**Diseño de una placa entrenadora basada en una FPGA de software libre**

Alejandro Sánchez Doblado

**Palabras clave:**

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

**Resumen:**

En esta memoria se expone el proceso de diseño de una placa de circuito impreso open source en la que coexista un microcontrolador y una FPGA, así como distintos periféricos, manteniendo la versatilidad de esta como objetivo principal. Para ello se hará uso de herramientas de código libre como KiCad e IceStudio.

**Índice de contenido**

[1. Introducción 7](#_Toc46255138)

[1.1. Objetivos 7](#_Toc46255139)

[1.2. Motivación 7](#_Toc46255140)

[1.3. Estructura de la memoria 8](#_Toc46255141)

[1.4. Metodología y planificación 8](#_Toc46255142)

[2. Estado del arte 10](#_Toc46255143)

[2.1. Idea del diseño 10](#_Toc46255146)

[2.2. Open Source 12](#_Toc46255147)

[2.3. FPGAs 12](#_Toc46255154)

[2.3.1. Introducción a las FPGAs 12](#_Toc46255155)

[2.3.2. FPGAs open source 15](#_Toc46255156)

[2.4. Microcontrolador 17](#_Toc46255157)

[2.4.1. Introducción a los microcontroladores 17](#_Toc46255158)

[2.5. Diferencias y similitudes entre un microcontrolador y una FPGA 22](#_Toc46255159)

[3. Componentes y herramientas software 26](#_Toc46255160)

[3.1. Lattice ICE40 26](#_Toc46255161)

[3.1.1. iCE40HX4k 27](#_Toc46255162)

[3.2. IceStudio 31](#_Toc46255163)

[3.3. STM32 33](#_Toc46255164)

[3.3.1. ARM Cortex 33](#_Toc46255165)

[3.3.2. Familia de microcontroladores STM32 34](#_Toc46255166)

[3.3.3. STM32F091VC 37](#_Toc46255167)

[3.3.4. STM32F4DISCOVERY 40](#_Toc46255168)

[3.4. Software para STM32 42](#_Toc46255169)

[3.4.1. STM32CubeIDE 42](#_Toc46255170)

[3.4.2. Arduino IDE 44](#_Toc46255171)

[3.5. Introducción al diseño de PCBs 45](#_Toc46255172)

[4. Implementación 50](#_Toc46255173)

[4.1. Flujo de trabajo y pautas de diseño del producto 50](#_Toc46255174)

[4.2. Arquitectura final del sistema 52](#_Toc46255175)

[4.3. Diseño de la FPGA y sus periféricos 54](#_Toc46255176)

[4.3.1. Chip FPGA 54](#_Toc46255177)

[4.3.2. ADC 54](#_Toc46255178)

[4.3.3. VGA 59](#_Toc46255179)

[4.4. Diseño del microcontrolador y sus periféricos 64](#_Toc46255180)

[4.4.1. Chip STM32 64](#_Toc46255181)

[4.4.2. Tarjeta SD 64](#_Toc46255182)

[4.5. Programación de los dispositivos 67](#_Toc46255183)

[4.6. Diseño SRAM y bus de comunicación 74](#_Toc46255184)

[4.7. Alimentación global del sistema 74](#_Toc46255185)

[4.7.1. Análisis de consumo 74](#_Toc46255186)

[4.7.2. Conector USB 75](#_Toc46255187)

[4.7.3. Regulador de tensión 78](#_Toc46255188)

[4.8. Placa de circuito impreso 81](#_Toc46255189)

[4.9. Producto final 81](#_Toc46255190)

[4.9.1. Especificaciones 81](#_Toc46255191)

[4.9.2. Pinout 81](#_Toc46255192)

[4.9.3. BOM 81](#_Toc46255193)

[5. Conclusiones y trabajo futuro 81](#_Toc46255194)

[Bibliografía 81](#_Toc46255195)

**Índice de figuras**

[Figura 1: Diagrama de bloques que representa la estructura del TFG para la implementación de un controlador PID para un robot balancín usando una FPGA libre [3] 11](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331639)

[Figura 2: Estructura de una FPGA [7] 14](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331640)

[Figura 3: Celda lógica de una FPGA [8] 14](#_Toc46331641)

[Figura 4: IceZUM Alhambra II 17](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331642)

[Figura 5: Logotipo FPGAwars 18](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331643)

[Figura 6: Componentes de un microcontrolador 19](#_Toc46331644)

[Figura 7: Arquitectura Von Neumann 20](#_Toc46331645)

[Figura 8: Arquitectura Harvard 20](#_Toc46331646)

[Figura 9: Protocolo de comunicación UART [24] 22](#_Toc46331647)

[Figura 10: Protocolo de comunicación I2C [25] 22](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331648)

[Figura 11: Protocolo de comunicación SPI [26] 23](#_Toc46331649)

[Figura 12: Funcionalidad FPGA y microcontrolador 24](#_Toc46331650)

[Figura 13: Dispositivo iCE40 28](#_Toc46331651)

[Figura 14: Diagrama de bloques de un PLB 29](#_Toc46331652)

[Figura 15: Par de salida diferencial 30](#_Toc46331653)

[Figura 16: iCE40HX4K-TQ144 32](#_Toc46331654)

[Figura 17: Ejemplo de un multiplexor en IceStudio 33](#_Toc46331655)

[Figura 18: Interior de un bloque en IceStudio 33](#_Toc46331656)

[Figura 19: Clasificación de las subfamilias STM32 36](#_Toc46331657)

[Figura 20: Interfaz ST MCU Finder 37](#_Toc46331658)

[Figura 21: Diagrama de bloques STM32F091VC 39](#_Toc46331659)

[Figura 22: ST-Link V1 (izquierda) y V2 (derecha) 41](#_Toc46331660)

[Figura 23: Visualización del contenido en la memoria de la STM32F4DISCOVERY mediante ST-Link Utility 41](#_Toc46331661)

[Figura 24: STM32F4DISCOVERY 42](#_Toc46331662)

[Figura 25: Diagrama de bloques STM32F4DISCOVERY [41] 43](#_Toc46331663)

[Figura 26: Configuración de pines en STM32CubeIDE 44](#_Toc46331664)

[Figura 27: STM32CubeIDE configuración del reloj 45](#_Toc46331665)

[Figura 28: Blinking led en el Arduino IDE para la STM32F4DISCOVERY 46](#_Toc46331666)

[Figura 29: Resistencia THT [48] 47](#_Toc46331667)

[Figura 30: Resistencia de montaje superficial 47](#_Toc46331668)

[Figura 31: A la izquierda PCB en software CAD. A la derecha misma PCB ya fabricada [46] 48](#_Toc46331669)

[Figura 32: Diagrama de flujo de trabajo en KiCad 49](#_Toc46331670)

[Figura 33: Diagrama de bloques de la placa entrenadora 54](#_Toc46331671)

[Figura 34: Esquemático del ADS7924. [54] 55](#_Toc46331672)

[Figura 35: Encapsulado del ADS7924 [54] 56](#_Toc46331673)

[Figura 36: Diagrama de transición de código en el ADS7924 58](#_Toc46331674)

[Figura 37: Ejemplo de layout para el ADS7924 58](#_Toc46331675)

[Figura 38: Esquemático del ADC en KiCad 59](#_Toc46331676)

[Figura 39: Conector VGA D-SUB 15 [50] 60](#_Toc46331677)

[Figura 40: Circuito conversor R-2R para la señal RED 62](#_Toc46331678)

[Figura 41: Temporización de la señal HSync en VGA de 640x480 pixeles [52] 63](#_Toc46331679)

[Figura 42: Regiones activa y no activas de una pantalla VGA [53] 64](#_Toc46331680)

[Figura 43: Esquemático VGA en KiCad 64](#_Toc46331681)

[Figura 44: Arquitectura interna de una tarjeta SD 65](#_Toc46331682)

[Figura 45: Slot para tarjetas microSD de la marca Molex 66](#_Toc46331683)

[Figura 46: Esquemático en KiCad para el slot microSD 67](#_Toc46331684)

[Figura 47: Diagrama de bloques del FTD2232H 68](#_Toc46331685)

[Figura 48: Configuración SPI Master [59] 69](#_Toc46331686)

[Figura 49: Esquemático de la memoria Flash en KiCad [61] 70](#_Toc46331687)

[Figura 50: Diagrama de selección del bootloader para el STM32F091VC 71](#_Toc46331688)

[Figura 51: Pines para la selección del dispositivo a programar 72](#_Toc46331689)

[Figura 52: Esquemático del FTDI en Kicad (1) 73](file:///D:\Google%20Drive\5\tfg\memoria\TFG_1.docx#_Toc46331690)

[Figura 53: Esquemático alimentación para el FTDI en KiCad 74](#_Toc46331691)

[Figura 54: Contenido de la EEPROM del FTDI 74](#_Toc46331692)

[Figura 55: Diagrama de bloques de la memoria SRAM 75](#_Toc46331693)

[Figura 56: pinout de la memoria SRAM 76](#_Toc46331694)

[Figura 57: Esquemático de la memoria SRAM en KiCad 77](#_Toc46331695)

[Figura 58: Etapa de entrada de un circuito CMOS 78](#_Toc46331696)

[Figura 59: Transmisión bidireccional en un sistema de buses 78](#_Toc46331697)

[Figura 60: Resistencia de pull-up en una línea de bus 79](#_Toc46331698)

[Figura 61: Circuito de retención de buses 80](#_Toc46331699)

