UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

Diseño e implementación de Kit de laboratorio a partir de STM32 con arquitectura ARM

Memoria

Autor: Ulric Rossell Currius

Director: Sebastián Tornil Sin

Convocatoria: Septiembre 2021

Resumen

Este documento pretende razonar y justificar la actualización del kit de laboratorio de informática industrial utilizado en el campus de la Escuela de Ingeniería de Barcelona Este (EEBE) de la Universidad Politécnica de Catalunya (UPC). Para ello se propone cambiar el procesador utilizado hasta ahora, el Intel 8051 de 8 bits, por el STM32 de 32 bits y arquitectura ARM, la más utilizada actualmente por fabricantes de dispositivos móviles.

Inicialmente, se hace una breve descripción del kit de laboratorio actual y se hace un estudio de la arquitectura ARM y del microprocesador elegido, el STM32F401RE. Para simplificar el proyecto y adaptarlo a la extensión de un TFG realizado por un solo alumno, se utiliza una placa de desarrollo, NUCLEO-F401RE, la cual incluye el microcontrolador y diferentes componentes, conectores y características que se detallan en el documento.

Una vez definido el componente clave del nuevo kit, el microcontrolador, se hace una selección de periféricos para realizar una PCB, en forma de shield, para la placa de desarrollo creando así un nuevo kit de laboratorio actualizado. Los componentes escogidos pretenden poder dar una continuidad al kit anterior y a las prácticas de laboratorio y metodología utilizadas hasta ahora para poder dar lugar a una transición lo más suave posible.

En este documento también se detalla y justifica el software utilizado para las diversas fases del proyecto y se hace un repaso sobre la programación dando códigos de ejmplo. También se definen las fases del prototipado físico y se muestra el resultado.

Al final del documento se hace un estudio económico del precio asociado al prototipo ya realizado y del coste de una posible producción en serie para finalizar dando las conclusiones del proyecto, justificando la necesidad de esta actualización.

Palabras clave: TFG, kit laboratorio, shield, ARM, STM32, NUCLEO, electrónica, PCB.

Resum

Aquest document pretén raonar i justificar la actualització del kit de laboratori de la informàtica industrial que s'utilitza en el campus de l'Escola d'Enginyeria de Barcelona Est (EEBE) de la Universitat Politècnica de Catalunya (UPC). Per a aquesta finalitat es proposa canviar el processador que s'utilitza actualment, l'Intel 8051 de 8 bits, per l'STM32 de 32 bits i arquitectura ARM, la més utilitzada actualment per fabricants de dispositius mòbils.

Inicialment, es fa una breu descripció del kit de laboratori actual i es fa un estudi de l'arquitectura ARM i del microprocessador escollit, l'STM32F401RE. Per a simplificar el projecte i adaptar-lo a la extensió d'un TFG realitzat per un sol alumne, s'utilitza una placa de desenvolupament, la NUCLEO-F401RE, que inclou el microcontrolador i diferents components, connectors i característiques que es detallen en aquest document.

Un cop definit el component clau del kit, el microcontrolador, es fa la selecció de perifèrics que formaran part de la PCB, en forma de shield, per a la placa de desenvolupament, creant així un kit de laboratori actualitzat. Els components escollits pretenen poder donar continuïtat al kit anterior i a les pràctiques de laboratori i metodologia utilitzades fins ara per a tenir una transició d'un a l'altre el més suau possible.

En aquest document també es detalla i justifica el software utilitzat per a les diverses fases del projecte i es fa un repàs a la programació, proporcionant codis d'exemple. També es defineixen les fases del prototipat físic i es mostra el resultat final obtingut.

Al final del document es fa un estudi econòmic del preu associat al prototip ja realitzat i del cost de una possible producció en sèrie per finalitzar donant les conclusions del projecte, justificant la necessitat d'aquesta actualització.

Paraules clau: TFG, kit laboratori, shield, ARM, STM32, NUCLEO, electrònica, PCB.

Abstract

This document aims to reason and justify the update of the industrial informatics laboratory kit used on the campus of the Barcelona East School of Engineering (EEBE) of the Polytechnic University of Catalonia (UPC). To do this, it is proposed to change the processor nowadays used, the 8-bit Intel 8051, for the 32-bit STM32 with ARM architecture, the currently most used architecture by mobile device manufacturers.

At the beginning, a brief description of the current laboratory kit is made. This document includes a study of the ARM architecture and the chosen microprocessor, the STM32F401RE. To simplify the project and adapt it to the extension of a TFG carried out by a single student, a development board is used, NUCLEO-F401RE, which includes the microcontroller and different components, connectors and characteristics that are detailed in the document.

Once the core of the new kit, the microcontroller, has been defined, a selection of peripherals is made to build a PCB, in the form of a shield, for the development board, creating a new updated laboratory kit. With the components selected is pretended to give continuity to the previous kit and to the hands-on laboratory sessions and methodology used until now in order have a transition as smooth as possible.

This document also details and justifies the software used during the phases of the project and shows briefly the programming showing coding examples. The phases of the physical prototyping are also defined and the obtained result is showed.

At the end of the document, an economic study of the prototype is made finalizing with the project end conclusions and justifying the need of this update.

Key words: Bachelor's Degree Thesis, Laboratory kit, ARM, STM32, NUCLEO, electronics, PCB.

Kurzfassung

Dieses Dokument soll begründen und beweisen die Aktualisierung des Versuchsraumkits für industrielle Informatik, das auf dem Campus der Barcelona Ost Schule für Ingenieur (EEBE) der Polytechnischen Universität aus Katalonien (UPC) verwendet ist. Dazu wird vorgeschlagen, den aktuell verwendeten Prozessor, 8 Bit Intel 8051, für den 32-Bit-STM32 mit ARM-Architektur zu ersetzen. ARM ist zurzeit die Architektur für Prozessoren, die Mobilgeräteherstellern am häufigsten verwenden.

Am Anfang erfolgt eine kurze Beschreibung des aktuellen Versuchsraumkits. Dieses Dokument enthält einen Bericht über die ARM-Architektur und den ausgewählten Mikroprozessor, den STM32F401RE. Um das Projekt zu vereinfachen und die Länge des Dokuments an die Länge von einem einzelnen Studenten durchgeführte Bachelorarbeit anzupassen, wird ein Entwicklungsboard verwendet, NUCLEO-F401RE, das den Mikrocontroller und verschiedene Komponenten, Anschlüsse und Eigenschaften enthält, dies wird im Dokument beschrieben.

Nachdem das wichtigste Element des neuen Kits, der Mikrocontroller, definiert wurde, wird eine Auswahl an Peripheriegeräten getroffen. Die Leiterplatte mit den Peripheriegeräten wird in Form eines Shields des Entwicklungsboard gebaut, wodurch ein aktualisiertes Versuchsraumkits entstehen wird. Mit den ausgewählten Komponenten soll dem vorherigen Kit und des bisher verwendeten Laborversuches und Methodik, um einen möglichst reibungslosen Übergang zu ermöglichen, Kontinuität verliehen werden.

Dieses Dokument beschreibt und begründet auch die während der Projektphasen verwendete Software und zeigt kurz die Programmierung mit Beispielen. Die Phasen die das Bauen den physischen Prototyp erfolgen, werden ebenfalls definiert und das erzielte Ergebnis wird gezeigt.

Am Ende des Dokuments wird eine wirtschaftliche Studie des Prototyps erstellt, und wird mit den Schlussfolgerungen des Projekts abgeschlossen und damit der Notwendigkeit diese Aktualisierung begründet.

Stichworte: Bachelorarbeit, Versuchsraum, Labor, Kit, ARM, STM32, NUCLEO, Elektronik, Leiterplatte.

Agradecimientos

Quiero agradecer a todos los profesores que he tenido tanto en la universidad como fuera de ella, por haber hecho crecer mis inquietudes y conocimientos y haberme hecho crecer tanto personal como profesionalmente realizando un gran trabajo que nunca se podrá valorar lo suficiente.

Para la realización de este Trabajo de final de grado de ingeniería electrónica industrial y automática he recibido mucho apoyo y ayuda de diferentes instituciones y personas que me gustaría reflejar por escrito.

En primer lugar, me gustaría agradecer a la universidad EEBE que durante estos cuatro años me ha acogido; en especial me gustaría agradecerle a mi tutor, Sebastián Tornil, haberme ayudado durante este largo trayecto mostrándose siempre accesible y siendo resolutivo. También mencionar a Fernando, del laboratorio de electrónica, que me ayudó a realizar la PCB.

En segundo lugar, me gustaría agradecer a Xicoy Electrònica S.L. y Gaspar Espiell por la confianza mostrada al ofrecerme mi primer trabajo como ingeniero antes si quiera de haber terminado el grado, poniéndome el equipo y material de la empresa siempre a mi disposición y dándome la flexibilidad horaria que necesitaba. Me gustaría también agradecer a todos mis compañeros que me hicieron sentir del primer al último día como en casa, con mención especial para Jaume Anguera, mi compañero de laboratorio electrónico, que estuvo siempre disponible para cualquier consulta y con el que compartí más momentos, entablando una gran relación profesional.

En tercer lugar, me gustaría agradecer a mi actual empresa JBC Soldering S.L. por confiar en mí en todo momento y prestarme su equipo e instalaciones para realizar el proyecto. Mención especial a mi responsable, Karl Schmid, por el apoyo, y al resto de compañeros del laboratorio por su ayuda y sus consejos.

Por último, no quisiera olvidarme de agradecer a mi familia y amigos que me han apoyado y han confiado en mi en todo momento, siendo mi principal motivación para seguir adelante.

Glosario

Término	Definición
µC	Abreviatura utilizada para referirse a un microcontrolador
µP	Abreviatura utilizada para referirse a un microprocesador
CPU	Siglas en inglés de unidad central de procesamiento (Central Processing Unit)
ARM	Siglas en inglés de máquinas RISC avanzadas (Advanced RISC Machines)
RISC	Siglas en inglés de set de instrucciones de computador reducido (Reduced Instruction Set Computer)
API	Siglas en inglés de interfaz de programación de aplicaciones (Application Programming Interface)
B	Abreviatura de Bytes (8 bits)
k	Abreviatura de kilo (10^3)
I	Abreviatura de Input (entrada)
O	Abreviatura de Output (salida)
GND	Abreviatura de Ground (Tierra)
ALU	Siglas en inglés de Unidad Aritmeticológica (Arithmetic Logic Unit)
AHB	Siglas en inglés de bus de alto rendimiento avanzado (Advanced High Performance Bus)
APB	Siglas en inglés de bus avanzado de periféricos (Advanced Peripherial Bus)
DMA	Siglas en inglés de acceso directo a la memoria (Direct Memory Acces)
SRAM	Siglas en inglés de memoria estática de acceso aleatorio (Static Random Acces Memory)
ADC	Siglas en inglés de convertidor analógico-digital (Analog-Digital Converter)
DAC	Siglas en inglés de convertidor digital-analógico (Digital-analog Converter)
MSB	Siglas en inglés de bit de mayor peso (Most Significant Bit)
LSB	Siglas en inglés de bit de menor peso (Less significant Bit)
SPP	Siglas en inglés de protocolo de puerto serie (Serial Port Protocol)
I^2C	Siglas en inglés de circuito inter-integrado (Inter-Integrated Circuit)
GPIO	Siglas en inglés de entradas/Salidas de propósito general (General Purpose Input/Output)
ISA	Siglas en inglés de Arquitectura de set de instrucciones (Instruction Set Architecture)
IA	Inteligencia Artificial
I^2C	Siglas en inglés de circuito inter-integrado (Inter Integrated Circuit)
SPI	Siglas en inglés de interfaz de periféricos serie. (Serial Peripherial Interface)
USART	Siglas en inglés de transmisor/receptor universal serie síncrono y asíncrono (Universal Synchronous and Asynchronous serial Reciever and Transmitter)

Índice de figuras

Índice de Tablas

Índice

1. Introducción

1.1. Objetivo general del trabajo

El objetivo principal de este proyecto es diseñar e implementar un nuevo kit de laboratorio para la asignatura de Informática Industrial de l'Escola d'Enginyeria Barcelona Est (EEBE), Campus diagonal-Besós de la UPC.

En concreto se plantea actualizar el kit que se está utilizando actualmente en el laboratorio de informática industrial, utilizando componentes más modernos, empezando por un microcontrolador de 32 bits, en este caso el STM32, en lugar del 8051 de 8 bits que se utiliza actualmente. También se plantea el uso de comunicaciones habituales en la informática actual como I^2C, SPI o USART, además de los clásicos sensores pasivos, activos y diferentes actuadores, diseñando también un código para comprobar el correcto funcionamiento de estos.

1.2. Objetivos específicos del trabajo

1.2.1. Estudio teórico

El primer objetivo específico consiste en un estudio teórico de diferentes temas.

En primer lugar, es necesario estudiar la arquitectura del micro a utilizar. Una vez comprendida la arquitectura, se debe analizar el microcontrolador que se utilizará, así como sus puertos, recursos y diferentes funcionalidades. Seguidamente, es importante analizar la placa de desarrollo NUCLEO que se utilizará y observar y entender todas sus posibilidades de uso, así como sus pines y las posibilidades que ofrece.

En segundo lugar, se deben comprender los diferentes rasgos característicos de programación para posteriormente poder adaptar los diferentes componentes a ésta.

1.2.2. Elección de componentes

Un objetivo específico básico del proyecto es la correcta elección de componentes que integraran el kit.

Para hacer la elección de componentes se utiliza como guía el kit actual de laboratorio (descrito en el apartado 2.) y se pretende hacer una elección representativa de componentes. Es importante utilizar diferentes tipos de sensores (p.ej. digital, analógico lineal, analógico no-lineal), actuadores para interactuar con el usuario en tiempo real y también diferentes tipos de comunicaciones para poder practicarlas.

1.2.3. Diseño electrónico

El diseño electrónico es un objetivo esencial para cualquier ingeniero electrónico, es importante elegir correctamente los pines de la placa a utilizar para cada componente y añadir los componentes pasivos necesarios, así como hacer una conexión correcta de cada pin, ya sea alimentación, ground o un pin específico de la placa.

1.2.4. Prototipado

Una vez hecho el diseño electrónico es muy importante realizar un prototipo para poder comprobar el correcto funcionamiento de éste. En este caso se propone diseñar una PCB a partir del diseño electrónico realizado y comprobado. Una vez seguro del diseño de la PCB se dispondrá a su realización y a soldar los diferentes componentes y conectores previstos.

Generalmente también se suelen realizar primero simulación, se ha hecho una muy simple para comprobar el funcionamiento del simulador más que del micro, ya que al trabajar con una placa de desarrollo resulta más simple comprobar los componentes directamente.

1.2.5. Programación

La programación es cada día más demandada en el mundo de la electrónica, hoy en día es un conocimiento básico necesario para cualquier ingeniero, más aún en el campo de la electrónica. Por este motivo es otro objetivo esencial en este proyecto, en el que se incluirá programación básica del micro y los sensores y actuadores y se ofrecerá la posibilidad de programar diferentes protocolos de comunicación como SPI, I^2C y USART, esta última con un módulo bluetooth planteado para comunicar-se con un dispositivo móvil.

1.2.6. Comparación y análisis

Una vez completados todos los objetivos necesarios para el funcionamiento del prototipo es importante valorar que ventajas y desventajas que ofrece frente al kit actual, valorar si es viable económicamente haciendo un presupuesto lo más realista posible y, valorar, también, alguna alternativa comercial ya prefabricada.

Una vez analizadas todas las posibilidades, es importante llegar a una conclusión concreta que de sentido al trabajo realizado.

1.3. Motivación

La motivación principal para la realización del proyecto ha sido la imposibilidad de realizar un proyecto físico durante el curso de Informática Industrial por culpa de la limitación de movilidad que supuso la pandemia mundial de COVID-19 durante el cuadrimestre de primavera del curso 2019-20.

También hay muchos otros factores que sirven como motivación de cara a la realización de este proyecto: en primer lugar, ver el estado del mercado actual del mercado de µC (microcontroladores) y estudiar la arquitectura ARM; en segundo realizar una placa PCB de diseño propio con los diferentes periféricos con todo lo que esto comporta; en tercer lugar el reto de diseñar un código propio para leer los diferentes sensores y escoger el comportamiento de los actuadores en consecuencia; y por último, estudiar a fondo el µC con todas sus posibilidades y empezar a conocer el uso de comunicaciones cómo el I^2C, SPI y USART.

1.4. Requisitos previos

Para poder realizar este trabajo es necesario tener un conjunto de conocimientos previos adquiridos, en este caso, durante todo el Grado de Ingeniería Electrónica Industrial y Automática.

Para la realización del proyecto se utilizarán diferentes conocimientos adquiridos a lo largo del grado en diferentes asignaturas:

1. Primero de todo para organizar el proyecto se utilizan herramientas proporcionadas en Proyectos de Ingeniería (PE).
2. Para el diseño electrónico se necesitarán conocimientos muy variados, empezando por electrónica básica de Sistemas electrónicos (STI) y Teoría de circuitos y Máquinas Eléctricas (TCME), siguiendo con electrónica más avanzada de Electrónica Analógica y digital (EAEIA y ELDI) y acabando con electrónica más orientada al uso en Informática Industrial (IIEIA).

3. Para el diseño i realización de la PCB se requieren conocimientos a nivel de Hardware (dispositivos físicos) adquiridos de manera transversal, a nivel software iniciados en Tecnología Electrónica (TEEIA) y a nivel práctico obtenidos en la misma asignatura.
4. Para la programación se necesitarán muchos conceptos adquiridos en Informática Industrial (IIEIA), aunque para poder adaptarse y ahondar en el software y utilizar los diferentes protocolos de programación será necesario un aprendizaje extra sobre todo en el ámbito de programación y de integración del hardware al proyecto.

2. Kit actual del laboratorio

El kit actual de laboratorio funciona con el micro ATMEL AT89C5131A-M de 8 bits compatible con 8051. Además del micro, el kit incluye una fuente de alimentación y diferentes periféricos todo ello interconectado y acondicionado con los componentes necesarios para el correcto funcionamiento del kit de laboratorio.

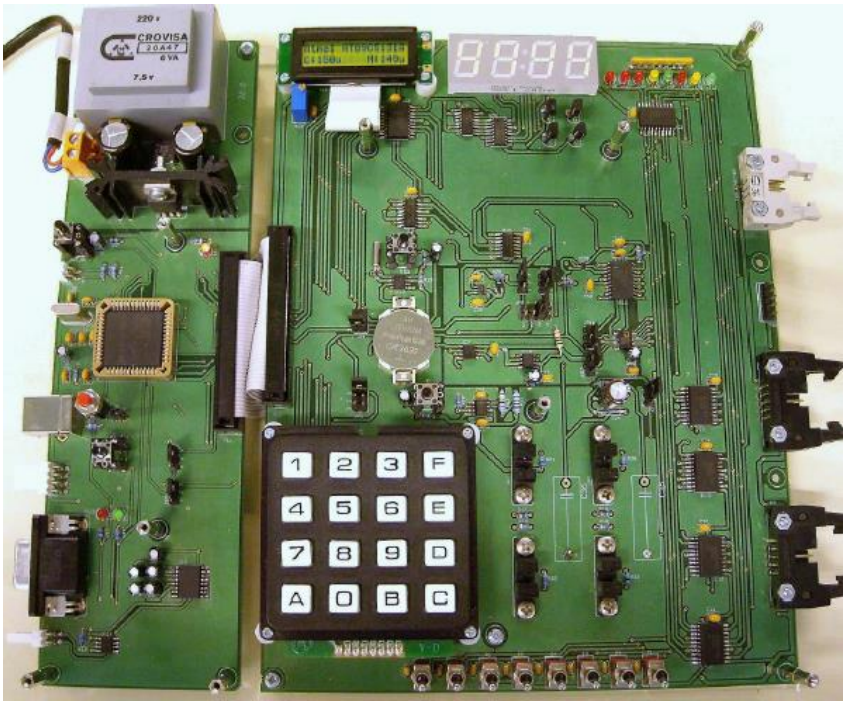


Figura 2.1.- *I2Kit actual, kit de laboratorio de Informática Industrial del campus EEBE*

Como se puede apreciar en la figura, el kit está formado por 2 placas PCB (ver **Figura 2.1**). En la de la izquierda tenemos la primera con alimentación, microcontrolador y USB para comunicación con el ordenador; y a la derecha la otra con el conjunto de periféricos y puertos de expansión.

2.1. Periféricos

El kit incluye los siguientes periféricos:

- Display 7 segmentos de 4 dígitos.
- Display LCD de 2 filas x 16 caracteres cada una.
- 8 diodos LED de diferentes colores (rojo, verde y amarillo).
- Teclado hexadecimal de 16 teclas no alimentado.
- 4 optoacopladores.
- 8 interruptores.
- 2 pulsadores para interrupciones externa INT0 e INT1.
- Convertidor ADC de 8 bits paralelo (ADC0804).
- Convertidor DAC de 8 bits paralelo (AD5330).
- Bus I^2C con ADC de 12 bits (ADS1000) y un reloj de tiempo real (DS1307) conectados.
- Bus SPI DAC de 10 bits (TLV5637) conectado.
- Bus USB.
- Comunicación serie.

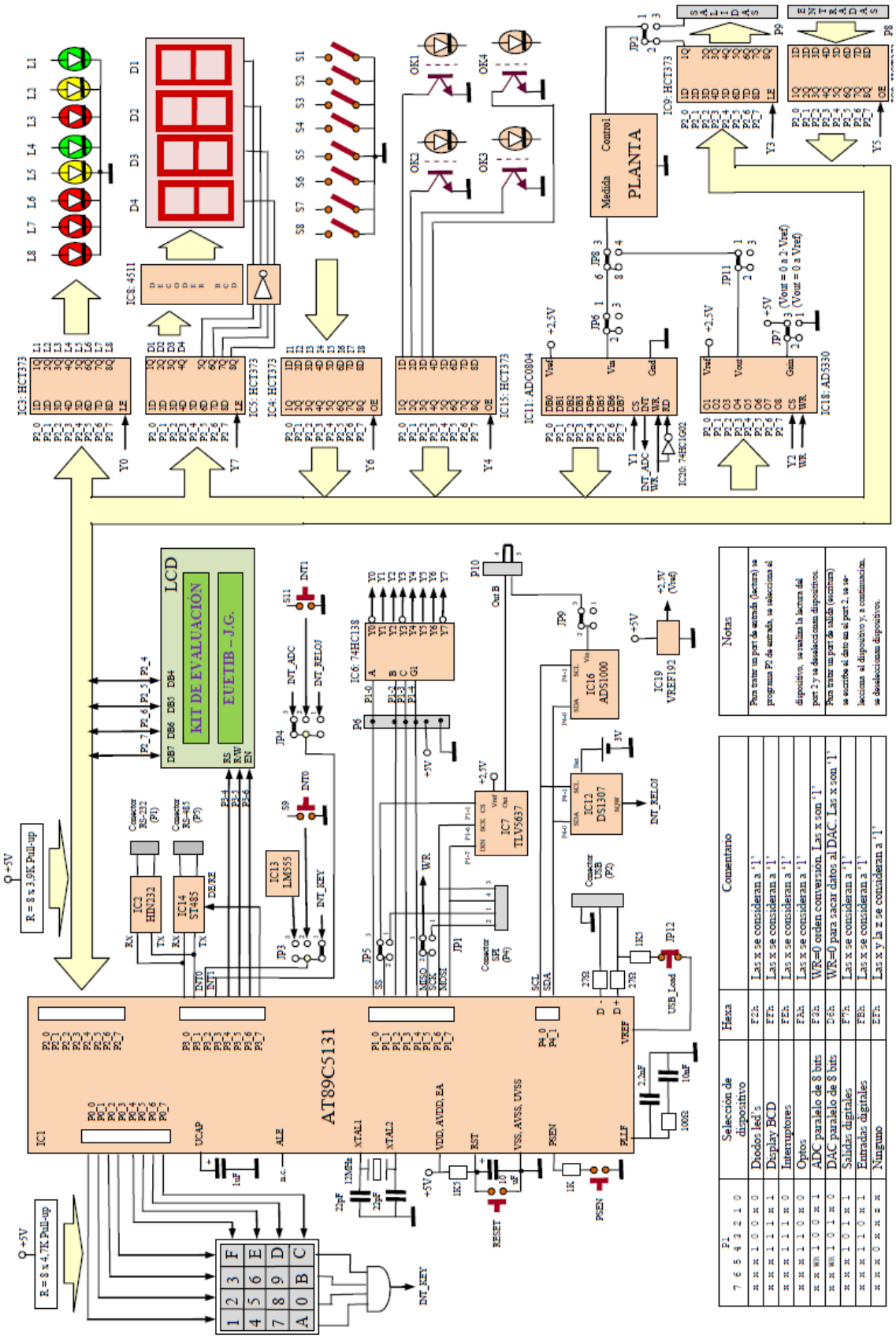


Figura 2.2.- *Diagrama de bloques del kit*

3. Arquitectura ARM

3.1. ¿Por qué ARM?

Dentro de todas las arquitecturas existentes en la actualidad se ha escogido la ARM por ser una de las predominantes hoy en día. Esta arquitectura es la más utilizada por fabricantes de micros, entre los cuales se encuentran STMicroelectronics, Atmel/Microchip, e Intel. Esta amplia oferta está claramente ligada a su éxito en el mundo de los smartphones y tabletas, así como en el de desarrolladores, que cada día más se decantan por esta arquitectura en el momento de escoger el componente esencial de cualquier proyecto, el microcontrolador.

3.2. Introducción

La arquitectura ARM está basada en un set de instrucciones de computador reducido, también llamado RISC por sus siglas en inglés (Reduced Instruction Set Computer). ARM (Advanced RISC Machines) es una familia de arquitecturas utilizada para microprocesadores y de la cual hay actualmente hasta la versión 9 ya presentada a principios de 2021[xiii].

A finales de 2020, una de las mayores empresas en el mundo de los semiconductores, NVIDIA, adquirió ARM por 40 mil millones de dólares[xii], obteniendo así SoftBank (conglomerado japonés al cual pertenecía ARM anteriormente) un beneficio directo de 8 mil millones de dólares en solo 4 años, después de haberla adquirido en 2016 por 32 mil millones.

ARM es una arquitectura de 32 bits que, aunque a partir de su versión V8-A, también se ha hecho una variante de 64 bits llamada ARM64.

3.3. Perfiles de arquitectura

Según la finalidad o el entorno de utilización podemos definir una serie de perfiles que optimizarán la arquitectura y priorizarán los recursos necesarios para cada caso.

3.3.1. Perfil A: Aplicación

El perfil A, destinado principalmente a soluciones para aplicaciones complejas, se basa en la serie de procesadores Cortex-A y Neoverse.

La serie de procesadores de aplicaciones Arm Cortex-A proporciona una gama de soluciones para dispositivos que realizan tareas informáticas complejas, como albergar una plataforma de sistema operativo (SO) enriquecida o admitir múltiples aplicaciones de software.

Estos perfiles de procesadores se utilizan mayoritariamente en PC, portátiles, televisores inteligentes, servidores, teléfonos inteligentes, automoción, almacenamiento en la nube y supercomputadoras.

Sus mayores ventajas frente a los otros perfiles son que ofrece el mayor rendimiento de todos los perfiles de arquitectura, es altamente eficiente en términos energéticos y que está optimizado para ejecutar sistemas operativos enriquecidos.

3.3.2. Perfil R: Tiempo real

El perfil R, destinado principalmente a aplicaciones de procesamiento en tiempo real, se basa en la serie de procesadores ARM Cortex-R.

Los procesadores de la serie Cortex-R ofrecen un procesamiento rápido, determinista y un alto rendimiento, también cumplen con las limitaciones del procesamiento en tiempo real en una gran variedad de situaciones. Estas unidades, combinan todas estas características en un

paquete optimizado en rendimiento, potencia consumida y superficie, lo que los convierte una solución fiable a sistemas sensibles que exigen una alta robustez ante los errores.

Estos perfiles de procesadores se utilizan mayoritariamente en equipos médicos, dispositivos de dirección, frenado y señalización de vehículos, equipos de almacenamiento de datos, redes y sistemas de control integrados.

Su mayor ventaja frente a los otros perfiles es su optimización para sistemas con requisitos en tiempo real.

3.3.3. Perfil M: Microcontrolador

El perfil M, destinado principalmente a microcontroladores versátiles, se basa en la serie de procesadores Cortex-M.

Esta familia de procesadores está optimizada para microcontroladores económicos y energéticamente eficientes.

Estos perfiles de procesadores se utilizan en una variedad de aplicaciones, incluidos dispositivos de consumo diario, industriales y de IoT, así como procesadores de seguridad, dispositivos integrados como wearables, pequeños sensores, módulos de comunicación y productos para Smart homes.

Su mayor ventaja frente a los otros perfiles es su reducido tamaño (diseñado para dispositivos pequeños), su bajo consumo de potencia y su alta eficiencia energética.

3.4. Características

La arquitectura ARM incluye las siguientes características típicas de las arquitecturas RISC:

- Un gran conjunto de registros uniforme.
- Una arquitectura load/store (cargar/almacenar), donde las operaciones de datos solo operan con el contenido de registros, no directamente con el de la memoria.
- Modos de direccionamiento simple, con todas las direcciones load/store determinadas solo por el contenido de los registros y las instrucciones.
- Longitud fija y uniforme de las instrucciones para simplificar la decodificación.

Además, también incluye otras características como:

- Control de las Unidades aritmeticológicas (ALU – Arithmetic Logic Unit) y unidad de desplazamiento (shifter) en la mayoría de las instrucciones de procesamiento de datos para maximizar el uso de éstas.
- Modos de direccionamiento de incremento y decremento automáticos para optimizar los bucles del programa.
- Múltiples instrucciones de load/store para maximizar el rendimiento de la ejecución.

Estas características adicionales a la arquitectura RISC básicas permiten a los procesadores ARM tener un buen balance de alto rendimiento, una reducción del tamaño del código, menos consumo y una menor área en silicio. Por este motivo actualmente los beneficios que les destacan frente a la competencia son:

- Seguridad integrada.
- Alto rendimiento y eficiencia energética.
- Gran ecosistema para soporte global.

- Generalizado en todos los mercados y ubicaciones.

3.4.1. RISC vs CISC

Como se comenta anteriormente, ARM es una arquitectura RISC. También existen sets de instrucciones tipo CISC (Complex Instruction Set Computer) que se diferencian de los anteriores de la siguiente manera.

El set de instrucciones RISC se basa en operaciones simples (hardware) con los cuales, a partir de código se pretende realizar operaciones más complejas; en cambio, el CISC se basa en una amplia variedad de operaciones simples y complejas (hardware), que permiten realizar cualquier operación utilizando pocas líneas de código.

Utilizar el primer set, conlleva la necesidad de escribir más código y, en consecuencia, precisa de más ciclos de procesamiento; al tratarse de instrucciones simples, es más complejo programar aplicaciones complicadas (aunque con la ayuda de un buen compilador se simplifica); por otro lado, al precisar de un hardware menos complejo, tiene un coste bastante menor.

RISC está orientado a aplicaciones que requieran una alta capacidad de procesamiento y se quieran programar en alto nivel. En cambio, CISC está más orientado a aplicaciones industriales que requieran una rápida respuesta con poco procesamiento (hay un gran número de aplicaciones industriales que ni si quiera agotan las posibilidades de los controladores CISC de 8 bits actuales).

En la **Tabla 3.1** se resumen estas diferencias entre estos dos tipos de arquitectura.

Tabla 3.1: *RISC vs CISC:*

	RISC	**CISC**
Significado siglas	Reduced Instruction Set Computer	Complex Instruction Set Computer
Instrucciones	Pocas y simples	Muchas y complejas
Carga de procesamiento	Software	Hardware
Velocidad de procesamiento	Menor	Mayor
Costo de fabricación	Menor	Mayor

3.5. Estado del arte

Esta arquitectura es una de las más utilizadas actualmente en el mundo, anualmente se producen miles de millones de dispositivos basados en la arquitectura ARM, entre los cuales destacan su utilización en prácticamente todos los teléfonos móviles y en la mayoría de tablets y wearables del mundo. Qualcomm, Samsung, Huawei y Apple (los cuatro fabricantes más grandes de chips para este tipo de dispositivo) utilizan tecnología ARM en sus procesadores, además de la CPU, hay una infinidad de componentes que incluyen esta misma ISA (Instruction Set Architecture) y se incluyen en los mismos dispositivos.

