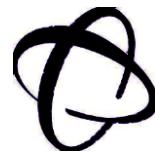




UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN INGENIERÍA
ELECTRÓNICA INDUSTRIAL



TRABAJO FIN DE GRADO

Diseño de placa de entrenamiento basada en una FPGA configurable mediante software libre.

Curso académico 2019/2020

Presentado por:

D. Alejandro Sánchez Doblado

Tutores:

Prof. D^a. Encarnación Castillo Morales

Prof. D. Luis Parrilla Roure



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

Diseño de placa de entrenamiento basada en una FPGA configurable mediante software libre.

Autor: Alejandro Sánchez Doblado

Directores: Encarnación Castillo Morales

Luis Parrilla Roure

Departamento: Electrónica y Tecnología de Computadores

Palabras clave:

FPGA, microcontrolador, PCB, IceZUM Alhambra, IceStudio, Open Source, STM32, STM32CubeIDE, KiCad.

Resumen:

A lo largo de esta memoria se expone el proceso de diseño de una placa entrenadora open source en la que coexisten un microcontrolador y una FPGA, así como distintos periféricos. Para ello se hará uso de herramientas, también de código libre, como KiCad para el diseño de la PCB e IceStudio para la configuración de la FPGA. El dispositivo se ha diseñado con la idea principal de ser utilizado para el control de drones aéreos mediante visión artificial, aunque manteniendo siempre su versatilidad como objetivo principal.



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

Training board design based on an FPGA configurable through open source software.

Author: Alejandro Sánchez Doblado

Directors: Encarnación Castillo Morales

Luis Parrilla Roure

Department: Electrónica y Tecnología de Computadores

Key words:

FPGA, microcontroller, PCB, IceZUM Alhambra, IceStudio, Open Source, STM32, STM32CubeIDE, KiCad.

Abstract:

Throughout this report, the design process of an open source training board in which a microcontroller and an FPGA coexist, as well as different peripherals, is presented. For this purpose, open source tools will be used, such as KiCad for the design of the PCB and IceStudio for the configuration of the FPGA. The device has been designed with the main idea of being used for the control of air drones through artificial vision, but always keeping its versatility as a main objective.



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

AUTORIZACIÓN DE LECTURA DE TRABAJO FIN DE CARRERA

Dña. **Encarnación Castillo Morales** y D. **Luis Parrilla Roure** profesores del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada, como director/es del Trabajo Fin de Grado titulado "**Diseño de placa de entrenamiento basada en una FPGA configurable mediante software libre.**" y realizado por el alumno **D. Alejandro Sánchez Doblado**.

CERTIFICA/N: que el citado Trabajo Fin de Grado, ha sido realizado y redactado por dicho alumno y autorizan su presentación.

Granada,

Fdo. **Encarnación Castillo Morales**

Fdo. **Luis Parrilla Roure**



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

AUTORIZACIÓN DE DEPÓSITO EN LA BIBLIOTECA

Yo, D. Alejandro Sánchez Doblado con DNI **47565002J**, autor del Trabajo Fin de Grado titulado "**Diseño de placa de entrenamiento basada en una FPGA configurable mediante software libre.**" realizado en la Universidad de Granada

DECLARO: explícitamente que asumo la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

AUTORIZO: al depósito de dicho Trabajo en la Biblioteca de la Universidad de Granada, y de la visualización a través de Internet.

Granada,

A handwritten signature in black ink, appearing to read "Alejandro Sánchez Doblado".

Fdo. D. Alejandro Sánchez Doblado

Agradecimientos

En primer lugar, me gustaría agradecer a mi madre por darme siempre fuerzas para seguir adelante, y a mi familia en general por darme la oportunidad y el apoyo necesario para terminar la titulación.

A mis amigas de la infancia, Paloma y Lucía. A mis amigos de Granada, en especial a Marcos, Martín, Carmen, Mireia, Victoria y Paula. A mis amigos del Erasmus, especialmente a Andrés. Por estar siempre en los buenos y en los malos momentos y por hacer de estos años de universidad, inolvidables.

Por último, a mis tutores Encarnación y Luis por su paciencia, permitirme aprender sobre el tema y ayudarme en todo lo posible.

Índice de contenido

1.	Introducción	21
1.1.	Estructura de la memoria	21
1.2.	Objetivos	21
1.3.	Motivación	21
1.4.	Metodología y planificación.....	22
2.	Estado del arte.....	24
2.1.	Idea inicial de diseño	24
2.2.	Open Source.....	26
2.3.	FPGAs	27
2.3.1.	Introducción a las FPGAs	27
2.3.2.	FPGAs open source.....	29
2.4.	Microcontrolador.....	32
2.4.1.	Introducción a los microcontroladores	32
2.5.	Diferencias y similitudes entre un microcontrolador y una FPGA.....	37
3.	Componentes y herramientas software.....	40
3.1.	Lattice ICE40.....	40
3.1.1.	iCE40HX4k.....	41
3.2.	IceStudio	45
3.3.	STM32	46
3.3.1.	ARM Cortex.....	47
3.3.2.	Familia de microcontroladores STM32	47
3.3.3.	STM32F091VC.....	50
3.3.4.	STM32F4DISCOVERY.....	54
3.4.	Software para STM32	56
3.4.1.	STM32CubeIDE	56
3.4.2.	Arduino IDE.....	58
3.5.	Introducción al diseño de PCBs.....	60
4.	Implementación.....	63
4.1.	Arquitectura del sistema.....	63
4.2.	Diseño de la FPGA y sus periféricos.....	65
4.2.1.	Chip FPGA	65

4.2.2.	ADC	69
4.2.3.	VGA.....	73
4.3.	Diseño del microcontrolador y sus periféricos	77
4.3.1.	Chip STM32.....	77
4.3.2.	Tarjeta SD	81
4.4.	Programación de los dispositivos	83
4.5.	Diseño SRAM y bus de comunicación.....	91
4.6.	Alimentación global del sistema.....	104
4.6.1.	Análisis de consumo	104
4.6.2.	Conector USB.....	105
4.6.3.	Regulador de tensión.....	108
4.7.	Placa de circuito impreso.....	114
4.8.	Producto final.....	120
5.	Conclusiones y trabajo futuro	126

Índice de figuras

Figura 1: Diagrama de Gantt de la planificación del TFG	22
Figura 2: Diagrama de bloques que representa la implementación de un controlador PID para un robot balancín usando una FPGA libre y un microcontrolador [2].	25
Figura 3: Estructura de una FPGA [6]	27
Figura 4: Celda lógica de una FPGA [6].....	28
Figura 5: IceZUM Alhambra II [13].....	30
Figura 6: Logotipo FPGAwars [14]	31
Figura 7: Componentes de un microcontrolador [16].....	32
Figura 8: Arquitectura Von Neumann [17]	33
Figura 9: Arquitectura Harvard [18]	34
Figura 10: Protocolo de comunicación UART [19].....	35
Figura 11: Protocolo de comunicación I2C [19]	36
Figura 12: Protocolo de comunicación SPI [19].....	36
Figura 13: Funcionalidad FPGA y microcontrolador.....	37
Figura 14: Dispositivo iCE40 [21]	41
Figura 15: Diagrama de bloques de un PLB [21]	42
Figura 16: Par de salida diferencial [22]	43
Figura 17: iCE40HX4K-TQ144 [23]	44
Figura 18: Ejemplo de un multiplexor en IceStudio	45
Figura 19: Interior de un bloque en IceStudio.....	46
Figura 20: Clasificación de las subfamilias STM32 [31]	49
Figura 21: Interfaz ST MCU Finder.....	50
Figura 22: Diagrama de bloques STM32F091VC [33]	51
Figura 23: ST-Link V1 (izquierda) y V2 (derecha) [35]	53
Figura 24: Visualización del contenido en la memoria de la STM32F4DISCOVERY mediante ST-Link Utility.	54
Figura 25: STM32F4DISCOVERY [23]	55
Figura 26: Diagrama de bloques STM32F4DISCOVERY [34].....	56
Figura 27: Configuración de pines en STM32CubeIDE	57
Figura 28: Configuración del reloj en STM32CubeIDE	58
Figura 29: Blinking led en el Arduino IDE para la STM32F4DISCOVERY	59
Figura 30: Tecnología THT (izquierda) y SMD (derecha) [40].....	60
Figura 31: Diagrama de flujo de trabajo en KiCad [44]	62
Figura 32: Diagrama de bloques de la placa entrenadora	64
Figura 33: Requerimientos de alimentación para la FPGA.....	66
Figura 34: Esquemático para los interruptores y LEDs de la FPGA	67
Figura 35: Esquemático de los pines de propósito general de la FPGA	67
Figura 36: Esquemático de la FPGA de Lattice	68
Figura 37: Esquemático del ADS7924. [46]	69
Figura 38: Encapsulado del ADS7924 [54].....	70

Figura 39: Diagrama de transición de código en el ADS7924	71
Figura 40: Esquemático del ADC en KiCad.....	72
Figura 41: Conector VGA D-SUB 15 [47].....	73
Figura 42: Circuito conversor R-2R para la señal RED	74
Figura 43: Temporización de la señal HSync en VGA de 640x480 pixeles [49]	76
Figura 44: Regiones activa y no activas de una pantalla VGA [50].....	76
Figura 45: Esquemático para el VGA en KiCad	77
Figura 46: Esquemático de alimentación para el microcontrolador.....	78
Figura 47: Esquemático del oscilador para el microcontrolador	79
Figura 48: Esquemático del microcontrolador en KiCad	80
Figura 49: Arquitectura interna de una tarjeta SD [53].....	81
Figura 50: Slot para tarjetas microSD [55]	82
Figura 51: Esquemático en KiCad para el slot microSD	83
Figura 52: Diagrama de bloques del FTD2232H [53].....	84
Figura 53: Configuración SPI Master [54]	85
Figura 54: Esquemático de la memoria Flash en KiCad [59]	86
Figura 55: Diagrama de selección del bootloader para el STM32F091VC [60]	87
Figura 56: Pines para la selección del dispositivo a programar	88
Figura 57: Esquemático del FTDI en Kicad.....	89
Figura 58: Esquemático de la alimentación para el FTDI en KiCad	90
Figura 59: Contenido de la EEPROM del FTDI	90
Figura 60: Diagrama de bloques de la memoria SRAM [61]	91
Figura 61: pinout de la memoria SRAM [61]	92
Figura 62: Esquemático de la memoria SRAM en KiCad	93
Figura 63: Etapa de entrada de un circuito CMOS [62].....	94
Figura 64: Transmisión bidireccional en un sistema de buses [62].....	94
Figura 65: Resistencia de pull-up en una línea de bus [62].....	95
Figura 66: Circuito de retención de buses [62]	96
Figura 67: Circuito simplificado del integrado de retención de buses [62]	97
Figura 68: Pinout del SN74ALVCH162827 [63].....	97
Figura 69: Circuito de retención de buses en KiCad	98
Figura 70: Módulo SRAM IS62WV51216BLL [65]	99
Figura 71: Código del microcontrolador para escribir en la SRAM	101
Figura 72: Configuración de la FPGA para lectura de la SRAM en IceStudio.	102
Figura 73: Contenido del bloque InOut.	102
Figura 74: Puerta Tri-estado [66].....	103
Figura 75: Pinout USB tipo C [67]	106
Figura 76: Esquemático del conector USB C en KiCad	107
Figura 77: Interior del circuito de protección ESD para USB [69]	107
Figura 78: Pinout del PAM2306 [70]	108
Figura 79: Diagrama de bloques PAM2306 [70].....	109
Figura 80: Implementación típica PAM2306 [70].....	109
Figura 81: Esquemático del PAM2306 en KiCad.....	111
Figura 82: Interruptor de la tensión para periféricos.....	112

Figura 83: Aplicación típica del LDO [71].....	112
Figura 84: LEDs indicadores de alimentación.....	113
Figura 85: Ejemplo de enrutado del ADC [46]	115
Figura 86: Antena para el oscilador del microcontrolador	116
Figura 87: Enrutado de la SRAM.....	116
Figura 88: Enrutado del par diferencial USB	117
Figura 89: Distintas secciones de la PCB	117
Figura 90: Diseño final simplificado de la PCB.....	118
Figura 91: Diseño final completo de la PCB.....	118
Figura 92: Generación de archivos Gerber.....	119
Figura 93: Visualización de los archivos Gerber	120
Figura 94: Modelo 3D de la placa (cara superior)	121
Figura 95: Modelo 3D de la placa (cara inferior).....	122
Figura 96: Modelo 3D de la placa (perfil)	122

Índice de tablas

Tabla 1: Comparativa de tipos de memorias.....	35
Tabla 2: Familia iCE40.....	40
Tabla 3: Función del pin BOOT0	52
Tabla 4: Pinout ADS7924	70
Tabla 5: VGA pinout.....	74
Tabla 6: Valores de RED para su correspondiente palabra digital	75
Tabla 7: Pinout tarjeta SD en modo SPI.....	82
Tabla 8: Descripción de las señales para la memoria Flash	85
Tabla 9: Descripción de pines del canal B para protocolo UART.....	87
Tabla 10: Descripción de pines de la memoria SRAM	92
Tabla 11: Tabla de verdad de la memoria SRAM	93
Tabla 12: Tabla de verdad del integrado de retención de buses	98
Tabla 13: Tabla de verdad para el módulo SRAM	99
Tabla 14: Consumo máximo en la alimentación de +3.3V	104
Tabla 15: Pinout del conector DX07S016JA1R1500	106
Tabla 16: Funciones especiales en los pines del microcontrolador	123
Tabla 17: Lista de materiales del proyecto simplificada	125
Tabla 18: Coste de fabricación de la PC.....	125

Índice de ecuaciones

Ecuación 1: Cálculo de las capacidades para el oscilador	78
Ecuación 2: Cálculo de la resistencia de pull-up.....	95
Ecuación 3: Cálculo de la bobina para el conversor	110
Ecuación 4: Cálculo de las resistencias para el conversor.....	110

Lista de acrónimos

- *ADC. Analog-To-Digital Converter.*
- *ASIC. Application-specific integrated circuit.*
- *BOM. Bills of Materials.*
- *CPLD. Complex Programmable Logic Device.*
- *DMA. Direct Memory Access.*
- *ECAD. Electronic Computer-Aided Design*
- *EEPROM. Electrically Erasable Programmable Read-Only Memory*
- *FPGA . Field Programmable Gate Array.*
- *GPIO. General-Purpose Input/Output.*
- *HDL. Hardware Description Language.*
- *I2C. Inter-Integrated Circuit.*
- *IDE. Integrated Development Environment.*
- *LDO. Low-Dropout regulator*
- *LED. Light-Emitting Diode.*
- *LQFP. Low-profile quad flat-package.*
- *LUT. Look-Up Table.*
- *PAL. Programmable Array Logic.*
- *PCB. Printed Circuit Board.*
- *PDR Power Down Reset.*
- *PID. Proporcional, Integrador, Derivador.*
- *PLB. Programmable Logic Blocks.*
- *PLL. Phase-locked loop.*
- *POR. Power on Reset.*
- *RAM. Random Access Memory.*
- *SD. Secure Digital*
- *SMT. Surface-Mount Technology.*
- *SPI. Serial Peripheral Interface.*
- *SWD. Single Wire Debugging.*
- *THT. Through-Hole Technology.*
- *TTL. Transistor-Transistor Logic*
- *VGA. Video Graphic Array*
- *VHDL. Very High-Speed Integrated Circuit Hardware Description Language*

1. Introducción

1.1. Objetivos

El objetivo general de este trabajo de fin de grado (en adelante, TFG) será diseñar una placa entrenadora basada en una FPGA configurable mediante software libre junto con un microcontrolador que pueda ser usada tanto por el departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada con fines educativos como por cualquier persona interesada en el diseño ya que tanto los componentes como el diseño de la placa serán de libre acceso.

Más concretamente, los objetivos de este trabajo son:

- Conocer en detalle los fundamentos de las FPGAs y de los microcontroladores, así como los ámbitos de aplicación de cada uno.
- Conocer y contribuir a la corriente, en auge, de herramientas software y hardware open source.
- Definir la arquitectura a implementar que mejor se aadecue tanto a los requisitos como a las limitaciones presentes.
- Estudio y elección de los componentes y herramientas apropiadas para el sistema.
- Diseño en esquemático de la placa entrenadora.
- Diseño de la placa de circuito impreso.
- Testeo de algunos componentes del sistema mediante el uso de placas de evaluación.

1.2. Motivación

La idea y motivación de este trabajo de fin de grado proviene, por una parte, de un TFG anterior en el que el estudiante tuvo que lidiar con un sistema que combinaba una FPGA y un microcontrolador para el control de un robot balancín. A raíz de dicho trabajo, el estudiante se percató de las limitaciones que presentaba el sistema con el que trabajaba y se propuso la elaboración de una placa de circuito impreso que solventara dichas limitaciones.

Por otra parte, se encuentra la motivación del propio estudiante de profundizar en el ámbito de los sistemas electrónicos digitales, concretamente en el diseño hardware e implementación software de sistemas programables, así como de colaborar y dar visibilidad a la comunidad open source.

1.3. Metodología y planificación

La metodología a seguir para la realización del trabajo fin de grado será la siguiente:

1. Primero se realizará un análisis rápido del problema y de las posibles soluciones.
2. Una vez detectado las necesidades del sistema se estudiará la arquitectura a implementar, así como los componentes que la integran.
3. El siguiente paso será, mediante placas desarrolladoras existentes en el mercado, testear los componentes y la sinergia entre ellos.
4. Una vez definido el sistema y elegidos los componentes definitivos, se procederá a diseñar la placa mediante software de diseño de placas de circuito impreso.
5. Por último, se realizará un análisis de los objetivos cubiertos por el diseño, problemas encontrados durante su desarrollo y posibles mejoras a implementar.

La memoria de este trabajo se realizará en paralelo con el diseño de la placa según se vaya avanzando.

El trabajo de fin de grado estaba siendo desarrollado cuando se declaró el estado de alarma nacional debido al COVID-19. Es por ello que el TFG tuvo que ser adaptado, eliminando el trabajo en laboratorio y la fabricación de la placa.

Con respecto a la forma de trabajar, el estudiante tratará de avanzar por su cuenta siempre que las circunstancias lo permitan. Sin embargo, al ser un trabajo de diseño sin referencias, habrá puntos donde el estudiante necesite intervención por parte de los tutores, ya sea para elegir un componente definitivo o para ayudar en el diseño global de la placa. Por ello se tratará de quedar una vez por semana, ya sea por correo o videollamada, para ir comentando el avance y las trabas encontradas.

En la Figura 1 podemos observar la planificación cronológica seguida por el estudiante para el desarrollo del trabajo de fin de grado.

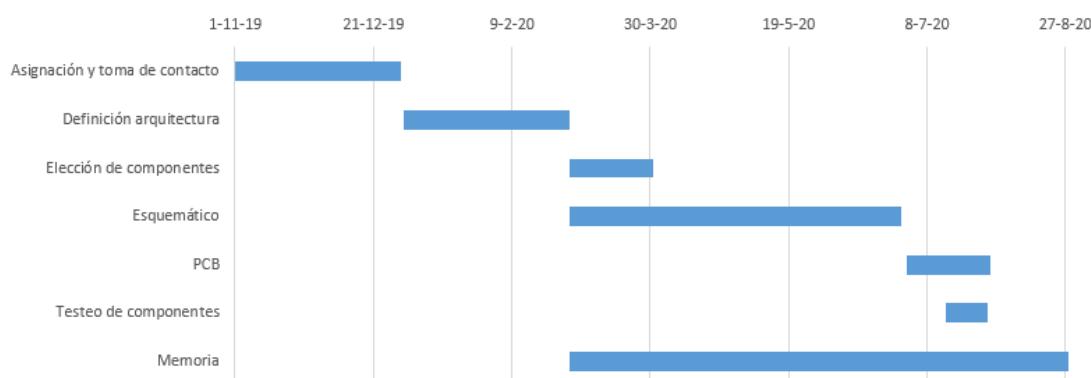


Figura 1: Diagrama de Gantt de la planificación del TFG

Al ser un proyecto *open source* todo el material será alojado en GitHub [1], para que cualquiera que lo desee pueda ver, descargar y modificar tanto los archivos del proyecto como la documentación.

1.4. Estructura de la memoria

La memoria de este TFG se estructurará en 5 capítulos acordes con el proceso de diseño de la placa:

- **Capítulo 1:** Breve introducción del trabajo a realizar, haciendo hincapié en los objetivos y la motivación del TFG, así como en la metodología y organización del mismo.
- **Capítulo 2:** Desarrollo de la idea del proyecto, donde, además, se explicará los principios básicos de las FPGAs y de los microcontroladores, así como sus respectivas aplicaciones.
- **Capítulo 3:** Descripción de los componentes principales elegidos para la implementación de la placa, así como de las herramientas software a utilizar.
- **Capítulo 4:** Estudio e implementación de la arquitectura final del sistema.
- **Capítulo 5:** Comentarios acerca del trabajo desarrollado, así como de posibles mejoras, conclusiones y repaso de objetivos logrados.

2. Estado del arte

En este capítulo se comentará la idea de diseño, así como una breve introducción a los elementos más relevantes que lo componen: la FPGA y el microcontrolador.

2.1. Idea inicial de diseño

Una placa entrenadora se conoce como un dispositivo de propósito general que permita al usuario probar ciertas configuraciones de manera sencilla y aprender sobre el tema, haciéndola muy utilizada en el ámbito educativo. Es por ello que para el diseño de nuestro dispositivo tendremos siempre en cuenta la versatilidad de la misma y que al ser un proyecto open source, el usuario interesado en ella siempre podrá modificarla a su gusto, quitando elementos que considere innecesarios para su uso y añadiendo aquellos que sí necesite.

Como ya se ha explicado en el apartado Motivación {1.2}, la idea de este TFG surgió a raíz de los problemas encontrados en TFG realizado anteriormente [2]. En dicho TFG se utilizó la placa de desarrollo basada en hardware libre IceZUM Alhambra II [3], diseñada por Eladio Delgado Mingorance en Pinos del Valle, Granada. Se hablará de dicha placa con más detalle en el apartado IceZUM Alhambra II {2.3.2.1} de este capítulo. El proyecto se basaba en la implementación de un controlador PID para un robot balancín usando una FPGA libre, así como de un microcontrolador, el ATMega328, presente en las famosas placas open source Arduino Uno [4].

El microcontrolador era el encargado de la adquisición de las señales mediante sensores, debido a la capacidad de estos dispositivos de ejecutar instrucciones secuenciales en bucle y a la gran comunidad que respalda a la placa Arduino Uno, la cual cuenta con una inmensa cantidad de sensores disponibles, baratos y fáciles de configurar gracias a las bibliotecas libres con las que cuenta.

Como se muestra en la Figura 2, la FPGA se encargaba de la implementación del controlador PID con las variables que recibía del microcontrolador y de controlar los motores que balanceaban al robot, gracias a la potencia de cálculo y nivel de paralelización con la que cuentan las FPGAs.

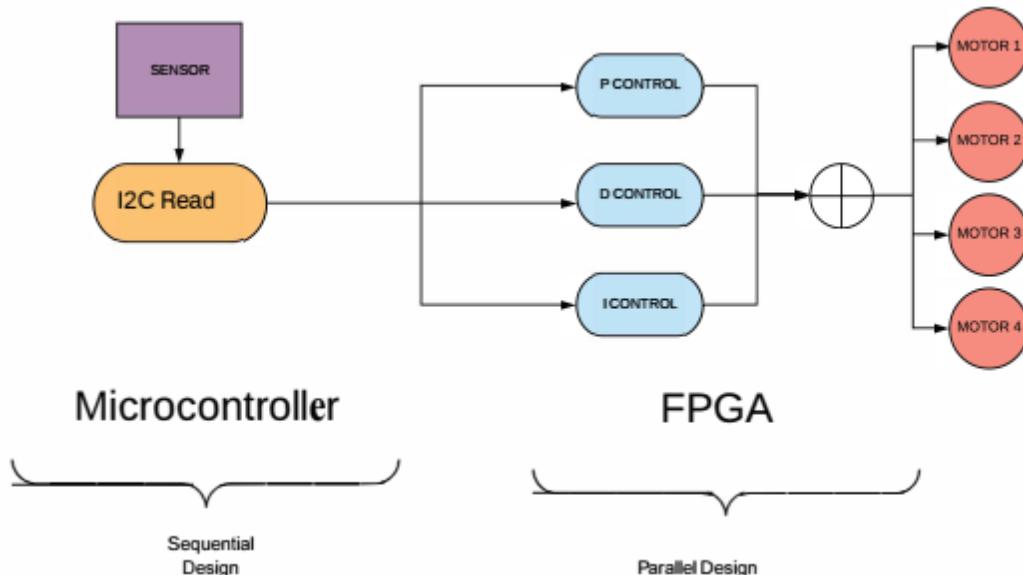


Figura 2: Diagrama de bloques que representa la implementación de un controlador PID para un robot balancín usando una FPGA libre y un microcontrolador [2].

De dicha aplicación se pudo observar la utilidad de un sistema que integre ambos dispositivos, obteniendo por un lado todas las ventajas de ambas y disminuyendo las carencias al complementarse la una a la otra. Sin embargo, el sistema contaba con una serie de fallos:

- Comunicación: Al contar con las dos placas discretas, el estudiante tuvo que implementar una forma de que ambas se pudiesen comunicar. Debido a la escasez de pines con las que dichas placas cuentan (20 cada una) y a que gran parte de ellos ya debían ser ocupados por sensores o motores, el estudiante optó por una comunicación serie unilateral, a pesar de la lentitud en la transmisión que esto conlleva.
- Pines: Como se acaba de comentar, uno de los principales problemas con los que contaba el sistema era la escasez de pines y la imposibilidad de ampliarlos por parte del usuario.
- Alimentación: Era necesaria 3 alimentaciones, una para la FPGA, otra para el microcontrolador y otra para los motores.

Además, el estudiante propuso el uso de este sistema para el control de drones mediante visión artificial, lo que provocó que se encontrase con una serie de trabas adicionales:

- Complejidad para la visualización de imágenes.
- Escasez de memoria.
- Microcontrolador con recursos muy limitados.

Teniendo en cuenta las limitaciones que se encontraron en el desarrollo del mencionado TFG, en el presente TFG se realiza el diseño de una placa que mantenga la coexistencia de una FPGA con un microcontrolador y que además sea capaz de suplir las carencias con las que dicho sistema se encontraba. Los objetivos generales que se intentan lograr se pueden resumir en:

1. Aumento del número de pines de la placa, tanto de propósito general como de comunicación mediante diversos protocolos.
2. Comunicación en paralelo, rápida y bidireccional entre la FPGA y el microcontrolador.
3. Microcontrolador con un mayor desempeño.
4. Mejora de la memoria del sistema, tanto de alta velocidad como de alta capacidad.
5. Alimentación unificada.
6. Puerto VGA para la visualización y depuración.

Finalmente, tras un estudio más detallado de las necesidades y posibilidades se ha optado por implementar una única placa que contenga:

- Una FPGA y un microcontrolador de 32 bits.
- Una memoria SRAM conectada a ambos dispositivos que sirva tanto de almacenamiento masivo de datos como de buffer de comunicación entre la FPGA y el microcontrolador.
- El mayor número de pines de ambos dispositivos posible.
- Un puerto VGA y una ranura para tarjeta microSD.
- Una sola alimentación para todos los componentes, así como capacidad para alimentar otros dispositivos periféricos.

2.2. Open Source

Open source (en español, código abierto) [5] es un movimiento tecnológico que comenzó en el siglo XX como resultado de una tensión creciente a la propiedad intelectual y al *Copyright*, que limita y encarece, según el movimiento, el libre desarrollo de productos.

Originalmente centrado en el software, el movimiento proponía la creación de software cuyo código fuente y otros derechos fuesen publicados bajo una licencia de código abierto o bajo dominio público que permitiese al usuario utilizar, cambiar y redistribuir el software a cualquiera y para cualquier propósito. Actualmente el movimiento cuenta con una relevancia importante, con aplicaciones en campos tan diversos tales como los

sistemas operativos (GNU/Linux), desarrollo CAD (FreeCAD), ofimática (Apache OpenOffice), etc.

A medida que el movimiento ha ido ganando aliados se ha extendido a otras ramas de la tecnología como es el hardware. En este ámbito el desarrollador proporciona los esquemáticos, lista de materiales, placa de circuito impreso, código fuente del lenguaje de descripción hardware.

Esta será pues, la filosofía a seguir en este trabajo de fin de grado, en el que se alojará en GitHub todo el material referente al proyecto, para que cualquier usuario interesado lo utilice y modifique a su interés.

2.3. FPGAs

2.3.1. Introducción a las FPGAs

Una FPGA [6] es un dispositivo programable que contiene bloques lógicos cuya interconexión y funcionalidad puede ser configurada en el momento mediante un lenguaje de descripción hardware. Permite reproducir desde funciones sencillas como puertas lógicas a funciones complejas como redes neuronales o simulaciones de arquitecturas de microcontroladores.

Las FPGAs se consideran una evolución de las PAL y los CPLD y por debajo de los ASIC por ser un dispositivo de propósito más general y por lo tanto ser más lento y con un mayor consumo. Sin embargo, las FPGAs cuentan con una gran cuota de mercado hoy en día gracias a que son reprogramables y que por ello permiten reducir costes y tiempos de desarrollo para aplicaciones en pequeñas cantidades y, por tanto, sus perspectivas de futuro son bastante prometedoras.

La arquitectura de una FPGA consiste en una serie de bloques lógicos, puertos I/O y buses de interconexión, como se muestran en la Figura 3.

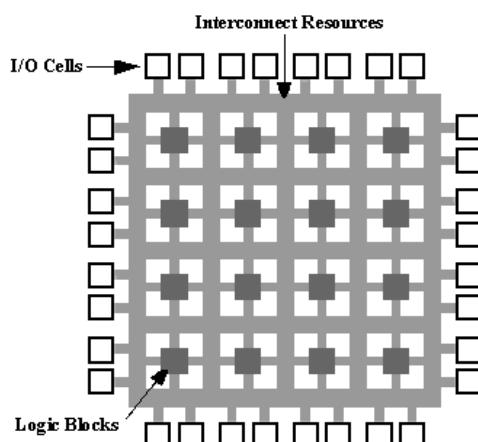


Figura 3: Estructura de una FPGA [6]

Normalmente, los bloques lógicos constan de una serie de celdas lógicas compuestas por una LUT de 4 entradas, un sumador completo y un biestable tipo D. En la Figura 4 podemos observar el contenido de una celda lógica.

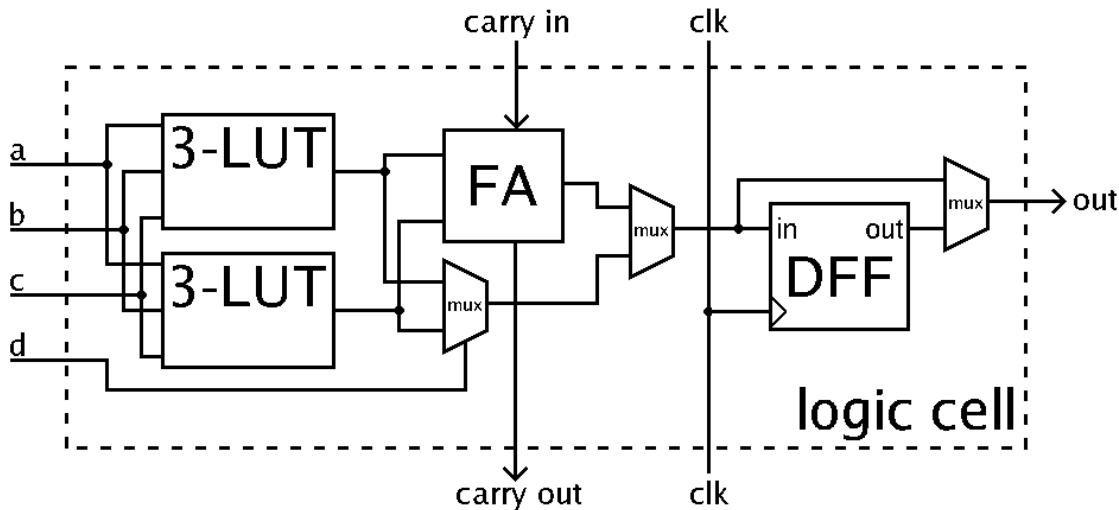


Figura 4: Celda lógica de una FPGA [6]

Por lo tanto, a la hora de programar la FPGA con una aplicación específica lo que se hace es elegir el contenido de las LUTs, así como las conexiones entre las diferentes celdas y bloques.

Para seleccionar el dispositivo FPGA más conveniente para nuestra aplicación tendremos que, principalmente, tener en cuenta el número de bloques lógicos de los que dispone el dispositivo para asegurarnos de que se pueda implementar en dicha placa.

Algunas de las FPGAs más recientes en el mercado cuentan con algunas funciones de alto nivel que sean muy frecuentes ya embebidas. Con ello se consigue reducir el espacio que requieren y maximizar la velocidad de cómputo en comparación a si se realizasen de forma independiente con bloques lógicos. Algunas de estas funciones son los multiplicadores, lógica I/O y memorias embebidas.

Las FPGAs cuentan con una o varias señales de reloj con ruteados y pistas específicas para transportar estas señales de forma que lleguen a los componentes con el menos desfase posible. También cuentan con PLLs para sintetizar nuevas frecuencias de reloj y atenuar las desviaciones de periodicidad de la señal (lo que se conoce en inglés como “*jitter*”).

Algunos de los campos de aplicación de las FPGAs son los siguientes:

- Procesamiento de imagen y video
- Audio
- Instrumentación científica

- Seguridad
- Prototipado ASIC
- Comunicaciones
- Computación de altas prestaciones

Más adelante, en Diferencias y similitudes entre un microcontrolador y una FPGA {2.5}, se desarrollará el motivo por el cual dichas aplicaciones son óptimas para esta clase de dispositivo.

En el mercado, actualmente, los principales fabricantes de FPGAs son Xilinx [7] y Altera (hoy en día una subsidiaria de Intel) [8], controlando entre ambas el 90% de la cuota de mercado.

Para definir el comportamiento de una FPGA, los usuarios recurren a un lenguaje de descripción de hardware (HDL). Actualmente los principales lenguajes de descripción hardware son Verilog [9] y VHDL [10].

VHDL fue creado por el departamento de defensa de los Estados Unidos de América en la década de 1980. Una de las características principales de dicho lenguaje es que cada tipo de dato debe ser predefinido y es más verboso que Verilog.

Verilog fue inventado por Phil Moorby en 1985 para Gateway Design Automation que más tarde fue comprada por Cadence Design. Cuenta con una sintaxis similar al lenguaje de programación C. Debido al éxito de VHDL sobre Verilog, Cadence decidió hacer de Verilog un lenguaje abierto y disponible para la estandarización.

Ambos lenguajes de descripción hardware son de alto nivel por lo que se necesita de un compilador que lo traduzca a un lenguaje que pueda ser entendido por la FPGA, el llamado bitstream. El bitstream es una secuencia de bits que configura la FPGA de acuerdo con la funcionalidad especificada en HDL. El paso de HDL al bitstream es realizado por un compilador (comúnmente denominado *toolchain*). Actualmente, el toolchain de las dos empresas (Xilinx y Altera) son propiedad intelectual de la compañía por lo que no se tiene acceso a su código fuente y obligando a adquirir su software si queremos programar su hardware. Según las compañías se mantiene en secreto para proteger el diseño del consumidor de ingeniería reversa y prevenir dañar las FPGAs con bitstreams inválidos. Sin embargo, gran parte de la comunidad considera que es una estrategia para evitar que las estructuras de sus chips puedan ser copiadas además lucrarse con la venta de las herramientas software.

2.3.2. FPGAs open source

Debido a lo comentado anteriormente, el desarrollo de estos dispositivos ha sido, durante años, exclusivo por parte de dichas compañías lo que ha podido frenar el nivel de desarrollo al que han podido llegar.

Sin embargo, en marzo de 2015 Clifford Wolf liberó el toolchain de la FPGA ICE40 [11] de la compañía Lattice mediante ingeniería inversa en lo que se conoce como proyecto

Icestorm [12]. Esto provocó que todo el ciclo de diseño de una FPGA se liberase permitiendo a cualquier usuario diseñar, programar y cargar una FPGA.

Desde entonces la comunidad ha diseñado tanto software como hardware permitiendo el libre desarrollo de las FPGAs que no para de crecer día a día.

2.3.2.1. IceZUM Alhambra II

Para el diseño de nuestra placa nos basaremos en la IceZUM Alhambra II fabricada por Alhambra Bits [13], en Granada. En la Figura 5 se muestra una imagen de la IceZUM Alhambra II. En el GitHub del proyecto [3] podemos encontrar toda la información necesaria al respecto de la placa:

- Esquemáticos
- Diseño de PCB
- BOM
- Ejemplos de código

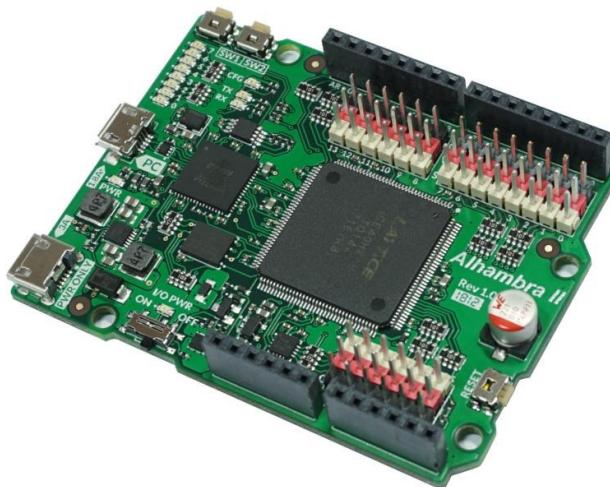


Figura 5: IceZUM Alhambra II [13]

Entre las características principales de la placa destacan las siguientes:

- FPGA ICE40HX4K.
- Compatible con IceStorm *toolchain* e IceStudio.
- Formato idéntico a Arduino Uno (compatible, por lo tanto, con sus *shields*).
- Programable mediante USB gracias al chip FT2232H.
- 32Mb de memoria flash.
- 20 GPIO pins.

- Un ADC de 12 bits y 4 canales.
- Oscilador de 12MHz.
- 2 *switches* de propósito general.
- Botón *reset*.
- Alimentado mediante micro USB (5V hasta 4.8A)

Siendo ésta un gran punto de partida para nuestro diseño y gracias a que es un proyecto open source, aprovecharemos toda la documentación con la que cuenta para así aprender a diseñar un sistema de esta complejidad de cero además de reutilizar partes del diseño para nuestra placa entrenadora. Además, la usaremos para realizar distintas pruebas sobre nuestro diseño.

2.3.2.2. FPGAwars

FPGAwars [14] (Figura 6) es un proyecto para acercar el mundo de las FPGAs libres y la electrónica digital en general a todo tipo de usuarios. La plataforma, que reside en GitHub, aporta material de libre acceso para adquirir conocimientos en diferentes campos, como FPGAs, Verilog y circuitos digitales. También realizan diversos talleres y charlas de divulgación.



Figura 6: Logotipo FPGAwars [14]

Parte del equipo de FPGAwars son los responsables del diseño, fabricación y comercialización de la IceZUM Alhambra así como del editor de código para FPGAs libres que veremos más adelante, IceStudio.

2.4. Microcontrolador

2.4.1. Introducción a los microcontroladores

En este apartado se introducirá el concepto de microcontrolador, así como algunos de sus componentes, características y aplicaciones más relevantes. Un microcontrolador [15] es un circuito integrado programable, capaz de ejecutar una serie de órdenes almacenadas en su memoria. Los microcontroladores suelen estar definidos por el número de bits y la frecuencia de reloj a la que operan. En la Figura 7 se muestran los componentes de un microcontrolador, que incluye las tres principales unidades funcionales de un computador:

- Unidad central de procesamiento: que interpreta las instrucciones del programa mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema
- Memoria: que almacena datos durante algún periodo de tiempo.
- Periféricos de entrada y/o salida (I/O): que es capaz de interactuar con los elementos externos del sistema de forma bidireccional.

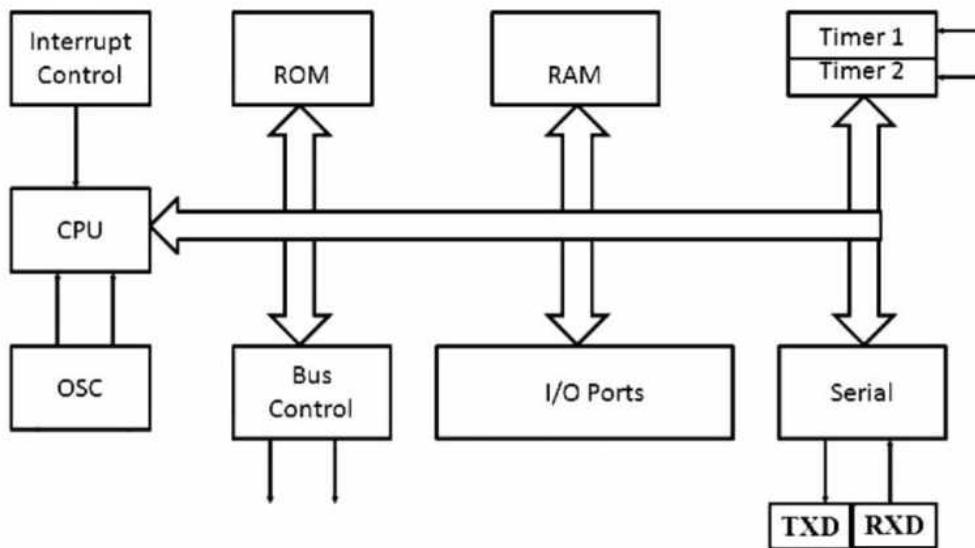


Figura 7: Componentes de un microcontrolador [16]

A la hora de elegir un microcontrolador es muy importante conocer las especificaciones que requiere la aplicación, para poder así elegir un microcontrolador que optimice el coste y su consumo de energía. El hecho de que contenga todos los circuitos necesarios para funcionar lo diferencian de un microprocesador, el cual necesita de elementos externos de memoria y periféricos de I/O para trabajar.

Un microcontrolador, como cualquier otra computadora, está basada en una de las arquitecturas descritas a continuación:

Arquitectura Von Neumann:

La arquitectura Von Neumann [17] (Figura 8) utiliza la misma memoria tanto para las instrucciones como para los datos, por lo que no puede realizar simultáneamente una búsqueda de instrucciones y una operación de datos.

