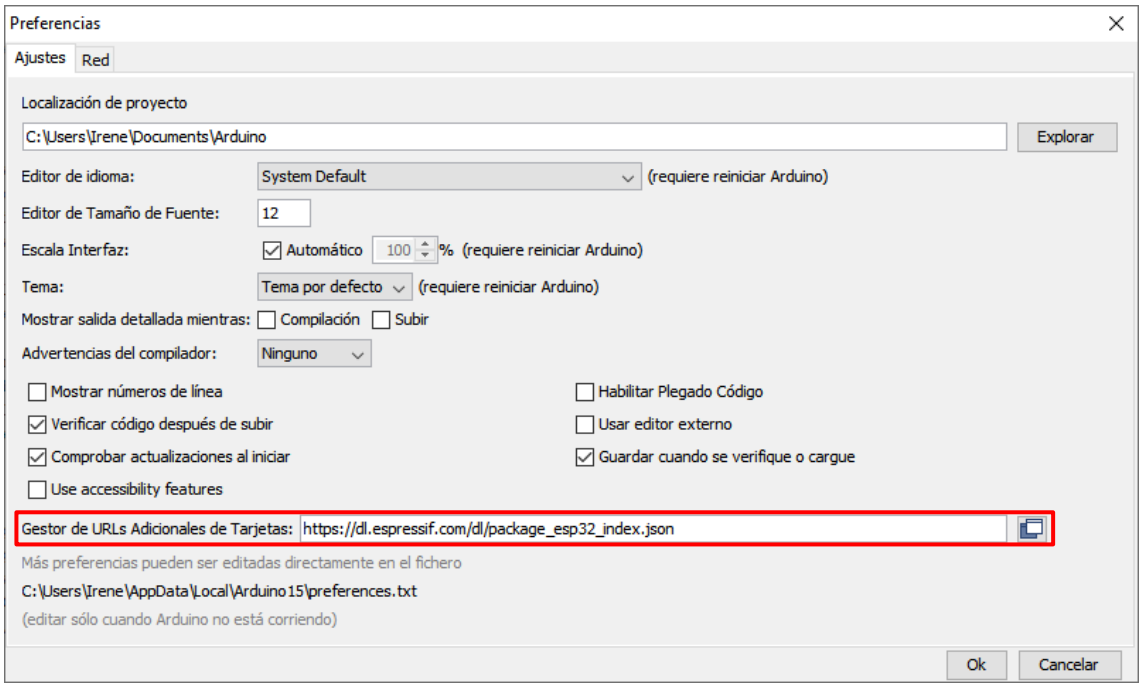


Figura 36. Gestor de tarjetas adicionales Arduino IDE.

Una vez instalado, ya tenemos completo acceso a nuestra placa. Además, yendo al apartado de las librerías, podremos probar diversos ejemplos, como el de BLE, tanto de servidor como de cliente.

En la Figura 37 mostramos el entorno. Hemos remarcado diferentes zonas importantes en el desarrollo en Arduino. En negro tenemos dos botones, el primero nos compila el proyecto, y el segundo lo carga en la placa. En rojo tenemos las funciones principales de Arduino. En el setup() irán las instrucciones que queremos que se ejecuten al comienzo del programa, y en el loop() las instrucciones que se irán comprobando continuamente en la ejecución del programa.

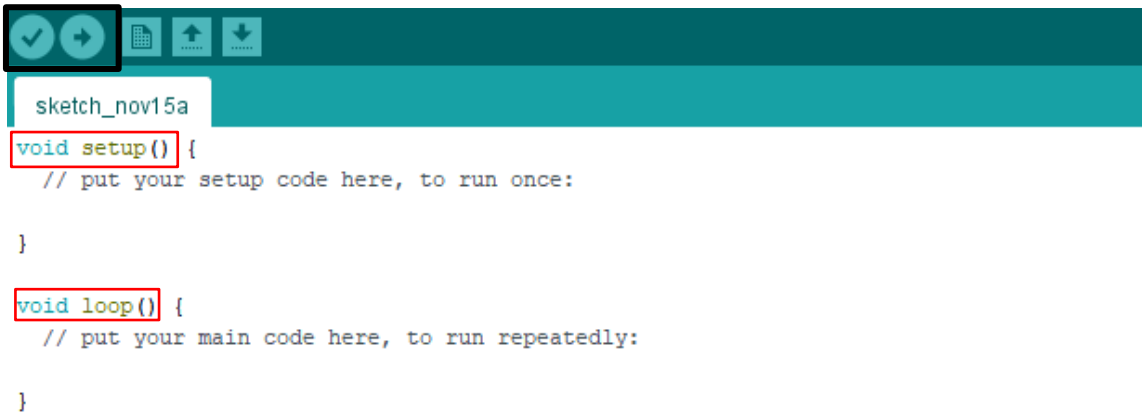


Figura 37. Arduino IDE.

5.4 Conclusiones

En este capítulo hemos explicado las herramientas que conforman el paquete STM32Cube, recomendado por el fabricante del pack STM32 para el desarrollo de software para el microprocesador elegido, desde la configuración de los pines hasta el funcionamiento con nuestro firmware de nuestro dispositivo.

También hemos mostrado el IDE de Arduino, que utilizaremos para programar el ESP32 debido a su sencillez de uso, y las librerías que incluye.

Habiendo visto ya una visión de cada módulo que queremos integrar en nuestra solución, en el capítulo 6 vamos a diseñar diferentes prototipos, y el producto final.

Capítulo 6. Diseño de la solución propuesta

6.1 Introducción

En este capítulo describiremos los diseños realizados para este proyecto. Incluiremos un prototipo Shield, tanto para el núcleo del pack STM32WB55 como para los ESP32, en los que tendremos las señales necesarias ya preparadas para trabajar con mayor facilidad. Un Shield es una extensión con la electrónica necesaria para operar con las señales necesarias.

También realizaremos una PCB con la electrónica necesaria para el correcto funcionamiento de los sensores.

Adicionalmente hemos incluido el diseño de un prototipo de la placa de adaptación para programar nuestro adaptador USB, el cual para ser programado a través del ST-LINK precisa de usar los pines de expansión de la placa.

También explicaremos el procedimiento previo de fabricación de los prototipos sobre software, sobre la que daremos directrices para la misma.

Finalmente, montaremos la placa en el laboratorio.

6.2 Altium Designer Release 10

Entorno de diseño para PCB. Cubre todos los aspectos del desarrollo electrónico con los editores y motores software necesarios para ello, incluyendo diseño físico, simulación de circuitos de señal mixta, análisis de la integridad de la señal y manufactura, o vista 3D [36].

Para la realización de un proyecto en Altium, primero debemos tener claro el diseño a implementar, y lo añadiremos al mismo con un esquema. En este daremos algunas instrucciones como el tamaño de los componentes y su huella, su colocación y la conexión de las señales.

Una vez tenemos el esquema compilado y sin errores, procederemos a realizar la PCB. Para ello, crearemos en nuestro proyecto una, y le importaremos los cambios realizados en el esquema. Nos aparecerán los diferentes componentes y sus interconexiones, los cuales tendremos que ir colocando según como queramos que sea nuestro diseño físico final, y pensando en el

rutado, pues es el paso que le sigue. Conectaremos todos los componentes mediante pistas, y comprobaremos que estamos cumpliendo todas las normas de diseño.

El editor PCB es un entorno gobernador por reglas. Esto significa que mientras hagamos cambios que cambien el diseño, como colocar pistas, mover componentes, o usar el *routing*, el software monitoriza cada acción y comprueba si el diseño cumple con las reglas de diseño. Si no, el error se remarcará inmediatamente como violación. Estas reglas hay que configurarlas previas al diseño, y así a medida que vamos diseñando se nos irá comprobando que no rompemos estas reglas [37].

Para terminar, una vez que comprobamos que no hay ningún error generaremos los archivos de fabricación, en especial los archivos gerber, que contiene información de manufactura de las diferentes capas, las cuales podremos elegir, y los taladros, información de manufactura para uso para máquinas de taladro controladas por números. Sobre estos archivos no trabajaremos en Altium, si no que trabajaremos con la herramienta CircuitCAM 5.0, de la que hablaremos posteriormente en este capítulo. En la Figura 38 vemos los diferentes archivos de fabricación que podemos generar.

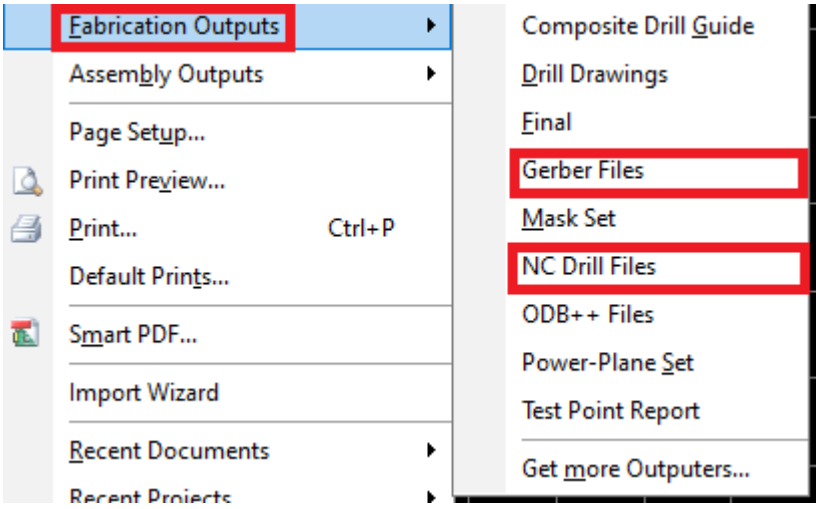


Figura 38. Archivos de fabricación.

Existen otros archivos que podemos generar en Altium, como el presupuesto, o un archivo que contenga la comprobación de las reglas de diseño.

6.2.1 Prototipo del Shield para el núcleo de STM32WB55.

Para el diseño del prototipo del Shield correspondiente al núcleo, y con el fin de evitar errores innecesarios con las medidas, hemos tomado como referencia el archivo de recursos correspondiente a las especificaciones de manufactura de la placa, en este caso la correspondiente a MB1355 [21].

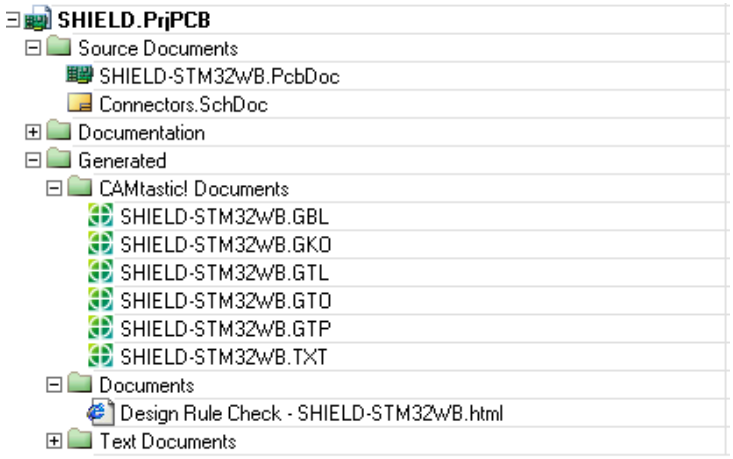


Figura 39. Directorio de diseño Shield.

En la Figura 39 tenemos el directorio del diseño del Shield, en el cual tenemos los documentos fuente, que contiene el esquema (Connectors) y el PCB (SHIELD-STM32WB). Además, contiene archivos generados, los gerber, que son los documentos CAMtastic de las diferentes capas que vamos a fabricar, y la comprobación de reglas de diseño.

En la Figura 40 tenemos el esquema inicial, que corresponde al Shield, sobre el cual hemos dejado solo los pines que vamos a utilizar. Hemos considerado conectores para los sensores que usen un protocolo I^2C, cinco LED de trabajo y un conector de 8 pines para futuras aplicaciones SPI, además de los pines necesarios de LPUART y UART para la comunicación Bluetooth. Los conectores de Arduino en principio no se usarán, por lo que se han desconectado.

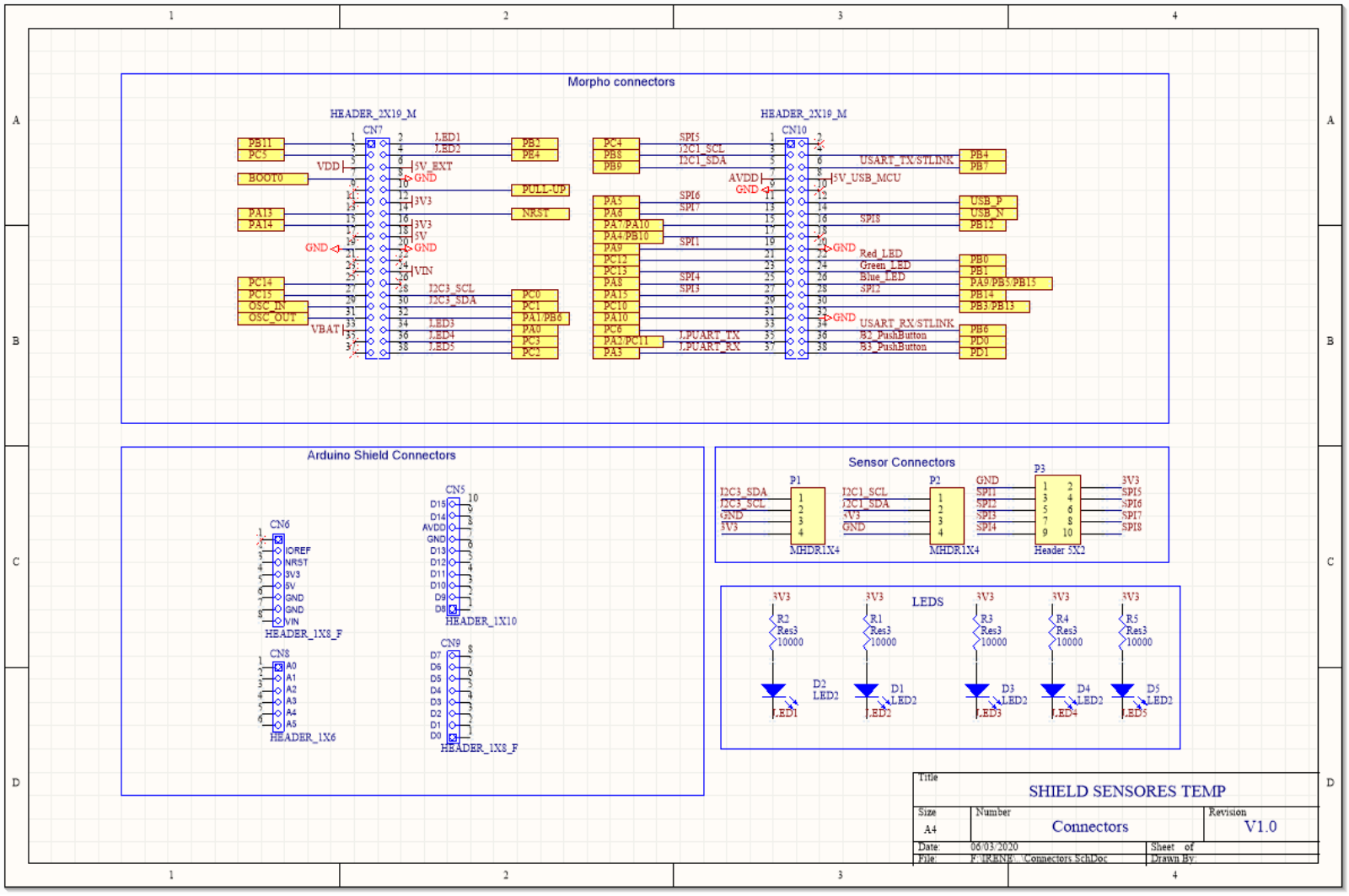


Figura 40. Esquema inicial conectores Shield para el núcleo STM32WB55.

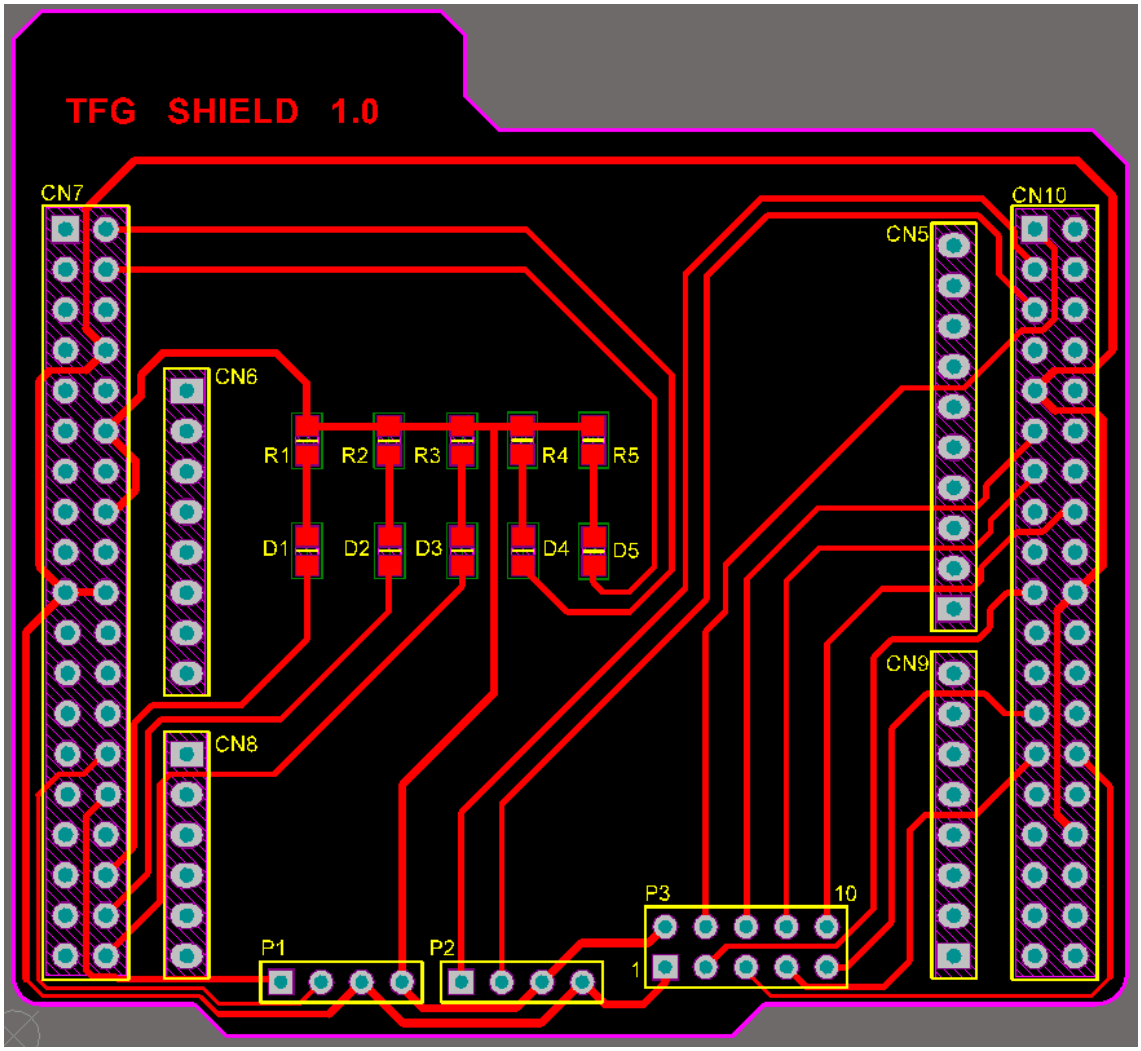


Figura 41. Versión 1.0 del Shield para el núcleo STM32WB55.