Se han vendido más de 100 mil millones de dispositivos ARM en los últimos 5 años y se espera la venta de 300 mil millones de unidades en los próximos años debido a la alta demanda de procesamiento. Al ritmo actual, prácticamente el 100% de los datos compartidos en el mundo serán procesados en ARM ya sea en la nube, en la red o en los dispositivos[xiii].

Actualmente la última versión disponible para su uso es ARMv9 presentada junto con sus características y novedades en 2021.

3.6. ARMv9

Tras una década desde la última versión de ARM, la versión 8, el 30 de marzo de 2021 se ha presentado su última actualización, la versión 9, que ha aparecido ante la creciente necesidad de procesamiento especializado para aplicaciones de creciente demanda como es la IA.

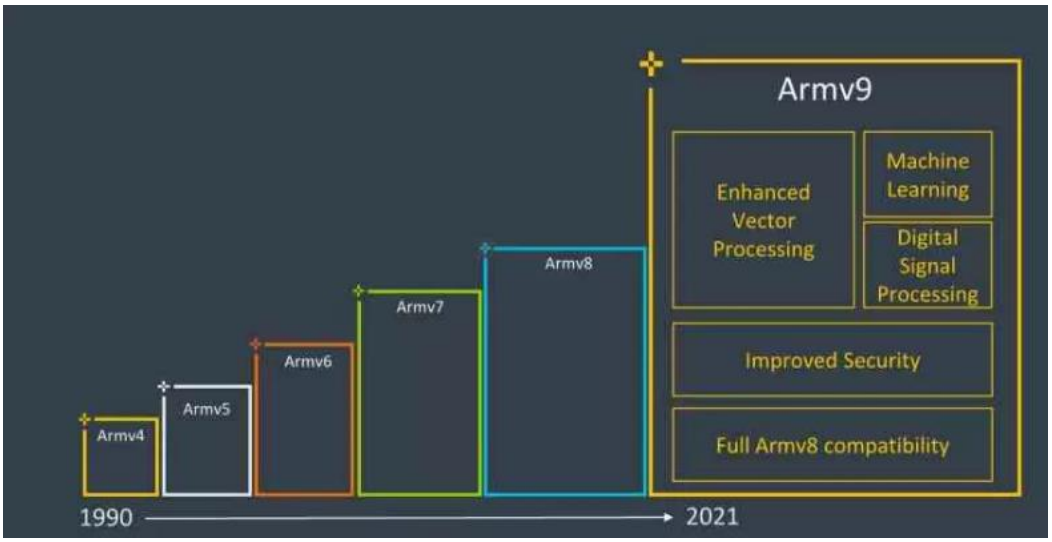


Figura 3.1.- *Evolución de ARM a lo largo de los años*

Hablando de la nueva ISA de ARM es importante decir que es totalmente compatible con la versión anterior, incluyendo también una serie de avances significativos empezando por las nuevas instrucciones SVE2 (Scalable Vector Extensions) que ayudarán a mejorar el rendimiento de los procesadores.

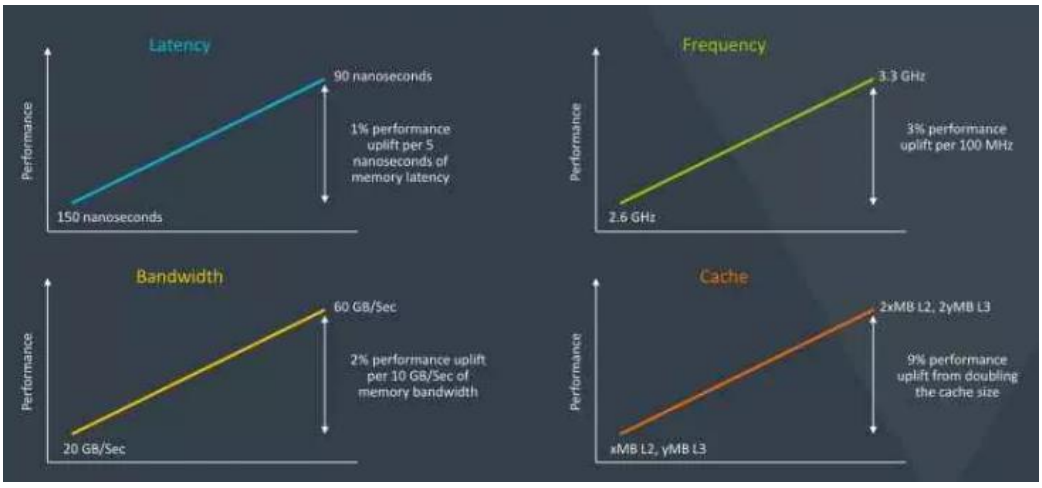


Figura 3.2.- *Mejoras de ARM v9 frente v8*

Otros importantes avances de este renovado set de instrucciones serán la reducción de la latencia de las instrucciones, un aumento de la velocidad media del clock, un aumento del ancho de banda de la memoria RAM y un considerable aumento de la caché.

Los rasgos que destaca ARM para su nueva versión son principalmente su seguridad, su capacidad de procesamiento adaptada a la inteligencia artificial (IA) y un mayor rendimiento por watt.

4. STM32F401xE

4.1. ¿Por qué STM32?

En los últimos años, de un 50% de las empresas y usuarios tienden a utilizar placas de desarrollo para sus prototipos o para realizar pruebas de software, hardware y nuevas tecnologías[6]. Este tipo de placas son muy útiles por su flexibilidad y adaptabilidad a cualquier proyecto de manera mucho más sencilla que crear un proyecto entero.

Dentro de las placas de desarrollo las más utilizadas son de diseño propio (26%) aunque el tipo que se va a utilizar, STM32 Nucleo, es utilizada aproximadamente por un 10% de los usuarios, justo por detrás de otras como Arduino con un 17% y Raspberry Pi con un 16%.

Otro factor importante en el momento de escoger este tipo de placa frente a otras es el uso de un solo microcontrolador (como en el 57% de los proyectos) de 32 bits (el más utilizado con un 61% [6]). Con un solo microcontrolador de estas características tenemos la simplicidad necesaria y a la vez una gran flexibilidad y capacidad computacional para nuestro proyecto.

4.2. Características

Estos microcontroladores están basados en ARM® Cortex®-M4 de 32 bits con una frecuencia de funcionamiento máxima de hasta 84 MHz.

Las características el procesador se resumen en la **Tabla 4.1**.

Tabla 4.1.- *Características de STM32F401RE[C]:*

Características	**NUCLEO-F401RE**
Microcontrolador de núcleo	Cortex®-M4
Frecuencia de funcionamiento máxima	84 MHz
Serie	STM32F4 Series
Línea de producto en la serie	STM32F401
Número de pines (R)	64 (encapsulado LQFP64)
Tamaño de la memoria Flash (E)	512 kB
Tamaño de la memoria SRAM	96 kB

Además de sus características básicas, el STM32 incluye:

- Un convertidor analógico/digital (ADC) de 12 bits de 16 canales.
- Hasta 11 timers:
  - 6 de 16 bits.
  - 2 de 32 bits.
  - 2 watchdogs.
  - 1 Sys Tick Timer.
- 50 entradas/salidas con posibilidad de interrupciones (todos los puertos son compatibles con 5 V)
- 12 interfaces de comunicación:
  - 3 I^2C
  - 3 USART
  - 4 SPIs
  - 1 SIDO
  - 1 comunicación avanzada USB 2.0 OTG.

En la **Figura 4.1** se representan esquemáticamente las partes que forman el microprocesador con un diagrama de bloques.

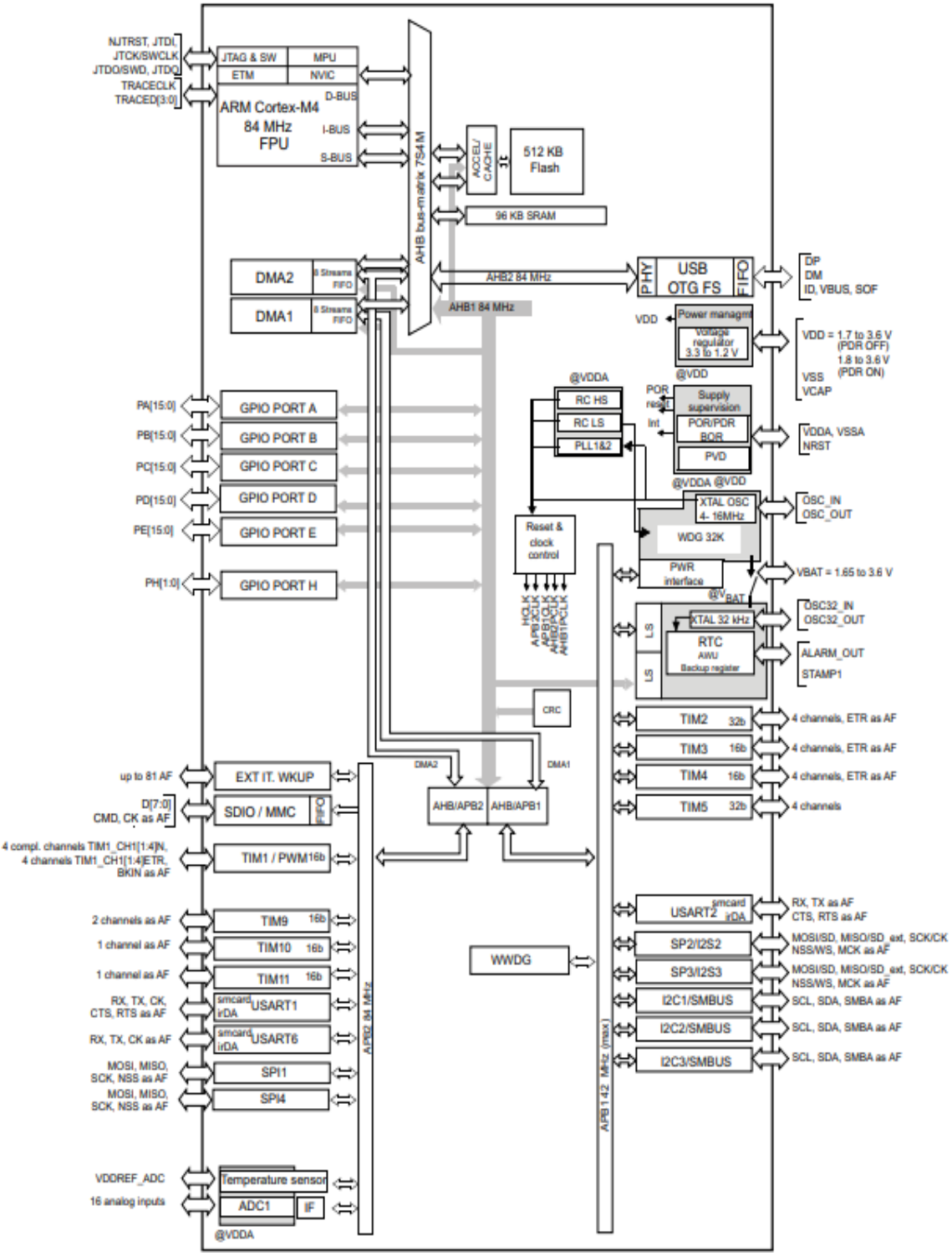


Figura 4.1.- *Diagrama de bloques del procesador STM32 (extraída de [C] figura 3)*

En la **Figura 4.2** se muestra la disposición de los 64 pines del STM32F401RE.

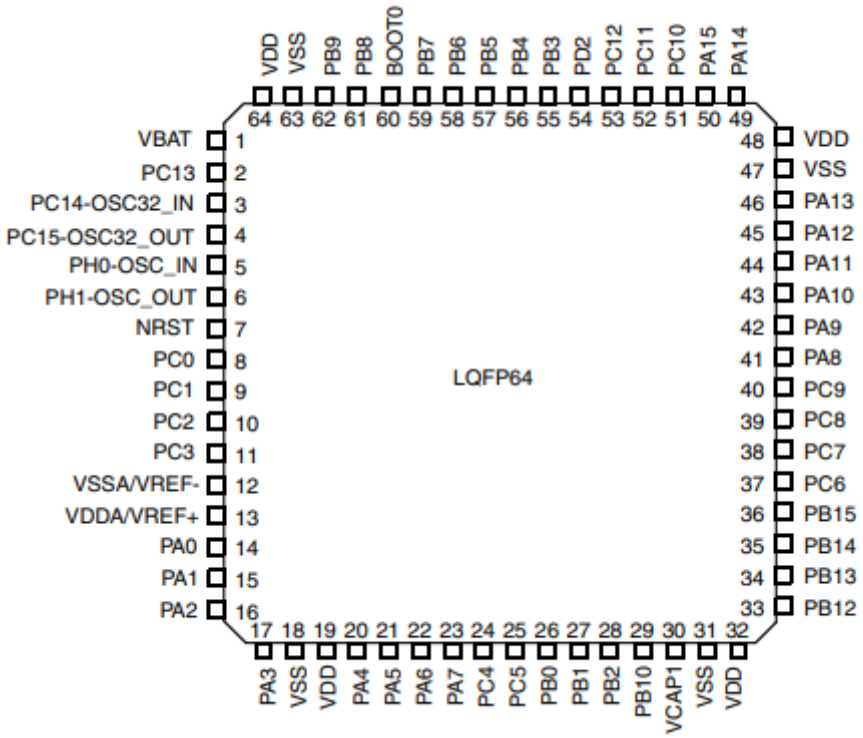


***Figura 4.2.**- Pinout de STM32 con encapsulado LQFP64 (extraída de [C] figura 12)*

4.3. Arquitectura

La arquitectura del µC consiste en una matriz multicapa de 32 bits con un bus de alto rendimiento avanzado (AHB) que interconecta:

- Seis masters (maestros):
  - Cortex®-M4
    - I-Bus
    - D-Bus
    - S-Bus
  - DMA1 (memoria de acceso directo) bus de memoria.
  - DMA2 bus de memoria
  - DMA2 bus de periféricos
- Cinco slaves (esclavos):
  - Memoria flash interna bus ICode (para buscar instrucciones).
  - Memoria flash interna bus DCode (para acceder al código de la memoria).
  - SRAM interna principal
  - Periféricos AHB1 incluyendo puentes entre AHB y APB (bus de periféricos avanzado) y periféricos de APB.
  - Periféricos AHB2

La matriz da acceso de master a slave, proporciona acceso concurrente y funcionamiento eficiente incluso cuando varios periféricos de alta velocidad funcionen simultáneamente (ver **Figura 4.3**).

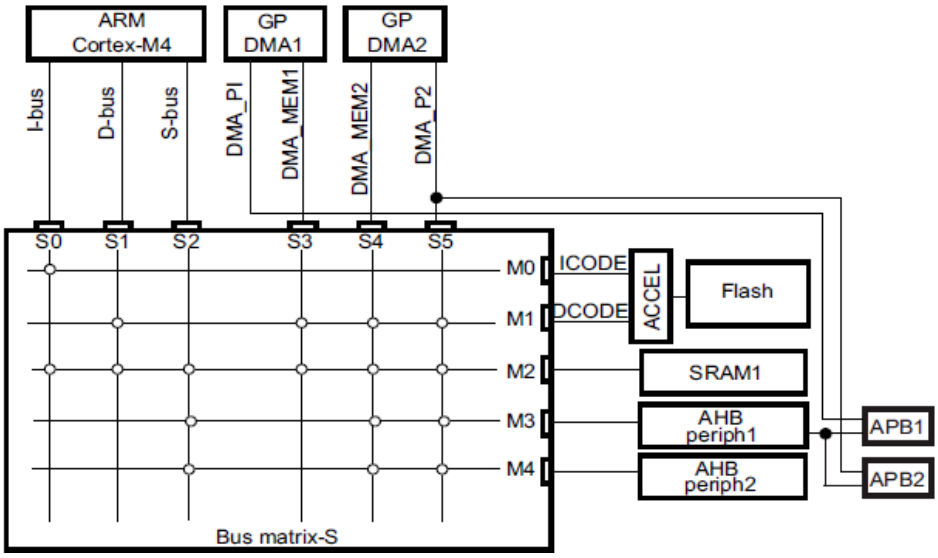


Figura 4.3.- *Arquitectura del sistema (extraída de [B] figura 1)*

4.4. Modos de inicialización

El bootloader (gestor de arranque) del programa es almacenado en la memoria ROM interna del dispositivo durante su producción. El principal objetivo de este, es descargar el programa de aplicación a la memoria flash interna a través de una comunicación serie disponible (USART, CAN, USB, …).

Dado que el micro tiene un mapa de memoria fijado, el área de código empieza en la dirección 0x0000 0000 (accesible por los buses ICode/Dcode) y el área de datos SRAM empieza en la dirección 0x2000 000 (accesible por el bus del sistema). Los Cortex®-M4 con FPU CPU siempre fijan el vector de reset en el bus ICode, lo que implica tener espacio disponible para el arranque solo en el área de código (típicamente, en la memoria flash). Los microcontroladores STM32F4 implementan un mecanismo especial que les permite inicializarse desde otras memorias, por lo que tiene 3 modos de inicialización:

- Inicio desde flash de usuario: se inicia desde la memoria flash interna del STM32. Es el modo de operación habitual.
- Inicio desde la memoria de sistema: se inicia desde la memoria de sistema definida por el fabricante durante su producción (ROM). Generalmente se utiliza para descargar el programa por el puerto serie, ya que en el bootloader definido por el fabricante puedes descargar el programa de la memoria flash. Este modo de inicialización, requiere ciertos pasos [C] y es relativamente complicado de utilizar, no está enfocado a un uso habitual.
- Inicio desde RAM interna: dado que la SRAM habitualmente no contiene suficiente memoria para almacenar el programa, este modo se suele utilizar para debugar ya que es más rápido y tiene más ciclos de lectura /escritura que la memoria flash. Una vez debugado el programa, se almacena en la memoria flash para su uso habitual.

4.5. Timers

En este apartado se resumen los diferentes timers disponibles en el procesador:

- TIM1: se puede ver como un generador de 3 fases de PWM multiplexado en 4 canales independientes que se pueden utilizar para:
  - Para capturar entradas.
  - Comparar salidas.
  - Generación de PWM
  - Modo de salida de un pulso.

TIM1 se configura como los contadores standard de 16 bits y tiene las mismas características que el resto de timers.

- TIM2 y TIM5: son contadores de 32 bits con auto recarga, contador up/down y prescaler de 16 bits.
- TIM3 y TIM4: son contadores de 16 bits con auto recarga, contador up/down y prescaler de 16 bits.

Los TIM2-5 cuentan con cuatro canales independientes con los mismos usos que el TIM1.

- TIM9: tiene dos canales independientes para captura de entrada/comparación de salida, PWM o modo de salida de un pulso.
- TIM10 y TIM11: son contadores de 16 bits con auto recarga, contador up y prescaler de 16 bits, juntos conforman un solo canal.

Los TIM9-11 pueden sincronizarse con los TIM2-5. También se pueden utilizar como simples contadores temporales.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	84
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	84
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	84

Figura 4.4.- *Comparativa entre diferentes timers del STM32*

- Watchdog independiente: contador de 12 bits con prescaler de 8 bits. Cuenta desde un clock generado por un oscilador interno RC de 32 kHz y actúa independientemente del clock principal del µC. Puede seguir en funcionamiento en estado de Stop y Standby del sistema. Se puede utilizar como watchdog (provocando un reinicio en caso de que el sistema se haya bloqueado) o como temporizador para la gestión del timeout una aplicación.
- Window Watchdog: contador de 7 bits que se puede configurar como watchdog. Utiliza el clock principal.
- SysTick timer: está dedicado a sistemas que operan en tiempo real pero también se puede usar como un contador down standard. Sus características:
    - Contador down de 24 bits.
    - Posibilidad de auto recarga.
    - Generacion de interrupción cuando el contador llega a 0.
    - Clock utilizado programable.

4.6. Pinout

En la **Tabla 4.2** se detallan los nombres de los 64 pines del STM34F401RE definiendo el tipo de pin del que se trata, su función principal y sus posibles funciones adicionales. La nomenclatura utilizada se define en la **Figura 4.5**.

Name	Abbreviation	Definition
Pin name	Unless otherwise specified in brackets below the pin name, the pin function during and after reset is the same as the actual pin name	
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input/ output pin
I/O structure	FT	5 V tolerant I/O
	B	Dedicated BOOT0 pin
	NRST	Bidirectional reset pin with embedded weak pull-up resistor
Notes	Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset	
Alternate functions	Functions selected through GPIOx_AFR registers	
Additional functions	Functions directly selected/enabled through peripheral registers	

Figura 4.5.- *Aclaración de la nomenclatura utilizada para los pines en la Tabla 4.2*

Tabla 4.2.- *Pinout del STM32F401xE con encapsulado LQFP64:*

PIN	Nombre	Tipo	Función	Función adicional
1	VBAT	S		
2	PC13	I/O	EVENTOUT	
3	PC14	I/O	EVENTOUT	OSC32_IN
4	PC15	I/O	EVENTOUT	OSC32_OUT
5	PH0	I/O	EVENTOUT	OSC_IN
6	PH1	I/O	EVENTOUT	OSC_OUT
7	NRST	I/O	EVENTOUT	
8	PC0	I/O	EVENTOUT	ADC1_IN10

9	PC1	I/O	EVENTOUT	ADC1_IN11
10	PC2	I/O	SPI2_MISO, I2S2ext_SD, EVENTOUT	ADC1_IN12
11	PC3	I/O	SPI2_MOSI, I2S2_SD, EVENTOUT	ADC1_IN13
12	VSSA	S		VREF -
13	VDDA	S		VREF +
14	PA0	I/O	USART2_CTS, TIM2_CH1/TIM2_ETR, TIM5_CH1, EVENTOUT	ADC1_IN0, WKUP
15	PA1	I/O	USART2_RTS, TIM2_CH2, TIM5_CH2, EVENTOUT	ADC1_IN1
16	PA2	I/O	USART2_TX, TIM2_CH3, TIM5_CH3, TIM9_CH1, EVENTOUT	ADC1_IN2
17	PA3	I/O	USART2_RX, TIM2_CH4, TIM5_CH4, TIM9_CH2, EVENTOUT	ADC1_IN3
18	VSS	S		
19	VDD	S		
20	PA4	I/O	SPI1_NSS, SPI3_NSS/I2S3_WS, USART2_CK, EVENTOUT	ADC1_IN4
21	PA5	I/O	SPI1_SCK, TIM2_CH1/TIM2_ETR, EVENTOUT	ADC1_IN5
22	PA6	I/O	SPI1_MISO, TIM1_BKIN, TIM3_CH1, EVENTOUT	ADC1_IN6
23	PA7	I/O	SPI1_MOSI, TIM1_CH1N, TIM3_CH2, EVENTOUT	ADC1_IN7
24	PC4	I/O	EVENTOUT	ADC1_IN14
25	PC5	I/O	EVENTOUT	ADC1_IN15
26	PB0	I/O	TIM1_CH2N, TIM3_CH3, EVENTOUT	ADC1_IN8
27	PB1	I/O	TIM1_CH3N, TIM3_CH4, EVENTOUT	ADC1_IN9
28	PB2	I/O	EVENTOUT	BOOT1
29	PB10	I/O	SPI2_SCK/I2S2_CK, I2C2_SCL, TIM2_CH3, EVENTOUT	
30	VACP1	S		
31	VSS	S		
32	VDD	S		
33	PB12	I/O	SPI2_NSS/I2S2_WS, I2C2_SMBA, TIM1_BKIN, EVENTOUT	
34	PB13	I/O	SPI2_SCK/I2S2_CK, TIM1_CH1N, EVENTOUT	
35	PB14	I/O	SPI2_MISO, I2S2ext_SD, TIM1_CH2N, EVENTOUT	
36	PB15	I/O	SPI2_MOSI/I2S2_SD, TIM1_CH3N, EVENTOUT	RTC_REFIN
37	PC6	I/O	I2S2_MCK, USART6_TX, TIM3_CH1, SDIO_D6, EVENTOUT	
38	PC7	I/O	I2S3_MCK, USART6_RX, TIM3_CH2, SDIO_D7, EVENTOUT	
39	PC8	I/O	USART6_CK, TIM3_CH3, SDIO_D0, EVENTOUT	
40	PC9	I/O	I2S_CKIN, I2C3_SDA, TIM3_CH4, SDIO_D1, MCO_2, EVENTOUT	
41	PA8	I/O	I2C3_SCL, USART1_CK, TIM1_CH1, OTG_FS_SOF, MCO_1, EVENTOUT	
42	PA9	I/O	I2C3_SMBA, USART1_TX, TIM1_CH2, EVENTOUT	OTG_FS_VBUS
43	PA10	I/O	USART1_RX, TIM1_CH3, OTG_FS_ID, EVENTOUT	
44	PA11	I/O	USART1_CTS, USART6_TX, TIM1_CH4, OTG_FS_DM, EVENTOUT	
45	PA12	I/O	USART1_RTS, USART6_RX, TIM1_ETR, OTG_FS_DP, EVENTOUT	
46	PA13	I/O	JTMS-SWDIO, EVENTOUT	
47	VSS	S		

48	VDD	S		
49	PA14	I/O	JTCK-SWCLK, EVENTOUT	
50	PA15	I/O	JTDI, SPI1_NSS, SPI3_NSS/I2S3_WS, TIM2_CH1/TIM2_ETR, JTDI, EVENTOUT	
51	PC10	I/O	SPI3_SCK/I2S3_CK, SDIO_D2, EVENTOUT	
52	PC11	I/O	I2S3ext_SD, SPI3_MISO, SDIO_D3, EVENTOUT	
53	PC12	I/O	SPI3_MOSI/I2S3_SD, SDIO_CK, EVENTOUT	
54	PD2	I/O	TIM3_ETR, SDIO_CMD, EVENTOUT	
55	PB3	I/O	JTDO-SWO, SPI1_SCK, SPI3_SCK/I2S3_CK, I2C2_SDA, TIM2_CH2, EVENTOUT	
56	PB4	I/O	NJTRST, SPI1_MISO, SPI3_MISO, I2S3ext_SD, I2C3_SDA, TIM3_CH1, EVENTOUT	
57	PB5		SPI1_MOSI, SPI3_MOSI/I2S3_SD, I2C1_SMBA, IM3_CH2, EVENTOUT	
58	PB6		I2C1_SCL, USART1_TX, TIM4_CH1, EVENTOUT	
59	PB7		I2C1_SCL, USART1_TX, TIM4_CH1, EVENTOUT	
60	BOOT0	I		V_{pp}
61	PB8		I2C1_SCL, TIM4_CH3, TIM10_CH1, SDIO_D4, EVENTOUT	
62	PB9		SPI2_NSS/I2S2_WS, I2C1_SDA, TIM4_CH4, TIM11_CH1, SDIO_D5, EVENTOUT	
63	VSS	S		
64	VDD	S		

5. NUCLEO-F401RE

Para facilitar el aprendizaje, STM creó una serie de placas de desarrollo llamadas NUCLEO. En concreto la placa escogida forma parte de las NUCLEO-64 (por el encapsulado del micro).

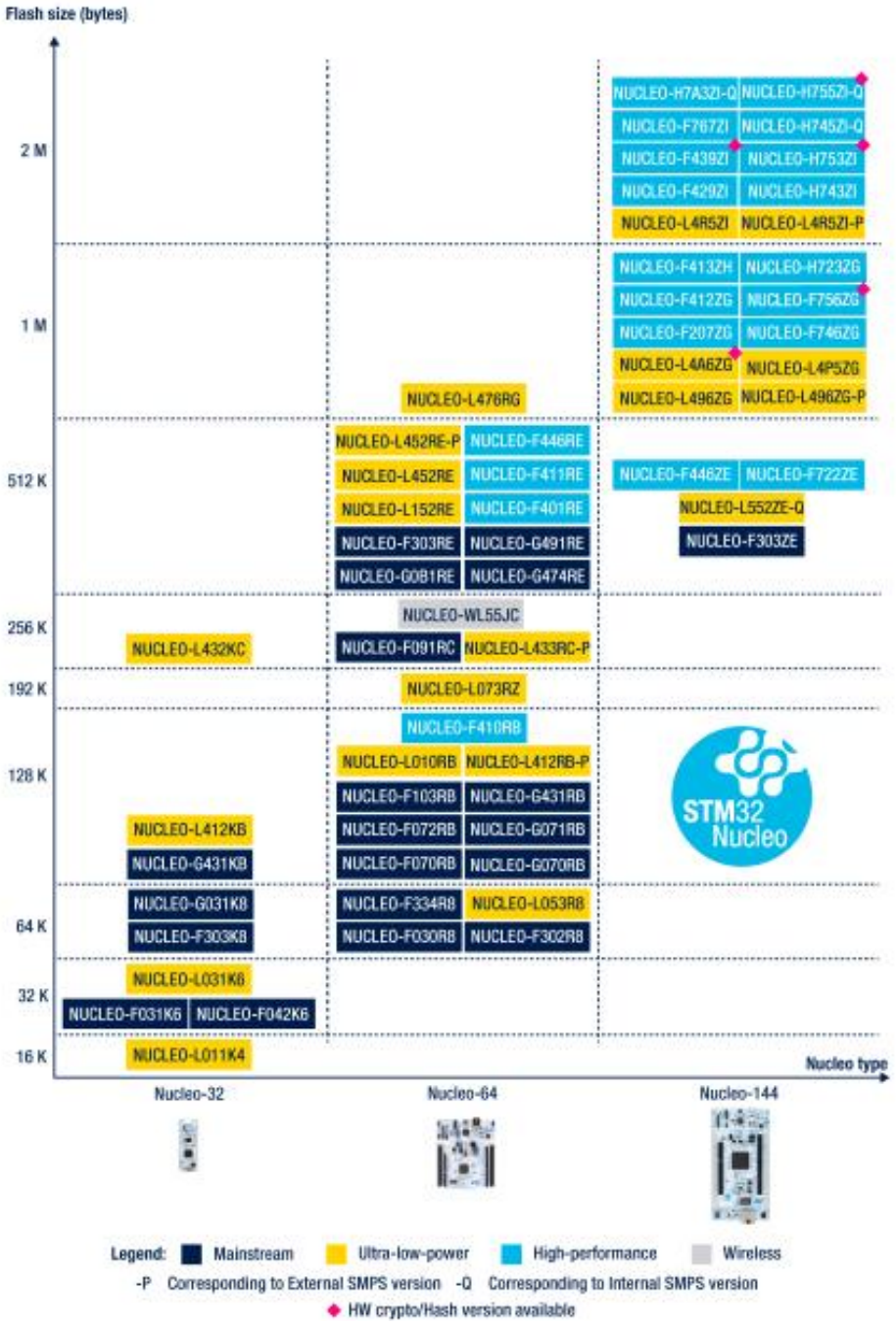


Figura 5.1.- *Clasificación de las placas STM32 NUCLEO*

Como se puede apreciar en la **Figura 5.1**, dentro de la familia NUCLEO, la placa escogida se encuentra dentro de la columna central, ya que el procesador tiene encapsulado LQFP64 con un patillaje de 64 pines. Dentro de esta familia se encuentra en el segundo escalón hablando del tamaño de su memoria flash con 512 kB solo por debajo de la NUCLEO-L-476RG. El fabricante ha catalogado la placa con la etiqueta de "high-performance" con lo que significa que puede ofrecer un alto rendimiento.