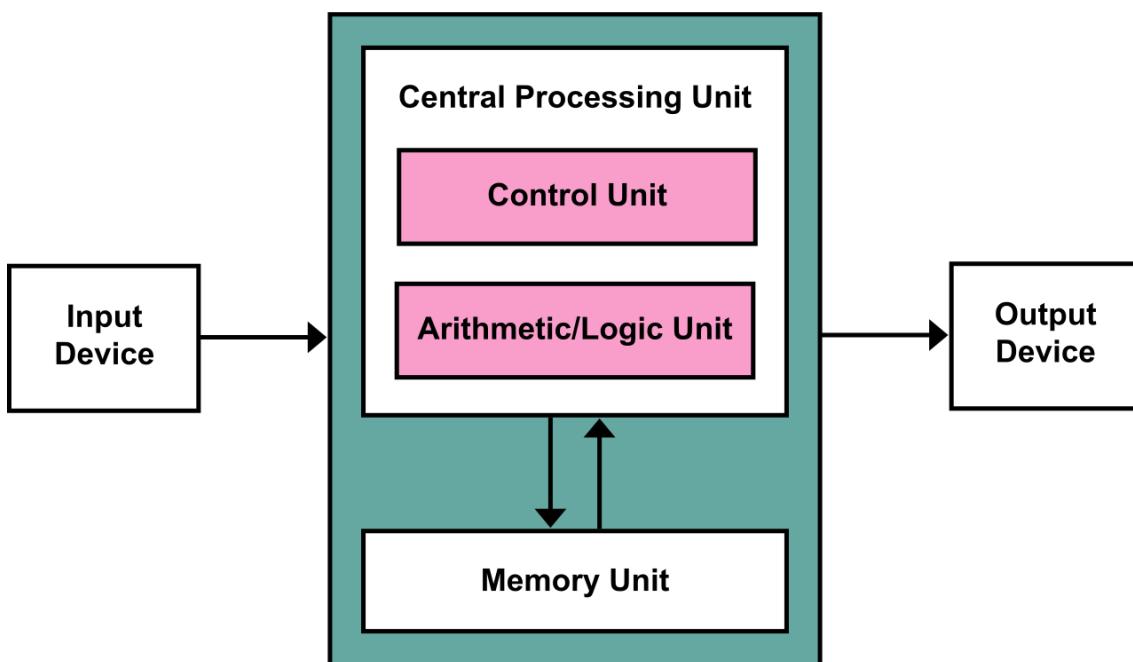


Figura 8: Arquitectura Von Neumann [17]

Arquitectura Harvard:

La arquitectura Harvard [18] (Figura 9) es una sucesora de las Von Neumann que trata de corregir el cuello de botella que se forma al compartir la memoria para datos e instrucciones. Para ello, esta arquitectura dispone de una memoria para datos y otra para instrucciones, cada una con su respectivo bus. Además, cada bus está segregado en datos, direcciones y control. La desventaja de esta arquitectura es, por tanto, el número de líneas de I/O que requiere el microcontrolador. Es la arquitectura más usada hoy en día, tanto en computadoras como en microcontroladores.



Figura 9: Arquitectura Harvard [18]

Para el diseño de nuestra placa se ha creído conveniente entrar un poco más en detalle en dos elementos que, aunque no son exclusivos de los microcontroladores, sí que están muy presente en ellos. Dichos elementos son: la memoria y los periféricos.

Memoria:

Como ya hemos comentado, la memoria se encarga de almacenar, ya sean datos o instrucciones, durante un periodo de tiempo. Existen distintos tipos de memorias cuyas características tales como capacidad, velocidad y volatilidad, difieren. A continuación, se explicarán brevemente las más relacionadas con el diseño de la placa:

- Memoria RAM: perteneciente al grupo de almacenamiento secundario, por detrás de las cachés. Cuentan con una alta velocidad de acceso, un almacenamiento bastante limitado y pierden la información cuando dejan de estar alimentadas. Pueden ser estáticas (SRAM, no requiere de circuito de refresco) o dinámicas (DRAM, sí necesitan de circuito de refresco). En microcontroladores se suelen utilizar para almacenar datos.
- Memoria EEPROM: sucesora de las EPROM que permite, como su nombre indica, borrarse eléctricamente sin necesidad de un aparato de rayos ultravioleta. Son memorias no volátiles de sólo lectura. Cuentan con una velocidad mucho menor que las memorias RAM, un coste muy inferior y un almacenamiento similar. Frecuentemente utilizadas en microcontroladores para el almacenamiento de las instrucciones del programa.
- Memoria flash: evolución de las EEPROM, permiten lectura y escritura de múltiples posiciones de memoria en la misma operación mediante impulsos eléctricos. Mayor velocidad que las EEPROM, no volátiles y de almacenamiento

superior. Se suelen utilizar en microcontroladores como medio de almacenamiento masivo de datos.

La Tabla 1 muestra una comparativa entre las distintas clases de memorias mencionadas.

	Categoría	Capacidad	Velocidad	Coste	Volatilidad
SRAM	R/W	Mb	ns	\$\$\$	Sí
EEPROM	R	Mb	us	\$\$	No
Flash	R/W	Gb	ns~us	\$	No

Tabla 1: Comparativa de tipos de memorias

Periféricos:

Se denomina periférico al dispositivo auxiliar e independiente conectado a la unidad central de procesamiento. Pueden ser de entrada, salida o entrada y salida. En nuestro sistema, los más relevantes serán:

- Entrada/Salida de propósito general (GPIO): Puertos configurados en el microcontrolador como entradas o salida y que se dejarán libres para que el usuario los use a su conveniencia. Ejemplos: leds, motores, sensores...
- Puertos de comunicación: permiten transmitir y recibir señales en forma de datos del exterior siguiendo algún protocolo de comunicación preestablecido [19]. En este ámbito, los protocolos más comunes son:
 - UART: Comunicación serie asíncrona presente en todos los microcontroladores que permiten al dispositivo comunicarse con dispositivos auxiliares de forma sencilla (Figura 10). Cuenta con dos líneas, una de recepción (RX) y otra de transmisión (TX). Puede operar en modo Simplex, Half Duplex o Full Duplex. Es lento y sólo permite un máster y un esclavo. Envía 8 bits por ronda.



Figura 10: Protocolo de comunicación UART [19]

- I2C: Comunicación serie síncrona usada comúnmente para comunicar el microcontrolador con módulos y sensores. Requiere de dos líneas, una de reloj (CLK) y una de datos (CDA). Permite varios masters y mantener hasta 128 esclavos por línea. Primero manda una señal de inicio, luego indica a quién se dirige, la dirección y por último el dato seguido de una condición de parada, tal y como se indica en la Figura 11.

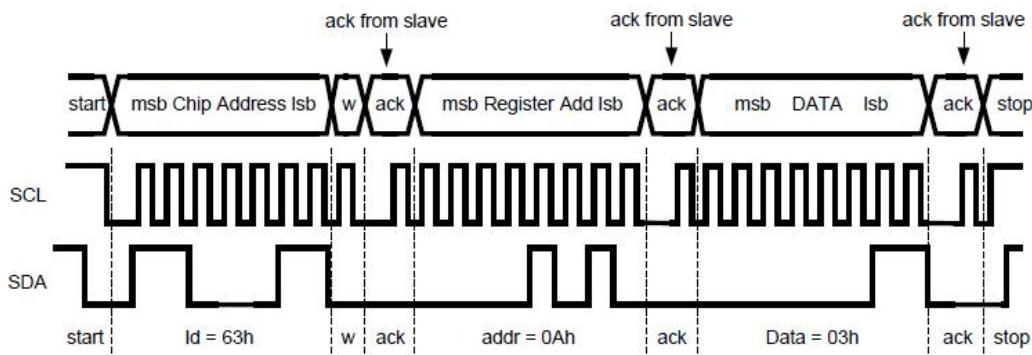


Figura 11: Protocolo de comunicación I2C [19]

- SPI: Protocolo de comunicación serie síncrona, similar al I2C, aunque más rápido. Opera en Full Duplex y requiere de 4 líneas. Permite varios esclavos, aunque un solo máster y la complejidad del sistema asciende con el número de dispositivos conectados. En la Figura 12 se incluye un ejemplo de funcionamiento del protocolo SPI.

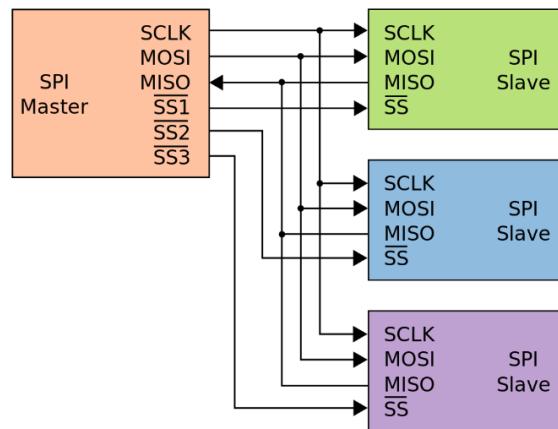


Figura 12: Protocolo de comunicación SPI [19]

Los microcontroladores son dispositivos muy versátiles por lo que, combinado con su bajo precio, hace que estén presentes en una gran variedad de aplicaciones hoy en día. A diferencia de una computadora, estos chips son utilizados para realizar una función

específica, por lo que son usados esencialmente en sistemas embebidos. Alguno de los campos de aplicación es:

- Electrónica de consumo: cámaras, teléfonos, microondas...
- Instrumentos médicos.
- Comunicaciones.
- Instrumentación y control de procesos.
- Industria automovilística.

2.5. Diferencias y similitudes entre un microcontrolador y una FPGA

A primera vista puede parecer que el microcontrolador y la FPGA tienen características muy parecidas. Ambos dispositivos reciben información del exterior mediante puertos de entrada y/o salida, almacenan datos en una memoria y opera con ellos mediante un bloque de procesamiento de datos (Figura 13).

Sin embargo, aunque tienen estas tres características en común, lo cierto es que una de ellas es completamente diferente en uno y en otro [20]. Hablamos pues, de la unidad de procesamiento.

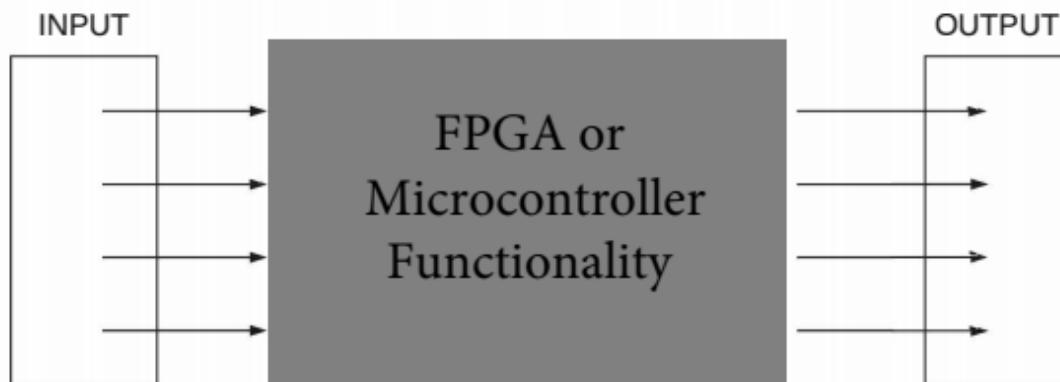


Figura 13: Funcionalidad FPGA y microcontrolador

Anteriormente hemos visto cómo funciona la unidad de procesamiento de una FPGA y la unidad de procesamiento de un microcontrolador. En el caso de la FPGA, vimos que la unidad de procesamiento está formada por una serie de bloques lógicos cuyas interconexiones se programan mediante software para obtener la implementación deseada, por tanto, tenemos una variación en el hardware del dispositivo. Sin embargo, en el caso del microcontrolador, se tiene un hardware fijo, el cual opera según unas instrucciones almacenadas previamente en su memoria.

Dicha variación en la arquitectura de la unidad de procesamiento provoca grandes cambios en el comportamiento del sistema:

- Los microcontroladores usan normalmente lenguajes de programación de alto nivel eficientes como C o C++. Las FPGAs utilizan lenguaje de descripción hardware como VHDL o Verilog.
- La potencia de un microcontrolador se define en gran parte por su frecuencia de reloj máxima, es decir, el número de instrucciones que puede ejecutar en un periodo de tiempo dado. Se podría decir que está limitada por el tiempo. La potencia de las FPGAs, sin embargo, viene dada por el número de bloques lógicos que contiene. Es decir, hay un límite físico a la hora de implementar una aplicación en una FPGA determinada. Se podría decir entonces que está limitada por su espacio.
- Las FPGAs, como su nombre indica, son dispositivos más flexibles ya que no requieren de un set de instrucciones, sino que se pueden modificar para realizar cualquier función lógica que nos imaginemos. Los microcontroladores, por el contrario, deben trabajar con una serie de instrucciones que sean conocidas por el dispositivo.
- También difieren en su forma de procesar las instrucciones. Los microcontroladores ejecutan cada una de las líneas del programa secuencialmente. Una FPGA, al tener una implementación física del código, se ejecutan en paralelo como podría hacerlo cualquier otro circuito. Algunos microcontroladores, sin embargo, ofrecen la posibilidad de tener un cierto grado de paralelismo en sus operaciones mediante lo que se conoce como pipeline. Por otra parte, resulta mucho más complejo implementar una secuencia de código en una FPGA, teniendo que recurrir a máquinas de estados que gestionen todo el proceso.
- El paralelismo presente en las FPGAs las hace idóneas para el control de interrupciones mediante las máquinas de estado finito. En microcontroladores se requiere de un ISR (*interrupt service routine*), una subrutina que avisa al microprocesador con las interrupciones.
- Las FPGAs son dispositivos más complejos de prototipar y customizar. Normalmente requieren escribir todo el código de cero y su código y forma de programar no es la habitual. Los microcontroladores sin embargo usan un lenguaje más frecuente y común y además suele haber librerías ya diseñadas que ahoran mucho del proceso.
- Las FPGAs tienen un mayor consumo debido a que contienen transistores inutilizados, árboles de reloj inefficientes y líneas de señales extendidas por todo el chip.

Como vemos, son más las características que difieren a ambos dispositivos de las que las unen. Por ello, una placa que englobe a ambas puede ser interesante para obtener las ventajas individuales de cada una y paliar los defectos de uno y otro. Así, se espera conseguir un dispositivo que sea capaz de:

1. Realizar tareas secuenciales de forma simple y rápida mediante el microcontrolador y a la vez permite tareas que requieran de un alto grado de paralelismo mediante la FPGA.
2. Implementar de forma sencilla instrucciones recurrentes mediante un lenguaje de programación de alto nivel y permitir la flexibilidad de los circuitos lógicos para tareas más específicas.
3. Repartir la carga de trabajo según convenga a un u otro dispositivo ya sea por saturación de un dispositivo, la propia naturaleza de la operación (secuencial o paralela) o por requerimientos de espacio o tiempo.
4. Permitir el uso de uno o ambos chips para adecuar el consumo de energía a la aplicación necesaria.

3. Componentes y herramientas software

En este capítulo expondremos con detalle los principales componentes elegidos para la implementación de nuestra placa entrenadora.

3.1. Lattice ICE40

En este apartado desarrollaremos con más detenimiento el dispositivo FPGA elegido para la implementación de nuestra placa de entrenamiento. Como ya comentábamos anteriormente en el apartado *FPGAs open source* {2.3.2}, nuestra elección se basa en una de las pocas FPGAs open source que existen actualmente en el mercado, el chip de Lattice iCE40. iCE40 es el nombre de una familia de FPGAs de bajo consumo y alto rendimiento de la compañía Lattice Semiconductor cuyo toolchain fue liberado en 2015. Los dispositivos de esta familia se han fabricado en un proceso de bajo consumo CMOS de 40nm. Dentro de esta familia de dispositivos nos encontramos principalmente con dos ramas de dispositivos:

- Una rama de bajo consumo denominada con las letras LP (*low power*).
- Y otra de alto rendimiento, definida por las letras HX (*high performance*).

En la Tabla 2 se muestra una comparativa con las características más importantes de los distintos dispositivos de cada rama.

	LP384	LP640	LP1K	LP4K	LP8K	HX1K	HX4K	HX8K
<i>Logic cells</i>	384	640	1280	3520	7680	1280	3520	7680
<i>RAM bits</i>	0	32K	64K	80K	128K	64K	80K	128K
<i>PLLs</i>	0	0	1	2	2	1	2	2
<i>I/O pins</i>	63	25	95	167	178	95	95	206
<i>Differential Input Pairs</i>	8	3	12	20	23	11	12	26

Tabla 2: Familia iCE40

Para nuestra aplicación utilizaremos el chip iCE40HX4K [21] por varias razones:

- Presenta un equilibrio adecuado en términos de número de celdas lógicas, precio, consumo y área.
- Es el mismo chip empleado en la IceZUM Alhambra II, lo que nos permite tanto tener una referencia en el diseño de la placa como poder reutilizar el software desarrollado para ésta, en el que entraremos en más detalle en el apartado *IceStudio* {3.2}.

3.1.1. iCE40HX4K

A continuación, se profundizará en las características principales del chip de Lattice iCE40HX4K.

La arquitectura del dispositivo (Figura 14) está formada por una matriz de bloques lógicos programables (PLB), una memoria configurable no volátil (NVCM), bloques de memoria RAM y bancos de pines I/O programables.

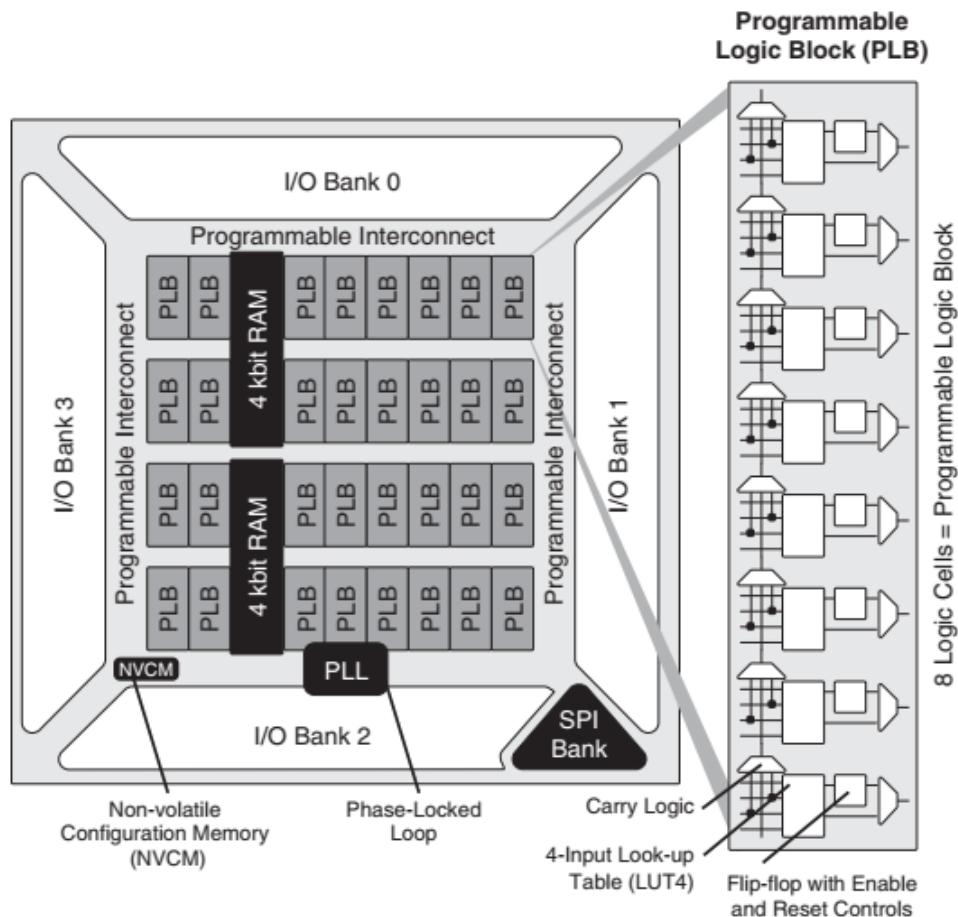


Figura 14: Dispositivo iCE40 [21]

La matriz central contiene en cada columna o bien celdas PLBs (Figura 15) o bien bloques de memoria de 4Kbits que se pueden configurar como RAM, ROM o FIFO. Los bancos de I/O se encuentran en el exterior del dispositivo y contienen además un buffer, referido como sysIO, que le permite operar con distintos estándares. Cuenta también con 2 PLL para multiplicar, dividir o desfasar señales de reloj, así como un bloque SPI que permite la configuración del dispositivo como veremos más adelante en Programación de los dispositivos {4.4}. Todo ello interconectado mediante una red configurable de señales.

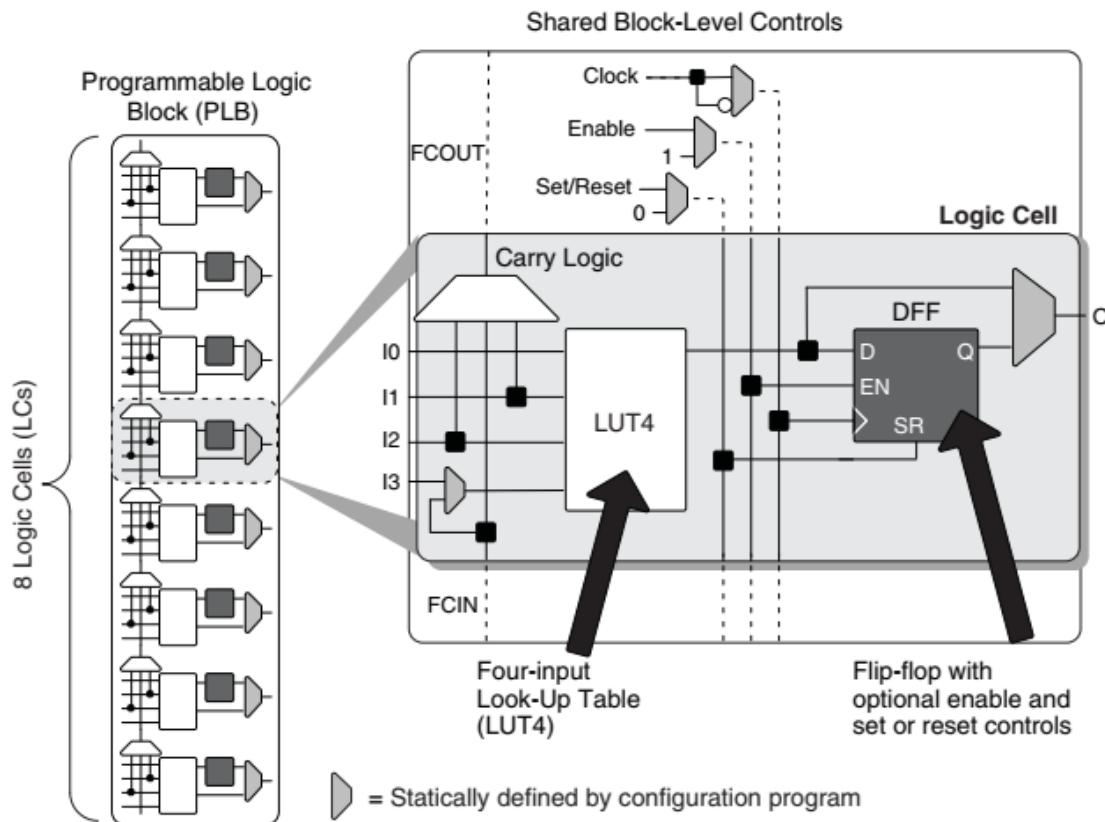


Figura 15: Diagrama de bloques de un PLB [21]

El dispositivo cuenta con 8 pines de entrada globales, dos por cada lado del chip. En estos pines, identificados como GBIN[7:0], podemos introducir señales de tipo reloj, *reset*, *enable*, etc. El PLL admite hasta 133MHz en la entrada, por lo que podríamos añadir una señal de reloj externa de hasta esa frecuencia y de hasta 275MHz en su salida. También pueden usarse si se desea como pines de propósito general. Además, el chip permite forzar a todos los pines I/O a un estado de alta impedancia, lo cual resulta muy útil para disminuir el consumo cuando no están en uso. Los pines de salida se pueden configurar con tres estándares diferentes, en función del valor de alimentación para los pines VCCIO. Dichas configuraciones son:

1. LVCMOS33: Salida 3.3V para VCCIO=3.3V
2. LVCMOS25: Salida 2.5V para VCCIO=2.5V
3. LVCMOS18: Salida 1.8V para VCCIO=1.8V

Además, cuenta con la posibilidad de hasta 12 entradas diferenciales LVDS25 [22] alimentando VCCIO con 2.5V o subLVDS alimentando VCCIO con 1.8V. Aunque el dispositivo no cuente con salidas diferenciales, el fabricante indica que es posible implementarlas configurando 2 pines de salida con una red de 3 resistencias tal y como se muestra en la Figura 16.

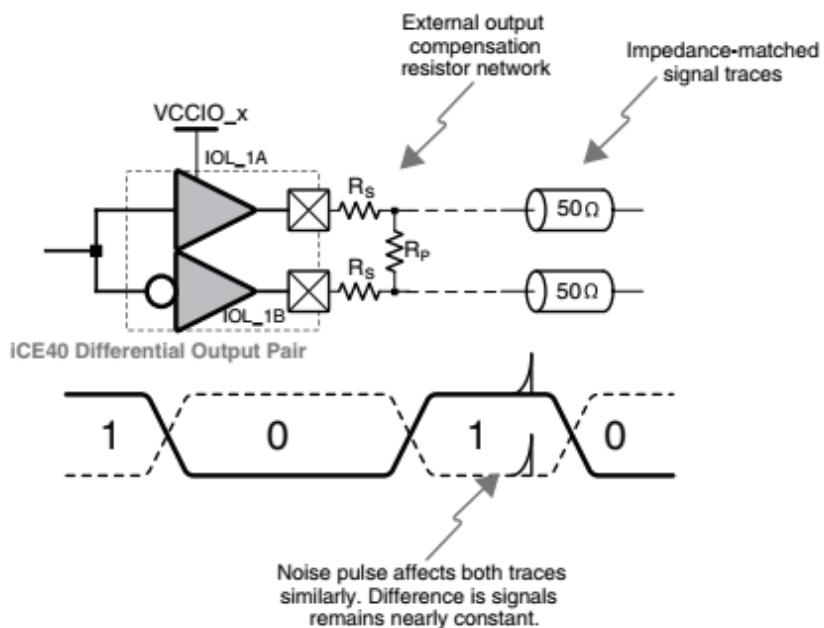


Figura 16: Par de salida diferencial [22]

El dispositivo cuenta con una señal de control POR que se desactiva cuando la alimentación alcanza un nivel establecido. Una vez la señal es desactivada, el núcleo lógico de la FPGA entra en funcionamiento. Con esto se consigue proteger y asegurar el buen funcionamiento de la FPGA.

Para implementar la configuración deseada en el dispositivo podemos recurrir a varios métodos:

1. Programar su memoria NVM mediante el puerto SPI.
2. Escribir en la memoria RAM de configuración (CRAM) la información contenida en la NVM.
3. Configurar la CRAM mediante una memoria flash utilizando el protocolo SPI.
4. Configurar la CRAM mediante un microprocesador utilizando el protocolo SPI.

Las dos primeras opciones son interesantes para un dispositivo que realice una función fija y ya desarrollada y testeada, ya que proporciona seguridad y estabilidad al sistema. Sin embargo, para nuestro caso, como queremos hacer una placa de entrenamiento en la que se implementarán diferentes configuraciones, es completamente inaceptable. Por lo tanto, recurriremos a una de las otras dos opciones. Parece intuitivo elegir la última opción, ya que la arquitectura que pretendemos implementar ya requiere de un microprocesador. Sin embargo, elegiremos la tercera opción por las siguientes razones:

1. Es el método utilizado para programar la IceZUM Alhambra II. Con esto, buscamos una vez más la compatibilidad entre las dos placas, tanto para facilitar el diseño como para reutilizar el software ya desarrollado para esta. Al fin y al cabo, esta es la esencia del open source.

2. Como veremos más adelante, el microcontrolador también necesita programarse varias veces y cumplir con determinadas funciones. Por lo tanto, podremos implementar un circuito que se encargue de programar ambos dispositivos según convenga como veremos en el apartado *Programación de los dispositivos* {4.4}.

Como se ha comentado anteriormente, destaca el bajo consumo del dispositivo gracias a la tecnología de fabricación empleada y a los modos de ahorro de energía que presenta. Los consumos más relevantes del dispositivo son los siguientes:

- El núcleo del sistema, en activo consume apenas 1.14mA llegando a los 22.3mA de pico en el arranque.
- Los PLL consumen 0.5uA en activo, 6.4mA en el arranque.
- Los bancos de I/O consumen 3.5mA en funcionamiento estable, 6.8mA en el arranque.

Haremos un análisis más detallado del consumo en el apartado Análisis de consumo {4.6.1}.

Por último, comentar los distintos encapsulados en los que se puede encontrar el chip:

- BG121: *Ball grid array* de 121 pines de los cuales 93 pines son de I/O y 13 pines diferenciales.
- CB132: *Ceramic ball-grid array* de 132 pines de los cuales 95 pines son de I/O y 12 pines diferenciales.
- TQ144: *Thin Quad Flat-Package* de 144 pines de los cuales 107 pines son de I/O y 14 pines diferenciales (Figura 17).

Para esta aplicación utilizaremos el empaquetado TQ144 al ser más sencillo de tratar para el diseño de la PCB y contar con un número mayor de pines GPIO disponibles.



Figura 17: iCE40HX4K-TQ144 [23]

3.2. IceStudio

En esta sección hablaremos de la herramienta IceStudio [24] desarrollada y mantenida por la comunidad FPGAwars.

IceStudio es una herramienta software de código libre multiplataforma. Está disponible en Windows, MAC y Linux. Construida sobre las bases del proyecto Icestorm usando Apio [25], un ecosistema inspirado en PlatformIO [26], para placas FPGA open source que permite verificar, sintetizar, simular y subir diseños en Verilog.

IceStudio es un entorno gráfico que permite desarrollar código en Verilog de forma muy intuitiva. Tiene una interfaz gráfica basada en bloques interconectados con los que podemos desarrollar el código que queremos de una forma sencilla (Figura 18), haciéndolo muy atractivo para usuario sin experiencia en Verilog. Sin embargo, en ningún momento se pierde flexibilidad ya que cada bloque puede contener el código Verilog que se desee (Figura 19), dejando al usuario trabajar con el nivel de abstracción que deseé. Por lo tanto, nos encontramos con una herramienta versátil, intuitiva y fuertemente apoyada por la comunidad FPGAwars que desarrolla bloques de forma similar a las librerías que se desarrollan para Arduino.

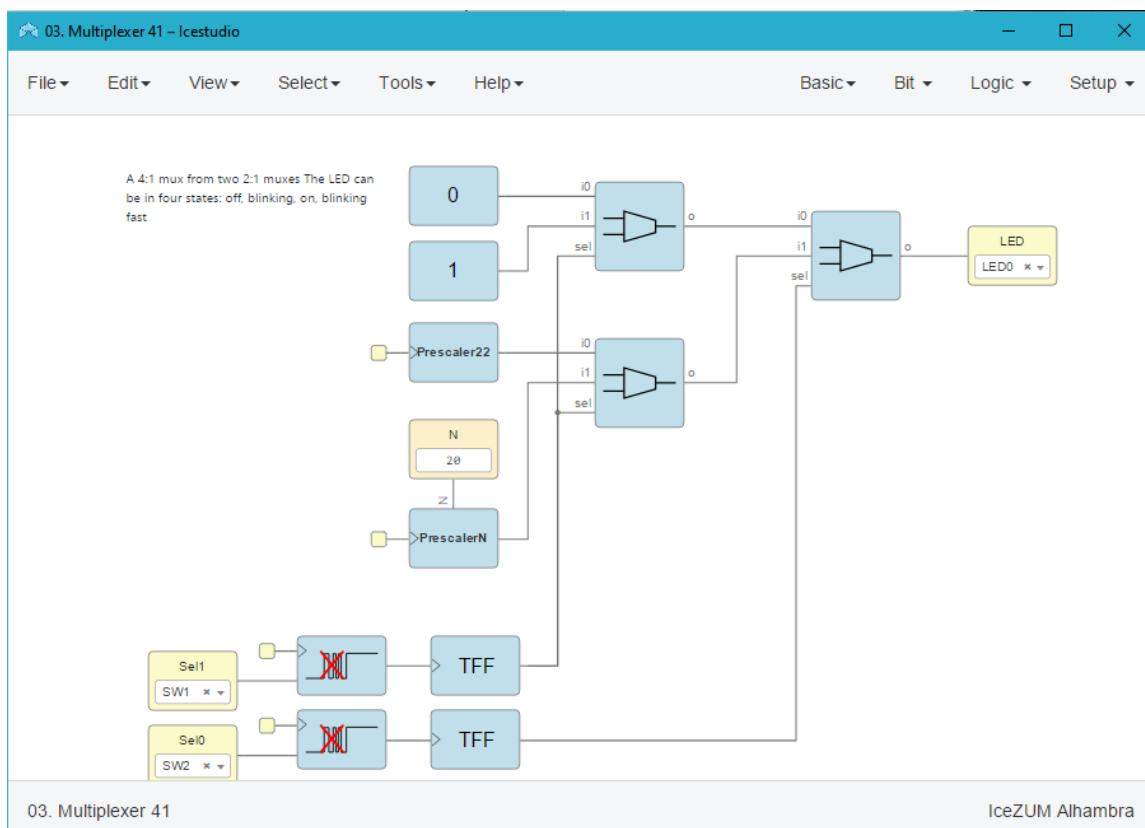


Figura 18: Ejemplo de un multiplexor en IceStudio

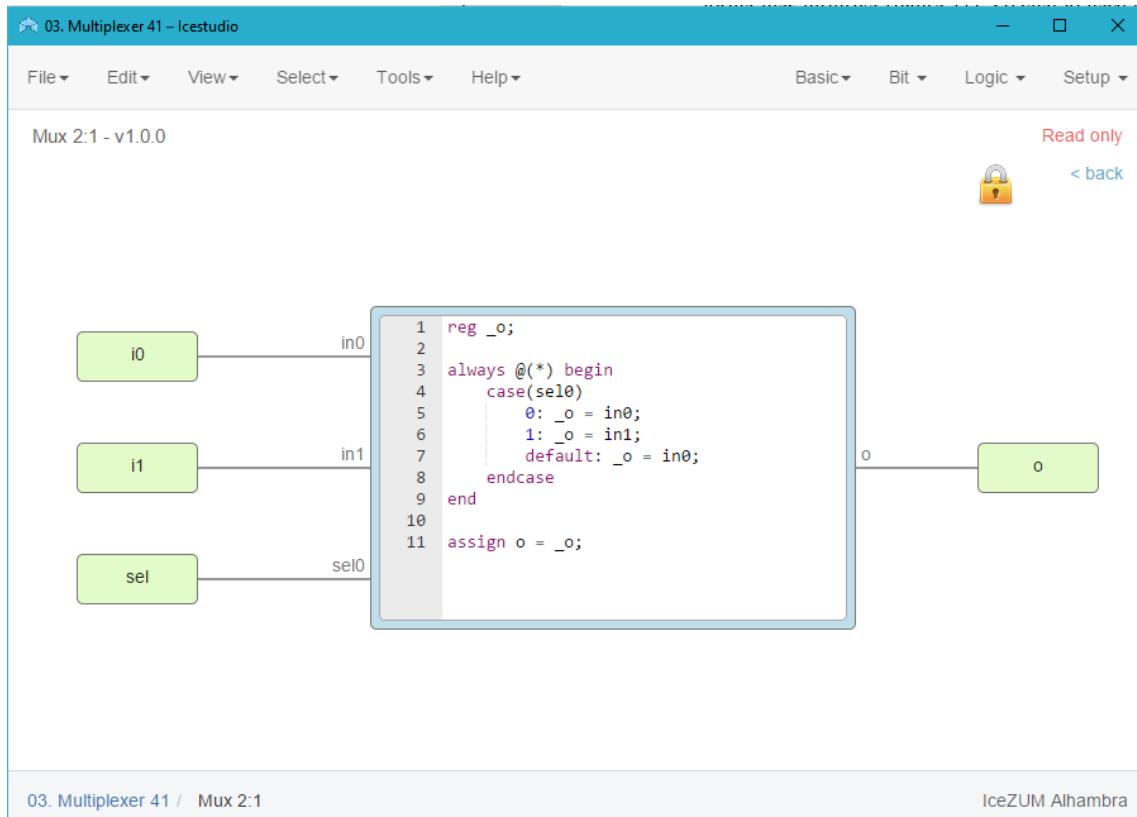


Figura 19: Interior de un bloque en IceStudio

Algunas de las placas soportadas por IceStudio son:

- IceZUM Alhambra I y II.
- Nandland Go board [27].
- BlackIce I y II [28].
- TinyFPGA [29], entre otras.

Nuestro propósito es lograr compatibilidad con este software de forma simple, gracias a la similitud con la IceZUM Alhambra II. Entraremos en más detalle al respecto en *Programación de los dispositivos {4.4}*.

3.3. STM32

En este apartado se introducirá brevemente a la arquitectura de procesadores ARM Cortex [30] en la que se basa la familia de microcontroladores STM32 [31] de la compañía STMicroelectronics [32] para después centrarnos en el chip que se elija en concreto, así como la placa de evaluación para realizar distintas pruebas.

3.3.1. ARM Cortex

ARM Cortex es una arquitectura de procesadores RISC, que debe contar con la licencia de Arm Holdings, con un conjunto de especificaciones relacionadas con el conjunto de instrucciones, el modelo de ejecución de estas, la organización y el diseño de la memoria, entre otras. Un compilador que sea capaz de generar código para esa arquitectura debe de funcionar en cualquier procesador certificado con esa arquitectura.

Arm Holdings es una empresa británica que desarrolla el conjunto de instrucciones y diseña la arquitectura de los núcleos, pero no fabrica dispositivos, sólo los certifica a terceros. Es por ello por lo que esta arquitectura está tan extendida y que existan una gran cantidad de fabricantes de silicio que la utilicen para sus procesadores, entre ellos STMicroelectronics, siendo la arquitectura más utilizada hoy en día.

En concreto, nos centraremos en la arquitectura Cortex-M, especializada en sistemas embebidos y ampliamente utilizada en SoCs, FPGAs y microcontroladores como el que utilizaremos de la familia STM32. Dicha arquitectura se caracteriza por estar optimizada tanto en coste como en eficiencia energética lo que la hace ideal para aplicaciones como el internet de las cosas, la industria de control de sistemas, industria automovilística, etc.

3.3.2. Familia de microcontroladores STM32

STM32 es una amplia familia de microcontroladores basados en la arquitectura Cortex-M mencionada anteriormente. Además del núcleo Cortex-M, estos microcontroladores cuentan con añadidos como memoria RAM, memoria flash e interfaz de depuración entre otros periféricos que veremos más en detalle para el microcontrolador seleccionado. El objetivo de este apartado no es el de entrar en detalle en la explicación de esta familia de microcontroladores, sino dar una idea general del conjunto para rápidamente poder ir acotando hasta encontrar el que mejor se adapte a las necesidades de nuestro proyecto.

Se ha optado por elegir un microcontrolador de esta familia por los siguientes motivos:

- Estar basados en la arquitectura Cortex-M. Esto garantiza la disponibilidad de varias herramientas disponibles en el mercado, muchas de ellas gratuitas incluso libres, ya que es todo un estándar en la industria con más de 50 billones de dispositivos vendidos hasta 2014.
- La gran variedad de dispositivos con los que cuenta, con diferentes rendimientos, precio y características, que permite encontrar aquel que mejor se adapta a las necesidades del diseñador.
- Basados en una arquitectura de 32 bits lo cual ofrece un rendimiento muy superior a otros basados en arquitectura de 8 o 16 bits.

- Con tolerancia para pines de 5V que facilita el uso de periféricos que trabajen a ese rango de voltajes.

Sin embargo, también cuenta con una serie de desventajas:

- La gama de dispositivos es tan amplia que puede resultar complejo decidirse por un chip en concreto si no se tiene claro las necesidades del proyecto.
- Su curva de aprendizaje es algo compleja, sobre todo si lo comparamos con otras plataformas como Arduino. Esto se debe, entre otras cosas, a lo disperso que se encuentra la documentación, especialmente la relacionada con el desarrollo de código para los dispositivos; a la multitud de plataformas compatibles, oficiales y no oficiales, y a que no cuenta con una comunidad de aficionados, sino que es más utilizado por profesionales del sector.

Dentro de la familia STM32 podemos clasificar las diferentes subfamilias en las siguientes categorías (Figura 20):

- Alto rendimiento: dedicados a tareas que requieran de un uso de CPU intensivo como en aplicaciones de inteligencia artificial, así como aplicaciones multimedia, llegando a trabajar a 480MHz.
- Mainstream: microcontroladores con una gran relación rendimiento-coste. Llegan a los 170MHz.
- Ultra bajo consumo: destinados a aplicaciones alimentadas con batería donde cuidar el consumo del sistema es esencial.
- Wireless: integran radio de 2.4GHz y bluetooth, muy populares en la comunidad maker y en educación.

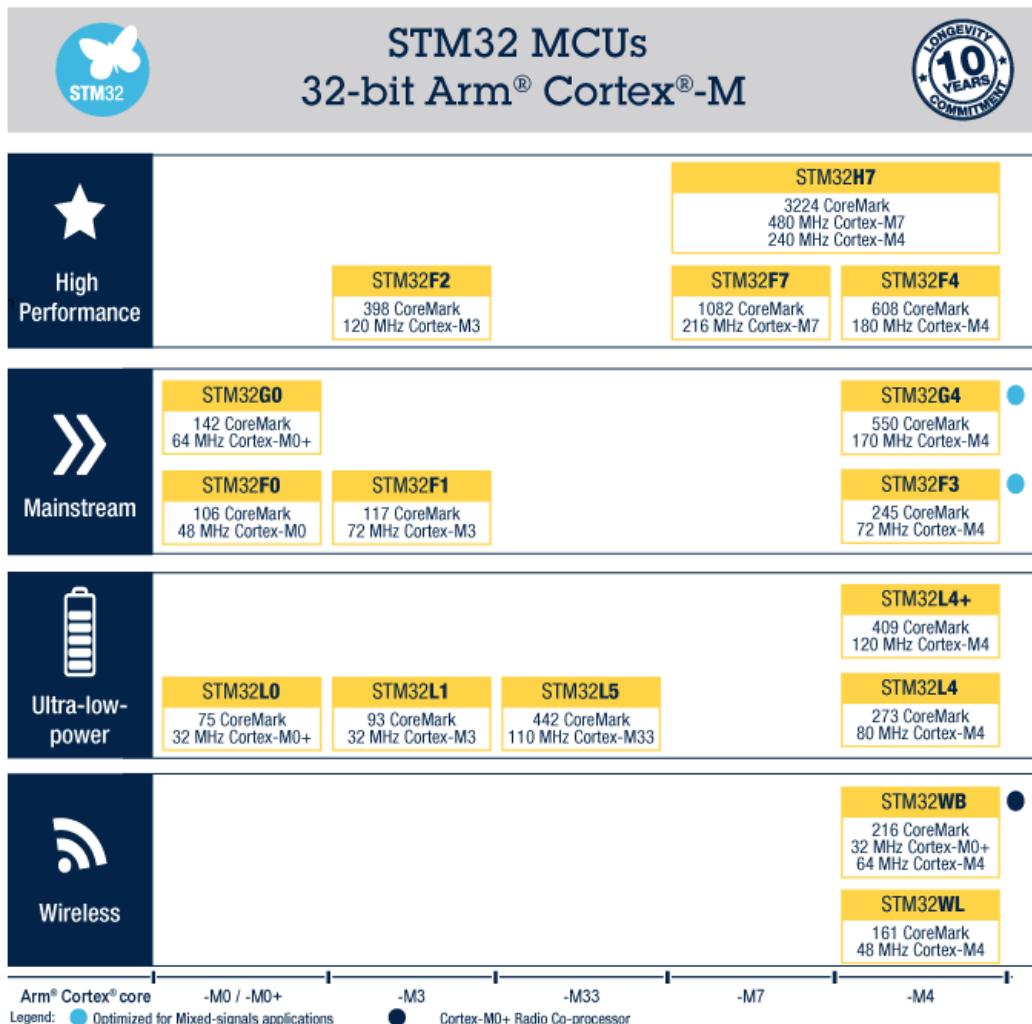


Figura 20: Clasificación de las subfamilias STM32 [31]

Para nuestra aplicación nos centraremos en la categoría mainstream ya que buscamos versatilidad, lo que requiere el mayor rendimiento al menor precio posible. Con ayuda del software “*ST MCU Finder*” (Figura 21) trataremos de encontrar el chip adecuado. Nos centraremos en la medida de lo posible en la subfamilia F0, la más modesta de todas las de su categoría, por razones de coste y consumo. Dicho software nos permite filtrar entre los más de 1500 dispositivos con los que cuenta la familia STM32 según las características que necesitemos. Aun no se han fijado con detalle las características de la placa y, por lo tanto, los requerimientos del microcontrolador. Sin embargo, si se ha comentado anteriormente que el producto final debe contar con:

- Una cantidad elevada de pines I/O.
- Varios pines con diversos protocolos de comunicación.
- Máxima memoria posible, tanto RAM como flash.
- Bajo coste.
- Menor consumo posible.