En la Figura 41 tenemos la primera versión del Shield rutado, todo en una única capa para que facilite la fabricación, ya que es un prototipo para pruebas.

Tabla VII. Asignación inicial de los pines en microcontrolador.

PROTOCOLO		HEADER	MICRO	NOMBRE
I2C SMBus Two Wire Interface	I2C1	3 - CN10	PB8	I2C1_SCL
		5 - CN10	PB9	I2C1_SDA
	I2C3	28 -CN7	PC0	I2C3_SCL
		30 - CN7	PC1	I2C3_SDA
GPIO output		34 -CN7	PA0	LED1
		36 - CN7	PC3	LED2
		38 - CN7	PC2	LED3
		2 -CN7	PB2	LED4
		4 -CN7	PE4	LED5
USART		34 - CN10	PB6	USART_RX
		6 - CN10	PB7	USART_TX
LPUART		35 - CN10	PA2	LPUART_TX
		37 - CN10	PA3	LPUART_RX
SPI	SPI1	1 - CN10	PC4	Sin definir
		11 - CN10	PA5	Sin definir
		13 - CN10	PA6	Sin definir
		16 - CN10	PB12	Sin definir
	SPI2	19 - CN10	PA9	Sin definir
		28 - CN10	PB14	Sin definir
		27 - CN10	PA15	Sin definir
		25 - CN10	PA8	Sin definir

La asignación inicial de los pines realizada se detalla en la Tabla VII. Posteriormente con esta tabla haremos la asignación de pines en la aplicación STM32CubeMX, como ya hemos explicado anteriormente.

Una aproximación del producto final la vemos en el modelo 3D de la Figura 42.

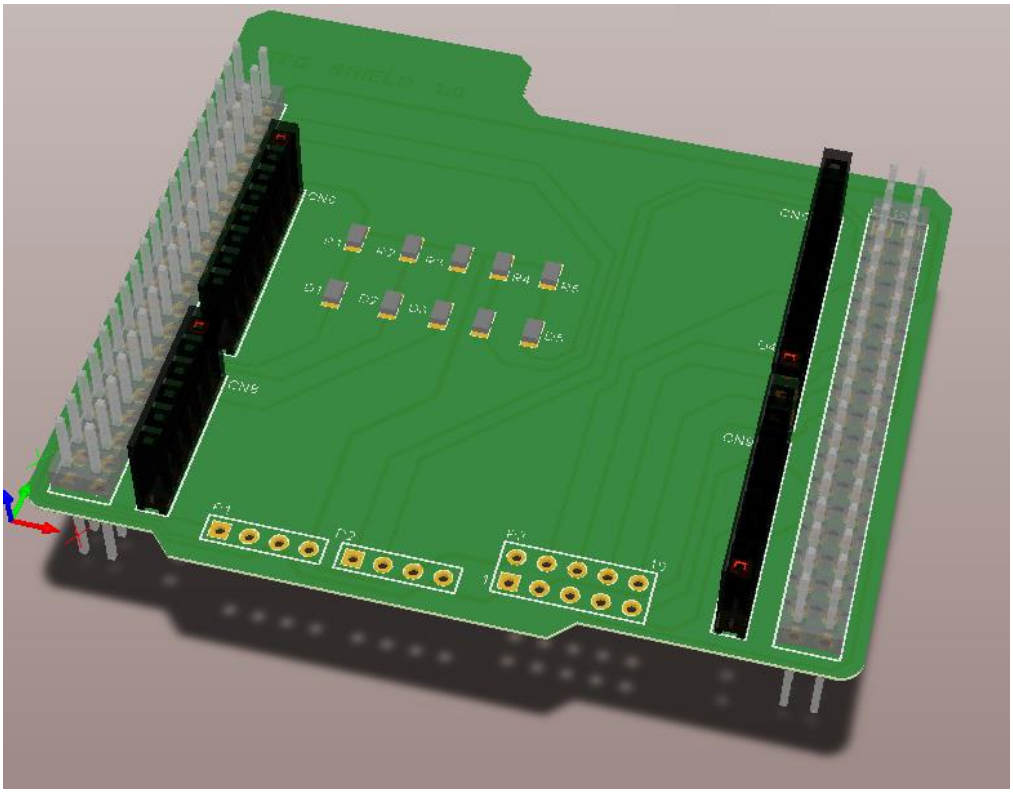


Figura 42. Modelo 3D Shield núcleo STM32WB55.

En este Shield la asignación de los pines físicos se recoge en la Tabla VIII. También lo representamos en la Figura 43.

Tabla VIII. Asignación pines físicos Núcleo STM32WB55.

CONECTOR 1 (P1)	
PIN 1	SDA 3
PIN 2	SCL 3
PIN 3	GND
PIN 4	3.3 V
CONECTOR 2 (P2)	
PIN 1	SCL 1
PIN 2	SDA 1
PIN 3	3.3 V
PIN 4	GND
CONECTOR 3$_{(2)}$ (P3)	
PIN 1	GND

PIN 2	3.3 V
PIN 3	SPI 1
PIN4	SPI 2
PIN 5	SPI3
PIN 6	SPI 4
PIN 7	SPI 5
PIN 8	SPI 6
PIN 9	SPI 7
PIN 10	SPI 8

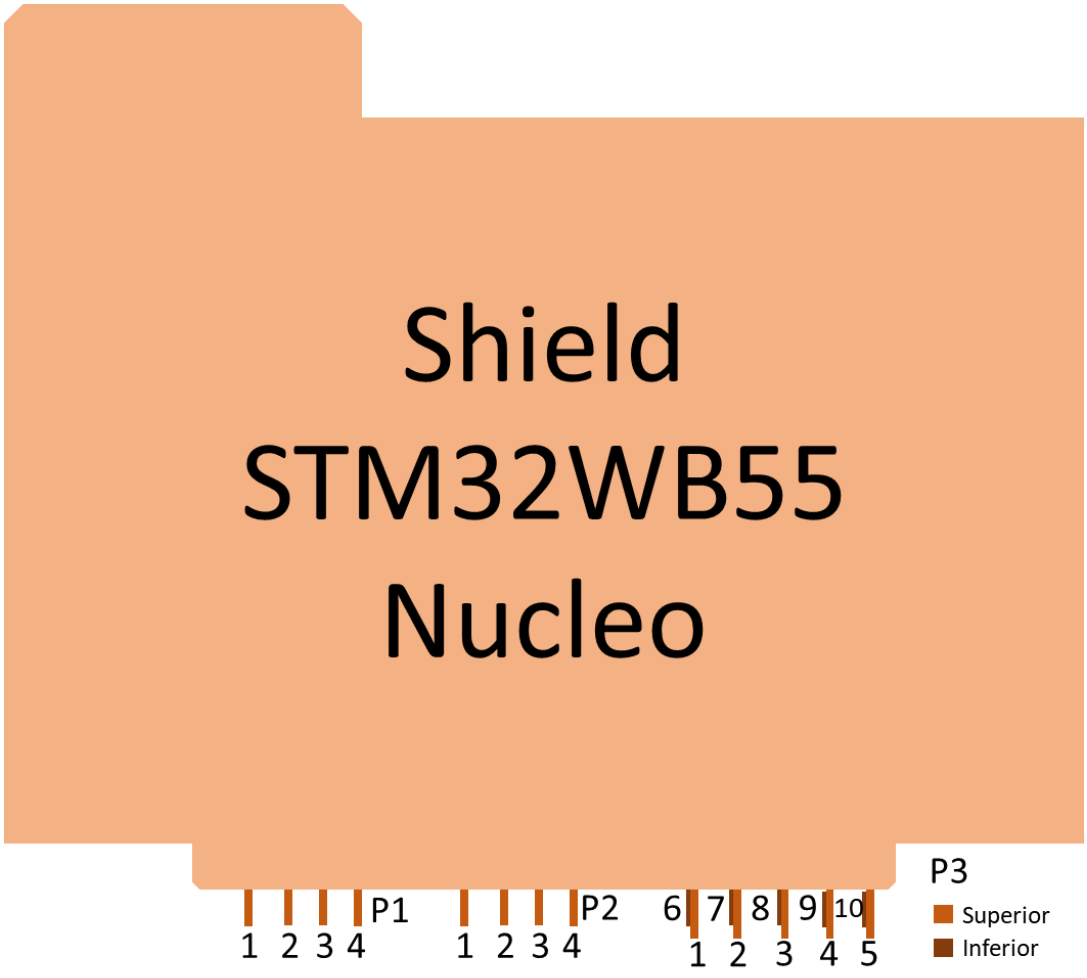


Figura 43. Representación Shield Núcleo STM32WB55.

6.2.2 Prototipo del PCB para sensores

De igual manera, el directorio para la PCB de los sensores es el mostrado en la Figura 44.

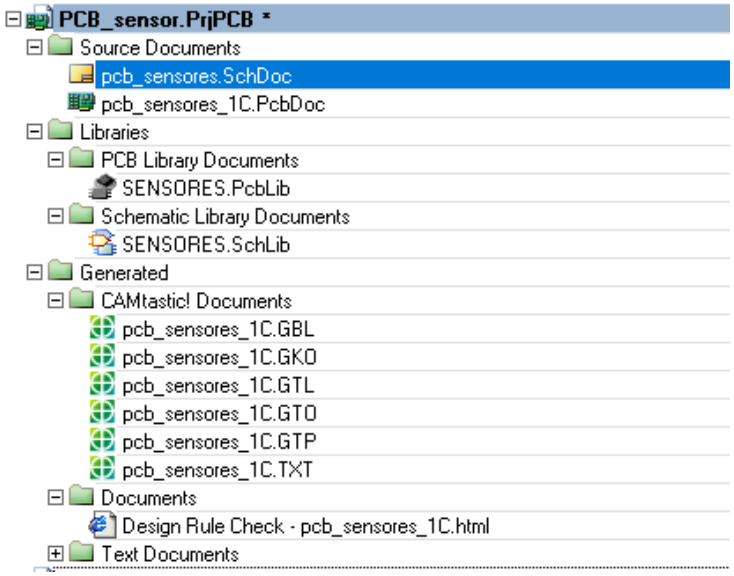


Figura 44. Directorio PCB sensores.

El diseño inicial para pruebas de los sensores se basará en los ya estudiados de la gama Melexis. Para esta PCB hemos tenido en cuenta el esquema propuesto de conexiones dado por el fabricante para trabajar en SMBus, protocolo compatible con el microprocesador, detallado en la Figura 45.

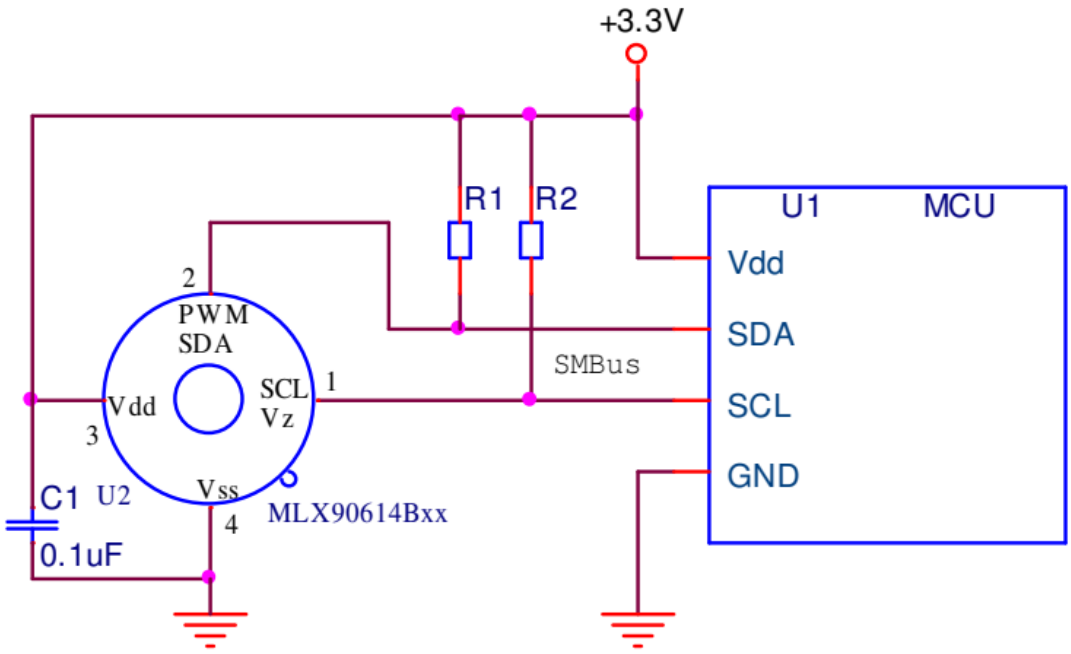


Figura 45. Conexión SMBus para sensor MLX90614 [15].

Para este diseño no teníamos huella disponible para el sensor, así que hemos consultado librerías en línea [38] para obtenerla, sobre la cual hemos comprobado también las medidas de separación de los pines, vital para la fabricación. En la Figura 46 la vista con medidas facilitada por el fabricante, y en la Figura 47 tenemos la huella de Altium con la distancia entre pines marcada, viendo que los diámetros coinciden.

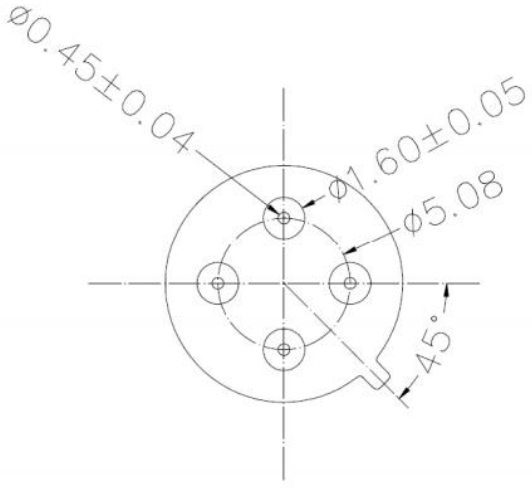


Figura 46. Medidas sensores Melexis [15].

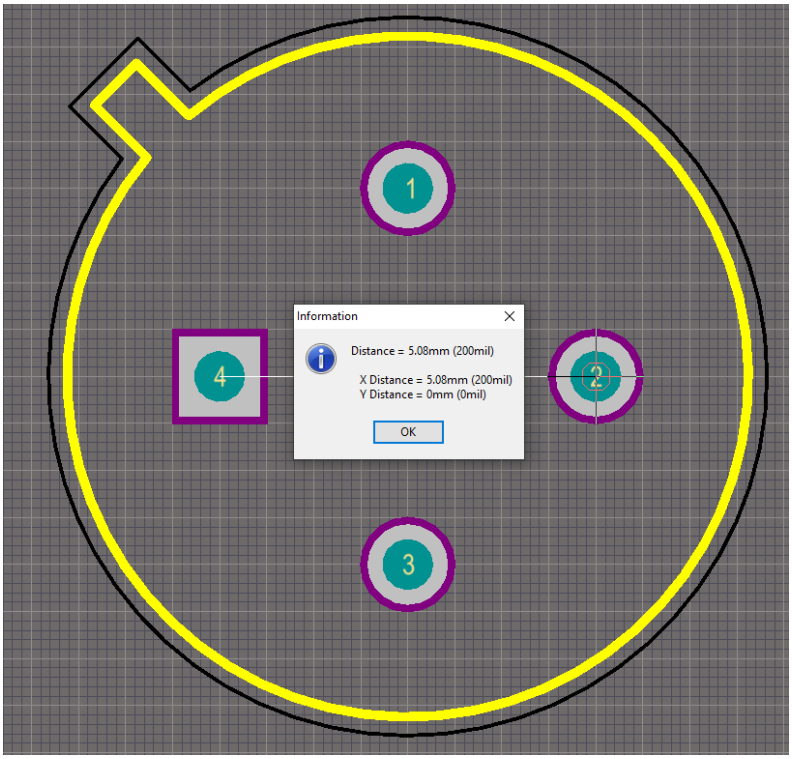


Figura 47. Huella Altium Melexis.

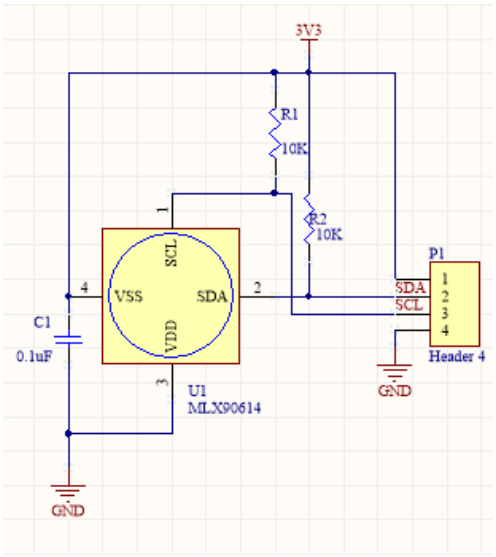


Figura 48. Esquema PCB sensores.

En la Figura 48 tenemos el esquema del primer prototipo para la PCB de los sensores, mientras que en la Figura 49 tenemos el diseño a una cara de esta, con el mismo fin de facilitar las cosas, ya que solo se trata de un prototipo.