5.1. Componentes

La placa de desarrollo incluye los siguientes componentes:

- µC STM32 con encapsulado LQPF64.
- Tres LEDs:
  - LD1 indica el estado de la comunicación USB, tiene 3 colores verde naranja y rojo que se encienden en función del estado de comunicación (para más información mirar apartado 6.4 de [A].
  - LD2 programable por el usuario (PA5).
  - LD3 indica que la placa está alimentada.
- Dos pulsadores:
  - B1 (azul) programable por el usuario (PC13).
  - B2 (negro) resetea el µC STM32.
- Dos maneras de acceder a los recursos:
  - Pines de conectividad equivalentes a ARDUINO® Uno V3.
  - Pines de acceso a los I/O del procesador.
- Alimentación flexible:
  - Por USB VBUS o alimentación externa (3,3 V, 5 V, 7-12 V).
  - Punto de acceso a la administración de la alimentación.
- Programador y "depurador" (debugger) con conector SWD en la placa
- Capacidad de reenumerar el USB con tres tipos de interfaces suportadas:
  - Puerto de comunicaciones virtual (Virtual COM port).
  - Almacenamiento masivo.
  - Puerto de depuración (debug port).

La placa en cuestión está dividida en dos partes. La superior, donde se encuentra el ST-LINK/V2-1 junto con la conexión USB y los pines de programación SWD; y la inferior, dónde se encuentra el STM32 con un conjunto de pines de acceso a este, y de expansión (ver **Figura 5.2**).

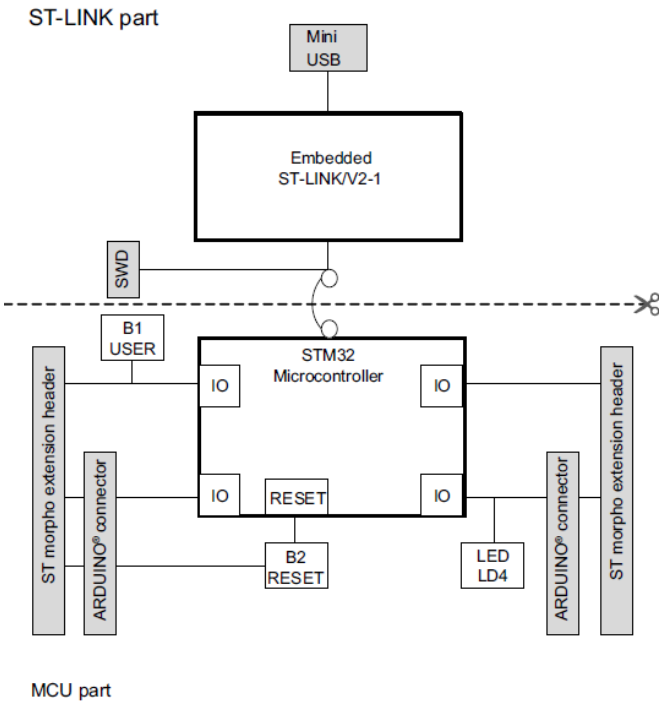


Figura 5.2.- *Diagrama de bloques del hardware (extraído de [A] Figura 2)*

En términos generales, en la parte superior sirve para programar el µC y alimentar la placa, también ofrece diferentes configuraciones a través de su pines y jumpers.

La parte inferior contiene la configuración básica para el funcionamiento de la unidad de control además de 3 LEDs y 2 pulsadores (comentados en el apartado anterior) junto con diferentes conectores de expansión y acceso a los puertos del STM32, además de un conjunto de configuraciones modificables en función del proyecto, analizadas todas las posibilidades, la configuración predeterminada se adapta perfectamente al proyecto.

Las dos placas pueden ser separadas para reducir el tamaño de la PCB y se pueden seguir utilizando conjuntamente si se conectan debidamente, pero esta característica no será utilizada en este proyecto.

A continuación, en la **Figura 5.3** se pueden observar la distribución de componentes y conectores en la placa vista desde arriba, en la **Figura 5.4** vista desde abajo y, en la **Figura 5.5**, se pueden ver las dimensiones detalladas de la placa.

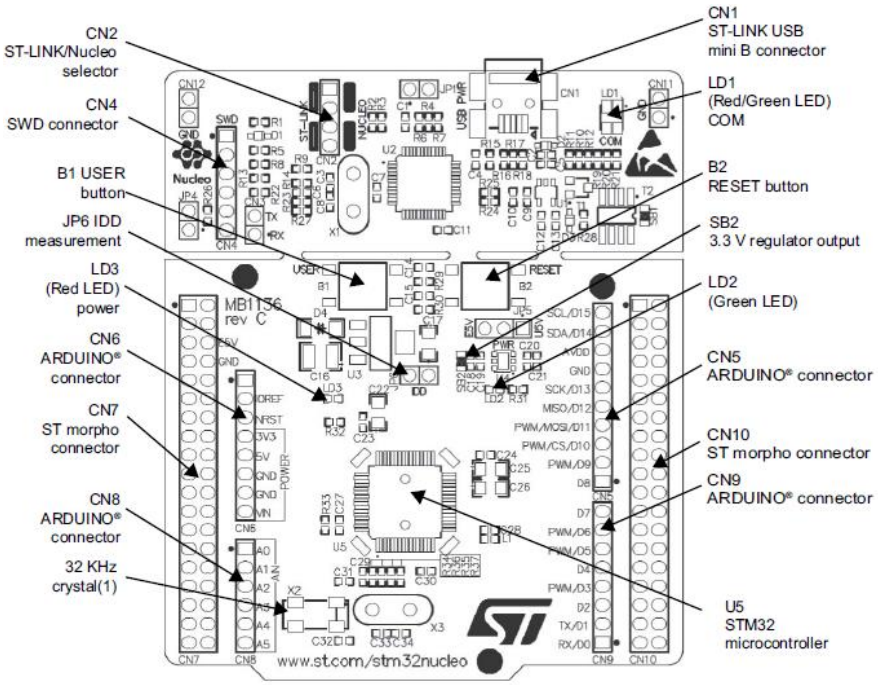


Figura 5.3.- *Distribución de la cara superior (TOP) (extraído de [A] Figura 3)*

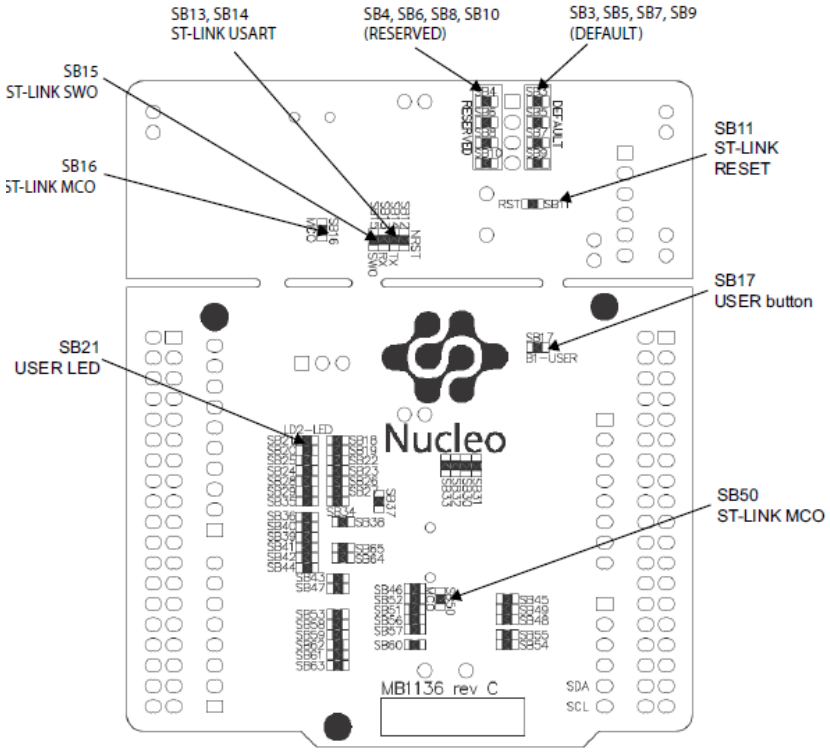


***Figura 5.4.**-Distribución de la cara inferior (BOTTOM) (extraída de [A] figura 4)*

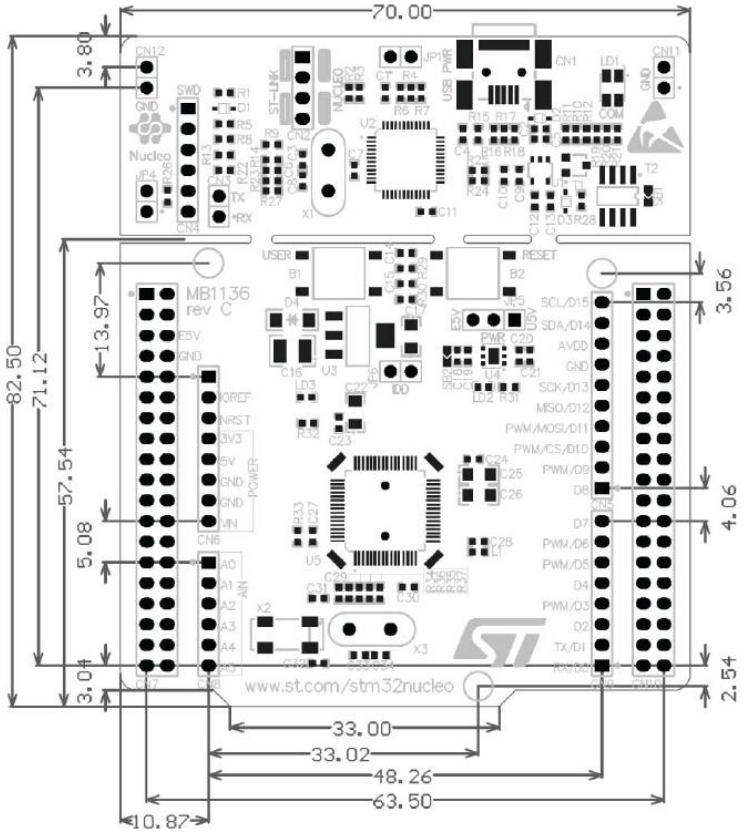


***Figura 5.5.**- Dimensiones de la placa (extraído de [A] Figura 5)*

5.2. Pinout

En la **Tabla 5.1** se puede observar dónde está conectado cada pin de los conectores CN7 y CN10, que dan acceso a los diferentes puertos del micro.

Tabla 5.1.- *Pinout de la placa (Tabla 29 de [A]):*

CN7 pines impares		CN7 pines pares		CN10 pines impares		CN10 pines pares	
Pin	Nombre	Nombre	Pin	Pin	Nombre	Nombre	Pin
1	PC10	PC11	2	1	PC9	PC8	2
3	PC12	PD2	4	3	PB8	PC6	4
5	VDD	E5V	6	5	PB9	PC5	6
7	BOOT0(1)	GND	8	7	AVDD	U5V(2)	8
9	-	-	10	9	GND	-	10
11	-	IOREF	12	11	PA5 (LD2)	PA12	12
13	PA13(3)	RESET	14	13	PA6	PA11	14
15	PA14(3)	+3,3 V	16	15	PA7	PB12	16
17	PA15	+5,0 V	18	17	PB6	-	18
19	GND	GND	20	19	PC7	GND	20
21	PB7	GND	22	21	PA9	PB2	22
23	PC13 (B1)	V_{IN}	24	23	PA8	PB1	24
25	PC14	-	26	25	PB10	PB15	26
27	PC15	PA0	28	27	PB4	PB14	28
29	PH0	PA1	30	29	PB5	PB13	30
31	PH1	PA4	32	31	PB3	AGND	32
33	VBAT	PB0	34	33	PA10	PC4	34
35	PC2	PC1 o PB9(4)	36	35	PA2	-	36
37	PC3	PC0 o PB8(4)	38	37	PA3	-	38

1. El estado predeterminado de BOOT0 es LOW. Se puede configurar como HIGH si se pone un jumper en el pin5-7 de CN7.
2. U5V son 5 V del conector ST-LINK/V2-1 USB.
3. PA13 y PA14 comparten señales SWD conectadas a ST-LINK/V2-1, no es recomendable utilizarlas como pines de I/O si la parte de ST-LINK no está cortada.
4. Mirar tabla 10 de [A] para más detalles.

En la **Tabla 5.2** se detalla la correspondencia pin a pin de los pines del micro con los pines dispuestos en la placa, al haber 76 pines de conexión en la NUCLEO y el micro de tener un encapsulado LQFP64 de 64 pines es lógico que haya conexiones destinadas a otras funciones como p. ej. Alimentación. También hay que tener en cuenta que la placa cuenta con diferentes componentes (pulsadores, LEDs, osciladores) que limitarán la funcionalidad de algunos pines.

Tabla 5.2.- *Correspondencia del pinout de la placa NUCLEO-STM32F401RE con el µC STM32F401RE:*

NUCLEO	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	7.9	7.10	7.11	7.12	7.13
STM32	51	52	53	54	64	-	60	63	-	-	-	IOREF	46
Nombre	PC10	PC11	PC12	PD2	VDD	-	BOOT0	GND	-	-	-	IOREF	PA13
NUCLEO	7.14	7.15	7.16	7.17	7.18	7.19	7.20	7.21	7.22	7.23	7.24	7.25	7.26
STM32	7	49	-	50	-	-	-	59	47	2	VIN	3	-
Nombre	RESET	PA14	3.3 V	PA15	5 V	GND	GND	PB7	GND	PC13	VIN	PC14	-
NUCLEO	7.27	7.28	7.29	7.30	7.31	7.32	7.33	7.34	7.35	7.36	7.37	7.38	-
STM32	4	14	5	15	6	20	1	26	10	9	11	8	-
Nombre	PC15	PA0	PH0	PA1	PH1	PA4	VBAT	PB0	PC2	PC1	PC3	PC0	-
NUCLEO	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10	4.11	4.12	4.13
STM32	40	39	61	37	62	25	13	-	18	-	21	45	22
Nombre	PC9	PC8	PB8	PC6	PB9	PC5	AVDD	U5V	GND	-	PA5	PA12	PA6
NUCLEO	4.14	4.15	4.16	4.17	4.18	4.19	4.20	4.21	4.22	4.23	4.24	4.25	4.26
STM32	44	23	33	58	-	38	31	42	28	41	27	29	36
Nombre	PA11	PA7	PB12	PB6	-	PC7	GND	PA9	PB2	PA8	PB1	PB10	PB15
NUCLEO	4.27	4.28	4.29	4.30	4.31	4.32	4.33	4.34	4.35	4.36	4.37	4.38	-
STM32	56	35	57	34	55	12	43	24	16	-	17	-	-
Nombre	PB4	PB14	PB5	PB13	PB3	AGND	PA10	PC4	PA2	-	PA3	-	-

5.3. Modos de alimentación

Dentro de la gran flexibilidad de la placa, además de poderse alimentar por USB, por donde también se puede programar y debugar, también nos ofrece la posibilidad de alimentarla externamente desde dos de sus pines.

Esta característica es muy interesante ya que en proyectos donde se consuma una corriente relativamente elevada, la fuente de alimentación de la placa no basta para alimentar a los periféricos, por ese motivo es habitual añadir una alimentación externa para alimentar estos componentes. Al añadir una fuente de alimentación externa, es relativamente incómodo utilizar dos fuentes (implican dos generalmente dos cables y dos transformadores), por ese motivo, una vez se alimenta externamente es interesante que se pueda alimentar directamente todo el conjunto sin tener que hacer un gran esfuerzo.

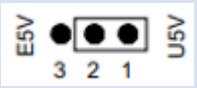
Los pines destinados a la alimentación externa son concretamente el pin 6 del conector CN7, llamado E5V, y el pin 24 de este mismo conector (que a su vez está puenteado con el pin 8 de CN6), llamado VIN. En la **Tabla 5.3** se detallan las características específicas de alimentación para cada uno de estos pines.

Tabla 5.3.- *Nombre, localización y características de los pines de alimentación externa:*

Nombre del pin	Conector y N.º de pin	Rango de voltaje	Corriente máximo	Limitaciones
E5V	CN7 pin 6	4,75-5,25V	500 mA	-
VIN	CN7 pin 24 CN6 pin 8	7,00-12,00V	800 mA	El corriente de entrada máximo depende directamente del voltaje: $V_{IN} = 7\ V \rightarrow I_{IN\ max} = 800\ mA$ $7V < V_{IN} \leq 9V \rightarrow I_{IN\ max} = 450\ mA$ $9V < V_{IN} \leq 12V \rightarrow I_{IN\ max} = 250\ mA$

Siempre que se quiera alimentar la placa de manera externa se debe tener el jumper 1 (JP1) abierto y el jumper 5 (JP5) en la posición deseada según se dese alimentar la placa (ver **Tabla 5.4**).

Tabla 5.4.- *Posicionamiento del jumper PJ5 para seleccionar un modo u otro de alimentación:*

Jumper	Descripción	Posición
JP5	Se utiliza U5V (ST-LINK VBUS) como fuente de alimentación (configuración por defecto)	
	Se utiliza E5V o VIN como fuente de alimentación externa	

El procedimiento a seguir para cambiar de una fuente de alimentación a otra es el siguiente:

1. Posicionar JP5 puenteando los pines 2 y 3.
2. Comprobar que JP1 está desconectado.
3. Comprobar que el voltaje de la fuente de alimentación externa es el deseado.
4. Conectar la fuente de alimentación externa en su pin correspondiente (VIN o E5V).
5. Comprobar que se enciende el LED LD3 de la placa.
6. Conectar la placa por USB (CN1) ya sea al ordenador o alimentación.

5.4. Conexión equivalente a ARDUINO® Uno V3

Otra característica extra añadida a esta placa de desarrollo es la compatibilidad pin a pin con una placa de ARDUINO® Uno V3 estándar, es decir que se puede conectar exactamente lo mismo que conectarías a una de ese tipo, haciéndola 100% compatible con shields prefabricadas y componentes ya testeados en placas de ese modelo. Esta compatibilidad es posible gracias a los conectores CN5, CN6, CN8 y CN9 que, como se puede ver en la **Figura 5.6**, están simplemente puenteados con pines de CN7 y CN10 por lo que no se trata de pines adicionales.

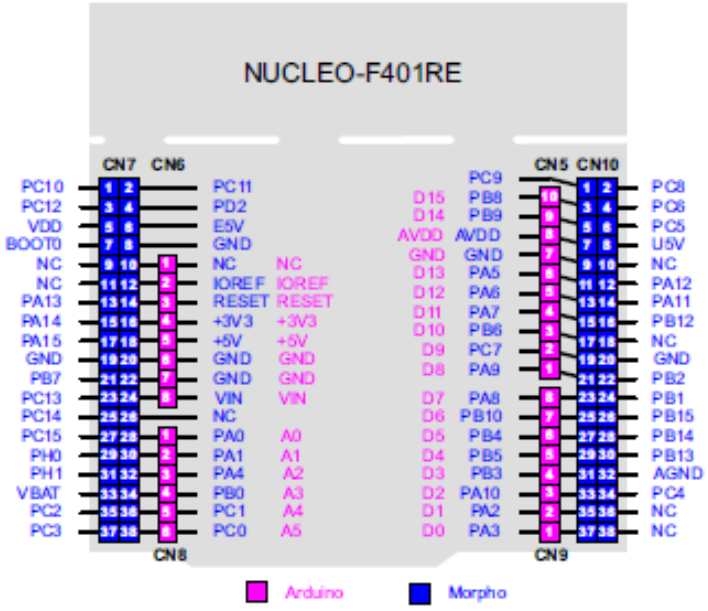


Figura 5.6.- *Diagrama de conexión predeterminada de la placa con shield de ARDUINO® Uno V3 (extraído de [A] Figura 18)*

Hoy en día, ARDUINO® es un estándar en electrónica, sobre todo para principiantes gracias a su simplicidad y la gran variedad de ejemplos publicados en la web. Esta compatibilidad tiene el objetivo de mostrar que se pueden adaptar proyectos creados con ARDUINO y además ofrece la posibilidad de ampliarlos o realizar proyectos más potentes en un futuro con la misma placa, ya que es más potente, tiene más pines y tiene un precio inferior (11,59 € contra 20,00 €).

Todos estos factores junto a un software específico, STM32 Cube IDE (aunque también se puede programar con la misma IDE de ARDUINO®), con el que se pueden configurar los pines de manera gráfica y muy intuitiva, puede ser un gancho para llevar a la comunidad a realizar cada día más proyectos con este tipo de placas, además de poder convertirse en un nuevo estándar como es hoy en día ARDUINO® para entrar por primera vez en el mundo de la electrónica.

Tabla 5.5.- *Equivalencia de los conectores de STM32 con los de ARDUINO® Uno V3:*

Conector	Pin	Pin de Uno V3	Pin del STM32	Función
Conectores de la izquierda				
CN6 Alimentación	1	NC	NC	-
	2	IOREF	IOREF	3,3 V Ref
	3	RESET	NRST	
	4	+ 3,3 V	+3V3	3,3 V I/O
	5	+ 5,0 V	+5V	5 V O
	6	GND	GND	Ground
	7	GND	GND	Ground
	8	V_{IN}	VIN	Alimentación externa
CN8 Analógico	1	A0	PA0	ADC1_0
	2	A1	PA1	ADC1_1
	3	A2	PA4	ADC1_4
	4	A3	PB0	ADC1_8
	5	A4	PC9 o PB9(1)	ADC1_11 (PC1) o I2C1_SDA (PB9)
	6	A5	PC0 o PB8(1)	ADC1_10 (PC0) o I2C1_SCL (PB8)
Conectores de la derecha				
CN5 Digital	10	D15	PB8	I2C1_SLC
	9	D14	PB9	I2C1_SDA
	8	AREF	AVDD	AVDD
	7	GND	GND	Ground
	6	D13	PA5	SPI1_SCK
	5	D12	PA6	SPI1_MISO
	4	D11	PA7	TIM1_CH1N o SPI1_MOSI
	3	D10	PB6	TIM4_CH1 o SPI1_CS
	2	D9	PC7	TIM3_CH2
	1	D8	PA9	-
CN9 Digital	8	D7	PA8	-
	7	D6	PB10	TIM2_CH3
	6	D5	PB4	TIM3_CH1
	5	D4	PB5	-
	4	D3	PB3	TIM2_CH2
	3	D2	PA10	-
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

1. Mirar en la tabla 10 de [A] para más detalles

6. Elección de componentes

Los componentes elegidos se basan en el kit actual de laboratorio descrito en el apartado 2.

En este apartado es clave coger una muestra representativa de diferentes sensores, comunicaciones y visualizaciones. En este sentido, al tener un objetivo didáctico, tiene sentido escoger sensores y actuadores básicos en la electrónica, sensores que midan la misma magnitud con diferentes métodos o que lo transmitan con diferentes protocolos para ver diferentes métodos para realizar una misma tarea y ver sus diferentes lecturas a través de algún tipo de visualización.

Teniendo en cuenta todos estos aspectos se han escogido los siguientes componentes:

- Sensores pasivos básicos:
  - 2 pulsadores NO[xxx].
  - 2 interruptores[xxxi].
  - 1 teclado matricial 4x4[xxxii].
- Actuadores básicos:
  - 4 LEDs de diferentes colores[xxix].
- Dispositivo de visualización:
  - 1 LCD 2x16 mini[xxxiii].
- Sensores de distancia:
  - 1 HC-SR04: sensor de distancia por ultrasonidos con salida digital[xxxiv].
  - 1 GP2Y0A21YK0F: sensor de distancia por infrarrojos con salida analógica no lineal[xxxvi].
- Sensores de temperatura:
  - 1 LM35: sensor de temperatura con salida analógica lineal[xxxv].
  - 1 ADT7420: sensor de temperatura que se comunica por I^2C[xxxvii].
  - 1 BME280: sensor de temperatura humedad y presión que se comunica por SPI[xxxviii].
- Comunicaciones:
  - 1 ADT7420: también sensor de temperatura (ver arriba), se comunica por I^2C.
  - 1 BME280: también sensor de temperatura (ver arriba), se comunica por SPI.
  - 1 HC-05: módulo Bluetooth que se comunica por comunicación serie[xxxix].

6.1. Pulsadores

Los pulsadores (P1 y P2) elegidos son pulsadores NO, normalmente abiertos, de 2 terminales.

Estos componentes simplemente se deben conectar a masa o voltaje, con su respectiva pull-up o pull-down y a un pin de entrada digital en la placa.

Figura 6.1.- *Pulsador equivalente con el escogido (izquierda) y representación esquemática de un pulsador normalmente abierto (derecha)*

6.2. Interruptores

Los interruptores (I1 e I2) escogidos son interruptores de 2 posiciones con 3 terminales que, a la práctica, serán 2, ya que dos de ellos estarán cortocircuitados.

Su conexión es exactamente igual que la de un pulsador, ya que esencialmente hacen la misma función, la única diferencia es que el pulsador recupera automáticamente su estado inicial después de ser pulsado y el interruptor no.

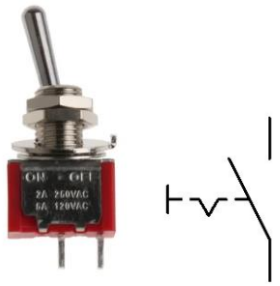


Figura 6.2.- *Interruptor equivalente al escogido (izquierda) y representación esquemática de un interruptor (derecha)*

6.1. Teclado matricial 4x4

El teclado matricial escogido es de 4 filas por 4 columnas, es no alimentado y tiene 8 pines, uno por cada fila y columna. Se puede utilizar de diversas maneras, pero en este caso simplemente se conectará a 8 pines distintos de la placa que puedan ser configurados como entrada y como salida. Habría que poner resistencias de pull-up/pull-down en todos los pines, pero al ser estos configurables en el micro se pueden omitir.

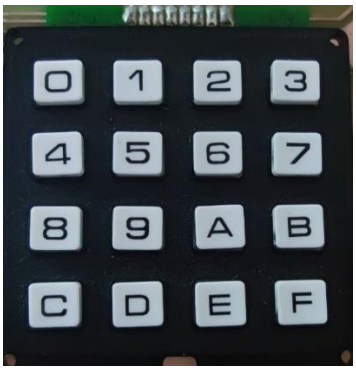


Figura 6.3.- *Teclado matricial 4x4 no alimentado*

Su funcionamiento es relativamente sencillo, es muy parecido al de cualquier pulsador. En este caso, al pulsar cualquier tecla se cortocircuitan su fila con su columna correspondiente, p.ej. siguiendo el teclado de la **Figura 6.3** si pulsáramos la tecla A cortocircuitaríamos el pin referente a la fila 3 (empezando desde arriba) con el de la columna 3 (empezando desde la izquierda).

Los pines están ordenados de izquierda a derecha de la siguiente manera: X1, X2, X3, X4, Y1, Y2, Y3, Y4. Siendo X1 la fila superior e Y1 la columna de la izquierda y siguiendo así la matriz.

6.2. LED

Se utilizarán LEDs (Light Emmitting Diode) de diferentes colores, verde, amarillo, rojo y azul (LED1-LED4 en este orden) que, aunque a la práctica sus tensiones AK no son exactamente iguales, los trataremos de la misma forma.

Los LED tienen polaridad, se tienen que alimentar por el ánodo con una tensión mayor a su tensión umbral (se pueden alimentar con 3,3 o 5 V indistintamente), es importante añadir una resistencia adecuada que determinará la intensidad que circulará por estos y conectar el cátodo a tierra para crear la diferencia de potencial.

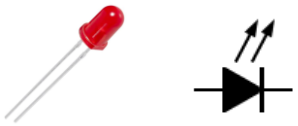


Figura 6.4.- *LED escogido (derecha) y representación esquemática de un LED (izquierda)*

Las tensiones umbral de los diferentes LED, observadas experimentalmente, son de entre 1,9 y 2,6 V obteniendo un valor medio de 2,25 V. Con este potencial, sabiendo que la placa alimenta sus salidas digitales a 3,3 V y que una media de 3 mA por LED es razonable se ha calculado siguiendo una simple ley de Ohm el valor necesario para las resistencias, en este caso se ha obtenido un valor de 350 Ω que se ha redondeado a 330 que es un valor normalizado que ya teníamos disponible (ver ecuación [6.1]).

$$\textbf{[6.1]} \quad R = \frac{\Delta V}{I} = \frac{3{,}3\ V - 2{,}25\ V}{3\ mA} = 350\ \Omega \cong 330\ \Omega$$

6.3. Visualizador LCD

EL visualizador LCD escogido es un visualizador de 2 filas con 16 caracteres en cada una de ellas.

El visualizador tiene 16 pines de conexión con el patillaje mostrado en la **Tabla 6.1**.

Tabla 6.1.- *Patillaje del visualizador LCD 2x16:*

Pin	Símbolo	Descripción
1	Vss	Ground (GND)
2	Vdd	Alimentación (5V)
3	V0	Ajuste de contraste
4	RS	Señal de selector de registro
5	R/W	Leer/escribir datos (Read/Write)
6	E	Señal Enable
7	DB0	Bus de datos pin 0
8	DB1	Bus de datos pin 1
9	DB2	Bus de datos pin 2
10	DB3	Bus de datos pin 3
11	DB4	Bus de datos pin 4
12	DB5	Bus de datos pin 5
13	DB6	Bus de datos pin 6
14	DB7	Bus de datos pin 7
15	LED+	Alimentación + de la retroiluminación
16	LED-	Alimentación - de la retroiluminación

Para la conexión de la LCD se necesita alimentación, un potenciómetro que permita ajustar el contraste (ver **Figura 6.5**), alimentación para la retroiluminación LED (igual a 6.2), tres pines digitales para RS (Register Select), R/W (Read/Write) y E (Enable) y para los datos se puede utilizar un bus de 4 o de 8 pines, en este caso se utilizará uno de 4, DB4-DB7 para ahorrar pines.

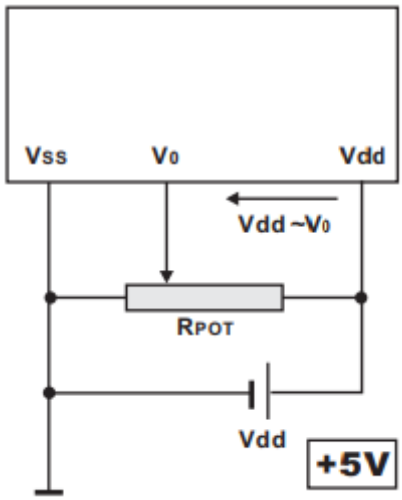


Figura 6.5.- *Diagrama de conexión de la alimentación del LCD*

Para la programación se puede observar en la **Tabla 6.2** el mapeado de registros según fila y carácter y en la **Figura 6.7** se pueden observar los diferentes caracteres que se pueden mostrar en el display en función del código dado en el bus de datos.

Tabla 6.2.- *Código de direcciones de los caracteres del visualizador LCD:*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Figura 6.6.- *LCD 2x16 con cable flex*

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	×	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	■

Figura 6.7.- *Mapa de caracteres mostrados en el display en función de los bits de entrada*

6.4. Sensor de distancia por ultrasonidos HC-SR04

El sensor HC-SR04 es un sensor de medida de distancia por ultrasonidos analógico con un rango de medida de entre 2 y 400 cm con una precisión de unos 3mm.

Figura 6.8.- *Sensor HC-SR04*

El sensor tiene 4 pines:

- Alimentación de 5 V.
- Disparador (TRIG).
- Eco (ECHO).
- Ground.

El conexionado del sensor es bastante simple, se tiene que alimentar con 5V y GND, el disparador irá conectado a la placa como una salida de ésta y el eco como una entrada.

En cuanto al funcionamiento, se debe dar un pulso de al menos 10 µs al disparador (también llamado trigger), que activará un ciclo de ultrasonidos a 40 kHz que a su vez activará la señal de eco (también llamada echo), que se desactivará una vez vuelva la señal rebotada de un objeto, pudiendo medir así la distancia a la que éste se encuentra en función del tiempo que haya tardado en volver esta señal.