[Figura 62: Circuito simplificado del integrado de retención de buses 81](#_Toc46331700)

[Figura 63: Pinout del SN74ALVCH162827 81](#_Toc46331701)

[Figura 65: Pinout USB tipo C [62] 85](#_Toc46331702)

[Figura 66: Esquemático del conector USB C en KiCad 86](#_Toc46331703)

[Figura 67: Interior del circuito de protección ESD para USB [64] 86](#_Toc46331704)

[Figura 68: Pinout del PAM2306 87](#_Toc46331705)

[Figura 69: Diagrama de bloques PAM2306 88](#_Toc46331706)

[Figura 70: Implementación típica PAM2306 88](#_Toc46331707)

[Figura 71: Esquemático del PAM2306 en KiCad 89](#_Toc46331708)

**Índice de tablas**

[Tabla 1: Comparativa de tipos de memorias 20](#_Toc46255257)

[Tabla 2: Familia iCE40 26](#_Toc46255258)

[Tabla 3: Función del pin BOOT0 39](#_Toc46255259)

[Tabla 4: Pinout ADS7924 56](#_Toc46255260)

[Tabla 5: VGA pinout 60](#_Toc46255261)

[Tabla 6: Valores de RED para su correspondiente palabra digital 61](#_Toc46255262)

[Tabla 7: Pinout tarjeta SD en modo SPI 65](#_Toc46255263)

[Tabla 8: Descripción de las señales para la memoria Flash 68](#_Toc46255264)

[Tabla 9: Descripción de pines del canal B para protocolo UART 70](#_Toc46255265)

[Tabla 10: Consumo máximo en la alimentación de +3.3V 75](#_Toc46255266)

[Tabla 11: Pinout del conector DX07S016JA1R1500 76](#_Toc46255267)

**Lista de acrónimos**

1. *FPGA . Field Programmable Gate Array.*
2. *PID. Proporcional, Integrador, Derivador.*
3. *BOM. Bills Of Materials.*
4. *PCB. Printed Circuit Board.*
5. *HDL. Hardware Description Language.*
6. *PAL. Programmable Array Logic.*
7. *CPLD. Complex Programmable Logic Device.*
8. *ASIC. Application-specific integrated circuit.*
9. *LUT. Look-Up Table.*
10. *PLL. Phase-locked loop.*
11. *VHDL. Very High-Speed Integrated Circuit Hardware Description Language.*
12. *GPIO. General-Purpose Input/Output.*
13. *ADC. Analog-To-Digital Converter.*
14. *RAM. Random Access Memory.*
15. *EEPROM. Electrically Erasable Programmable Read-Only Memory*
16. *I2C. Inter-Integrated Circuit.*
17. *SPI. Serial Peripheral Interface.*
18. *PLB. Programmable Logic Blocks.*
19. *LQFP. Low-profile quad flat-package.*
20. *DMA. Direct Memory Access.*
21. *POR. Power On Reset.*
22. *PDR Power Down Reset.*
23. *SWD. Single Wire Debugging.*
24. *IDE. Integrated Development Environment.*
25. *LED. Light-Emitting Diode.*
26. *THT. Through-Hole Technology.*
27. *SMT. Surface-Mount Technology.*
28. *ECAD. Electronic Computer-Aided Design*
29. *VGA. Video Graphic Array*
30. *TTL. Transistor-Transistor Logic*
31. *LDO. Low-DropOut regulator*
32. *SD. Secure Digital*

# Introducción

## Objetivos

El objetivo general de este trabajo de fin de grado (en adelante, TFG) será diseñar y fabricar una placa entrenadora basada en una FPGA de 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 adecue tanto a los requisitos como a las limitaciones presentes.
* Estudio y elección de los componentes y herramientas apropiadas para el sistema.
* Diseño de la placa de circuito impreso.
* Testeo de algunos componentes del sistema mediante el uso de placas de evaluación.

## 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. A raíz de dicho trabajo, el estudiante se percató de las limitaciones que presentaba el sistema con el que trabajaba y propuso la elaboración de una placa de circuito impreso que solventara dichas limitaciones. Todo ello se explicará más detenidamente en *Idea del diseño.*

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

## Estructura de la memoria

La memoria de este TFG se estructurará en 6 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.
* **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 de la arquitectura final del sistema.
* **Capítulo 5:** Comentarios acerca del trabajo desarrollado, así como de posibles trabajos futuros, conclusiones y repaso de objetivos logrados.

## 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. Como se ha indicado anteriormente, la estructura de la memoria y el proceso de desarrollo de la placa serán muy similares.

Con respecto a la forma de trabajar, el alumno 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 alumno 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 en persona o por correo, para ir comentando el avance y las trabas encontradas.

# Estado del arte



## Idea del diseño

Como ya se ha explicado en *Motivación*, la idea de este TFG surgió a raíz de los problemas encontrados en uno anterior. En dicho TFG se utilizó la placa de desarrollo basada en hardware libre IceZUM Alhambra II [1], diseñada por Eladio Delgado Mingorance en Pinos del Valle, Granada. Se hablará de dicha placa con más detalle en el apartado *IceZUM Alhambra II* 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, también libre, el ATMega328 [2] presente en las famosas placas Arduino Uno.

El microcontrolador era el encargado de la adquisición de las señales mediante sensores, debido a la capacidad de estos dispositivos de ejecutar instrucciones periódicas 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 1, 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 : Diagrama de bloques que representa la estructura del TFG para la implementación de un controlador PID para un robot balancín usando una FPGA libre [3]

De dicha aplicación se puede ver 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 manejar 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 ya se ha comentado, uno de los principales problemas con los que contaba el sistema era la escasez de pines y la imposibilidad de ampliarlos de forma sencilla 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 encontrasen trabas adicionales:

* Para la adquisición y 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 de 32 bits 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.
7. …?

## Open Source

Open source (en español, código abierto) [4] [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 (en adelante BOM), placa de circuito impreso (en adelante PCB) código fuente del lenguaje de descripción hardware (en adelante HDL). Esta será pues, la filosofía a seguir en este trabajo de fin de grado, en el que se alojará en GitHub [6] todo el material referente al proyecto, para que cualquier usuario interesado lo utilice y modifique a su interés.



## FPGAs

### Introducción a las FPGAs

Una FPGA [7] es un dispositivo programable que contiene bloques lógicos cuya interconexión y funcionalidad puede ser configurada en el momento mediante un lenguaje de descripción hardware (HDL). 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.

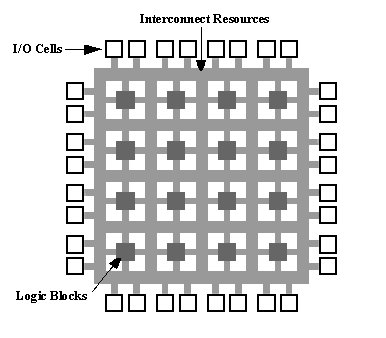
La arquitectura de una FPGA consiste en una serie de bloques lógicos, puertos I/O y buses de interconexión, como se muestran en la figura 2.

Figura : Estructura de una FPGA [7]

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.

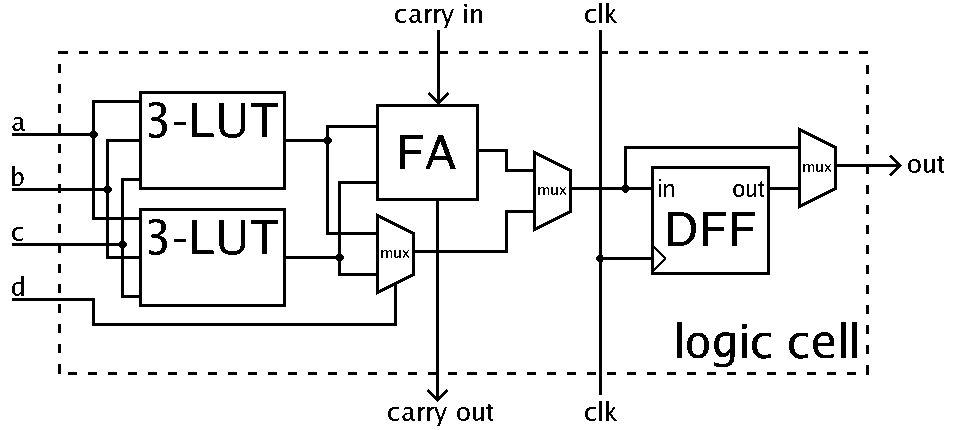


Figura : Celda lógica de una FPGA [8]

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.

A la hora de 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 que el 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 (skew) posible. También cuentan con PLLs para sintetizar nuevas frecuencias de reloj y atenuar las desviaciones de periodicidad de la señal (lo que se conoce 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 [9] y Altera (hoy en día una subsidiaria de Intel) [10], 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 [11] y VHDL [12].

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.

### 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 [13] de la compañía Lattice [14] mediante ingeniería inversa en lo que se conoce como proyecto Icestorm [15]. 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. Actualmente, incluso compañías como Xilinx han decidido aportar a la comunidad open source con sus proyectos como la plataforma de software Vitis [16].