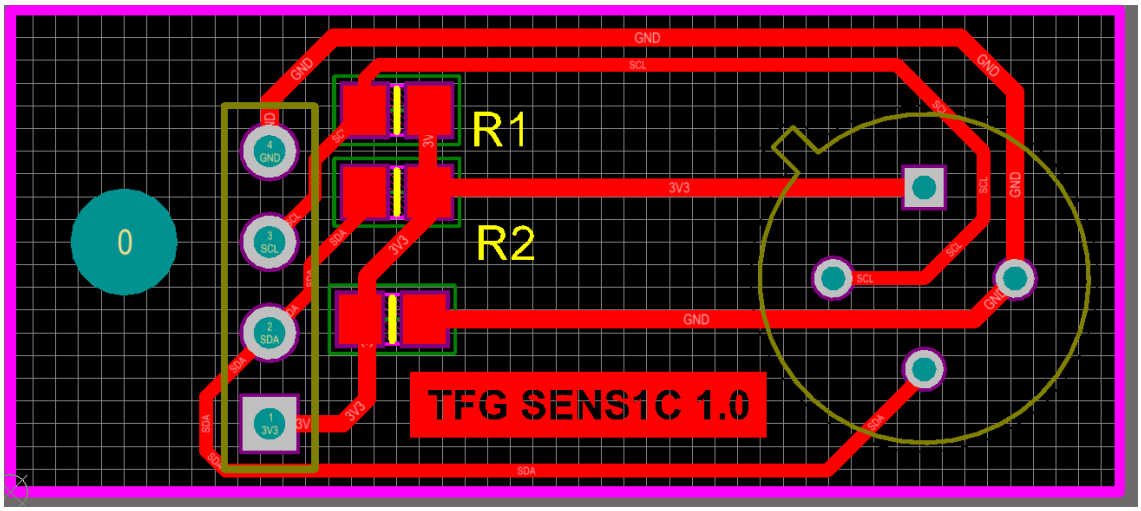


Figura 49. PCB sensores.

Ilustramos el resultado de fabricación en la Figura 50.

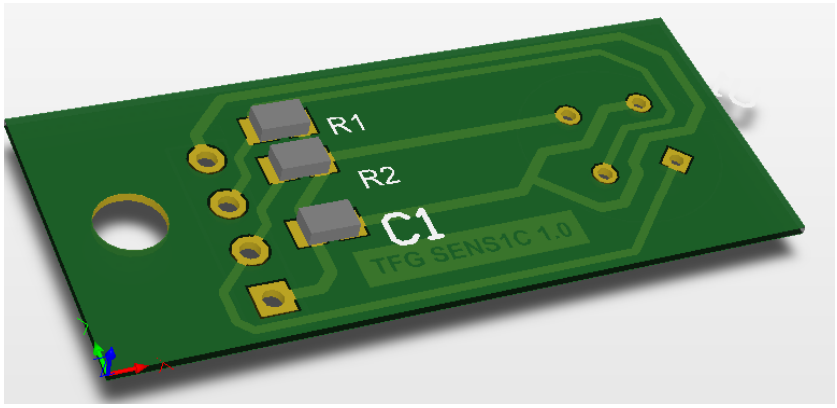


Figura 50. Modelo 3D PCB sensores.

6.2.3 Prototipo de placa de adaptación

Una de las formas de programar el adaptador USB es a través del ST-LINK y los pines de extensión. Esta manera es la más eficaz, pues nos permite adicionalmente depurar el código. Para ello, necesitamos una placa de adaptación, y obtener las señales del núcleo, las cuales se hallan puenteadas. Visualmente lo vemos en la Figura 51.

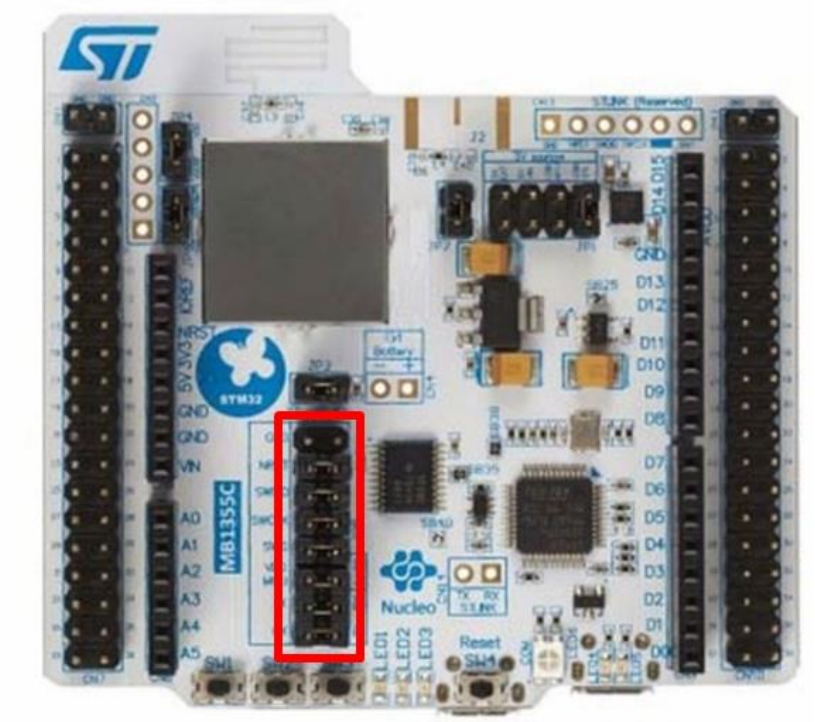


Figura 51. Señales para ST-LINK.

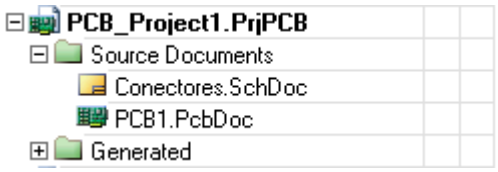


Figura 52. Directorio placa de adaptación.

En la Figura 52 tenemos el directorio de nuestra placa de adaptación, que incluye el esquema con los conectores y la PCB. Para el diseño de esta última, como en la fabricación del Shield, hemos tomado en los recursos de la web el archivo correspondiente a los datos de manufactura del adaptador, en este caso MB1293C [21].

En la Figura 53 mostraremos el esquema con la distribución de señales. Además de las necesarias para la programación, hemos añadido una salida de I^2C para poder realizar pruebas adicionales.

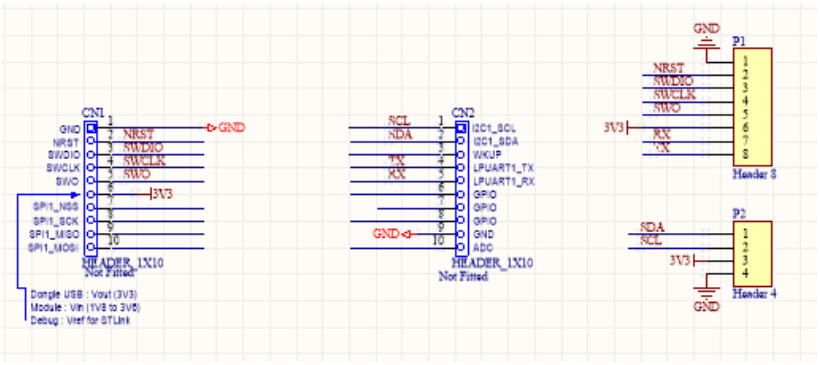


Figura 53. Esquema placa de adaptación.

En la Figura 54 dispondremos del modelo de la PCB realizada, con el enrutado pertinente. Es una configuración sencilla, ya que hemos precisado de pocas señales. En ella, pincharemos el adaptador USB para realizar la programación, o utilizar uno de los sensores.

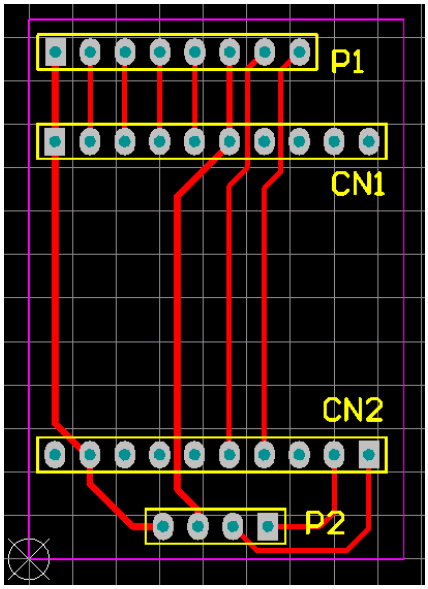


Figura 54. PCB placa de adaptación.

6.2.4 Prototipo del Shield para el ESP32

Para trabajar con el ESP32 hemos realizado también un Shield o placa de expansión. En la Figura 55 tenemos el diseño del esquema, con las señales necesarias para trabajar con este módulo.

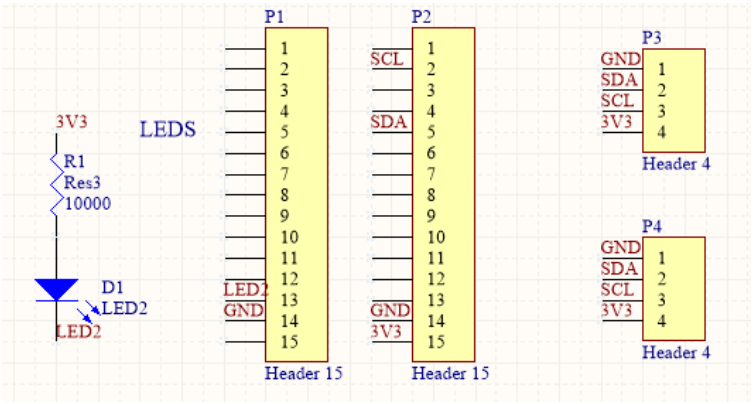


Figura 55. Esquema Shield ESP32.

En este caso, y como el ESP32 tiene una única línea designada para I^2C, hemos sacado las señales D21 y D22, que corresponden a SCL y SDA respectivamente, hacia dos entradas, que es donde conectaremos, por un lado, el sensor, y por otro lado, el *array* de sensores. Adicionalmente hemos añadido un LED de control, que saldrá de la señal D13. Posteriormente, hemos creado la PCB a partir de esta información, ilustrada en la Figura 55.

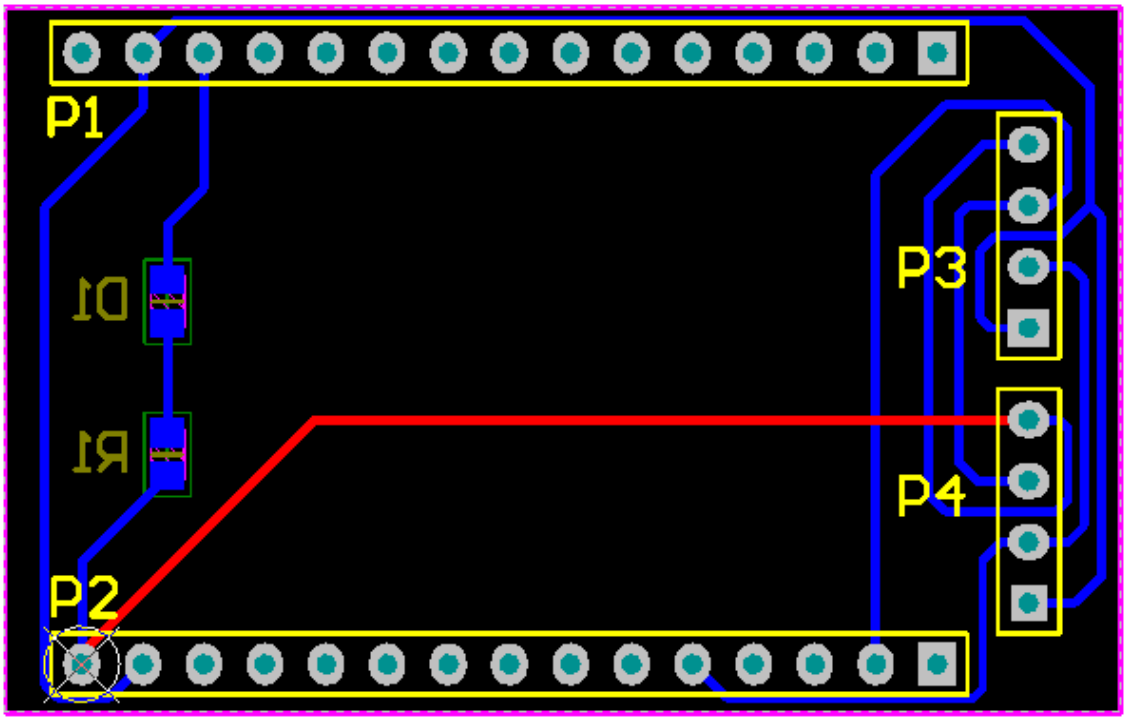


Figura 56. PCB para Shield de ESP32.

En Figura 57 la vemos el modelo 3D de este diseño, aproximando el resultado de fabricación.

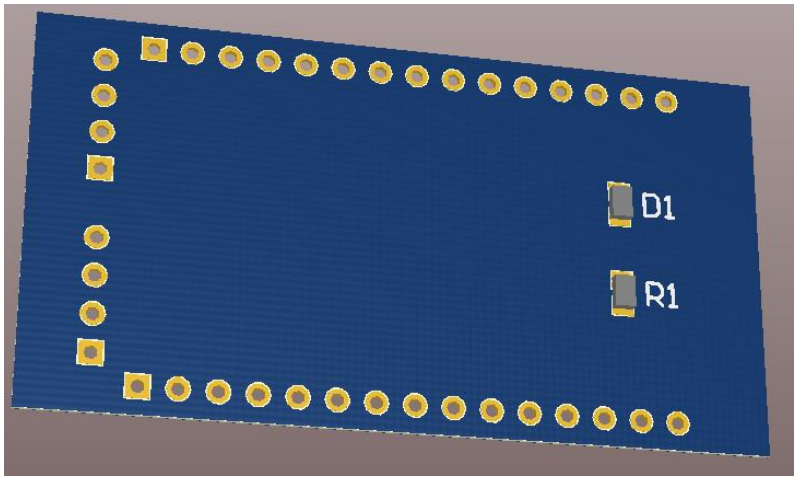


Figura 57. Modelo 3D Shield ESP32.

La asignación de pines que hemos hecho en este caso se recoge en la Tabla IX, y se ilustra en la Figura 58.

Tabla IX. Asignación conectores ESP32.

CONECTOR 1 (P1)		
PIN 1	3V3	3.3 V
PIN 2	D22	SCL
PIN 3	D21	SDA
PIN 4	GND	GND
CONECTOR 2 (P2)		
PIN 1	3V3	3.3 V
PIN 2	D22	SCL
PIN 3	D21	SDA
PIN 4	GND	GND

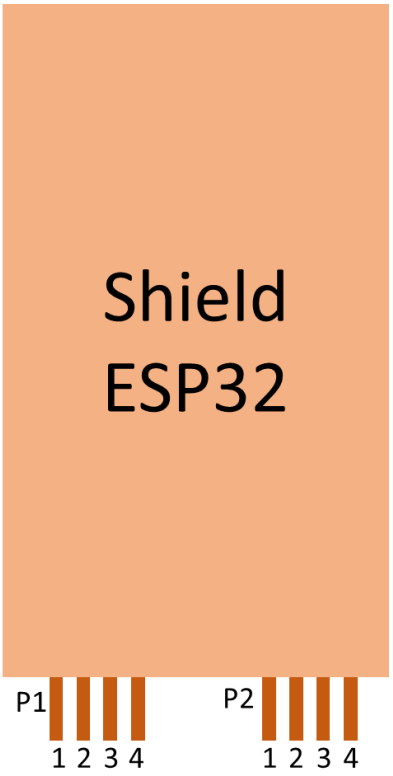


Figura 58. Representación Shield ESP32.

6.3 CircuitCAM 5.0

CircuitCAM es un programa que combina herramientas CAD y CAM. Podemos utilizarlo para importar, comprobar y editar datos de producción de placas de circuitos impresos en diferentes formatos CAM, y exportarlos de nuevo en este formato. Es particularmente útil para calcular los canales de aislamiento entre las pistas conductoras en un prototipo de placa de circuito impreso usando fresadoras de LPKF [39], que será quien nos proporcione este software al usar la máquina fresadora del mismo.

Nosotros usaremos este programa para fabricar de las PCB anteriormente mencionadas. Antes que nada, debemos importar una a una las capas que deseamos fabricar, los gerber que ya habremos generado en el Altium, además de los taladros. También debemos decirle al programa a qué corresponde cada uno de los archivos importados. Principalmente importaremos 4 archivos: TopLayer (capa superior), BottomLayer (capa inferior), DrillPlated (taladros) y BoardOutline (bordes externos de la PCB). En la Figura 59 vemos el importador

de los archivos con los que vamos a trabajar y las diferentes opciones que se contemplan.

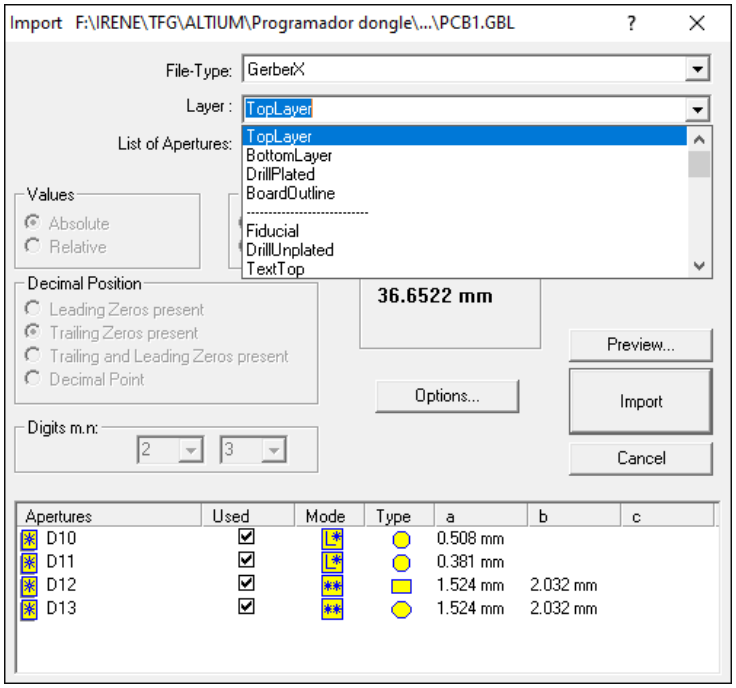


Figura 59. Importador CircuitCam 5.0.