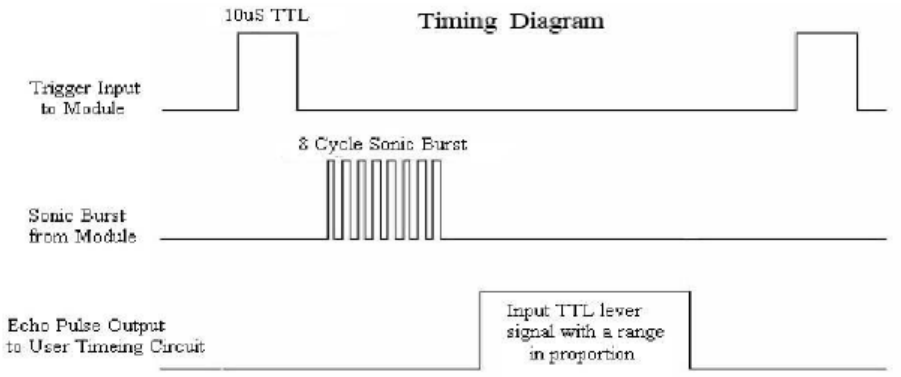


Figura 6.9.- *Diagrama de uso del sensor de distancia por ultrasonidos HC-SR04*

El fabricante recomienda realizar ciclos de medida de más de 60 ms para prevenir que las señales de disparo y eco se solapen.

Para calcular la distancia se medirá el tiempo entre la subida y la bajada de echo y se utilizará la siguiente fórmula para calcular la distancia en cm:

$$\textbf{[6.2]} \quad x[cm] = \frac{\Delta t[\mu s]}{58[\mu s/cm]}$$

6.5. Sensor de distancia por infrarrojos GP2Y0A21YK0F

El sensor GP2Y0A21YK0F es un sensor de medida de distancia por infrarrojos analógico con un rango de medida de entre 10 y 80 cm.

Figura 6.10.- *Sensor GP2Y0A21YK0F*

El sensor tiene 3 pines:

- Alimentación de 5 V.
- Salida (GP2Y).
- Ground.

El conexionado del sensor no da lugar a duda, se alimenta con 5 V y GND y el pin de salida se conectará a un pin de la placa configurado como entrada analógica (I).

En cuanto al funcionamiento es muy simple, una vez conectado a la alimentación, hasta el output de la primera lectura, el pin de salida será inestable y esta estabilidad durará unos 50 ms, a partir de aquí el dispositivo medirá automáticamente cada 40 ms aproximadamente y reflejará la última medición de distancia realizada.

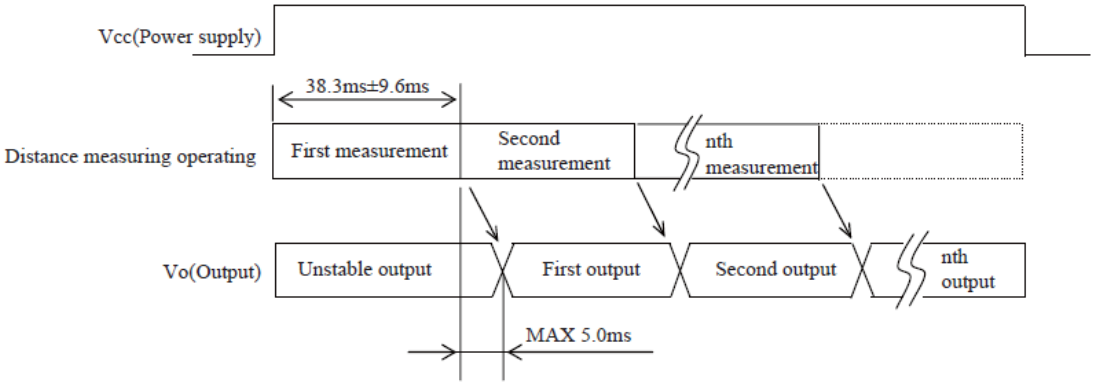


Figura 6.11.- *Diagrama de funcionamiento del sensor GP2Y0A21YK0F*

Para calcular la distancia tenemos dos opciones, utilizar el voltaje de salida o su inverso. Cómo se puede observar en la **Figura 6.12** y la **Figura 6.13** el inverso presenta una mayor linealidad, por lo tanto, será más fácil utilizar este diagrama para calcular la distancia. Haciendo una regresión lineal utilizando los puntos de 1 V con 0,035 cm^{-1} y de 2,3 V con 0,1 cm^{-1} obtenemos una expresión [6.3].

$$\textbf{[6.3]} \quad \frac{1}{x[cm]} = 0{,}05 \cdot V_{Out}[V] - 0{,}015 \rightarrow x[cm] = \frac{1}{0{,}05 \cdot V_{Out}[V] - 0{,}015}$$

Podemos mejorar la función añadiendo una segunda recta para las distancias mayores de 28 cm aproximadamente, para esto utilizaremos el punto anterior de 1 V con 0,035 cm^{-1} y 0,5 V con 1/60 cm^{-1}. Utilizando estos puntos obtenemos la expresión [6.4].

$$\textbf{[6.4]} \quad \frac{1}{x[cm]} = 0.036666667 \cdot V_{Out}[V] - \frac{1}{600} \rightarrow x[cm] = \frac{1}{0.036666667 \cdot V_{Out}[V] - \frac{1}{600}}$$

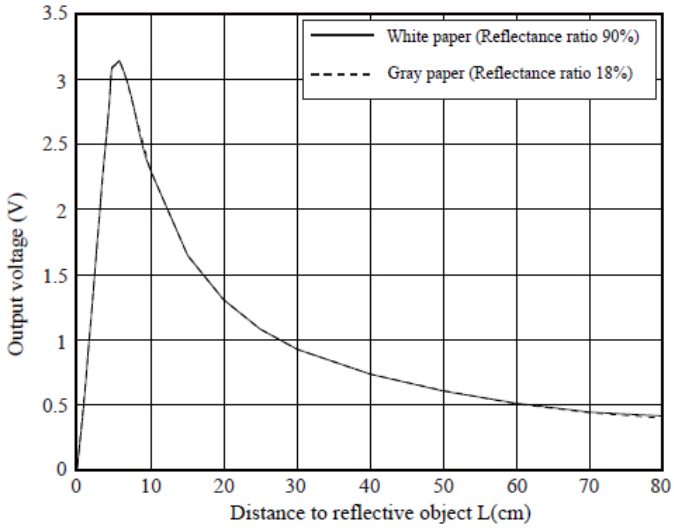


Figura 6.12.- *Curva de respuesta del sensor GP2Y0A21YK0F, voltaje de salida en función de la distancia del objeto*

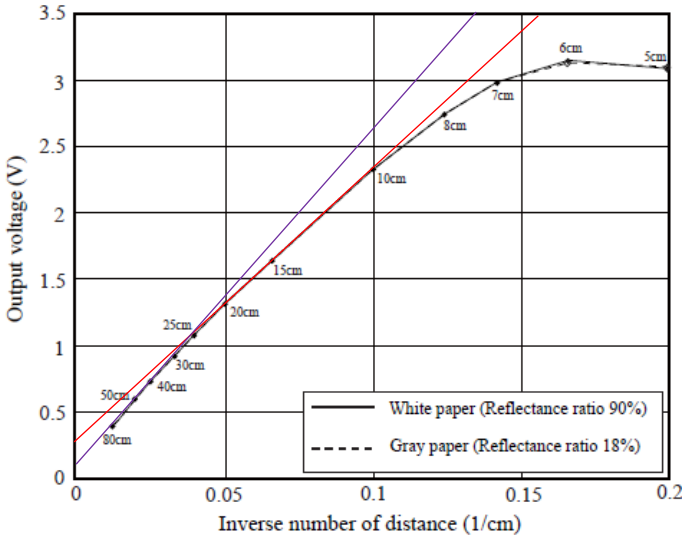


Figura 6.13.- *Curva de respuesta del sensor GP2Y0A21YK0F, voltaje de salida en función de la inversa de la distancia del objeto, en rojo la primera aproximación y en lila la segunda, hechas ambas utilizando regresión lineal con 2 puntos*

6.6. Sensor de temperatura LM35

El sensor LM35 es un sensor de temperatura analógico con un rango de medidas de -55 a 150°C, con una escala lineal de +10 mV/°C con 0 V a 0°C. Como en este proyecto trabajaremos en temperaturas ambiente habituales en interiores podemos utilizar el sensor en escala reducida (sin utilizar voltajes ni temperaturas negativas) por lo que el sensor se alimentará con 5 V y podrá medir de 2 a 150°C.

Figura 6.14.- *Sensor LM35*

El sensor tiene 3 pines:

- Alimentación de 5 V (se puede alimentar desde 4 hasta 20 V).
- Salida (LM35).
- Ground.

El conexionado del sensor en este modo de funcionamiento es muy simple, solo hay que alimentarlo con 5 V y GND y conectar la salida directamente a un pin de entrada de la placa (ver **Figura 6.15**.

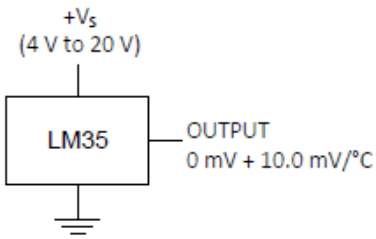


Figura 6.15.- *Diagrama de conexionado LM35*

Para calcular la temperatura simplemente se dividirá el valor de tensión entre los 10mV que se han mencionado anteriormente utilizando la expresión [6.5].

$$\textbf{[6.5]} \quad T[°C] = \frac{V_{Out}[mV]}{10\left[\frac{mV}{°C}\right]}$$

6.7. Sensor de temperatura ADT7420

El sensor ADT7420 es un sensor de temperatura digital con un rango de medidas de -40 a 150°C. Este sensor, contiene un ADC (convertidor analógico-digital) de 16 bits que monitoriza y digitaliza la temperatura con una resolución máxima de 0,0078°C y transmite los datos a través de su puerto I^2C. En el proyecto se utilizará la placa Pmod TMP2 que lleva el sensor incorporado (**Figura 6.16**).

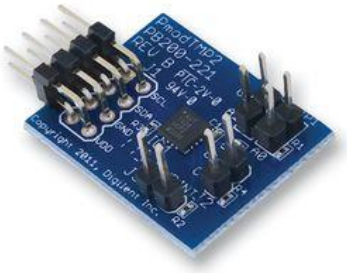


Figura 6.16.- *Placa Pmod TMP2 que incluye el sensor ADT7420*

El sensor tiene 8 pines:

- Alimentación (V_{DD}) puede ser de 2,7 hasta 5,5 V.
- Indicador de sobrecalentamiento típico (CT).
- Indicador de sobrecalentamiento o sobre enfriamiento (INT).
- Primer pin de selección de la dirección del serial del bus I^2C (A0).
- Segundo pin de selección de la dirección del serial del bus I^2C (A1).
- Entrada del reloj del serial I^2C (SCL).
- Entrada/salida de datos del serial I^2C (SDA).
- Ground.

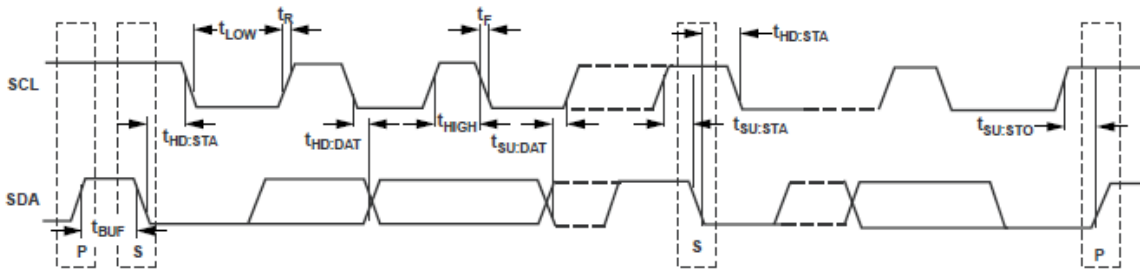


Figura 6.17.- *Diagrama de funcionamiento del sensor ADT7420*

En su modo de utilización normal el ADC del sensor tarda 240 ms en hacer la conversión analógico-digital y funciona de manera continua, una vez acaba la conversión de un valor lo almacena en un registro que tiene integrado, toma el siguiente y empieza a convertirlo. Se puede acceder al registro donde está almacenado el valor de la temperatura a través de su puerto I^2C.

Los pines CT e INT tienen el valor HIGH, como predeterminado, cuando el valor de conversión supera los 64°C el pin INT pasa a LOW y cuando se superan los 147°C hace lo propio el pin CT, aunque estos valores son programables.

Para calcular la temperatura en el modo de conversión a 13 bits hay que tener en cuenta que el bit de mayor peso (MSB) indica el signo de la lectura, 0 para positivo y 1 para negativo; que los 3 bits de menor peso (LSBs) indican los flags de temperatura crítica, alta y baja. Teniendo en cuenta que nos quedan 12 bits para la representación del valor de temperatura (positiva o negativa, el fabricante nos indica que cada unidad representa 0,0625°C por lo que no habría problema para representar las temperaturas límite teóricas. Teniendo en cuenta estos factores, la temperatura se calcularía siguiendo la expresión [6.6] (teniendo en cuenta sólo los 12 bits y suponiéndola positiva). Para temperaturas negativas en cambio utilizaríamos la expresión [6.7].

$$\textbf{[6.6]} \quad T[°C] = ADC_{Output_{12bit}} \cdot 0{,}0625°C = \frac{ADC_{Output_{12bit}}}{16°C^{-1}}$$

$$\textbf{[6.7]} \quad T[°C] = \frac{ADC_{Output_{12bit}} - 2^{13}}{16} = \frac{ADC_{Output_{12bit}} - 8192}{16}$$

Para configurar el sensor hay que modificar el registro configuration que se encuentra en la dirección 0x03. Como se puede ver en la **Figura 6.18** la configuración descrita es la que está por defecto, así que no hará falta configurarlo.

Bit	Default Value	Type	Name	Description
[1:0]	00	R/$\overline{W}$	Fault queue	These two bits set the number of undertemperature/overtemperature faults that can occur before setting the INT and CT pins. This helps to avoid false triggering due to temperature noise. 00 = 1 fault (default). 01 = 2 faults. 10 = 3 faults. 11 = 4 faults.
2	0	R/$\overline{W}$	CT pin polarity	This bit selects the output polarity of the CT pin. 0 = active low. 1 = active high.
3	0	R/$\overline{W}$	INT pin polarity	This bit selects the output polarity of the INT pin. 0 = active low. 1 = active high.
4	0	R/$\overline{W}$	INT/CT mode	This bit selects between comparator mode and interrupt mode. 0 = interrupt mode 1 = comparator mode
[6:5]	00	R/$\overline{W}$	Operation mode	These two bits set the operational mode for the ADT7420. 00 = continuous conversion (default). When one conversion is finished, the ADT7420 starts another. 01 = one shot. Conversion time is typically 240 ms. 10 = 1 SPS mode. Conversion time is typically 60 ms. This operational mode reduces the average current consumption. 11 = shutdown. All circuitry except interface circuitry is powered down.
7	0	R/$\overline{W}$	Resolution	This bit sets up the resolution of the ADC when converting. 0 = 13-bit resolution. Sign bit + 12 bits gives a temperature resolution of 0.0625°C. 1 = 16-bit resolution. Sign bit + 15 bits gives a temperature resolution of 0.0078°C.

Figura 6.18.- *Registro configuraciónADT7420*

6.8. Sensor de temperatura, humedad y presión BME280

El sensor BME280 es un sensor de temperatura, humedad y presión. El sensor tiene una alta precisión, una multitud de funciones y un tamaño muy reducido (2,5 x 2,5 x 0,9 mm). Puede utilizar SPI (ver **Figura 6.19**) e I^2C cosa que facilita su protoipado.

El sensor tiene 8 pines:

- Entrada de selección de chip (CSB).
- Entrada del serial de datos (SDI).
- Entrada del serial del reloj (SCK).
- Salida del serial de datos (SDO)
- Alimentación para la interfaz digital (V_{DDIO}).
- Ground.
- Alimentación analógica V_{DD}.
- Ground

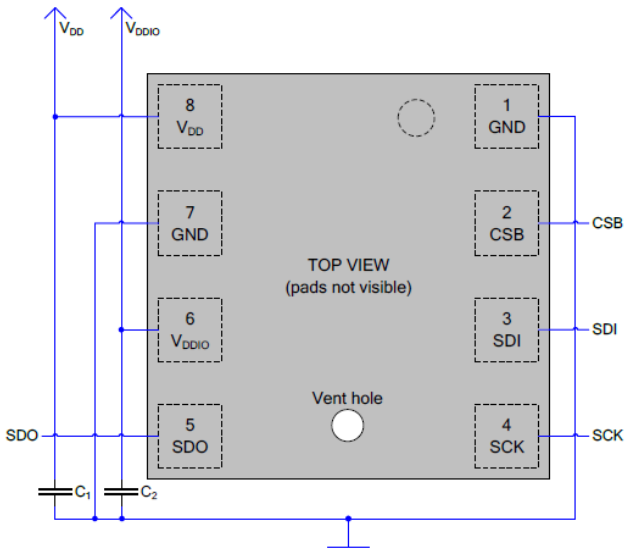


Figura 6.19.- *Diagrama de conexión del sensor BME280 para su funcionamiento por SPI con 4 cables, C_1 y C_2 recomendados de 100nF*

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1…0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88…0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

Figura 6.20.- *Mapa de los registros de memoria del sensorBME280*

Para integrar más fácil y rápidamente el sensor en el prototipo, se ha utilizado una placa que incluye el chip junto con otros elementos para facilitar su integración directamente sin añadir más componentes. La placa escogida es la SEN0236 (ver **Figura 6.21**).

Figura 6.21. *-Placa SEN0236, integra BME280 junto con otros componentes, vistas TOP y BOTTOM*

BME280 puede transmitir por SPI o I^2C, en este caso, se quiere que se comunique por SPI, así que se utilizaran los pines ya preparados para conectar directamente con la nomenclatura siguiente:

- MISO (master in slave out): bus de datos de slave a master.
- SCLK (clock): señal de clock para sincronizar los dispositivos.
- CS (chip select): señal digital para seleccionar el slave.
- MOSI (master out slave in): bus de datos de master a slave.
- GND (Ground): potencial de referencia.
- VCC (power source): alimentación, a 3,3 o 5V.

6.9. Módulo de comunicaciones Bluetooth HC-05

El módulo HC-05, es un módulo de comunicaciones Bluetooth con protocolo de puerto serie (SPP). Este ofrece comunicación Bluetooth 2.0+EDR (velocidad mejorada), tiene una velocidad máxima de transmisión de 3 Mbps y transmite en la banda de 2,4 GHz.

El módulo tiene 6 pines:

- Estado (STATE).
- Puerto de comunicación serie RXD.
- Puerto de comunicación serie TXD.
- Alimentación (3,3 o 5 V).
- Ground.
- KEY.

Figura 6.22.- *Módulo Bluetooth HC-05*

7. Software utilizado

7.1. Programación del microcontrolador

El lenguaje de programación más utilizado por desarrolladores es el C con un 56%, seguido por C++ con un 23% y con un importante crecimiento de Python en los últimos años [6]; por este motivo, el lenguaje de programación utilizado es C.

Una vez elegido el lenguaje de programación para el microcontrolador se debes escoger el IDE o compilador, después de ver las opciones disponibles se han valorado finalmente dos de ellas:

- STM32CubeIDE: es una plataforma de desarrollo C/C++ avanzada con configuración de periféricos, generación de código, compilación de código y funciones de depuración para microcontroladores y microprocesadores STM32. Este software integra la configuración de STM32 y las funcionalidades de creación de proyectos de STM32CubeMX para ofrecer una experiencia de herramienta todo en uno y ahorrar tiempo de instalación y desarrollo. Para trabajar primero se debe seleccionar una MCU o MPU STM32 vacía, o un microcontrolador o microprocesador preconfigurado a partir de la selección de una placa o la selección de un ejemplo, a partir de la selección se crea el proyecto y se genera el código de inicialización. En cualquier momento durante el desarrollo, el usuario puede volver a la inicialización y configuración de los periféricos o middleware y regenerar el código de inicialización sin impacto en el código de usuario.
- Keil µVision5: µVision es una plataforma de desarrollo de software basada en ventanas que combina un editor moderno y robusto con un administrador de proyectos y una herramienta de creación de instalaciones. Integra todas las herramientas necesarias para desarrollar aplicaciones integradas, incluido un compilador C/C++, un ensamblador de macros, un enlazador / localizador y un generador de archivos HEX.

Al valorar ambas opciones se debe tener en cuenta por una parte que el software facilitado y recomendado por el fabricante es STM32CubeIDE y por otra que Kiel es un software ya conocido por el profesorado y parte del estudiantado, ya que es el que actualmente se utiliza para programar los microcontroladores del kit actual.

Una vez valoradas y comparadas ambas opciones se elige utilizar el software del fabricante ya que es realmente intuitivo, muy flexible, facilita la programación incluyendo librerías propias y generando código automáticamente y no requiere de un segundo software como es el caso de Keil con STMCubeMX para poder programar el microcontrolador.

Un punto a tener en cuenta es que con Keil la programación se orienta más a un bajo nivel y no ofrece tantas ayudas y librerías para programar, cosa que favorece el aprendizaje de programación a bajo nivel, que por otra parte, cada vez es menos utilizada por su complejidad; por este motivo es bueno saber que existe esta opción por si se desea trabajar en un futuro de manera diferente pero parece que la tendencia actual del mercado es trabajar cada día a más alto nivel.

***Figura 7.1**.- Logo del software de programación del microcontrolador elegido, STM32 CubeIDE*

7.2. Diseño electrónico y PCB

Para la realización del diseño electrónico y PCB existen en el mercado muchísimas posibilidades, de entre ellas se han escogido finalmente dos variantes

- Altium: es el software de diseño de PCB líder en la industria que combina esquemas, diseño y todo lo que necesita en un entorno para diseñar sin esfuerzo placas de circuito impreso[xviii].
- EasyEDA: es una herramienta de diseño de PCB en línea más fácil y poderosa que permite a los ingenieros electrónicos, educadores, estudiantes, creadores y entusiastas diseñar y compartir sus proyectos. Esta es una herramienta de diseño integrada en el catálogo de componentes LCSC y el servicio JLCPCB PCB que ayuda a los usuarios a ahorrar tiempo para convertir sus ideas en productos reales[xix].

Altium es actualmente el software más utilizado en el diseño de PCB en el mundo, es relativamente simple de usar y contiene una gran cantidad de librerías que simplifican mucho el trabajo para diseñar PCBs y obtener modelos 3D. Altium es un software de pago y no incluye muchos de los componentes que se han detallado anteriormente, ya que vienen con circuitos de acondicionamiento y estes no se integran habitualmente en PCB, por este motivo se deben de crear esquemáticos y footprints para la mayoría de componentes cosa que complica mucho el trabajo y casi descarta la opción de crear un modelo 3D.

Inicialmente el esquemático y PCB se realizaron con Altium, pero después de las recomendaciones del equipo de laboratorio de electrónica de EEBE se decidió cambiar a EasyEDA.

EasyEDA es un software totalmente gratuito y online que no es necesario ni descargar para crear esquemáticos o PCB, los diseños se guardan directamente en la nube siendo así accesibles desde cualquier lugar o terminal, además dispone de una infinidad de librerías creadas y compartidas por otros usuarios. En estas librerías se ha necesitado de la misma manera crear muchos esquemáticos y footprints de componentes que ya están disponibles para cualquier usuario.

Al fin y al cabo, ambos softwares son muy parecidos para la aplicación en cuestión y no resulta un gran cambio pasar de una a otra en términos de usabilidad, aunque una vez utilizada easyEDA se ve claramente que para proyectos personales es una herramienta muy potente y válida que además te ofrece la posibilidad de mandar a producir tu PCB directamente a un fabricante de china y que te llegue en unas 3 semanas.

Figura 7.2.- *Logo del software de diseño electrónico y de PCB elegido, easyEDA*

7.3. Simulación

Para la simulación, se han valorado las opciones de PROTEUS, MultiSIM y OrCAD.

En este caso no ha habido mucha duda ya que PROTEUS es un completo programa que permite diseñar y simular circuitos electrónicos de forma práctica y accesible[xx] mientras que los otros dos valorados son más adecuados para otras aplicaciones como para simular circuitos puramente analógicos.

Este software puede simular cualquier funcionalidad de manera simple con su versión 8, ya que esta incluye el µC por defecto.

En este caso, al contar con una placa de desarrollo no se han realizado muchas simulaciones, aunque sí se ha comprobado que funcione correctamente con la aplicación ejemplar más simple, el encendido de un LED como se puede ver en la **Figura 7.3**.

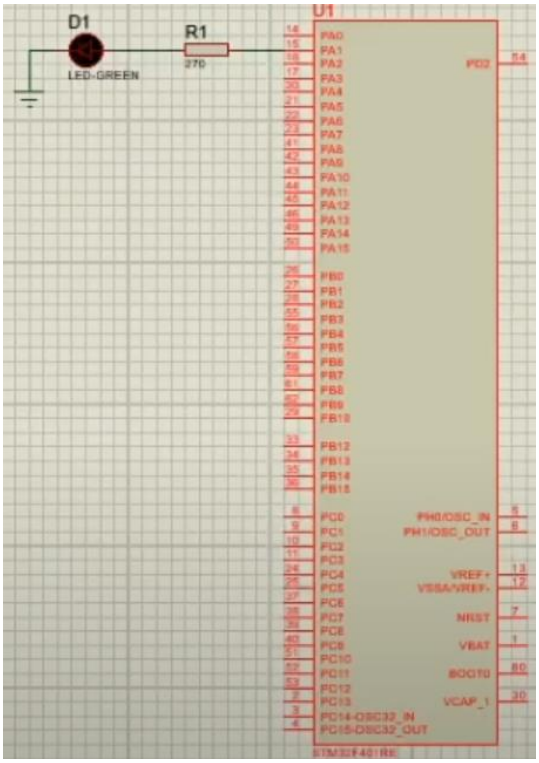


Figura 7.3.- *Simulación ejemplar básica de encendido de LED con PROTEUS*

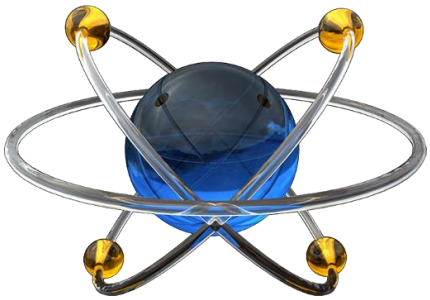


Figura 7.4.- *Logo del software de simulación elegido, PROTEUS*

8. Diseño electrónico y prototipado

Para la realización del trabajo se han seguido una serie de pasos fundamentales para cualquier proyecto de ingeniería.

En primera instancia se han estudiado los conceptos teóricos necesarios para poder aplicar durante el proceso. Una vez estudiados estos conceptos, basándose en un prototipo físico ya existente (apartado 2.), se han escogido un conjunto de periféricos (apartado 6.) que permitan el estudio y puesta en práctica de los conceptos que se trabajan en las asignaturas actuales tanto de Informática Industrial como de algunas optativas y, una vez escogidos, se han adquirido los dispositivos físicos para comprobar su morfología i circuito. La mayoría de circuitos adquiridos ya incluían el acondicionamiento para poder-los prácticamente conectar a la placa de desarrollo, cosa que realmente no ha facilitado el proyecto en sí, ya que se ha tenido que comprobar cada componente uno a uno igualmente, lo que sí ha simplificado significativamente es el diseño electrónico y la complejidad de la PCB. Una vez claros los componentes y sus conexiones, se han vuelto a revisar los recursos y pines que ofrece el micro para poder elegir donde conectar cada componente y si son necesarios elementos extra para la placa, p. ej. pines para conectar una fuente de alimentación externa.

Una vez clara la distribución de pines se debe hacer una distribución de componentes por la placa que permita que los conectores y los sensores pasivos interactivos sean accesibles, que los actuadores sean visibles, que los sensores que midan mismas magnitudes estén agrupados, etc. y realizar la PCB con el diseño escogido. La PCB se realizará de manera manual, prácticamente casera y se detallará el proceso, junto con el posterior taladrado y soldado de componentes terminando con el prototipo final.

8.1. Elección de pines

Un paso fundamental para el prototipado es la correcta elección de los pines para cada componente comprobando la compatibilidad de la funcionalidad y disponibilidad del pin del micro (**Figura 8.1**) y del del componente.

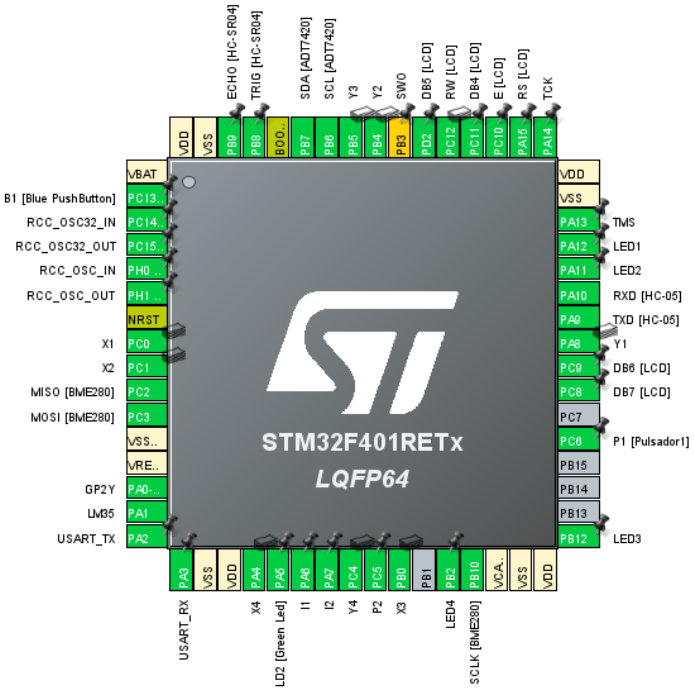


Figura 8.1.- *Distribución de los pines del micro para los componentes del kit en STM32 CubeIDE*

En caso de que varios pines sean compatibles elegir el que facilite al máximo el diseño de la PCB. El resultado de los pines escogidos se puede ver en la **Figura 8.1** y en la **Tabla 8.1**.

Tabla 8.1.- *Conexionado de la PCB de componentes con la placa de desarrollo:*

Nombre		Descripción	Pin componente	Pin NUCLEO F401-RE
1	LD3	LED placa programable	Ánodo	PA5
2	B1	Pulsador placa	NO	PC13
3	P1	Pulsador 1	NO	PC5
4	P2	Pulsador 2	NO	PC6
5	I1	Interruptor 1	-	PA7
6	I2	Interruptor 2	-	PA6
7	LED1	LED 1	Ánodo	PA12
8	LED2	LED 2	Ánodo	PA11
9	LED3	LED 3	Ánodo	PB12
10	LED4	LED 4	Ánodo	PB2
11	X1	Teclado matricial	X1	PC0
12	X2	Teclado matricial	X2	PC1
13	X3	Teclado matricial	X3	PB0
14	X4	Teclado matricial	X4	PA4
15	Y1	Teclado matricial	Y1	PA8
16	Y2	Teclado matricial	Y2	PB4
17	Y3	Teclado matricial	Y3	PB5
18	Y4	Teclado matricial	Y4	PC4
19	DB4	LCD 16x2	DB4 (11)	PC11
20	DB5	LCD 16x2	DB5 (12)	PD2
21	DB6	LCD 16x2	DB6 (13)	PC9
22	DB7	LCD 16x2	DB7 (14)	PC8
23	E	LCD 16x2	E (6)	PC10
24	RW	LCD 16x2	Read/Write (5)	PC12
25	RS	LCD 16x2	RS (4)	PA15
26	TRIG	HC-SR04	TRIGGER	PB8
27	ECHO	HC-SR04	ECHO	PB9
28	GP2Y	GP2Y0A21YK0F	V_O	PA1
29	LM35	LM35	V_{OUT}	PA0
30	SCL	ADT7420	SCL	PB6
31	SDA	ADT7420	SDA	PB7
32	INT	ADT7420	INT	-
33	CSB	ADT7420	CSB	-
34	MISO	BME280	MISO	PC2
35	SCLK	BME280	SCLK	PB10
36	CS	BME280	CS	PC7
37	MOSI	BME280	MOSI	PC3
38	KEY	HC-05	KEY	-
39	TXD	HC-05	TXD	PA10
40	RXD	HC-05	RXD	PA9
41	STAT	HC-05	STATE	-

8.2. Diseño electrónico

El conexionado necesario para cada componente se ha detallado en el apartado 6. , en este apartado, se muestra el diseño electrónico completo realizado con 2 herramientas distintas detalladas en el apartado 7.2, Altium y EasyEDA. La versión realizada en Altium (**Figura 8.2**) se trata de una primera versión que no es la implementada finalmente en la PCB ya que por recomendación del personal de laboratorio de la universidad se acabó utilizando EasyEDA para el diseño final del prototipo.