#### IceZUM Alhambra II

Para el diseño de nuestra placa nos basaremos en la IceZUM Alhambra II [17] diseñada por Eladio Delgado en Pinos del Valle, Granada. En la figura 5 se puede ver una imagen de la IceZUM Alhambra II. Toda la información necesaria al respecto de la placa se encuentra alojada en GitHub:

* Esquemáticos
* Diseño de PCB
* BOM

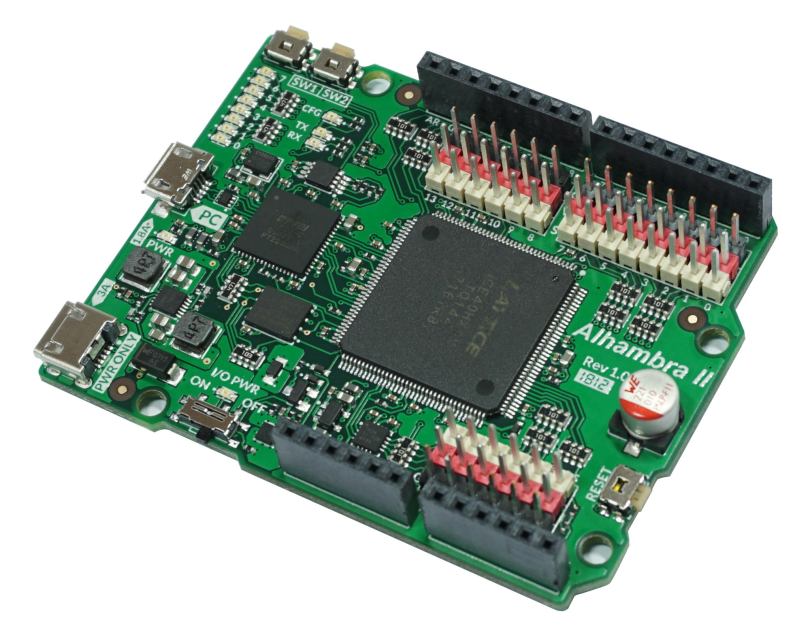
Ejemplos de códigoEntre las características principales de la placa destacan las siguientes:

Figura : IceZUM Alhambra II

* 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 FTDI2232H.
* 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 USB (5V hasta 4.8ª)

Siendo ésta un gran punto de partida para nuestra placa, pero que iremos modificando según las necesidades que se vayan fijando. Además, la usaremos para realizar distintas pruebas sobre nuestro diseño.

#### FPGAwars

FPGAwars [18] (figura 5) 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 : Logotipo FPGAwars

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.

## Microcontrolador

### 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 [19] es un circuito integrado programable, capaz de ejecutar una serie de órdenes almacenadas en su memoria. El primer microcontrolador de la historia fue el Intel 4004 de 4 bits fabricado en 1971 [20]. Los microcontroladores suelen estar definidos por el número de bits y la frecuencia de reloj a la que operan. En la figura 6 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.

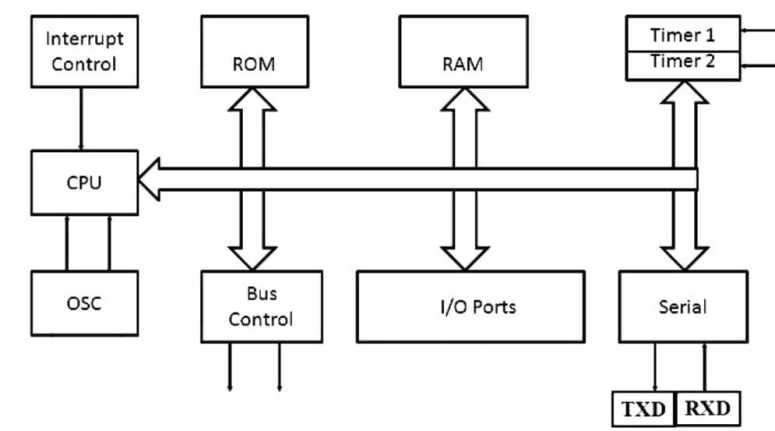


Figura : Componentes de un microcontrolador

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 [21] (figura 7) utiliza la misma memoria (RAM) tanto para las instrucciones como para los datos, por lo que no puede realizar simultáneamente una búsqueda de instrucciones y una operación de datos.

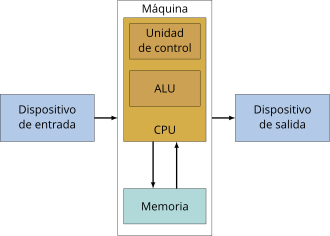


Figura : Arquitectura Von Neumann

#### Arquitectura Harvard

La arquitectura Harvard [22] (figura 8) 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.

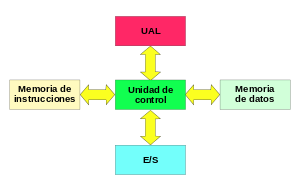


Figura : Arquitectura Harvard

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 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 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.

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

Tabla : 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 [23]. 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 9). 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.

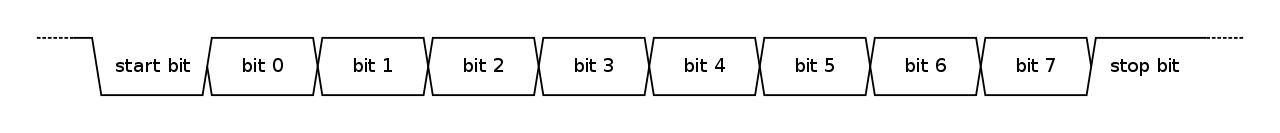


Figura : Protocolo de comunicación UART [24]

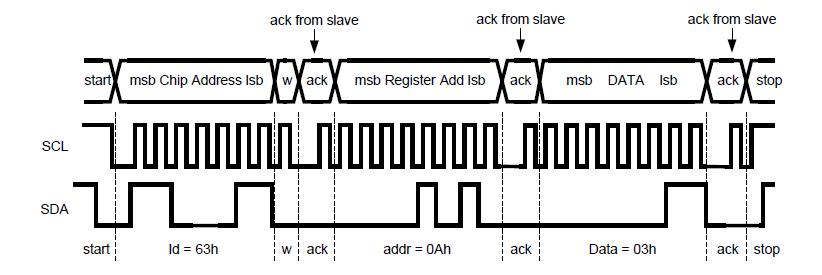
* + I2C: Comunicación serie síncrona usada comúnmente para comunicar el microcontrolador con módulos y sensores. Requiere de dos líneas, una de reloj (CLK) y una de datos (CDA). Permite varios masters y mantener hasta 128 esclavos por línea. Primero manda una señal de inicio, luego indica a quién se dirige, la dirección y por último el dato seguido de una condición de parada, tal y como se indica en la figura 10.

Figura : Protocolo de comunicación I2C [25]

* + 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 11 se incluye un ejemplo de funcionamiento del protocolo SPI.

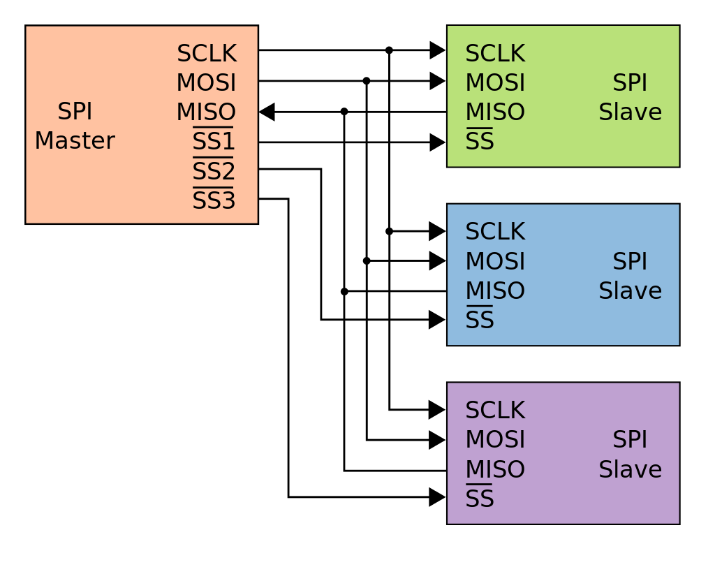


Figura : Protocolo de comunicación SPI [26]

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 es 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.

## 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 12).

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

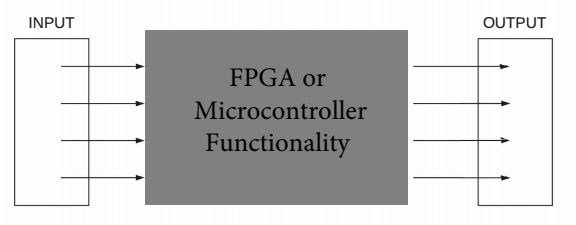


Figura : Funcionalidad FPGA y microcontrolador

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.
* El paralelismo presente en las FPGAs las hace idóneas para el control de interrupciones mediante lo que se conoce como 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 ahorran mucho del proceso.
* Las FPGAs tienen un mayor consumo debido a que contienen transistores inutilizados, árboles de reloj ineficientes 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.

# Componentes y herramientas software

## 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”,* nuestra elección se basa en una de las pocas FPGAs open source que existen actualmente en el mercado, el chip de Lattice iCE40 [27].

iCE40 es el nombre de una familia de FPGAs de bajo consumo y alto rendimiento de la compañía Lattice Semiconductor [28] cuyo toolchain fue liberado en 2015 como se comentaba en el apartado *FPGAs open source*. 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** |
| Logic cells | 384 | 640 | 1280 | 3520 | 7680 | 1280 | 3520 | 7680 |
| RAM bits | 0 | 32K | 64K | 80K | 128K | 64K | 80K | 128K |
| PLLs | 0 | 0 | 1 | 2 | 2 | 1 | 2 | 2 |
| I/O pins | 63 | 25 | 95 | 167 | 178 | 95 | 95 | 206 |
| Differential Input Pairs | 8 | 3 | 12 | 20 | 23 | 11 | 12 | 26 |

Tabla : Familia iCE40

Para nuestra aplicación utilizaremos el chip iCE40HX4K 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*.