Cuando ya tenemos las capas necesarias puestas, seleccionaremos el entorno donde queremos trabajar y aislaremos todas las capas. También borraremos las capas que no son necesarias, y daremos algunos valores a los taladros y fresas que utilizaremos en la fabricación, para que la fresadora nos facilite la misma. Una vez realizamos todo esto tenemos nuestro diseño listo para la fabricación. Realizaremos unos cortes en los extremos para que se sostenga la PCB durante la fabricación, pero lo suficientemente pequeños como para que nos sea fácil separarla del material de fabricación.

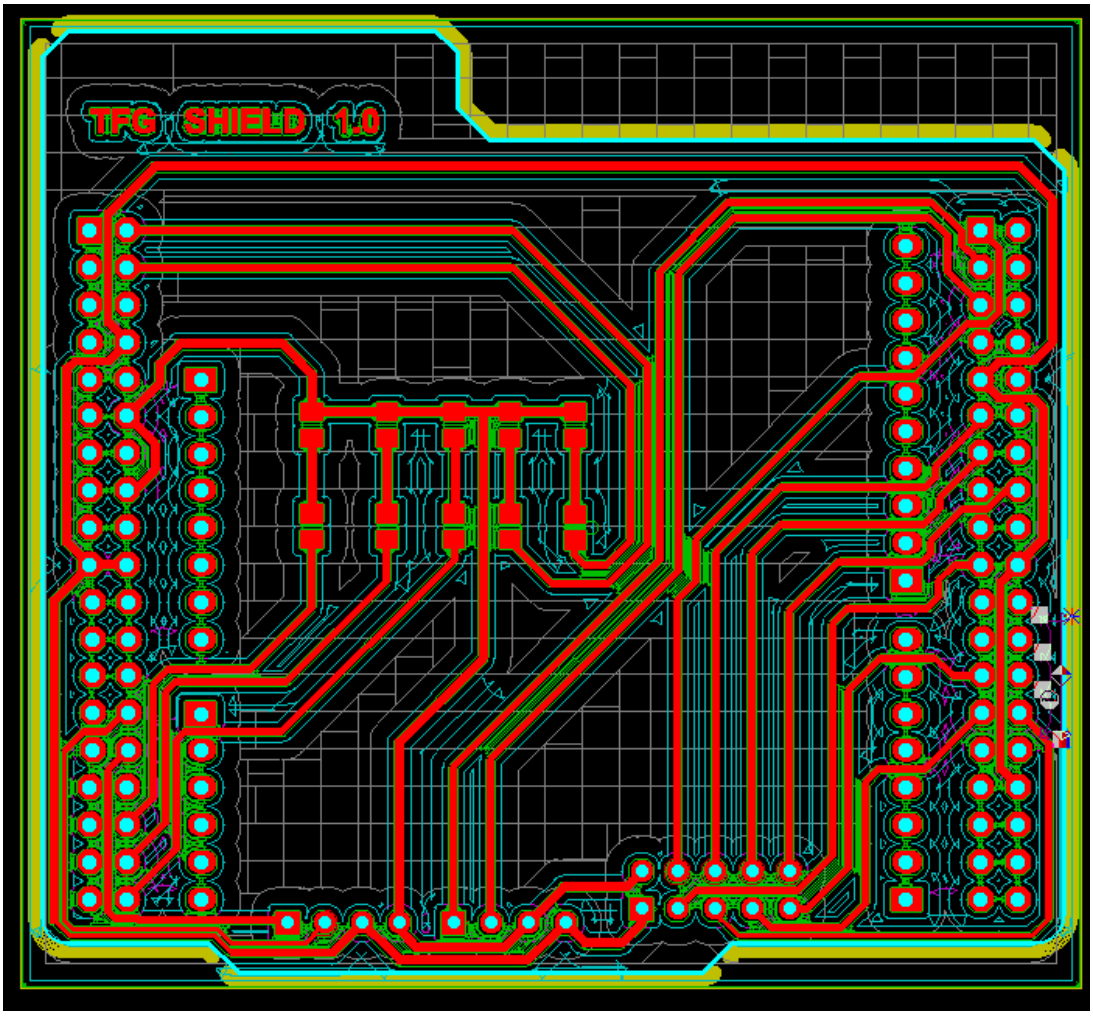


Figura 60. Archivo de fabricación Shield.

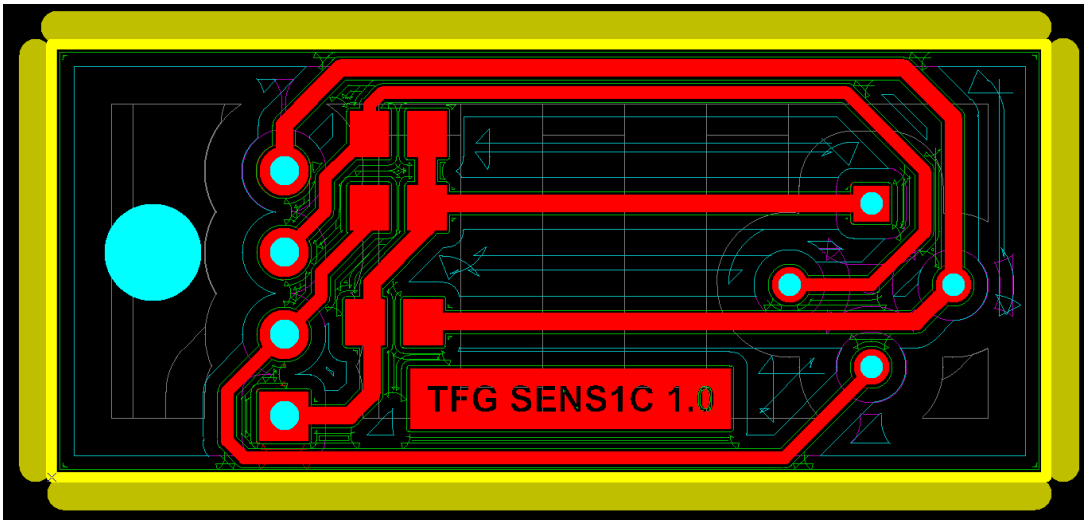


Figura 61. Archivo de fabricación de la PCB para sensores.

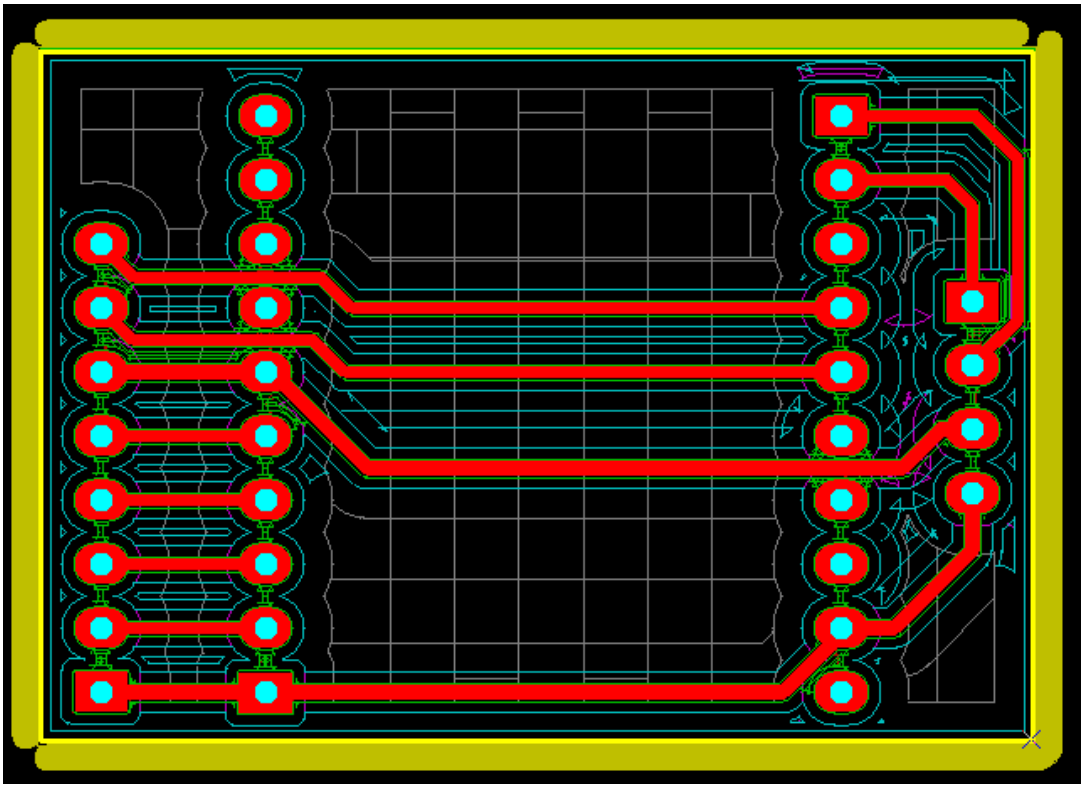


Figura 62. Archivo de fabricación de la placa de adaptación.

En las Figura 60, Figura 61 y Figura 62 tenemos una captura de los archivos de fabricación, que serán exportados en formato CAM para importarlos posteriormente en la fresadora y realizar la fabricación.

6.5 Fabricación del Shield

Para la fabricación del Shield final partimos de la Figura 56, en la que tenemos el diseño de la PCB. Como hemos comentado en el apartado anterior, debemos generar los archivos de fabricación, mostrado en la Figura 63.

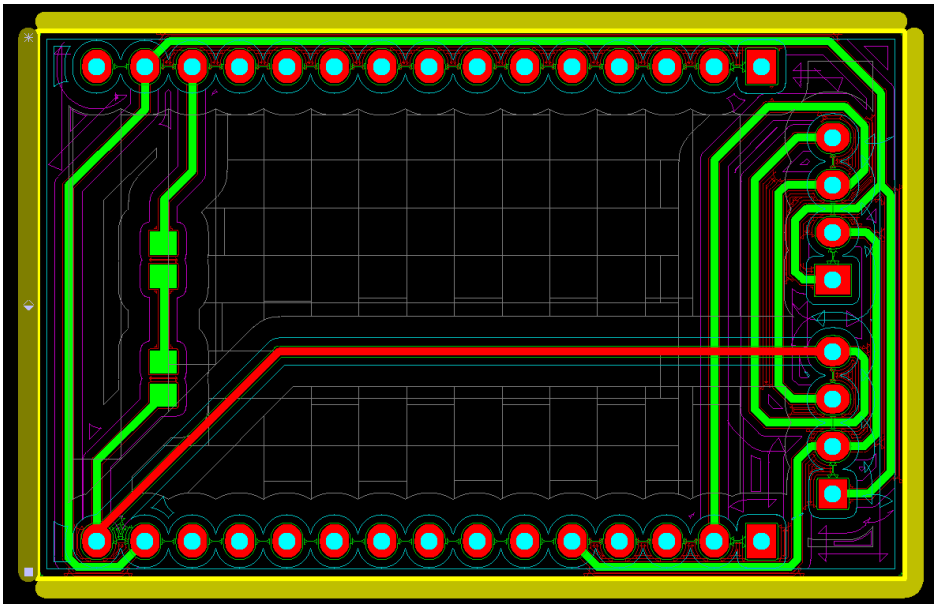


Figura 63. Archivo de fabricación Shield ESP32.

Posteriormente hemos hecho uso de la máquina de fresado del laboratorio, a la cual le hemos cargado los archivos de fabricación. Una vez obtenida la PCB, en la estación de soldadura, hemos soldado los pines, el led y la resistencia. El resultado de este proceso se ilustra en la Figura 64.

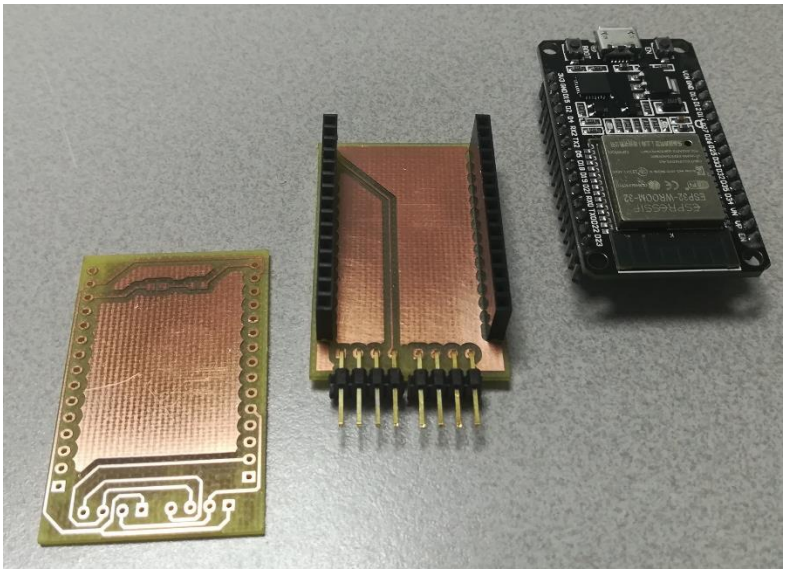


Figura 64. PCBs fabricadas y ESP32.

De izquierda a derecha, tenemos la PCB fabricada, la que ya está soldada y el dispositivo ESP32, que es donde se va a conectar.

6.4 Conclusiones

En este capítulo hemos realizado una vista general de cómo realizar un proceso de diseño global para cualquier PCB, concretando también sobre nuestro proyecto. Para ello hemos explicado los programas utilizados, Altium Release 10 y CircuitCam 5.

Capítulo 7. Integración en las plataformas

7.1 Introducción

En este capítulo trataremos el desarrollo del firmware para nuestras plataformas, desde la definición de sus pines hasta la completa integración en la misma. Además, introduciremos el bus I^2C, concretando en SMBus, y las comunicaciones Bluetooth.

La estrategia de desarrollo a seguir ha sido, primero transmitir los datos de un único sensor a través del puerto serial usando la USART. Posteriormente hemos añadido un segundo sensor, implementando la comunicación en el puerto serial con el USB, con el fin de que no sea necesario utilizar un conversor RS-232. Finalmente, hemos abordado las comunicaciones BLE.

7.2 Definición de pines

En la Figura 65 mostramos la definición de los pines del microprocesador correspondiente a la placa principal, con cada una de las señales utilizadas para el funcionamiento de nuestro sistema electrónico.

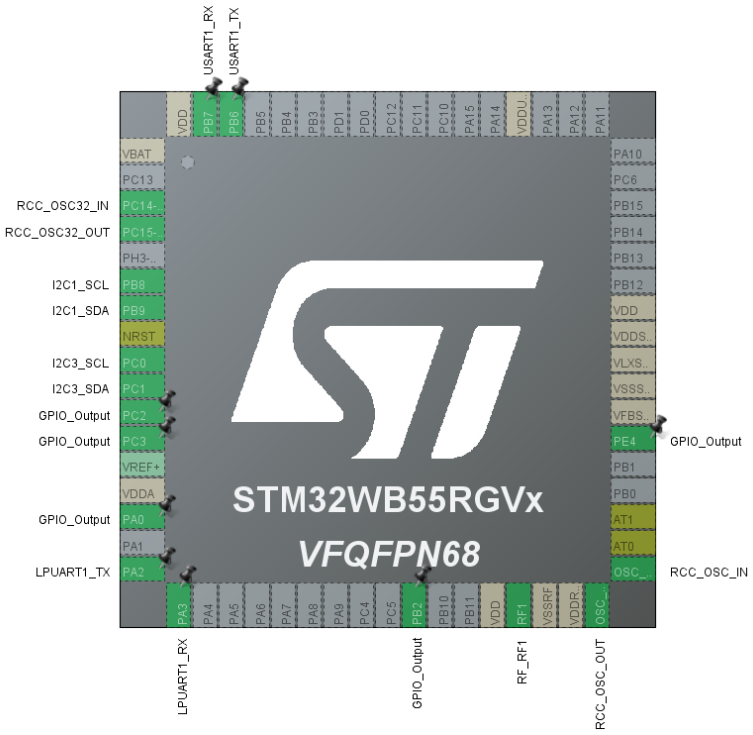


Figura 65. Asignación de pines en el microprocesador.

7.3 Desarrollo del firmware

8.3.3 Comunicación con los sensores

Vamos a dar una breve explicación de las librerías utilizadas por los sensores para desarrollar tanto las pruebas como el código final.