Como se menciona anteriormente, el diseño electrónico es relativamente simple una vez se han analizado y comprobado todos los componentes uno a uno, especialmente los que incluían placas de acondicionamiento, ya que inicialmente se habían estudiado los componentes aislados con su acondicionamiento necesario y finalmente se tuvo que comprobar las placas individualmente pare ver que incluían y qué no.

Como se puede observar en el esquema realizado con Altium (**Figura 8.2**), el resultado se muestra de manera profesional y estandarizada para componentes comunes como resistencia, LED e interruptores y de manera esquemática y totalmente entendible para los dispositivos más complejos, ya sean componentes en sí, como el LM35, o sean placas con acondicionamiento, como el BME280 con su placa SEN0236.

En cambio, en el esquema realizado con EasyEDA (**Figura 8.3**), se muestra un resultado menos formal, con mucha variedad de colores para diferentes partes, p.ej. cables en verde y componentes discretos en rojo. Este esquema es igualmente funcional y totalmente comprensible.

Es importante remarcar que, al utilizar componentes con placas, mayoritariamente no se encontraban en las librerías de los editores así que tuvieron que ser creados de cero a partir del dispositivo físico y sus hojas de datos.

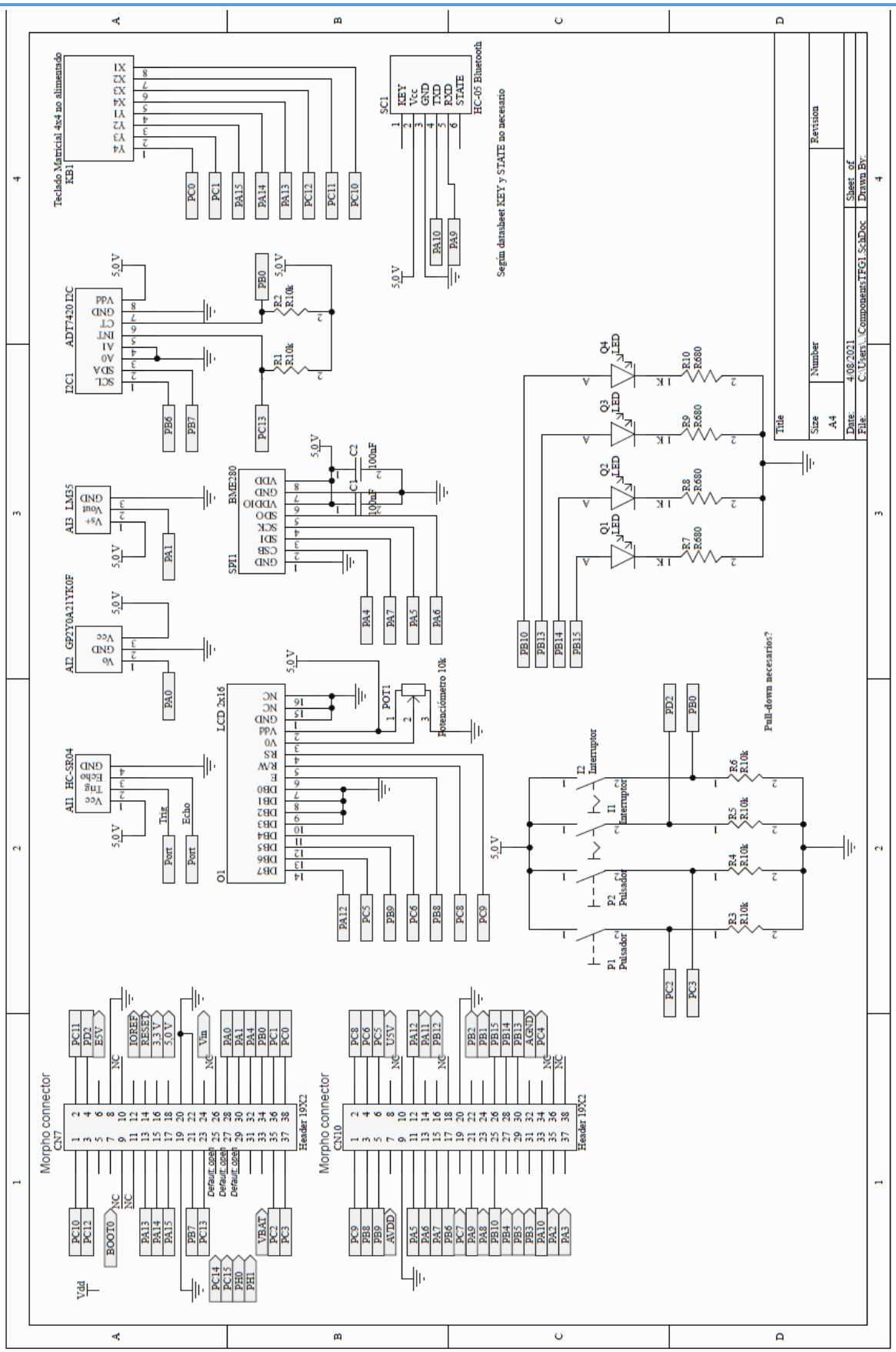


Figura 8.2.- *Esquema del diseño electrónico de los periféricos realizado con Altium*

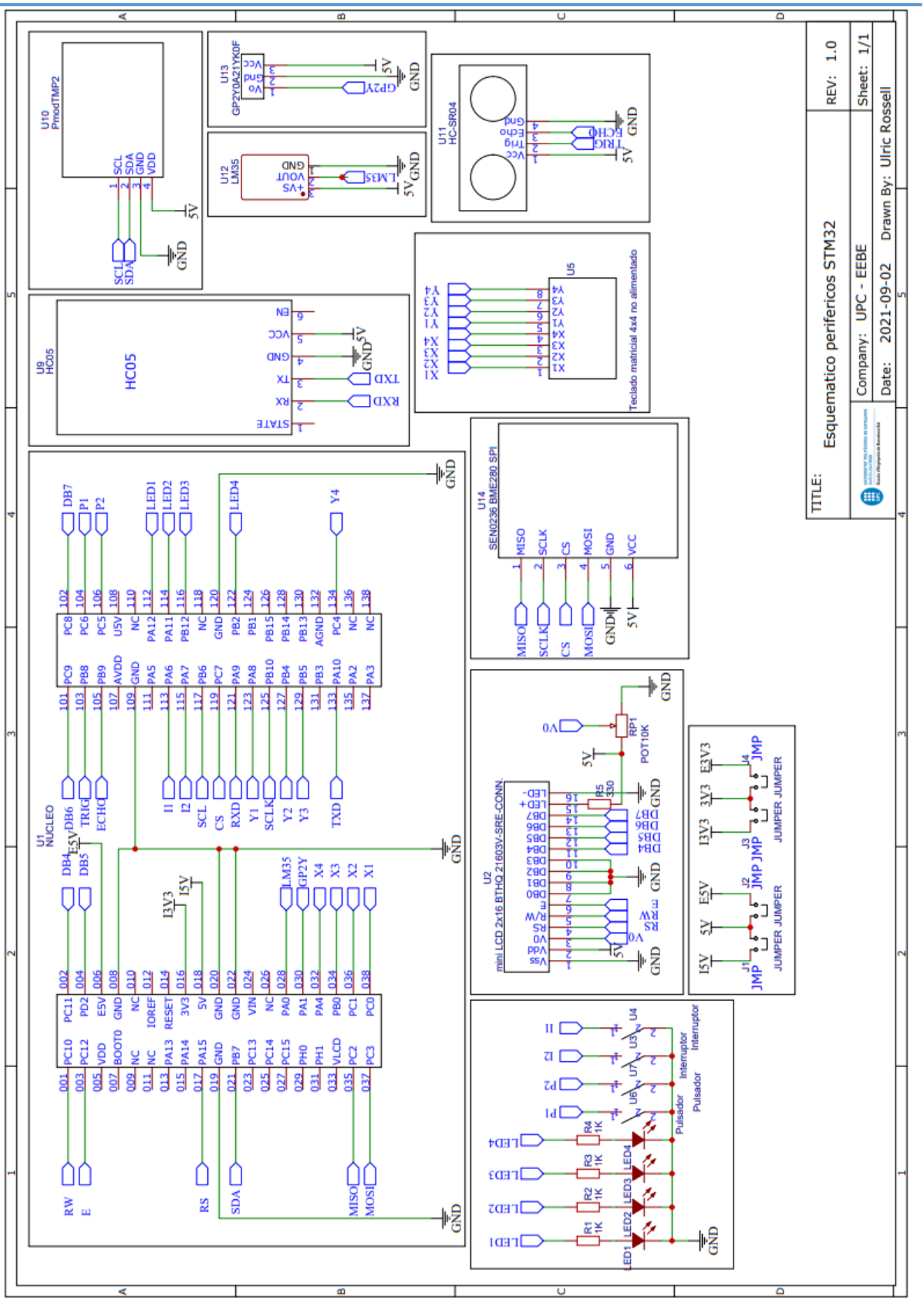


Figura 8.3.- *Esquema del diseño electrónico de los periféricos realizado con EasyEDA*

8.3. PCB

La PCB ha sido realizada con una placa de fibra de vidrio fotosensible positiva de 100x160 de doble cara ya que es la que había disponible en el laboratorio de electrónica de la universidad y se adaptaba a las necesidades del proyecto.

Figura 8.4.- *Ejemplo de una placa como la utilizada para el prototipo*

A partir del diseño electrónico comentado en el apartado 8.2, con la ayuda del software de edición de EasyEDA y las directrices del personal de laboratorio se diseña la PCB teniendo en cuenta que:

- Los conectores deberían situarse en los bordes exteriores de la placa.
- Los actuadores (p.ej. LED) tienen que ser visibles para el usuario.
- Los sensores interactivos con el usuario como pulsadores tienen que ser accesibles.
- Los componentes que realizan mediciones de la misma magnitud tiene sentido que estén agrupados.
- Las pistas deben ser lo más gruesas y cortas posibles y estar alejadas la una de la otra.
- Añadir un plano de masa, siempre a una distancia prudencial de las pistas, ayuda a gastar menos ácidos para la realización de la placa.
- Hacer los agujeros pasantes de un diámetro muy pequeño, ya que solo servirán como guía para la broca utilizada para hacerlos.

Siguiendo estas directrices se diseñó la PCB. En la **Figura 8.6** se puede observar la cara TOP del diseño final de la PCB, en la **Figura 8.7** la cara BOTTOM, en la **Figura 8.8** la distribución final de los componentes y en la **Figura 8.9** las tres anteriores junto con los agujeros pasantes, superpuestas.

Es importante comentar que para realizar el montaje tipo shield se han añadido unos conectores soldados a la placa para poder montar y desmontar la placa fácilmente. También se han añadido unos pines con jumpers para alimentar los periféricos desde la placa o externamente, así como para alimentar a todo el conjunto con una fuente de alimentación externa.

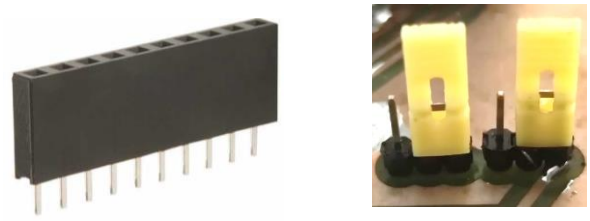


Figura 8.5.- *Conectores hembra para shield (izquierda) y pines con jumper para alimentar a los periféricos y/o la placa (derecha)*

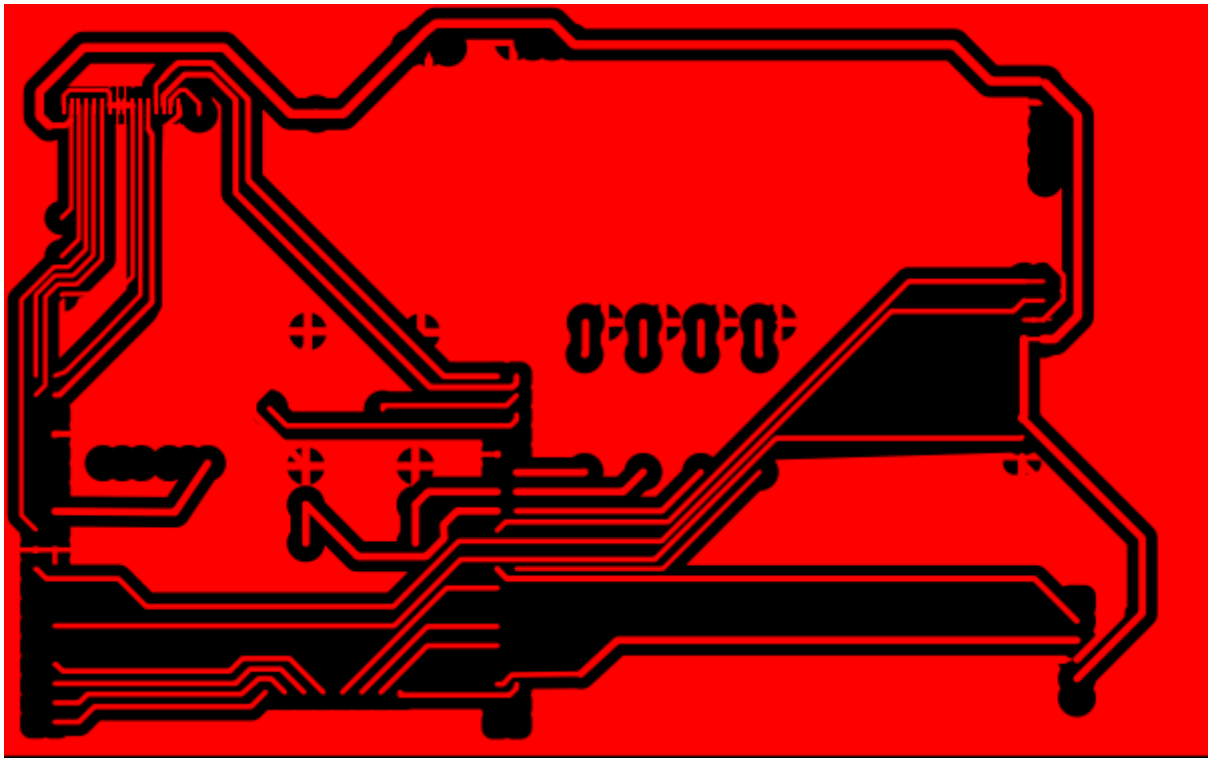


Figura 8.6.- *Pistas TOP PCB*

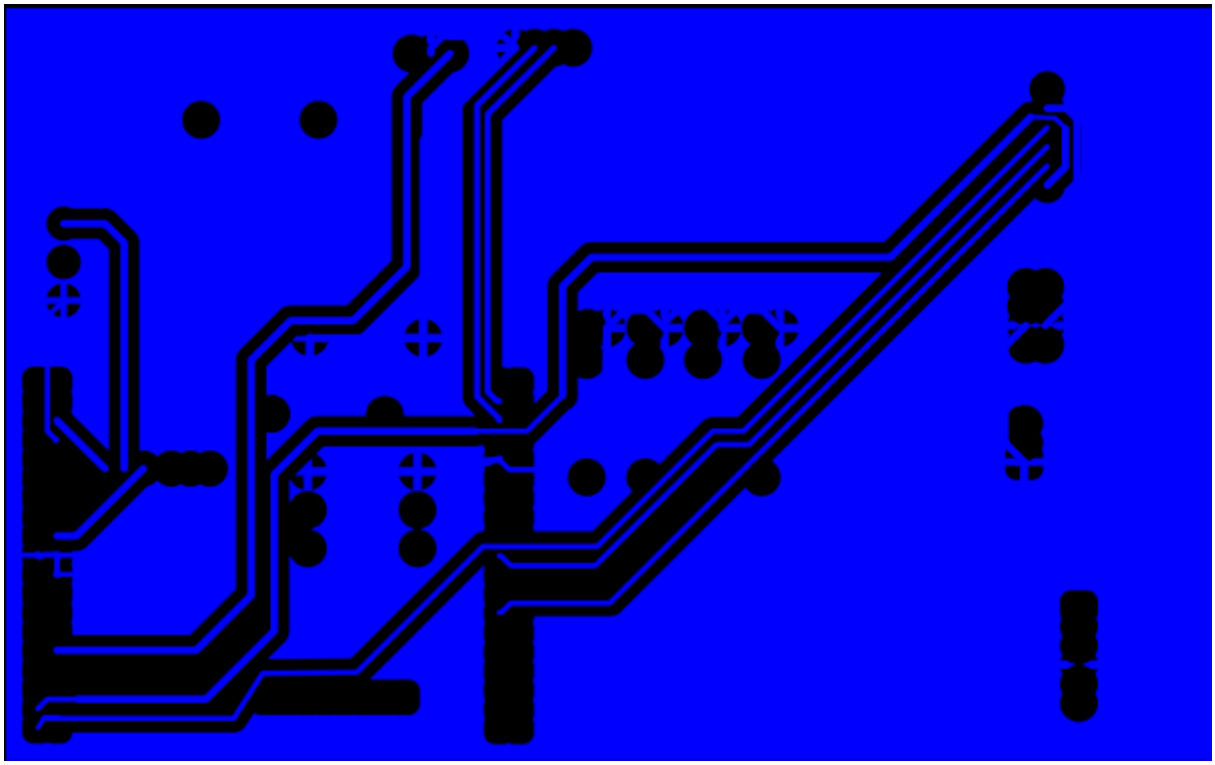


Figura 8.7.- *Pistas BOTTOM PCB*

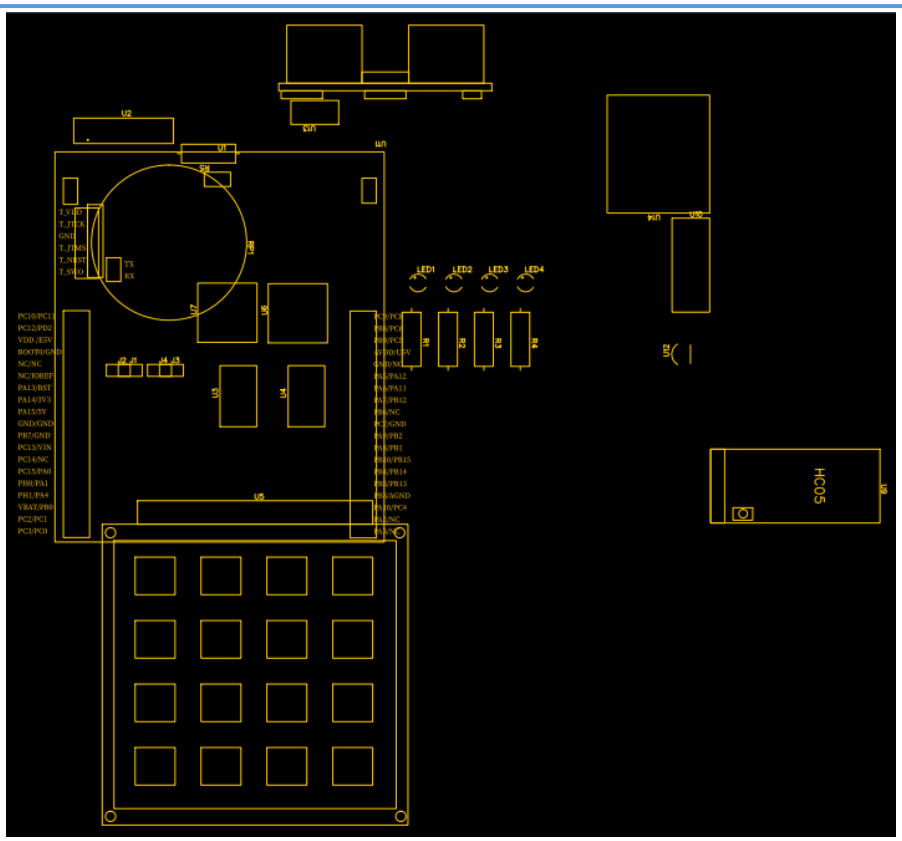


Figura 8.8.-*Distribución de componentes PCB*

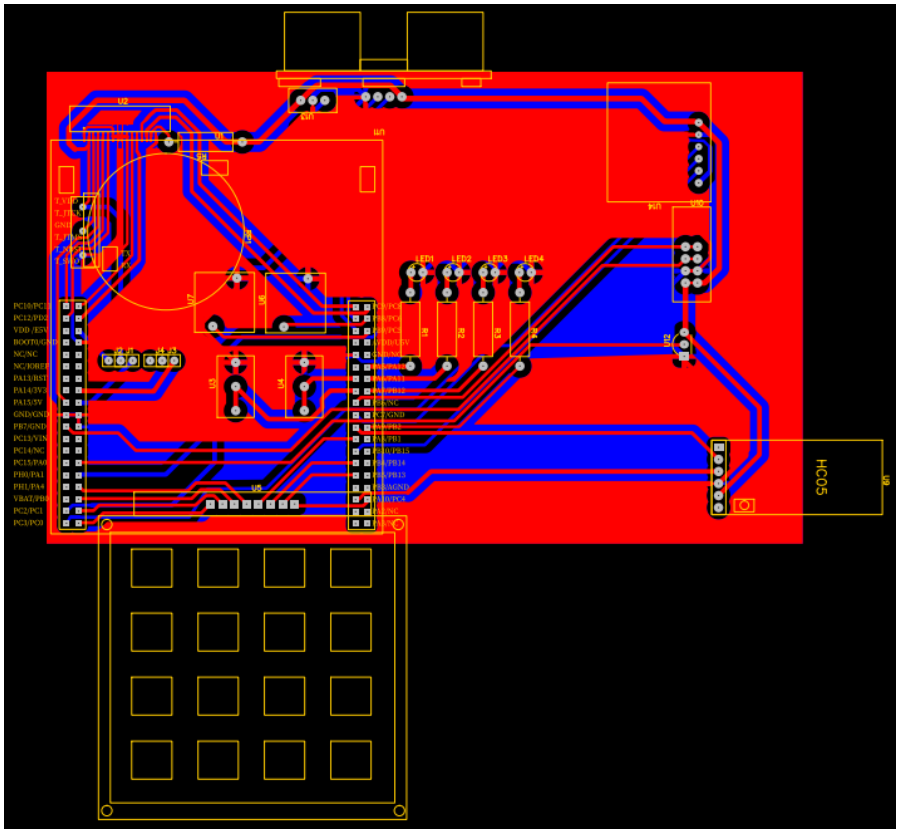


Figura 8.9.- *Pistas TOP y BOTTOM, agujeros pasantes y componentes superpuestos PCB*

Una vez realizado el diseño de la PCB, se imprime el fotolito, en este caso dos copias para cada cara asegurando una máxima opacidad que resulta en un mejor resultado. Las directrices seguidas para la impresión y fijación del fotolito son las siguientes:

- Imprimir el fotolito con una impresión de calidad (máxima opacidad) en un papel transparente, imprimiendo en la copistería de la universidad ni con 3 copias por cara se llegó a tener una opacidad suficiente y se tuvo que recurrir a una copistería especializada.
- Imprimir el fotolito por la cara correcta para que la impresión quede en el lado de la placa, para evitar que por el grueso del "papel" se filtre luz que puede empeorar el resultado final (puede conllevar tener que imprimir con efecto espejo).
- Una vez impresos los fotolitos, pegarlos de manera que queden perfectamente alineados, primero cada capa entre sí y después la top y la bottom alineando perfectamente los agujeros pasantes (recordar que la parte opaca impresa tiene que estar en contacto con la placa).

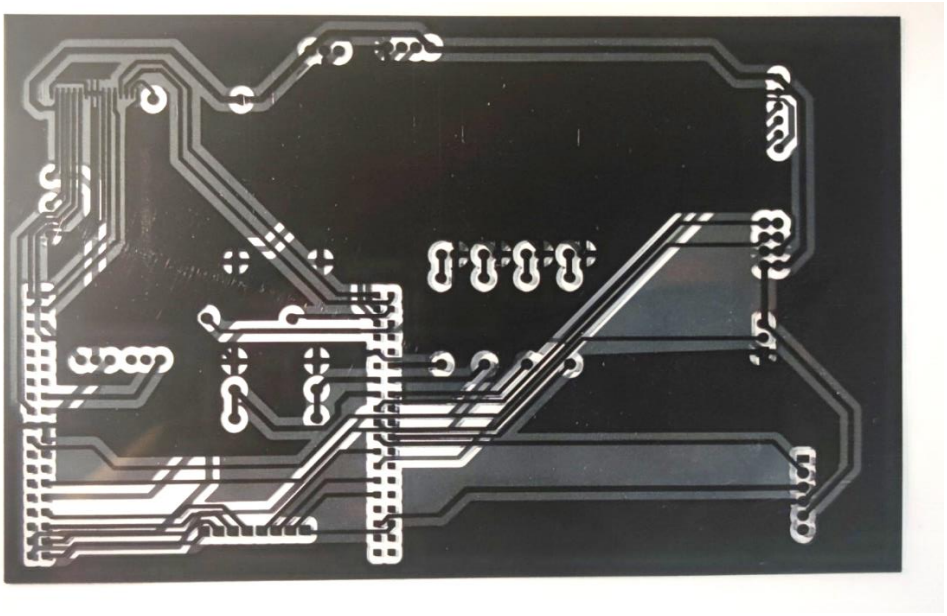


Figura 8.10.- *Fotografía del fotolito final utilizado para el prototipo*

El siguiente paso consiste en colocar la placa entre los fotolitos y ponerla en la insoladora para reblandecer las zonas de la emulsión de la palca a las que les da la luz (**Figura 8.11**), el tiempo de insolado habitual ronda los 2,5-3 minutos.

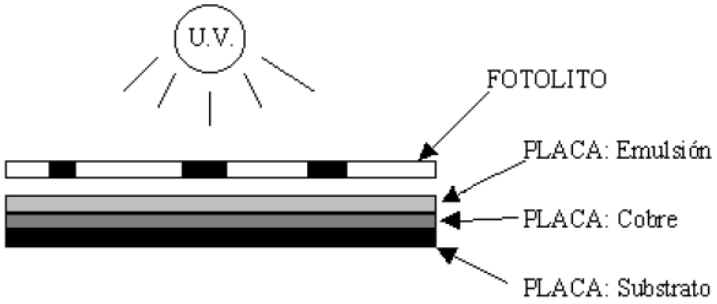


Figura 8.11.- *Representación esquemática del proceso de insolación en placas fotosensibles positivas*[7]

A continuación, se pasa al revelado, el objetivo de este procedimiento es eliminar la resina reblandecida y dar consistencia a la que ha sido protegida por las partes opacas del fotolito. Para hacer esto se utiliza una mezcla de sosa cáustica muy diluida en su justa medida que tarda entre 1 y 2 minutos en realizar el proceso por completo (**Figura 8.12**). Es necesario limpiar la placa posteriormente con abundante agua y secarla para eliminar los restos de líquido revelador.

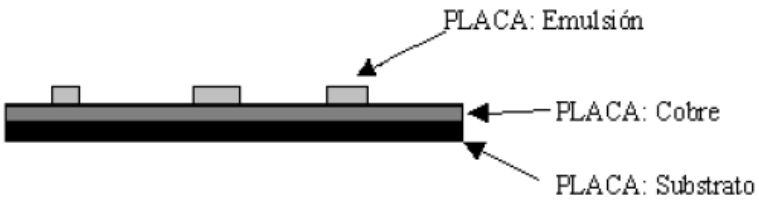


Figura 8.12.- *Representación esquemática del resultado del proceso de revelado en una placa emulsionada positiva* [7]

El siguiente paso a realizar es el atacado del cobre, para el que se utiliza un atacador rápido, una mezcla de ¼ ácido clorhídrico con ¼ de agua oxigenada de 110 VOL y ½ de agua. Este proceso dura escasos minutos, pero se trata de una reacción exotérmica (que libera calor) y desprende gases irritantes, por ese motivo es obligatorio el uso de extractores de aire durante el atacado con este producto. También hay que tener en cuenta que la disolución es corrosiva y que en consecuencia hay que utilizar los EPI (guantes, gafas, …) para protegerse de posibles salpicaduras. Una vez finalizado el proceso se debe aclarar la placa con abundante agua y secarla (hay que tener en cuenta que los desechos resultantes de este proceso son altamente contaminantes y se deben tratar como tal).

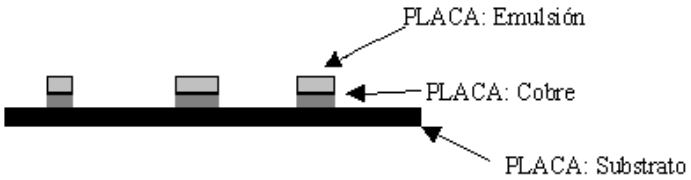


Figura 8.13.- *Representación esquemática del resultado del proceso de atacado al cobre en una placa emulsionada positiva*[7]

Una vez llegados a este paso ya tenemos una placa con las pistas dibujadas sobre el sustrato. El cobre aún no es visible, ya que sigue protegido por la resina endurecida en la insolación. Esta resina protege al cobre, ya que se oxida muy rápida y fácilmente y solo se debe retirar justo antes de realizar los siguientes pasos, taladrar los agujeros pasantes y soldar los componentes.

Justo antes de empezar el taladrado, se limpia la resina con acetona o quita esmalte y un paño que no deje residuos para dejar las pistas de cobre al aire (dejándola como la de la **Figura 8.14**)[7].

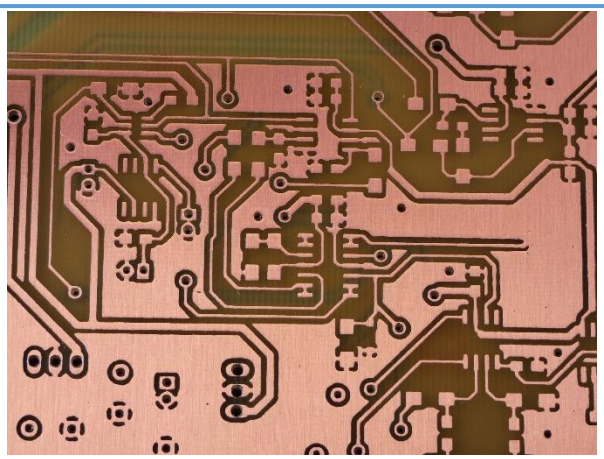


Figura 8.14.- *Placa PCB casera con el cobre visible*

Para el taladro se utiliza un taladro vertical fijo (**Figura 8.15**) con las brocas adecuadas para cada componente y se hacen uno a uno los agujeros. Al haber realizado los PADs tan reducidos, muchos de ellos saltan (especialmente los que no tienen ninguna pista conectada), fallo que no se considera crítico para el resultado.

Figura 8.15.- *Taladro vertical (Dremel) utilizada para taladrar la PCB*

Una vez hechos los agujeros se limpia otra vez la palca y se pasa a realizar el soldado de los componentes y conectores. En este caso se han priorizado las soldaduras firmes hasta con exceso de estaño a las finas, ya que se quieren fortalecer las uniones de los PADs y pistas débiles.