### iCE40HX4k

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

La arquitectura del dispositivo (figura 13) 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.

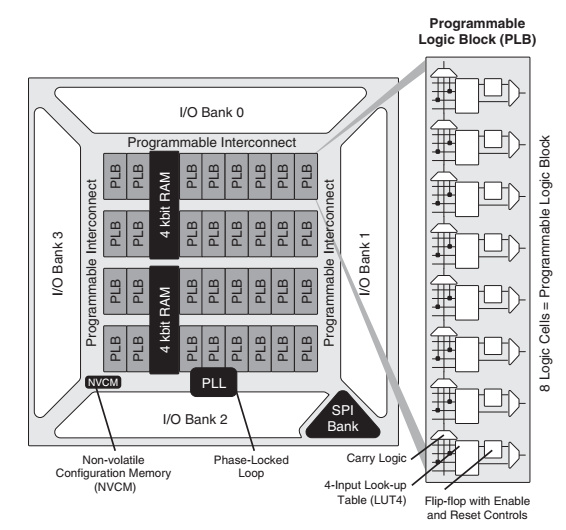


Figura : Dispositivo iCE40

La matriz central contiene en cada columna o bien celdas PLBs (figura 14) 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. Todo ello interconectado mediante una red configurable de señales.



Figura : Diagrama de bloques de un PLB

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 [29] 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 15.

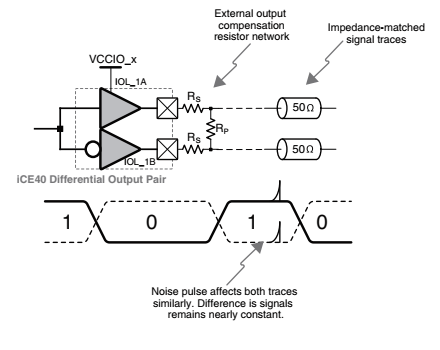


Figura : Par de salida diferencial

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 NVCM mediante el puerto SPI.
2. Escribir en la memoria RAM de configuración (CRAM) la información contenida en la NVCM.
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, optaremos por 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*.

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.

Aunque haremos un análisis más detallado de esto en el apartado de Alimentación del sistema.

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 16).

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.



Figura : iCE40HX4K-TQ144

## IceStudio

En esta sección hablaremos de la herramienta IceStudio [30] 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 [31], un ecosistema inspirado en PlatformIO [32], 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 muy intuitiva (figura 17). Lo cual lo hace muy atractivo para usuario más novatos sin experiencia en Verilog pero sin perder en ningún momento flexibilidad ya que cada bloque puede contener el código Verilog que se desee (figura 18), dejando al usuario trabajar con el nivel de abstracción que desee. Por lo tanto, nos encontramos con una herramienta versátil, intuitiva y potente apoyada por la comunidad FPGAwars que desarrolla bloques de forma similar a las librerías que se desarrollan para Arduino y que hace de este microcontrolador una opción interesante para iniciados en el ámbito de los microcontroladores.

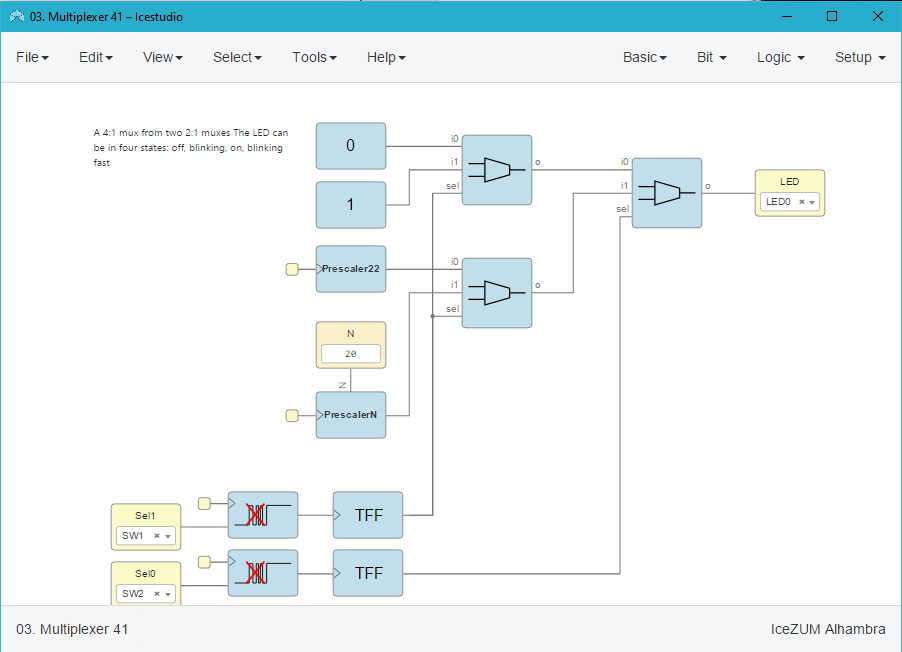


Figura : Ejemplo de un multiplexor en IceStudio

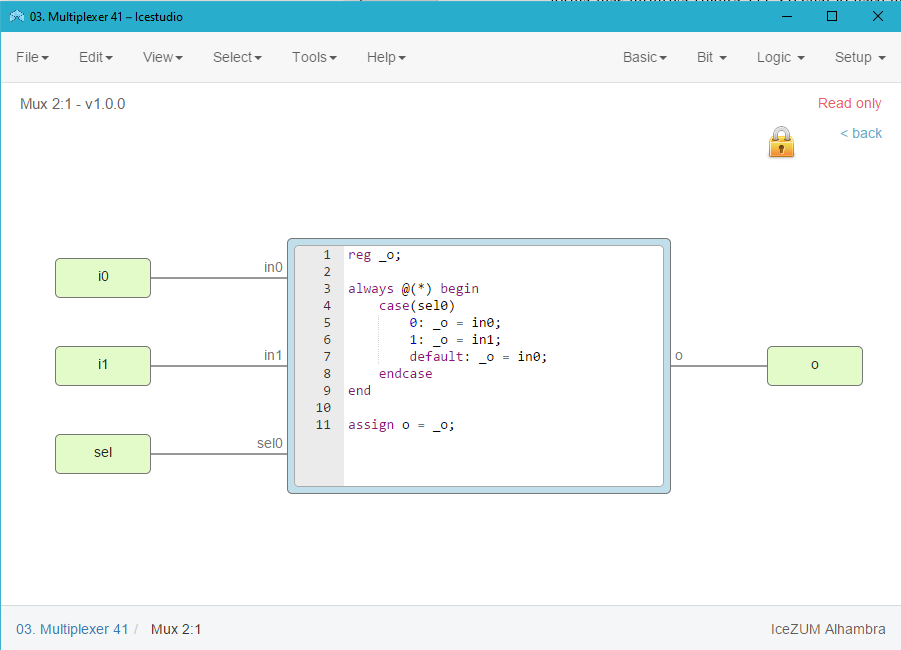


Figura : Interior de un bloque en IceStudio

Algunas de las placas soportadas por IceStudio son:

* IceZUM Alhambra I y II
* Nandland Go board [33]
* BlackIce I y II [34]
* TinyFPGA [35] 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*.

## STM32

En este apartado se introducirá brevemente a la arquitectura de procesadores ARM Cortex [36] en la que se basa la familia de microcontroladores STM32 [37] de la compañía STMicroelectronics [38] para después centrarnos en el chip que se elija en concreto, así como la placa de evaluación para realizar distintas pruebas.

### ARM Cortex

ARM Cortex es una arquitectura de procesadores RISC, que debe contar con la licencia de Arm Holdings [38], 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.

### 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 en 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 mas 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 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 cuente con una comunidad tan extensa.

Dentro de la familia STM32 podemos clasificar las diferentes subfamilias en las siguientes categorías (figura:

* 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.

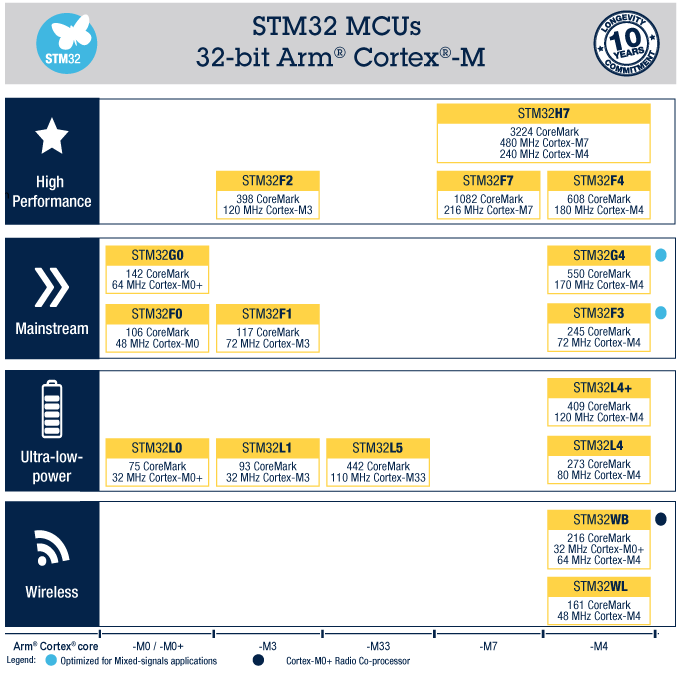


Figura : Clasificación de las subfamilias STM32

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” 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 ha especificado con detalle las características de la placa y, por lo tanto, los requerimientos del microcontrolador. De ello se tratará en profundidad en el capítulo 4, *Implementación.* 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

Con esas características y centrándonos en la subfamilia F0 encontramos el STM32F091VC [40] (figura 20), que será el que utilicemos en el diseño de nuestra placa.