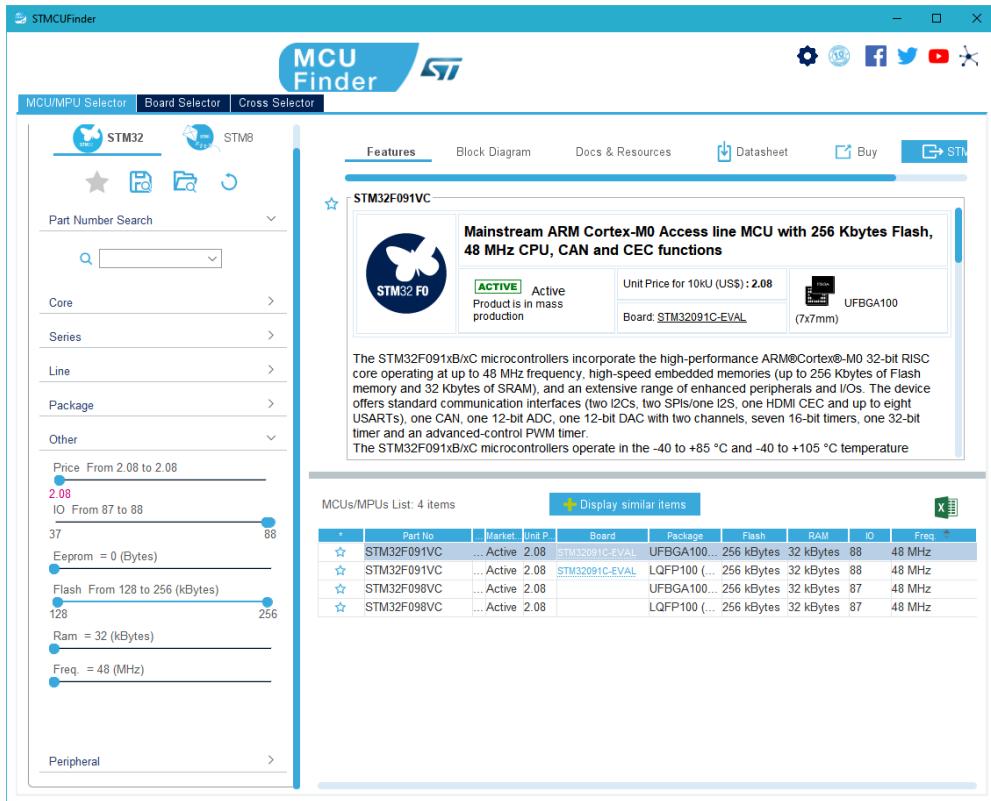


Figura 21: Interfaz ST MCU Finder

Con esas características y centrándonos en la subfamilia F0 encontramos el STM32F091VC [33], que será el que utilicemos en el diseño de nuestra placa por razones de rendimiento, coste y número de pines accesibles.

Para encontrar una placa de evaluación podríamos usar el mismo software. Sin embargo, las restricciones para este dispositivo son más holgadas ya que simplemente servirá para testear diversos componentes. Buscando una relación rendimiento-precio adecuada optamos por una STM32F4DISCOVERY [34].

3.3.3. STM32F091VC

En este apartado introduciremos el microcontrolador seleccionado para nuestro diseño, haciendo un repaso de sus características más relevantes al igual que ya se hizo en el apartado *iCE40HX4k* {3.1.1} con ese dispositivo.

Basado en la arquitectura Cortex-M0 optimizada para dispositivos pequeños y de poco coste, cuenta con un reloj interno de 8MHz y permite operar hasta una frecuencia de 32MHz mediante un reloj externo. Nuestra configuración dispone de 256Kbytes de memoria Flash, así como de 32Kbytes de memoria SRAM. El diagrama de bloques del microcontrolador lo podemos ver en la Figura 22.

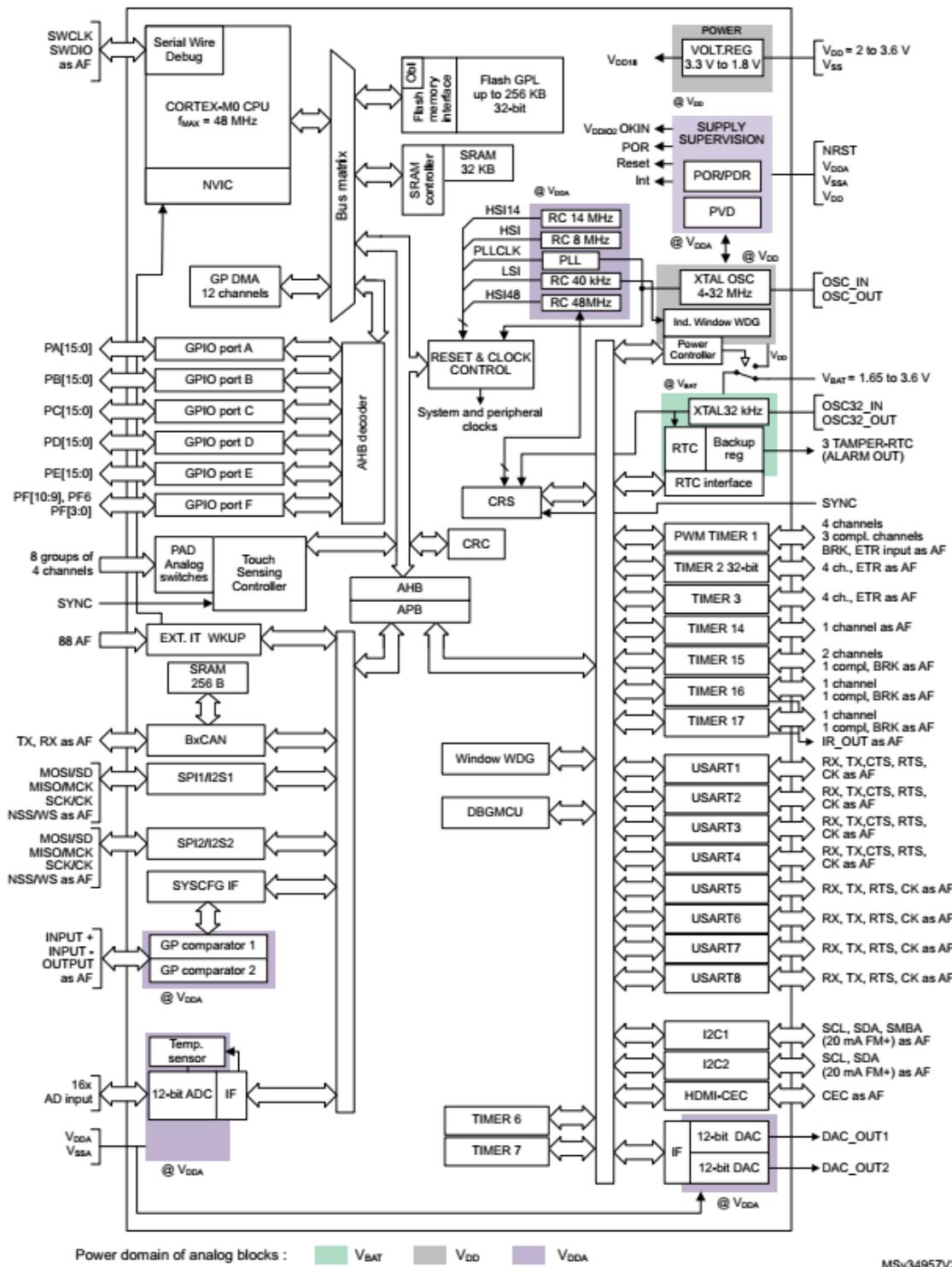


Figura 22: Diagrama de bloques STM32F091VC [33]

Optaremos por el encapsulado LQFP100, al ser más sencillo de trabajar que el otro del que dispone, UFBGA100. En dicho encapsulado nos encontramos con 88 GPIOs. De los cuales, tenemos que:

- 2 pines permiten comunicarse mediante SPI
- 8 pines se comunican mediante UART
- 2 pines implementan I2C

- 1 puede utilizarse como bus CAN.

El chip incluye algunas características extras interesantes como:

- Un conversor analógico digital integrado de 12 bits con hasta 16 canales externos.
- Conectados internamente al conversor hay un sensor de temperatura que proporciona un valor de tensión lineal además de dos referencias del voltaje de alimentación.
- Dos conversores DAC de 12 bits que permite obtener salidas analógicas.
- 9 temporizadores que pueden utilizarse para diversas aplicaciones como medir la longitud de un pulso de entrada, generar pulsos de salida (PWM) o para el conversor digital analógico.
- Dos comparadores con referencia de voltaje programable, interna o externa.
- Un DMA de 12 canales que permite a los periféricos acceder a la memoria del sistema sin necesidad de intervención por parte de la CPU.

Para el arranque del microcontrolador contamos con diferentes opciones:

- Desde la memoria Flash.
- Desde la memoria SRAM del sistema.

Cuenta además con un pin específico que selecciona la opción que se desea utilizar (Tabla 3).

<i>BOOT0 pin</i>	Modo
0	Memoria Flash
1	Memoria SRAM

Tabla 3: Función del pin *BOOT0*

Cuando el pin *BOOT0* está en la opción de memoria SRAM (es decir, a un valor alto) al arrancar el dispositivo, este se reconoce en estado *empty* y permite programar su memoria Flash mediante protocolos de comunicación como UART o I2C como veremos más adelante en el apartado de *Programación de los dispositivos {4.4}*. También sucede cuando la memoria Flash está vacía.

Con respecto a la alimentación del dispositivo hay que destacar que:

- Cuenta con señales POR y PDR para asegurar el correcto funcionamiento del chip activando una señal de *reset* cuando la alimentación cae por debajo de un umbral.
- Cuenta con varios modos de ahorro de energía:
 - Modo *sleep*: sólo para la CPU, los periféricos siguen funcionando.

- Modo *stop*: para tanto CPU como los periféricos (por lo tanto, la señal de reloj externa), pero conserva el contenido de la SRAM y los registros. Mantiene algunos pines para poder reactivarse.
- Modo *standby*: detiene la CPU, los periféricos y se pierde el contenido tanto de la SRAM como de los registros.
- Consume un máximo de 120mA por sus pines de alimentación.
- Aporta hasta 25mA en cada pin I/O y con un total de 80mA.

Por último, comentar su compatibilidad con la herramienta de STMicroelectronics, ST-Link [35] (Figura 23). ST-Link es un dispositivo que permite la programación y depuración de microcontroladores STM8 y STM32 mediante la comunicación SWD con interfaz JTAG. Es compatible con varios softwares, tanto de ST (STM32CubeMX y STM32 ST-Link Utility (Figura 24)) como de terceros (IAR EWARM o Keil MDK-ARM µVision). Se comunica con el PC mediante interfaz USB 2.0.



Figura 23: ST-Link V1 (izquierda) y V2 (derecha) [35]

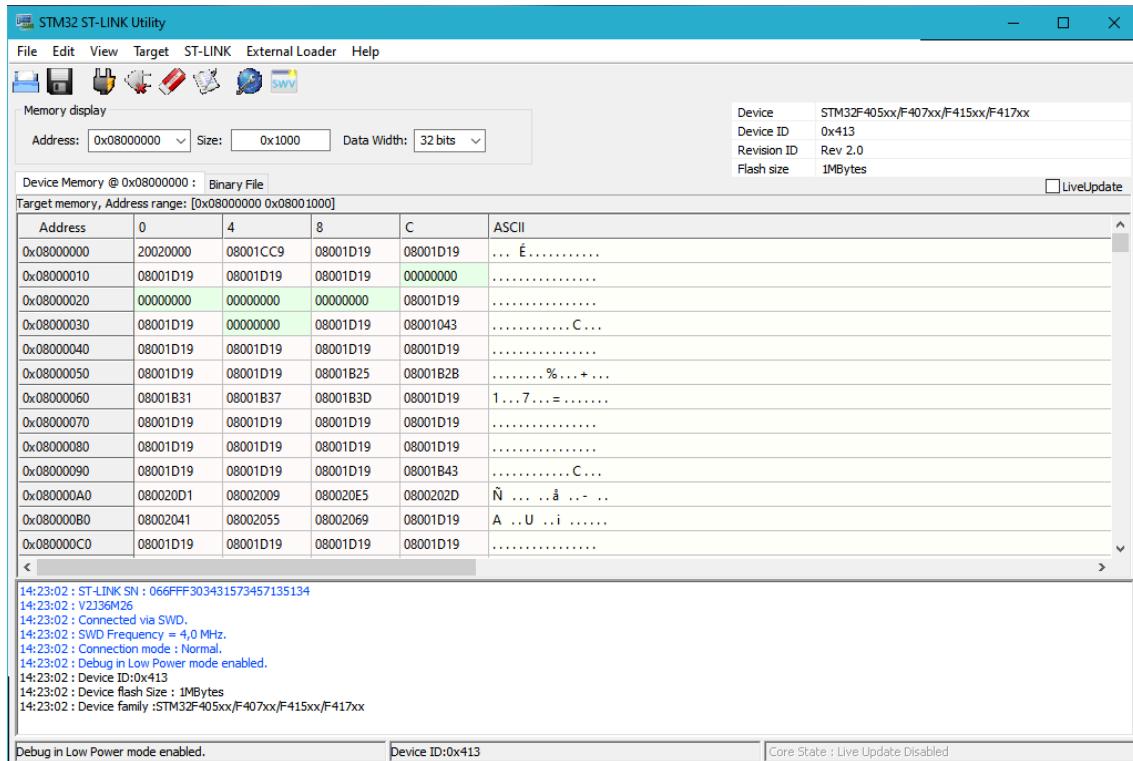


Figura 24: Visualización del contenido en la memoria de la STM32F4DISCOVERY mediante ST-Link Utility.

3.3.4. STM32F4DISCOVERY

Las placas de evaluación permiten un prototipado sencillo sobre los microcontroladores. A veces, una vez testeada la aplicación se pasará a diseñar una placa propia que contenga al microcontrolador como en nuestro caso, y otras, sin embargo, será suficiente con usar una de estas placas. En el mercado existen actualmente 3 tipos de placa de evaluación que contienen un microcontrolador STM32:

- *Nucleo Boards*: Incluyen sólo el microcontrolador y el ST-Link, además de los componentes necesarios para su funcionamiento. Baratas, adecuadas para prototipado genérico y compatibles con los *shields* de Arduino.
- *Discovery Boards*: Incluyen, además de lo comentado en las Núcleo, acceso a todos sus pines I/O y algunas características especiales como, giroscopios, salida de audio, etc. Se utilizan para realizar prototipados más específicos.
- *Evaluation Boards*: Diseñadas para una demostración y desarrollo completo de los microcontroladores STM32. Suelen permitir explotar todas las características del microcontrolador.

Para esta memoria se utilizará la placa STM32F4DISCOVERY (Figura 25 y Figura 26) para comprobar la sinergia de la FPGA, el microcontrolador y la memoria SRAM. Las características más relevantes de la placa son:

- Contienen el microcontrolador STM32F407 con 1Mbyte de memoria Flash y 192Kbyte de RAM.
- Incluye el ST-Link V2.
- Alimentación de 5V a través de USB mini B o de 3V o 5V mediante pines.
- Acelerómetro de 3 ejes y micrófono integrado.
- 8 LEDs y 2 botones.
- Conector micro USB OTG y mini-Jack.
- Acceso a todos sus pines I/O libre.

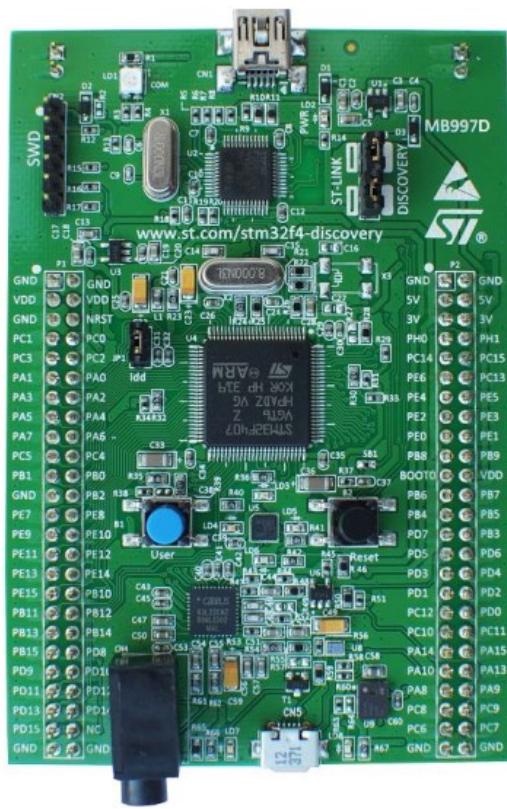


Figura 25: STM32F4DISCOVERY [23]

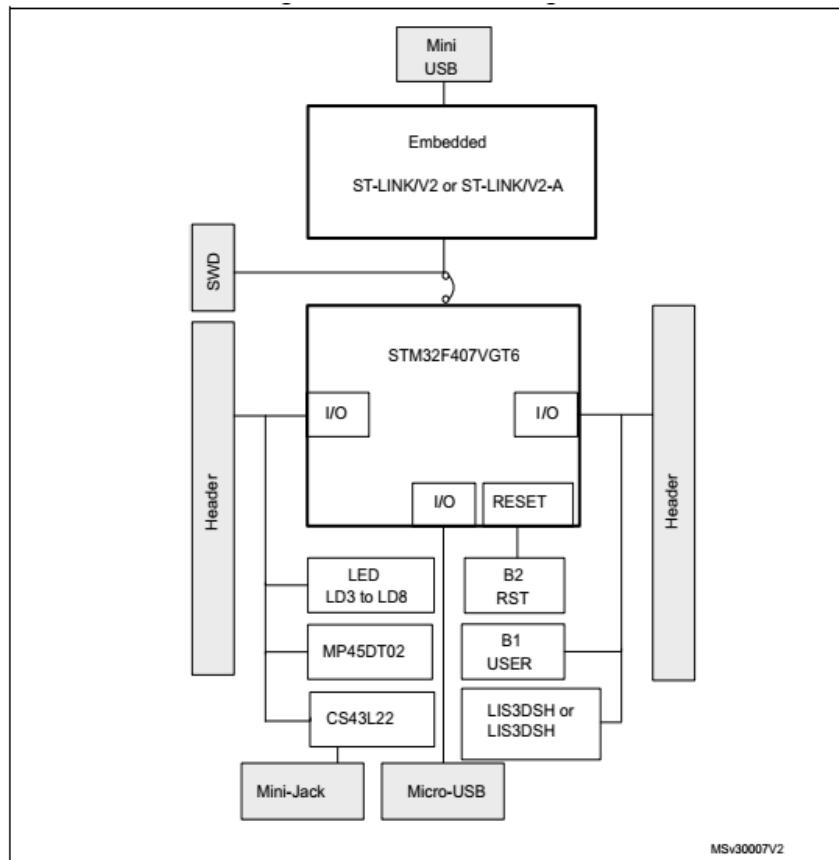


Figura 26: Diagrama de bloques STM32F4DISCOVERY [34]

3.4. Software para STM32

Para programar y depurar microcontroladores de la familia STM32 se puede optar por una gran variedad de soluciones software, tanto de ST como de terceros. Sin embargo, en este documento nos centraremos en dos en concreto:

- STM32CubeIDE [36]
- Arduino IDE [37]

3.4.1. STM32CubeIDE

Hasta 2019 el proceso de generación de código, compilación, programación y depuración de microcontroladores STM32 era relativamente complejo. El problema era que había que recurrir a diversas aplicaciones:

- STM32CubeMX para la generación de código.
- STM32Programmer para la programación del dispositivo.
- STM32Monitor y ST-Link Utility para la visualización y depuración.

Sin embargo, recientemente STMicroelectronics lanzó STM32CubeIDE una herramienta multiplataforma que engloba a todas esas aplicaciones. Con ella podemos:

- Escribir y compilar código en C/C++.
- Configurar los pines periféricos (Figura 27).
- Ajustar la frecuencia del reloj (Figura 28).
- Programar dispositivos
- Depurar mediante software y mediante ST-Link.
- Monitorizar variables.

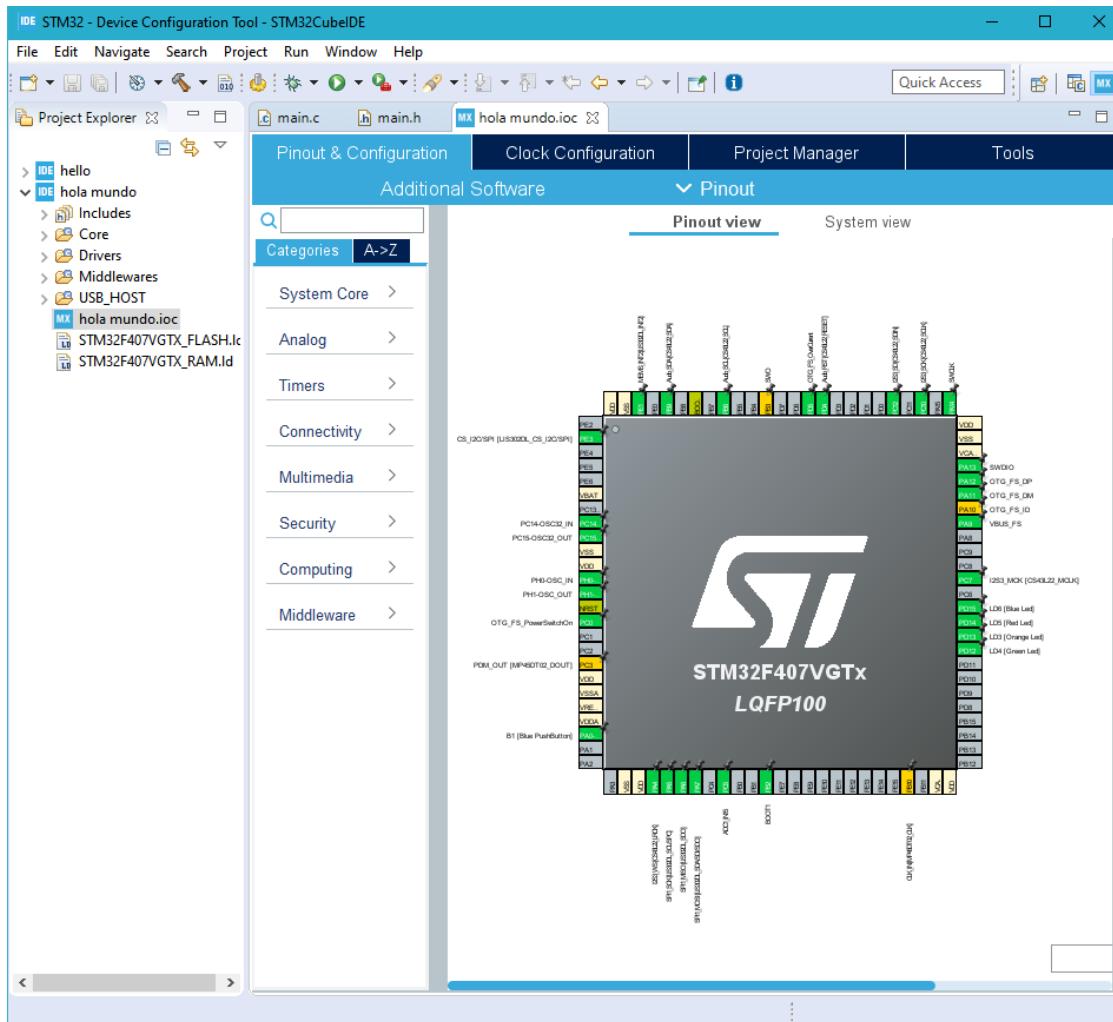


Figura 27: Configuración de pines en STM32CubeIDE

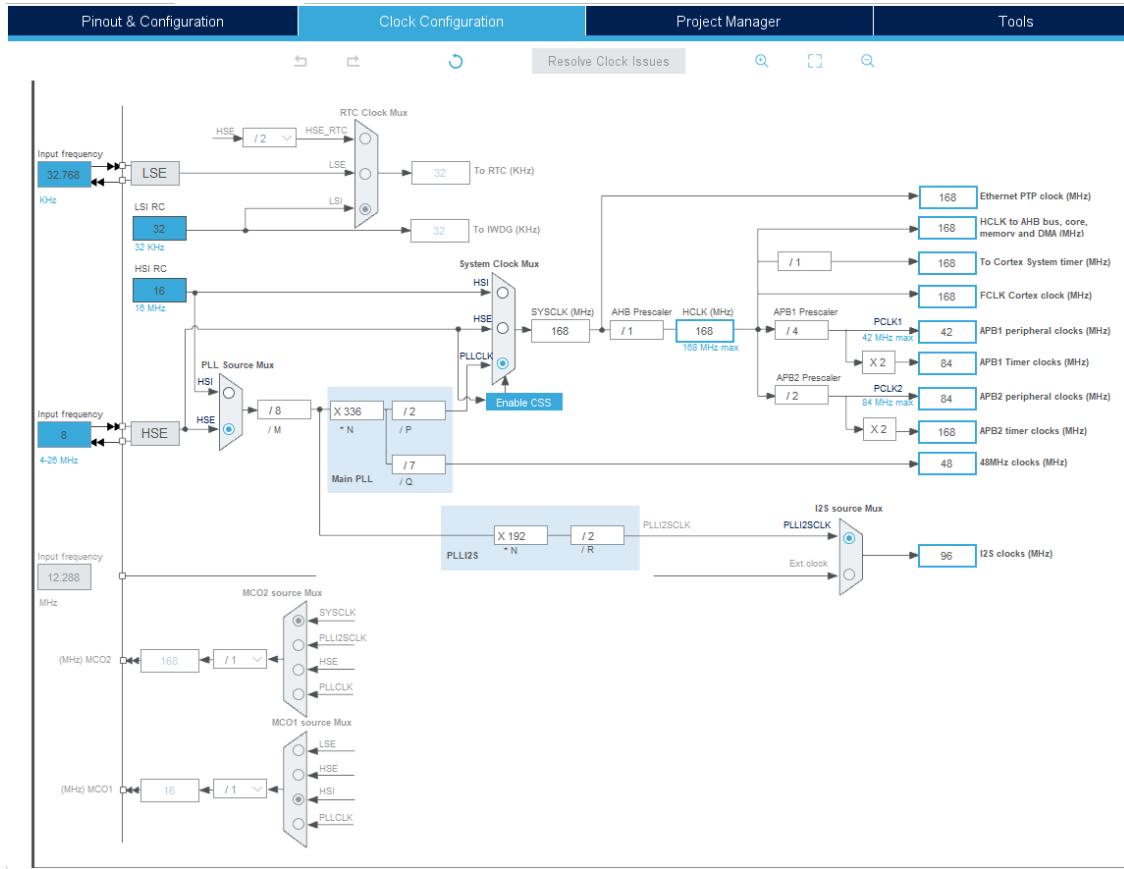


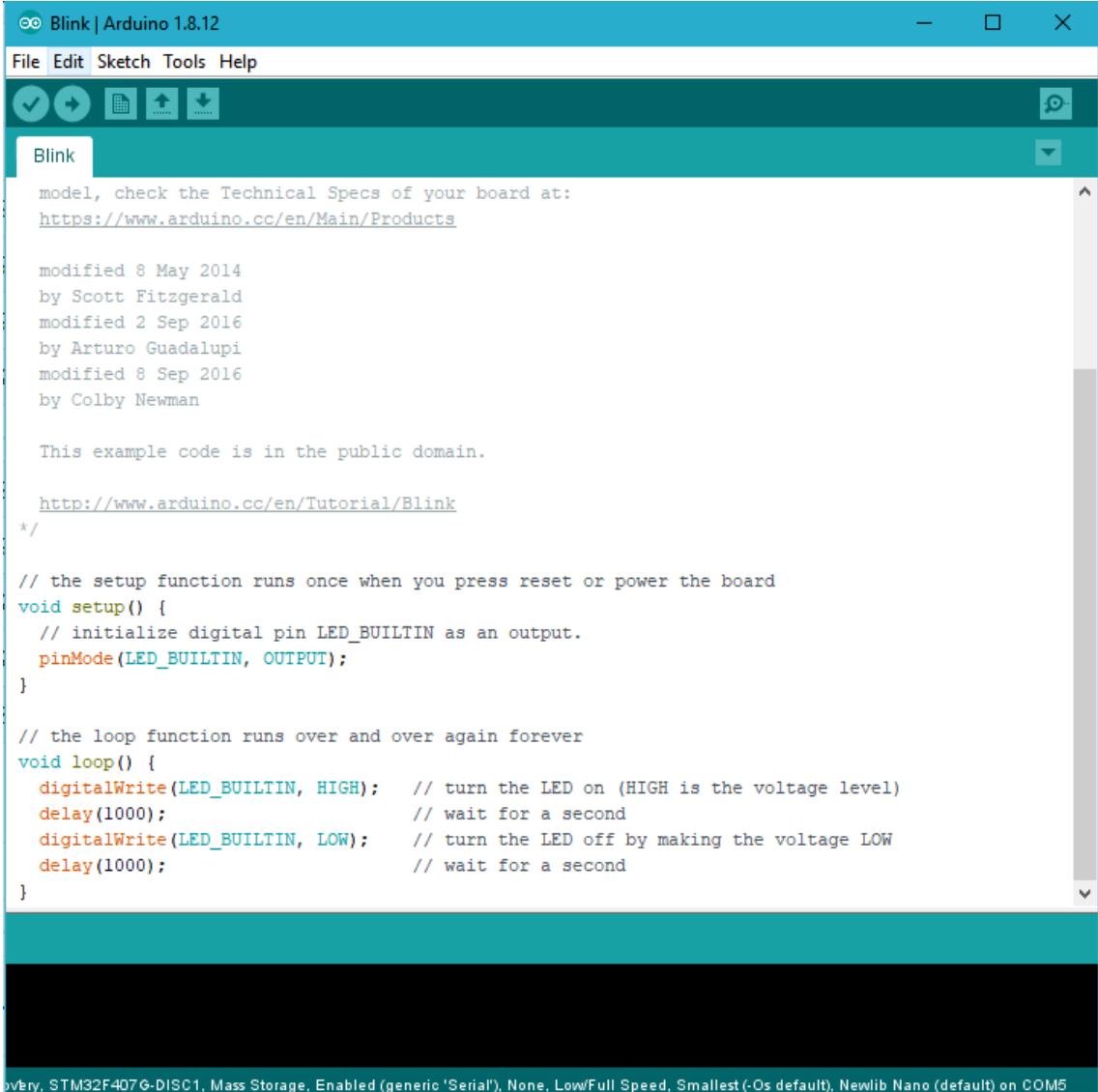
Figura 28: Configuración del reloj en STM32CubeIDE

3.4.2. Arduino IDE

La otra opción que vamos a comentar brevemente en este texto es el ampliamente conocido Arduino IDE. Arduino es una compañía open source de software y hardware que diseña y produce placas microcontroladoras, así como herramientas para su desarrollo. La herramienta más conocida es Arduino IDE, una aplicación multiplataforma que permite escribir código en C/C++ y subirlo a placas compatibles.

Lo que nos interesa de esta aplicación es que gracias al proyecto STM32duino [38], podemos usarla para desarrollar y subir código a nuestra placa entrenadora STM32F4DISCOVERY (Figura 29). A pesar de que la herramienta STM32CubeIDE sea muy completa, lo cierto es que puede resultar algo tediosa para generar programas simples, especialmente si no se está familiarizado con el código de las STM32 y su librería HAL. Es por ello, que cuando se requiera probar una aplicación simple en la placa de evaluación se optará por esta sencilla e intuitiva herramienta.

Sin embargo, cabe recalcar que Arduino IDE no será compatible con la placa que pretendemos diseñar. Por lo tanto, para la programación de esta se debe optar por las herramientas que proporciona el fabricante y las de terceros que sean compatibles.



The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 1.8.12". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Upload. The main window displays the "Blink" sketch. The code is as follows:

```
model, check the Technical Specs of your board at:  
https://www.arduino.cc/en/Main/Products  
  
modified 8 May 2014  
by Scott Fitzgerald  
modified 2 Sep 2016  
by Arturo Guadalupi  
modified 8 Sep 2016  
by Colby Newman  
  
This example code is in the public domain.  
  
http://www.arduino.cc/en/Tutorial/Blink  
*/  
  
// the setup function runs once when you press reset or power the board  
void setup() {  
    // initialize digital pin LED_BUILTIN as an output.  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on (HIGH is the voltage level)  
    delay(1000);                      // wait for a second  
    digitalWrite(LED_BUILTIN, LOW);       // turn the LED off by making the voltage LOW  
    delay(1000);                      // wait for a second  
}
```

At the bottom of the IDE, there is a status bar with the text: "Discovery, STM32F407G-DISC1, Mass Storage, Enabled (generic 'Serial'), None, Low/Full Speed, Smallest (-Os default), Newlib Nano (default) on COM5".

Figura 29: Blinking led en el Arduino IDE para la STM32F4DISCOVERY

3.5. Introducción al diseño de PCBs

Una PCB [39] es un soporte mecánico que permite fijar y conectar eléctricamente los diferentes componentes electrónicos utilizando pistas grabadas sobre una o más láminas de un material conductor sobre una base no conductora. Las pistas son generalmente de cobre, mientras que la base se fabrica de resinas de fibra de vidrio reforzada como la baquelita.

Existen dos tecnologías diferentes empleadas en la fabricación de PCB (Figura 30):

- THT: utiliza agujeros conductores que atraviesan las capas de la placa de circuito impreso para lograr el montaje y conexión de los distintos componentes mediante soldadura. Cada vez más en desuso ya que suele ocupar más área, son más delicados y resisten peor el calor.
- SMT: se emplea un montaje superficial de los componentes por lo que no se atraviesan las capas de la PCB. Esta tecnología ha superado a la THT por su reducido tamaño y la reducción de las interferencias electromagnéticas (especialmente en alta frecuencia) entre otras ventajas. Sin embargo, dificulta el ensamblado manual por lo que se suele optar por métodos automáticos.

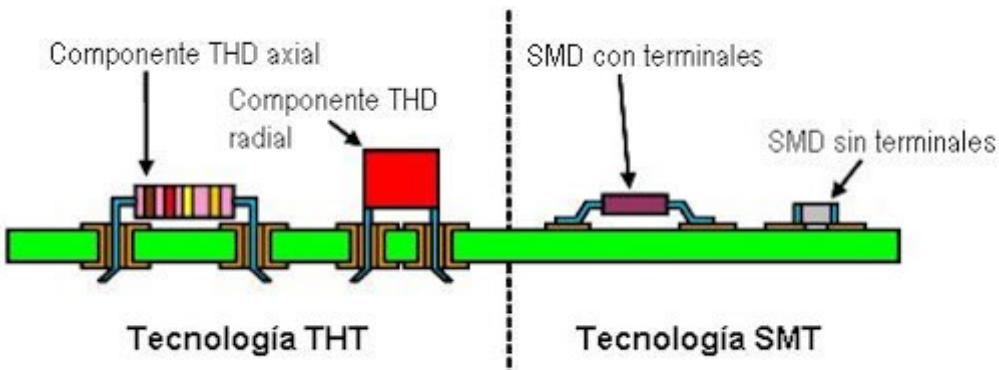


Figura 30: Tecnología THT (izquierda) y SMD (derecha) [40]

Debido a la gran cantidad de componentes y a que la mayoría de ellos ni siquiera disponen de un encapsulado THT optaremos por la segunda opción (SMT) para nuestro diseño.

Hoy en día el procedimiento de diseño se hace mediante el uso de herramientas CAD adecuadas para este propósito. Las herramientas destinadas al diseño de circuitos impresos deben cubrir necesidades como la creación y simulación del esquemático, creación del *layout*, análisis y preparación para la fabricación. Actualmente existen varias herramientas en el mercado, entre las que destacan:

- Altium Designer: Software desarrollado por Altium Limited. La licencia tiene un alto coste. Es posiblemente la herramienta más utilizada a nivel comercial.
- Eagle: Herramienta de Autodesk Inc. con una versión gratuita para estudiantes. Dicha versión limita el tamaño de la placa y su número de capas a dos.
- KiCad: paquete de software libre que permite el diseño de esquemáticos, PCBs y componentes.

Para este TFG utilizaremos KiCad [41] al ser una herramienta open source. Con ello pretendemos que cualquiera que esté interesado pueda analizar e incluso modificar el esquemático y la PCB a su antojo.

Las fases principales que conforman el proceso de diseño de PCB son:

1. Creación de los símbolos de los componentes.
2. Diseño del esquemático (conexionado de los componentes).
3. Diseño de la huella de los componentes.
4. Enrutado de las pistas de conexión (*layout*).
5. Verificación y creación de los archivos para la fabricación (Gerber).

En la Figura 31 podemos apreciar un diagrama del flujo de trabajo en KiCad para el diseño de una PCB. Como observa en la figura, KiCad separa su flujo de trabajo en varias aplicaciones independientes:

- EESchema para el diseño del esquemático
- PCBNew para el enrutado de las pistas de la PCB.
- GerbView para la visualización de los archivos Gerber [42], un formato de archivo estándar en la industria que contiene la información necesaria para la fabricación de la PCB

Además, cuenta con otras aplicaciones integradas para la creación de componentes y sus huellas, así como para el cálculo de impedancias, ancho de pistas, etc.

También utilizaremos LibraryLoader [43], un programa que nos permitirá integrar fácilmente componentes, huellas y modelos 3D de la web de los proveedores a la librería de nuestro proyecto en KiCAD.

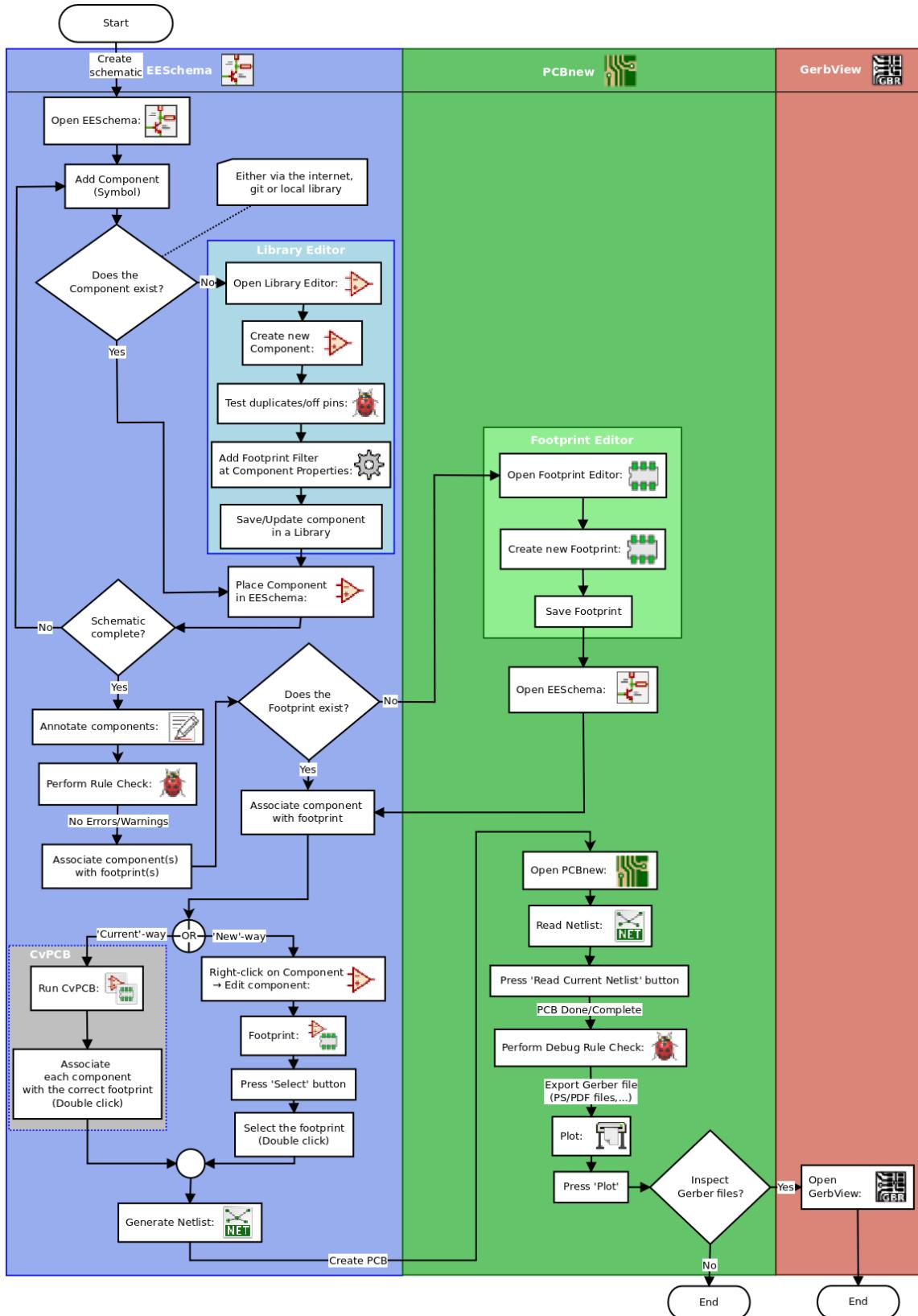


Figura 31: Diagrama de flujo de trabajo en KiCad [44]

4. Implementación

En este capítulo haremos un recorrido al proceso de diseño de la placa de entrenamiento desde la idea original hasta el producto final. En primer lugar, comentaremos la metodología empleada para el diseño de la placa. A continuación, repasaremos las necesidades de la placa y acto seguido pasaremos a detallar la arquitectura a gran escala que implementaremos en la placa. Luego nos centraremos en cada sección de la arquitectura e indicaremos los componentes empleados y su función, así como el diseño en esquemático del sistema. Cuando sea posible, se probarán y comentarán los distintos subsistemas mediante las placas de desarrollo de las que disponemos. Posteriormente indicaremos aspectos relevantes con respecto al enrulado de la placa de circuito impreso, mostrando el resultado final y su modelo 3D. Para terminar, resumiremos las especificaciones del producto, así como cualquier otra información que pueda resultar de interés.