Una vez realizada la soldadura, es importante hacer una comprobación de continuidad de todos las pines de cada componente con su pista y su destino, ya que al tratarse de una placa realizada y soldada a mano por un estudiante inexperto en este ámbito pueden mostrarse falsas conexiones o cortocircuitos "invisibles", en este caso, dos pistas que estaban muy juntas pero parecían ópticamente totalmente separadas mostraban continuidad entre ellas, así que con la ayuda de un destornillador plano fino se resiguió las pistas poco a poco vigilado no dañar ninguna de ellas hasta hacer desaparezer el puente.

8.4. Prototipo final

Después de realizar todos los pasos descritos, en la **Figura 8.16** se puede observar el resultado obtenido, una PCB en forma de shield para la placa de desarrollo NUCLEO-STM32F401RE que cumple todas las especificaciones mencionadas y que ofrece una funcionalidad completa de todos sus componentes.

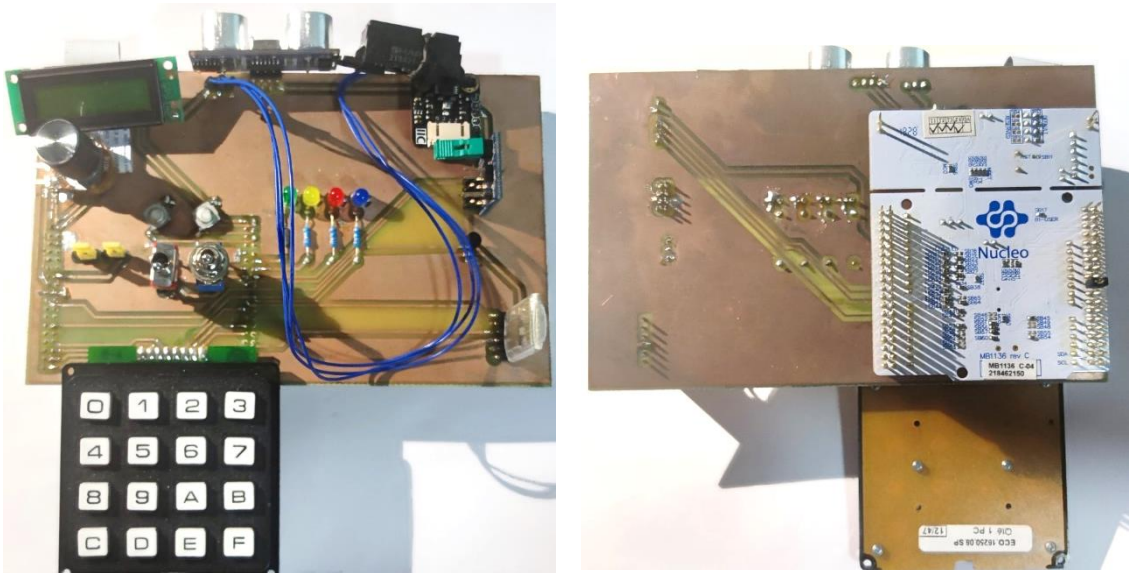


***Figura 8.16.**-Prototipo finalizado*

9. Programación

9.1. Inicialización de un nuevo proyecto

Antes de pasar a la programación, es importante entender la configuración que hay que realizar al empezar un nuevo proyecto. Se ha utilizado la versión 1.5.1 de STM32CubeIDE para toda la configuración.

Una vez abierto el STM32CubeIDE se tiene que crear un nuevo proyecto STM32 (ver **Figura 9.1**).

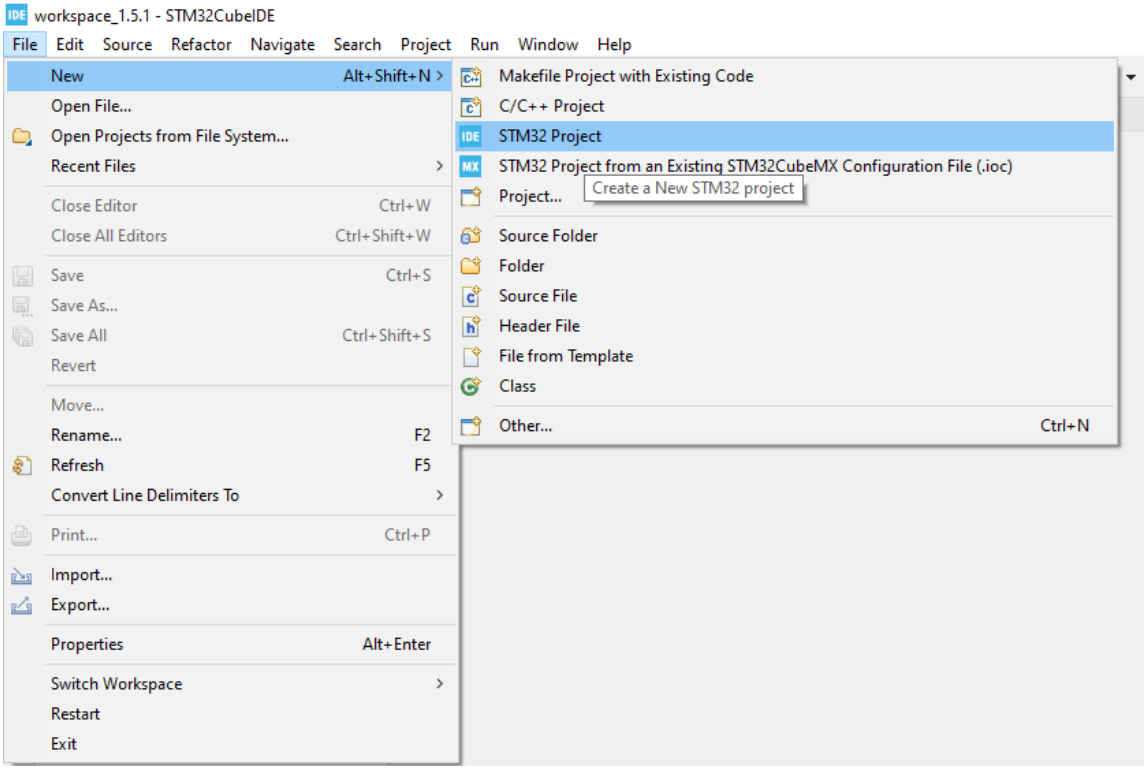


***Figura 9.1.**- Captura de pantalla de STM32CubeIDE, crear nuevo proyecto STM32*

Para crear el nuevo proyecto, una vez descargados los archivos necesarios, sale un pop-up en el que se puede seleccionar el microcontrolador o placa de desarrollo (ver **Figura 9.2**). En este caso se debe seleccionar la pestaña de placas y buscar la placa en cuestión, la NUCLEO-F401RE, se selecciona y se le da a next.

Una vez seleccionada la placa, se pasa a la pantalla de configuración del proyecto (**Figura 9.3**). Hay que insertar el nombre del proyecto, p.ej. práctica 0, se selecciona el directorio o se deja el configurado por defecto. Otras opciones configurables son el lenguaje de programación, en este caso en C, el binary type, en este caso ejecutable y el tipo de proyecto, STM32Cube. Se dejan el resto de configuraciones por defecto y se clica finish.

Una vez finalizada la configuración del proyecto aparece un pop-up preguntando si se quiere configurar todos los periféricos por defecto, se le da que sí para configurar los periféricos conectados al micro en la placa.

En la **Figura 9.4** se puede ver la configuración predeterminada del micro en la placa NUCLEO-F401RE.

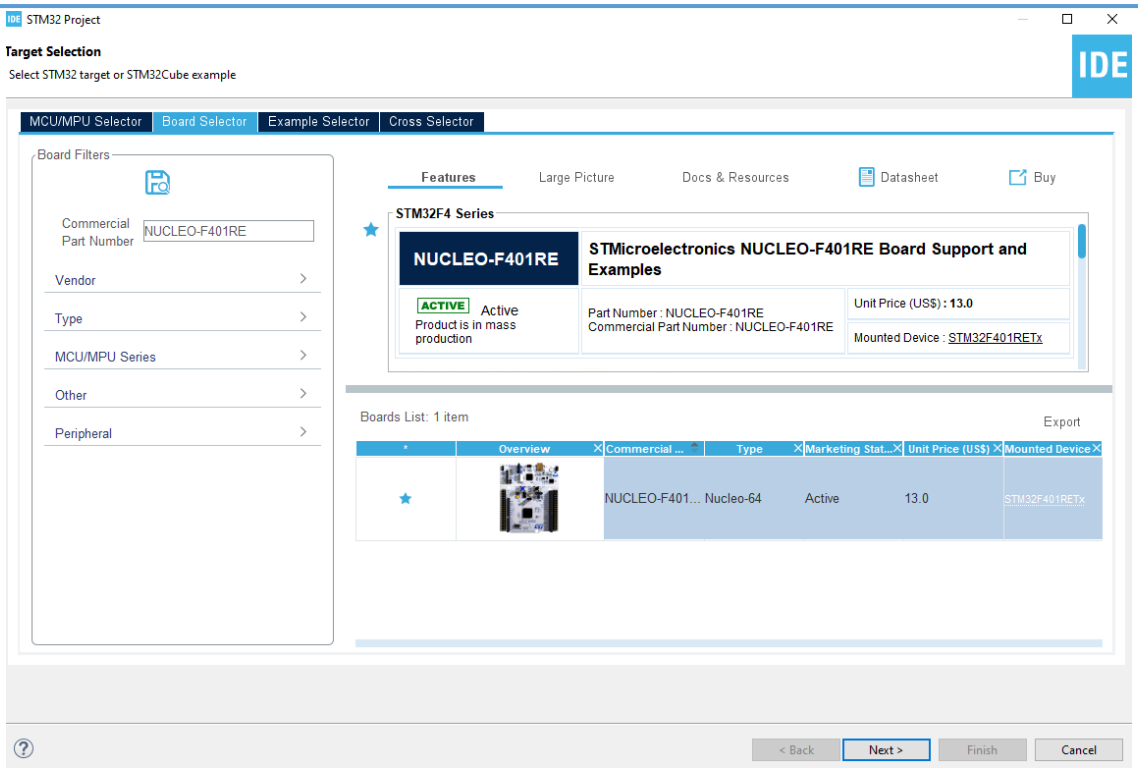


Figura 9.2.- *Captura de pantalla de STM32CubeIDE, elección de microcontrolador/placa para un nuevo proyecto*

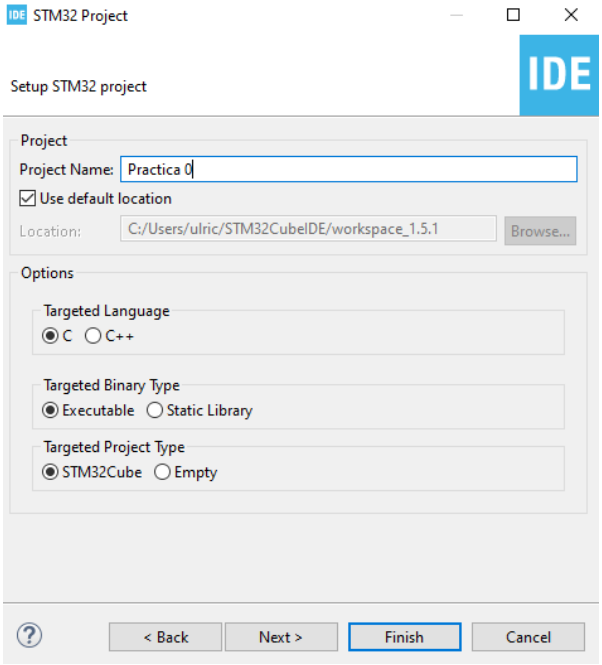


Figura 9.3.- *Captura de pantalla de STM32CubeIDE, configuración de un nuevo proyecto*

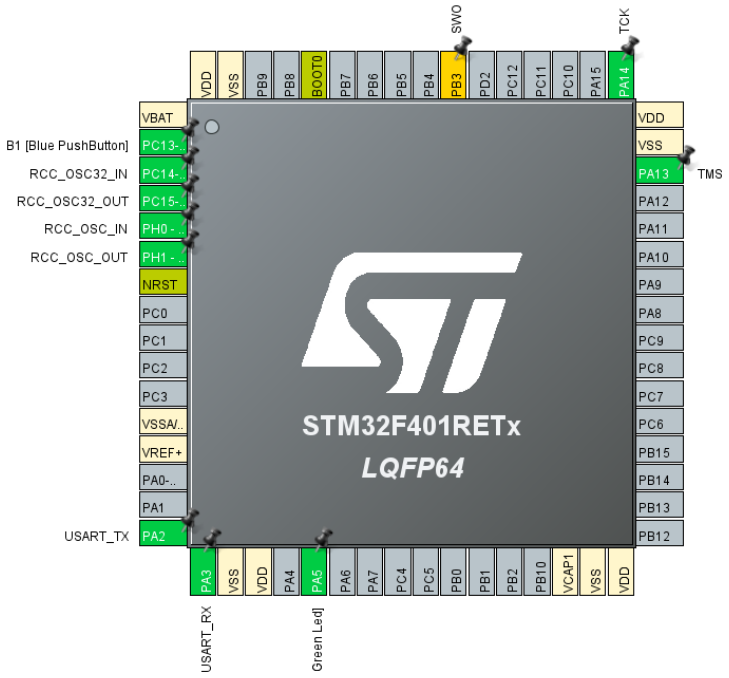


Figura 9.4.- *Captura de pantalla de STM32CubeIDE, configuración por defecto del microcontrolador en NUCLEO-F401RE*

Una vez realizados estos pasos ya se tiene el proyecto listo para configurar los diferentes periféricos que se deseen, así como los diferentes recursos del micro.

En caso de querer pasar a la pantalla de programación solo hay que pulsar el botón de crear código a partir de la configuración  .

Para añadir el código solo hay que entrar en el archivo que se quiera modificar y escribir el código en el espacio que se encuentra en el archivo especialmente dedicado para el código de usuario. En caso de que se incluya código en algún sitio que no esté especialmente dedicado para este fin y se quiera volver a generar código automáticamente, éste desaparecerá (en la **Figura 9.5** se muestra un ejemplo de cómo está delimitado el espacio destinado a la escritura de código por parte del usuario).

```
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
```

Figura 9.5.- *Delimitación del espacio destinado a el código del usuario*

9.2. Programas desarrollados

En este apartado se mostrarán y comentarán diferentes códigos realizados para el prototipo. Los códigos creados se harán para trabajar diferentes aspectos como en las prácticas originales de la asignatura Informática Industrial.

Actualmente, en la asignatura de informática industrial hay 7 prácticas que tratan los siguientes temas:

- Práctica 1: introducción al kit y al entorno de desarrollo[8].
  - Conocer las características, arquitectura y operativa del kit.
  - Comprender el entorno.
  - Identificar y acceder a los registros y al mapa de memoria del micro.
- Práctica 2: programación en ensamblador[9].

  - Instrucciones de transferencia para accesos de memoria, registros y puertos.
  - Instrucciones lógica y boleanas.
  - Instrucciones de control de flujo para implementar condicionales y bucles.
  - Programación de rutinas.
  - Directivas.
- Práctica 3: teclados y visualizadores[10].
  - Control de periféricos de entrada/salida típicos.
- Práctica 4: temporizadores e interrupciones[11].
  - Trabajar el uso del mecanismo de interrupciones.
  - Trabajar el uso de temporizadores.
- Práctica 5: programación en C[12].
  - Tipos de datos y ubicación de variables.
  - Acceso a recursos del micro.
  - Operaciones aritméticas y booleanas.
  - Control de flujo.
  - Programación de funciones.
- Práctica 6: comunicaciones serie[13].
  - Utilización del canal serie del micro.
- Práctica 7: conversión ADC[14].
  - Control de convertidores ADC.

Adaptando las prácticas que se utilizan actualmente al nuevo kit y sus características, se han creado códigos de ejemplo tratando diferentes temas que, adaptándolos convenientemente, podrían servir como prácticas para trabajar con este nuevo kit. Los códigos realizados tratan en concreto los siguientes temas:

- Código de ejemplo 1:
  - Librerías HAL.
  - Configurar puertos.
  - Crear y asignar variables.
  - Lectura de entradas digitales.
  - Escritura de salidas digitales.
  - Uso de la función HAL_Delay.
  - Programación en c:
    - Uso de bucles for.
    - Uso de bucles while
    - Uso de instrucciones condicionales if.
- Código de ejemplo 2:
  - Crear e importar archivos .h
  - Crear e importar archivos .c
  - Control de teclado.
  - Control de LCD.
- Código de ejemplo 3:
  - Uso de temporizadores.
  - Uso de interrupciones.
- Código de ejemplo 4:
  - Utilización de ADC.
  - Debugger (depurador de código).

9.2.1. Código de ejemplo 1

Para este código se utilizarán los 2 pulsadores, los 2 interruptores y los 4 LED. El objetivo de este código es mostrar los siguientes aspectos:

- Librerías HAL.
- Configurar puertos.
- Crear y asignar variables.
- Lectura de entradas digitales.
- Escritura de salidas digitales.
- Programación en c:
    - Uso de bucles for.
    - Uso de bucles while
    - Uso de instrucciones condicionales if.

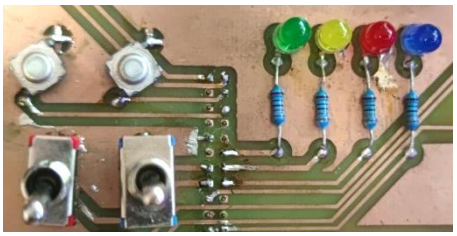


Figura 9.6.- *Periféricos utilizados para el código de ejemplo 1*

Primero de todo se tienen que configurar, los pulsadores e interruptores como solo están conectados a tierra en su otro terminal se tienen que configurar con pull-ups internos. Los LED se pueden dejar por defecto.

Esta práctica seguiría la línea de las actuales prácticas 1[8] y 2[9] . La práctica 5[12] de laboratorio también estaría incluida, en esta y siguientes, ya que la programación se realiza en c. El objetivo es crear un programa totalmente secuencial que permita entender el funcionamiento de lectura y escritura de puertos digitales utilizando delays, bucles y condicionales, todo progrmado en C y utilizando librerías HAL.

El código realiza diferentes "rutinas" mostradas en los LED de manera totalmente secuencial en función de si se activa un pulsador o un interruptor.

9.2.2. Código de ejemplo 2

Para este código se utilizarán la LCD y el teclado matricial. El objetivo de este código es mostrar los siguientes aspectos:

- Programación en c:
    - Crear e importar archivos .h
    - Crear e importar archivos .c
- Control del teclado.
- Control de LCD.

Figura 9.7.- *Periféricos utilizados para el código de ejemplo 2*

Primero de todo se tiene que configurar la LCD, el teclado matricial se configurará con código ya que los puertos tendrán que ir cambiando de entrada a salida.

Esta práctica seguiría la línea de la actual práctica 3[10] y se añadirían los apartados de crear e importar archivos .h y .c. El objetivo es aprender a utilizar estos periféricos de entrada/salida típicos, crear archivos .h de configuración y .c de código (funciones) e importar archivos ya creados.

El código creado, inicializa la LCD con el texto "Codigo de ejmplo 2" y lee continuamente el teclado matricial, al detectar que se ha pulsado una tecla muestra la tecla pulsada en la LCD.

9.2.3. Código de ejemplo 3

Para este código se utilizarán los 2 pulsadores, los 2 interruptores y los 4 LED. El objetivo de este código es mostrar los siguientes aspectos:

- Uso de temporizadores
- Uso de interrupciones

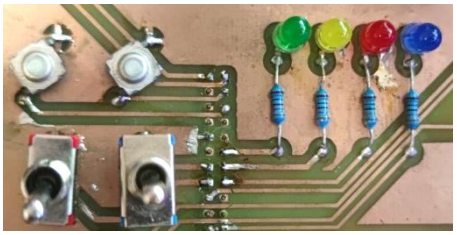


Figura 9.8.- *Periféricos utilizados para el código de ejemplo 3*

Primero de todo se tienen que configurar los pulsadores, interruptores y LED de la misma manera que en el apartado 9.2.1.

Esta práctica seguiría la línea de la actual práctica 4[11]. El objetivo es aprender a utilizar los temporizadores y las interrupciones.

El código creado muestra el funcionamiento de los pulsadores y el interruptor I1 por interrupciones mientras se ejecuta un código con el interruptor I2. P1 enciende el LED2, P2 lo apaga y I1 apaga el LED 3 y enciende el LED 4. La rutina principal de simplemente desactiva los LED1, LED2 y LED 4 y activa el LED3, dejando un delay de 2,5 s después para poder comprobar que las interrupciones funcionan correctamente. Además, se programa un temporizador que cuando desborda cada medio segundo hace cambiar de estado el LED 1.

9.2.4. Código de ejemplo 4

Para este código se utilizarán el sensor de temperatura LM35, el sensor de distancia de infrarrojos GP2Y0A21YK0F y el sensor de temperatura integrado en la placa NUCLEO. El objetivo de este código es mostrar los siguientes aspectos:

- Utilización de ADC.
- Ventana de depuración (debug) de programa.

Figura 9.9.-*Periféricos utilizados para el código de ejemplo 4*

En este caso hay que configurar los tres puertos del mismo ADC a los que están conectados los sensores, ajustando los parámetros según convenga. Para trabajar el debugger, los valores obtenidos se leerán directamente desde este para ver como varían las variables en tiempo real.

Esta práctica seguiría la línea de la actual práctica 7[14]. El objetivo es aprender a utilizar los ADC.

El código implementado permite la lectura en tiempo real de la temperatura con dos sensores distintos, el LM35 y el interno del micro; también permite hacer una lectura de distancia con el GP2Y0A21YK0F. La lectura se realiza observando las variables temp, lm35 y gp2y en modo debugger.

9.3. Protocolos de comunicación

Como inicialmente no se ha creado código para los periféricos que se comunican con diferentes protocolos de comunicación se ha hecho un resumen de estos protocolos.

9.3.1. I^2C

I^2C (circuito inter-integrado) es un protocolo de comunicación para ser utilizado en bus desarrollado por Philips en los años 80. Es un sistema de comunicaciones relativamente lento (comparado con la tecnología actual), pero es suficientemente rápido para ser utilizado con sensores o displays.

Este tipo de comunicación es muy fácil de utilizar desde la perspectiva de hardware, ya que solo cuenta con dos pines, SDA y SCL. I2C es una línea bidireccional, es decir, los datos pueden ir en ambos sentidos, además es un half-duplex, es decir, solo se puede comunicar un dispositivo a la vez. Este bus tiene una jerarquía, tiene un master (maestro) y uno o más slaves (esclavos). Como se puede observar en la **Figura 9.10** hay un pull-up en cada línea por lo que si no se comunica nadie la línea estará en alto.

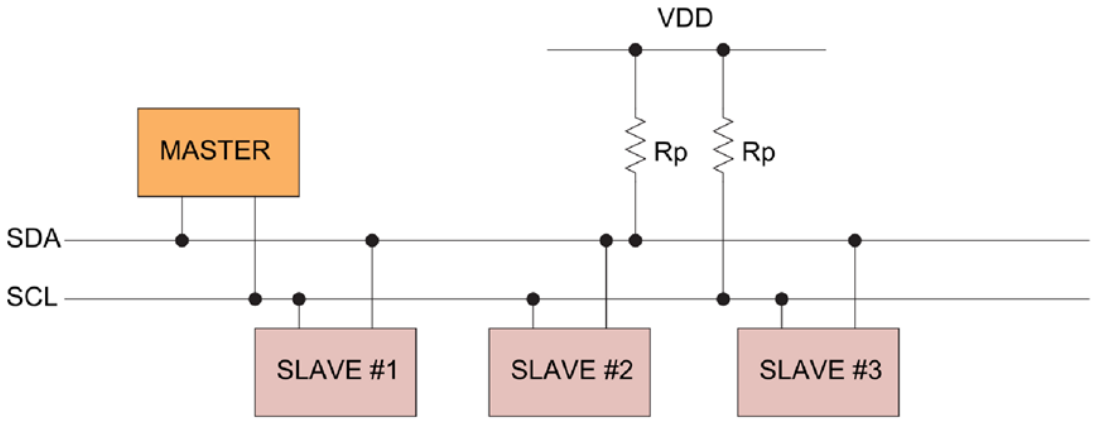


Figura 9.10.- *Diagrama generalizado de conexión* I^2C [xiv]

El master es el único elemento del bus que puede hablar y cuando el slave es llamado por su dirección única (cada slave tiene que tener una dirección única dentro de un mismo bus, ya que, si no, intentarían comunicarse a la vez).

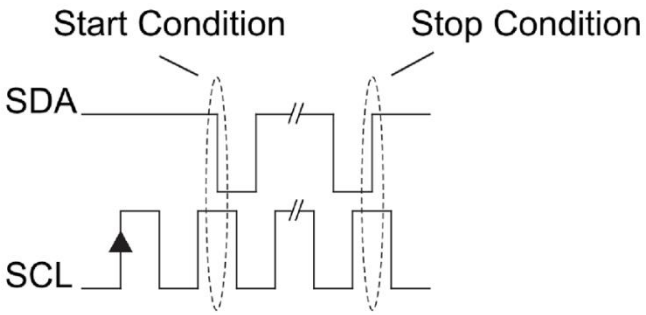


Figura 9.11.- *Condición de inicio/ fin de la comunicación*

Para saber cuándo habla el master hay una condición de inicio y una de fin como se puede ver en la **Figura 9.11**, la condición de inicio de comunicación es un flanco de bajada en SDA mientras SCL está en alto y la condición de fin, en cambio, es un flanco de subida de SDA con SCL también en alto.

Después de dar la orden de inicio, el master envía la dirección del slave en cuestión de 7 bit con un octavo bit que indica si quiere leer o escribir. Justo a continuación el periférico llamado por su dirección debe poner SDA en bajo para confirmar que ha recibido la instrucción (bit de acknowledge ACK) ya que como hemos visto anteriormente el canal tiene pull-up que lo dejará en alto si nadie habla.

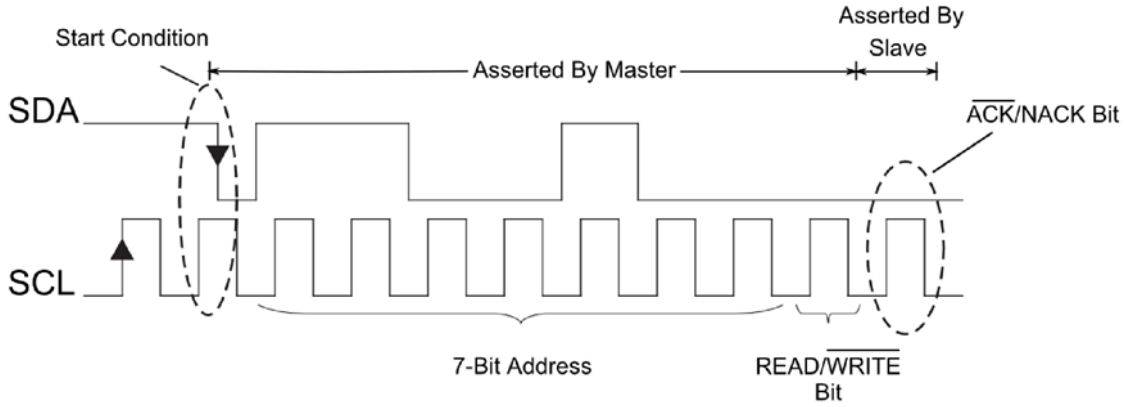


Figura 9.12.- *Inicio de comunicación master-slave*

En caso de que el slave tenga más de un registro y se quiera leer uno en concreto, el master deberá de empezar con write y después del ACK del slave, le enviará la dirección a leer. A continuación, el máster volverá a enviar el comando de inicio, la dirección del dispositivo con un 1 como LSB (read) y el slave enviará los datos del registro deseado después del bit ACK.

9.3.2. SPI

SPI (Serial Peripherial Interface) es un estándar de comunicaciones síncrono que se usa principalmente para la transferencia de información entre circuitos integrados que se encuentran a poca distancia dentro de ensamblados electrónicos.

SPI fue desarrollada alrededor del año 1985 por Motorola, empresa que hoy en día forma parte de NXP Semiconductors. Desde su creación, éste protocolo de comunicación, se ha convertido en un estándar empleado por muchos fabricantes en el mundo de los semiconductores, especialmente en los microcontroladores. SPI tiene las siguientes ventajas:

- Es una interfaz con direccionamiento de hardware simple que es flexible en la cantidad de datos transmitidos.
- Utiliza un sistema master-slave, con un master que puede manejar varios slaves a la vez.
- Se puede conectar en mod full-duplex con el que se obtienen velocidades de transmisión muy levadas.
- No usa un protocolo de tamaños de paquetes estándar, siendo ideal para la transmisión de grandes paquetes de datos.

Con SPI se pueden utilizar un máximo de 4 líneas (ver **Figura 9.13**). El master, generalmente un µC, es el que genera la señal de clock (SCK) y de selección de chip (CS) y por las líneas MOSI (Master Out Slave In) y MISO (Master In Slave Out) se realiza la función completa de multiplexor.

- MOSI: lleva los datos del master al slave.
- MISO: lleva los datos del slave al master.
- SCK: sincroniza los dispositivos.
- CS: selecciona y habilita un slave.

En caso de que solo haya un solo master y un solo slave, la línea de selección de chip se puede eliminar forzando el pin al estado habilitado permanentemente. En caso de que el slave solo pueda enviar datos la línea MOSI también puede ser eliminada (comunicación semidúplex), en este caso, la conexión sería de solamente dos cables.

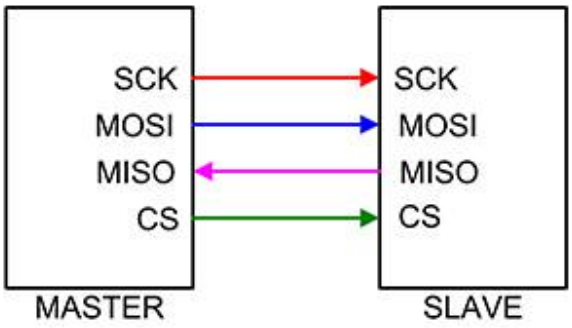


Figura 9.13.- *Esquema de conexión básico de SPI*

En SPI hay 2 configuraciones para transferir datos de un master a varios slave y viceversa, como se puede ver en la **Figura 9.14**, la primera, llamada conexión directa o paralela, consiste en conectar un pin CS de cada slave directamente al master para seleccionar el chip con el que comunicarse; el primer método es puede utilizar fácilmente cuando hay pocos esclavos en el mismo bus, este método está limitado por los pines del micro destinados a esta función. En caso de disponer de varios slave y no disponer de suficientes recursos del micro o no querer utilizarlos para este fin, existe la posibilidad de conectar los CS en un mismo bus y el bus MOSI con una conexión encadenada de manera que los dispositivos transfieren los datos de un dispositivo a otro por un registro de turnos.

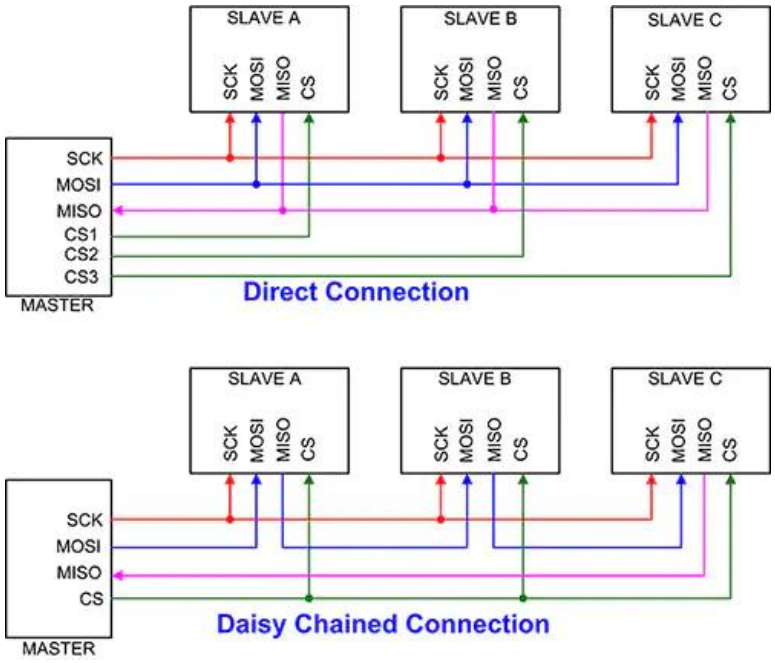


Figura 9.14.- *Esquema de las 2 variantes de interconexión de un master con variso slaves en SPI*

El master genera y controla la señal de reloj, que tiene dos atributos: la polaridad (CPOL) y la fase (CPHA). Estos atributos controlan el flanco del reloj. La configuración de CPOL y CPHA determinan la polaridad del reloj y el flanco activo para la temporización de los datos, habitualmente se utiliza el Modo 1 donde CPOL es 0 (inactivo a nivel bajo) y CPHA es 1 (flanco de cambio de activo a inactivo),0 ver **Figura 9.15**.