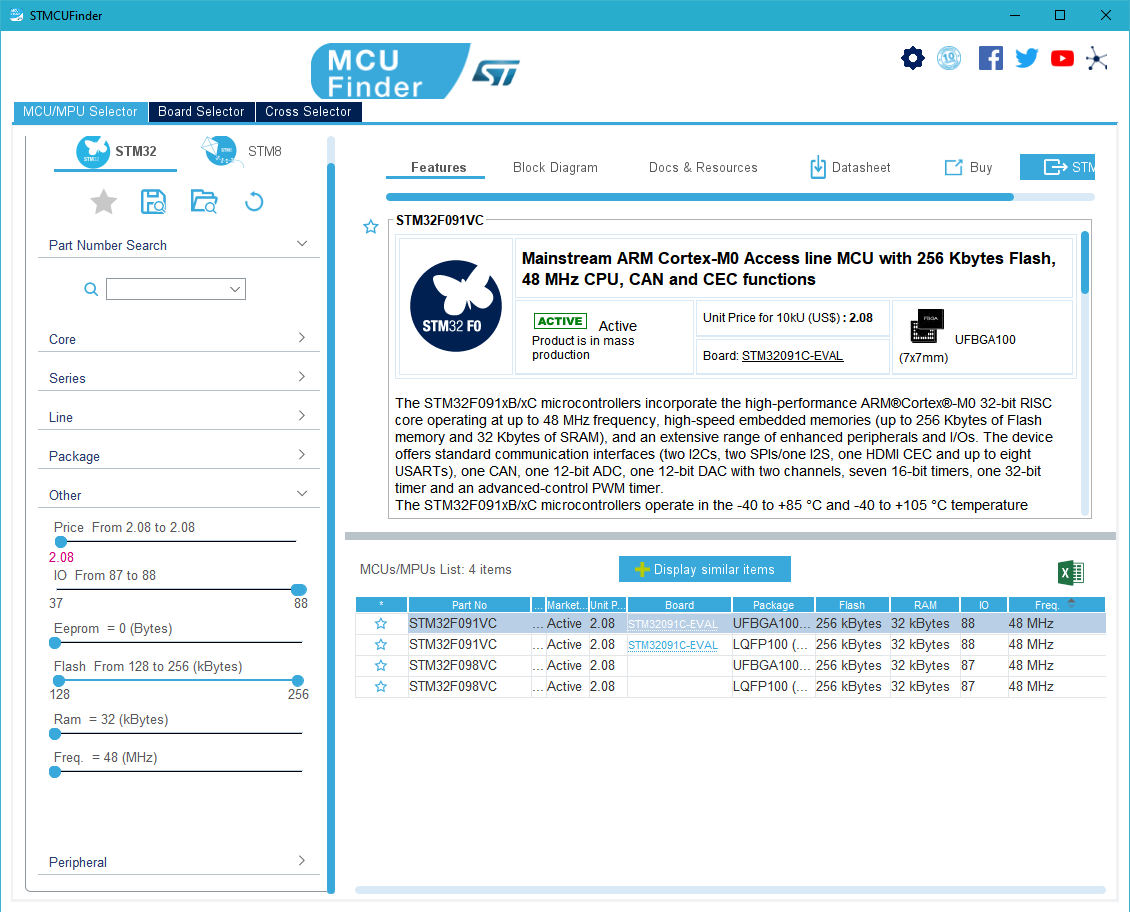


Figura 20: Interfaz ST MCU Finder

Para la 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 [41].

### STM32F091VC

En este apartado introduciremos el microcontrolador que emplearemos en nuestro diseño, haciendo un repaso de sus características más relevantes al igual que ya se hizo en el apartado *iCE40HX4k* 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 a 48MHz 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 21.

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 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.

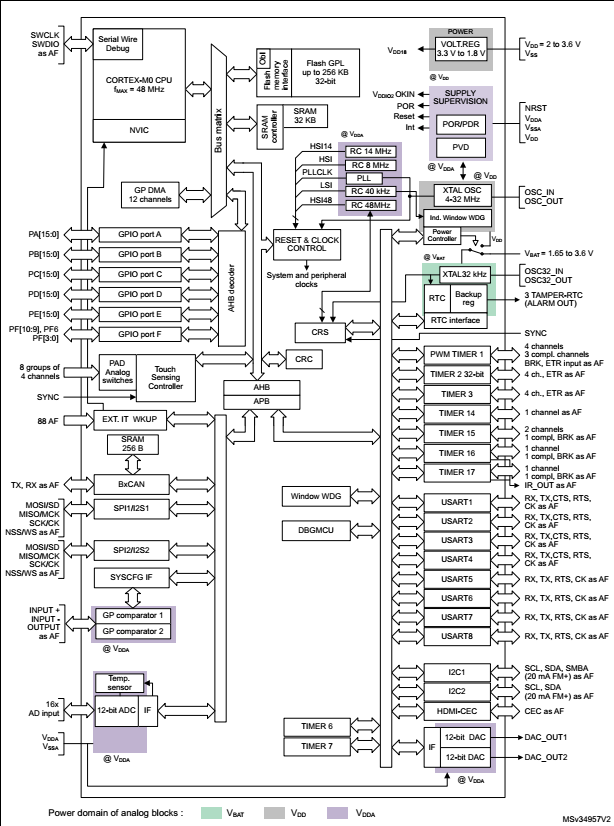


Figura : Diagrama de bloques STM32F091VC

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).

|  |  |
| --- | --- |
| BOOT0 pin | Modo |
| 0 | Memoria Flash |
| 1 | Memoria SRAM |

Tabla : Función del pin BOOT0

Además, 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.* 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 [42] (figura 22). 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 23) como de terceros (IAR EWARM o Keil MDK-ARM µVision). Se comunica con el PC mediante interfaz USB 2.0.



Figura : ST-Link V1 (izquierda) y V2 (derecha)

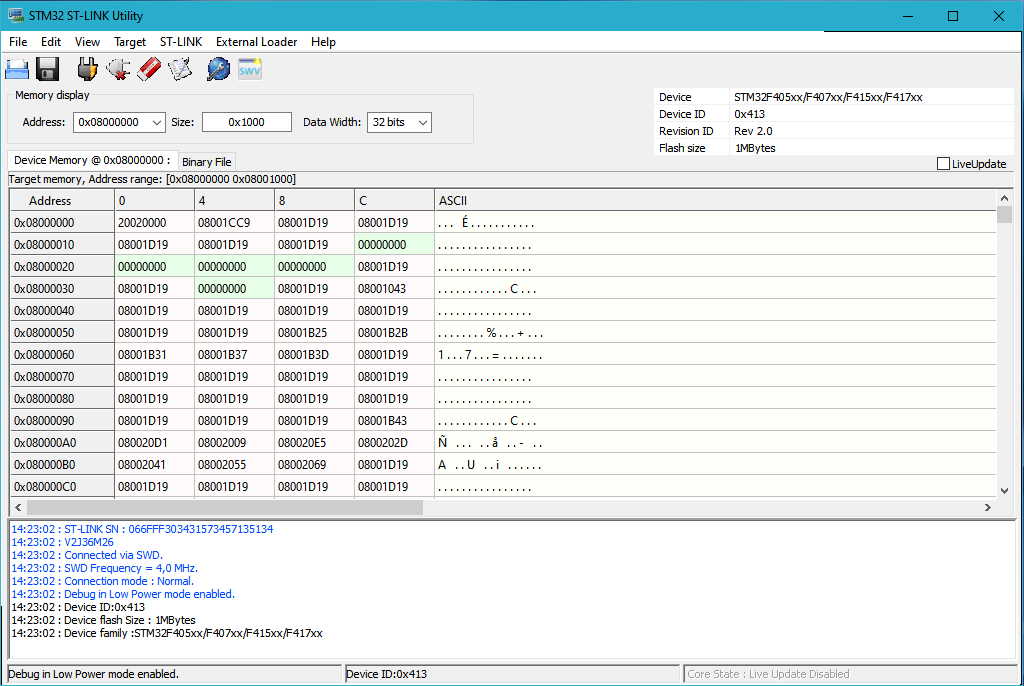


Figura : Visualización del contenido en la memoria de la STM32F4DISCOVERY mediante ST-Link Utility

### 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 nuestro 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:

* Núcleo 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 24 Y 25). Las características más relevantes 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.