4.1. Arquitectura del sistema

A lo largo de esta memoria hemos ido introduciendo los componentes principales de la placa de entrenamiento, así como una idea general del diseño. Cabe comentar que el proceso de definición de la arquitectura del sistema ha sido muy cambiante a lo largo del tiempo. Sin embargo, no consideramos necesario comentar todas las posibles soluciones con las que dimos anteriormente sino centrarnos exclusivamente en la arquitectura final del sistema. Dicho esto, en este apartado indicaremos primero los distintos módulos que componen la placa, así como el apartado concreto donde se podrá encontrar más información al respecto. En la Figura 32 podemos ver un diagrama de los distintos módulos que compondrán nuestra placa de entrenamiento, así como las conexiones entre ellos. Los módulos que componen el sistema se pueden agrupar en:

- Una entrada USB, que permita tanto la alimentación como la programación de ambos dispositivos mediante un PC. La estudiaremos en el subapartado Conector USB {4.6.2}.
- Un módulo de alimentación que proporcione los distintos niveles de tensión requeridos por el sistema. Detallado en la sección Regulador de tensión {4.6.3}.
- Un módulo que transforme la información recibida desde el PC a un conjunto de datos que puedan recibir y entender la FPGA y el microcontrolador. En la figura 33 se indica como FTDI, y se explicará con detenimiento en el apartado Programación de los dispositivos {4.4}.
- Una FPGA. Detallado en el subapartado Chip FPGA {4.2.1}.
- Un microcontrolador. Explicado con profundidad en el subapartado Chip STM32 {4.3.1}.

- Una memoria SRAM compartida por la FPGA y el microcontrolador. Detallado en el apartado Diseño SRAM y bus de comunicación {4.5}.
- Un ADC para la FPGA, el cual repasaremos en el subapartado ADC {4.2.2}.
- Una salida VGA. Comentada en la sección VGA {4.2.3}.
- Un slot para tarjetas microSD del cual hablaremos en el subapartado Tarjeta SD {4.3.2}.

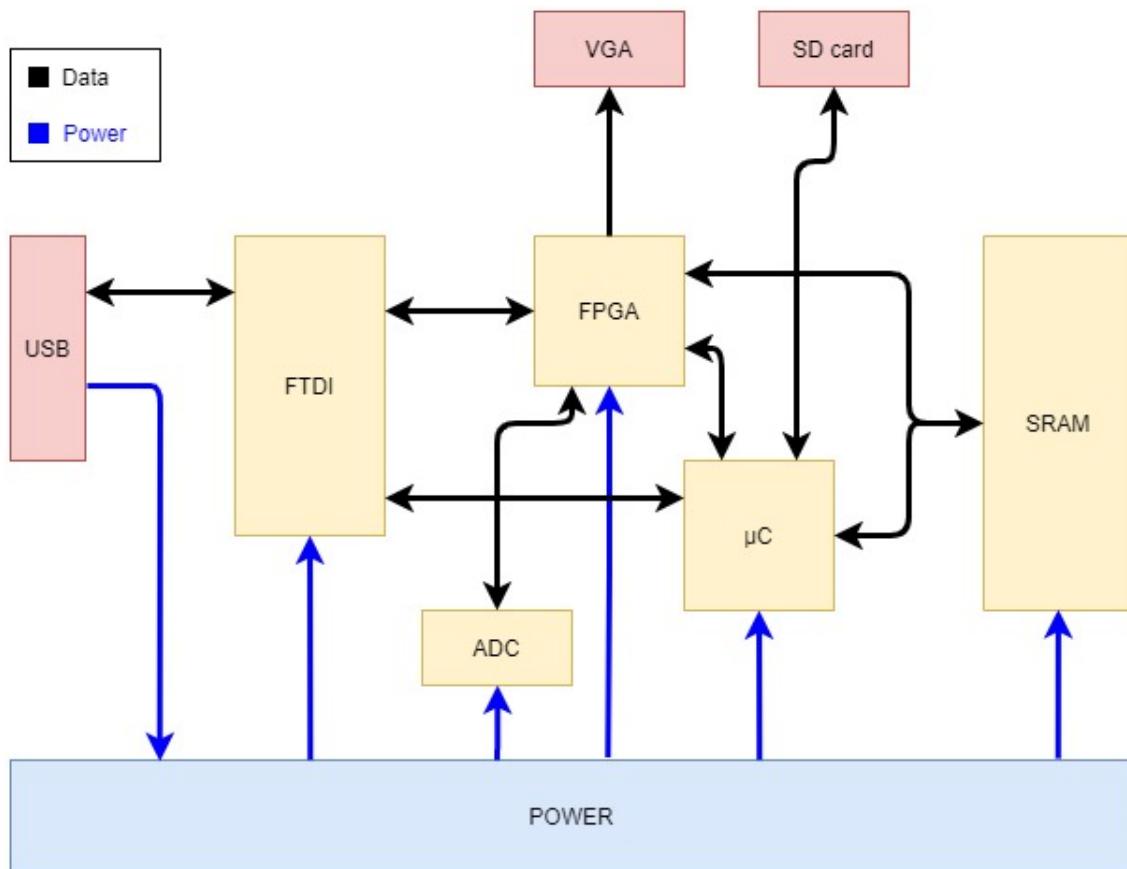


Figura 32: Diagrama de bloques de la placa entrenadora

Para el proceso de selección de componentes se ha seguido los criterios que exponemos a continuación:

1. Las características del componente: Evidentemente, el factor más limitante a la hora de seleccionar un componente es que cumpla con los requisitos que necesitamos.
2. Como segundo criterio estableceremos el precio y el tamaño del lote. Independientemente de que la placa sea fabricada o no, el objetivo es diseñar un producto real que pueda llevarse a cabo. Por lo tanto, hemos tenido muy en cuenta el precio de cada componente y hemos dado prioridad a componentes que se vendan en unidad sobre aquellos que vienen en lote, a pesar de ser más baratos los segundos. Ello se debe a que si en algún momento se fabrica la placa será principalmente de forma unitaria. Para algunos componentes como

resistencias y condensadores si se ha optado por elegir lotes, de manera que el total de dichos componentes utilizados en una sola placa se acerque al número de unidades del lote. Respecto a esto, añadir también que, siempre que ha sido posible, se ha optado por el menor número de componentes diferentes, intentando siempre reutilizar valores de los que ya disponíamos.

3. Tamaño del componente: Nos interesa implementar una placa de tamaño lo más reducido posible, así que, cuando se permita, se optará por los componentes lo más pequeños posibles. Esto es especialmente relevante en componentes como resistencias, condensadores, leds, etc., que cuentan con diferentes estándares de empaquetado disponibles.
4. Por último, se ha dado preferencia a aquellos componentes que, desde la web del fabricante o del proveedor, disponían del modelo de huella para ECAD. Con ello simplemente se pretende ahorrar tiempo en el desarrollo de la placa. Esto es especialmente relevante en los conectores externos, ya que en características, precio y tamaño son muy similares.

Una vez definida la arquitectura y elegidos los componentes procederemos al diseño del esquemático de los distintos módulos en los siguientes apartados de este capítulo.

4.2. Diseño de la FPGA y sus periféricos

En este apartado comentaremos la implementación de la FPGA en la PCB, así como los periféricos asociados a ella.

4.2.1. Chip FPGA

Comenzaremos comentando brevemente los requerimientos del chip de Lattice para su correcto funcionamiento [45]. Podemos clasificar estos requerimientos en las siguientes categorías:

- Alimentación: el dispositivo requiere de varios niveles de tensión. Por una parte, necesita de +1.2V para los pines de *VCC* y *VCCPLL*, de +3.3V para los pines *VCCIO*, *SPI_VCC* y de +2.5V para el pin *VPP_2V5*. Por otra parte, requiere de condensadores de desacoplo en cada una de estas entradas de alimentación. Optaremos por condensadores cerámicos multicapas de 100nF. Los +3.3V y +1.2V los conseguimos mediante reguladores como veremos en el apartado Alimentación global del sistema {4.6}, y los +2.5V los conseguiremos reduciendo los +3.3V mediante un diodo Schottky. Los PLL del dispositivo contienen bloques analógicos por lo que el fabricante recomienda utilizar una masa diferente y conectar los pines a la alimentación mediante unas resistencias de 100Ω para

disminuir ruidos e interferencias. Se puede observar todo lo comentado en este punto en la Figura 33.

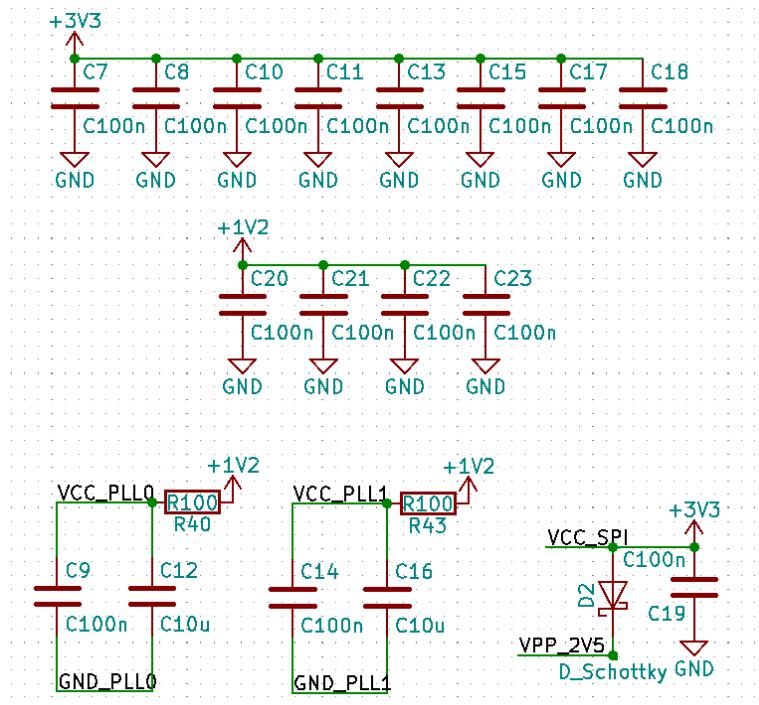


Figura 33: Requerimientos de alimentación para la FPGA

- Configuración de pines: La hoja de datos del fabricante recomienda utilizar una resistencia de $10\text{k}\Omega$ a VCC_{IO_2} en el pin $CRESET_B$ al igual que en el pin $CDONE$. Los requerimientos para la configuración de la FPGA los comentaremos más adelante en el apartado Programación de los dispositivos {4.4}.
- Periféricos: Además de lo estrictamente necesario ya comentado, se le ha añadido al dispositivo una serie de LEDs e interruptores que pueden ser de utilidad para el usuario, así como varios pines de propósito general. Por lo tanto, la FPGA contará con 4 LEDs de diferentes colores, 2 interruptores y 26 pines de propósito general. Los LEDs se han implementado con una resistencia en serie de 470Ω para disminuir la corriente que pase por ellos. Los interruptores se han anclado a un nivel lógico alto mediante una resistencia de $1\text{k}\Omega$ y se les ha añadido una resistencia de $10\text{k}\Omega$ y un condensador de 100nF a masa para evitar ruido. También cuenta con un interruptor de reset, así como un pin externo dedicado a ello por si necesitamos hacer un reset sin necesidad de accionar el interruptor. Recordemos que la señal de reset en la FPGA se produce a un nivel lógico bajo. Podemos ver lo comentado en este punto en la Figura 34 y Figura 35.

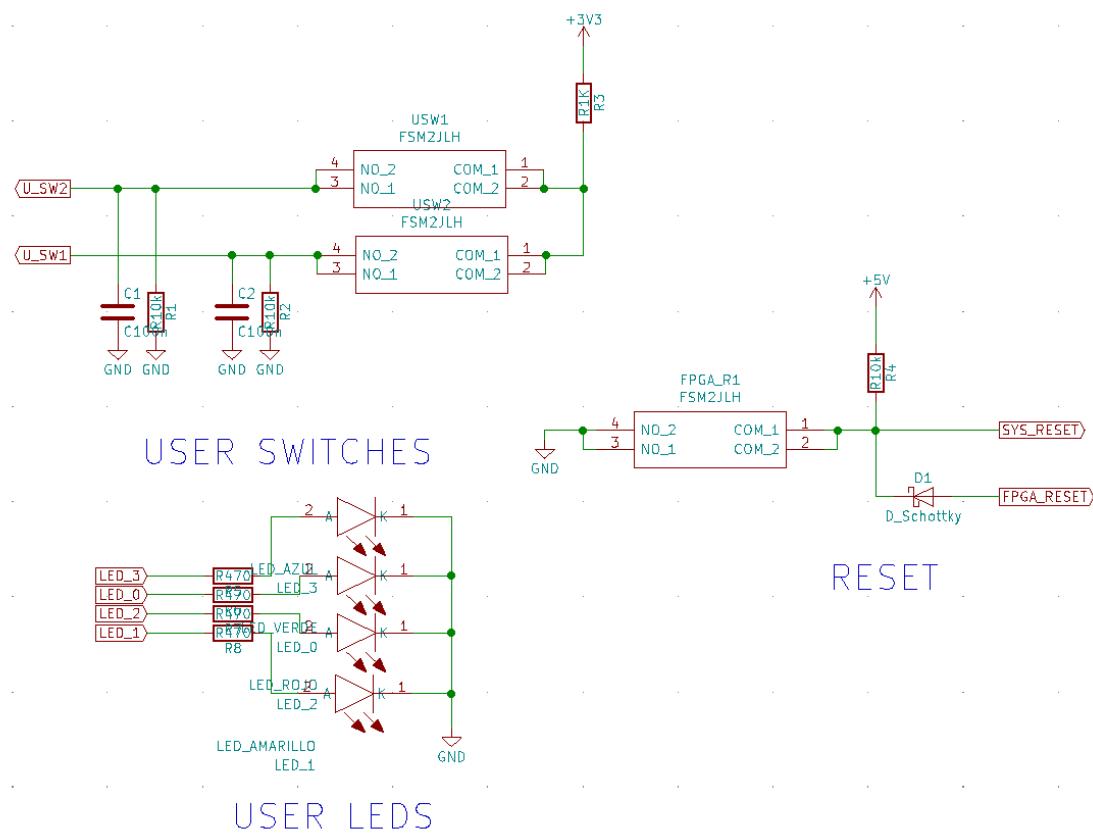


Figura 34: Esquemático para los interruptores y LEDs de la FPGA

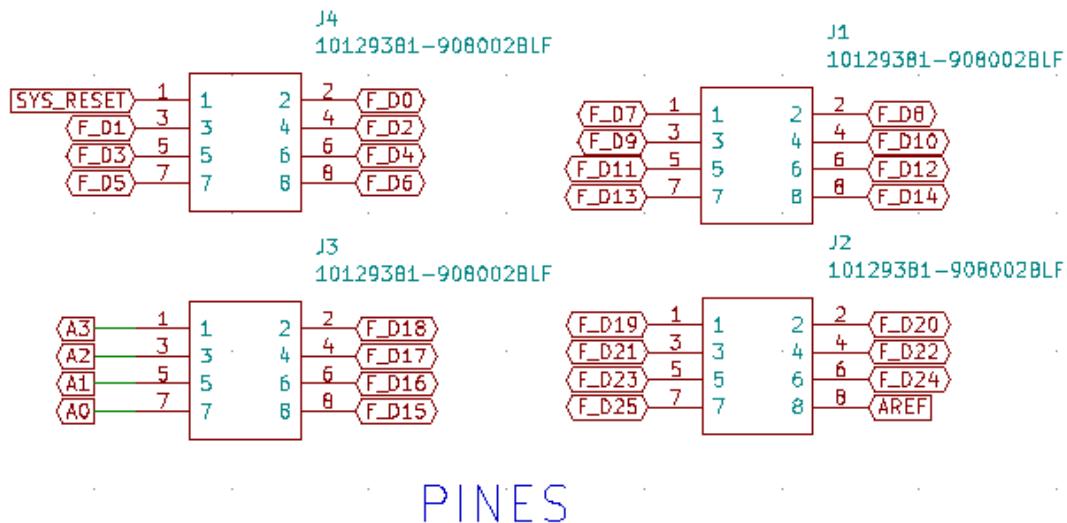


Figura 35: Esquemático de los pines de propósito general de la FPGA

Por último, comentar que hemos introducido en el pin 49 (GBIN5) una señal de reloj de 12MHz. Recordemos que los pines GBIN permiten utilizarse para entrada/salida de señales de reloj, así como de *reset* o *enable*. En la Figura 36 se muestra el esquemático

completo del chip de la FPGA con todas las correspondientes conexiones, algunas que ya hemos comentado en este apartado y muchas otras que no, las cuales se irán viendo con posterioridad.

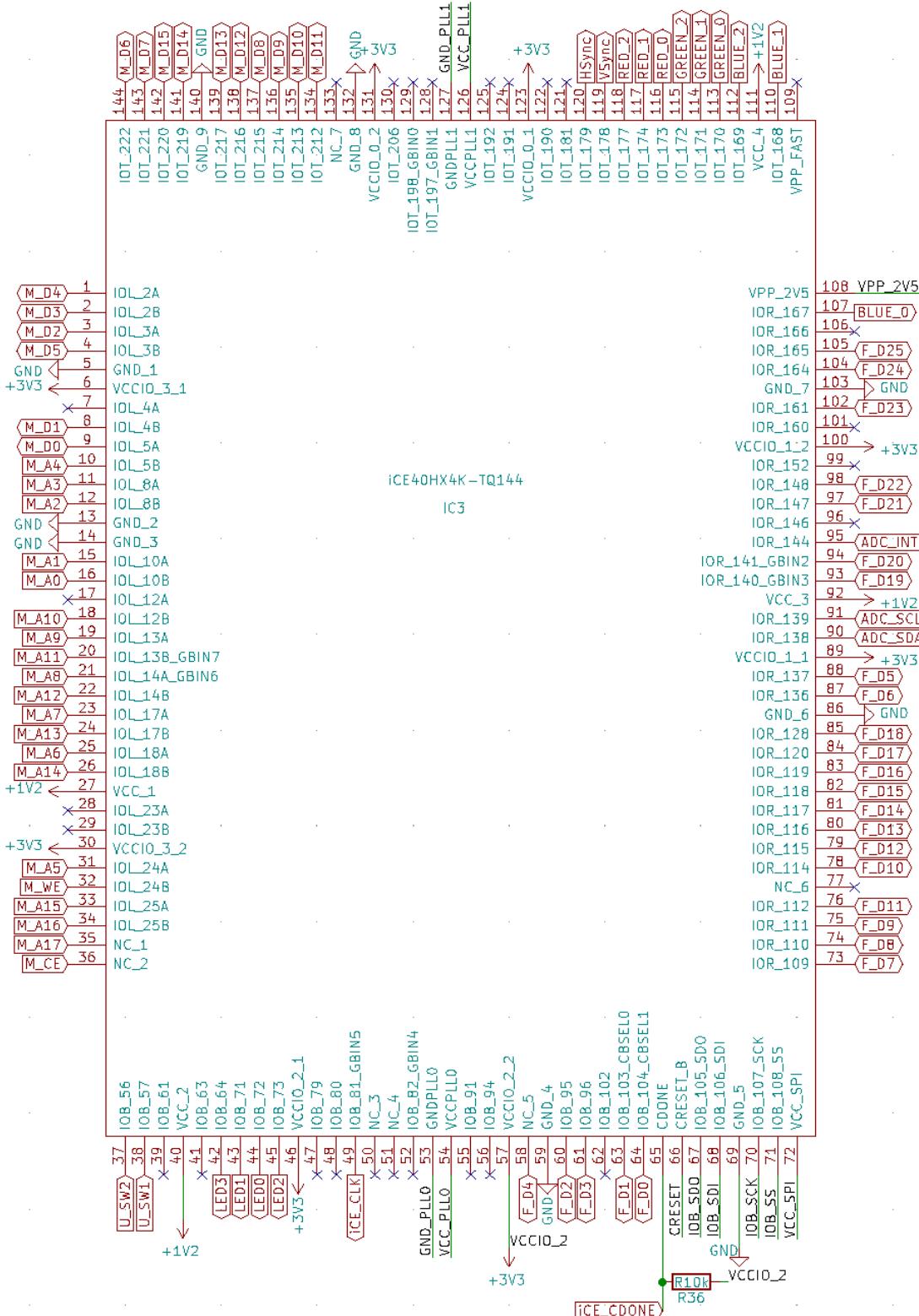


Figura 36: Esquemático de la FPGA de Lattice

4.2.2. ADC

Un ADC es un circuito que se encarga de convertir las señales analógicas en señales digitales. Puesto que la FPGA sólo entiende de señales digitales, se ha creído oportuno incluir un conversor analógico digital para que pueda interactuar con señales analógicas como sensores de luz y temperatura al igual que hace la IceZUM Alhambra II.

Para ello, hemos optado por el ADS7924 [46] de Texas Instruments. El ADS7924 es un conversor analógico digital de 4 canales con 12 bits de resolución. Permite mandar la información convertida mediante un módulo I²C y selecciona el canal adecuado en cada momento mediante un multiplexor. Podemos ver el esquemático del circuito simplificado en la Figura 37.

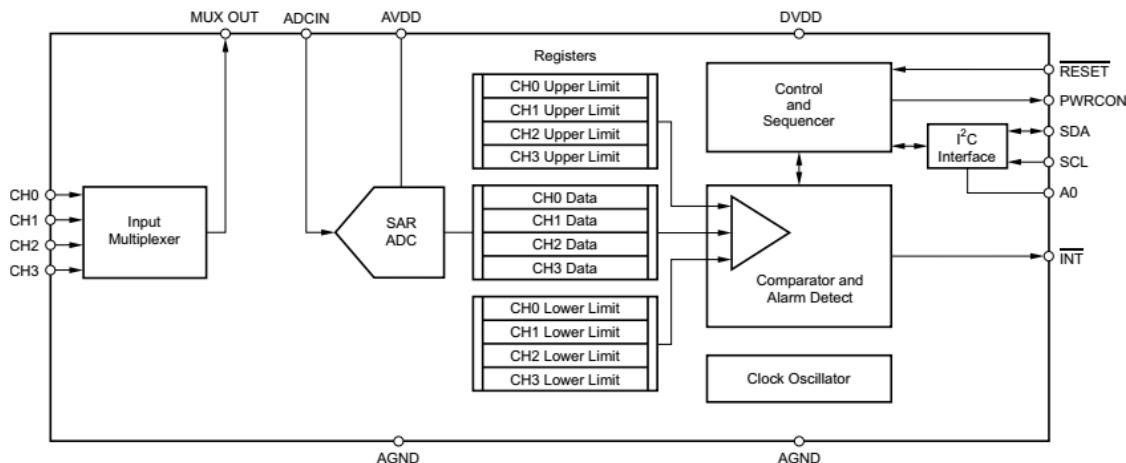


Figura 37: Esquemático del ADS7924. [46]

Su bajo consumo en operación y su prácticamente consumo nulo en espera lo hace perfecto para aplicaciones con alimentación en batería o sistemas de bajo consumo. Incluye un registro de datos y comparadores para cada entrada que permite minimizar el tiempo de atención que requiere por parte del microcontrolador o, en nuestro caso, la FPGA.

En la Figura 38 podemos ver el encapsulado del ADC con los respectivos pines y en la Tabla 4 se detalla el nombre, número y función de cada uno de los pines.

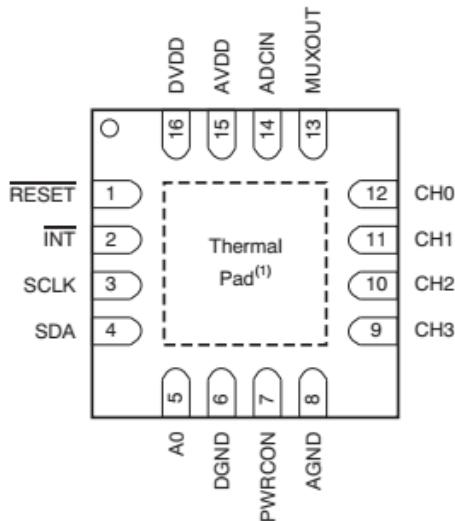


Figura 38: Encapsulado del ADS7924 [54]

PIN No.	PIN NAME	I/O	Descripción
1	RESET	Entrada digital	Reset externo, activo a valor bajo
2	INT	Salida digital	Pin de interrupción, activo a valor bajo
3	SCLK	Entrada digital	Entrada de reloj
4	SDA	Salida digital	Salida de datos
5	A0	Entrada digital	Selección de dirección I2C
6	DGND	Digital	Tierra digital
7	PWRCON	Salida Digital	Pin de control de energía
8	AGND	Analógica	Tierra analógica
9	CH3	Entrada analógica	Entrada del canal 3
10	CH2	Entrada analógica	Entrada del canal 2
11	CH1	Entrada analógica	Entrada del canal 1
12	CH0	Entrada analógica	Entrada del canal 0
13	MUXOUT	Salida analógica	Salida del multiplexor
14	ADCIN	Entrada analógica	Entrada del ADC
15	AVDD	Analógica	Alimentación analógica
16	DVDD	Digital	Alimentación digital

Tabla 4: Pinout ADS7924

Tiene un error de offset de hasta 5 LSBs con una deriva térmica de 0.01 LSB/°C y un error de ganancia de hasta el 0.2% con una deriva térmica de 0.6ppm/°C. Su diagrama de transición de código lo encontramos en la Figura 39, el cual corresponde al típico conversor con cuantificación uniforme salvo para el último bit.

En la Figura 40 podemos ver el esquemático final del ADS7924 en KiCad. Del cual podemos destacar lo siguiente:

- Aunque el conversor admite hasta 6V en su entrada, para la analógica se ha utilizado un regulador de baja caída de tensión para estabilizar esa tensión, ya

que los conversores son muy sensibles a las variaciones en la alimentación haciéndonos perder precisión. La salida del LDO es de 3.3V.

- También se ha dado la opción a modificar el valor de referencia del conversor mediante un pin externo *AREF* filtrado mediante un núcleo de ferrita y condensadores de desacoplo.
- La alimentación digital se ha separado de la analógica y se ha utilizado los 3.3V del sistema con condensadores de desacoplo. Esta tensión es para la circuitería digital del conversor y no es tan crítica como la anterior, su único requisito es que sea igual o menor que la analógica.
- Se han incluido las resistencias de *pull-up* típicas de una línea I_C.
- Las entradas analógicas se han protegido con unas resistencias serie de 100Ω.

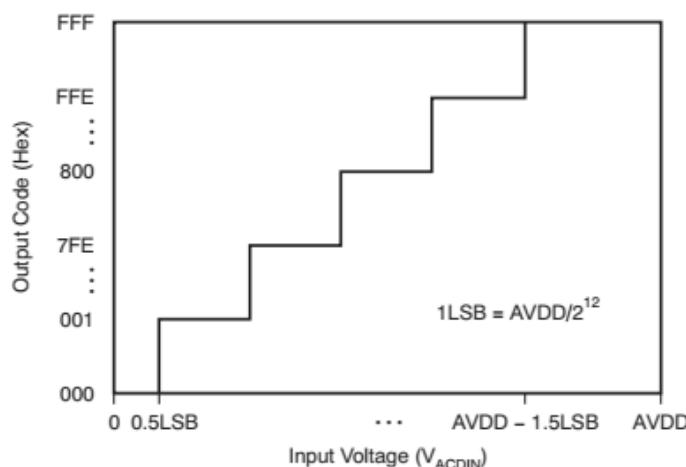


Figura 39: Diagrama de transición de código en el ADS7924

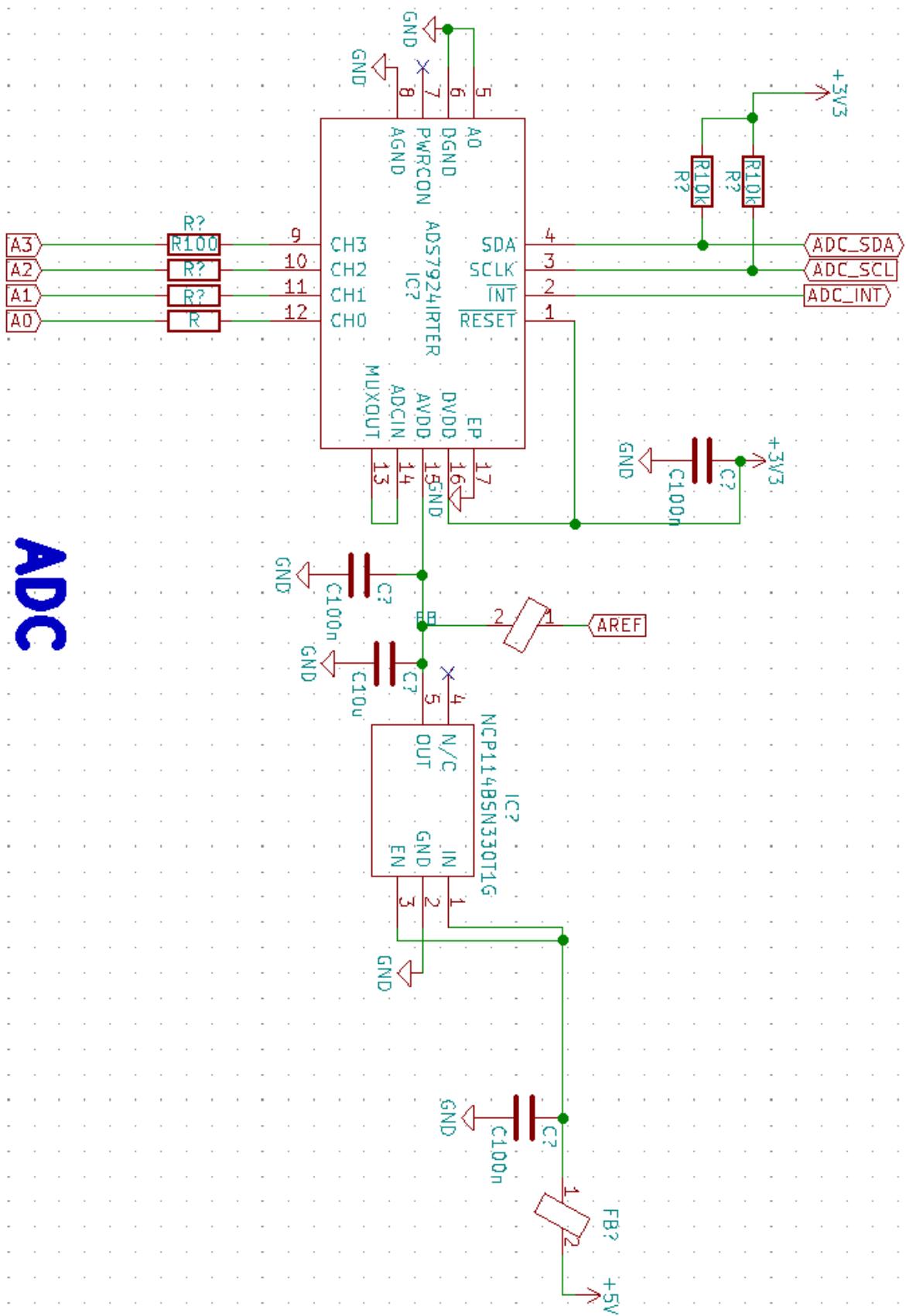


Figura 40: Esquemático del ADC en KiCad

4.2.3. VGA

VGA [47] es un estándar gráfico frecuentemente utilizado en monitores, tarjetas gráficas y en placas FPGA para transmitir señales de video. Diseñado por IBM en 1987, transmite su información de forma analógica mediante 3 señales para RGB y 2 señales de sincronismo. Utiliza un conector D-SUB de 15 pines en 3 filas (Figura 41). Originalmente VGA soportaba una resolución máxima de 600x800 con una profundidad de 8 bits de color, pero actualmente admite hasta 2048x1536 pixeles con 18 bits de profundidad de color. Fue sustituido por DVI que mezclaba señales digitales y analógicas y actualmente está en desuso en electrónica de consumo frente a otros conectores como HDMI o Display Port debido a que estas proporcionan una mejor calidad de imagen al trabajar solamente con señales digitales.



Figura 41: Conector VGA D-SUB 15 [47]

Sin embargo, debido a lo barato que es de implementar y lo sencillo que es de utilizar, optaremos por implementarlo en nuestra placa de evaluación. Con ello podremos obtener una señal de video para visualizar en un monitor externo colores RGB, texto, imágenes de baja resolución o básicamente lo que la aplicación requiera.

En la Tabla 5 se indican la función de cada pin en un conector VGA. Nosotros simplemente utilizaremos los 3 pines RGB y los de la señal síncrona horizontal y vertical.

Número del pin	Nombre del pin	Función del pin
Pin 1	RED	Canal rojo
Pin 2	GREEN	Canal verde
Pin 3	BLUE	Canal azul
Pin 4	N/C	No conectar
Pin 5	GND	Tierra (HSync)
Pin 6	RED_RTN	Retorno del rojo
Pin 7	GREEN_RTN	Retorno del verde

<i>Pin 8</i>	BLUE_RTN	Retorno del azul
<i>Pin 9</i>	+5V	+5V
<i>Pin 10</i>	GND	Tierra (VSync)
<i>Pin 11</i>	N/C	No conectar
<i>Pin 12</i>	SDA	Datos I2C
<i>Pin 13</i>	HSync	Sincronización horizontal
<i>Pin 14</i>	VSync	Sincronización vertical
<i>Pin 15</i>	SCLA	Velocidad del reloj I2C

Tabla 5: VGA pinout

El conector VGA se conectará a la FPGA y no al microcontrolador debido a la facilidad que tienen estos dispositivos para mandar las señales que requiere la FPGA. A continuación, entraremos un poco más en detalle en las características de estas dos clases de señales y por qué son idóneas para la FPGA.

Por un lado, las señales RGB son señales analógicas cuyo valor máximo debe ser 0.7V. Como la salida de los pines de la FPGA es de 3.3V deberemos de reducirlo a 0.7V. Además, utilizaremos 3 bits para cada color, lo que nos proporcionará una profundidad de color de 512 colores. Para transformar el valor digital a un valor analógico utilizaremos un conversor digital analógico R-2R que es barato y sencillo de implementar para palabras pequeñas como ocurre en nuestro caso. El circuito se muestra en la Figura 42 para el ejemplo de la señal RED. Las resistencias se han escogido de forma que:

- Cuando todos los bits estén a 1 el valor de V_o sea 0.7V.
- La resistencia Thévenin vista desde “RED” coincide con la resistencia interna de entrada de los monitores VGA (75Ω) para mejorar los transitorios en la línea.
- Sean valores comerciales.

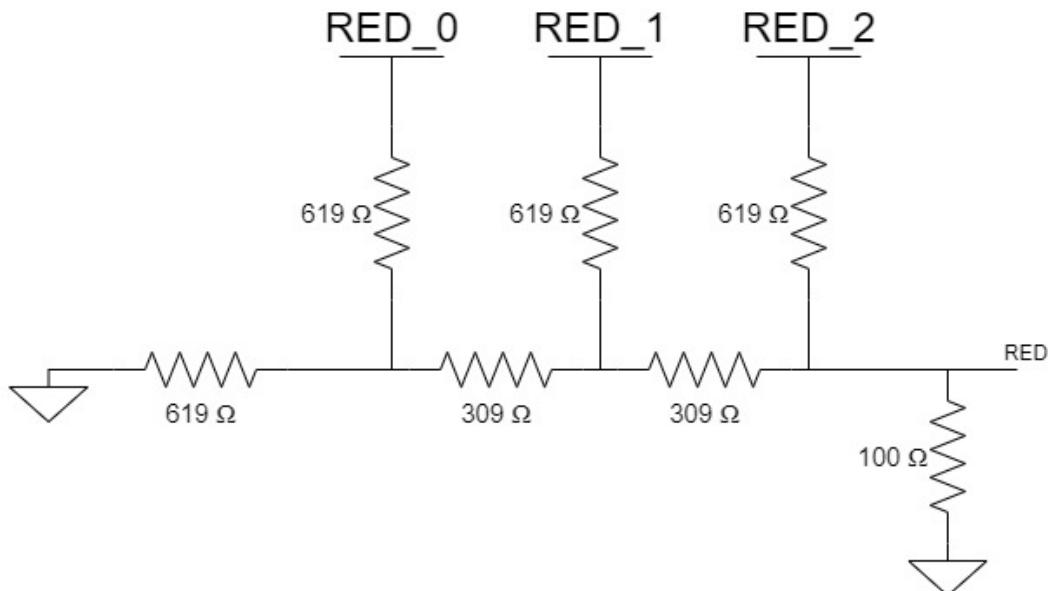


Figura 42: Circuito conversor R-2R para la señal RED

Simulando el circuito en LTSpice [48] obtenemos los resultados de la Tabla 6:

RED _ 2	RED _ 1	RED _ 0	RED (V)
0	0	0	0
0	0	1	0.100
0	1	0	0.201
0	1	1	0.302
1	0	0	0.402
1	0	1	0.504
1	1	0	0.604
1	1	1	0.705

Tabla 6: Valores de RED para su correspondiente palabra digital

Por otro lado, los pines *HSync* y *VSync* son señales TTL cuyo valor bajo es una tensión inferior a 0.8V y su valor alto es una tensión superior a 2V, por lo que en principio no habría que convertirlas a ningún valor en concreto. Son las encargadas de la sincronización con la pantalla, recorriéndola de izquierda a derecha y de arriba abajo, y su origen se remonta a las pantallas de rayos catódicos. Estas dos señales son las responsables de que una salida VGA sea tan apropiada para una FPGA dado que en ellas es muy sencillo de generar señales cuadradas controladas en el tiempo.

La señal comienza en la esquina superior izquierda y recorre la primera fila de la pantalla de izquierda a derecha. Cuando llega al final, transcurre un tiempo hasta que esta vuelve al comienzo de la ahora segunda fila, lo que se conoce como tiempo de retorno. Esto obliga a que la señal *HSync* deba estar a cero durante ese tiempo. Sin embargo, para centrar la imagen en la pantalla es necesario implementar lo que se conoce como *front* y *back porch* (Figura 43), unos tiempos donde la señal debería estar a 0 por estar retornando pero que sin embargo se fuerza a valor alto. Lo mismo ocurre con la señal *VSync* aunque a menor frecuencia. Tenemos, por tanto, una región activa de la pantalla y una inactiva, como se muestra en la Figura 44. Estas señales son las que nos permiten conocer en qué lugar de la pantalla estamos en cada momento.

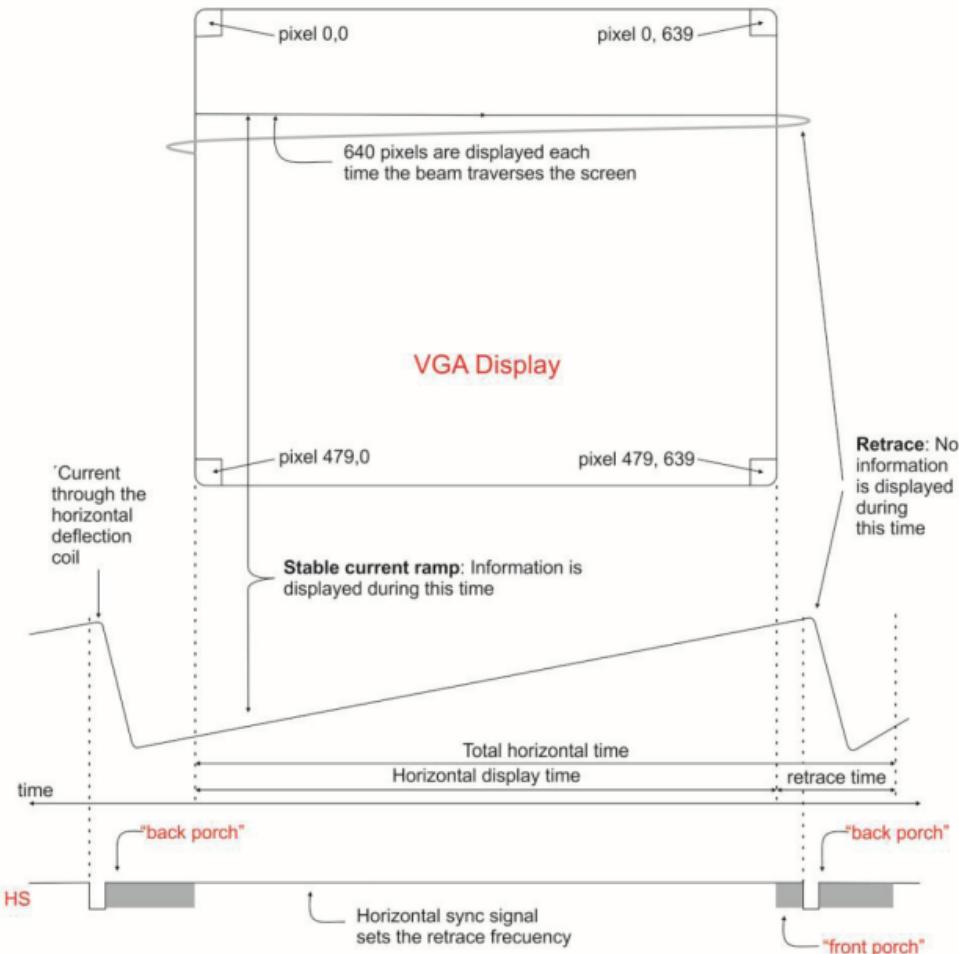


Figura 43: Temporización de la señal HSync en VGA de 640x480 pixeles [49]

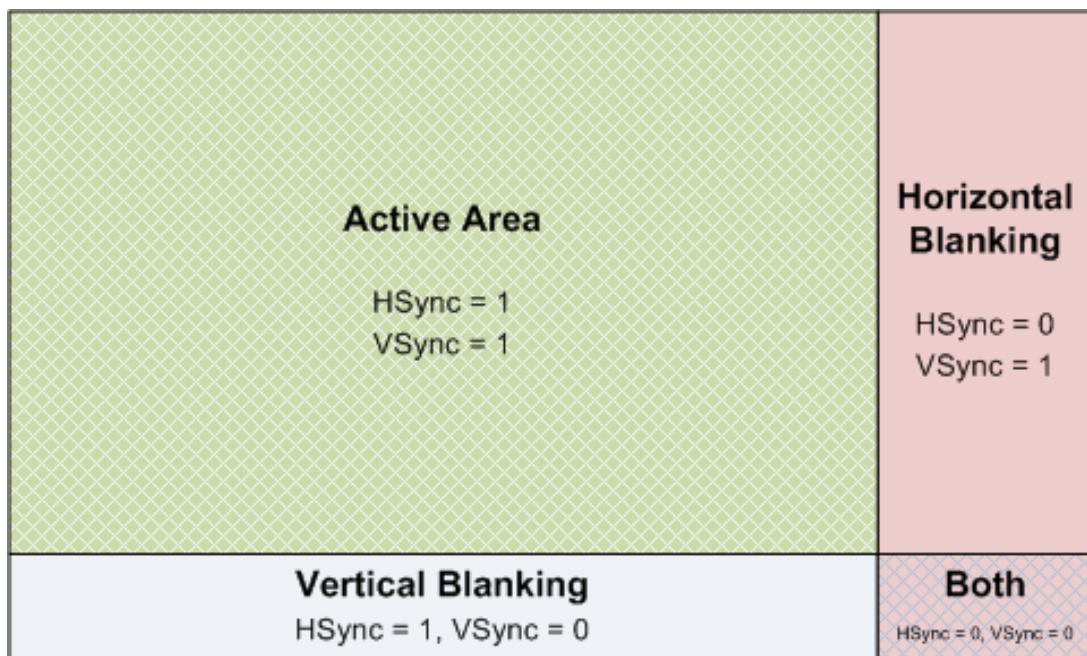


Figura 44: Regiones activa y no activas de una pantalla VGA [50]

Por último, se muestra en la Figura 45 el esquemático del circuito para la salida VGA en KiCad, donde podemos ver el conector de salida VGA y las 3 redes de resistencias R-2R.