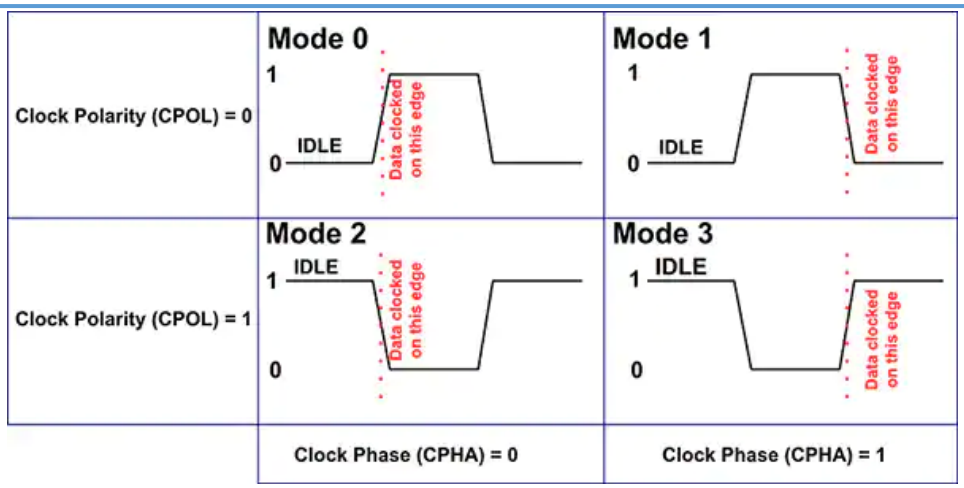


Figura 9.15.- *Esquema de atributos del clock en SPI*

9.3.3. Comunicación USART

USART (Universal Synchronous and Asynchronous serial Reciever and Transmitter) es un tipo de comunicación serial altamente flexible en los que entre sus características destacan:

- Operación full-duplex.
- Registros de transmisión/recepción independientes.
- Posibilidad de operación síncrona (half-duplex) o asíncrona (full-duplex).
- Detección de errores.
- Filtro de ruido.
- Doble velocidad en asícrono.

USART se emplea en comunicaciones duales, es decir, que se pueden transmitir y recibir datos simultáneamente. Los datos son transmitidos de manera serial lo que significa que sólo un bit es transmitido por el canal en un instante determinado de tiempo, cosa que hace este tipo de interfaces sencillas y baratas de implementar y en consecuencia fueron el sistema más utilizado hasta la aparición del protocolo USB.

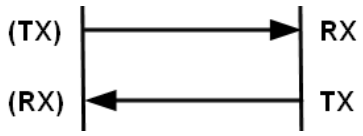


Figura 9.16.- *Diagrama de interconexión USART*

La interconexión se realiza con 2 cables (two-wire connection) Tx (Transmisión) y Rx (recepción) dónde el Tx de un dispositivo está conectado al Rx del otro y viceversa, de manera que si sólo se conecta uno de ellos también habría transmisión, pero solo de manera unidireccional. La comunicación UART garantiza comunicaciones de gran confiabilidad hasta a grandes distancias, pero no son recomendables para comunicar más de dos elementos.

Este formato funciona por ventanas también llamadas frames, que tienen las siguientes características:

- Un bit de parada.
- Bits de datos (5-9 según configuración).

- 1-2 bits de parada.
- Se pueden incluir bits de paridad para la detección y corrección de errores.

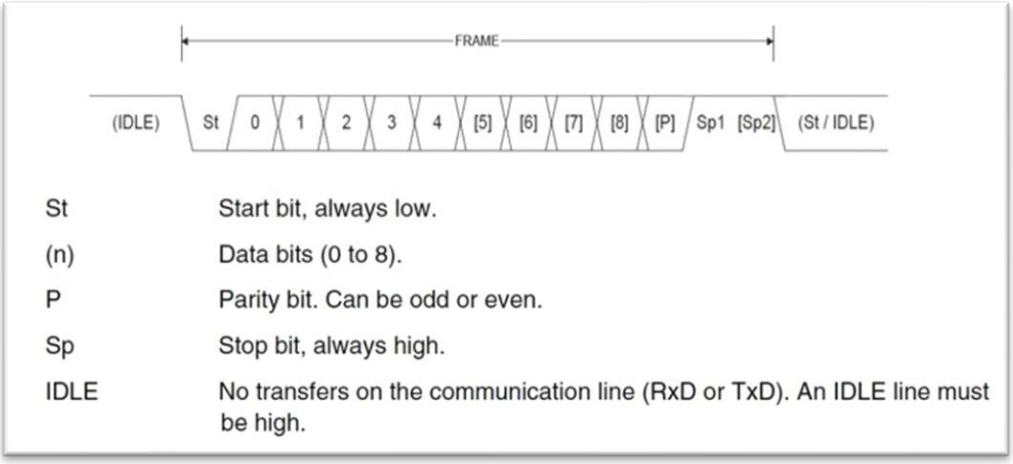


Figura 9.17.- *Ejemplo de ventana de transmisión en USART*

Este tipo de comunicación es muy utilizado actualmente para programar desde un ordenador microprocesadores de sistemas embebidos por lo que no se requieren micros con un bootloader precargado.

10. Análisis económico

En este apartado se pretende mostrar, analizar y desglosar el costo de la realización del prototipo del proyecto realizado incluyendo todos sus gastos derivados. Después, se hará una aproximación del costo de la producción en serie del kit si se quisiera poner a la venta al público y el precio final de venta correspondiente.

10.1. Coste del prototipo

Para calcular el coste total del prototipo se tienen que tener en cuenta tanto los materiales para la realización del kit como las horas dedicadas a cada parte del proyecto. Al final se hará una suma para conocer el coste total del proyecto.

10.1.1. Coste de los materiales

El coste individual de cada artículo adquirido para la realización del prototipo junto con el coste total antes y después de impuestos se resume en la **Tabla 10.1**.

Tabla 10.1.- *Lista de materiales utilizados para el prototipo con sus proveedores y costes:*

Posición	Componente	Cantidad	Proveedor	Precio ud. (€ sin IVA)
1	NUCLEO F401RE	1	RS	12,86
2	LED	5	RS	0,32
3	Pulsadores	2	Digi-Key	0,31
4	Interruptores	2	RS	2,47
5	Teclado matricial 4x4	1	RS	17,42
6	LCD 2x16 mini	1	Digi-Key	7,56
7	HC-SR04	1	Tiendaotec	1,95
8	LM35	1	RS	1,95
9	GP2Y0A21YK0F	1	Farnell	9,83
10	ADT7420	1	Digi-Key	5,91
11	BME280	1	Digi-Key	12,81
12	HC-05	1	Diotronic	12,37
13	K78L05	1	Farnell	2,23
14	Pack conectores shield	1	Amazon	13,99
15	Pack 120 cables	1	Amazon	6,24
16	Pack 200 jumpers	1	Amazon	10,64
17	Pack 650 cond. Cer.	1	Amazon	13,22
18	Pack 60xLM7805	1	Amazon	13,29
19	Pack 20 potenciómetros	1	Amazon	9,98
20	Total de envíos proveedores		-	11,32
21	PCB Em. Pos. 100x160	1	EBF	5,79
22	Fotolitos PCB	6	Copisteria EEBE	0,41
23	Fotolitos PCB	4	CopyCrom	1,65
Precio total (sin IVA)				**185,58**
Precio total (con IVA)				**224,55**

10.1.2. Coste de recursos humanos

En la **Tabla 10.2** se resumen los costes de los recursos humanos destinados al prototipo, en este caso, no se incluyen las horas de desplazamiento para realizar diferentes trabajos o gestiones ni tampoco las de los profesionales que han servido de apoyo para la realización del proyecto como p.ej. el tutor, personal de laboratorio o ingenieros y técnicos consultados.

Los precios de los trabajos por hora (h) son los mismos ya que los trabajos han sido realizados por la misma persona y habitualmente la tarifa horaria se basa en los conocimientos y experiencia previas del operario. También tiene sentido poner la misma tarifa para cualquier actividad, ya que al fin y al cabo si el operario tiene un salario fijo el coste por hora para una empresa sería el mismo independientemente de la actividad realizada.

Tabla 10.2.- *Resumen de los recursos humanos destinados al prototipo con su coste en caso de un proyecto interno:*

Posición	Concepto	Horas	Precio/h	Total
1	Estudio de ARM	32	23,00	736,00
2	Estudio del microcontrolador y placa de desarrollo	40	23,00	920,00
3	Estudio del kit anterior	8	23,00	184,00
4	Estudio de componentes y placas	40	23,00	920,00
5	Comprobación de funcionamiento de los componentes	80	23,00	1840,00
4	Estudio del software de diseño	32	23,00	736,00
6	Diseño electrónico PCB	52	23,00	1196,00
7	Montaje de la PCB	24	23,00	552,00
8	Revisión y corrección de PCB	32	23,00	736,00
9	Estudio del software para el microcontrolador	80	23,00	1840,00
10	Realización de pruebas y códigos de ejemplo	80	23,00	1840,00
11	Redacción de documentación	120	23,00	2760,00
12	Horas de consulta y seguimiento	12	23,00	276,00
13	Revisión y corrección de la documentación	8	23,00	184,00
Total (sin IVA)		640		14720,00
Total (conIVA)		640		17811,20

10.1.3. Coste total del prototipo

En el coste total del prototipo se incluyen todos los costes detallados anteriormente en los apartados 10.1.1. y 10.1.2. . El propósito de este apartado es ver el precio de venta de este proyecto, por este motivo, se incluirán los impuestos y un margen razonable de un 20% desglosado en un 12% de margen para gastos generales (en los que se incluyen materiales y recursos humanos) y un 8% de beneficio neto para la empresa.

Tabla 10.3.- *Resumen del coste total del prototipo:*

Concepto	Coste (€)
Coste de materiales	185,58
Coste recursos humanos	14720,00
Margen de gastos generales (12%)	1788,67
Margen de beneficio (8%)	1182,45
Impuestos (21%)	3756,21
Coste total	**21642,91**

Una vez calculados todos los costes mencionados anteriormente, el coste total de venta del proyecto junto con el prototipo es de **21642,91 €**, en la **Tabla 10.3** se puede ver el resumen detallado. En cambio, si el proyecto se realizase internamente, el gasto simplemente sería la suma de los materiales y los recursos humanos, obteniendo un valor de **14905,58 €** que supondría una reducción de un **31,13%** ya que no tendríamos que pagar los impuestos ni tendríamos que pagar el margen de beneficio ni seguridad que necesitaría cualquier empresa externa**.**

10.2. Producción en serie

Un valor a tener muy en cuenta en el momento de realizar un proyecto es el precio final de venta al público si se quisiera comercializar. En el caso de este proyecto en concreto, al haberse realizado una placa que incluye distintos periféricos que pueden ser utilizados para una gran variedad de aplicaciones es razonable pensar que es un artículo que se puede llegar a fabricar en serie para reducir así el coste unitario del producto para el posible comprador final.

10.2.1. Coste de producción

En caso de hacer este proyecto, un factor clave sería cuantas unidades se pretenden producir, ya que cuantas más unidades se hagan de golpe menor será el precio de producción, aunque puede derivar en gastos de almacenaje y de devaluación del producto. Para este caso concreto, se plantea el escenario de que se han realizado varios proyectos, se realiza una serie relativamente pequeña de cada uno para ver la aceptación que tiene en el mercado y una vez puesto a la venta, se observa el volumen de ventas y a partir de allí se decide de cuál de los productos se desea hacer otra serie y en qué cantidad.

La primera serie sería de 100 unidades, a partir de aquí se calcularán todos los costes. Otros puntos a tener en cuenta es que se integrarían los circuitos de las placas utilizadas para el prototipo directamente a la propia para abaratar costes y simplificar procesos, ya que supone un proceso de diseño electrónico relativamente rápido y simple. En la misma página dónde te ofrecen la fabricación de la PCB también te ofrecen los componentes, y el montaje. Una vez observados todos los gastos se obtiene que el coste de producción de la primera producción de 100 unidades sería de **4081,48 €** con el gasto desglosado en la **Tabla 10.4**.

***Tabla 10.4.**-Coste de producción de 100 unidades:*

Posición	Concepto	Cantidad	Precio	Precio ud.
1	PCB	100	92,78	0,930
2	Montaje	100	212,23	2,120
3	Resistencias	2000	14,00	0,007
4	Condensadores	1000	3,00	0,003
5	Diodos	200	3,20	0,016
6	Transistores	100	2,00	0,020
7	Buffer para BME280	100	22,90	0,229
8	LEDs	500	85,50	0,171
9	Pulsadores	200	30,20	0,151
10	Interruptores	200	140,04	0,702
11	Teclado matricial 4x4	100	633,80	6,338
12	LCD 2x16 mini	100	643,20	6,432
13	HC-SR04	100	108,33	1,083
14	LM35	100	28,94	0,289
15	GP2Y0A21YK0F	100	754,00	7,540
16	ADT7420	100	462,10	4,621
17	BME280	100	88,56	0,886
18	HC-05	100	194,80	1,948
19	K78L05	100	183,90	1,839
20	Conectores hembra 38 pines	200	350,00	1,75
21	Conector alimentación	100	28,00	0,28
Coste total 100 unidades (sin IVA)			4081,48	
Coste total 100 unidades (con IVA)			4938,59	

10.2.2. Coste unitario

En este apartado se pretende calcular el coste unitario de producción para la placa de periféricos diseñada. Una vez calculado el coste de producción de la primera tirada vamos a proceder a calcular el coste unitario de producción de cada placa, como ya tenemos muchos datos, simplemente calcularemos el coste de producción unitario, que en este caso simplemente se dividirán los 4081,48€ calculados en el apartado anterior entre 100, obteniendo un coste unitario de producción **40,82€**.

10.2.3. Precio de venta al público

Sabiendo el precio del prototipo, el precio de producción de la primera serie de 100 unidades y el coste que esta producción unitariamente comporta, podemos calcular un precio de venta lo más realista posible teniendo en cuenta que se pretende recuperar la inversión realizada y obtener un beneficio. Llegados a este punto, podemos proponer diversas hipótesis de precios de venta en el mercado planteando distintos escenarios.

En la ecuación [10.1] se describe el punto de equilibrio del proyecto, que significa el punto en el que se recupera la inversión y se empieza a tener beneficios donde $C_{prototipo}$ es el coste del prototipo (interno, 14905,58), $C_{producción}$ el coste de producción unitario (40,82) y P_{venta} el precio de venta unitario al público (se divide entre 1,21 para tener en cuenta el IVA), todos en €. Las uds son las unidades producidas y vendidas (planteamos el escenario de que se venden todas las unidades producidas), que es la única variable en la ecuación por lo que, en función del precio de venta, nos determinará un punto de equilibrio u otro.

$$\textbf{[10.1]} \quad C_{prototipo} + C_{producción} \cdot uds. = \frac{P_{venta}}{1{,}21} \cdot uds$$

Fijando diferentes valores de precio de venta o de unidades podemos fijar el punto de equilibrio en un punto u otro. Es importante tener en cuenta que en cuantas menas unidades se fije el punto de equilibrio, mayor será el beneficio por las mismas unidades vendidas. En la **Tabla 10.5** se resumen las diferentes hipótesis de precio de venta, todas son válidas y, dependiendo de diferentes factores, especialmente del estado del mercado, la oferta, el precio de otros fabricantes que produzcan dispositivos similares y de la demanda por parte de los consumidores se podría fijar un precio de mercado u otro.

Tabla 10.5.- *Hipótesis de precio de venta, punto de equilibrio y beneficio en % después de p.e.:*

Precio de venta (sin IVA)	Precio de venta (con IVA)	Punto de equilibrio	% beneficio
189,88	229,75	100	64,88
100,44	121,54	250	49,05
70,63	85,46	500	34,88
55,73	67,43	1000	22,11
181,81	219,99	106	64,09
165,28	199,99	120	62,23
123,95	149,99	180	55,42
82,63	99,99	357	41,81
66,10	79,99	590	31,60

El % de beneficio se refiere al % del precio de venta (con IVA) que no está destinado a su producción o impuestos que, en caso de haber sobrepasado el punto de equilibrio, serían beneficios.

En la **Tabla 10.5** se resumen los coses del proyecto y los posibles precios de venta calculados.

11. Impacto medioambiental

Cualquier equipo electrónico fabricado tiene un doble impacto medioambiental, por un lado, el impacto generado para la obtención de la materia prima utilizada para su fabricación y por otro, el generado al final de su vida útil.

El aumento del uso de equipamiento electrónico a crecido de manera descontrolada en las últimas décadas hecho que ha provocado una explotación minera masiva que aún así, no es suficiente para cubrir la demanda actual provocando problemas de stock en los últimos tiempos. Este hecho es también consecuencia del COVID-19, que a inicios de 2020 produjo una reducción de la oferta y un gran aumento de la demanda producida por el confinamiento, ya que la mayoría de la población tuvo que pasar del contacto físico al remoto recurriendo al uso de dispositivos electrónicos[xxvi].

El mayor problema de la electrónica de hoy en día es que hay pocos equipos que permanezcan en las manos del mismo consumidor más de un par de años, provocando así una gran cantidad de residuos[xxv]. Las industrias, p. ej. los fabricantes de smartphones, impulsan constantemente a los clientes a comprar siempre la última versión y descartar equipos en buen estado simplemente por haber pasado de moda.

Los residuos de aparatos eléctricos y electrónicos (RAEE)son los derivados de la electrónica de consumo, como ordenadores, lavadoras o batidoras, que una vez dejan de funcionar se vuelven inservibles ya pasan a ser considerados como RAEE.

La legislación de los RAEE viene marcada por el Real Decreto 110/2015, en el que se define lo que es un RAEE y se detalla un modelo de gestión eficiente para estos. Estos elementos, pueden contener sustancias peligrosas como cadmio, mercurio o plomo o gases que afecten el calentamiento global ambos perjudiciales para la salud humana.

Los equipos electrónicos tienen materiales valorizables que tienen que recuperarse en su última etapa de vida, como residuo, a través del reciclado, siendo un ejemplo claro de economía circular. En este modelo entran en juego las 4 R: reducir, reutilizar reparar y reciclar para obtener un beneficio social, medioambiental y colaborar a la sostenibilidad. Los aparatos electrónicos están clasificados en 7 categorías que se pueden observar en la **Figura 11.1**.

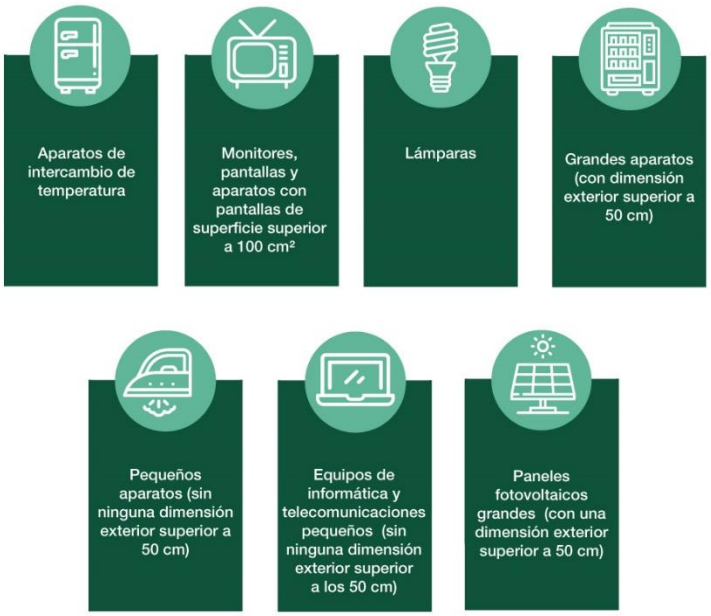


Figura 11.1.- *Clasificación de AEE (imagen de www.ecolec.com)*

En el caso concreto del kit fabricado, se encuentra en la categoría de equipos de informática y telecomunicaciones pequeños, concretamente en el apartado de otros elementos de informática, con código 06005, con el que debería ser catalogado cuando se quiera desechar. Este tipo de elementos tiene que estar siempre etiquetado correctamente con el símbolo de la **Figura 11.2**. Es importante destacar que las soldaduras se han realizado con estaño sin plomo, que reduce los posibles desechos peligrosos creados al pasar a ser un RAEE.

Figura 11.2.- *Símbolo que indica que un aparato tiene que ser recogido de manera selectiva*

El Real Decreto 110/2015 obliga a los productores de aparatos eléctricos y electrónicos a adoptar las medidas necesarias para que los residuos de estos aparatos, puestos por ellos en el mercado, dispongan de sistemas de recogida selectiva y tengan una correcta gestión medioambiental.

La gestión de los RAEE sigue habitualmente el siguiente orden: recogida y transporte, recepción y almacenamiento, clasificación, desmontaje manual y separación de componentes peligrosos, trituración de materiales valorizables i separación de materiales y expedición para valoración externa.

Los puntos de recogida habituales son los puntos limpios, almacenes propios de empresas de distribución y centros de agrupación de carga.

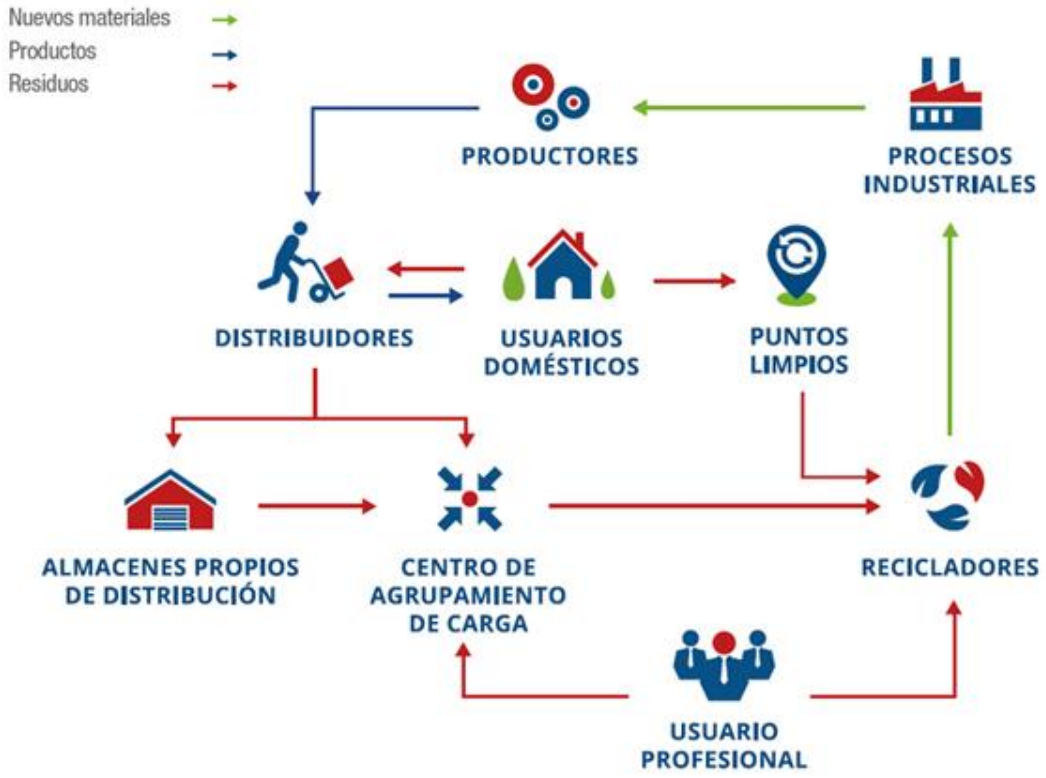


Figura 11.3.- *Ciclo de vida de los AEE (imagen de www.ecotic.es)*

Como se puede observar en la **Figura 11.3** la meta final del proceso de gestión de los RAEE es convertirlos en nuevos recursos una vez llegados al fin de su vida útil.

12. Conclusiones

El objetivo principal del trabajo se ha cumplido, se ha diseñado un kit de laboratorio que puede ser utilizado para el departamento de informática industrial siguiendo las directivas de la asignatura.

12.1. Estudio teórico

El estudio teórico realizado tanto de la arquitectura y su estado del arte como del microcontrolador y placa objeto del proyecto, indica que el microcontrolador escogido es de los más utilizados en la actualidad y que cada vez más empresas lo escogen o se plantean escogerlo para futuros proyectos. El microcontrolador es muy potente y versátil y, en conclusión, es una muy buena elección para integrar en un kit sin que se tenga que pensar en cambiarlo por otro a medio plazo y que supone una mejora sustancial en términos de potencial y versatilidad frente el actual.

12.2. Elección de componentes

Los componentes escogidos cumplen con los requisitos establecidos, ya que cuentan con dispositivos de entrada y salida digitales, sensores analógicos y digitales de distinta naturaleza y dispositivos con diferentes protocolos de comunicación.

12.3. Diseño electrónico

El diseño electrónico ha sido realizado con éxito y cumple con las especificaciones necesarias.

12.4. Prototipado

El prototipado ha resultado ser más complicado de lo que parecía inicialmente y se ha obtenido un resultado funcional pero, sin el acabado profesional deseado. La realización de PCBs de manera manual con elementos con pitch o pads pequeños (como los conectores para shield) no es la más adecuada.

12.5. Programación

En cuanto a la programación se han creado códigos de ejemplo que cumplen con los requisitos de las prácticas actuales, aunque no se ha llegado a tratar prácticamente el tema de las comunicaciones, que no era un objetivo principal del proyecto.

12.6. Comparación y análisis

En primer lugar, es importante señalar algunas ventajas y desventajas que ofrece este kit frente al anterior:

- Ventajas:
  - Arquitectura ARM.
  - 32 bits.
  - Actual, se utiliza en las empresas, punto importante de cara al futuro laboral del estudiante.
  - Es muy flexible, se pueden configurar multitud de funcionalidades a través de software sin necesidad de tocar hardware.
  - Ofrece una cantidad mucho mayor de temporizadores e interrupciones.
  - El micro tiene ADC integrado.
  - Software propio más sencillo e intuitivo.
  - Más compacto y portable.
  - Muy escalable, se puede llegar a hacer proyectos de mucha más complejidad.

- Desventajas:
    - Se tiene que fabricar o adquirir (económicamente supone un gasto, aunque asumible)
    - Contiene menos componentes, aunque son suficientes para la extensión de la asignatura y más.
    - Se acostumbra a trabajar con un mayor nivel de abstracción, aunque es la tendencia actual de la programación.

Un punto a tener en cuenta siempre en el mundo capitalista es el económico, para saber si el kit está a precio de mercado se tiene que comparar con kits prefabricados.

STMicroelectronics ofrece placas llamadas Discovery, que ya integran diversos periféricos. Una placa con periféricos interesantes compatibles con la asignatura sería la STM32F412ZG-DISCOVERY que integra los siguientes componentes:

- STM32F412ZGT6.
- ST-LINK/V2-1.
- LCD TFT a color de 1,54" con pantalla capacitiva.
- I2S Audio Codec con Jack estéreo, micrófono y salida de altavoz.
- Micrófonos estéreo digitales MEMS.
- Conector Micro SD.
- Conector de extensión para I^2C.
- Flas de 128 Mbit por SPI 4 cables.
- Botón RESET y Joystick.
- 4 LED programables para el usuario.
- Diferentes tipos de alimentación.
- Conectores equivalentes a arduino Uno.

Parece la opción más razonable, por los periféricos que incluye, además tiene pines disponibles que permitirían ampliar los periféricos con los que faltarían, sensores analógicos, por ejemplo (aunque el micro tiene de integrados internamente). Su precio es de 31,72 €.

Comparando ambos kits, no parece una opción óptima para un kit universitario de laboratorio, aunque si una posibilidad a analizar en un futuro TFG por ejemplo, ya que la utilización de una polaca estándar de este tipo podría suponer que el alumno lo pudiera adquirir y utilizar desde casa facilitando así su uso y aprendizaje.

Tabla 12.1.- *Resumen de los costes y de los posibles precios de venta:*

Concepto	**Coste**
Coste del prototipado (sin externalizar)	14905,58€
Coste del prototipado (externalizando)	21642,91 €
Coste de producción de 100 unidades	4081,48 €
Coste unitario de producción	40,82 €
Coste de venta al público (equilibrio 180 uds.)	149,99 €
Coste de venta al público (equilibrio 106 uds.)	219,99 €

En la **Tabla 12.1** se pueden observar un resumen del posible coste del kit, en mi opinión es un precio muy ajustado, teniendo en cuenta de que se trata de un proyecto hecho a medida y que ha sido llevado a cabo por un estudiante. Para una institución como la universidad, teniendo en cuenta que es un material necesario para el aprendizaje, es un precio más que abordable.

12.7. Conclusión final

El kit diseñado y fabricado, es una actualización lógica del kit que se está utilizando actualmente. El microcontrolador que se utiliza actualmente fue desarrollado en el 1980 y, aunque se ha ido actualizando, está prácticamente en desuso. El mundo de la electrónica está en constante cambio y el cambio por un controlador mucho más potente, con más recursos parece obligado a corto plazo.

La actualización del kit, supondría también un enfoque diferente de los estudiantes a la electrónica, ya que podría mostrar el potencial actual de esta.

Teniendo en cuenta todo esto, en mi opinión, la implementación de este kit sería una buena opción de cara a cambiar definitivamente el equipo de laboratorio que se utiliza actualmente.

Referencias

Bibliografía

[1] Seal, D., ***"ARM Architecture Reference Manual"***, *Addison-Wesley, EEUU, 9ºedición, julio de 2005.*

[2] Ryzhyk, L., ***"The ARM Architecture"***, *Chicago University, Illinois, EEUU, junio de 2006.*

[3] ARM Limited©, ***"ARMv7-M Architecture Reference Manual"***, *Cambridge, Inglaterra, versión 8, junio de 2018.*

[4] ARM Limited©, ***"ARM Architecture Reference Manual Armv8"***, *Cambridge, Inglaterra, versión 7, enero de 2021.*

[5] Gámiz, J., Tornil, S., **"INSTRUCTOR I2Kit"**, UPC, Barcelona, España, mayo 2018.

[6] AspenCore©, **"2019 Embedded Markets Study"**, EETimes embedded, marzo 2019.

[7] Manzanares, M., "**Prácticas de tecnología electrónica: Práctica PCB"**, UPC, Barcelona, España, noviembre 2019.

[8] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 1: Introducció al µinstructor I2Kit. Entorn de desenvolupament d'aplicacions"**, UPC, Barcelona, España, mayo 2018.

[9] Gámiz, J., Segura, J., Tornil, S., **"Práctica 2: Programación en ensamblador"**, UPC, Barcelona, España, mayo 2018.

[10] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 3: Teclados y visualizadores"**, UPC, Barcelona, España, mayo 2018.

[11] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 4: Temporizadores e interrupciones"**, UPC, Barcelona, España, mayo 2018.

[12] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 5: Programación en C"**, UPC, Barcelona, España, mayo 2018.

[13] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 6: Comunicaciones serie"**, UPC, Barcelona, España, mayo 2018.

[14] Gámiz, J., Segura, J., Tornil, S., **"Pràctica 7: Conversión ADC"**, UPC, Barcelona, España, mayo 2018.