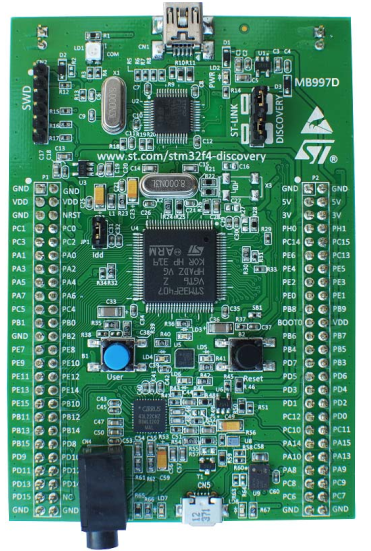


Figura : STM32F4DISCOVERY



Figura : Diagrama de bloques STM32F4DISCOVERY [41]

## 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
* Arduino IDE

### 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 [43] una herramienta multiplataforma que engloba a todas esas aplicaciones. Con ella podemos:

* Escribir y compilar código en C/C++.
* Configurar los pines periféricos (figura 26).
* Ajustar la frecuencia del reloj (figura 27).
* Programar dispositivos
* Depurar mediante software y mediante ST-Link.
* Monitorizar variables.

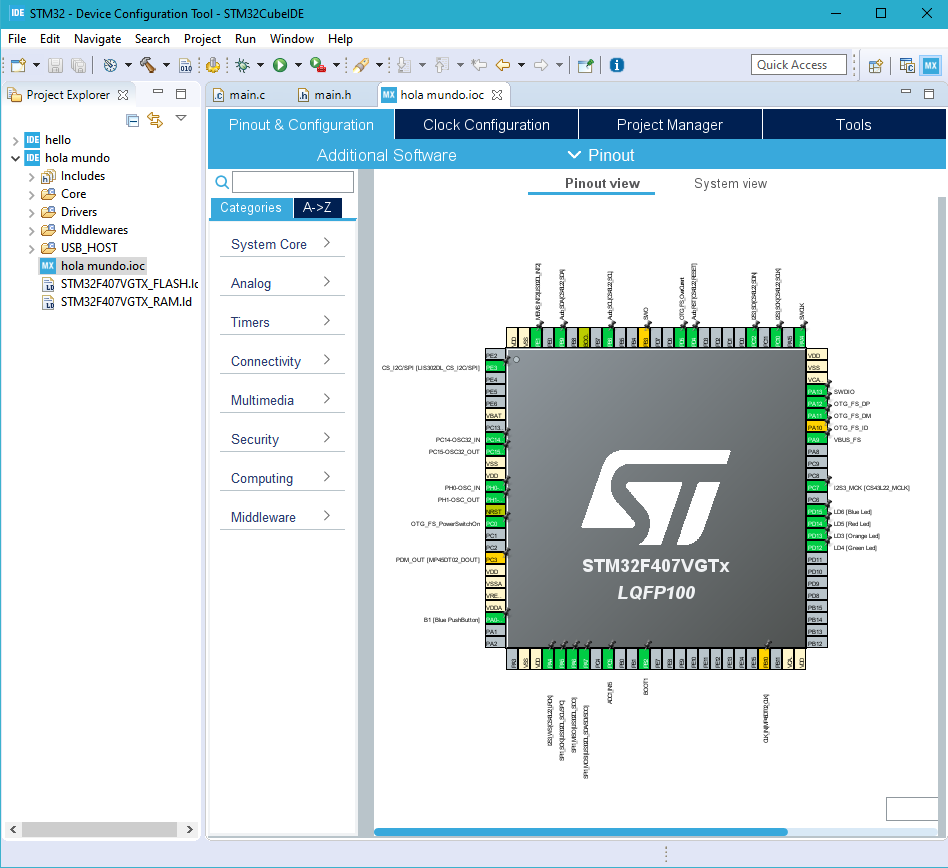


Figura : Configuración de pines en STM32CubeIDE

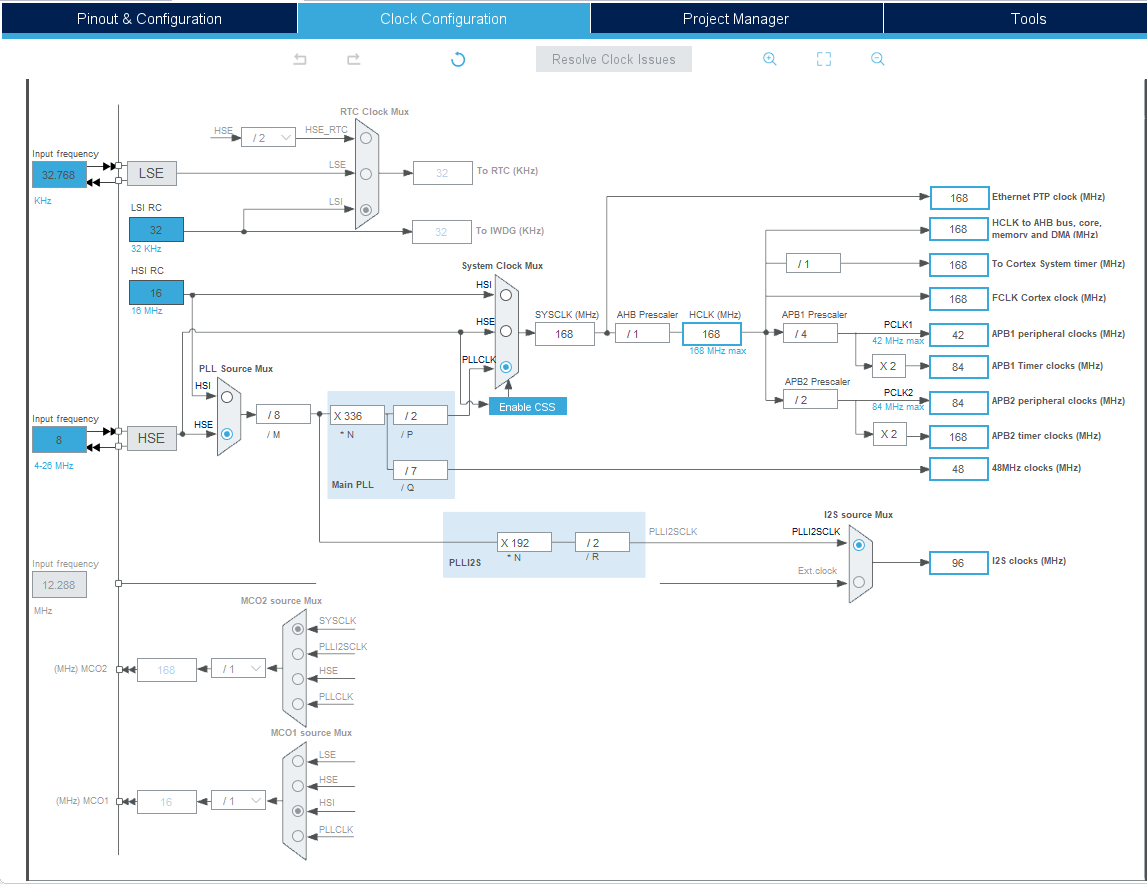


Figura : STM32CubeIDE configuración del reloj

### Arduino IDE

La otra opción que vamos a comentar brevemente en este texto es el ampliamente conocido Arduino IDE [44]. Arduino [45] 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 [46], podemos usarla para desarrollar y subir código a nuestra placa entrenadora STM32F4DISCOVERY (figura 28). 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 la librería HAL [47]. Es por ello, que cuando se requiera probar una aplicación sencilla 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 este, se debe optar por las herramientas que proporciona el fabricante y las de terceros que sean compatibles.

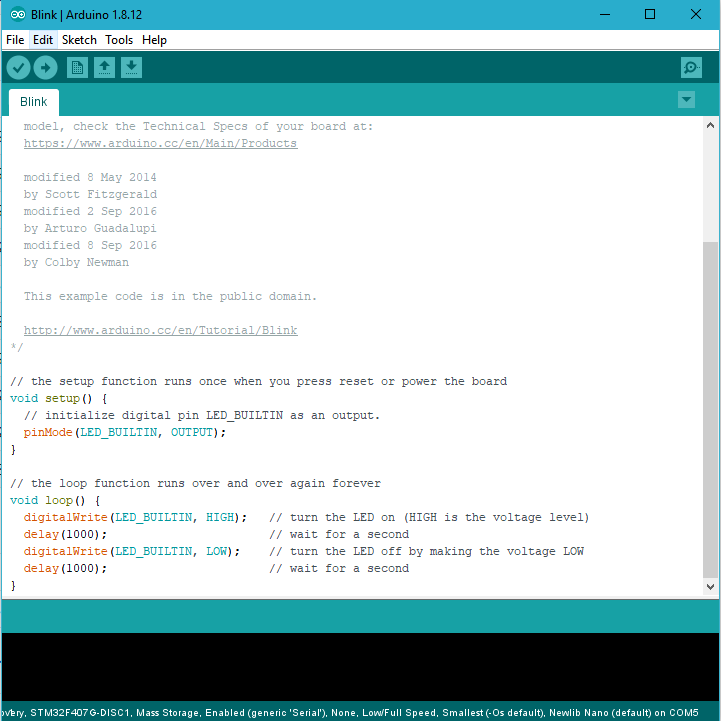


Figura 28: Blinking led en el Arduino IDE para la STM32F4DISCOVERY

## Introducción al diseño de PCBs

Una PCB [47] 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 PCBs:

* 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.