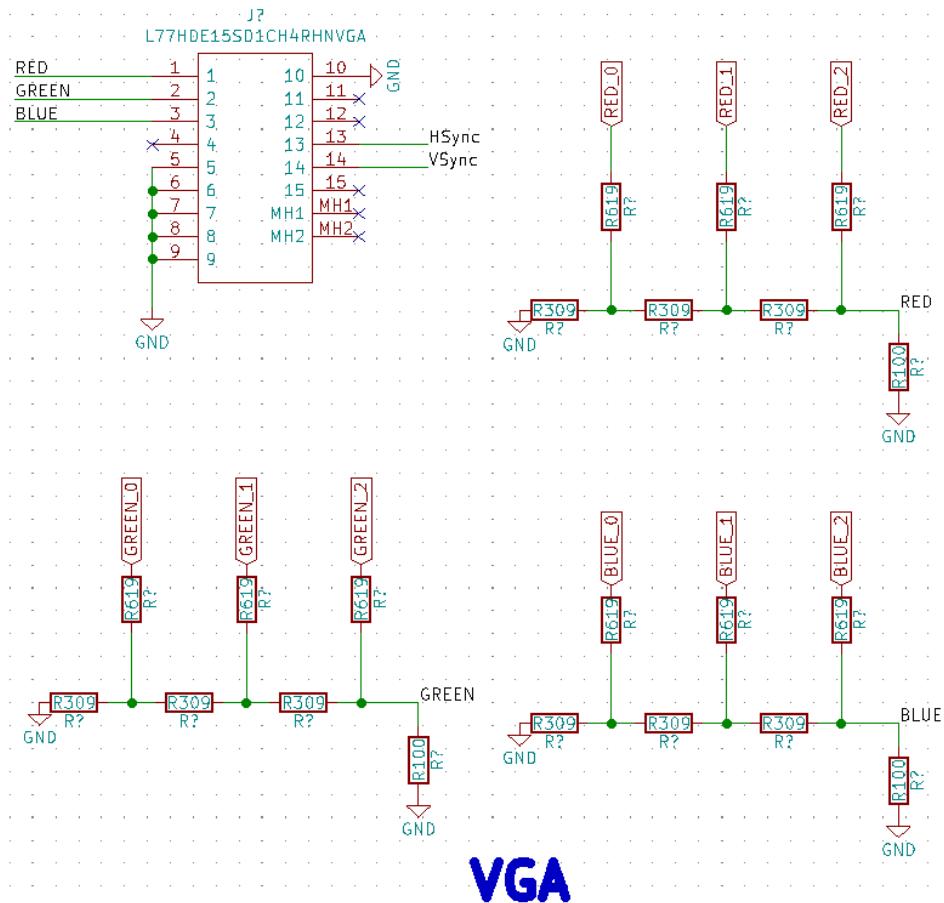


Figura 45: Esquemático para el VGA en KiCad

4.3. Diseño del microcontrolador y sus periféricos

4.3.1. Chip STM32

De forma similar a como ya se hizo en el apartado Chip FPGA {4.2.1}, en este apartado desarrollaremos los requerimientos de hardware por parte del microcontrolador para su correcto funcionamiento [51].

Desde el punto de vista de la alimentación el microcontrolador cuenta con 3 entradas de alimentación diferentes:

1. **VDD**: Alimentación global del dispositivo, se conectará a +3.3V.
2. **VDDA**: Alimentación para el conversor analógico interno del microcontrolador. Se conectará a +3.3V.
3. **VDDIO**: Alimentación para algunos pines I/O dedicados. Alimentado también a +3.3V.

Como ya es costumbre, necesitaremos condensadores cerámicos de desacoplo para cada una de las entradas de alimentación además de algunos electrolíticos de mayor capacidad para mantener estable la tensión. Dicho esto, colocaremos un condensador cerámico de 100nF lo más cerca posible de cada una de las entradas de VDD además de uno electrolítico de 4.7uF para cada conjunto. Para $VDDA$ utilizaremos la misma línea de tensión que para VDD , pero filtrada mediante un núcleo de ferrita además de un condensador cerámico de 10nF y otro de 1uF. Para el pin $VDDIO$ utilizaremos uno cerámico de 100nF y otro electrolítico de 4.7uF. En la Figura 46 podemos observar el resultado final.

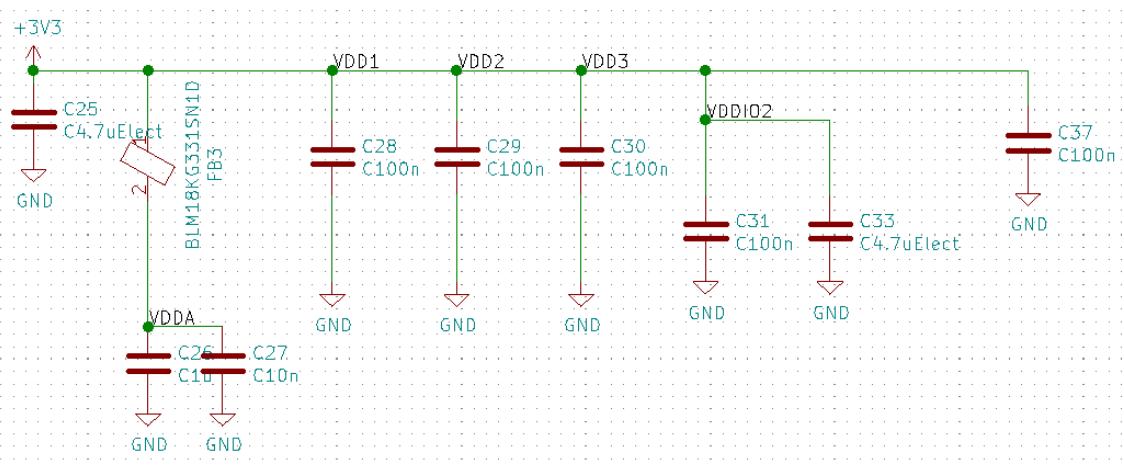


Figura 46: Esquemático de alimentación para el microcontrolador

El microcontrolador cuenta con un reloj interno de 8MHz, pero le podemos añadir uno externo de hasta 32MHz. Optamos por uno genérico de 32MHz con una capacidad de carga de $C_L=15\text{pF}$. Suponiendo las capacidades de la línea en $C_s=5\text{pF}$ y con la Ecuación 1 obtenemos que necesitaremos dos condensadores de 20pF a cada lado del cristal para su correcto funcionamiento.

$$C_L = \frac{C_{L1} \cdot C_{L2}}{C_{L1} + C_{L2}} + C_s$$

Ecuación 1: Cálculo de las capacidades para el oscilador

Se puede cambiar el oscilador por el que el usuario considere más apropiado simplemente cambiando también con él los condensadores que deben acompañarlo. Además, se ha separado la masa del oscilador de la del sistema mediante un anillo para un mejor desempeño del oscilador. En la Figura 47 podemos ver el esquemático para el oscilador.

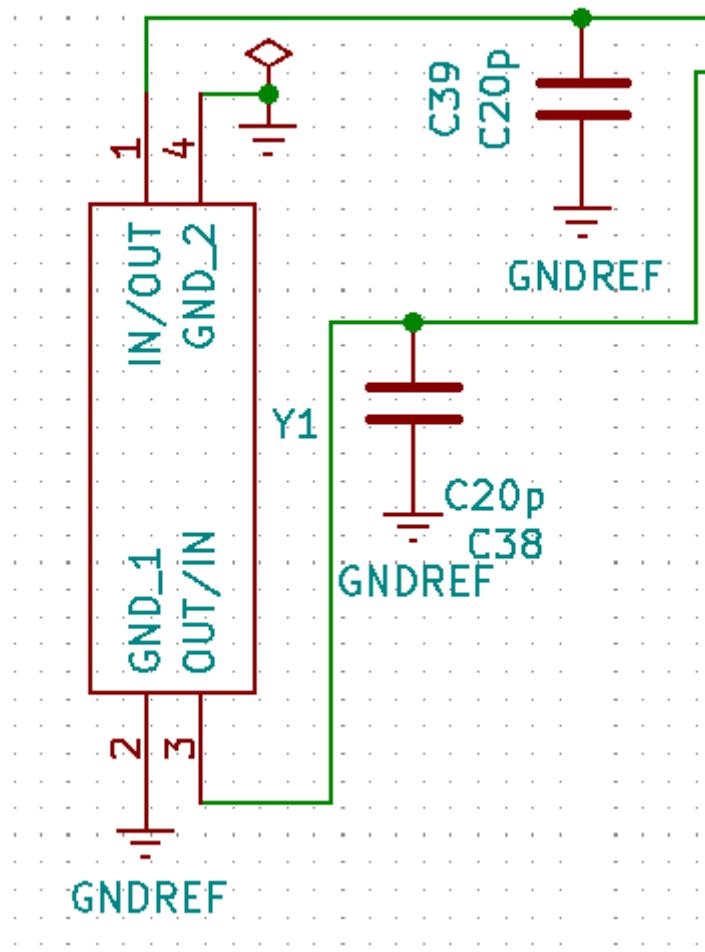


Figura 47: Esquemático del oscilador para el microcontrolador

Además de lo estrictamente necesario ya comentado, añadiremos un puerto SWD para la programación y depuración del microcontrolador mediante el dispositivo ST-Link que mencionábamos anteriormente en el apartado STM32 {3.3}. Por último, al igual que para la FPGA se han añadido 4 LEDs de diferentes colores, 2 interruptores para el usuario y uno de *reset*, así como acceso a 20 pines digitales, 4 analógicos y 1 para poder hacer *reset* sin necesidad del accionar el interruptor. En la Figura 48 se encuentra el esquemático completo del chip del microcontrolador. Algunos de los pines que se utilizan ya se han comentado y otros se comentarán en apartados posteriores.

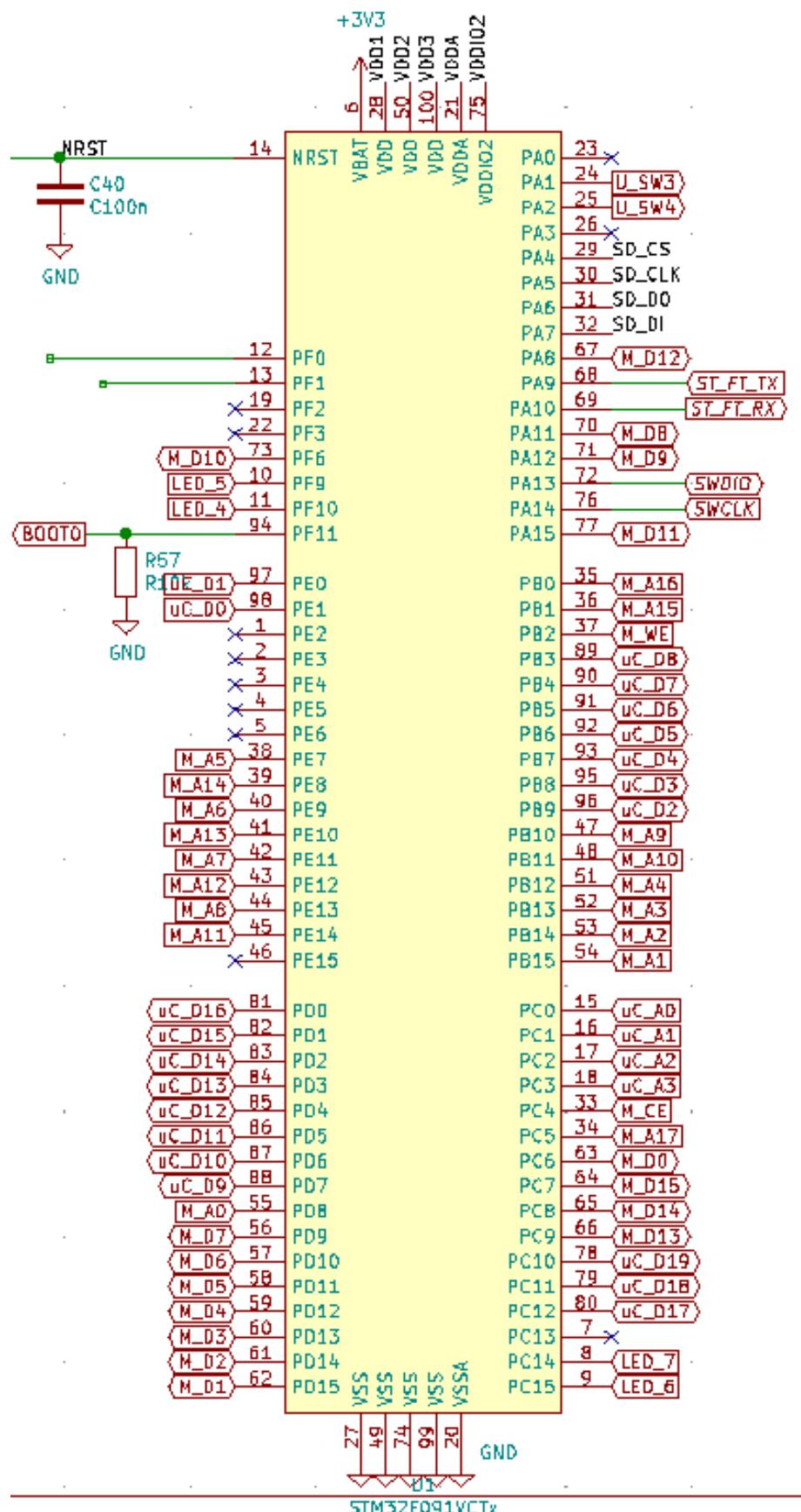


Figura 48: Esquemático del microcontrolador en KiCad

4.3.2. Tarjeta SD

SD [52] es un dispositivo en formato de tarjeta de memoria ampliamente utilizado en dispositivos portátiles como *smartphones* o cámaras digitales. Fue desarrollado por SanDisk, Panasonic y Toshiba en el año 1999 como una evolución de las tarjetas MMC.

Incorporaremos una ranura para tarjetas microSD en nuestro diseño con el fin de otorgar al mismo de un mecanismo de almacenamiento masivo externo y extraíble. Gracias a ello el usuario podrá almacenar una gran cantidad de datos procedentes del microcontrolador en la tarjeta y extraerla fácilmente para, por ejemplo, leerlo con posterioridad en un PC u otro dispositivo. O escribir datos en la tarjeta con los que luego el microcontrolador deba operar. Podemos ver la arquitectura interna de una tarjeta SD en la Figura 49.

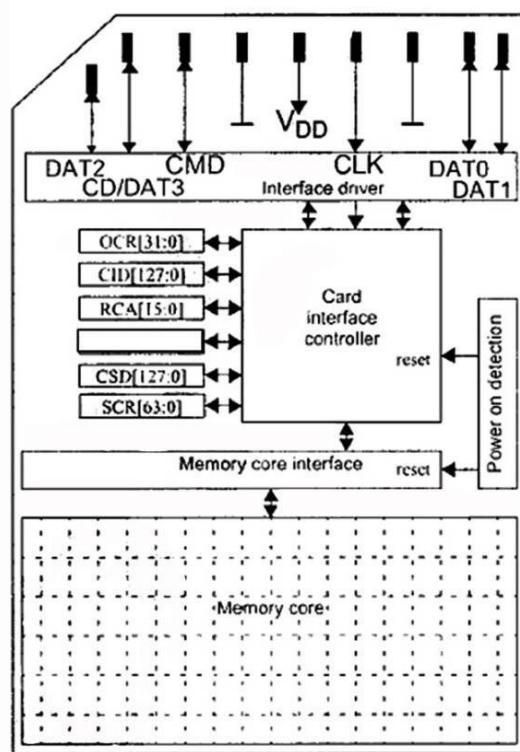


Figura 49: Arquitectura interna de una tarjeta SD [53]

Una tarjeta SD puede trabajar en modo SD o en modo SPI. Resulta obvio optar por la segunda opción dada la facilidad que tiene nuestro microcontrolador para comunicarse mediante ese protocolo. En la Tabla 7 se muestran los pines de una tarjeta microSD, así como la función que desempeñan para el modo SPI. Destacar que la alimentación debe ser a 3.3V, en nuestro caso no habría problema ya que la salida de los pines del microcontrolador está a 3.3V.

PIN NO.	PIN NAME	Descripción
1	X	Sin usar en modo SPI
2	CS	<i>Chip Select</i>
3	DI	<i>Data Input</i>
4	VDD	Alimentación +3.3V
5	SCLK	<i>Serial Clock</i>
6	VSS	Tierra
7	DO	<i>Data Out</i>
8	X	Sin usar en modo SPI

Tabla 7: Pinout tarjeta SD en modo SPI

Para nuestro diseño utilizaremos el slot de la marca Molex [54] de la Figura 50. En la Figura 51 podemos ver el esquemático del circuito correspondiente. Una vez más, se han añadido condensadores de desacople que se colocarán lo más cercano posible a las patillas de la ranura para tarjetas microSD.

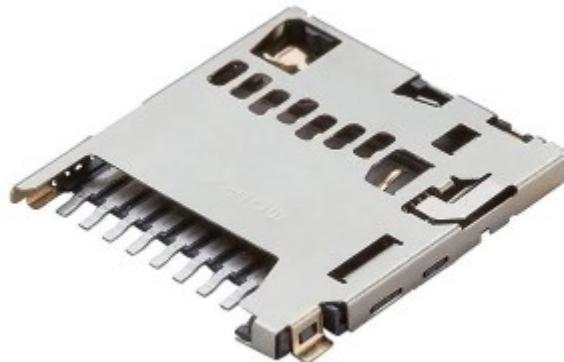
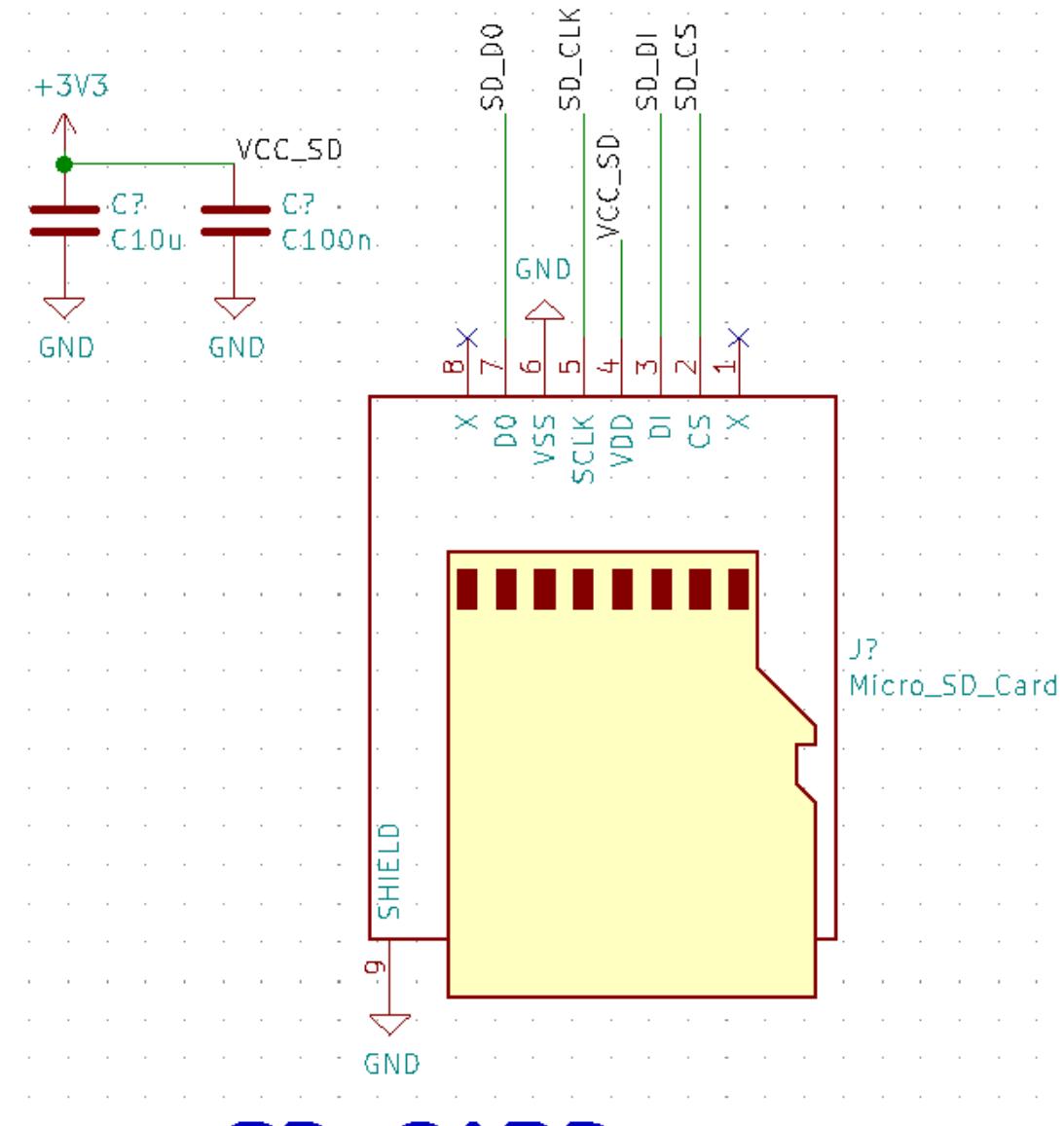


Figura 50: Slot para tarjetas microSD [55]



SD CARD

Figura 51: Esquemático en KiCad para el slot microSD

4.4. Programación de los dispositivos

En este apartado se comentará el circuito encargado para la programación de la FPGA y del microcontrolador. Como ya hemos mencionado anteriormente, nuestra intención es la compatibilidad con la IceZum Alhambra II, por lo que nos basaremos en su implementación.

El chip encargado de este proceso será el FT2232H [56] de la marca FTDI. El dispositivo en cuestión es un conversor USB-serie de alta velocidad frecuentemente utilizado en este tipo de aplicaciones. Con él, podremos mandar y recibir información desde nuestra

placa a un ordenador mediante USB. Contiene dos canales y permite la conversión de los datos a diversos protocolos como UART, FIFO, MPSSE (*Multi-Protocol Synchronous Serial Engine*). Podemos ver un diagrama de bloques del dispositivo en la Figura 52.

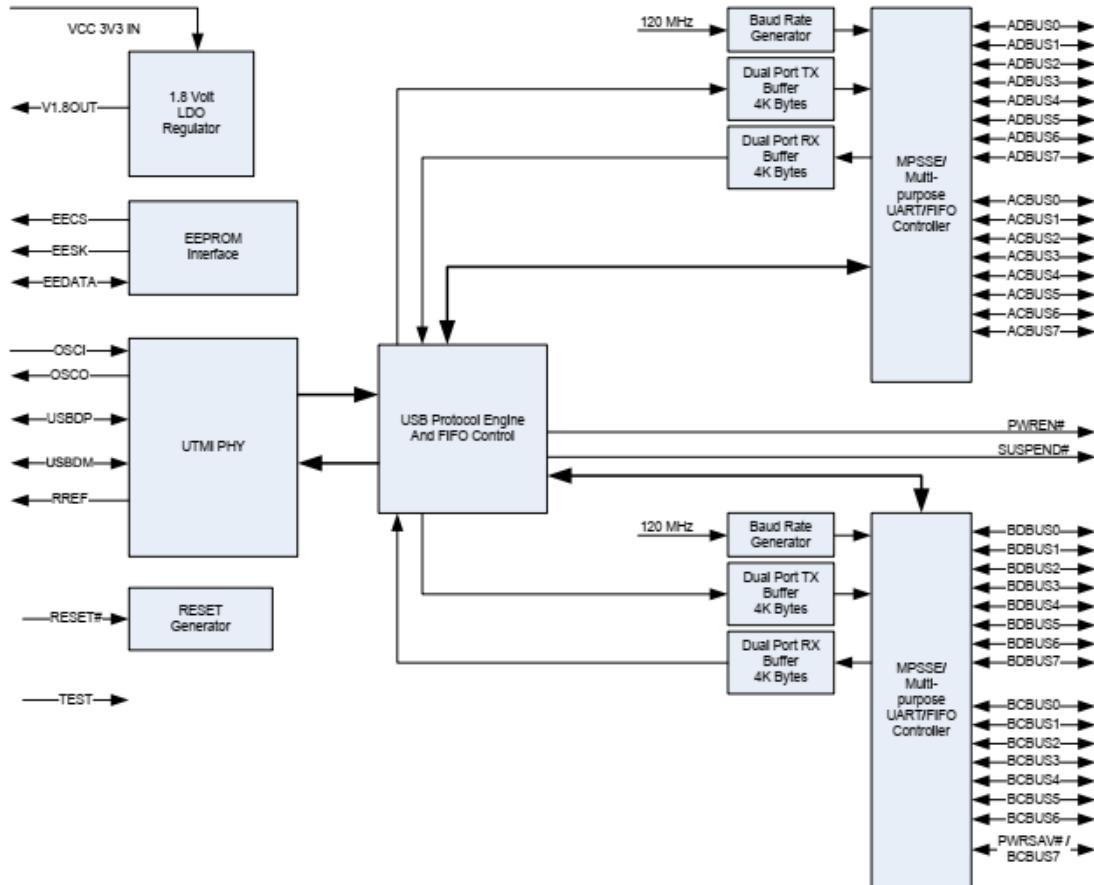


Figura 52: Diagrama de bloques del FTD2232H [53]

Todo el protocolo USB se maneja en el chip por lo que el usuario no tiene que preocuparse de programarlo según los estándares USB correspondientes. Para nuestra aplicación configuraremos el dispositivo de la siguiente forma:

- Canal A como FIFO para programar la FPGA.
- Canal B como UART para programar el microcontrolador.

Para la FPGA ya comentábamos en el apartado iCE40HX4k que, de entre todas las opciones de las que dispone, lo implementaríamos mediante una memoria flash la cual leería el chip de la FPGA mediante el banco SPI. Esta configuración se denomina “*SPI Master Configuration Interface*” [57] ya que la FPGA actúa como máster sobre la memoria Flash como podemos ver en la Figura 53.

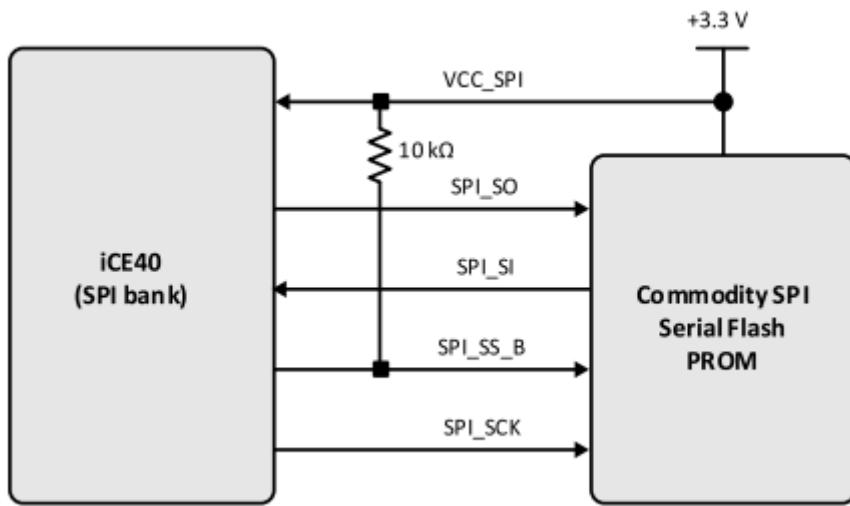


Figura 53: Configuración SPI Master [54]

La definición de los pines presentes en la Figura 53 la encontramos en la Tabla 8.

Nombre de la señal	Descripción
VCC_SPI	+3.3V
SPI_SO	SPI Serial Output desde la FPGA
SPI_SI	SPI Serial Input hacia la FPGA
SPI_SS	SPI Slave Select
SPI_SCK	SPI Slave Clock

Tabla 8: Descripción de las señales para la memoria Flash

La memoria flash no viene incluida en el chip de la FPGA por lo que deberemos de proporcionarla nosotros. Para el correcto funcionamiento del sistema, debe de cumplir con una serie de características:

- Operar entre 1.8V y 3.3V.
- Debe soportar el comando 0x0B “Fast Read” así como el 0xB9 “Deep Power Down”.
- Contener, al menos, 135183 bytes.
- Soportar la frecuencia de funcionamiento máxima de la FPGA.
- Debe poder aceptar comandos 10us después de cumplir con sus requisitos de funcionamiento (*power-on conditions*). Se puede no cumplir esta característica manteniendo *CRESET_B* en bajo, bien mediante hardware o mediante software, hasta que la memoria está en completo funcionamiento.

En nuestro caso hemos optado por la memoria Flash W25Q32JV de Winbond [58], que cumple con todos los requisitos anteriores salvo el último, que se solucionará mediante software al igual que lo hace la IceZUM Alhambra II. En la Figura 54 encontramos la

conexión en esquemático entre la memoria y la FPGA, así como las señales que salen hacia el FTDI.

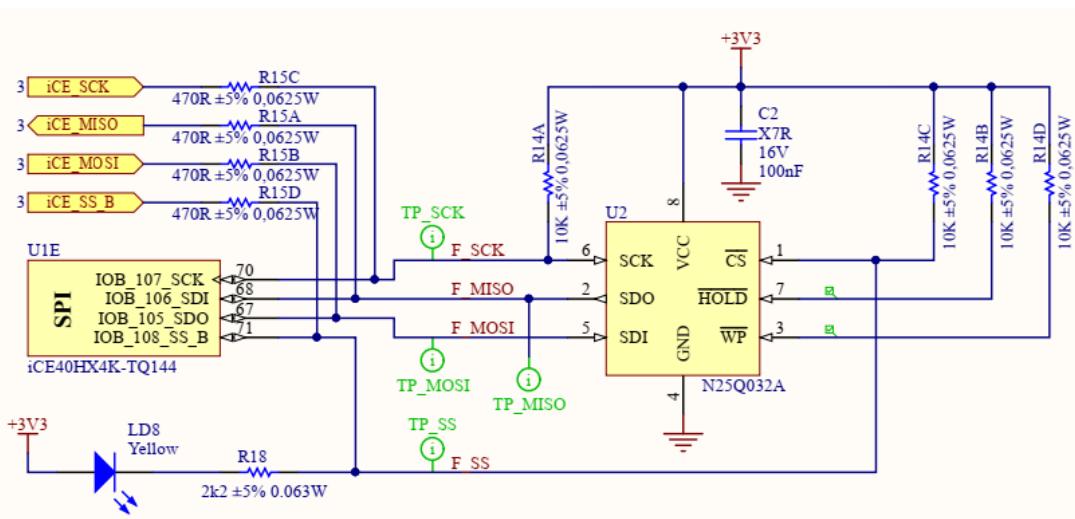


Figura 54: Esquemático de la memoria Flash en KiCad [59]

Se ha incluido un LED con una resistencia en serie para indicar cuándo la memoria está siendo programada o cuándo la FPGA está leyendo su contenido. También se ha añadido un condensador de desacoplo para la alimentación de la memoria Flash y resistencias de *pull-up* relativamente grandes para asegurar un bajo consumo en *standby*. Como se puede ver, las señales del FTDI y de la FPGA están en paralelo. Cuando se está programando la memoria procederán del FTDI y cuando no, la FPGA leerá de la memoria Flash.

Hemos comentado que la memoria se programa mediante SPI, sin embargo, hemos dicho que el canal correspondiente del FTDI, el A, será FIFO. Esto se debe, a que configurando el canal como FIFO obtenemos los 8 bits como datos de I/O. Ya debe ser el software, IceStudio, el que se encargue de adecuar dichos bits de datos a los correspondientes con el protocolo SPI.

Para programar el microcontrolador [60] simplemente deberemos de introducir los datos mediante UART en los pines indicados para ello (*PIN9* y *PIN10*) además de seleccionar el arranque mediante SRAM con el pin *BOOT0*. En la Figura 55 podemos ver el proceso de selección del *bootloader* para el microcontrolador.

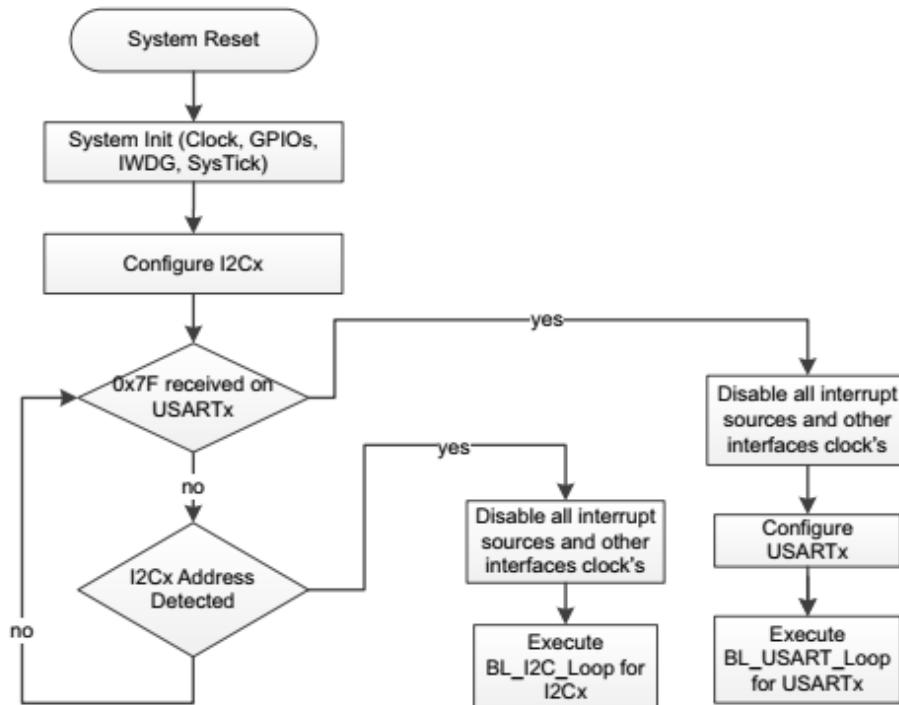


Figura 55: Diagrama de selección del bootloader para el STM32F091VC [60]

La descripción de pines que utilizaremos del canal B para SPI se puede encontrar en la Tabla 9.

PIN NO.	NOMBRE	DESCRIPCIÓN
38	TXD	Salida del transmisor
39	RXD	Entrada del receptor
55	TXLED	LED TXD. Valor bajo cuando se envía un dato hacia el microcontrolador.

Tabla 9: Descripción de pines del canal B para protocolo UART

Cabe destacar que, aunque ambos canales utilicen protocolos diferentes, la información en ambos canales será idéntica por lo que deberemos indicar de forma externa qué dispositivo es el que estamos programando en cada momento. Para ello utilizaremos 4 pines auxiliares como los de la Figura 56. Así, si colocamos un jumper sobre los pines *iCE_CS_B* e *iCE_CS*, la memoria SPI de la FPGA recibirá la señal *chip enable* y se cargará con la información del PC. Si ponemos un jumper sobre los pines *BOOT0* y *+3V3*, el microcontrolador arrancará en el modo *empty* que ya hemos comentado con anterioridad y procederá a programar su memoria Flash con el contenido que le llegue del PC mediante UART. Si no colocamos ninguno de los dos, ambos dispositivos ejecutarán el contenido ya cargado en sus memorias.

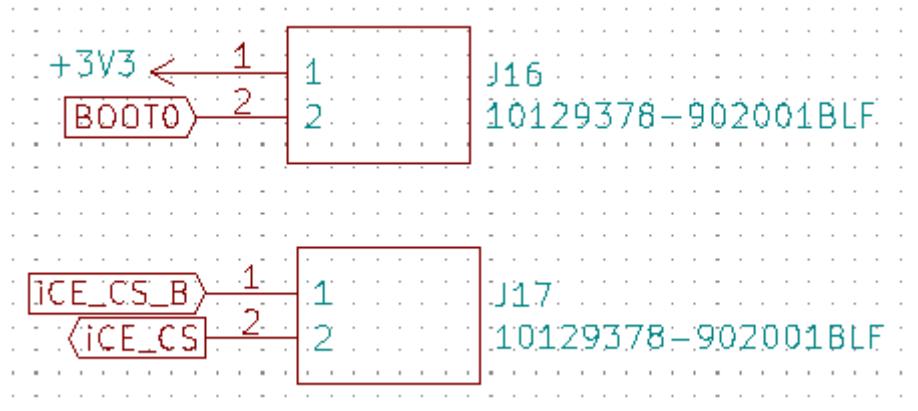


Figura 56: Pines para la selección del dispositivo a programar

Por último, vamos a comentar el circuito resultante para el FTDI. En la Figura 57 encontramos el diseño del esquemático para el FTDI en KiCad. De la figura, cabe destacar que:

- Al pin 2 se ha conectado un oscilador CMOS de precisión a 12MHz. Es el mismo que la señal de reloj externa que usaremos para la FPGA.
- 93LC56C es una memoria EEPROM de 2K. Su función es almacenar la configuración deseada para el FTDI.
- El pin **TEST** se ha conectado a masa como indica el fabricante.
- El pin **REF** se ha conectado a tierra mediante una resistencia de $12\text{k}\Omega$ como indica el fabricante y el de **RESET** a **+3.3V** ya que se activa con un valor bajo.
- Los pines **PWREN** y **SUSPEND** no se utilizan.
- Los pines **DM** y **DP** contienen las dos señales diferenciales USB.

En la Figura 58 podemos ver las distintas señales de alimentación del dispositivo. Dichas señales son:

- **VREGIN**: Entrada de **+3.3V** del regulador integrado en el FTDI. Desacoplado con **100nF**.
- **VREGOUT**: Salida de **+1.8V** del regulador integrado. Desacoplado con un condensador de **10uF** y conectado a **VCORE**.
- **VCORE**: Entrada principal de alimentación de **+1.8V**. Se ha conectado desde **VREGOUT** y cada entrada se ha desacoplado con un condensador de **100nF**.
- **VLPLL**: Entrada de alimentación de **+3.3V** del PLL. Desacoplado mediante un filtro LC.
- **VPHY**: Entrada de alimentación USB PHY de **+3.3V**. Desacoplado mediante un filtro LC.
- **GND** y **AGND**: Tierra y tierra analógica respectivamente.

Por último, el código que ha de contener la memoria EEPROM del FTDI se muestra en la Figura 59.

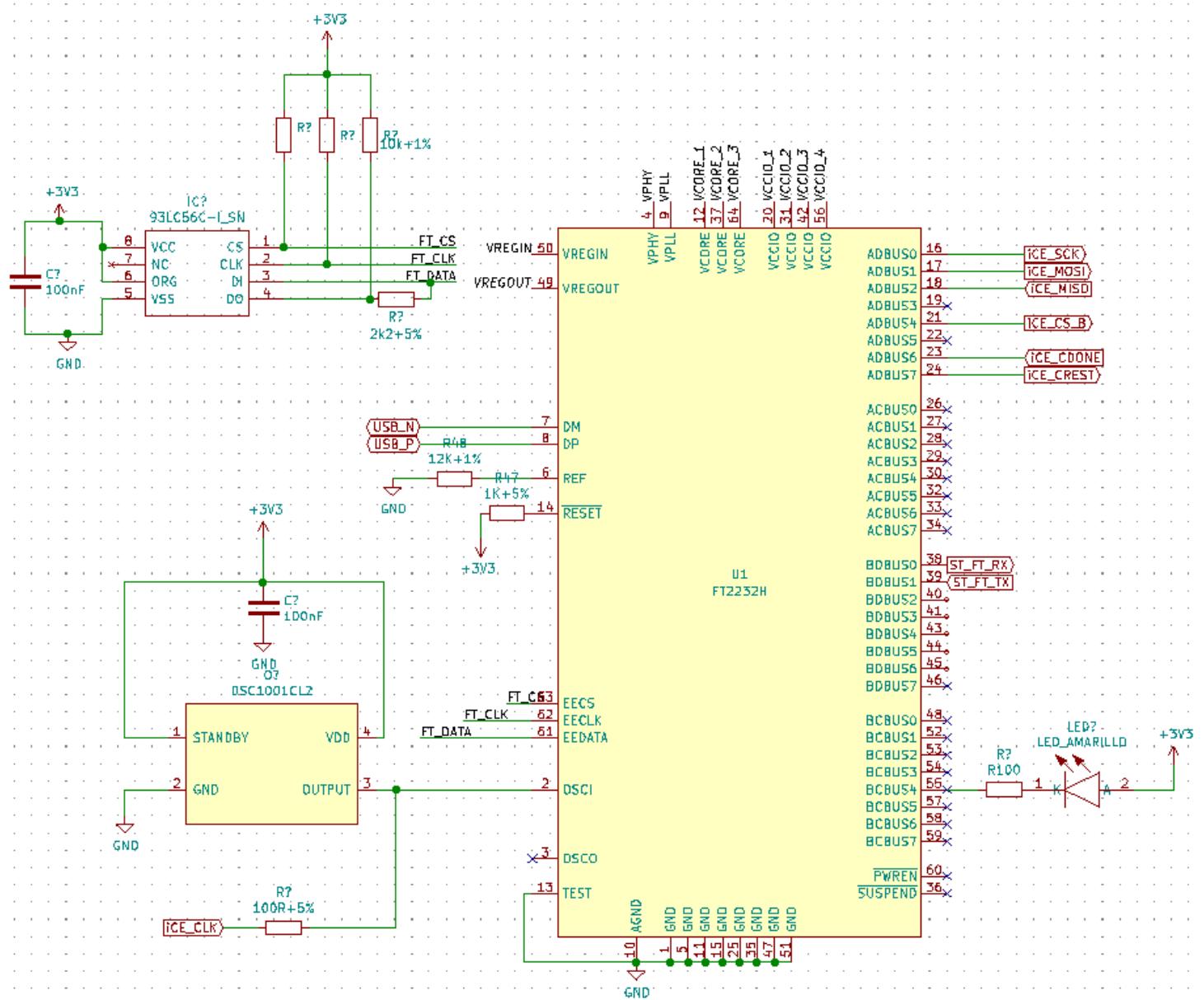


Figura 57: Esquemático del FTDI en Kicad

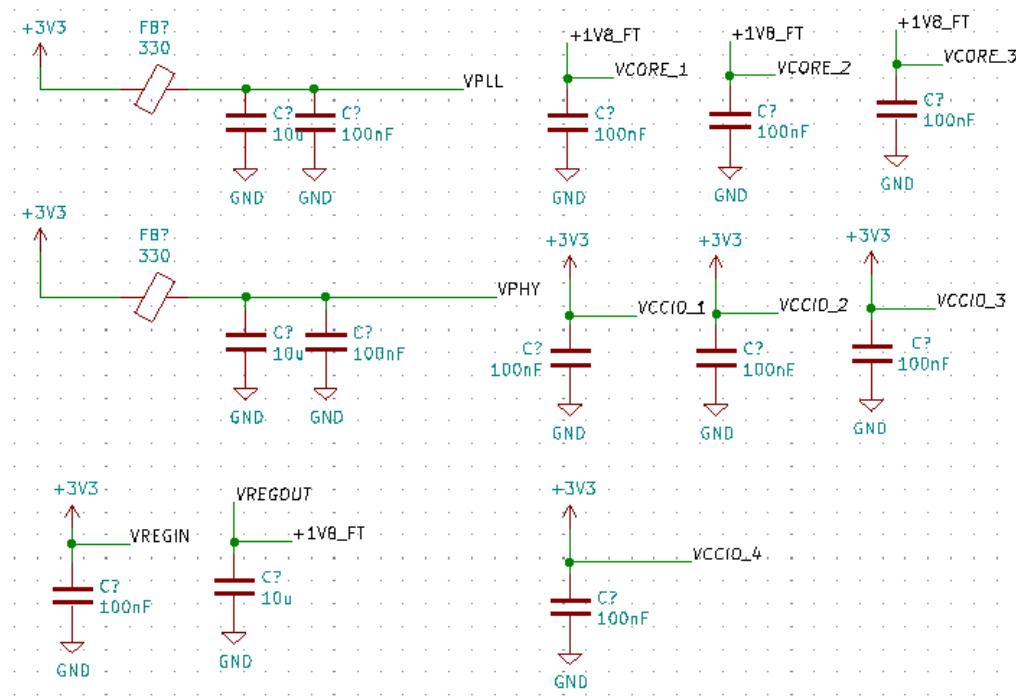


Figura 58: Esquemático de la alimentación para el FTDI en KiCad

```

1  #-- Chip type: FT2232H
2  eeprom_type = 0x0056
3
4  #-- USB descriptor
5  vendor_id=0x0403
6  product_id=0x6010
7  usb_version=4369
8
9  #-- USB config descriptor
10 remote_wakeup=false
11 selfPowered=false
12 suspend_pull_downs=false
13 max_power=500
14 use_serial=false
15 change_usb_version=false
16
17 #-- USB string descriptor
18 manufacturer="Mareldem"
19 product="Alhambra II v1.0 - P01-004"
20
21 #-- Hardware specific
22
23 in_is_isochronous=false
24 out_is_isochronous=false
25
26 ## Port A
27 cha_type=FIFO
28 cha_vcp=false
29
30 ## Port B
31 chb_type=UART
32 chb_vcp=true
33
34 filename="A2-eeprom-image.bin"

```

Figura 59: Contenido de la EEPROM del FTDI

4.5. Diseño SRAM y bus de comunicación

Como se comentaba al principio de esta memoria, en el apartado Idea inicial de diseño {2.1}, alguno de los principales inconvenientes con lo que se encontró el estudiante a la hora de implementar su aplicación eran:

- Por una parte, la escasa memoria de alta velocidad de la que disponían tanto el microcontrolador utilizado, Arduino Uno, como la FPGA, IceZUM Alhambra II.
- Por otra parte, la escasez de pines de propósito general de ambas placas limitaba la posibilidad de comunicación entre los dos dispositivos.