Hojas de datos y manuales de usuario

[A] STMicroelectronics®, ***"UM1724 User manual",*** *versión 14, agosto de 2020.*

[B] STMicroelectronics®, ***"RM0368 Reference manual",*** *versión 5, diciembre de 2018.*

[C] STMicroelectronics®, **"STM32F401xD STM32F401xE"**, versión 3, enero de 2015.

Webgrafía

[i] https://www.st.com/en/microcontrollers-microprocessors/stm32f401re.html
Consultada a 12.01.2021

[ii] http://www.programacion11.com/uploads/2/6/0/5/26056302/arminstrucciones.pdf
Consultada a 14.01.2021

[iii] https://tallerelectronica.com/2018/01/09/aprende-ensamblador-arm-de-manera-sencilla-y-visual/
Consultada a 14.01.2021

[iv] https://www.youtube.com/watch?v=KZtByM825sg&ab_channel=ElectroClub
Consultada a 14.01.2021

[v] https://www.youtube.com/watch?v=hyZS2p1tW-g&ab_channel=Digi-Key
Consultada a 16.01.2021

[vi] https://www.libreservo.com/es/articulo/software-para-programar-stm32
Consultada a 07.02.2021

[vii] https://os.mbed.com/platforms/ST-Nucleo-F401RE/
Consultada a 17.02.2021
[viii] https://www.arm.com/why-arm/architecture/cpu
Consultada a 14.03.2021
[ix] https://www.muycomputer.com/2021/03/31/arm-anuncia-armv9/
Consultada a 14.03.2021
[x] https://hardzone.es/noticias/procesadores/nueva-isa-armv9/
Consultada a 15.03.2021
[xi] https://www.xataka.com/empresas-y-economia/oficial-nvidia-adquiere-arm-40-000-millones-dolares
Consultada a 18.03.2021
[xii] https://www.arm.com/company/news/2021/03/arms-answer-to-the-future-of-ai-armv9-architecture
Consultada a 20.03.2021
[xiii] https://developer.arm.com/ip-products/processors/cortex-r
Consultada a 20.03.2021
[xiv] https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html
Consultada a 12.06.21
[xv] https://www.digikey.es/es/articles/why-how-to-use-serial-peripheral-interface-simplify-connections-between-multiple-devicessp
Consultada a 13.06.21
[xvi] https://www.st.com/en/development-tools/stm32cubeide.html
Consultada a 18.06.21
[xvii] https://www.keil.com/support/man/docs/uv4/uv4_overview.htm
Consultada a 19.06.21
[xviii] https://www.altium.com/
Consultada a 21.06.21
[xix] https://easyeda.com/page/about
Consultada a 21.06.21
[xx] https://www.enerxia.net/portal/index.php?option=com_content&view=article&id=406:electronica-proteus-simulador-digital-y-analogico&catid=61&Itemid=142
Consultada a 01.07.21
[xxi] https://www.youtube.com/watch?v=wxpGckPe0VI&list=PLvHxU_pY8Jf9l9cEDZSklosN3XHMrSq0F&index=6
Consultada a 05.07.21
[xxii] https://store.arduino.cc/products/arduino-uno-rev3
Consultada a 10.08.21
[xxiii] https://estore.st.com/en/products/evaluation-tools/product-evaluation-tools/mcu-mpu-eval-tools/stm32-mcu-mpu-eval-tools/stm32-nucleo-boards/nucleo-f401re.html
Consultada a 10.08.21
[xxiv] https://www.youtube.com/watch?v=ZFK7uBLCABs
Consultada a 20.08.21
[xxv] https://www.grupobraceli.com/2018/09/27/el-impacto-ambiental-de-los-residuos-electricos-y-electronicos/
Consultada a 30.08.21
[xxvi] https://www.elmundo.es/tecnologia/2021/05/24/60a7c903fdddff53328b464a.html
Consultada a 31.08.21
[xxvii] https://ecolec.es/informacion-y-recursos/sobre-los-raee/

Consultada a 31.08.21

Links de compra para los componentes del prototipo

[xxviii] NUCLEO-F401RE:
https://es.rs-online.com/web/p/kits-de-desarrollo-de-microcontroladores/8029425/?cm_mmc=ES-PLA-DS3A-_-google-_-PLA_ES_ES_Raspberry_Pi_%26_Arduino_y_M%C3%B3dulos_de_Desarrollo_Whoop-_-(ES:Whoop!)+Kits+de+Desarrollo+de+Microcontroladores-_-8029425&matchtype=&aud-827186183686:pla-338363142920&gclid=Cj0KCQiA9P__BRC0ARIsAEZ6irimk4IfMOUk2ZmpgN1tDzVhmGeVF-xcAEO2AJWjpAIAAoAtGcEVNdwaAmJdEALw_wcB&gclsrc=aw.ds

[xxix] LEDs:
https://es.rs-online.com/web/p/leds/2285988/

[xxx] Pulsadores:
https://www.digikey.es/product-detail/es/omron-electronics-inc-emc-div/B3F-1022/SW403-ND/44674?utm_adgroup=Tactile%20Switches&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Switches&utm_term=&productid=44674&gclid=CjwKCAjw7rWKBhAtEiwAJ3CWLNIFtjvWTkQdQdlSxZOrf3oOW2zfdHyA5MiCeeVFPJAV-PXbYwPYUhoCKagQAvD_BwE

[xxxi] Interruptores:
https://es.rs-online.com/web/p/interruptores-de-palanca/7347154

[xxxii] Teclado matricial 4x4 no alimentado:
https://es.rs-online.com/web/p/teclados-numericos/0146222/

[xxxiii] Visualizador mini LCD 2 filas 16 caracteres:
https://www.digikey.com/es/products/detail/midas-displays/MC21603A6W-SPR-V2/13970973

[xxxiv] Sensor HC-SR04:
https://www.tiendatec.es/arduino/modulos/392-modulo-hc-sr04-sensor-distancia-por-ultrasonidos-para-arduino-y-raspberry-pi-8403921180011.html?kk=a4c6365-177409dddba-36d58&gclid=Cj0KCQiApY6BBhCsARIsAOI_GjZ_TP25ZKzU00OaK6quPYPn86O-vTAv5RhO_KA-wjjk5-8olvYIyUYaAhNuEALw_wcB&utm_source=kelkooes&utm_medium=cpc&utm_campaign=kelkooclick&utm_term=tiendatec+MODULO+HC-SR04+SENSOR+DISTANCI&from=kelkooes

[xxxv] Sensor LM35:
https://es.rs-online.com/web/p/circuitos-integrados-de-sensores-de-temperatura-y-humedad/1977272/?cm_mmc=ES-PLA-DS3A-_-google-_-CSS_ES_ES_Semiconductores_Whoop-_-(ES:Whoop%21)+Circuitos+Integrados+de+Sensores+de+Temperatura+y+Humedad-_-1977272&matchtype=&aud-826607888547:pla-305134321580&gclid=Cj0KCQiApY6BBhCsARIsAOI_GjbF4sGzUNKJPxLm-RvgBelGJi9lqfAff5BU7CW-9xO1JWIpRHJ9j4YaAtmOEALw_wcB&gclsrc=aw.ds

[xxxvi] Sensor GP2Y0A21YK0F:
https://es.farnell.com/sharp/gp2y0a21yk0f/sensor-distancia-analgico/dp/1243869?gclid=CjwKCAjw7rWKBhAtEiwAJ3CWLBXBRCBKguP-x0peoMkkoKhp-Bb5xduw0hxok4exqvm2v50jp7UQJBoCyNIQAvD_BwE&mckv=scaQmwVKd_dc|pcrid|4

90629289402|plid||kword||match||slid||product|1243869|pgrid|114774185956|ptaid|aud-366647999927:pla-361296785674&CMP=KNC-GES-GEN-SHOPPING-SMEC-Whoops-Low-Desktop-Title-Changes-10-Aug-21&gross_price=true

[xxxvii] Placa con sensor ADT7420: https://www.digikey.es/product-detail/es/analog-devices-inc/EVAL-ADT7420-PMDZ/EVAL-ADT7420-PMDZ-ND/7324253

[xxxviii] Placa con sensor BME280: https://www.digikey.es/product-detail/es/dfrobot/SEN0236/1738-1312-ND/7597055

[xxxix] Placa con módulo bluetooth HC-05: https://diotronic.com/accesorios-y-sensores/15249-bluetooth-hc-05-tx-module

[xl] Regulador de tensión lineal K78L05: https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwiXmr6nlJfzAhXDiNUKHc9YABsYABAGGgJ3cw&ae=2&ohost=www.google.com&cid=CAESQOD2ARigcLWuZQnzYu9et78EyBQ5-SJ0tKHoOmdIP81FEZf1T8c8cig1vXyLB-Hrft7j7HhaMEyhZ8bLvoY9R6M&sig=AOD64_0IeM6VOOK7NWSi8UcK5hZcSZmAMg&ctype=5&q=&ved=2ahUKEwjloLenlJfzAhVBVhoKHWeXBt0Q9aACegQIARBu&adurl=

[xli] Pack 110 conectores para shield de 2.54 mm: https://www.amazon.es/gp/product/B01M5GP3RH/ref=ppx_od_dt_b_asin_title_s00?ie=UTF8&psc=1

[xlii] Pack 120 cables para prototipado: https://www.amazon.es/gp/product/B01NGTXASZ/ref=ppx_yo_dt_b_asin_title_o07_s00?ie=UTF8&psc=1

[xliii] Pack 200 jumpers: https://www.amazon.es/gp/product/B07K9LCKH7/ref=ppx_yo_dt_b_asin_title_o07_s00?ie=UTF8&psc=1

[xliv] Pack 650 condensadores cerámicos de distintas capacidades: https://www.amazon.es/gp/product/B07PP7ZKTJ/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&psc=1

[xlv] Pack 60 reguladores de tensión LM de diferentes valores: https://www.amazon.es/gp/product/B07KFG71NL/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&psc=1

[xlvi] Pack 20 potenciometros: https://www.amazon.es/gp/product/B07WQG1RNZ/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

Link de compra para la producción en serie

[xlvii] PCB, montaje, Resistencias, Condensadores, Diodos, Transistores, LEDs, Pulsadores, Interruptores: https://www.pcb.market/es-es/home/index

Anexo A: Código de ejemplo 1

- Archivo main.c:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body
  * @author               : Ulric Rossell Currius
  * @concept        : TFG, diseño e implementación de kit de laboratorio a
  *                       partir de STM32 con arquitectura ARM

******************************************************************************
*
  * @attention
  *
  *Partially automatic generated code by STM
  *
  *Code created for an educational purpose
  *

******************************************************************************
*
  *//* USER CODE END Header */
/* Includes ------------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ----------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
-*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
-*/
UART_HandleTypeDef huart2;
```

```
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */


  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  //inicializamos variables
  int P1;
  int P2;
  int I1;
  int I2;
```

```
  int i;
  //Inicializamos los 4 LED en el estado que deseamos, en este caso todos
apagados excepto el 3 (rojo)
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, RESET);
        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
        HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, SET);
        HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, RESET);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //Leemos el estado de los pulsadores e interruptores hasta que se
active alguno

        P1 = HAL_GPIO_ReadPin(P1_GPIO_Port, P1_Pin);
        P2 = HAL_GPIO_ReadPin(P2_GPIO_Port, P2_Pin);
        I1 = HAL_GPIO_ReadPin(I1_GPIO_Port, I1_Pin);
        I2 = HAL_GPIO_ReadPin(I2_GPIO_Port, I2_Pin);

        //una vez se activa un pulsador o interruptor se activa una
secuencia determinada

        //Se activa un pulsador
        if ((P1&P2) != 1){
              //Se enciende el LED4 indicando que nos encontramos en una
secuencia
              HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, SET);

              HAL_Delay(2000);

              //Al cabo de 2 segundos se pone el semáforo en verde
              HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, SET);
              HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, RESET);

              HAL_Delay(5000);
              //Al cabo de 5 segundos se pone en ámbar
              HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
              HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
              HAL_Delay(2000);
              //Al cabo de 2 segundos parpadea en ámbar
              for (i = 0; i<8; i++){
                    HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                    HAL_Delay(500);
              }
              //Al final de la secuencia se deja en rojo y se apaga el LED4

              HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
              HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, SET);

              HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, RESET);
        }

        //Se activa un interruptor
        if(I1|I2){
              //Se apaga el verde en caso de que esté encendido y se pone en
rojo
              HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, RESET);
```

```
                HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, SET);
                /*
                 * mientras algún interruptor esté activo se deja el semáforo
en rojo
                 * y con el led de que indica que se está en una rutina
parapadeando
                 * indicando que no se puede hacer nada hasta que se apague el
interruptor
                 */
                while (I1|I2){
                      HAL_GPIO_TogglePin(LED4_GPIO_Port, LED4_Pin);
                      I1 = HAL_GPIO_ReadPin(I1_GPIO_Port, I1_Pin);
                      I2 = HAL_GPIO_ReadPin(I2_GPIO_Port, I2_Pin);
                      HAL_Delay(200);
                }
                //Una vez apagado el interruptor se deja el LED4 de secuencia
encendido
                HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, SET);
                HAL_Delay(2000);
                //Después de estar 2 segundos en rojo se pone en ambar
intermitente
                HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, RESET);

                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                HAL_Delay(500);
                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                HAL_Delay(500);
                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                HAL_Delay(500);
                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                HAL_Delay(500);
                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
                HAL_Delay(500);
                HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);

                //Se pone en verde y se apaga el azul para acabar la secuencia
                HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, SET);

                HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, RESET);
        }
        /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
```

```
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|LED2_Pin|LED1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LED4_Pin|LED3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin LED2_Pin LED1_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|LED2_Pin|LED1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : I2_Pin I1_Pin */
  GPIO_InitStruct.Pin = I2_Pin|I1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : P1_Pin P2_Pin */
  GPIO_InitStruct.Pin = P1_Pin|P2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : LED4_Pin LED3_Pin */
  GPIO_InitStruct.Pin = LED4_Pin|LED3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

Anexo B: Código de ejemplo 2

- Archivo main.c:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body
  * @author               : Ulric Rossell Currius
  * @concept        : TFG, diseño e implementación de kit de laboratorio a
  *                       partir de STM32 con arquitectura ARM

******************************************************************************
*
  * @attention
  *
  *Partially automatic generated code by STM
  *
  *Code created for an educational purpose
  *

******************************************************************************
*
  *//* USER CODE END Header */
/* Includes ------------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ----------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */
//incuimos los archivos .h
#include "lcd.h"
#include "keypad.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
-*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
-*/
```

```
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
      //definimos una variable que nos dirá la tecla pulsada
      char tecla;
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  LCD_init();
  keypad_init();

  //Mostramos texto de inicialización en LCD
  LCD_write_text("Codigo de");
```

```
  LCD_cmd(LCD_LINEA2);
  LCD_write_text("ejemplo 2");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //Leemos la tecla pulsada
        tecla = keypad_read();

        //Si se ha pulsado alguna tecla la mostramos en la LCD
        if(tecla != 0){
              LCD_cmd(LCD_CLEAR);
              LCD_cmd(LCD_LINEA1);
              LCD_write_text("  Tecla:");
              LCD_cmd(LCD_LINEA2);
              LCD_char(tecla);
        }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
```

```
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|RS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, DB7_Pin|DB6_Pin|E_Pin|DB4_Pin
                          |RW_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DB5_GPIO_Port, DB5_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin RS_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|RS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : DB7_Pin DB6_Pin E_Pin DB4_Pin
                           RW_Pin */
  GPIO_InitStruct.Pin = DB7_Pin|DB6_Pin|E_Pin|DB4_Pin
                          |RW_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : DB5_Pin */
  GPIO_InitStruct.Pin = DB5_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(DB5_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
  {
  }
```

```
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

- Archivo keypad.h:

```
#ifndef __KEYPAD__
#define __KEYPAD__

#include "stm32f4xx_hal_def.h"

#define             X1_PIN       GPIO_PIN_0          //     PC0
#define      X1_PORT             GPIOC

#define             X2_PIN       GPIO_PIN_1          //     PC1
#define      X2_PORT             GPIOC

#define             X3_PIN       GPIO_PIN_0          //     PB0
#define      X3_PORT             GPIOB

#define             X4_PIN       GPIO_PIN_4          //     PA4
#define      X4_PORT             GPIOA

#define             Y1_PIN       GPIO_PIN_8          //     PA8
#define      Y1_PORT             GPIOA

#define             Y2_PIN       GPIO_PIN_4          //     PB4
#define      Y2_PORT             GPIOB

#define             Y3_PIN       GPIO_PIN_5          //     PB5
#define      Y3_PORT             GPIOB

#define             Y4_PIN       GPIO_PIN_4          //     PC4
#define      Y4_PORT             GPIOC
void keypad_init(void);
char keypad_read(void);
#endif
```

- Archivo keypad.c:

```
#include "stm32f4xx_hal.h"
#include "keypad.h"

void keypad_init(void){
	GPIO_InitTypeDef GPIO_InitStruct;

	/* GPIO Ports Clock Enable */
	__HAL_RCC_GPIOA_CLK_ENABLE();
	__HAL_RCC_GPIOB_CLK_ENABLE();

	//se inicializan las cuatro filas (X1-X4) como salidas
	GPIO_InitStruct.Pin       = X1_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Pull      = GPIO_NOPULL;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(X1_PORT, &GPIO_InitStruct);

	GPIO_InitStruct.Pin       = X2_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Pull      = GPIO_NOPULL;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(X2_PORT, &GPIO_InitStruct);

	GPIO_InitStruct.Pin       = X3_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Pull      = GPIO_NOPULL;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(X3_PORT, &GPIO_InitStruct);

	GPIO_InitStruct.Pin       = X4_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Pull      = GPIO_NOPULL;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(X4_PORT, &GPIO_InitStruct);

	//Se inicializan a nivel bajo
	HAL_GPIO_WritePin(X1_PORT, X1_PIN, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(X2_PORT, X2_PIN, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(X3_PORT, X3_PIN, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(X4_PORT, X4_PIN, GPIO_PIN_RESET);

	//se configuran las columnas (y1-y4) como entradas con pull-up
	GPIO_InitStruct.Pin       = Y1_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull      = GPIO_PULLUP;
	HAL_GPIO_Init(Y1_PORT, &GPIO_InitStruct);

	GPIO_InitStruct.Pin       = Y2_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull      = GPIO_PULLUP;
	HAL_GPIO_Init(Y2_PORT, &GPIO_InitStruct);

	GPIO_InitStruct.Pin       = Y3_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull      = GPIO_PULLUP;
	HAL_GPIO_Init(Y3_PORT, &GPIO_InitStruct);
```

```
	GPIO_InitStruct.Pin       = Y4_PIN;
	GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull      = GPIO_PULLUP;
	HAL_GPIO_Init(Y4_PORT, &GPIO_InitStruct);

}

//se mapea el teclado
char keypad_read(void){
	char letras[4][4]={{'0','1','2','3'},
	                   {'4','5','6','7'},
	                   {'8','9','A','B'},
	                   {'C','D','E','F'}};
	int  i=0;
	char valor=0;

//se lee la tecla pulsada
	for(i=0;i<4;i++){
		if(i==0){
			HAL_GPIO_WritePin(X1_PORT, X1_PIN,  GPIO_PIN_RESET);
			HAL_GPIO_WritePin(X2_PORT, X2_PIN,  GPIO_PIN_SET);
			HAL_GPIO_WritePin(X3_PORT, X3_PIN,    GPIO_PIN_SET);
			HAL_GPIO_WritePin(X4_PORT, X4_PIN,  GPIO_PIN_SET);

			HAL_Delay(10);
			while ( ( HAL_GPIO_ReadPin(Y1_PORT, Y1_PIN) ) == 0
){valor=letras[0][0];}    //0
			while ( ( HAL_GPIO_ReadPin(Y2_PORT, Y2_PIN) ) == 0
){valor=letras[0][1];}    //1
			while ( ( HAL_GPIO_ReadPin(Y3_PORT, Y3_PIN) ) == 0
){valor=letras[0][2];}    //2
			while ( ( HAL_GPIO_ReadPin(Y4_PORT, Y4_PIN) ) == 0
){valor=letras[0][3];}    //3
		}

		if(i==1){
			HAL_GPIO_WritePin(X1_PORT, X1_PIN,  GPIO_PIN_SET);
			HAL_GPIO_WritePin(X2_PORT, X2_PIN,  GPIO_PIN_RESET);
			HAL_GPIO_WritePin(X3_PORT, X3_PIN,    GPIO_PIN_SET);
			HAL_GPIO_WritePin(X4_PORT, X4_PIN,  GPIO_PIN_SET);

			HAL_Delay(10);
			while ( ( HAL_GPIO_ReadPin(Y1_PORT, Y1_PIN) ) == 0
){valor=letras[1][0];}    //4
			while ( ( HAL_GPIO_ReadPin(Y2_PORT, Y2_PIN) ) == 0
){valor=letras[1][1];}    //5
			while ( ( HAL_GPIO_ReadPin(Y3_PORT, Y3_PIN) ) == 0
){valor=letras[1][2];}    //6
			while ( ( HAL_GPIO_ReadPin(Y4_PORT, Y4_PIN) ) == 0
){valor=letras[1][3];}    //7
		}

		if(i==2){
			HAL_GPIO_WritePin(X1_PORT, X1_PIN,  GPIO_PIN_SET);
			HAL_GPIO_WritePin(X2_PORT, X2_PIN,  GPIO_PIN_SET);
			HAL_GPIO_WritePin(X3_PORT, X3_PIN,    GPIO_PIN_RESET);
			HAL_GPIO_WritePin(X4_PORT, X4_PIN,  GPIO_PIN_SET);

			HAL_Delay(10);
```

```
                    while ( ( HAL_GPIO_ReadPin(Y1_PORT, Y1_PIN) ) == 0
){valor=letras[2][0];}    //8
                    while ( ( HAL_GPIO_ReadPin(Y2_PORT, Y2_PIN) ) == 0
){valor=letras[2][1];}    //9
                    while ( ( HAL_GPIO_ReadPin(Y3_PORT, Y3_PIN) ) == 0
){valor=letras[2][2];}    //A
                    while ( ( HAL_GPIO_ReadPin(Y4_PORT, Y4_PIN) ) == 0
){valor=letras[2][3];}    //B
              }

              if(i==3){
                    HAL_GPIO_WritePin(X1_PORT, X1_PIN,  GPIO_PIN_SET);
                    HAL_GPIO_WritePin(X2_PORT, X2_PIN,  GPIO_PIN_SET);
                    HAL_GPIO_WritePin(X3_PORT, X3_PIN,    GPIO_PIN_SET);
                    HAL_GPIO_WritePin(X4_PORT, X4_PIN,  GPIO_PIN_RESET);

                    HAL_Delay(10);
                    while ( ( HAL_GPIO_ReadPin(Y1_PORT, Y1_PIN) ) == 0
){valor=letras[3][0];}    //C
                    while ( ( HAL_GPIO_ReadPin(Y2_PORT, Y2_PIN) ) == 0
){valor=letras[3][1];}    //D
                    while ( ( HAL_GPIO_ReadPin(Y3_PORT, Y3_PIN) ) == 0
){valor=letras[3][2];}    //E
                    while ( ( HAL_GPIO_ReadPin(Y4_PORT, Y4_PIN) ) == 0
){valor=letras[3][3];}    //F
              }
       }

       //se devuelve el valor de la letra pulsada
       return valor;
}
```

- Archivo lcd.h (código no propio):

```
#include "stm32f4xx_hal.h"

//comandos LCD
#define     LCD_CLEAR         0x01    //Limpia pantalla
#define     LCD_HOME          0x02    //Retorno al inicio
#define     LCD_CURSOR_ON 0x0F    //Cursor on
#define     LCD_CURSOR_OFF      0x0C    //Cursor off
#define     LCD_LINEA1        0x00     //Promera Fila
#define     LCD_LINEA2          0XC0   //Segunda Fila
#define     LCD_LINEA3          0x94   //Tercera Fila
#define     LCD_LINEA4          0xD4   //Cuarta Fila
#define     LCD_LEFT            0x10   //Cursor a la izquierda
#define     LCD_RIGHT           0x14   //Cursor a la derecha
#define     LCD_ROT_LEFT  0x18   //Rotar a la izquierda
#define     LCD_ROT_RIGHT 0x1C   //Rotar a la derecha
#define     LCD_OFF             0x08   //apaga el display

//Declaración de funciones
void LCD_init(void);                    //Inicializa LCD
void LCD_write_nible(unsigned char);    //Envia nible a LCD
void LCD_cmd(unsigned char comando);    //Envia comando a LCD
void LCD_char(char caracter);           //Envia caracteer a LCD
void LCD_write_text(char *dato);
void LCD_gotoxy(char x, char y);        //Posiciona cursor
void LCD_custom_char(unsigned char loc, unsigned char *msg);
```

```
void delay(void);

void delay(void)
{
      uint16_t i;
      for(i=0;i<2000;i++);
}

//inicializacion LCD
void LCD_init(void)
{
    HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DB4_GPIO_Port, DB4_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DB5_GPIO_Port, DB5_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DB6_GPIO_Port, DB6_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DB7_GPIO_Port, DB7_Pin, GPIO_PIN_RESET);


    HAL_Delay(20);              //Espera que se estabilicen puertos

    LCD_write_nible(0x03);       //por fabricante
    HAL_Delay(5);
    LCD_write_nible(0x03);       //por fabricante
    HAL_Delay(1);
    LCD_write_nible(0x03);       //por fabricante
    HAL_Delay(1);
    LCD_write_nible(0x02);
    HAL_Delay(1);               //Configurar modo 4 bits

    LCD_cmd(0x28);
    LCD_cmd(LCD_CURSOR_OFF);     //Cursor apagado
    LCD_cmd(LCD_CLEAR);
    LCD_cmd(0x06);               //Modo incremental, sin desplazamiento
    LCD_cmd(0x80);               //Address DDRam superior izquierda
}

//envia nible al puerto de datos

void LCD_write_nible(unsigned char nible)
{
    if (nible & (1<<0))
        HAL_GPIO_WritePin(DB4_GPIO_Port, DB4_Pin, GPIO_PIN_SET);
    else
        HAL_GPIO_WritePin(DB4_GPIO_Port, DB4_Pin, GPIO_PIN_RESET);

    if (nible & (1<<1))
        HAL_GPIO_WritePin(DB5_GPIO_Port, DB5_Pin, GPIO_PIN_SET);
    else
        HAL_GPIO_WritePin(DB5_GPIO_Port, DB5_Pin, GPIO_PIN_RESET);

    if (nible & (1<<2))
        HAL_GPIO_WritePin(DB6_GPIO_Port, DB6_Pin, GPIO_PIN_SET);
    else
         HAL_GPIO_WritePin(DB6_GPIO_Port, DB6_Pin, GPIO_PIN_RESET);

    if (nible & (1<<3))
        HAL_GPIO_WritePin(DB7_GPIO_Port, DB7_Pin, GPIO_PIN_SET);
    else
```

```
        HAL_GPIO_WritePin(DB7_GPIO_Port, DB7_Pin, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_SET);
    delay();
    HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_RESET);
}

//Enviar comando a la LCD
void LCD_cmd(unsigned char comando)
{
    HAL_GPIO_WritePin(RS_GPIO_Port, RS_Pin, GPIO_PIN_RESET);
    LCD_write_nible(comando >> 4);
    LCD_write_nible(comando & 0x0F);
    if(comando == LCD_CLEAR || comando == LCD_HOME)
      HAL_Delay(1);
    else
       delay();
}

//Envia caracter a la LCD
void LCD_char(char caracter)
{
    HAL_GPIO_WritePin(RS_GPIO_Port, RS_Pin, GPIO_PIN_SET);
    LCD_write_nible(caracter >> 4);
    LCD_write_nible(caracter & 0x0F);
    delay();
}

//Envia cadena de carácteres a la LCD
void LCD_write_text(char *dato)
{
    while (*dato){
        LCD_char(*dato);    // Envia el dato al LCD
        dato++;             // Incrementa el buffer de dato
    }
}

//Cursor de posición xy
void LCD_gotoxy(char x, char y)
{
    char posicion;
    switch (y)
    {
        case 1: posicion = 0x00 + x - 1; break;
        case 2: posicion = 0x40 + x - 1; break;
        case 3: posicion = 0x14 + x - 1; break;
        case 4: posicion = 0x54 + x - 1; break;
        default: posicion = 0x00 + x - 1; break;
    }
    LCD_cmd(0x80 | posicion);
}

//Personalizar carácter
void LCD_custom_char(unsigned char loc,unsigned char *msg)
{
    unsigned char i;
    if(loc<8){
        LCD_cmd(0x40+(loc*8));
        for(i=0;i<8;i++)
```

```
            LCD_char(msg[i]);
    }
}
```

Anexo C: Código de ejemplo 3

Archivo main.c:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body
  * @author               : Ulric Rossell Currius
  * @concept        : TFG, diseño e implementación de kit de laboratorio a
  *                       partir de STM32 con arquitectura ARM

******************************************************************************
*
  * @attention
  *
  *Partially automatic generated code by STM
  *
  *Code created for an educational purpose
  *

******************************************************************************
*
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ----------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
-*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
-*/
TIM_HandleTypeDef htim2;
```

```
TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
-*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
-*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
-*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  int I2;
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
```

```
  //Inicializamos el temporizador 2
  HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        I2 = HAL_GPIO_ReadPin(I2_GPIO_Port, I2_Pin);

        if (I2){
              HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, RESET);
              HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
              HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, SET);
              HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, RESET);

              HAL_Delay(2500);
        }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 84-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 500000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
```

```
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|LED2_Pin|LED1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LED4_Pin|LED3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin LED2_Pin LED1_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|LED2_Pin|LED1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : I2_Pin */
  GPIO_InitStruct.Pin = I2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(I2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : I1_Pin */
  GPIO_InitStruct.Pin = I1_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(I1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : P1_Pin P2_Pin */
  GPIO_InitStruct.Pin = P1_Pin|P2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : LED4_Pin LED3_Pin */
  GPIO_InitStruct.Pin = LED4_Pin|LED3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

}

/* USER CODE BEGIN 4 */
//Rutina de interrupciones
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	//dependiendo del dispositivo que haya activado la inerrupción hacemos
una acción u otra
	if (GPIO_Pin == P1_Pin){
		HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, SET);
	}
	else if(GPIO_Pin == P2_Pin){
		HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET);
	}
	else if(GPIO_Pin == I1_Pin){
		HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, RESET);
		HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, SET);
	}
}

//Timer desbordado
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
```

```
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

Anexo D: Código de ejemplo 4

- Archivo main.c:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body
  * @author               : Ulric Rossell Currius
  * @concept        : TFG, diseño e implementación de kit de laboratorio a
  *                       partir de STM32 con arquitectura ARM

******************************************************************************
*
  * @attention
  *
  *Partially automatic generated code by STM
  *
  *Code created for an educational purpose
  *

******************************************************************************
*
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
-*/
#include "main.h"

/* Private includes ----------------------------------------------------------
-*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
-*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
-*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
-*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
-*/
ADC_HandleTypeDef hadc1;
```

```
DMA_HandleTypeDef hdma_adc1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint32_t adc_value[3];
float temp;
float lm35;
float gp2y;
float gp2yV;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
```

```
  MX_USART2_UART_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start_DMA(&hadc1, adc_value, 3);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
```

```
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data
Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ENABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 3;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Configure for the selected ADC regular channel its corresponding rank
in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
  sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
```

```
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
#define V25 0.76
#define pendiente 0.0025
#define VSENSE 3.3/4095

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
 temp = ((VSENSE*adc_value[2] - V25) / pendiente) + 25;
 lm35 = (VSENSE*adc_value[0])/0.01;
 gp2yV = (VSENSE*adc_value[1]);
 if (gp2yV<1){
        gp2y = 1/(0.03666666667*gp2yV-1/600);
 }else{
        gp2y = 1/(0.05*gp2yV-0.015);
 }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
```

```
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```