7.3.3.1 Sensores MLX90614

Para comunicarnos con los sensores MLX90614 hemos usado la librería desarrollada por Adafruit. Los registros de este sensor son los siguientes, remarcados en amarillo los que vamos a leer a lo largo del programa:

Memoria RAM:

```
#define MLX90614_RAWIR1 0x04
#define MLX90614_RAWIR2 0x05
#define MLX90614_TA 0x06
#define MLX90614_TOBJ1 0x07
#define MLX90614_TOBJ2 0x08
```

Memoria EEPROM:

```
#define MLX90614_TOMAX 0x20
#define MLX90614_TOMIN 0x21
#define MLX90614_PWMCTRL 0x22
#define MLX90614_TARANGE 0x23
#define MLX90614_EMISS 0x24
#define MLX90614_CONFIG 0x25
#define MLX90614_ADDR 0x2E
#define MLX90614_ID1 0x3C
#define MLX90614_ID2 0x3D
#define MLX90614_ID3 0x3E
#define MLX90614_ID4 0x3F
```

Además, también se nos define la dirección I^2C del dispositivo.

```
#define MLX90614_I2CADDR 0x5A
```

La librería incluye también diversas funciones para leer el sensor, pero las que vamos a utilizar son las siguientes:

```
bool Adafruit_MLX90614::begin(void) {
  Wire.begin();
  return true;
}
```

Esta función comienza las comunicaciones de I^2C haciendo uso del comando Wire.begin(). Además, devuelve verdadero cuando se ha inicializado.

```
double Adafruit_MLX90614::readEmissivity(void) {
  uint16_t ereg = read16(MLX90614_EMISS);
  return ((double)ereg) / 65535.0;
}
```

```
double Adafruit_MLX90614::readObjectTempC(void) {
  return readTemp(MLX90614_TOBJ1);
}
```

```
double Adafruit_MLX90614::readAmbientTempC(void) {
  return readTemp(MLX90614_TA);
}
```

En estas tres funciones devolvemos la lectura de los registros correspondientes a la emisividad, la temperatura ambiente y la temperatura del objeto, de las que haremos uso en este proyecto. Las siguientes dos funciones son las funciones que utilizamos para acceder a los registros que indicamos al principio de este apartado, que contienen la información necesaria. Estas funciones pasan por parámetros el registro de lectura.

```
float Adafruit_MLX90614::readTemp(uint8_t reg) {
  float temp;
  temp = read16(reg);
  temp *= .02;
  temp -= 273.15;
  return temp;
}
```

```
uint16_t Adafruit_MLX90614::read16(uint8_t a) {
  uint16_t ret;
  Wire.beginTransmission(_addr);
  Wire.write(a);
  Wire.endTransmission(false);
  Wire.requestFrom(_addr, (size_t)3);
  ret = Wire.read();
  ret |= Wire.read() << 8;
  uint8_t pec = Wire.read();
  return ret;
}
```

7.3.3.2 *Array* de sensores AMG8833

La comunicación con el *array* de sensores AMG8833 también tiene una librería diseñada por Adafruit. De igual manera que con los sensores de temperatura, vamos a trabajar con una dirección I^2C.

```
#define AMG88xx_ADDRESS (0x69)
```

En este caso, la librería estructura los registros con enumeraciones. De igual manera, remarcamos las que usa el proyecto:

```
enum {
  AMG88xx_PCTL = 0x00,
  AMG88xx_RST = 0x01,
  AMG88xx_FPSC = 0x02,
  AMG88xx_INTC = 0x03,
  AMG88xx_STAT = 0x04,
  AMG88xx_SCLR = 0x05,
  // 0x06 reserved
  AMG88xx_AVE = 0x07,
  AMG88xx_INTHL = 0x08,
  AMG88xx_INTHH = 0x09,
  AMG88xx_INTLL = 0x0A,
  AMG88xx_INTLH = 0x0B,
  AMG88xx_IHYSL = 0x0C,
  AMG88xx_IHYSH = 0x0D,
  AMG88xx_TTHL = 0x0E,
  AMG88xx_TTHH = 0x0F,
  AMG88xx_INT_OFFSET = 0x010,
  AMG88xx_PIXEL_OFFSET = 0x80
};
```

```
enum power_modes {
  AMG88xx_NORMAL_MODE = 0x00,
  AMG88xx_SLEEP_MODE = 0x01,
  AMG88xx_STAND_BY_60 = 0x20,
  AMG88xx_STAND_BY_10 = 0x21
};
```

```
enum sw_resets {
AMG88xx_FLAG_RESET = 0x30,
AMG88xx_INITIAL_RESET = 0x3F
};
```

```
enum frame_rates {
AMG88xx_FPS_10 = 0x00,
AMG88xx_FPS_1 = 0x01
};
```

```
enum int_enables {
AMG88xx_INT_DISABLED = 0x00,
 AMG88xx_INT_ENABLED = 0x01
};
```

```
enum int_modes {
AMG88xx_DIFFERENCE = 0x00,
AMG88xx_ABSOLUTE_VALUE = 0x01
};
```

Además, define unas constantes para trabajar con los sensores:

```
#define AMG88xx_PIXEL_ARRAY_SIZE 64
#define AMG88xx_PIXEL_TEMP_CONVERSION .25
```

Para inicializar el dispositivo haremos uso de la función begin dada por la librería. Pasaremos por parámetros la dirección I^2C, aunque hay una definida por defecto. Nos devuelve true si se inició correctamente.

```
bool Adafruit_AMG88xx::begin(uint8_t addr) {
  _i2caddr = addr;
  _i2c_init();
  _pctl.PCTL = AMG88xx_NORMAL_MODE;
  write8(AMG88xx_PCTL, _pctl.get());
  _rst.RST = AMG88xx_INITIAL_RESET;
  write8(AMG88xx_RST, _rst.get());
  disableInterrupt();
  _fpsc.FPS = AMG88xx_FPS_10;
  write8(AMG88xx_FPSC, _fpsc.get());
  delay(100);
  return true;
  }
```

Y para leer el sensor usamos la función readPixels, que pasa por parámetros el buffer donde vamos a almacenar el resultado, y el tamaño de ese buffer.

```
void Adafruit_AMG88xx::readPixels(float *buf, uint8_t size) {
  uint16_t recast;
  float converted;
  uint8_t bytesToRead =
     min((uint8_t)(size << 1), (uint8_t)(AMG88xx_PIXEL_ARRAY_SIZE << 1));
  uint8_t rawArray[bytesToRead];
  this->read(AMG88xx_PIXEL_OFFSET, rawArray, bytesToRead);
  for (int i = 0; i < size; i++) {
    uint8_t pos = i << 1;
    recast = ((uint16_t)rawArray[pos + 1] << 8) | ((uint16_t)rawArray[pos]);
    converted = int12ToFloat(recast) * AMG88xx_PIXEL_TEMP_CONVERSION;
    buf[i] = converted;
  }
}
```

7.3.4 Comunicación usando la UART.

Para comunicarnos de manera alámbrica, hemos añadido en el código, tanto del cliente como del servidor la siguiente función:

```
void printSensor() {
  Serial.print("#E1");
  Serial.print(mlx.readEmissivity());
  Serial.println(";");
  Serial.print("#A1");
  Serial.print(mlx.readAmbientTempC());
  Serial.println(";");
  Serial.print("#O1");
  Serial.print(mlx.readObjectTempC());
  Serial.println(";");
  String p = "";
  for (int i = 1; i <= AMG88xx_PIXEL_ARRAY_SIZE; i++) {
    p += String(pixels[i - 1], 2) + ",";
  String pixels1 = "#P1" + p + ";";
 Serial.println(pixels1);
}
```

Este fragmento de código se ejecutará cuando la conexión BLE no esté establecida.

7.3.5 Comunicaciones Bluetooth

Para las comunicaciones BLE hemos utilizado los ejemplos disponibles en el entorno de Arduino, tanto del servidor como del cliente. Vamos a mostrar el código de ambos. Adicionalmente, se muestran mensajes de depurado a lo largo de la ejecución de los dos.

En el servidor, primero, incluimos las librerías necesarias:

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

Definimos las constantes correspondientes al UUID del servicio y la característica:

```
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

Se define el puntero a la característica BLE. Aquí vamos a recibir los datos del cliente.

```
static BLECharacteristic *pCharacteristic;
```

En el setup del programa vamos a hacer varias cosas. Antes que nada, iniciamos las comunicaciones seriales para poder depurar el programa. Después, creamos el servidor dándole un nombre. Creamos el servicio y la característica, a la cual le damos un valor inicial, y corremos el servidor. Una vez funcionando, comenzamos a mostrarnos a otros dispositivos, añadiendo el servicio a la lista de objetos que mostramos al exterior.

```
void setup() {
  Serial.begin(115200);
  BLEDevice::init("Long name works now");
  BLEServer *pServer = BLEDevice::createServer();
  Definimos el dispositivo como tal.
  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(CHARACTERISTIC_UUID,
  BLECharacteristic::PROPERTY_READ |
  BLECharacteristic::PROPERTY_WRITE);
  pCharacteristic->setValue("Characteristic defined");
  pService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
}
```

Por último, en el bucle principal vamos obteniendo el valor de la característica y la imprimimos en pantalla, de manera que podemos ver continuamente qué estamos recibiendo.

```
void loop() {
  std::string characteristic = pCharacteristic->getValue();
    Serial.println(characteristic.c_str());
  delay(1000);
}
```

De manera análoga, en el cliente incluimos la librería de gestión del BLE.

```
#include "BLEDevice.h"
```

En este caso, definimos el servicio y característica a la que nos vamos a conectar que debe ser la misma que en el servidor, si no, no se van a conectar.

```
static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
static BLEUUID    charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
```

Se definen varias constantes globales. Las tres primeras las usaremos como *flags*, la siguiente es el puntero a la característica BLE. Aquí vamos a enviar los datos al servidor, y la última es el manejador para el servidor.

```
static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;
```

Se prototipan las funciones que se van a desarrollar después para que podamos usarlas en cualquier ámbito del programa.

```
static void notifyCallback( BLERemoteCharacteristic*
pBLERemoteCharacteristic, uint8_t* pData, size_t length, bool isNotify);
bool connectToServer();
```

En el cliente se utilizan dos clases. La primera es la correspondiente al cliente per se, y la segunda es la clase correspondiente al escaneo de servidores BLE. Encuentra el primero que muestra el servicio que buscamos.

```
class MyClientCallback : public BLEClientCallbacks {
    void onConnect(BLEClient* pclient) {
    }
    void onDisconnect(BLEClient* pclient) {
      connected = false;
      Serial.println("onDisconnect");
    }
};
```

```
class MyAdvertisedDeviceCallbacks:
  public BLEAdvertisedDeviceCallbacks {
    void onResult(BLEAdvertisedDevice advertisedDevice) {
      Serial.print("BLE Advertised Device found: ");
      Serial.println(advertisedDevice.toString().c_str());
      if (advertisedDevice.haveServiceUUID() &&
      advertisedDevice.isAdvertisingService(serviceUUID)) {
        BLEDevice::getScan()->stop();
        myDevice = new BLEAdvertisedDevice(advertisedDevice);
        doConnect = true;
        doScan = true;
      }
    }
};
```

```
void setup() {
  Serial.begin(115200);
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new
  MyAdvertisedDeviceCallbacks());
  pBLEScan->setInterval(1349);
  pBLEScan->setWindow(449);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(5, false);
  }
```

En el bucle principal primero vamos a realizar la conexión al servidor BLE. Si nos podemos conectar, actualizaremos el valor de la característica, y la enviaremos al servidor.

```
void loop() {

  if (doConnect == true) {
    if (connectToServer()) {
      Serial.println("We are now connected to the BLE Server.");
     } else {
      Serial.println("We have failed to connect to the server; there is nothing more
      we will do.");
    }
    doConnect = false;
  }
  if (connected) {
    String newValue = "Time since boot: " + String(millis()/1000);
    Serial.println("Setting new characteristic value to \"" + newValue + "\"");
    pRemoteCharacteristic->writeValue(newValue.c_str(), newValue.length());
  }else if(doScan){
     BLEDevice::getScan()->start(0);
  }
  delay(1000);
}
```

Esta función nos sirve para obtener el UUID de la característica obtenida del servidor. Pasa por parámetros la característica, los datos que contiene, su longitud, y si se ha notificado o no.

```
static void notifyCallback( BLERemoteCharacteristic*
pBLERemoteCharacteristic, uint8_t* pData, size_t length, bool isNotify) {
  Serial.print("Notify callback for characteristic ");
  Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());

  Serial.print(" of data length ");
  Serial.println(length);
  Serial.print("data: ");
  Serial.println((char*)pData);
}
```

Esta función del cliente nos devuelve true cuando se hace una conexión exitosa.