Para solventar dichas limitaciones, en el diseño de nuestra placa se ha incorporado una memoria SRAM que, conectada al microprocesador y a la FPGA simultáneamente, permiten solucionar ambos inconvenientes. Por un lado, ampliando la memoria total del sistema, y por otro, estableciendo un bus de comunicación microcontrolador-memoria-FPGA que permita al usuario intercambiar datos de forma conveniente entre los dos dispositivos mediante la SRAM.

La memoria SRAM elegida para la implementación es la IS61WV25616EDBLL [61], una memoria RAM estática de alta velocidad con 256K palabras de 16 bits, lo que da lugar a un total de 4Mb de memoria. Es asíncrona, tiene salida triestado y un consumo en activo de 85mW. Además, cuenta con módulos de detección y corrección de errores. Podemos ver su diagrama de bloques en la Figura 60.

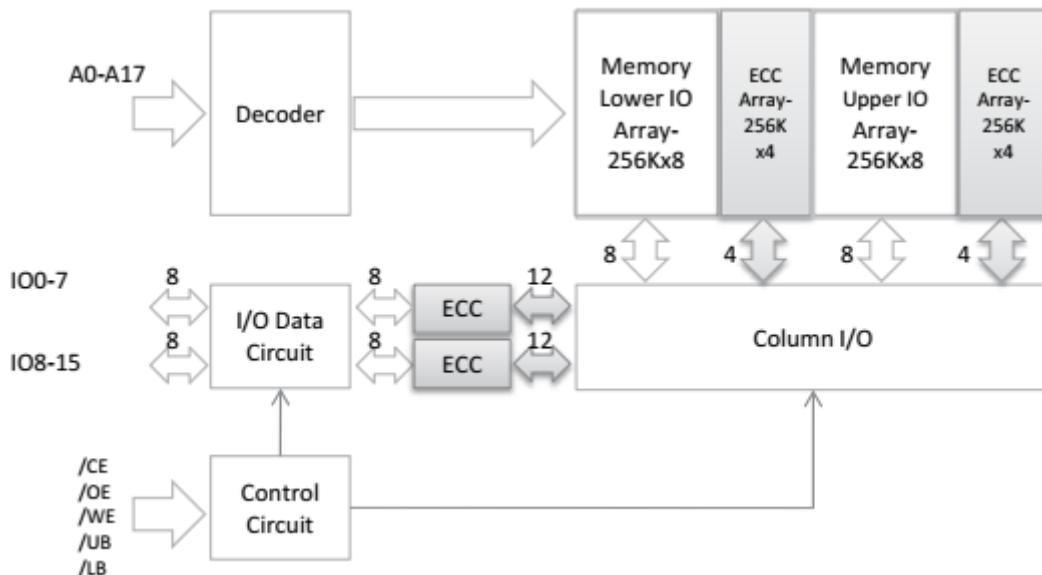


Figura 60: Diagrama de bloques de la memoria SRAM [61]

En la Tabla 10 se muestra la descripción de pines de la memoria y en la Figura 61 el pinout del encapsulado.

PIN NO.	Descripción
A0 - A17	<i>Address Input</i>
I/O0 - I/O15	<i>Data Input/Output</i>
CE	<i>Chip enable input</i> (activo en valor bajo)
OE	<i>Output enable input</i> (activo en valor bajo)
WE	<i>Write enable input</i> (activo en valor bajo)
LB	<i>Lower-byte control</i> (activo en valor bajo)
UB	<i>Upper-byte control</i> (activo en valor bajo)
NC	No conectar
V _{DD}	Alimentación
GND	Masa

Tabla 10: Descripción de pines de la memoria SRAM

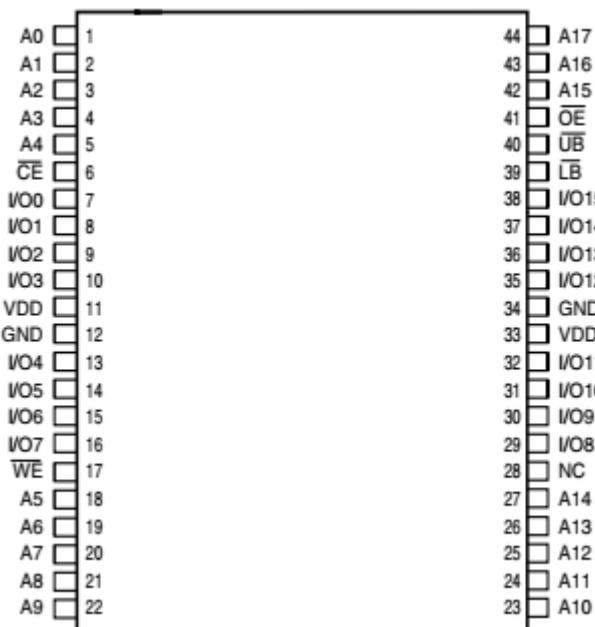


Figura 61: pinout de la memoria SRAM [61]

Los pines de control *LB* y *UB* no los usaremos y los fijaremos a masa. El de *Output enable* (*OE*) tampoco será necesario ya que la señal *WE* controla tanto la lectura como la escritura de la memoria. Por tanto, podemos representar la tabla de verdad de la memoria en la Tabla 11. De ella podemos deducir que siempre que la memoria no esté siendo utilizada, es conveniente forzar la señal *CE* a un valor alto para reducir el consumo de energía y hacer que la salida esté en alta impedancia.

Modo	WE	CE	I/O	Corriente en V _{DD} máxima (mA)
Not Selected	X	H	High-Z	$I_{SB}=15$
Read	H	L	D _{OUT}	$I_{CC}=50$
Write	L	L	D _{IN}	$I_{CC}=50$

Tabla 11: Tabla de verdad de la memoria SRAM

Dicho esto, el esquemático queda como en la Figura 62 donde sólo se ha puesto las masas, nombres a las líneas y añadido condensadores de desacoplo cerámicos en las entradas de alimentación.

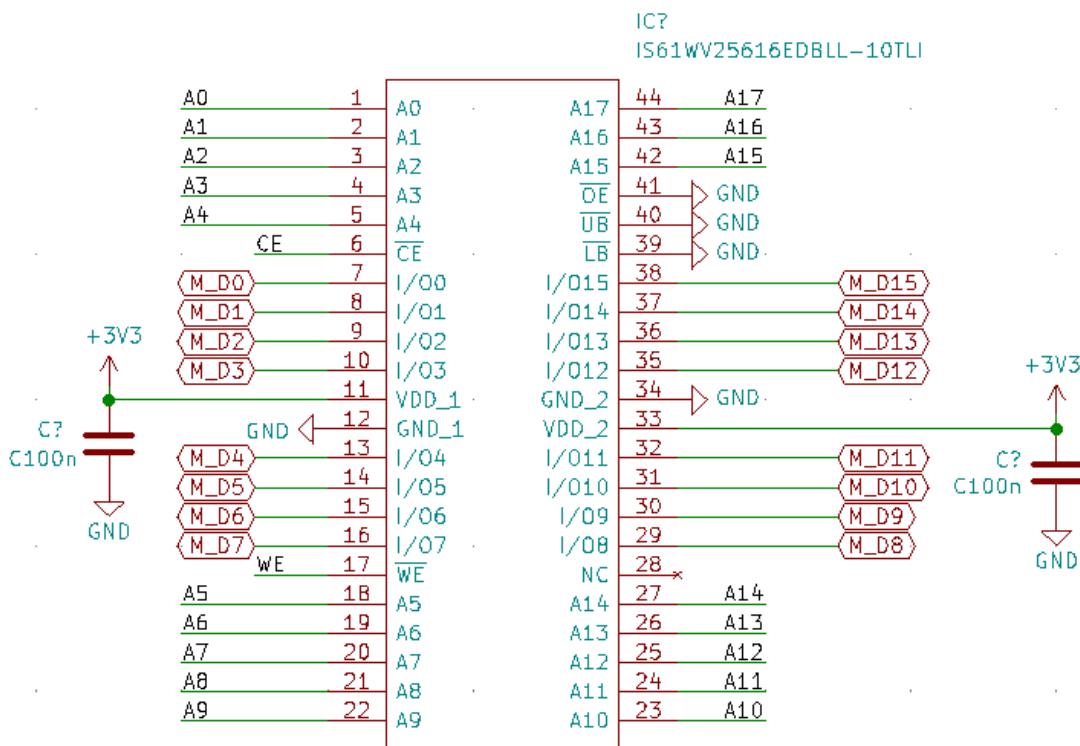


Figura 62: Esquemático de la memoria SRAM en KiCad

Por otra parte, a la hora de diseñar sistemas con dispositivos CMOS se debe prestar especial atención a los casos en los que todas las líneas de los buses están inactivas. Esto puede desembocar en niveles indefinidos, aumentando el consumo del dispositivo e incluso provocar oscilación que afecte tanto a los componentes a nivel interno en términos de funcionalidad, como a la compatibilidad electromagnética del sistema completo [62].

La entrada de un circuito CMOS es un inversor como el de la Figura 63 que desacopla el circuito interno de la señal externa.

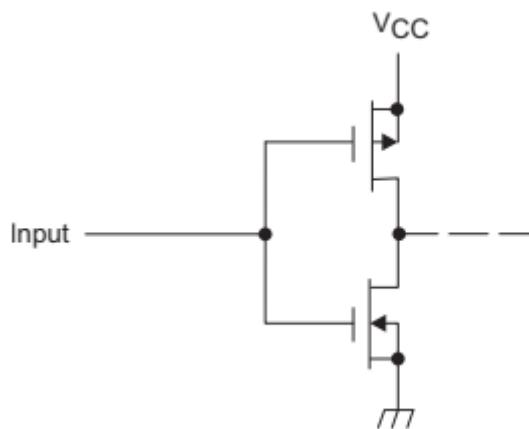


Figura 63: Etapa de entrada de un circuito CMOS [62]

Si la entrada es de un nivel adecuado, conducirá o bien el transistor P si es un valor bajo o el N si es un valor alto. En cualquiera de los casos, el transistor opuesto entra en corte y no circula corriente por ellos. Es por ello que el consumo en reposo de los circuitos CMOS es tan bajo. Sin embargo, si la entrada está entre los dos valores lógicos admitidos, ambos transistores conducen parcialmente aumentando el consumo del circuito. Además, si la entrada cambia de forma muy lenta de un valor a otro, el circuito comenzará a oscilar.

Cuando las líneas son unidireccionales, el fenómeno antes expuesto deja de ser un problema ya que siempre hay un controlador activo al final de la línea que asegura un valor lógico definido. Sin embargo, en un sistema de buses (Figura 64) como en nuestro caso, las líneas son bidireccionales y se debe prestar especial atención cuándo todas las salidas están inactivas. Al no haber un controlador que defina el valor de la línea, las corrientes de fugas de los componentes conectados provocan un nivel de tensión indeterminado.

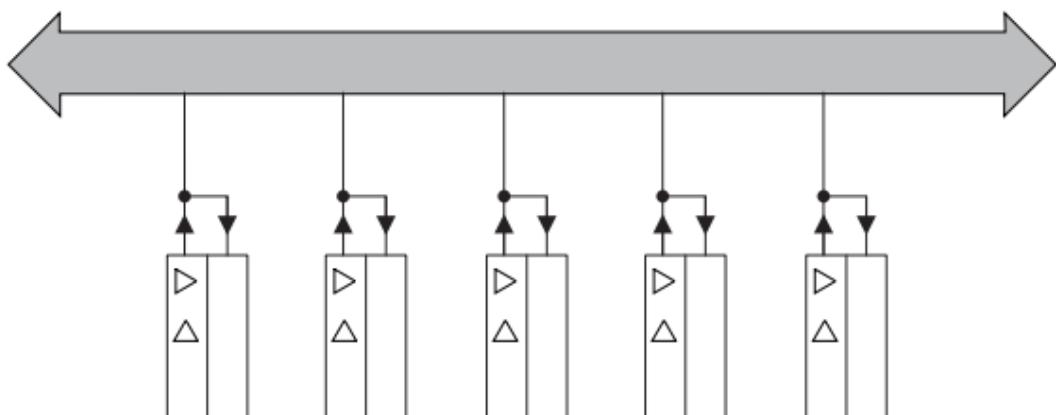


Figura 64: Transmisión bidireccional en un sistema de buses [62]

Para solucionarlo podemos optar por varios métodos:

- La forma más sencilla y que no requiere de ningún componente extra es realizar un control apropiado del bus. Es decir, tener en consideración que la duración del bus en estado inactivo sea lo suficientemente corta para que no produzca un cambio en el nivel de tensión. El problema es que las corrientes de fugas son muy dependientes de la temperatura, llegando a doblar su valor por cada 10°C, por lo que tener en consideración este factor complica mucho la ejecución de esta solución.
- Otro método similar al anterior, utilizado por los buses PCI, es que el último componente que ha hecho uso del sistema se mantenga activo. Aquí, el problema recae en que aumenta el consumo y se necesita una cierta comunicación entre los dispositivos para solicitar y acceder al bus.
- Una solución ampliamente utilizada es implementando resistencias de *pull-up* o *pull-down* que aseguren el valor lógico de la línea (Figura 65). La resistencia debe ser lo suficientemente grande para no aumentar en exceso el consumo del sistema, pero no tanto que haga que el sistema se ralentice. La resistencia de *pull-up* se puede calcular con la Ecuación 2:

$$R_p = \frac{t_r}{2.2 \cdot C_s \cdot n}$$

Ecuación 2: Cálculo de la resistencia de *pull-up*

Donde t_r es el tiempo de subida necesario, C_s la capacidad de la línea y n el número de dispositivos conectados. La hoja de características de la memoria SRAM indica que el *slew rate* debe ser superior a 1ns/V. Como alimentamos a 3.3V, el tiempo de subida y bajada debe de ser 3.3ns. La capacidad C_s de la línea la estimamos a 20pF y como conectamos un microcontrolador y una FPGA, n será igual a 2. El resultado es una resistencia inferior a 400Ω, lo que provocaría un consumo excesivo en reposo.

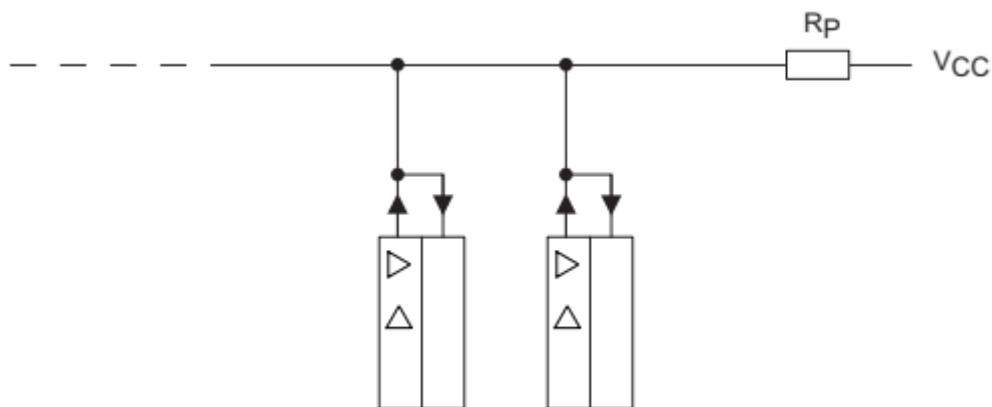


Figura 65: Resistencia de *pull-up* en una línea de bus [62]

- Otra alternativa es un circuito de retención de buses similar al de la Figura 66. El funcionamiento de estos circuitos es realimentar la salida de una puerta no inversora a su entrada mediante una resistencia de *feedback*, creando así un biestable. Imaginemos que un dispositivo fuerza la línea del bus a un estado de valor alto. La salida de la puerta no inversora será también un valor alto y por la resistencia no circulará corriente. Si el dispositivo pasa a un estado inactivo, la resistencia de *feedback* fuerza el valor de la línea al valor anterior y la única corriente que circula por ella son las de fugas. El mayor consumo en este circuito se produce durante las transiciones de valor bajo a alto y viceversa.

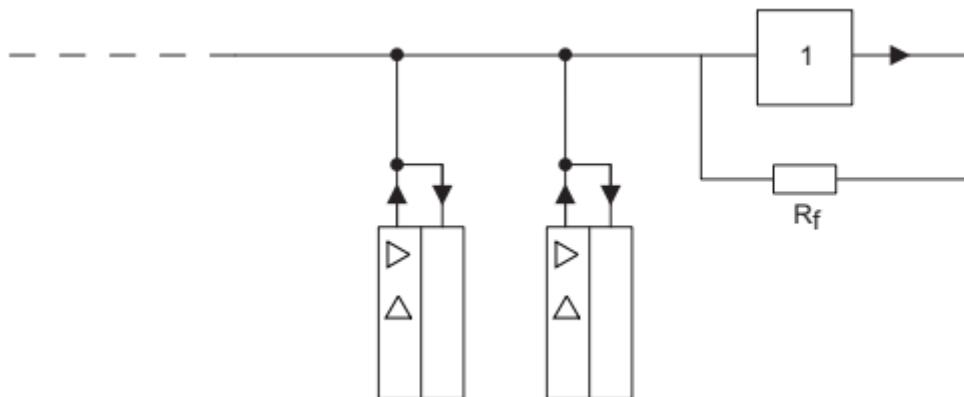


Figura 66: Circuito de retención de buses [62]

- Por último, y basándose en la solución anterior, existen circuitos integrados de retención de buses. Para nuestra placa utilizaremos el SN74ALVCH16827 [63] que permite forzar las entradas inactivas a un valor lógico además de poder forzar la salida a un estado de alta impedancia, todo ello sin necesidades de resistencias externas. El circuito simplificado del integrado se encuentra en la Figura 67. Los transistores Q1 y Q2 aíslan el circuito interno del exterior, además de amplificar e invertir la señal. Los transistores Q3 y Q4 son los encargados de retener el valor lógico a un nivel adecuado, volviendo a invertir la señal y realimentándose con la primera etapa. Para visualizarlo imaginemos que la entrada se define a un valor alto y que, por lo tanto, la salida de la segunda etapa será también de valor alto. Entonces, el transistor Q3 conducirá. Si la entrada disminuye por cualquier motivo, Q3 suministra corriente y compensa la disminución. Si el valor es bajo, es el transistor Q4 el encargado de mantener la entrada a valor bajo. El pinout del dispositivo está representado en la Figura 68 y su tabla de verdad en la Tabla 12.

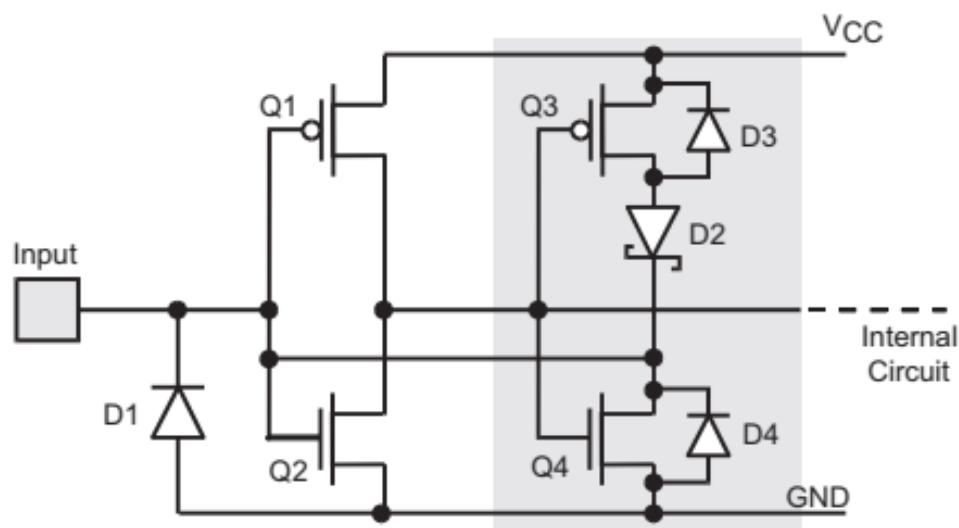


Figura 67: Circuito simplificado del integrado de retención de buses [62]

1OE1	1	56	1OE2
1Y1	2	55	1A1
1Y2	3	54	1A2
GND	4	53	GND
1Y3	5	52	1A3
1Y4	6	51	1A4
Vcc	7	50	V _{CC}
1Y	8	49	1A5
1Y6	9	48	1A6
1Y7	10	47	1A7
GND	11	46	GND
1Y8	12	45	1A8
1Y9	13	44	1A9
1Y10	14	43	1A10
2Y1	15	42	2A1
2Y2	16	41	2A2
2Y3	17	40	2A3
GND	18	39	GND
2Y4	19	38	2A4
2Y5	20	37	2A5
2Y6	21	36	2A6
V _{CC}	22	35	V _{CC}
2Y7	23	34	2A7
2Y8	24	33	2A8
GND	25	32	GND
2Y9	26	31	2A9
2Y10	27	30	2A10
2OE1	28	29	2OE2

Figura 68: Pinout del SN74ALVCH162827 [63]

	OE1	OE2	A	Y
<i>L</i>	<i>L</i>	<i>L</i>	<i>L</i>	
<i>L</i>	<i>L</i>	<i>H</i>	<i>H</i>	
<i>H</i>	<i>X</i>	<i>X</i>	<i>Z</i>	
<i>X</i>	<i>H</i>	<i>X</i>	<i>Z</i>	

Tabla 12: Tabla de verdad del integrado de retención de buses

En la Figura 69 podemos ver el esquemático del circuito de retención de buses implementado en KiCad. Las señales que utilizarán este circuito son sólo las de control de la memoria y las de dirección. Dado que el circuito integrado cuenta con 4 pines de alimentación se le ha añadido 4 condensadores de desacople de 100n, uno para cada pin. Las señales de control *OE* sirven para poner las líneas en alta impedancia cuando su valor es alto como se puede ver en la Tabla 12. Dado que es precisamente esto lo que pretendemos evitar con este circuito no haremos uso de esta función y las fijaremos a un valor lógico bajo mediante resistencias de *pull-down*. Recordemos que tanto la FPGA como el microcontrolador sí que podrán poner estas líneas en alta impedancia pero que este dispositivo se encargará de que a la memoria SRAM siempre llegue un valor definido.

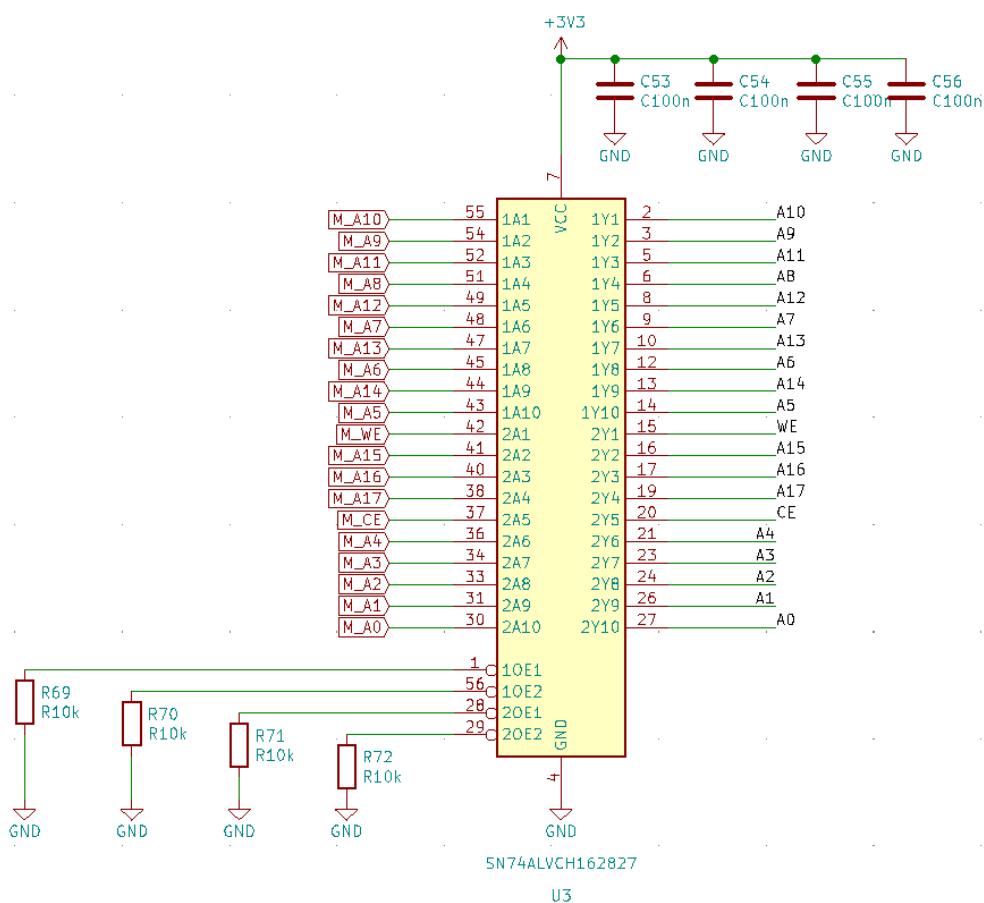


Figura 69: Circuito de retención de buses en KiCad

Para terminar esta sección haremos uso de la IceZum Alhambra II y de la placa entrenadora STM32F4DISCOVERY para comprobar la sinergia de ambos dispositivos con

una memoria SRAM y comentar como se debe gestionar. Para ello contamos con un módulo de memoria SRAM IS62WV51216BLL [64] de la marca Waveshare [65] como el de la Figura 70. El módulo cuenta con una memoria SRAM de 2MB organizada en 128K palabras de 16 bits. Al ser un circuito de alta velocidad, el prototipado se vuelve complejo y propenso a ruidos. Para simplificarlo y procurar el mejor funcionamiento posible, a pesar del uso de una *protoboard* y de cables, utilizaremos sólo las líneas necesarias. Estas líneas son:

- Pines de alimentación y tierra.
- Sólo un pin de dirección, los demás a tierra.
- Sólo un pin de dato, los demás a tierra.
- Líneas de control *WE* y *OE* además de anclar *CE* a 0.

La tabla de verdad de la memoria la observamos en la Tabla 13.



Figura 70: Módulo SRAM IS62WV51216BLL [65]

	WE	OE	Output
<i>Output Disabled</i>	H	H	Z
	H	L	DOUT
	L	H	DIN

Tabla 13: Tabla de verdad para el módulo SRAM

Una vez montado el circuito pasamos a la configuración del microcontrolador y de la FPGA. Como ya se comentó en el apartado Diferencias y similitudes entre un microcontrolador y una FPGA {2.5}, el procedimiento para ambos dispositivos es completamente diferente. Nuestra aplicación consistirá en:

1. El microcontrolador escribirá un 1 en la dirección de memoria 1.
2. El microcontrolador indicará a la FPGA que ha terminado la operación.
3. La FPGA leerá el dato en la dirección 1.
4. Si el dato es un 1, encenderá un LED.

Con esta aplicación simplemente se trata de mostrar el procedimiento que debe seguir el sistema para el uso compartido de la memoria SRAM. Si bien es cierto que no es la única forma, siempre se debe de tener cuidado a la hora de acceder a la memoria para evitar colisiones entre el microcontrolador y la FPGA, así como de no ocupar las líneas de la memoria cuando no se esté haciendo uso de ella. Es el usuario final el que debe considerar esta limitación y adecuarla a su uso en particular. En este caso en concreto, para simplificarlo, hemos supuesto que el microcontrolador sólo escribe y la FPGA sólo lee.

El código implementado en el microcontrolador podemos verlo en la Figura 71. Lo más destacable del código es como actuamos con los pines de entrada/salida. Al acceder a una memoria que está siendo compartida con más dispositivos es imprescindible que los dispositivos que no se estén usando la memoria no interfieran en ella. Esto lo conseguimos gracias a la configuración de pines del microcontrolador. Si mientras el microcontrolador no está usando la memoria definimos los pines que acceden a esta como entrada en lugar de salida, evitamos que el microcontrolador interfiera en la memoria, quedando ésta libre para los demás dispositivos. Lo mismo haremos más adelante, aunque de otra forma, con la FPGA. Por lo demás, el código es bastante sencillo. Declaramos los pines que van a utilizarse y una variable que indique si hemos escrito en la memoria o no. Al entrar en la función *loop* escribimos en la memoria, cambiamos el valor de esa variable para no volver a escribir y le indicamos a la FPGA que hemos terminado el proceso de escritura.

Para la configuración de la FPGA la idea es similar pero la técnica es completamente diferente. Normalmente para secuenciar procesos en una FPGA se recurren a máquinas de estado que gestionan todo el proceso. Sin embargo, al ser una aplicación tan sencilla podremos hacerlo todo de forma combinacional. En la Figura 72 muestra una captura del código en IceStudio. Para ello nos basamos en el bloque *InOut* [66], el cual podemos ver en más detalle en la Figura 73.

```
1 // Arduino pin numbers
2 const int WE = PA1;
3 const int OE = PA0;
4 const int ADD0 = PA2;
5 const int DAT0 = PA3;
6 const int FPGA= PB1;
7
8 bool w=true; //variable de escritura
9
10 void setup() {
11     pinMode(WE, INPUT); //declaramos pines como entrada
12     pinMode(OE, INPUT);
13     pinMode(ADD0, INPUT);
14     pinMode(DAT0, INPUT);
15     pinMode(FPGA, OUTPUT);
16 }
17
18 void loop() {
19
20     if(w==true){ //si aun no se ha escrito, escribimos
21         escribir();
22         w=false; //cambiamos el valor de la variable
23         digitalWrite(FPGA,HIGH); //indicamos a la FGPA que hemos terminado
24     }
25 }
26
27
28 void escribir(){ //funcion escribir
29
30     pinMode(WE, OUTPUT); //pines como salida
31     pinMode(OE, OUTPUT);
32     pinMode(ADD0, OUTPUT);
33     pinMode(DAT0, OUTPUT);
34
35     digitalWrite(ADD0, HIGH); //escribimos 1 en la direccion 1
36     digitalWrite(DAT0, HIGH);
37     delay(50); //pequeño delay para evitar fallos
38
39     digitalWrite(WE, LOW); //comando escribir
40     delay(50); //pequeño delay para evitar fallos
41
42     digitalWrite(WE, HIGH); //reset a los pines
43     digitalWrite(ADD0, LOW);
44     digitalWrite(DAT0, LOW);
45
46     pinMode(WE, INPUT); //volvemos a definir los pines como entrada
47     pinMode(OE, INPUT);
48     pinMode(ADD0, INPUT);
49     pinMode(DAT0, INPUT);
```

Figura 71: Código del microcontrolador para escribir en la SRAM

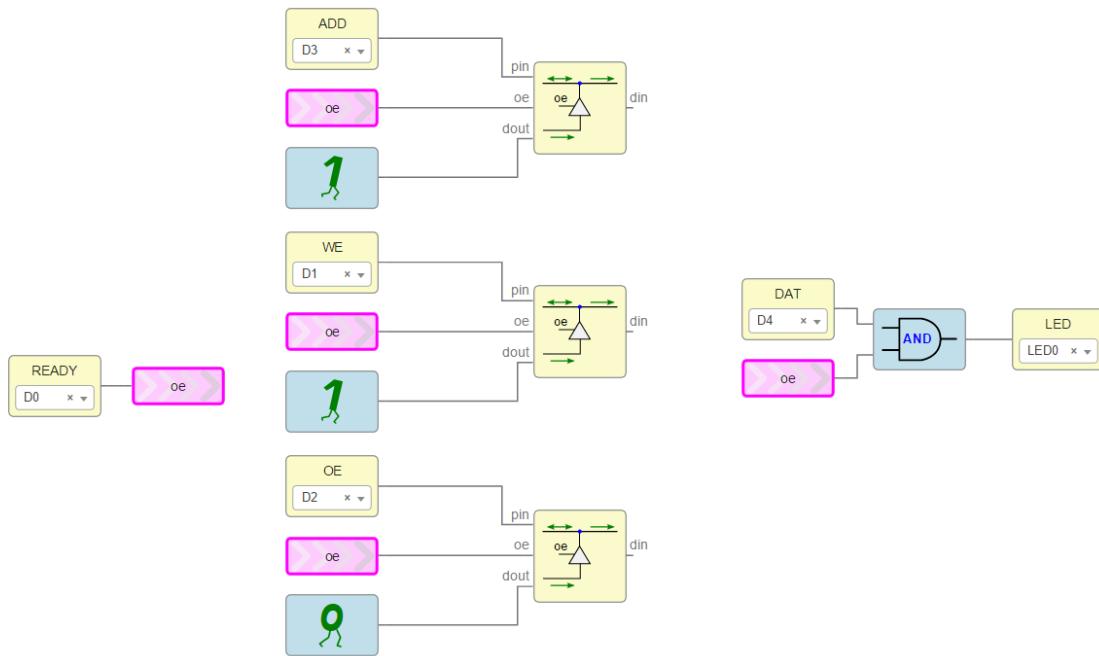


Figura 72: Configuración de la FPGA para lectura de la SRAM en IceStudio.

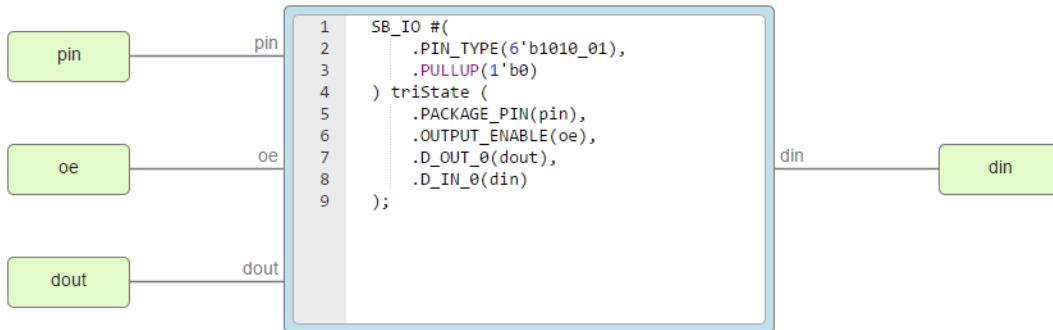


Figura 73: Contenido del bloque InOut.

El bloque *InOut* permite configurar los pines de la FPGA como entrada o salida en función de una señal de control *OE*. Si la señal está a 1, el puerto es configurado como salida, si está a 0, el puerto se configura como entrada. Con este bloque conseguimos por lo tanto una puerta tri-estado que nos permite desconectar físicamente el pin de la FPGA del exterior como se puede ver en la Figura 74. El funcionamiento de la FPGA será, por tanto, el siguiente:

- Si la señal recibida por el microcontrolador (*READY*) es un 0 (aún no ha escrito) las señales de control y de dirección están configuradas como entrada y no influyen en el estado de la memoria SRAM.
- Cuando el estado de la señal *READY* pasa a ser 1, los pines de control y dirección de la FPGA pasan a ser de salida y se procede a leer el dato almacenado en la dirección 1.

- Si ese dato es un 1, se encenderá el *LEDO* de la FPGA.

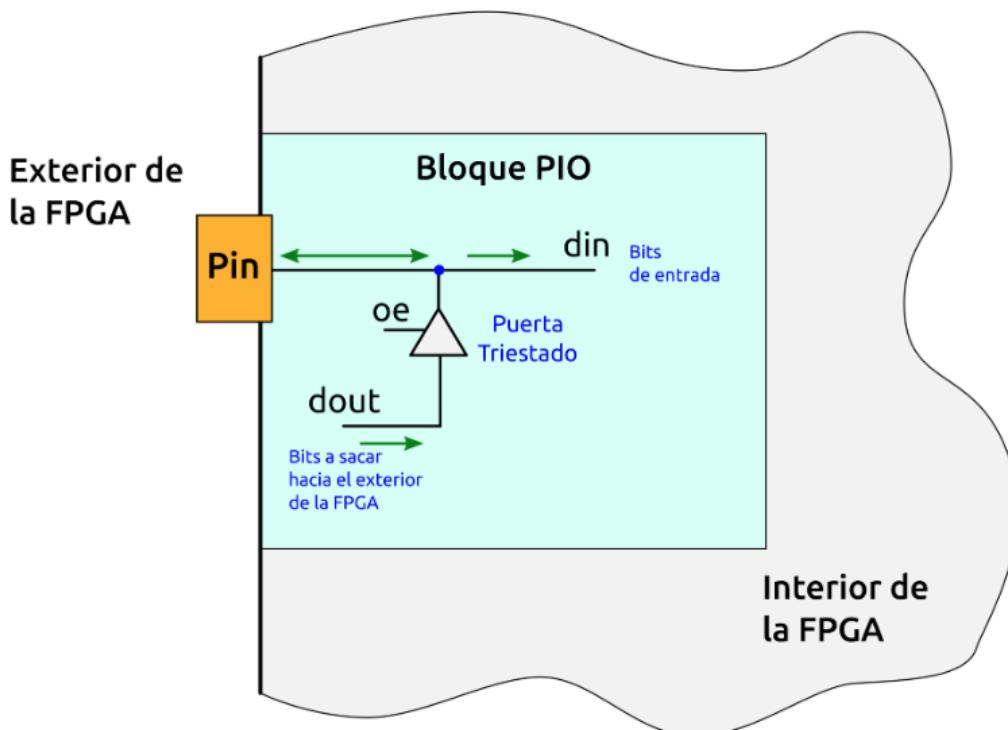


Figura 74: Puerta tri-estado [66]

Para terminar la aplicación sería más correcto que, una vez leído el dato, la FPGA vuelve a configurar sus pines como entrada para así dejar libre el acceso a la SRAM. Sin embargo, para ello habría que recurrir a la implementación de una máquina de estados finito que gestione el proceso y dado que el fin de esta aplicación es únicamente dar una ligera idea de la correcta gestión de la SRAM se ha considerado conveniente terminar aquí el proceso.

Para terminar este apartado, comentar que, aunque la memoria esté conectada a ambos dispositivos, el uso que se le dé a ésta depende únicamente del usuario final, pudiendo éste utilizarla de manera que:

- Sea utilizada por uno sólo de los dispositivos.
- Se reparta según las necesidades, quedando una parte de la misma dedicada a cada dispositivo.
- Un dispositivo sólo escriba y el otro lea, como en la aplicación que hemos desarrollado anteriormente.
- Sea completamente compartida y quede a responsabilidad del usuario evitar el colapso en el acceso y los datos.
- Cualquier otro uso.

4.6. Alimentación global del sistema

En este apartado se realizará un análisis de los requerimientos energéticos del sistema. Por una parte, se analizarán los consumos de cada componente y se decidirá cuál es la mejor forma de alimentar el sistema. Por otra parte, se indicarán los niveles necesarios de tensión de los distintos componentes, así como la forma de obtenerlos.

4.6.1. Análisis de consumo

En la Tabla 14 se muestran los principales componentes que se alimentan a +3.3V, así como sus máximos consumos de corriente. Esto no quiere decir en ningún momento que el sistema vaya a consumir esa cantidad de corriente en operación, sino que el sistema de alimentación debe estar preparado para suministrar esa corriente para funcionar en las peores condiciones posibles. Por ejemplo, para el caso del microcontrolador su consumo dependerá de la temperatura del chip, la frecuencia de operación, el número de periféricos activos, etc.