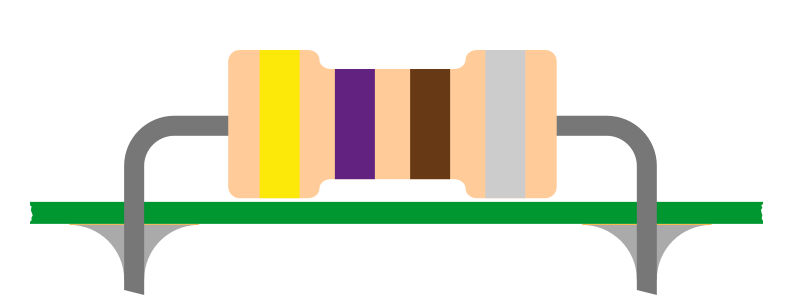


Figura : Resistencia THT [48]



Figura : Resistencia de montaje superficial

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 (figura 31) 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.

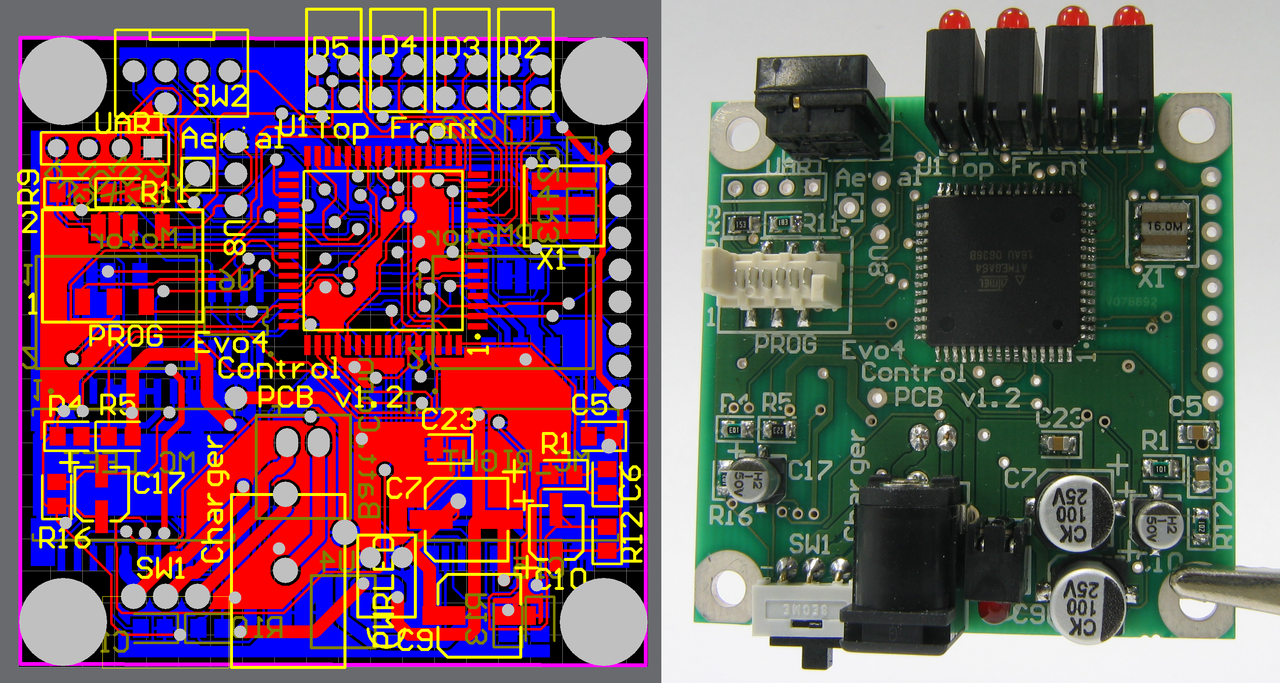


Figura : A la izquierda PCB en software CAD. A la derecha misma PCB ya fabricada [46]

Para este TFG utilizaremos KiCad al ser una herramienta open source. Con ello pretendemos que cualquiera que esté interesado pueda acceder y modificar el esquemático y la PCB a su antojo ya que todo el material necesario estará alojado en GitHub.

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

1. Diseño del esquemático y creación de los símbolos de los componentes.
2. Diseño del footprint de los componentes y de las pistas de conexión (layout).
3. 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.

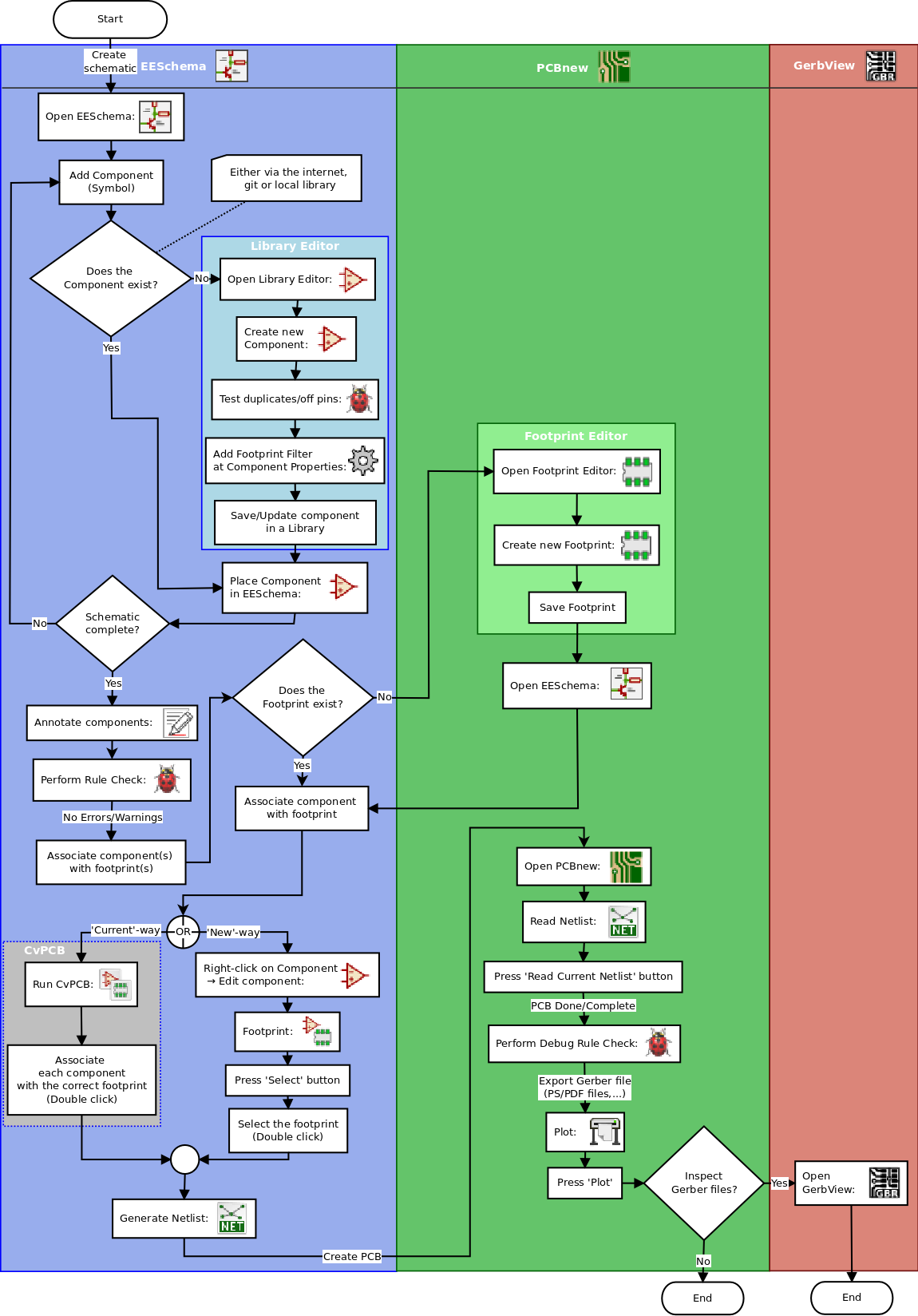


Figura : Diagrama de flujo de trabajo en KiCad

Como se puede observar en la figura 31, 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 creación de los archivos Gerber

# Implementación

## Flujo de trabajo y pautas de diseño del producto

En este capítulo se guiará al lector a través del proceso de diseño de la placa entrenadora una vez que ya es conocida su finalidad y sus componentes principales.

Primero haremos un repaso a 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 parte individual de la arquitectura e indicaremos los componentes empleados y su utilidad, 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. Luego, presentaremos el layout final de la placa de circuito impreso y su modelo 3D. Por último, resumiremos las especificaciones del producto e indicaremos su pinout.

Para el proceso de selección de componentes se ha seguido la serie de 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 distintos, intentando siempre reutilizar valores ya disponibles.
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.