```
bool connectToServer() {
  Serial.print("Forming a connection to ");
  Serial.println(myDevice->getAddress().toString().c_str());
  BLEClient*  pClient  = BLEDevice::createClient();
  Serial.println(" - Created client");
  pClient->setClientCallbacks(new MyClientCallback());
  pClient->connect(myDevice);
  Serial.println(" - Connected to server");
  BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
  if (pRemoteService == nullptr) {
    Serial.print("Failed to find our service UUID: ");
    Serial.println(serviceUUID.toString().c_str());
    pClient->disconnect();
    return false;
  }
  Serial.println(" - Found our service");
  pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
  if (pRemoteCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID: ");
    Serial.println(charUUID.toString().c_str());
    pClient->disconnect();
    return false;
     }
  Serial.println(" - Found our characteristic");
  if (pRemoteCharacteristic->canRead()) {
    std::string value = pRemoteCharacteristic->readValue();
    Serial.print("The characteristic value was: ");
    Serial.println(value.c_str());
  }
  if (pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);
  connected = true;
  return true;
}
```

7.3.6 Código final

El código final del proyecto se encuentra anexado a este documento. Por un lado, tenemos el archivo BLE_Server, que contiene el programa servidor comentado y, por otro lado, el archivo BLE_Cliente, que contiene el programa cliente.

7.4 Tramas de la comunicación

El protocolo para la comunicación final de las tramas que hemos definido lo describimos en la Tabla X (los bytes están ordenados desde el más significativo al menos significativo). En la última columna describimos la trama ejemplo:

Tabla X. Tramas del programa.

Byte	Significado	Valor	Comentarios
1	Comienzo de trama	#	Inicia la transmisión
2	Dato del sensor	A	Temperatura ambiente MLX90614
		E	Emisividad MLX90614
		O	Temperatura objeto MLX90614
		P	Temperaturas AMG8833
3 - N-1	Dato	float	Será un único byte si estamos trabajando con los sensores, y en el caso del *array*, será una ráfaga de 64 bytes
N	Fin de trama	;	Finaliza la transmisión

Estas tramas se utilizan para la versión final, correspondiente a los programas ESP32_CLIENTE y ESP32_SERVIDOR. Para el resto de las tramas, anexo a este documento se encuentran los manuales de uso de los programas, en el cual vienen con detalle las otras versiones realizadas.

Un ejemplo de trama recibida sería la siguiente:

#OR110.85;

El análisis de la trama por bytes es:

- ‘#’ Comienzo de trama
- ‘O’ Medida de temperatura de un objeto
- 110.85 Float de la temperatura de ese objeto
- ‘;’ Fin de trama

7.5 Conclusiones

En este capítulo se realiza la descripción de las comunicaciones que vamos a implementar, tanto para la lectura del sensor como para la transferencia de datos. Se ha hecho uso del bus I^2C del microprocesador, para el cual deberemos tener algunas consideraciones adicionales relativas al funcionamiento del sensor, ya descrito en el capítulo 2, como para las comunicaciones con el PC.

En cuanto a la implementación, se ha realizado de manera modular y afrontando problema a problema usando finalmente el entorno Arduino, pues nos ofrece una sencillez de desarrollo del código que no nos da el entorno de STM32WB55. Primero, hemos utilizado las librerías de cada uno de los sensores, entendiendo el funcionamiento por separado, para después integrarlo con la librería de las comunicaciones BLE.

Terminado este capítulo, vamos a mostrar las diferentes pruebas que hemos realizado en las dos plataformas, y luego sobre el sistema real, con una máquina CNC, que se encargará de realizar la imagen térmica.

Capítulo 8. Banco de pruebas

8.1 Introducción

En este capítulo mostraremos las pruebas realizadas del sistema electrónico. Primero realizaremos las pruebas sobre cada sensor, para luego probarlo en el sistema real.

Esperamos principalmente que los sensores nos den una lectura de la temperatura correcta a través del monitor serial, aplicando comunicaciones por cable o inalámbricas.

También trabajaremos con un sistema real, una máquina CNC, que llevará un sistema de control para posicionar la PCB que queremos testear. Tendremos además un software que irá preguntando por los parámetros de temperatura y reconstruirá la imagen térmica.

8.2 Pruebas del sistema electrónico sobre STM32WB55

Para comprobar las lecturas del sensor, hemos añadido en el main del programa unas funciones que, a través de la USART y un puerto serie, lean la petición que le haga el usuario, y muestran los resultados de las temperaturas. Para ello, hemos usado un conversor RS-232.

Con la asignación de pines ya realizada, en el bucle principal implementamos la comunicación.

Primero recibimos los datos del sensor gracias a la librería. Posteriormente, a través de la UART y un terminal serial el usuario introducirá su petición y la recibiremos. Leeremos la trama y la reconstruiremos para no perder ningún dato. Luego compararemos el mensaje, y en función de qué hayamos recibido, transmitiremos al terminal serial el dato. En la Figura 66 mostramos los resultados del sensor. Esta prueba corresponde al programa STM32_UART_1SENSOR.

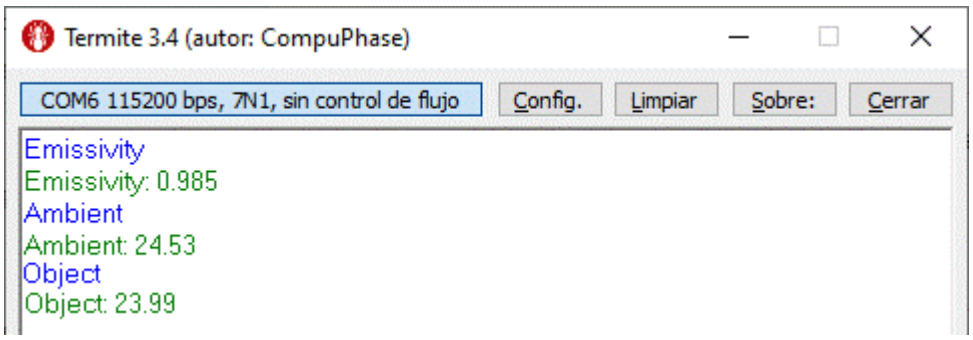


Figura 66. Prueba lectura del sensor por RS-232 en STM32WB55.

Las comunicaciones de esta versión corresponden a las de la Figura 67.

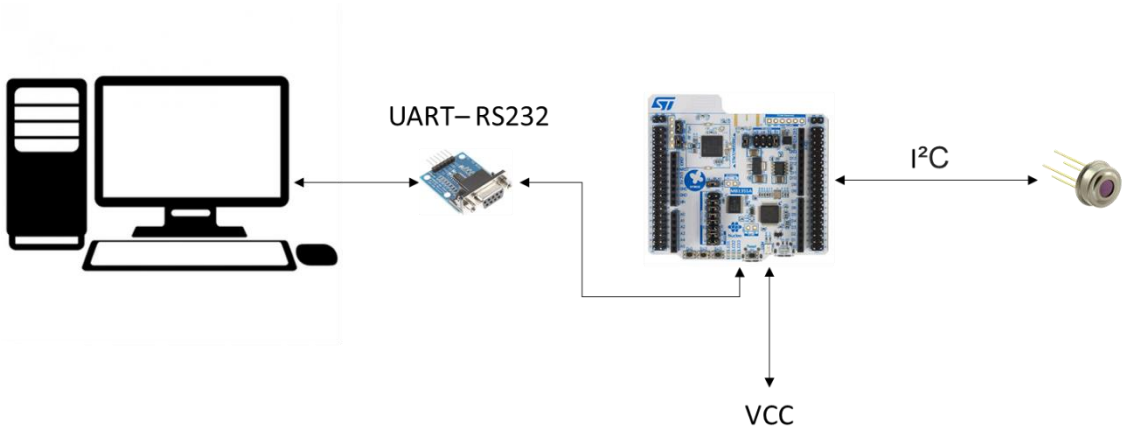


Figura 67. Comunicaciones versión STM32_UART_1SENSOR.elf.

Posteriormente hemos hecho pruebas del sistema con la comunicación a través del USB, sin usar el conversor, gracias a que el núcleo las tiene implementadas. En la Figura 68 tenemos los resultados obtenidos, ya con las tramas definidas. Esta prueba corresponde al programa STM32_USB_2SENSORES.

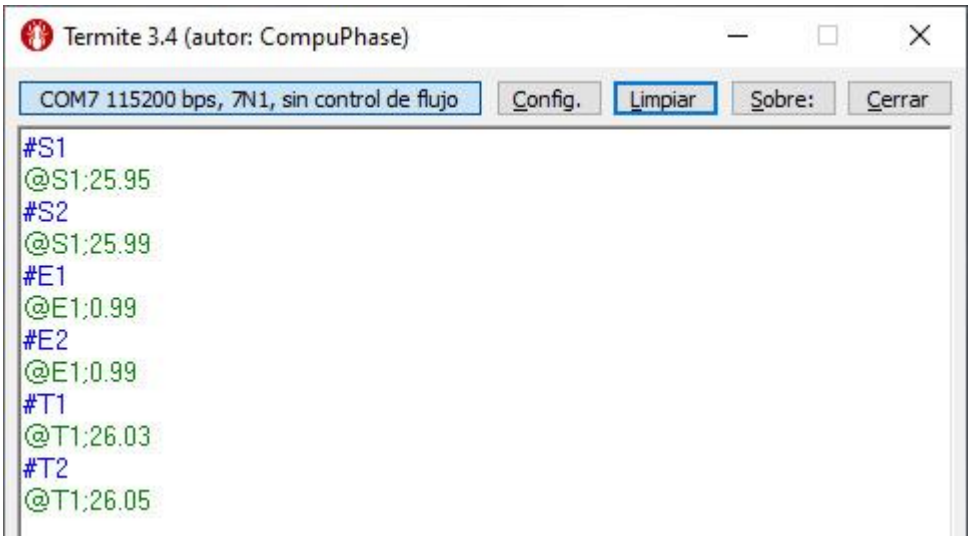


Figura 68. Prueba lectura del sensor por USB en STM32WB55.

Estas comunicaciones se ilustran en la Figura 69.

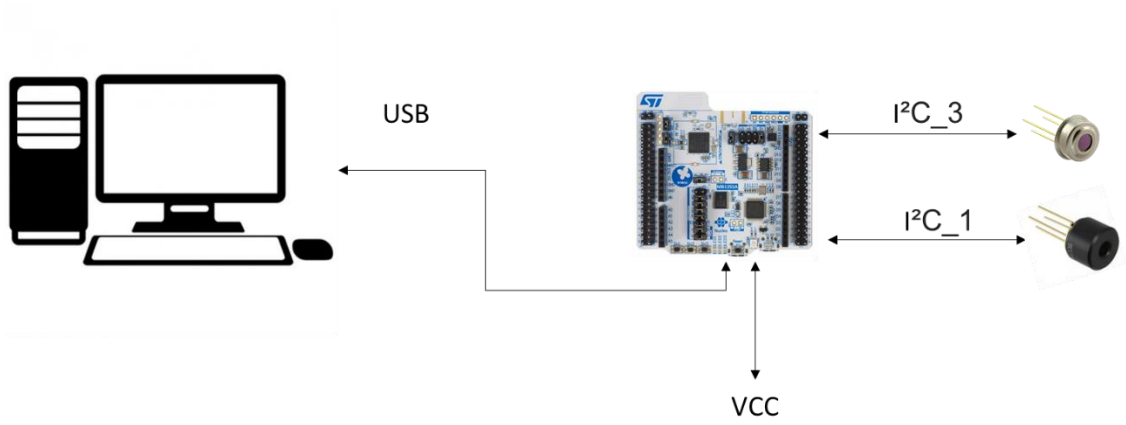


Figura 69. Comunicaciones versión STM32_USB_2SENSORES.elf.

8.3 Pruebas del sistema electrónico sobre ESP32

De igual manera, hemos probado el sensor MLX90614 con un programa ejemplo incluido con el paquete de Adafruit sencillo realizado en Arduino, usando la librería descrita, y pasando por pantalla los diferentes datos que recibimos. Los resultados los podemos ver en la Figura 70.

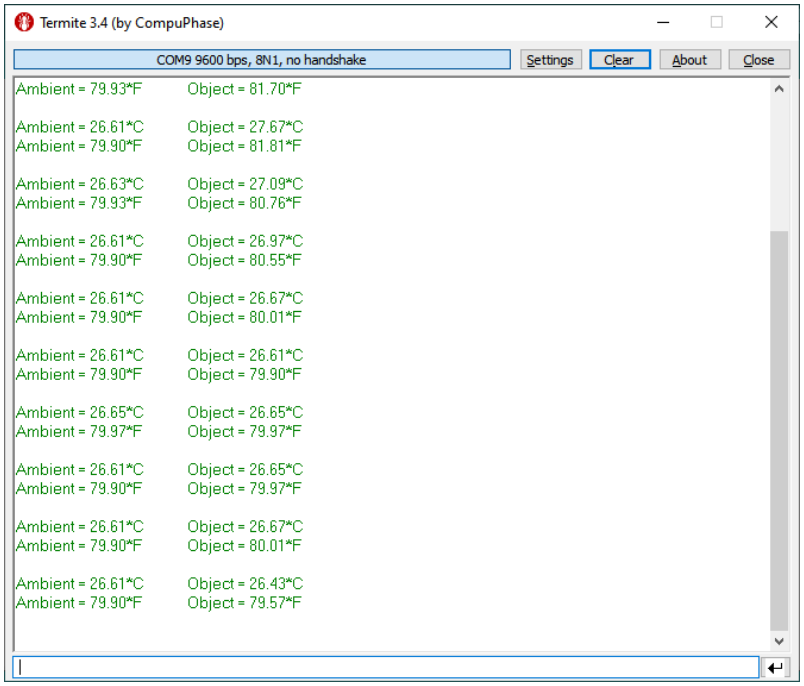


Figura 70. Prueba lectura sensor en ESP32.

También hemos hecho pruebas de temperatura solo con el *array* de sensores de SparkFun, usando otro ejemplo incluido, así estudiando la manera en la que recibimos los datos para poder construir la trama. Los primeros resultados obtenidos los vemos en la Figura 71.

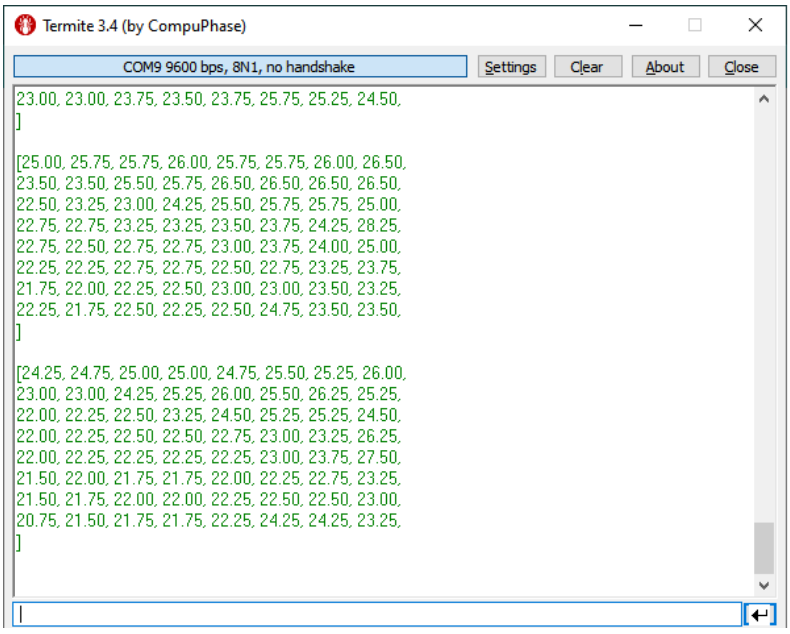


Figura 71. Prueba lectura array de sensores en ESP32.

Ya teniendo en funcionamiento los sensores y la comunicación de una placa con el PC procedemos a probar el BLE, obteniendo los resultados de la Figura 72. En ella, vemos a la izquierda el cliente, con los mensajes de depurado, está enviando la trama al servidor, en el que también vemos la trama recibida. En la práctica, el lado del cliente no lo veremos por el monitor serial, pues irá alimentado en el sistema pertinente. Esta captura corresponde a, en la derecha el programa ESP32_SERVIDOR, y en la izquierda, el programa ESP32_CLIENTE.

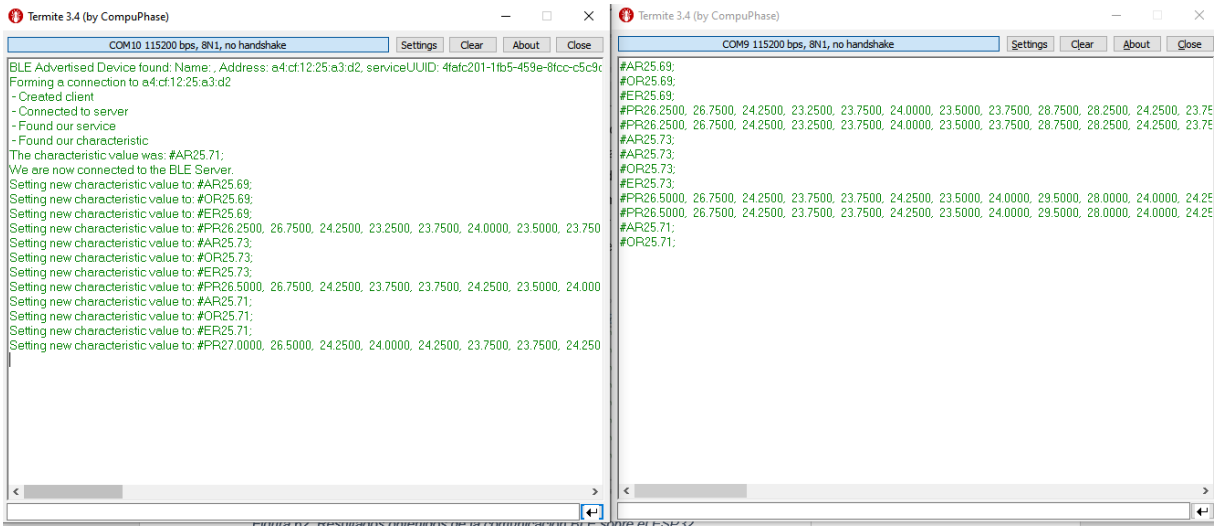


Figura 72. Pruebas BLE sobre ESP32.

En este caso, el esquema de las comunicaciones de los programas servidor y cliente los tenemos en la Figura 73.

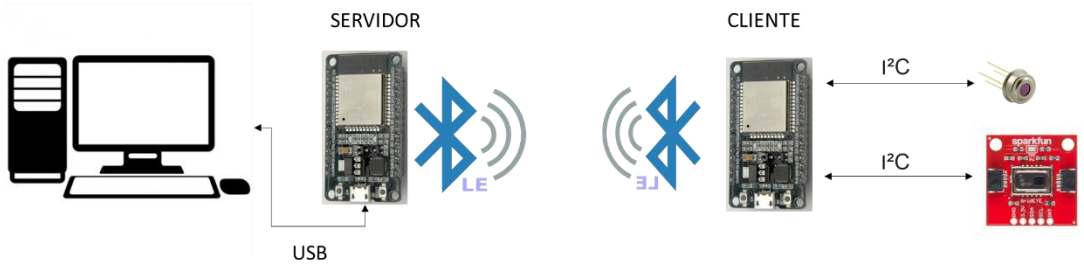


Figura 73. Comunicaciones ESP32_SERVIDOR y ESP32_CLIENTE.

Adicionalmente, hemos configurado tanto el servidor como el cliente para que trabajen por USB, sin necesitar del BLE. Esta comunicación se ilustra en la

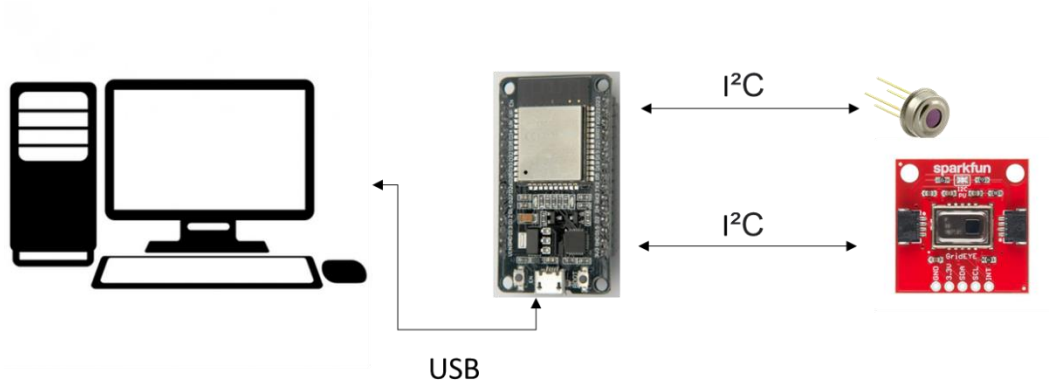


Figura 74. Comunicaciones USB usando programas de ESP32.

8.4 Pruebas sobre el sistema real

Para este apartado hemos hecho dos pruebas reales: por un lado, hemos realizado un perfil de temperatura. Por otro lado, hemos realizado una imagen térmica. Ambas son pruebas que se podrían realizar en un taller de reparaciones de PCB.

Para el perfil de temperatura, hemos simulado un proceso de soldadura, obteniendo la gráfica de la Figura 75. En rojo, hemos representado aproximadamente cómo sería la curva teórica, mientras que en azul representamos la curva real. Vemos que hay tres tramos: la curva ascendente, que aumenta aproximadamente un grado por segundo. Posteriormente la curva se mantiene unos segundos para comenzar a descender en el mismo orden hasta los 100ºC, en el que se estabiliza.

En nuestro proceso de soldadura, la muestra comenzó con una temperatura de alrededor de 70ºC. En este caso hemos usado el sensor MLX90614-AAA, pues el array de sensores no tolera estos rangos de temperatura. En la Figura 76 ilustramos el sistema que hemos usado para esta prueba.

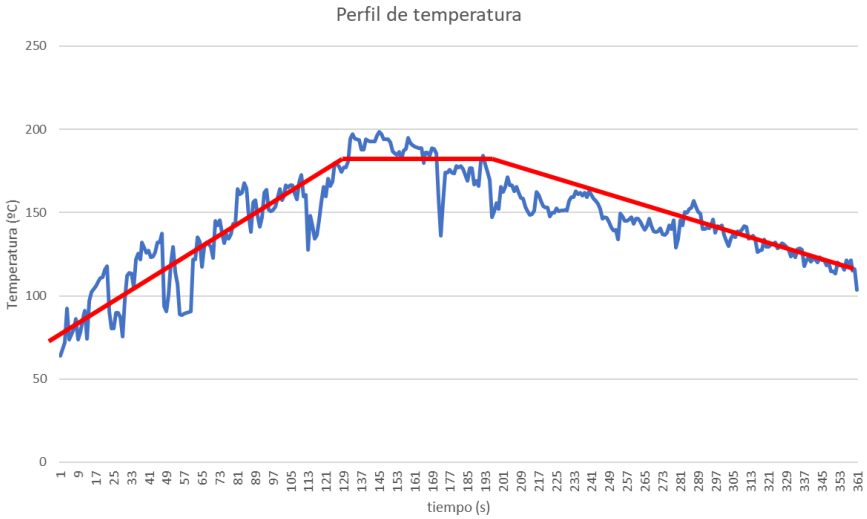


Figura 75. Perfil de temperatura para proceso de soldadura.

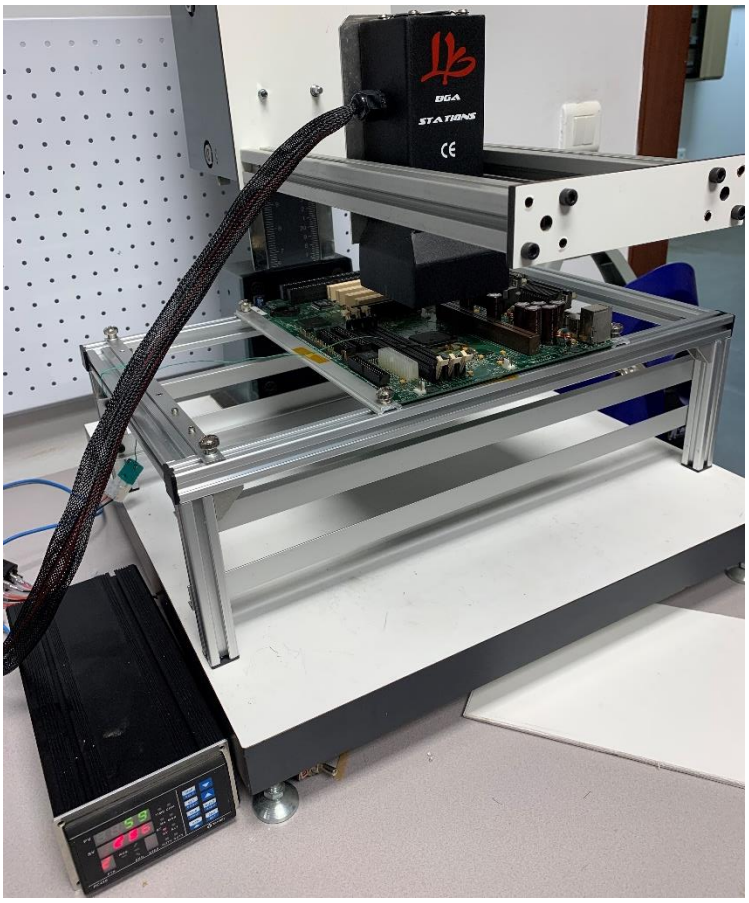


Figura 76. Proceso de soldadura.

Para la prueba de la imagen térmica, en el que detectamos los puntos calientes de un proceso de reparación, hemos dispuesto de una fuente de alimentación conecta a una carga resistiva, como se ve en la Figura 77.

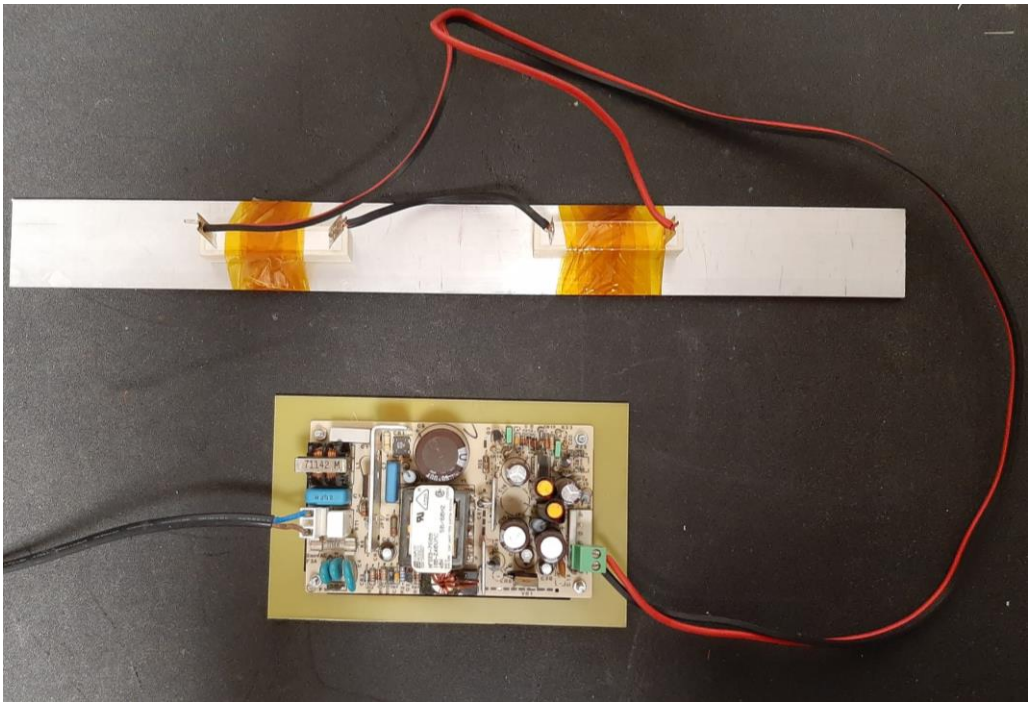


Figura 77. Simulación de proceso de reparación.

Primero, hemos obtenido una primera medida de la fuente de alimentación sin conectar, para obtener su mapa térmico, como vemos en la Figura 78, para tener una referencia.

21,0	20,8	20,8	20,5	20,5	20,8	20,8	20,5
21,0	20,8	20,3	20,5	20,5	20,5	21,0	20,8
21,0	21,0	20,5	20,8	20,8	21,0	20,8	21,3
21,3	20,3	20,5	20,5	20,5	20,5	21,0	21,0
21,3	20,3	20,8	20,5	20,5	20,5	20,0	20,8
20,8	20,8	20,5	20,3	20,3	21,0	21,3	21,3
21,0	20,8	20,5	20,5	20,5	20,8	20,8	21,0
21,0	21,3	21,0	21,0	20,5	21,3	20,8	21,0

Figura 78. Mapa térmico fuente de alimentación en reposo.

Posteriormente, la hemos alimentado y hemos esperado un tiempo para tomar la segunda medida. Cuando se ha calentado, hemos obtenido el mapa térmico de la Figura 79.

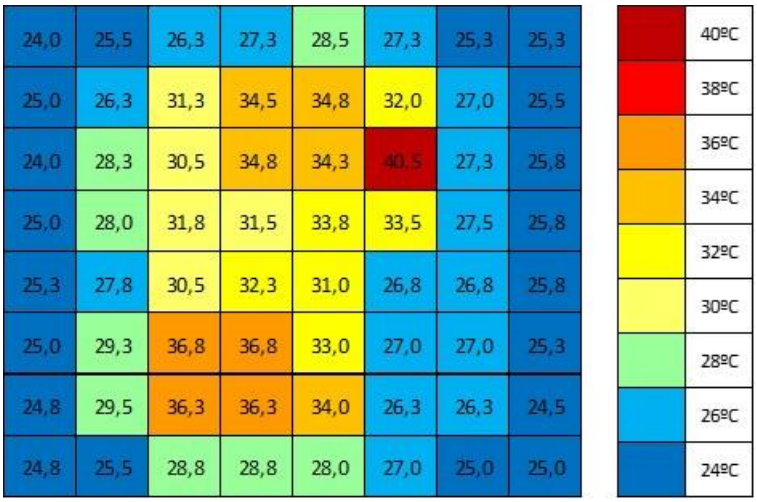


Figura 79. Mapa térmico fuente de alimentación conectada.

En esta prueba hemos utilizado la trama obtenida del array de sensores.

8.5 Conclusiones

En este capítulo hemos revisado las diferentes etapas para la prueba de los sensores. Inicialmente hemos probado con un ejemplo sencillo sobre la plataforma STM32WB55, usando un RS-232. También hemos probado la transmisión a través del USB.

A continuación, hemos probado sobre la plataforma ESP32 los dos tipos de sensores. Finalmente, hemos integrado los sensores con las comunicaciones BLE.

Capítulo 9. Conclusiones y trabajos futuros