Dispositivo	Parámetro	Descripción	Corriente (mA)
<i>FT2232H</i>	<i>I_{reg}</i>	Corriente del regulador	150
	<i>I_{ccphy}</i>	Corriente <i>PHY</i> en operación	60
	<i>I_{DD}</i>	Corriente de alimentación	6
	<i>I_{CC} write</i>	Corriente al escribir	2
	<i>I_{VDD}</i>	Corriente máxima en el total de pines <i>VDD</i>	120
	<i>I_{CCIOPEAK}</i>	Corriente de pico en arranque para todos los bancos IO	6.8mA*8=54.5
	<i>I_{SPI PEAK}</i>	Corriente de pico en arranque para el banco SPI	6.8
<i>DSC1001</i>	<i>I_{CC}</i>	Corriente en escritura	25
	<i>I_{DVDD}</i>	Corriente en <i>DVDD</i>	0.01
	<i>I_{AVDD}</i>	Corriente en <i>AVDD</i>	0.08
<i>93LC56C</i>	<i>I_{GND}</i>	Corriente en <i>Vin</i> y <i>EN</i> para una salida de 0.1mA	0.1+0.1=0.2
	<i>I_{CC}</i>	Corriente en operación	30
<i>STM32F091VC</i>	<i>I_{CC}</i>	Corriente en operación	0.75
	TOTAL		455.34mA
<i>iCE40HX4K</i>			
<i>W25132JV</i>			
<i>ADS7924</i>			
<i>NCP114</i>			
<i>IS61WV25616EDBL</i>			
<i>L</i>			
<i>SN74ALVCH162836</i>			

Tabla 14: Consumo máximo en la alimentación de +3.3V

Dada la suma total obtenida, parece razonable suponer que la alimentación a +3.3V debe aportar un mínimo de 500mA para asegurar el correcto funcionamiento de todos los componentes del sistema en las peores condiciones posibles. Además, se debe tener en cuenta el consumo extra de periféricos como el slot para tarjetas microSD, LEDs y dispositivos externos.

Por otro lado, la FPGA debe alimentarse a +1.2V con un consumo 22.3mA por cada V_{cc} , lo que hace un total de 89.2mA. Para los PLL de la FPGA también se necesitan +1.2V con una corriente de 6.4mA. Al tener dos entradas para los PLL suman 12.8mA. En total necesitamos 102mA para la alimentación de +1.2V.

Por último, debemos tener en cuenta que interesa sacar varios pines de alimentación tanto de 5V como de 3.3V al ser dos valores muy utilizados en distintos módulos y sensores.

4.6.2. Conector USB

Para alimentar nuestra placa es necesario aportarle energía desde el exterior. Para ello, los más utilizados son los conectores macho DC y los conectores USB. Puesto que nuestra placa necesita conectarse a un PC para programarse parece adecuado optar por un conector USB también para su alimentación.

Existen distintos tipos de conectores USB figura:

- Tipo A
- Tipo B, mini B y micro B
- Tipo C

Siendo el tipo C (Figura 75) el más reciente de todos. Presenta una serie de ventajas frente a sus predecesores:

- Es pequeño, comparándose en tamaño con uno tipo micro B.
- Soporta una potencia de hasta 100W por lo que es ideal para transmitir cantidades considerables de energía.
- Es reversible debido a la disposición de sus pines.
- Contiene 24 pines permitiendo transmitir señales mediante protocolos HDMI, DisplayPort o Thunderbolt entre otros.

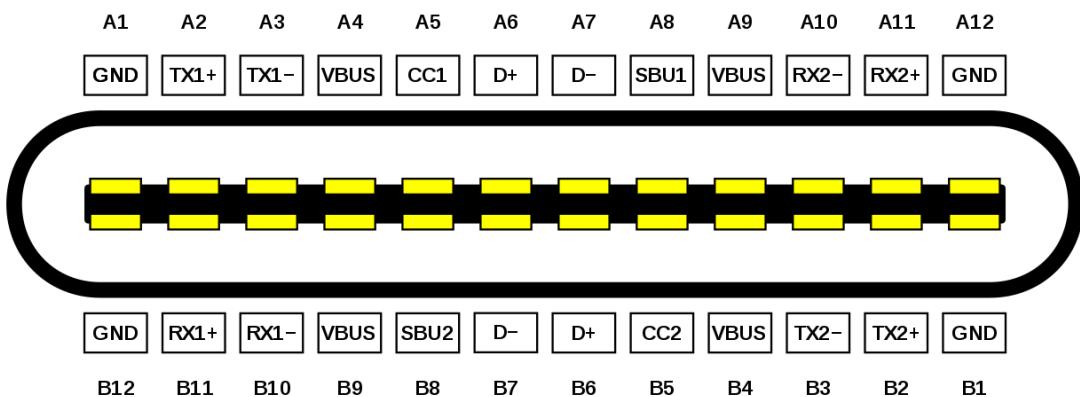


Figura 75: Pinout USB tipo C [67]

Debido a sus características es el nuevo estándar USB y reemplaza cada vez más a sus predecesores en todos los ámbitos. Nuestro diseño contará con un conector tipo C debido a la compatibilidad con USB 2.0 y a que permite transportar más energía. La IceZUM Alhambra II necesita de 2 conectores micro USB para obtener 5V y una intensidad de hasta 4.8A. Sin embargo, nuestra placa admitirá 5V con una corriente de 5A mediante un sólo conector.

Pretendemos hacer el conector equivalente a USB 2.0 para simplificar la compatibilidad con la IceZUM Alhambra II y, por tanto, con IceStudio. Para ello sólo debemos usar los pines de *GND*, *V_{BUS}* y *D+* y *D-*. Utilizaremos el conector de 16 pines DX07S016JA1R1500 [68] cuyo pinout encontramos en la Tabla 15.

<i>Pad</i>	1	2	3	4	5	6	7	8	9	10	11	12
<i>PIN(s)</i>	A1/ B12	A4/ B9	A5	B8	B7	A6	B6	A7	B5	A8	A9/ B4	A12/ B1
<i>Uso</i>	GND	V _{BUS}	NC	NC	D-	D+	D+	D-	NC	NC	V _{BUS}	GND

Tabla 15: Pinout del conector DX07S016JA1R1500

Los pines del 13 al 17 son de protección y van conectados a masa. En la Figura 76 podemos observar el esquemático del conector USB tipo C en KiCad. Del esquemático cabe mencionar los siguientes puntos:

- Se ha incorporado un diodo TVS para proteger al sistema de picos transitorios de voltaje.
- La línea *V_{BUS}* es de +5V y se ha ramificado mediante un filtro LC a una línea +5F para indicar que son 5V ya filtrados. Dicha línea será la que utilizaremos en el siguiente apartado para la alimentación del conversor *DC-DC Buck*.

- Se ha añadido una matriz de diodos de protección ESD [69] para las líneas de datos USB. En la Figura 77 vemos el interior del dispositivo. Este dispositivo se considera necesario debido a, entre otros motivos, la creciente tasa de datos que se transmite a través de una línea USB y a que los humanos podemos generar niveles considerablemente altos de ESD que puede ser descargado a la placa mediante el conector USB.

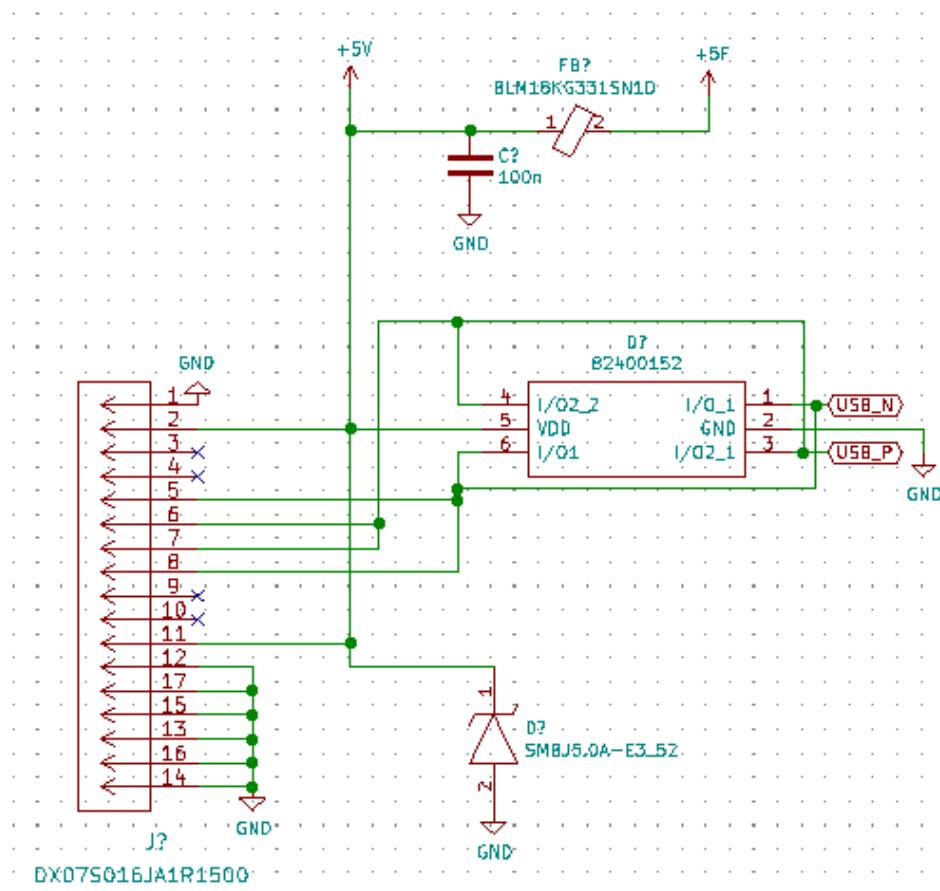


Figura 76: Esquemático del conector USB C en KiCad

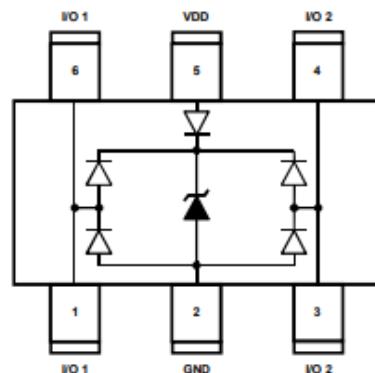


Figura 77: Interior del circuito de protección ESD para USB [69]

4.6.3. Regulador de tensión

Como ya se comentó en el apartado *Análisis de consumo {4.6.1}*, nuestra placa necesita de varios niveles de tensión para funcionar correctamente. Para ello debemos de recurrir a reguladores de tensión. Existen varias soluciones, entre las que destacan:

- LDO: Reguladores de tensión de baja caída como el que utilizamos anteriormente para obtener la tensión estable del conversor ADC en el apartado *ADC {4.2.2}*. Su funcionamiento se basa en disipar en forma de calor la potencia que no necesitamos. Son pequeños, económicos y muy estables. Sin embargo, su uso sólo es recomendable si la diferencia de tensión entrada-salida es pequeña o si no nos importa ni el calor disipado ni la energía desperdiciada.
- Conversores *DC-DC Buck*: Conversores ampliamente utilizados que permiten reducir la tensión de entrada a una tensión de salida deseada. Almacena la energía en componentes inductivos por lo que, aunque su eficiencia no sea del 100%, sí que es más alta que para un LDO. Adecuados si necesitamos mayor potencia y una diferencia tensión de entrada-salida elevada.

Debido a lo expuesto anteriormente optaremos por un conversor *DC-DC Buck* para transformas los 5V de entrada a 3.3V y 1.2V. Para ello utilizaremos el PAM2306 [70], un conversor DC-DC de dos canales, con eficiencia de hasta 96% y corriente de salida de 1A por canal. El pinout del dispositivo lo encontramos en la Figura 78, el diagrama de bloques para un canal en la Figura 79 y una implementación típica en la Figura 80.

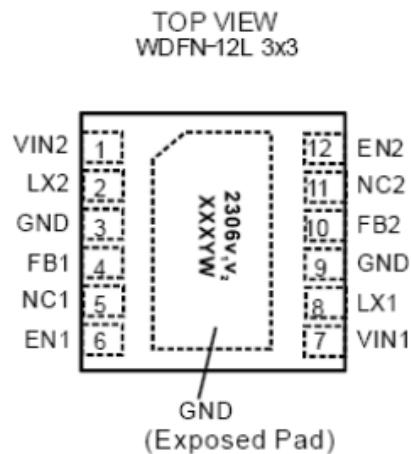


Figura 78: Pinout del PAM2306 [70]

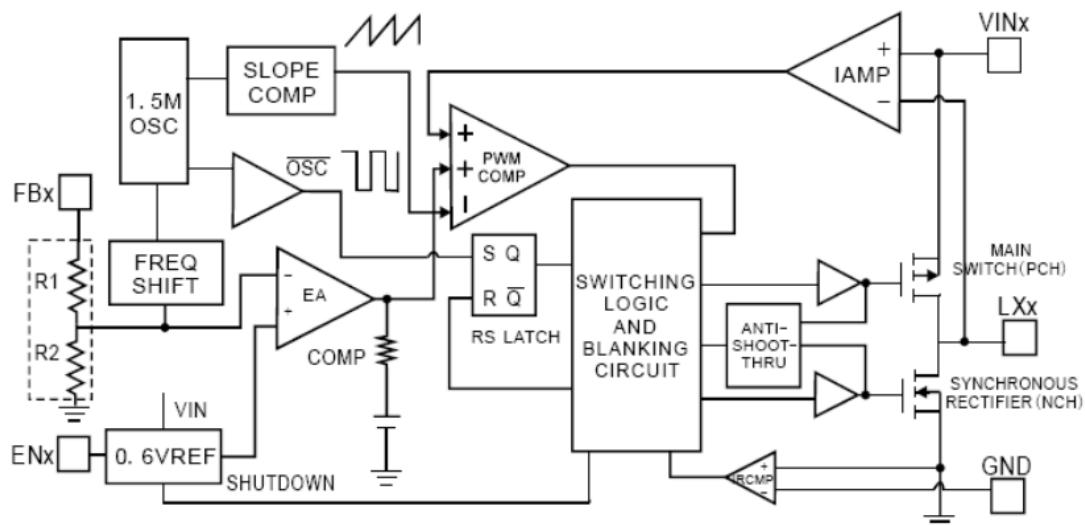


Figura 79: Diagrama de bloques PAM2306 [70]

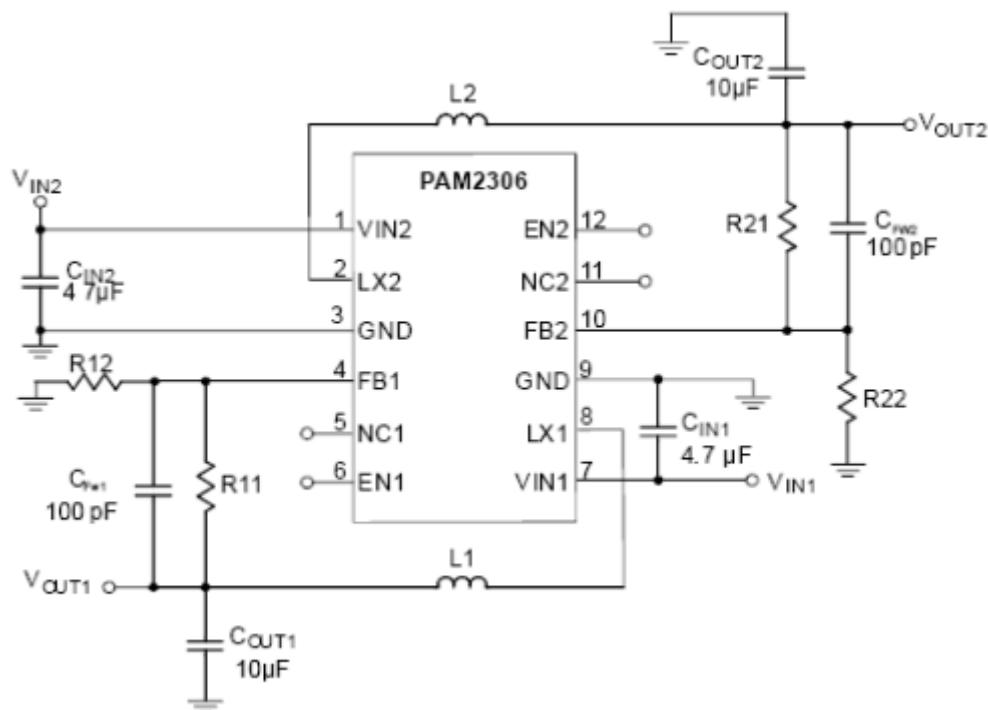


Figura 80: Implementación típica PAM2306 [70]

Basándonos en las figuras anteriores y a la hoja de datos del fabricante procedemos a configurar el convertidor. El valor de las inductancias decidirá el rizado presente en la corriente de salida según la Ecuación 3:

$$\Delta I_L = \frac{1}{f \cdot L} \cdot V_{OUT} \cdot \left(1 - \frac{V_{OUT}}{V_{IN}}\right)$$

Ecuación 3: Cálculo de la bobina para el conversor

Optaremos por una inductancia de valor típico $L=4.7\mu\text{H}$ para ambos canales, así como con una baja resistencia DC para mayor eficiencia.

Para C_{IN} y C_{OUT} optaremos por condensadores de 10nF cerámicos ya que su alta corriente de rizado, su alto voltaje y su baja ESR los hacen ideales reguladores conmutados.

Las resistencias $R1$ y $R2$ nos determinan el voltaje de salida según la Ecuación 4:

$$Vo = 0.6 \cdot \left(1 + \frac{R1}{R2}\right)$$

Ecuación 4: Cálculo de las resistencias para el conversor

Por lo tanto, para el canal que necesitamos 3.3V optaremos por $R1=45\text{k}\Omega$ y $R2=10\text{k}\Omega$. Para 1.2V optaremos por $R1=10\text{k}\Omega$ y $R2=10\text{k}\Omega$.

En la Figura 81 podemos observar el esquemático final del convertidor. Como se indicó antes, la entrada V_{IN} del convertidor es la línea filtrada $+5\text{VF}$ además de desacoplarlo con condensadores de 10nF . Los pines EN se han puesto también a $+5\text{V}$ ya que se activan en valor alto. Por último, comentar que se ha añadido un diodo Zener en la tensión de $+3.3\text{V}$ para fijar la tensión ya que muchos dispositivos se conectarán a esa línea.

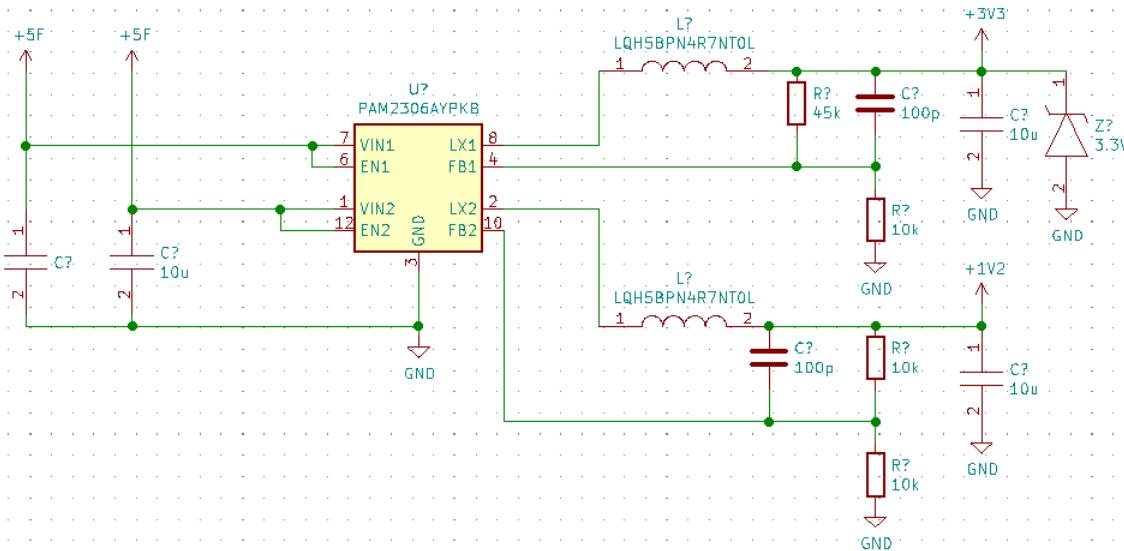


Figura 81: Esquemático del PAM2306 en KiCad

Por último, como se comentaba anteriormente, muchos dispositivos externos se conectarán a la placa, por lo que resulta adecuado tener de unos pines externos de alimentación. Para ello se han añadido 4 pines para la alimentación 5V, otros 4 para +3V3, así como 8 para GND. Sin embargo, no se recomienda utilizar estos pines para alimentar otros dispositivos como norma general ya que, si se sobrecargan, puede provocar un mal funcionamiento del sistema, especialmente en la línea de 3.3V. Por solucionar esto, se ha implementado además un interruptor que permite activar o desactivar niveles de tensión independientes para los periféricos que se conecten a la placa como se muestra en la Figura 82. En el podemos apreciar un interruptor a la izquierda que permitirá al transistor entrar en conducción o corte y así controlar la tensión de +5VP. Dicha tensión esta filtrada y estabilizada mediante un condensador de aluminio de $220\mu F$ dado que debe soportar varios dispositivos conectados a él. Además, en este caso hemos añadido un LDO que rebaje los +5VP a +3.3VP para permitir también conectar dispositivos externos que requieran de ese nivel de alimentación. El LDO es un NCP708MU330TAG [71] que permite proporcionar hasta 1A en la salida y consumiendo en reposo solamente $200\mu A$. En la Figura 83 podemos ver una aplicación típica del dispositivo. A la entrada del LDO hemos colocado un condensador electrolítico de $100nF$ para reducir el rizado en la tensión de entrada y a la salida se ha incluido otro condensador de aluminio de $220\mu F$ por el mismo motivo que se incorporó el anterior. Cabe indicar que si la placa no necesita de ningún periférico alimentándose por estos pines se recomienda desactivar para así reducir el consumo de la placa.

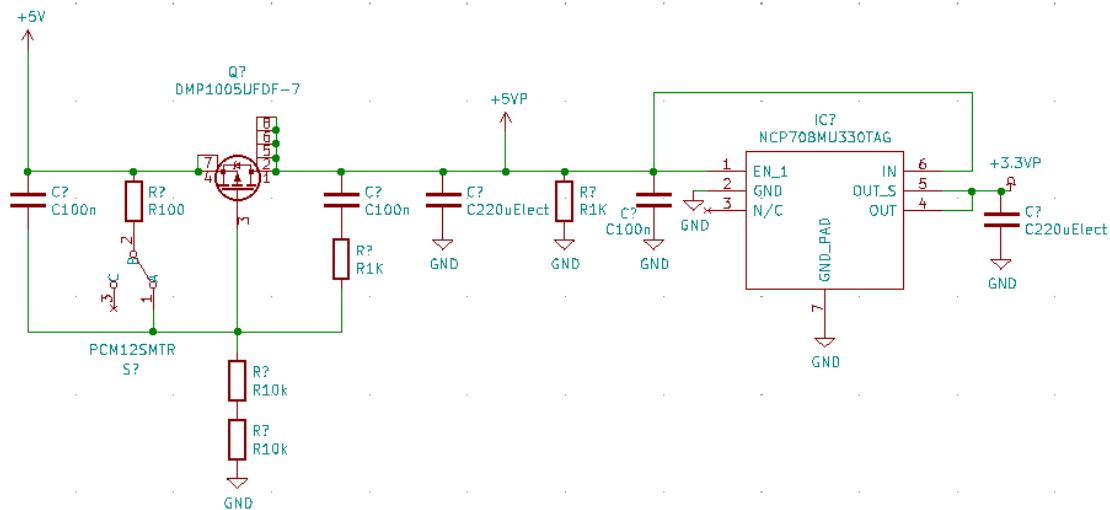


Figura 82: Interruptor de la tensión para periféricos

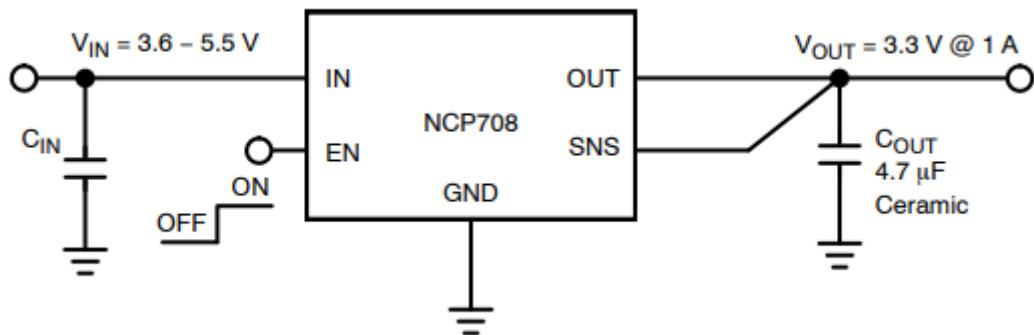


Figura 83: Aplicación típica del LDO [71]

Además, se han añadido dos LEDs para indicar al usuario el estado de la placa. Uno de ellos es para indicar que la placa está alimentada mientras que el otro es para indicar que los pines de alimentación independientes están activados. En la Figura 84 podemos encontrar sus esquemáticos en KiCad.

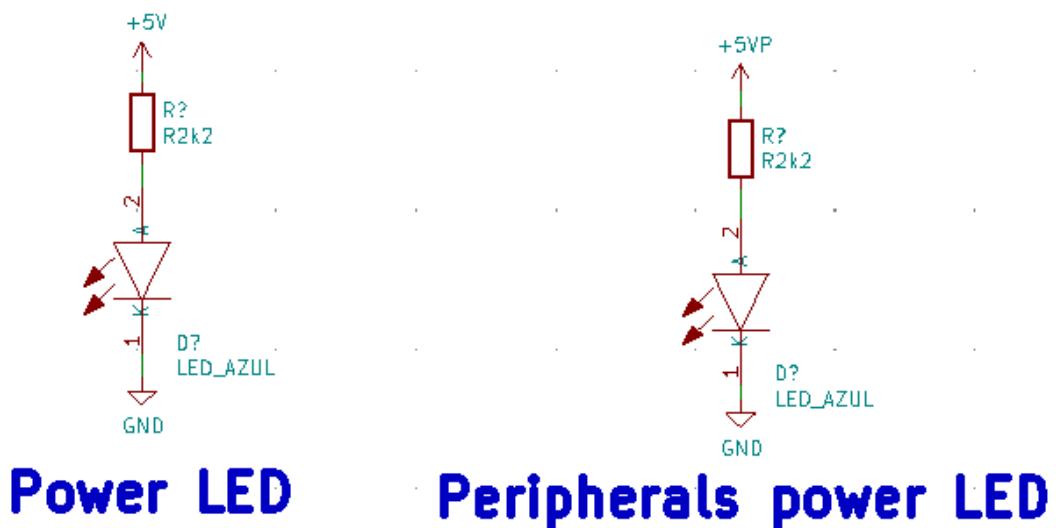


Figura 84: LEDs indicadores de alimentación

4.7. Placa de circuito impreso

En este apartado comentaremos brevemente, una vez que hemos completado el diseño en esquemático de la placa, las consideraciones más relevantes a tener en cuenta para el diseño de la placa de circuito impreso, así como el resultado final.

El producto final contará con 4 capas de cobre con un grosor de 1.57mm, un estándar en la industria. La capa superior e inferior se utilizarán para el trazado de las distintas pistas que componen el circuito, mientras que las capas intermedias se usarán como planos de alimentación. Colocando un plano de masa en la segunda capa conseguimos un camino sencillo de retorno para la alimentación y las señales, una mayor disipación de calor y reducir en gran medida el ruido y las interferencias al reducir la impedancia a masa. Con un plano de alimentación en la tercera capa se consigue una mayor facilidad para repartir las líneas de voltaje por la placa además de evitar los sobrecalentamientos que provocaría la circulación de corriente por pistas más estrechas.

El ancho de las pistas será de 0.2mm para pistas estándares, 0.3mm para aquellas que requieran de un mayor flujo de corriente o sean más sensibles a ruidos e interferencias y de 0.5mm para las de alimentación siempre y cuando sea posible. El tamaño de las vías será de 0.6/0.3mm. Además, se ha evitado usar ángulos de 90° optando por los de 45° para facilitar el flujo de la corriente.

La placa contará con varias zonas bien diferenciadas:

- FPGA
- ADC
- Microcontrolador
- Memoria SRAM
- Alimentación
- FTDI
- Periféricos

Estructurar la placa mediante estos grupos ayudará a la hora de colocar los componentes y facilitará el trabajo de enrutado. Siempre que sea posible, los componentes se colocarán en la misma orientación y teniendo en cuenta la topología de cada uno. Evidentemente no entraremos en detalle en el proceso de colocación de cada uno, pero sí comentaremos algunos de mayor relevancia como pueden ser:

- El ADC de la FPGA: Un conversor de este tipo, como su nombre indica, tiene una parte analógica y otra digital que conviene mantener separadas para evitar interferencias. Podemos ver un ejemplo de enrutado en la Figura 85.

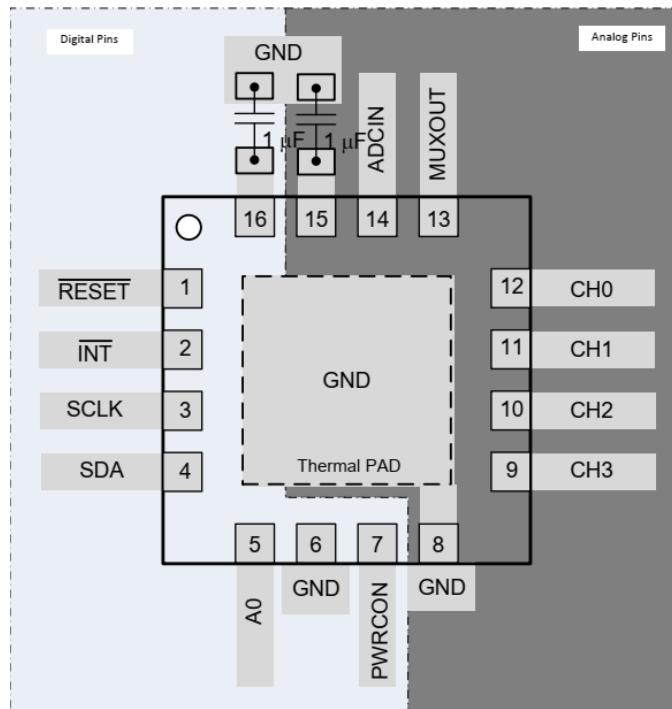


Figura 85: Ejemplo de enrutado del ADC [46]

- Los dispositivos de alimentación, en concreto los conversores y reguladores, suelen disipar mucha energía, por lo que es conveniente utilizar pistas anchas, polígonos y varias vías para evitar sobrecalentamientos.
- El reloj del microcontrolador es un componente extremadamente sensible debido a su alta frecuencia de funcionamiento. Es por ello que es recomendable crear una especie de antena que lo aísle del resto del sistema además de mantenerlo separado de otros componentes y líneas del sistema. Encontramos su enrutado en la Figura 86.
- Las líneas de conexión con la SRAM deben ser lo más directa posibles y cruzarse lo menos posible. En caso de cruzarse es recomendable que no lo hagan con la misma orientación. El resultado puede verse en la Figura 87.
- Las señales de información de entrada del USB son señales diferenciales. Es importante que la longitud de las pistas sea similar, que se eviten, en la medida de lo posible, el uso de vías, que ambas estén posicionadas simétricamente respecto a un eje para evitar desfases y que se elija un ancho de pista que proporcione una impedancia de 90Ω o similar. El resultado puede verse en la Figura 88.

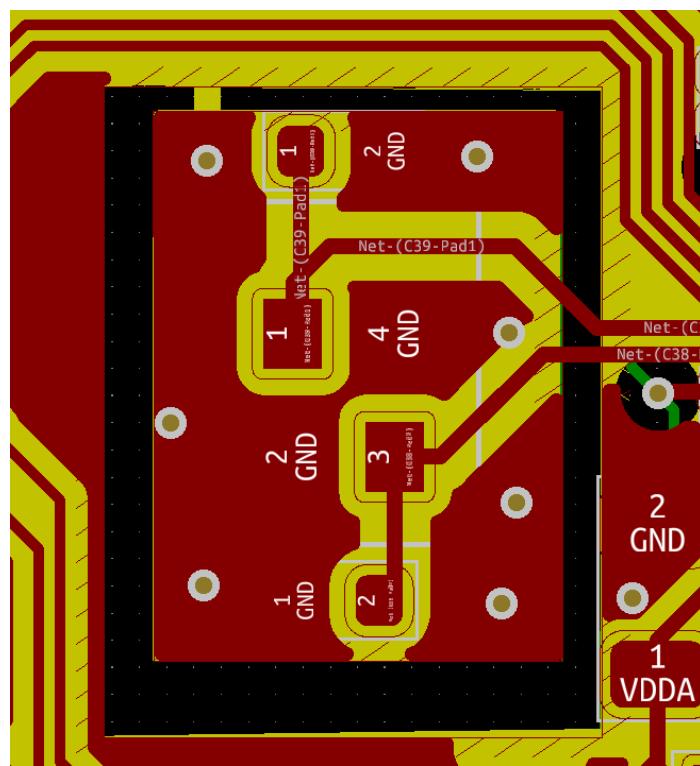


Figura 86: Antena para el oscilador del microcontrolador

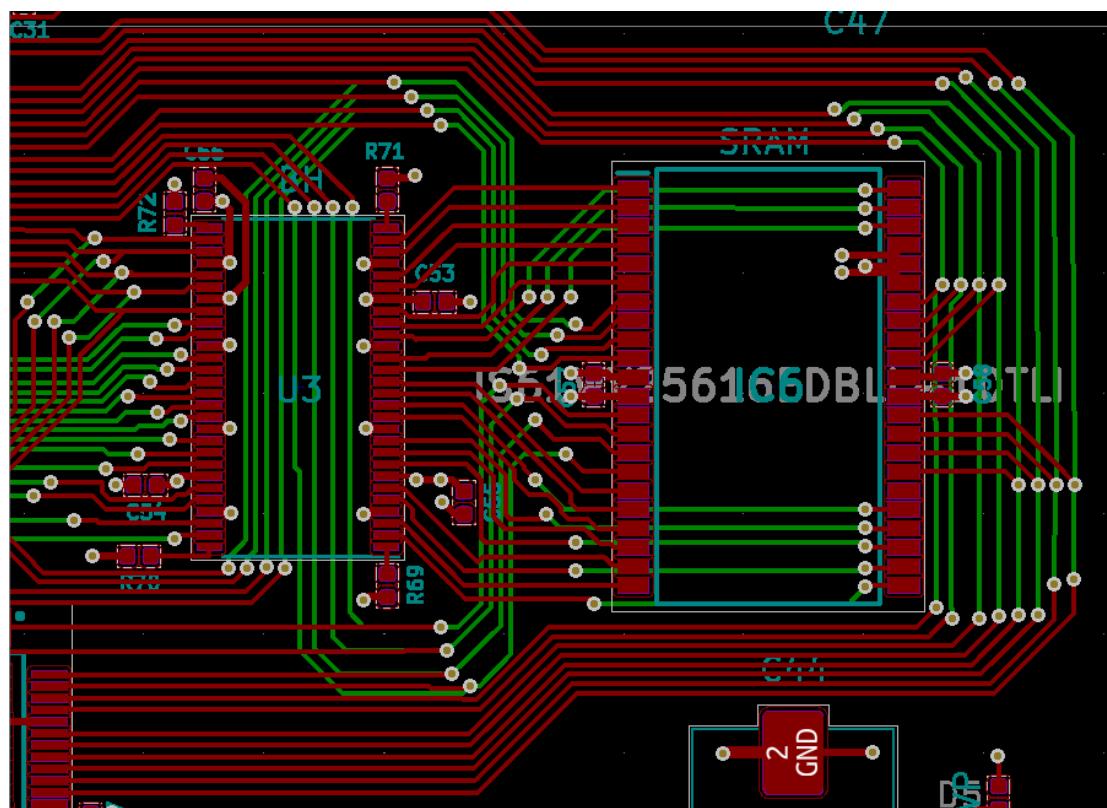


Figura 87: Enrutado de la SRAM

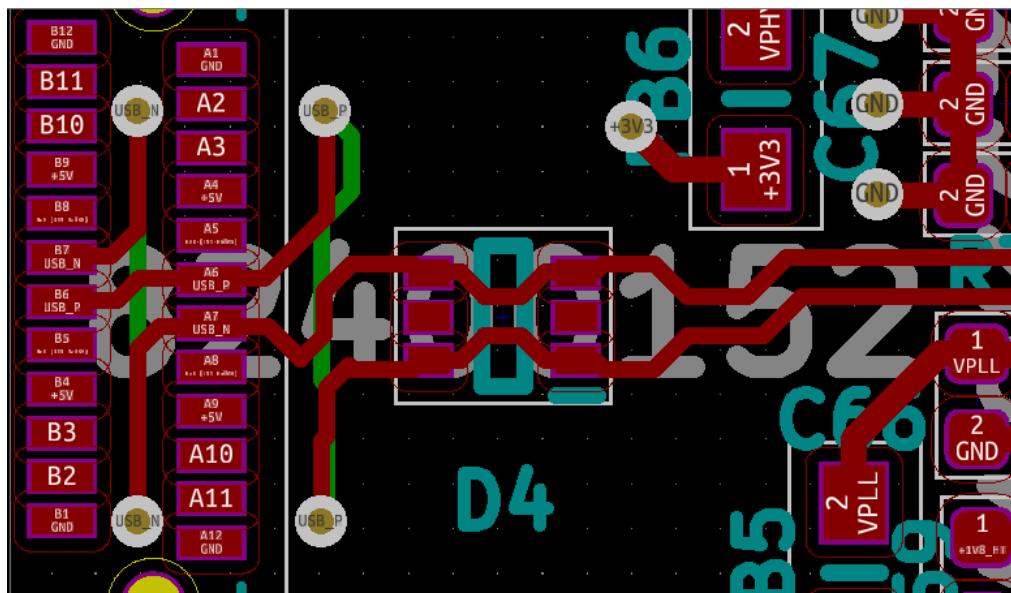


Figura 88: Enrutado del par diferencial USB

A continuación, en la Figura 89 se muestra una representación de las diferentes zonas de la PCB. Para observar la PCB con mayor detalle se recomienda acceder al GitHub del proyecto [1], descargar los archivos y visualizarlo en KiCad o algún visualizador para archivos Gerber. En la Figura 90 se observa el resultado final mostrando solamente las capas superior e inferior y ocultando los planos y polígonos para facilitar la visualización. En la Figura 91 podemos ver el diseño completo mostrando todas las capas.

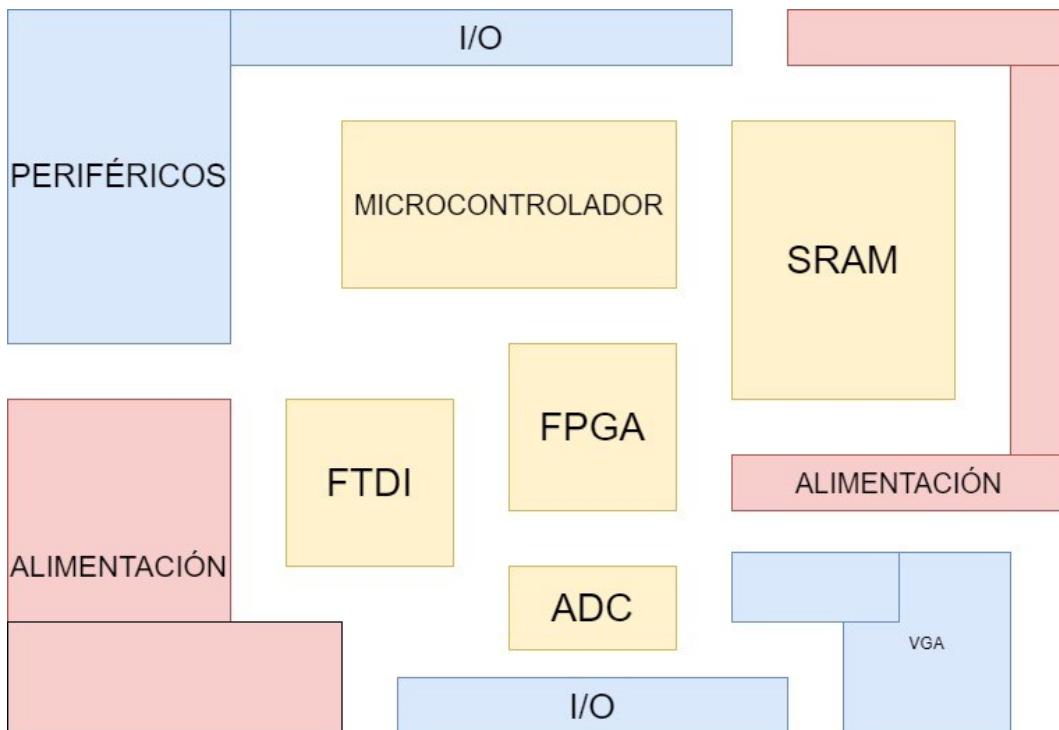


Figura 89: Distintas secciones de la PCB

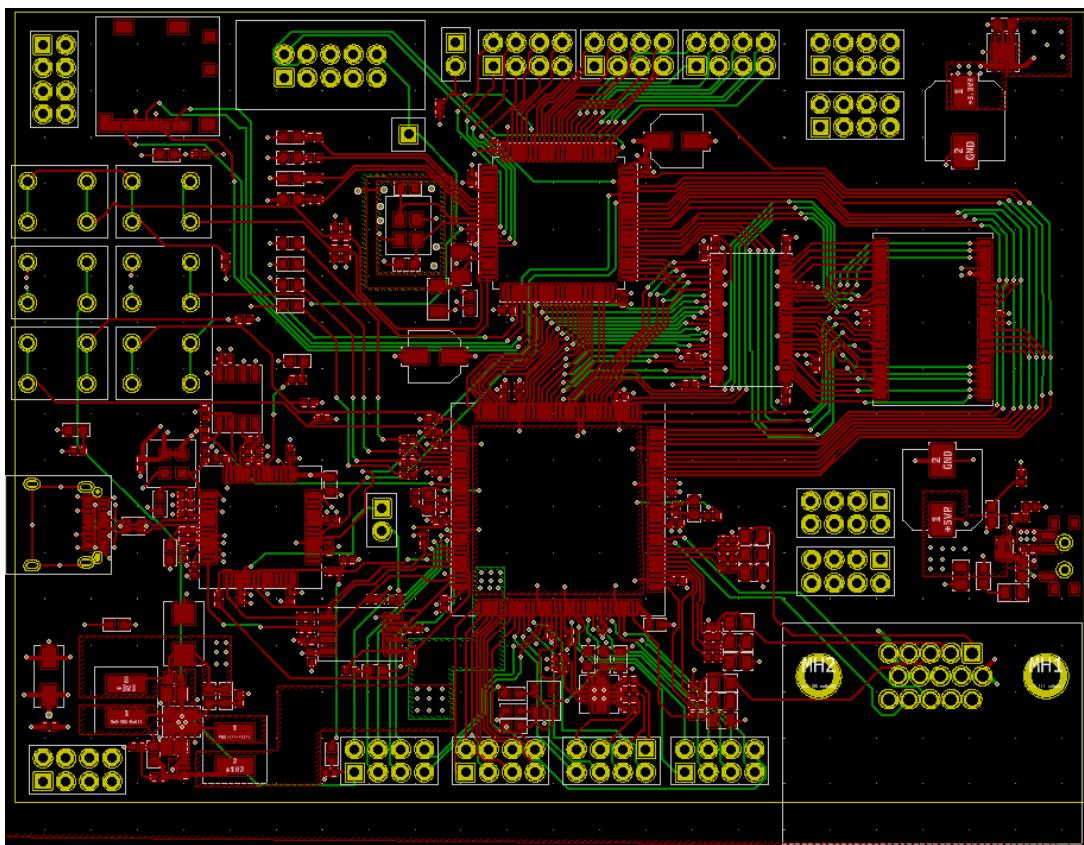


Figura 90: Diseño final simplificado de la PCB

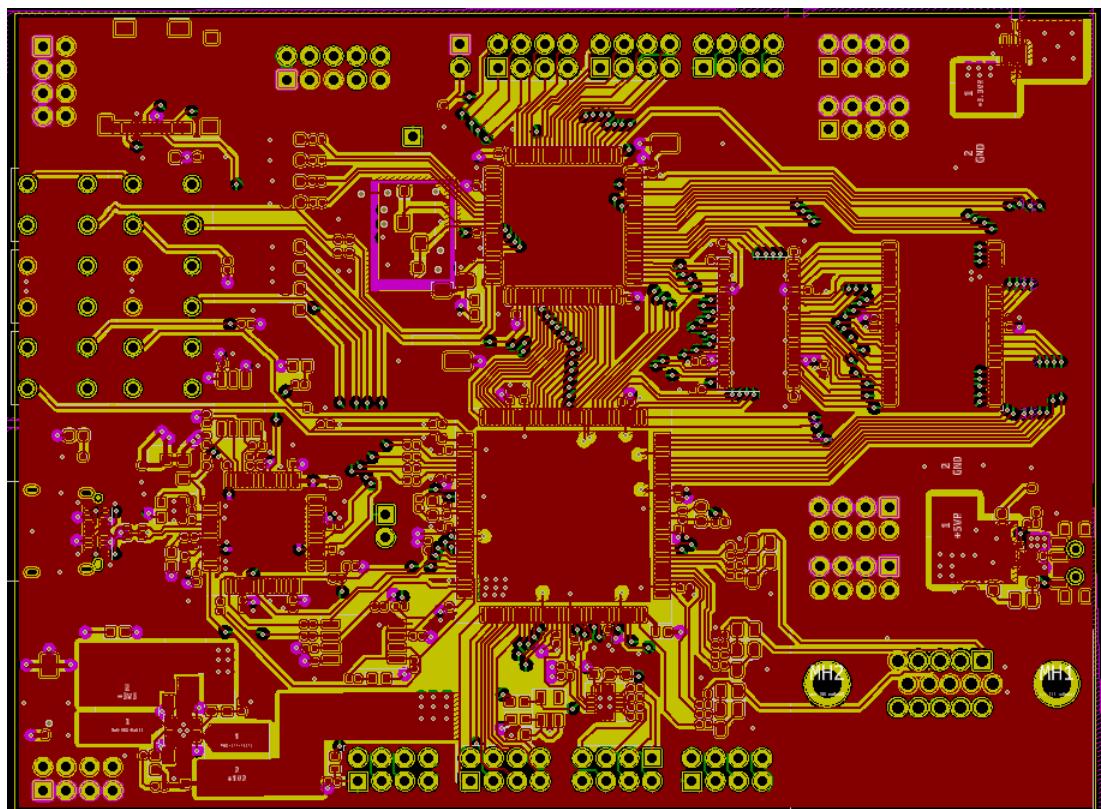


Figura 91: Diseño final completo de la PCB

La dimensión de la PCB es de 117x87 mm frente a los 80x55.1 mm de la IceZUM Alhambra II.