En el presente documento hemos descrito diversas tecnologías de monitorización de temperatura, y las plataformas de desarrollo donde podíamos integrarlas. Hemos expuesto las herramientas de trabajo para las dos plataformas (STM32WB55 y ESP32), y trabajado con ellas para la elección de la placa que más se ajustara a nuestras necesidades. Hemos detallado las diferentes librerías para usar los sensores, y las diferentes comunicaciones a aplicar en el marco del proyecto. Finalmente hemos realizado pruebas y explicado los resultados obtenidos, tanto en el sistema electrónico como en el sistema real.

9.1 Conclusiones del proyecto

Después del análisis de las dos plataformas, y trabajar sobre ellas, hemos decidido trabajar sobre la plataforma ESP32, debido a su tamaño y al coste de esta, además de la sencillez que se nos presentaba a la hora de trabajar en el proyecto, limitado por las 300 horas estipuladas en el proyecto de la asignatura Trabajo de Fin de Grado. A la hora de realizar un Shield para ella, también es más simple debido a su forma regular y, por lo tanto, menos costoso.

De la forma que hemos diseñado la solución, nos permite varias formas de trabajo. Por un lado, cumpliendo con el marco del proyecto, implementamos las comunicaciones BLE.

Por otro lado, el haber implementado las comunicaciones a través del cable, nos permite conectar ambas placas al ordenador, y poder obtener temperaturas desde dos fuentes diferentes. Esto podría ser utilizado para medir temperaturas desde posiciones diferentes sobre el mismo sistema electrónico que estemos testeando, sin tener que cambiar el programa que hayamos flasheado en la memoria. Las características expuestas nos permiten realizar un mapa de temperatura de una manera más rápida.

Analizando los resultados obtenidos, podemos decir que aplicar este módulo de medida de temperatura es una solución funcional, pues nos da las temperaturas con un error poco considerable para esta aplicación, y a bajo coste, por lo que podemos decir que los objetivos se han cumplido.

Anexado a este documento se encuentran los manuales de uso de los programas sobre los que hemos trabajado, que son los siguientes:

STM32_UART_1SENSOR. En este programa se implementan las comunicaciones con un solo sensor a través del monitor serial usando comunicaciones RS-232, gracias a las librerías.

STM32_USB_2SENSORES. En este programa se implementan las comunicaciones con uno o dos sensores de la gama MLX90614 a través del puerto serie usando comunicaciones USB.

ESP32_SERVIDOR. En este programa se incluyen las comunicaciones BLE, desde el lado del servidor, y con los sensores, tanto los de la gama MLX90614, como el *array* AMG8833, implementados.

ESP32_CLIENTE. En este programa contiene las comunicaciones BLE, desde el cliente, y con los sensores, tanto los de la gama MLX90614, como el *array* AMG8833, también implementados.

9.2 Trabajos futuros

Siguiendo con la línea del proyecto, en el futuro se podría desarrollar una PCB a medida, en lugar de hacer uso de un módulo comercial, al que le añadimos el Shield. Así, tendríamos una electrónica más compacta y de menor tamaño, y que incorpore adicionalmente la recarga de la batería.

Otra futura solución a partir de este trabajo podría ser el diseño de cajas con impresión 3D a medida de la electrónica, facilitando así su instalación en las máquinas correspondientes.

Con respecto a los dispositivos utilizados, podríamos buscar nuevos *array* de sensores de mayor resolución. Esto encarecería el proyecto, pero obtendríamos imágenes de mayor resolución. También podríamos incluir algún sensor matricial capaz de trabajar a mayores temperaturas, y así monitorizar mejor los procesos de soldadura. Un ejemplo podría ser el TinkerForge 278 Bricklet Thermal Imager. Con un precio bastante más elevado, tiene una mayor

resolución de píxeles (80x60), y además soporta temperaturas de hasta 450ºC. Además, nos permite definir un punto de medida que obtenga la temperatura mínima, máxima y promedio de un área definida [40]. Podemos ver este dispositivo en la Figura 80.

Figura 80. Cámara TinkerForge 278.

Referencias

[1] J. Xu, J. Li, and Y. Jiang, “Components Locating in PCB Fault Diagnosis Based on Infrared Thermal Imaging,” in *2009 Second International Conference on Information and Computing Science*, 2009, vol. 2, pp. 7–9, doi: 10.1109/ICIC.2009.109.

[2] A. A. Sarawade and N. N. Charniya, “Detection of Faulty Integrated Circuits in PCB with Thermal Image Processing,” Jan. 2019, doi: 10.1109/ICNTE44896.2019.8946061.

[3] S. Y. Huang, C. W. Mao, and K. S. Cheng, “A VQ-based approach to thermal image analysis for printed circuit boards diagnosis,” *IEEE Trans. Instrum. Meas.*, vol. 54, no. 6, pp. 2381–2388, Dec. 2005, doi: 10.1109/TIM.2005.858546.

[4] F. f. Song, X. He, P. Lai, and R. wang, “The Study of infrared radiation thermal imaging technology for temperature testing,” in *2012 13th International Conference on Electronic Packaging Technology & High Density Packaging*, 2012, pp. 1336–1339, doi: 10.1109/ICEPT-HDP.2012.6474853.

[5] H. Moldovan, M. Marcu, and M. Vladutiu, “PCB Testing Using Infrared Thermal Signatures,” in *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*, 2005, vol. 3, pp. 1970–1974, doi: 10.1109/IMTC.2005.1604516.

[6] F. A. Al-Obaidy, Furat; Yazdani, Farhang; Mohammandi, “Fault detection using thermal image based on soft computing methods: Comparative study,” *Microelectronics Reliability*, pp. 56–64, 2017.

[7] Z. Dong and L. Chen, “Image registration in PCB Fault Detection based on infrared thermal imaging,” in *Chinese Control Conference, CCC*, Jul. 2019, vol. 2019-July, pp. 4819–4823, doi: 10.23919/ChiCC.2019.8866191.

[8] Todoelectronica, “Cámaras térmicas fijas.” https://www.todoelectronica.com/es/27608-camaras-termicas-fijas (accessed Mar. 06, 2020).

[9] Todoelectronica, “Cámara Termográfica ETS320 FLIR.” https://www.todoelectronica.com/es/camara-termografica-flir-ets320-de-montaje-fijo-para-comprobar-componentes-p-114147.html (accessed Feb. 28, 2020).

[10] N. Semiconductors, “I2C-bus specification and user manual,” 2014, doi: 10.1007/s10792-016-0274-8.

[11] I. System Management Interface Forum, “System Management Bus (SMBus) Specification,” no. March, pp. 1–59, 2018.

[12] K. Townsend, “Introduction to Bluetooth Low Energy,” 2014. .

[13] Punch Through, “How GAP and GATT Work,” 2013. https://punchthrough.com/how-gap-and-gatt-work/ (accessed Mar. 25, 2020).

[14] R. Hatton, “Understand Electronics,” *Electron. Educ.*, vol. 1996, no. 2, pp. 34–34, 1996, doi: 10.1049/ee.1996.0051.

[15] Melexis, *MLX90614 family Single and Dual Zone Datasheet*. 2019.

[16] Cetronic, “Adafruit AMG8833 8x8 Thermal Camera Sensor.” Accessed: Feb. 28, 2020. [Online]. Available: https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor.

[17] SparkFun, “AMG8833, SparkFun Grid-EYE Infrared Array Breakout - (Qwiic).” https://media.digikey.com/pdf/Data Sheets/Sparkfun PDFs/SEN-14607_Web.pdf (accessed Nov. 18, 2020).

[18] SparkFun, “Qwiic GRID-Eye Infrared Array (AMG88xx) Hookup Guide.”

[19] SmartPrototyping, “What is Qwiic?” https://www.smart-prototyping.com/Qwiic.html (accessed Nov. 18, 2020).

[20] “AMG8833 KKMOON,” [Online]. Available: https://es.aliexpress.com/item/4000606897173.html?spm=a2g0s.9042311.0.0.274263c0VDhPF8.

[21] STElectronics, “P-NUCLEO-WB55 - Bluetooth™ 5 and 802.15.4 Nucleo Pack including USB dongle and Nucleo-68 with STM32WB55 MCUs,

supports Arduino™ Uno V3 and ST morpho connectivity - STMicroelectronics.” https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/p-nucleo-wb55.html (accessed Feb. 28, 2020).

[22] STElectronics, “P-NUCLEO User Manual,” vol. 3304, no. January, pp. 1–148, 2012.

[23] STElectronics, “STM32WB - Bluetooth, Wireless Microcontrollers (MCU) - STMicroelectronics.” https://www.st.com/en/microcontrollers-microprocessors/stm32wb-series.html (accessed Feb. 28, 2020).

[24] STElectronics, “STM32WB55xx,” no. February, 2020.

[25] E. Systems, “ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems.” https://www.espressif.com/en/products/socs/esp32 (accessed Nov. 15, 2020).

[26] E. Systems, “ESP32 Series Datasheet,” 2020. Accessed: Nov. 15, 2020. [Online]. Available: https://www.espressif.com/en/support/download/documents.

[27] E. Systems, “ESP32-WROOM-32D and ESP32-WROOM-32U modules,” 2019. Accessed: Nov. 15, 2020. [Online]. Available: www.espressif.com/en/subscribe.

[28] StudioPieters, “ESP32 – PinOut.” https://www.studiopieters.nl/esp32-pinout/ (accessed Nov. 15, 2020).

[29] Warex, “Esp32.” https://warex.com.co/esp32/ (accessed Nov. 15, 2020).

[30] STElectronics, “STM32Cube - Discover the STM32Cube Ecosystem - STMicroelectronics.” https://www.st.com/content/st_com/en/stm32cube-ecosystem.html (accessed Feb. 28, 2020).

[31] STElectronics, “STM32CubeMX - STM32Cube initialization code generator - STMicroelectronics.” https://www.st.com/en/development-tools/stm32cubemx.html (accessed Feb. 28, 2020).

[32] The Eclipse Foundation, “About the Eclipse Foundation.” https://www.eclipse.org/org/ (accessed Nov. 18, 2020).

[33] STElectronics, “Product status link STM32CubeIDE Integrated development environment for STM32 products.” Accessed: Feb. 28, 2020. [Online]. Available: www.st.com.

[34] STElectronics, “STM32CubeProg - STM32CubeProgrammer software for all STM32 - STMicroelectronics.” https://www.st.com/en/development-tools/stm32cubeprog.html (accessed Feb. 28, 2020).

[35] Arduino, “What is Arduino?” https://www.arduino.cc/en/Guide/Introduction (accessed Nov. 15, 2020).

[36] S. Riege, “Exploring Altium Designer | Altium Designer 20.0 User Manual | Documentation,” 2020. https://www.altium.com/documentation/altium-designer/exploring-altium-designer#:~:text=Welcome to Altium Designer%2C which,collaborative PCB design environment available. (accessed Feb. 28, 2020).

[37] S. Riege, “From Idea to Manufacture - Driving a PCB Design through Altium Designer | Altium Designer 20.0 User Manual | Documentation,” 2020. https://www.altium.com/documentation/altium-designer/from-idea-to-manufacture-driving-a-pcb-design-through-altium-designer (accessed Feb. 28, 2020).

[38] Melexis, “MLX90614ESF footprint & symbol by Melexis Technologies.” https://www.snapeda.com/parts/MLX90614ESF-AAA-000-TU/Melexis Technologies NV/view-part/ (accessed Feb. 28, 2020).

[39] L. L. & E. AG, *Manual CircuitCAM 5.0*. Garbsen, 2004.

[40] “TinkerForge 278 Bricklet Thermal imager TinkerForge .” https://www.conrad.com/p/tinkerforge-278-bricklet-thermal-imager-tinkerforge-2255032?WT.srch=1&vat=true (accessed Nov. 18, 2020).

Presupuesto

Para el desarrollo de este trabajo hemos precisado el uso de diferentes recursos, humanos y materiales, desde la documentación previa necesaria, pasando por el diseño de la PCB y finalizando en el desarrollo e implementación del software, incluyendo las pruebas del sistema. Vamos a recoger todos los costes asociados en los siguientes apartados.

1. Recursos de fabricación

Hemos contado con la colaboración del Laboratorio de Fabricación de Prototipos y Sistemas Electrónicos del IUMA, que cuentan con los equipos de fabricación y test de circuitos electrónicos necesarios para desarrollar las diferentes tareas del prototipado. En la Tabla XI recogemos los diferentes costos asociados a las máquinas.

Tabla XI. Costos de recursos de fabricación.

Recurso	Coste total	Coste por hora
Máquina fresadora	20.000,00€	25,00€
Estación de soldadura	1.800,00€	5,00€

Finalmente, estimaremos un costo para este apartado de 100,00€.

2. Recursos Hardware

Para el desarrollo de este proyecto hemos precisado diferentes recursos hardware, los cuales tienen un costo asociado. En la Tabla XII los listaremos.

Tabla XII. Costos de recursos hardware.

Recurso	Coste estimado
P-NUCLEO-WB55 pack	40,19€
ESP32-WROOM-32D	6,38€
Ordenador personal	800,00€
Cables y equipo auxiliar	20,00€
TOTAL	866,57€

3. Recursos Software

Igualmente hemos precisado licencias de recursos software, las cuales han sido utilizadas para diversos proyectos y no únicamente para este, así pues, los costos derivados los mostramos en la Tabla XIII.

Tabla XIII. Coste de recursos Software.

Recurso	Tipo de licencia	Coste de la licencia	Mantenimiento anual
Altium Designer Release 10	Universitaria	100,00€	-
CircuitCAM 5.0	Empresa	-	-
STM32CubeMX	Pública	-	-
STM32CubeIDE	Pública	-	-
STM32CubeProg	Pública	-	-
Arduino IDE	Pública	-	-
		Total	100,00€

4. Recursos Humanos

En la Tabla XIV reflejaremos el coste por hora en función del montante anual que supone tener contratado a un ingeniero de telecomunicaciones desempeñando tareas de investigador en proyectos[1].

Tabla XIV. Costos de recursos humanos.

Tarea	Coste/hora	Horas	Días	Coste total
T1. Estudio inicial sobre mediciones térmicas.	16,65€/hora	5	2	166,50€
T2. Elección de plataforma de trabajo.	16,65€/hora	5	2	166,50€
T3. Elección de los sensores.	16,65€/hora	5	2	166,50€
T4. Diseño de la PCB.	16,65€/hora	5	15	1.248,75€
T5. Fabricación y montaje.	16,65€/hora	5	10	832,50€
T6. Programación del firmware.	16,65€/hora	5	15	1.248,75€
T7. Pruebas del sistema electrónico.	16,65€/hora	5	4	333,00€
T8. Pruebas sobre el sistema real.	16,65€/hora	5	4	333,00€
T9. Documentación.	16,65€/hora	5	4	333,00€
			TOTAL	4.662,00€

5. Material fungible

En este apartado se asocian los gastos derivados de la fabricación, componentes electrónicos[2], sustratos o estaño. Los costes directos de los componentes se reflejan en la Tabla XV.

[1] Según la resolución del BOULPGC del 4 de noviembre de 2010.
[2] Precios obtenidos en la web de DigiKey.

Tabla XV. Costos material fungible.

Material	Cantidad	Costo unidad	Costo total
Sensor MLX90614-AAA	2	12,94€	25,88€
Sensor MLX90614-ACC	1	18,11€	18,11€
***Array* de sensores AMG8833**	1	24,98€	24,98€
Proceso de fabricación	1	150,00€	150,00€
		TOTAL	218,97€

6. Coste total del proyecto

Concluyendo, con la suma total de los costes ya mencionados obtenemos el coste total del proyecto, recogido en la tabla 11.

Recursos	Coste
1. Recursos de fabricación	100,00€
2. Recursos hardware	866,57€
3. Recursos software	100,00€
4. Recursos humanos	4.662,00€
5. Material fungible	218,97€
Subtotal	5.947,54€
IGIC (7%)	416,33€
Coste total del proyecto	6.363,87€

Dña. Irene Merino Fernández declara que el presupuesto del presente proyecto asciende a seis mil trescientos sesenta y tres euros con ochenta y siete céntimos (6.363,87€).

Las Palmas de Gran Canaria, a 26 de noviembre de 2020.

Fdo.: Irene Merino Fernández

ANEXOS

MANUAL DE USUARIO
ESP-32 WROOM32

Irene Merino Fernández

Noviembre 2020

Versión 1

ÍNDICE

ÍNDICE DE TABLAS

1. Introducción

Este documento recoge la información de uso para la placa ESP-32 WROOM32, versión de 30 pines de GPIO (General Purpose Input/Output) como parte de un módulo de test de sistemas electrónicos.

Para cada programa tenemos dos modos de uso: una placa conectada al ordenador, y dos placas que se comunican entre ellas. Una de las dos se conectará al PC, el servidor, y la otra se integrará en el sistema de medida, que podría ser una máquina CNC, o usarlo como sistema portable de medida.

Indicaremos los pines de uso para las medidas de temperatura, los programas funcionales existentes y una descripción de las tramas del sistema.

2. Pinout ESP32-WROOM-32D

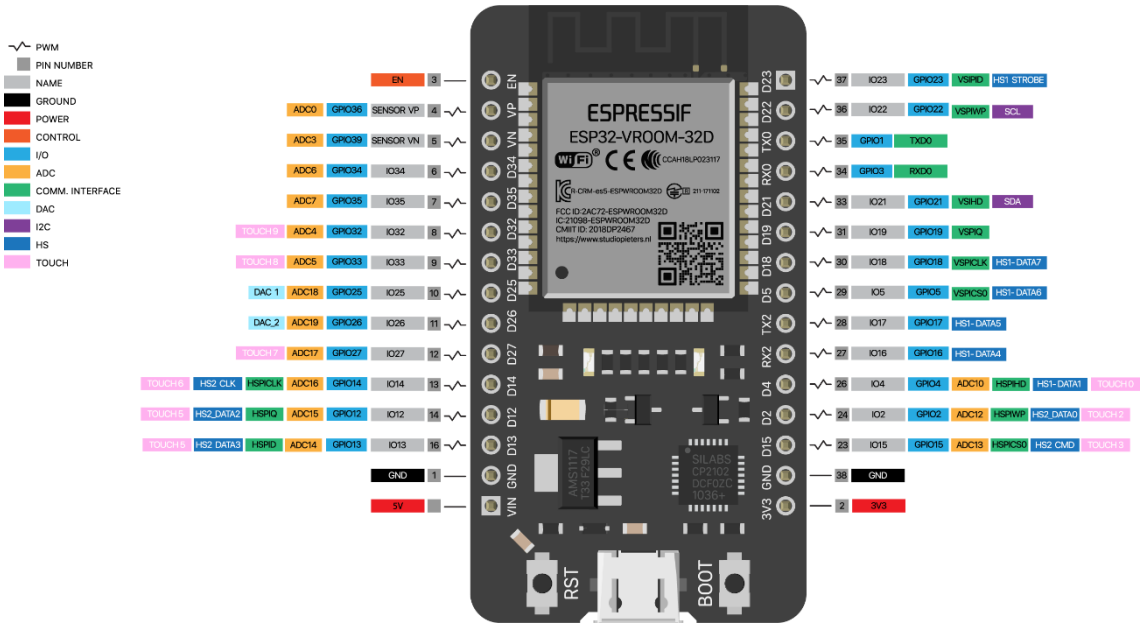


Figura 81. Pinout ESP32WROOM32D

En la Figura 81 mostramos el *pinout* de la plataforma con la que vamos a trabajar. En el siguiente apartado describiremos con más detalle la placa de expansión y las señales de las que vamos a hacer uso.

3. Conexionado

El conexionado necesario(1) para comunicarnos con la placa se recoge en la Tabla XVI.

Tabla XVI. Conexionado Shield.

CONECTOR 1 (P1)		
PIN 1	3V3	3.3 V
PIN 2	D22	SCL
PIN 3	D21	SDA
PIN 4	GND	GND
CONECTOR 2 (P2)		
PIN 1	3V3	3.3 V
PIN 2	D22	SCL
PIN 3	D21	SDA
PIN 4	GND	GND

(1) Este conexionado es necesario para la plataforma que vaya a realizar las medidas de temperatura. Si vamos a usar el sistema en modo servidor, no utilizaremos ninguna señal adicional.

Además, hemos sacado un LED de control, conectado a D13. Los pines reflejados en la Tabla XVI

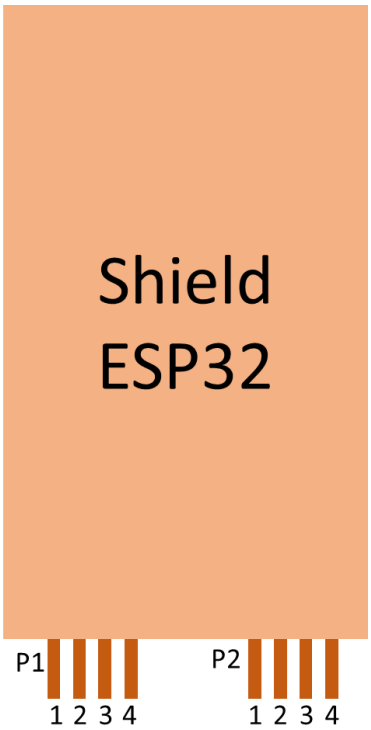


Figura 82. Shield ESP32.

4. Programas

Para cargar estos programas en la placa necesitamos el IDE de Arduino, disponible en la Web oficial, que adicionalmente nos instalará los drivers para esta placa. En las preferencias del entorno debemos añadir al gestor de tarjetas adicionales la siguiente URL:

https://dl.espressif.com/dl/package_esp32_index.json

Las tramas en ambos casos se reciben a través del puerto serie sin necesidad de conexionado adicional. Para su visualización la configuración del puerto serie es la siguiente:

- Tasa de baudios: 115200
- Bits de datos: 8
- Bits de stop: 1

Hay que tener en cuenta que para cargar programas en ESP32, mientras se está cargando en la placa debemos mantener pulsado el botón de BOOT.

En la Tabla XVII tenemos las tramas, que se aplican a todas las versiones. La comunicación se hace desde la placa hacia el ordenador, en ningún caso al revés.

Tabla XVII. Tramas de los programa.

Byte	Significado	Valor	Comentarios
1	Comienzo de trama	#	Inicia la transmisión
2	Dato del sensor	A	Temperatura ambiente MLX90614
		E	Emisividad MLX90614
		O	Temperatura objeto MLX90614
		P	Temperaturas AMG8833
3 - N-1	Dato	float	Será un único byte si estamos trabajando con los sensores, y en el caso del *array*, será una ráfaga de 64 bytes
N	Fin de trama	;	Finaliza la transmisión

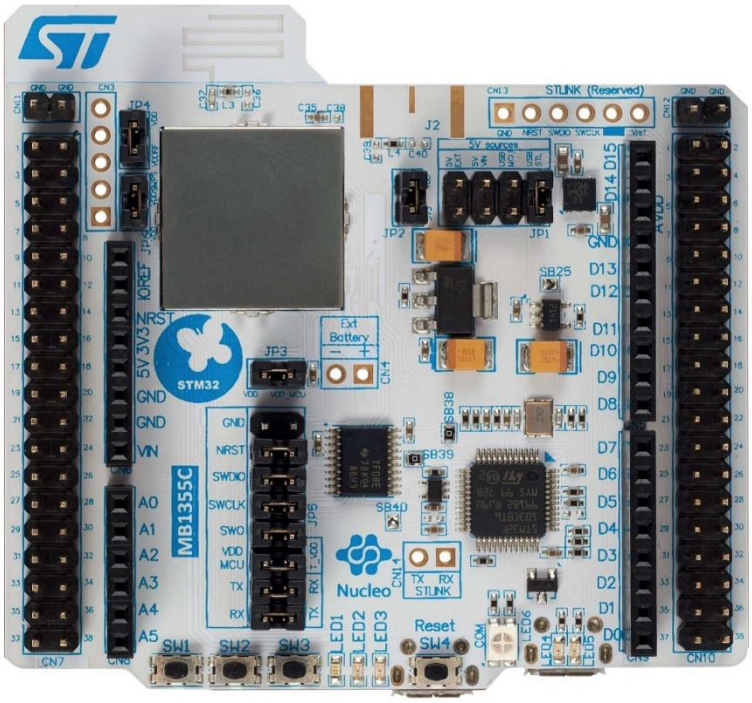
Las versiones existentes para la plataforma seleccionada (DOIT ESP32 DEVKIT 1) son las siguientes:

- **ESP32_CLIENTE.ino**

En esta versión implementamos el cliente para la comunicación BLE. También incluye comunicaciones de una única placa. Para ambos casos precisamos de la placa de adaptación.

- **ESP32_SERVIDOR.ino**

En esta versión se implementa el servidor para la comunicación BLE y, adicionalmente, comunicaciones para el uso de una única placa. En este caso, solo precisamos de la placa de adaptación en el caso de usar una placa



MANUAL DE USUARIO

STM32WB55

Irene Merino Fernández

Noviembre 2020

Versión 1

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

En este documento vamos a recoger la información referente al uso del núcleo STM32WB55 como parte de un módulo de comunicaciones para el test de sistemas electrónicos.

Por un lado, daremos el conexionado necesario en la placa de expansión para el correcto funcionamiento de este.

Por otro lado, indicaremos las versiones funcionales existentes de los programas, y una breve descripción de estos, detallando las tramas que utilizaremos para la comunicación a través del puerto serie.

2. Pinout Núcleo STM32WB55

Figura 83. Pinout STM32WB55.

En la Figura 83 tenemos la placa con su correspondiente *pinout*. Estos pines son programables haciendo uso del programa STM32CubeMX.

3. Conexionado

El conexionado necesario(1) para comunicarnos con la placa se recoge en la Tabla I.

Tabla I. Conexionado Shield.

CONECTOR 1 (P1)		
PIN 1	PB8	SDA 3
PIN 2	PB9	SCL 3
PIN 3	Común	GND
PIN 4	CN7-16 y CN7-12	3.3 V
CONECTOR 2 (P2)		
PIN 1	PC0	SCL 1
PIN 2	PC1	SDA 1
PIN 3	CN7-16 y CN7-12	3.3 V
PIN 4	Común	GND
CONECTOR 3(2) (P3)		
PIN 1	Común	GND
PIN 2	CN7-16 y CN7-12	3.3 V
PIN 3	PC4	SPI 1
PIN4	PA5	SPI 2
PIN 5	PA6	SPI3
PIN 6	PB12	SPI 4
PIN 7	PA9	SPI 5
PIN 8	PB14	SPI 6
PIN 9	PA15	SPI 7
PIN 10	PA8	SPI 8

(1) Para establecer las conexiones usando la USART, se requiere conexión con los pines de transmisión/recepción que incluye la placa y un dispositivo RS-232, en este caso corresponden a los pines 6 y 34 del conector CN10, señales PB6 y PB7. (Ver Figura 83).

(2) Ninguno de los programas funcionales utiliza estos pines. Son pines genéricos que quedan abiertos para futuras aplicaciones.

También se han sacado 5 señales de GPIO que se conectan a los LED del Shield. Se encuentran en las señales PA4, PC3, PC2, PB2 y PE4.

En la Figura 84 se encuentran estos pines representados.

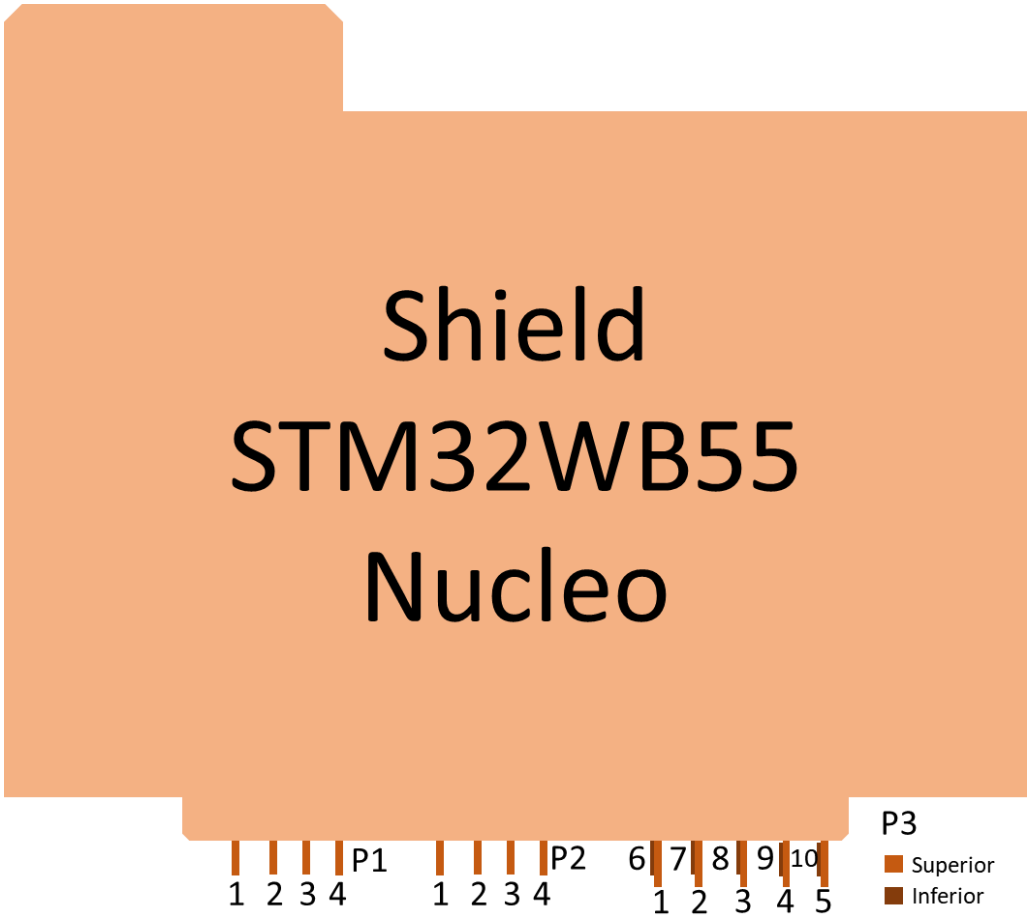


Figura 84. Shield núcleo STM32WB55

4. Programas

Para cargar estos programas en la placa necesitamos la herramienta STM32Programmer, disponible en la Web de STM. Para programarlo, nos tenemos que conectar haciendo uso de la conexión ST-LINK, tanto en el programa como en la placa, teniendo en cuenta los jumpers de alimentación.

En todas las versiones nos vamos a conectar haciendo uso del puerto serie, para el cual la configuración es la siguiente:

- Tasa de baudios: 115200
- Bits de datos: 7
- Bits de stop: 1

Las versiones existentes para esta plataforma son las siguientes:

- **STM_UART_1SENSOR.elf**

En esta versión implementamos la comunicación con un sensor MLX90614. Las tramas se reciben de manera continuada a través de la USART, la comunicación se realiza desde la placa hacia el ordenador.

La forma de las tramas es:

  - ✓ Emmisivity: datos (formateados a 3 decimales).
  - ✓ Ambient: datos (formateados a 2 decimales).
  - ✓ Object: datos (formateados a 2 decimales).

- **STM_USB_2SENSORES.elf**

En esta versión implementamos la comunicación con dos sensores MLX90614. En este caso la comunicación se hace a través del USB, no precisamos de conexionado adicional.

Las tramas en este caso las recibimos cuando se pregunta a través del puerto serie. La forma en la que nos comunicamos a través del puerto serie se recoge en la Tabla II.

Tabla II. Tramas

Byte	Valor	Significado
Byte 1	@	Comunicación placa - PC
	#	Comunicación PC - placa
Byte 2	E	Emisividad
	A	Temperatura ambiente del sensor
	O	Temperatura objeto del sensor
Byte 3	1	Sensor de temperatura I2C1
	2	Sensor de temperatura I2C3
Byte 4	float	Dato de temperatura
Byte 5	;	Fin de trama (solo en comunicación placa - PC)