Por último, generamos los archivos Gerber de las capas que queramos (Figura 92) y podemos acudir a la sección Gerber Viewer de KiCad o incluso cualquier otra aplicación para visualizar los archivos generados. En la Figura 93 podemos observar la cara frontal de la PCB mediante un visor de archivos Gerber online [72].

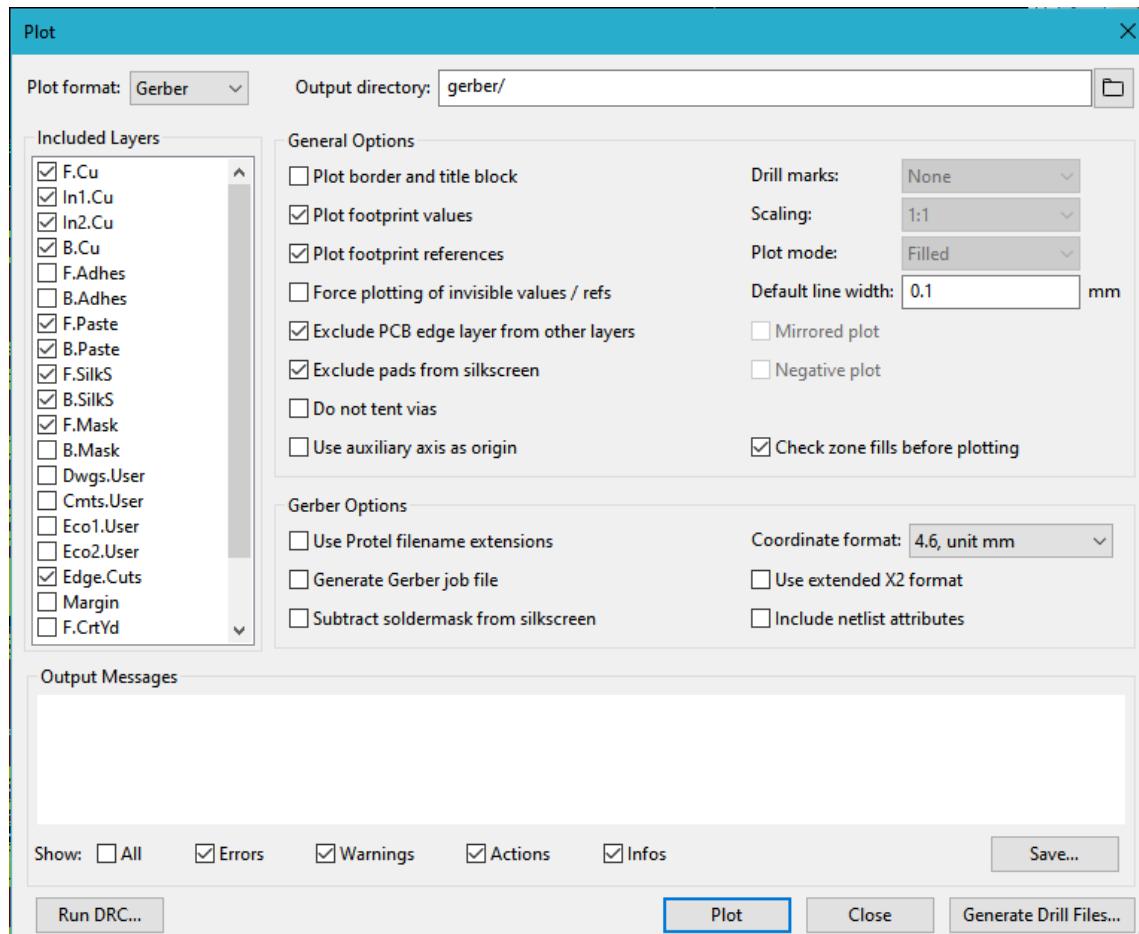


Figura 92: Generación de archivos Gerber

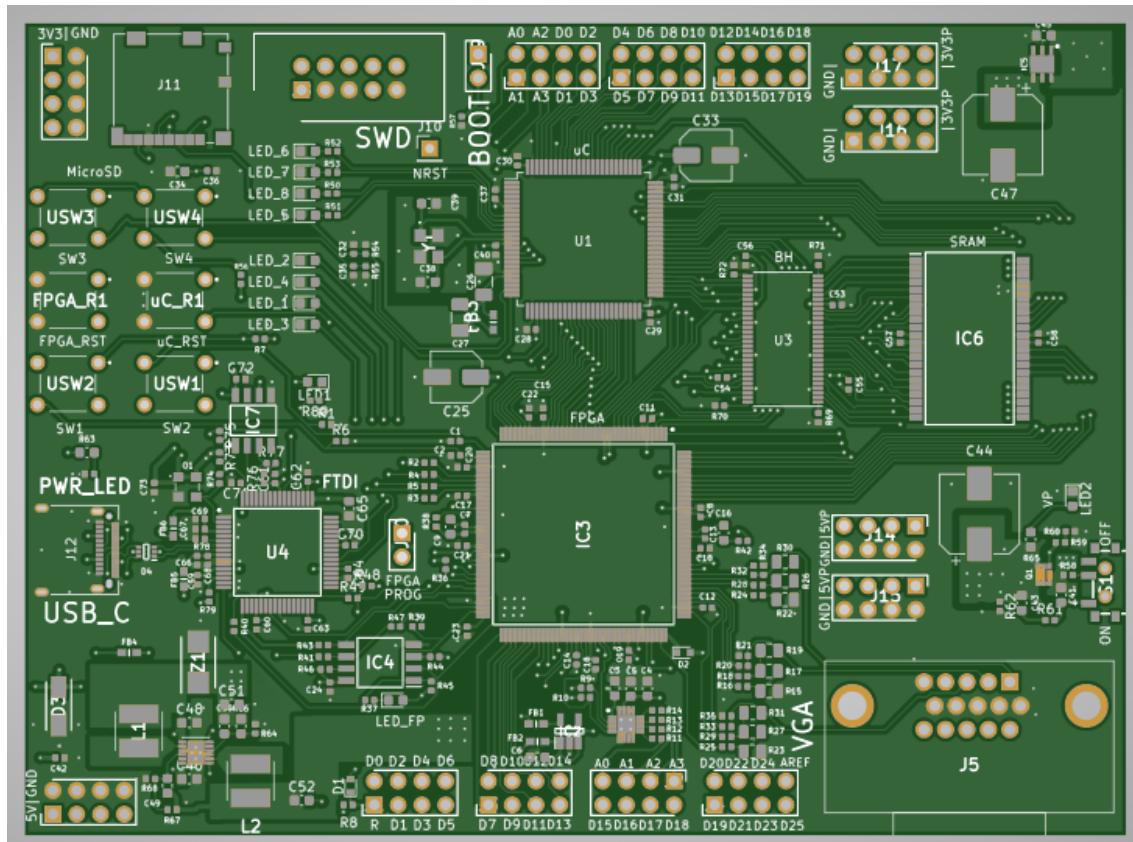


Figura 93: Visualización de los archivos Gerber

4.8. Producto final

Para terminar este capítulo se presentará, a modo de resumen, las especificaciones más relevantes del diseño, pinout del dispositivo, imágenes del producto en 3D, precio y otras consideraciones varias.

Especificaciones para la FPGA:

- Chip Lattice iCE40HX4K con 3520 LUTs y 80Kbits de RAM.
- 4 LEDs y 2 interruptores de propósito general.
- 26 pines digitales y 4 analógicos I/O.
- Botón físico y pin accesible de *reset*.
- Salida de video VGA.
- Memoria flash de 32Mb para *toolchain* y datos.
- Señal de reloj de 12MHz.
- Programable mediante la herramienta libre IceStudio.

Especificaciones para el microcontrolador:

- Microcontrolador STM32F091VCT6.
- Memoria flash de 256Kbyte y 32Kbyte de memoria RAM.

- Reloj interno de 8MHz y reloj externo de 32MHz.
- 4 LEDs y 2 interruptores de propósito general.
- Botón físico y pin accesible de *reset*.
- 20 pines digitales y 4 analógicos I/O.
- Conector SWD para utilizarlo con ST-Link.
- Slot para tarjeta microSD.
- 1 puerto de comunicación SPI, 1 puerto I2C, 5 UART y 1 USART.
- Programable mediante USB o mediante SWD.

Especificaciones globales:

- Alimentación mediante USB tipo C de hasta 5A.
- 12 pines de alimentación a 5V y otros 12 pines a 3.3V.
- 24 pines de GND.
- Interruptor para los pines de alimentación periféricos.
- Memoria SRAM de alta velocidad compartida de 4MB con circuito de retención de buses y circuito de corrección de errores integrado.
- LED de encendido y LED indicador de alimentación de periféricos.

En las Figura 94, Figura 95 y Figura 96 podemos encontrar un modelo 3D de la PCB.

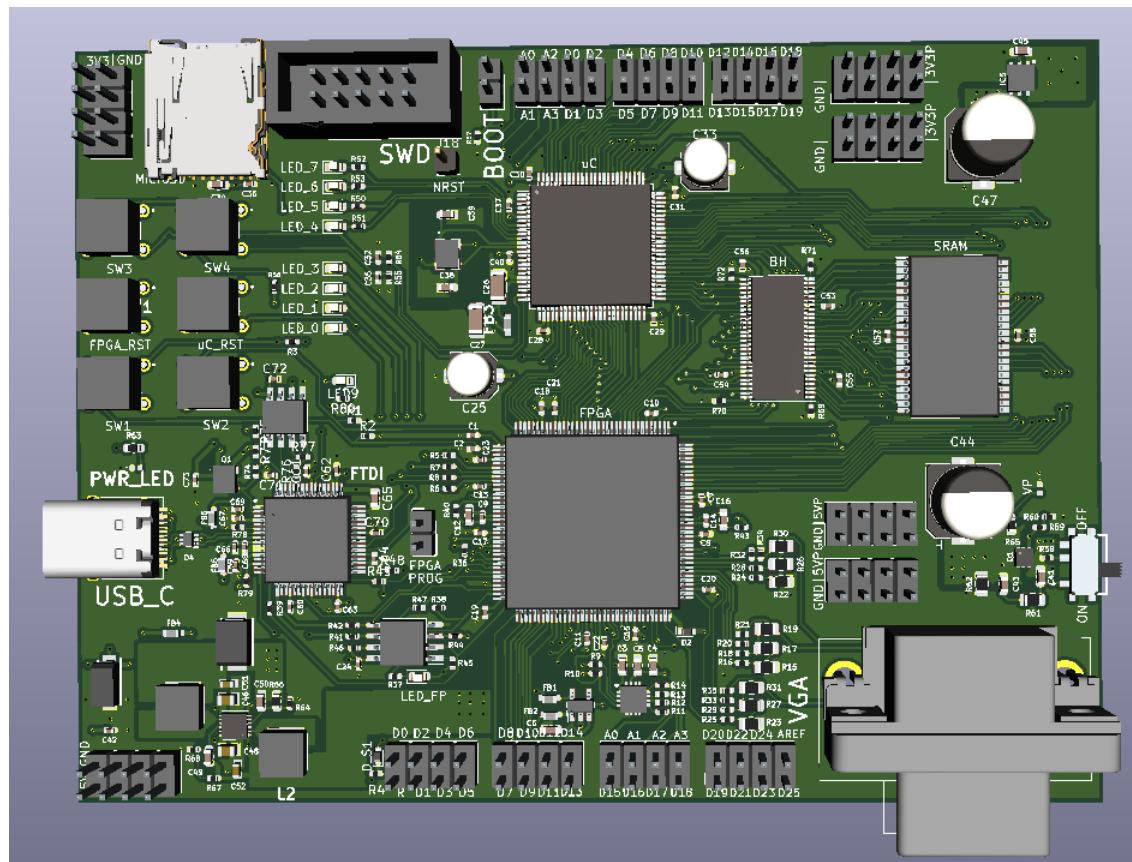


Figura 94: Modelo 3D de la placa (cara superior)

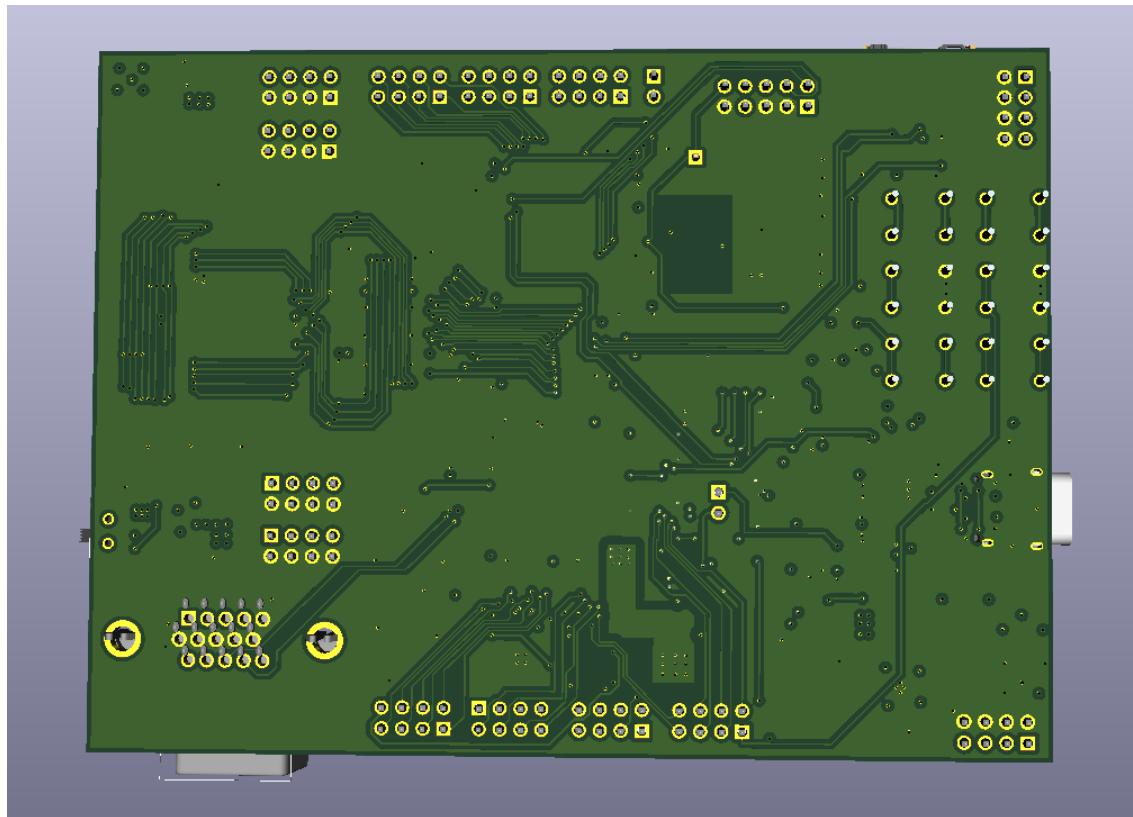


Figura 95: Modelo 3D de la placa (cara inferior)

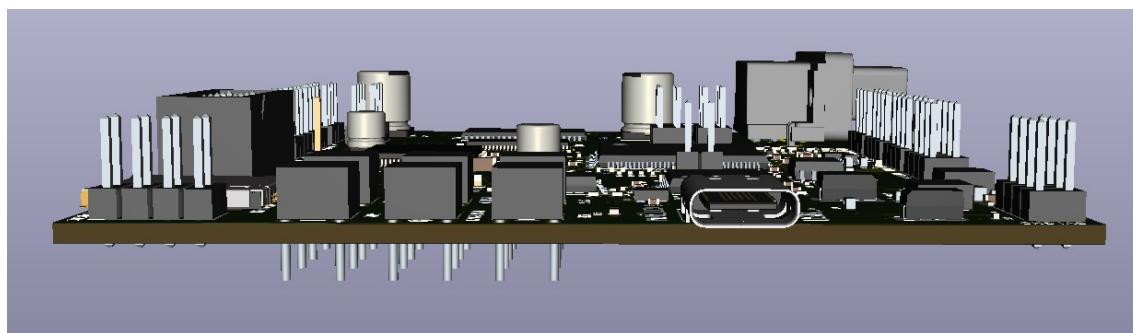


Figura 96: Modelo 3D de la placa (perfil)

En la Tabla 16 se muestra las funciones especiales que pueden desempeñar los pines del microcontrolador:

Pin	UART	SPI	I2C	Otro
A0	UART1_TX			EVENT
A1	UART1_RX			EVENT
A2	UART2_TX	SPI_MISO		EVENT
A3	UART2_RX	SPI_MOSI		EVENT
D0				EVENT

			TIM1_CH1
D1			EVENT TIM2_CH1
D2	SPI_SS	I2C_SDA	EVENT TIM2_CH1
D3		I2C_SCL	TIM1_CH1
D4	UART3_RX	I2C_SDA	TIM2_CH1N
D5	UART3_TX	I2C_SCL	TIM1_CH1N
D6			WAKEUP1 TIM3_CH2
D7	UART4_RX		TIM3_CH1
D8	UART4_TX		TIM2_CH2
D9	USART_CK		
D10	USART_RX		
D11	USART_TX		
D12	USART_RTS	SPI_MOSI	
D13	USART_CTS	SPI_MISO	
D14	UART4_RX		
D15		SPI_SCK	
D16		SPI_SS	
D17	UART4_TX		
D18	UART5_RX		
D19	UART5_TX		

Tabla 16: Funciones especiales en los pines del microcontrolador

Con respecto a la FPGA, sólo el pin *D19* y *D20* tienen una función especial. Siendo estos pines *GBIN3* y *GBIN2*, y, por lo tanto, permitiendo la entrada y salida de señales de reloj, así como de *reset* y *enable*.

En la Tabla 17 se muestra un listado de materiales simplificado, quedando alojando en el GitHub del proyecto uno más detallado en Excel por si fuese necesario consultarlos. Los componentes se han buscado en dos proveedores diferentes, RS Components [23] y Mouser Electronics [55]. El precio mostrado es el unitario multiplicado por unidad a 22 de agosto de 2020.

Referencia	Descripción	Uds.	PRECIO
C1, C2, C7, C8, C10-C15, C17-C24, C28-C32, C35-C37, C40, C42, C53-C64, C66-C73	100nF	48	3.456
C5, C9, C16, C34, C46, C48, C51, C52, C65	10uF	9	2.088
C3, C4, C6, C41, C43, C45	100nF	6	0.306
C25, C33	4.7uF Electrolítico	2	0.528
C26	1uF	1	0.263
C27	10nF	1	0.082
C38, C39	20pF	2	0.229
C44, C47	220uF Electrolítico	2	0.638

<i>C49, C50</i>	100pF	2	0.148
<i>D1, D2</i>	D_Schottky	2	0.078
<i>D3</i>	824500500	1	0.225
<i>D4</i>	82400152	1	0.48
<i>FB1-FB6</i>	BLM18KG331SN1D	6	0.228
<i>IC1</i>	ADS7924IRTER	1	1.60
<i>IC2</i>	NCP114BSN330T1G	1	0.2
<i>IC3</i>	iCE40HX4K-TQ144	1	5.34
<i>IC4</i>	W25Q32JVSSIQ	1	0.584
<i>IC5</i>	NCP708MU330TAG	1	0.39
<i>IC6</i>	IS61WV25616EDBLL-10TLI	1	2.33
<i>IC7</i>	93LC56C-I_SN	1	0.203
<i>J1-J4, J6, J7, J9, J13-20</i>	Pin Header 4x2	15	4.2
<i>J5</i>	Conector VGA	1	1.38
<i>J8</i>	Conector SWD	1	1.26
<i>J10</i>	Pin Header 1x1	1	0.085
<i>J11</i>	Conector Micro SD	1	2.92
<i>J12</i>	Conector USB C	1	1.60
<i>L1, L2</i>	LQH5BPN4R7NT0L	2	0.42
<i>LED1, LED_1, LED_8, LED_FP1</i>	LED_AMARILLO	4	0.4
<i>LED2, LED_2, LED_7, PWR_LED1</i>	LED_AZUL	2	0.474
<i>LED_3, LED_6</i>	LED_VERDE	2	0.332
<i>LED_4, LED_5</i>	LED_ROJO	2	0.292
<i>O1</i>	DSC1001CL2	1	0.805
<i>Q1</i>	DMP1005UFDF-7	1	0.362
<i>R1, R6, R8-R10, R36, R44-R48, R54, R55, R57, R59, R60, R66-R73, R75, R76, R78 R2-R5, R39-R41, R43, R50-R53</i>	R10k	27	1.215
<i>R7, R49, R56, R61, R62, R79 R11-R14, R21, R34, R35, R38, R42, R58, R74, R80 R15, R17, R19, R22, R23, R26, R27, R30, R31 R16, R18, R20, R24, R25, R28, R29, R32, R33 R37, R63, R65, R77 R64</i>	R470	12	0.4548
<i>S1</i>	R1K	4	0.176
<i>U1</i>	R100	14	1.68
<i>U2</i>	R309	9	0.9
<i>U3</i>	R619	9	0.216
<i>U4</i>	R2k2	4	0.04
	45k	1	0.79
<i>S1</i>	PCM12SMTR	1	0.80
<i>U1</i>	STM32F091VCTx	1	4.26
<i>U2</i>	PAM2306AYPKB	1	0.344
<i>U3</i>	SN74ALVCH162827	1	1.56
<i>U4</i>	FT2232H	1	4.19

<i>FPGA_R1, USW1-USW4,</i>	FSM2JLH	6	0.60
<i>uC_R1</i>			
Y1	ECS-320-12-33B-7KM-TR	1	0.33
Z1	Zener 3.3V	1	0.12
TOTAL		217	51.60€

Tabla 17: Lista de materiales del proyecto simplificada

Para la fabricación de la PCB optamos por JLCPCB [73], una empresa internacional de prototipado y fabricación de PCB especializada en producción de pequeños lotes. Sin la colocación de componentes, el precio de fabricación de 5 unidades (mínimo permitido) es de 37.36€. En la Tabla 18 podemos ver el desglose de precio:

<i>Descripción</i>	<i>Precio</i>
<i>Tarifa de ingeniería</i>	27.10€
<i>Precio de la placa</i>	4.49€
<i>Envío a España</i>	5.77€
Total	37.37€

Tabla 18: Coste de fabricación de la PC

5. Conclusiones y trabajo futuro

En este capítulo, como término de este documento, haremos un repaso de los objetivos logrados y las posibles mejoras a implementar en el diseño. Durante el proceso de diseño de una placa de entrenamiento basada en una FPGA de software libre hemos conseguido:

- **Analizar las necesidades del sistema inicial.** En nuestro caso, como detallamos en el apartado Idea inicial de diseño {2.1}, los principales inconvenientes con los que lidiaba el sistema anterior era el número de pines de propósito general accesibles, la escasez de memoria del sistema y el desempeño del microcontrolador elegido. Todas ellas, y algunas más, han sido resueltas por nuestro diseño.
- **Estudio de la arquitectura a implementar y elección de los componentes principales.** Se ha sido capaz de, entre todas las opciones posibles, elegir aquella que más se adecue a las necesidades encontradas, tanto a nivel de arquitectura como de componente.
- **Estudio de los componentes y diseño en esquemático de la placa de entrenamiento.** Gracias a los diseños de la IceZUM Alhambra II alojados en GitHub (además de otros menos relevantes) se ha conseguido integrar en una sola placa componentes de diversas categorías, comprender el funcionamiento de cada uno de ellos y configurarlos de forma adecuada para su correcto funcionamiento.
- **Diseño de la PCB.** Ha sido necesario introducirse al diseño de placas de circuito impreso, así como familiarizarse con el software open source de KiCad. El proyecto, al contar con líneas analógicas y digitales, osciladores y líneas alimentación y de alta velocidad, además de la responsabilidad de integrarlo todo en el menos espacio posible, ha resultado un reto considerable teniendo en cuenta la nula experiencia previa del estudiante en el diseño de placas de circuito impreso.
- **Testeo de diversos componentes.** Aunque la situación del curso académico 2019/2020 debido a la crisis sanitaria producida por el COVID-19 no ha sido favorecedora para el trabajo en laboratorio, se ha intento testear algunos de los componentes del sistema mediante algunas placas de evaluación como la de memoria SRAM y la del lector de tarjetas microSD.

Con todo lo expuesto en los puntos anteriores, se puede concluir que se han cumplido los objetivos de este trabajo de fin de grado. Sin embargo, quedan algunas características que se podrían mejorar, así como trabajo pendiente por hacer:

- **Hacer modular el diseño del sistema.** Aunque la placa cuente con características básicas y versátiles, es cierto que cuenta con algunos añadidos que no todo el mundo puede necesitar para su proyecto y que acaban incrementando el precio. Por ello, se propone una versión muy simplificada del sistema que sólo cuente

con los componentes esenciales y que permita expandir sus funcionalidades mediante el uso de módulos (botones, VGA, lector de microSD, entre otros).

- **Testear el diseño actual e implementar su integración con IceStudio.** Como se comentaba anteriormente en Metodología y planificación {1.3}, el estado de alarma nacional declarado durante el curso 2019/2020 debido a la crisis sanitaria del COVID-19 han limitado el desarrollo del TFG en el laboratorio, imposibilitando pedir material y el acceso a los laboratorios de la universidad. Es por ello que se considera de vital importancia de un testeo de los diversos bloques que componen la placa de entrenamiento, así como de desarrollar el firmware necesario para su compatibilidad con IceStudio.
- **Fabricación de la placa e implementación de una aplicación.** Por último, al igual que lo comentado en el punto anterior, la fabricación de la placa de circuito impreso ha sido inviable debido al estado de alarma. Por lo tanto, y una vez cumplido con el punto anterior, se deja como trabajo pendiente mandar a fabricar la PCB, montar y soldar todos sus componentes e implementar alguna aplicación que permita dar una idea representativa de la capacidad del sistema.

Bibliografía

- [1] A. S. Doblado, «GitHub TFG_2020_ASD,» [En línea]. Available: https://github.com/asdoblado96/TFG_2020_ASD. [Último acceso: 20 Agosto 2020].
- [2] J. O. Cerezo, «GitHub 2017-tfg-juan-ordonez,» [En línea]. Available: <https://github.com/RoboticsLabURJC/2017-tfg-juan-ordonez>. [Último acceso: 20 Agosto 2020].
- [3] Alhambra Bits, «GitHub Alhambra-II-FPGA,» [En línea]. Available: <https://github.com/FPGAwars/Alhambra-II-FPGA>. [Último acceso: 20 Agosto 2020].
- [4] Arduino, «Arduino,» [En línea]. Available: <https://www.arduino.cc/>. [Último acceso: 20 Agosto 2020].
- [5] Open Source Initiative, «Open Source,» [En línea]. Available: <https://opensource.org/>. [Último acceso: 20 Agosto 2020].
- [6] Wikipedia, «Wikipedia FPGA,» [En línea]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array. [Último acceso: 20 Agosto 2020].
- [7] Xilinx, «Xilinx,» [En línea]. Available: <https://www.xilinx.com/>. [Último acceso: 20 Agosto 2020].
- [8] Intel, «Intel,» [En línea]. Available: <https://www.intel.com/content/www/us/en/products/programmable.html>. [Último acceso: 20 Agosto 2020].
- [9] Wikipedia, «Wikipedia Verilog,» [En línea]. Available: <https://en.wikipedia.org/wiki/Verilog>. [Último acceso: 20 Agosto 2020].
- [10] Wikipedia, «Wikipedia VHDL,» [En línea]. Available: <https://en.wikipedia.org/wiki/VHDL>. [Último acceso: 20 Agosto 2020].
- [11] Lattice, «Lattice iCE40,» [En línea]. Available: <http://www.latticesemi.com/iCE40>. [Último acceso: 20 Agosto 2020].
- [12] Clifford, «Project Icestorm,» [En línea]. Available: <http://www.clifford.at/icestorm/>. [Último acceso: 20 Agosto 2020].
- [13] Alhambra Bits, «Alhambra Bits,» [En línea]. Available: <https://alhambrabits.com/>. [Último acceso: 20 Agosto 2020].

- [14] FPGAwars, «GitHub FPGAwars,» [En línea]. Available: <http://fpgawars.github.io/>. [Último acceso: 20 Agosto 2020].
- [15] Wikipedia, «Wikipedia - Microcontroller,» [En línea]. Available: <https://en.wikipedia.org/wiki/Microcontroller>. [Último acceso: 20 Agosto 2020].
- [16] Descubre Arduino, «Descubre Arduino,» [En línea]. Available: <https://descubreaduino.com/microcontrolador/>. [Último acceso: 20 Agosto 2020].
- [17] Wikipedia, «Wikipedia - von Neumann architecture,» [En línea]. Available: https://en.wikipedia.org/wiki/Von_Neumann_architecture. [Último acceso: 20 Agosto 2020].
- [18] Wikipedia, «Wikipedia - Harcard architecture,» [En línea]. Available: https://en.wikipedia.org/wiki/Harvard_architecture. [Último acceso: 20 Agosto 2020].
- [19] Seeedstudio, «Seedstudio,» [En línea]. Available: <https://www.seeedstudio.com/blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses>. [Último acceso: 20 Agosto 2020].
- [20] Ourpcb, «Ourpcb,» [En línea]. Available: <https://www.ourpcb.com/fpga-vs-microcontroller.html>. [Último acceso: 20 Agosto 2020].
- [21] Lattice Semiconductor, «Datasheet iCE40 LP/HX,» [En línea]. Available: http://www.latticesemi.com/view_document?document_id=49312. [Último acceso: 20 Agosto 2020].
- [22] Lattice Semiconductor, «Datasheet Using Differentiall I/O,» [En línea]. Available: http://www.latticesemi.com/view_document?document_id=47960. [Último acceso: 20 Agosto 2020].
- [23] RS Components, «RS Components,» [En línea]. Available: <https://es.rs-online.com/web/>. [Último acceso: 20 Agosto 2020].
- [24] IceStudio, «IceStudio,» [En línea]. Available: <https://icestudio.io/>. [Último acceso: 20 Agosto 2020].
- [25] FPGAwars, «GitHub - Apio,» [En línea]. Available: <https://github.com/FPGAwars/apio>. [Último acceso: 20 Agosto 2020].
- [26] PlatformIO, «PlatformIO,» [En línea]. Available: <https://platformio.org/>. [Último acceso: 20 Agosto 2020].

- [27] NandLand, «NandLand,» [En línea]. Available: <https://www.nandland.com/goboard/introduction.html>. [Último acceso: 20 Agosto 2020].
- [28] MyStorm, «GitHub - Mystorm-org,» [En línea]. Available: <https://github.com/mystorm-org/BlackIce-II>. [Último acceso: 20 Agosto 2020].
- [29] TinyFPGA, «TinyFPGA,» [En línea]. Available: <https://tinyfpga.com/>. [Último acceso: 20 Agosto 2020].
- [30] ARM, «ARM,» [En línea]. Available: <https://www.arm.com/products/silicon-ip-cpu>. [Último acceso: 20 Agosto 2020].
- [31] STMicroelectronics, «STM32,» [En línea]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>. [Último acceso: 20 Agosto 2020].
- [32] STMicroelectronics, «ST,» [En línea]. Available: https://www.st.com/content/st_com/en.html. [Último acceso: 20 Agosto 2020].
- [33] STMicroelectronics, «Datasheet STM32F091VC,» [En línea]. Available: <https://www.st.com/resource/en/datasheet/stm32f091vc.pdf>. [Último acceso: 20 Agosto 2020].
- [34] STMicroelectronics, «Datasheet STM32F4DISCOVERY,» [En línea]. Available: https://www.st.com/resource/en/data_brief/stm32f4discovery.pdf. [Último acceso: 20 Agosto 2020].
- [35] STMicroelectronics, «ST-Link-V2,» [En línea]. Available: <https://www.st.com/en/development-tools/st-link-v2.html>. [Último acceso: 20 Agosto 2020].
- [36] STMicroelectronics, «STM32CubeIDE,» [En línea]. Available: <https://www.st.com/en/development-tools/stm32cubeide.html>. [Último acceso: 20 Agosto 2020].
- [37] Arduino, «ArduinoIDE,» [En línea]. Available: <https://www.arduino.cc/en/main/software>. [Último acceso: 20 Agosto 2020].
- [38] STM32Duino, «GitHub - STM32duino,» [En línea]. Available: <https://github.com/stm32duino>. [Último acceso: 20 Agosto 2020].
- [39] Wikipedia, «Wikipedia - PCB,» [En línea]. Available: https://en.wikipedia.org/wiki/Printed_circuit_board. [Último acceso: 20 Agosto 2020].

- [40] Dpto. Electrónica – CIFP Tartanga LHII, «Tartanga,» [En línea]. Available: <http://bga.blog.tartanga.eus/descripcion-del-proyecto/>. [Último acceso: 20 Agosto 2020].
- [41] KiCad, «KiCad,» [En línea]. Available: <https://kicad-pcb.org/>. [Último acceso: 20 Agosto 2020].
- [42] Wikipedia, «Wikipedia - Archivos Gerber,» [En línea]. Available: [https://es.wikipedia.org/wiki/Gerber_\(formato_de_archivo\)](https://es.wikipedia.org/wiki/Gerber_(formato_de_archivo)). [Último acceso: 20 Agosto 2020].
- [43] SamacSys, «SamacSys,» [En línea]. Available: <https://www.samacsy.com/library-loader/>. [Último acceso: 20 Agosto 2020].
- [44] KiCad, «KiCad Getting Started docs,» [En línea]. Available: https://docs.kicad-pcb.org/5.1/en/getting_started_in_kicad/getting_started_in_kicad.html. [Último acceso: 20 Agosto 2020].
- [45] Lattice Semiconductor, «iCE30 Hardware Checklist,» [En línea]. Available: http://www.latticesemi.com/view_document?document_id=47779. [Último acceso: 20 Agosto 2020].
- [46] Texas Instrument, «Datasheet ADS7924,» [En línea]. Available: <https://www.ti.com/lit/gpn/ads7924>. [Último acceso: 20 Agosto 2020].
- [47] Wikipedia, «Wikipedia - VGA,» [En línea]. Available: https://en.wikipedia.org/wiki/Video_Graphics_Array. [Último acceso: 20 Agosto 2020].
- [48] Analog Devices, «LTSpice,» [En línea]. Available: <https://www.analog.com/en/design-center/design-tools-and-calculators/lts spice-simulator.html>. [Último acceso: 20 Agosto 2020].
- [49] C. A. Ramos-Arreguin, J. C. Moya Morales, J. . M. Ramos-Arreguin y J. C. Pedraza Ortega, «ResearchGate,» Diciembre 2012. [En línea]. Available: https://www.researchgate.net/publication/257743920_FPGA_Open_Architecture_Design_for_a_VGA_Driver#read.
- [50] NandLand, «NandLand - VGA Introduction,» [En línea]. Available: <https://www.nandland.com/goboard/vga-introduction-test-patterns.html>. [Último acceso: 20 Agosto 2020].
- [51] STMicroelectronics, «Getting started with SM32F0 hardware development.,» [En línea]. Available: https://www.st.com/resource/en/application_note/dm00051986-getting

- started-with-stm32f0x1x2x8-hardware-development-stmicroelectronics.pdf. [Último acceso: 20 Agosto 2020].
- [52] Wikipedia, «Wikipedia - SD card,» [En línea]. Available: https://en.wikipedia.org/wiki/SD_card. [Último acceso: 20 Agosto 2020].
- [53] D. T. Veeramanikandasamy, «Researchgate - SD card architecture,» [En línea]. Available: https://www.researchgate.net/figure/SD-Card-Architecture_fig2_301566757. [Último acceso: 20 Agosto 2020].
- [54] Molex, «Description Micro SD Card Connector,» [En línea]. Available: https://www.molex.com/pdm_docs/sd/1040310811_sd.pdf. [Último acceso: 20 Agosto 2020].
- [55] Mouser Electronics, «Mouser,» [En línea]. Available: <https://www.mouser.es/>. [Último acceso: 20 Agosto 2020].
- [56] Future Technology Devices International Ltd., «Datasheet FT2232H,» [En línea]. Available: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf. [Último acceso: 20 Agosto 2020].
- [57] Lattice Semiconductor, «iCE40 Programming and Configuration,» [En línea]. Available: http://www.latticesemi.com/view_document?document_id=46502. [Último acceso: 20 Agosto 2020].
- [58] Winbond, «Datasheet W25Q32JV,» [En línea]. Available: <https://www.winbond.com/resource-files/w25q32jv%20spi%20revc%2008302016.pdf>. [Último acceso: 20 Agosto 2020].
- [59] AlhambraBits, «GitHub - Alhambra II Schematics,» [En línea]. Available: https://github.com/FPGAwars/Alhambra-II-FPGA/blob/master/doc/PDF/Alhambra_II_Schematics.PDF. [Último acceso: 20 Agosto 2020].
- [60] STMicroelectronics, «STM32 microcontroller system memory boot mode,» [En línea]. Available: https://www.st.com/resource/en/application_note/cd00167594-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf. [Último acceso: 20 Agosto 2020].
- [61] ISSI, «Datasheet IS61WV25616EDBLL,» [En línea]. Available: <http://www.issi.com/WW/pdf/61-64WV25616EDBLL.pdf>. [Último acceso: 20 Agosto 2020].

- [62] Texas Instruments, «System Considerations For Using Bus-hold Circuits To Avoid Floating Inputs,» [En línea]. Available: https://www.ti.com/lit/an/scla015b/scla015b.pdf?ts=1598264092019&ref_url=https%253A%252F%252Fwww.google.com%252F. [Último acceso: 20 Agosto 2020].
- [63] Texas Instruments, «Datasheet SN74ALVCH16827,» [En línea]. Available: https://www.ti.com/lit/ds/symlink/sn74alvch16827.pdf?ts=1598264207407&ef_url=https%253A%252F%252Fwww.google.com%252F. [Último acceso: 20 Agosto 2020].
- [64] ISSI, «Datasheet IS62WV51216BLL,» [En línea]. Available: <http://www.issi.com/WW/pdf/62WV51216ALL.pdf>. [Último acceso: 20 Agosto 2020].
- [65] Waveshare, «IS62WV51216BLL SRAM Board,» [En línea]. Available: https://www.waveshare.com/wiki/IS62WV51216BLL_SRAM_Board. [Último acceso: 20 Agosto 2020].
- [66] Obijuan, «GitHub - Cuadernos-tecnicos-FPGAs-libres,» [En línea]. Available: <https://github.com/Obijuan/Cuadernos-tecnicos-FPGAs-libres/wiki/CT.3:-Pines-de-Entrada-Salida>. [Último acceso: 20 Agosto 2020].
- [67] Wikipedia, «Wikipedia - USB C,» [En línea]. Available: <https://en.wikipedia.org/wiki/USB-C>. [Último acceso: 20 Agosto 2020].
- [68] JAE, «JAE - DX07S016JA1R1500,» [En línea]. Available: https://www.jae.com/en/connectors/series/detail/product?id=91780&type_code=T1010. [Último acceso: 20 Agosto 2020].
- [69] Würth Elektronik, «Datasheet TVS Diode Array WE-TVS,» [En línea]. Available: <https://docs.rs-online.com/41f6/0900766b8107e996.pdf>. [Último acceso: 20 Agosto 2020].
- [70] Diodes Incorporated, «Datasheet PAM2306,» [En línea]. Available: http://wiki.amperka.ru/_media/products:raspberry-pi-zero-w:pam2306aypke_datasheet.pdf. [Último acceso: 20 Agosto 2020].
- [71] ON Semiconductor, «Datasheet NCP708,» [En línea]. Available: <https://www.onsemi.com/pub/Collateral/NCP708-D.PDF>. [Último acceso: 20 Agosto 2020].
- [72] PCBway, «PCBway,» [En línea]. Available: <https://www.pcbway.com/project/OnlineGerberViewer.html>. [Último acceso: 20 Agosto 2020].

- [73] JLPCB, «JLPCB,» [En línea]. Available: <https://jlpcb.com/>. [Último acceso: 20 Agosto 2020].