## Arquitectura final del sistema

En este apartado se indicarán los distintos módulos que componen la placa, así como el apartado concreto donde el lector podrá encontrar más información al respecto. En la figura 33 podemos ver los distintos módulos que componen el sistema, así como las conexiones entre ellos. A continuación, se indican los módulos que componen el sistema:

* Una entrada USB, que permita tanto la alimentación como la programación de ambos dispositivos mediante un PC. Se repasará en el subapartado *Conector USB.*
* Un módulo de alimentación que proporcione los distintos niveles de tensión requeridos por el sistema. Detallado en la sección *Convertidor DC-DC*.
* 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 se indica como FTDI, y se explicará con detenimiento en el apartado *Programación de los dispositivos*.
* Una FPGA. Detallado en el subapartado *Chip FPGA*.
* Un microcontrolador. Se explicará con profundidad en el subapartado *Chip STM32.*
* Una memoria SRAM. Detallado en el apartado *Diseño SRAM y bus de comunicación*.
* Una salida VGA. Comentado en la sección *VGA.*
* Un ADC para la FPGA, el cual repasaremos en el subapartado *ADC.*
* Un slot para tarjetas microSD del cual hablaremos en el subapartado *Tarjeta SD.*

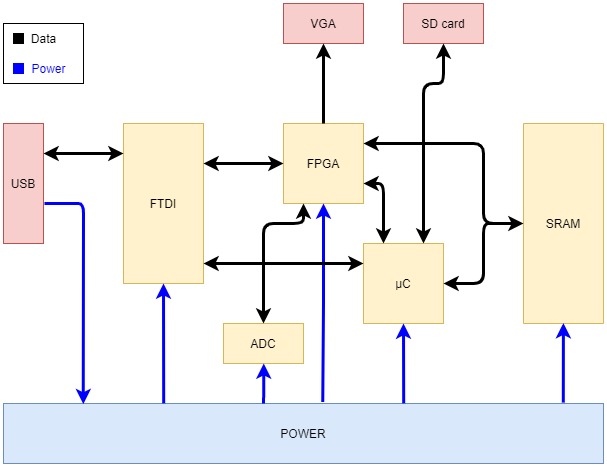


Figura : Diagrama de bloques de la placa entrenadora

## Diseño de la FPGA y sus periféricos

### Chip FPGA

### ADC

Un ADC es un circuito que se encarga de convertir las señales analógicas en señales digitales. Puesto que la FPGA sólo entiende 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 [54] 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 34.

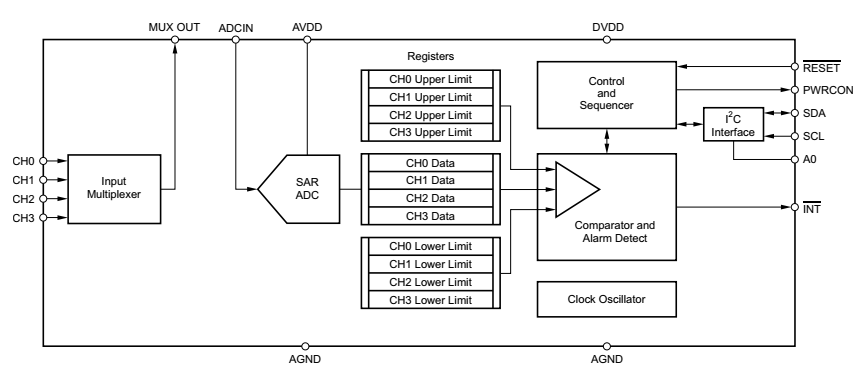


Figura : Esquemático del ADS7924. [54]

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 35 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.

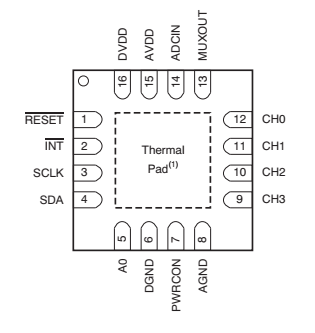


Figura : Encapsulado del ADS7924 [54]

|  |  |  |  |
| --- | --- | --- | --- |
| 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 : 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 podemos ver en la figura 36, el cual corresponde al típico conversor con cuantificación uniforme salvo para el último.

En la figura 38 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 I2C.
* Las entradas analógicas se han protegido con unas resistencias serie de 100Ω.
* Para la PCB se separará la parte analógica de la digital para evitar interferencias como se muestra en la figura 38. Se hablará de ello con más detalle en el apartado *Placa de circuito impreso.*

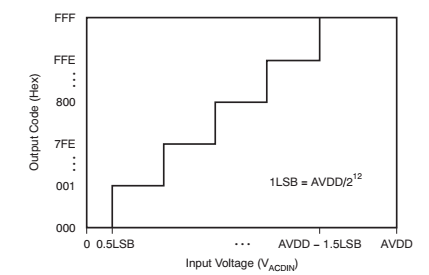


Figura : Diagrama de transición de código en el ADS7924

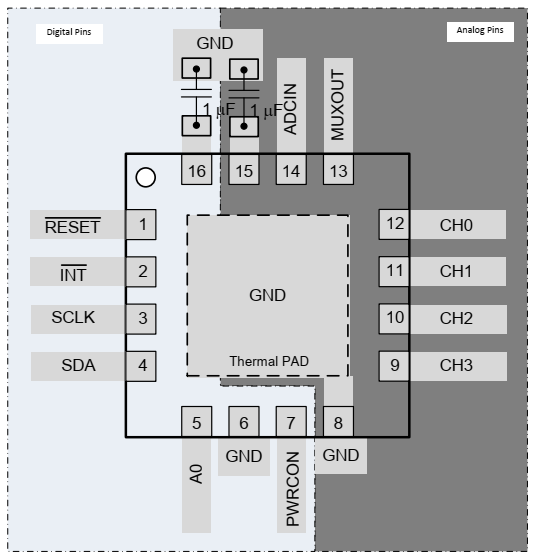


Figura : Ejemplo de layout para el ADS7924

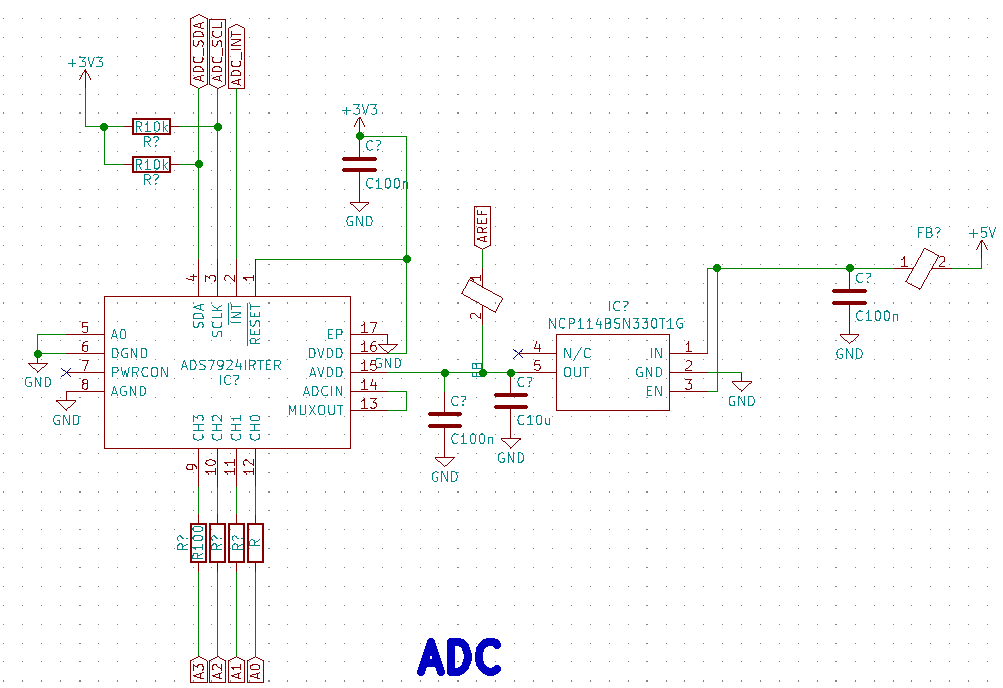


Figura : Esquemático del ADC en KiCad

### VGA

VGA [50] 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 34). 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 sólo con señales digitales.



Figura : Conector VGA D-SUB 15 [50]

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 4 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 |
| Pin 8 | BLUE\_RTN | Retorno del azul |
| Pin 9 | +5V | +5V |
| Pin 10 | GND | Tierra (VSync) |
| Pin 11 | N/C | No conectar |
| Pin 12 | SDA | Datos I2C |
| Pin 13 | HSync | Sincronización horizontal |
| Pin 14 | VSync | Sincronización vertical |
| Pin 15 | SCLA | Velocidad del reloj I2C |

Tabla : 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 son 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 35 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 Vo sea 0.7V.
* La resistencia Thévenin vista desde “RED” coincida con la resistencia interna de entrada de los monitores VGA (75 Ω) para mejorar los transitorios en la línea.
* Sean valores comerciales.

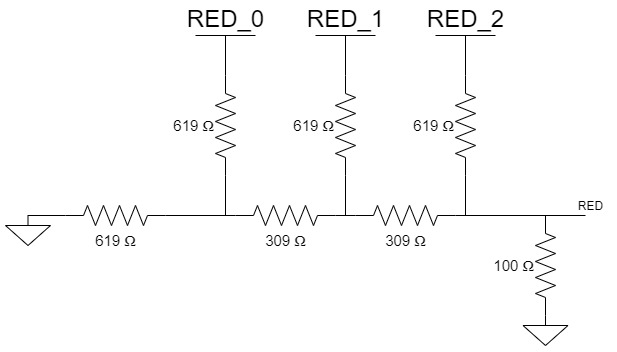


Figura : Circuito conversor R-2R para la señal RED

Simulando el circuito en LTSpice [51] obtenemos los resultados de la tabla 5:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 : Valores de RED para su correspondiente palabra digital

Por otro, 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 (retrace time). 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 36), unos tiempos donde la señal debería a 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 37. Estas señales son las que nos permiten conocer en qué lugar de la pantalla estamos en cada momento.

Los tiempos que debe tener cada señal están estandarizados según la frecuencia del reloj y la resolución y la tasa de refresco del monitor y pueden encontrarse en la web.

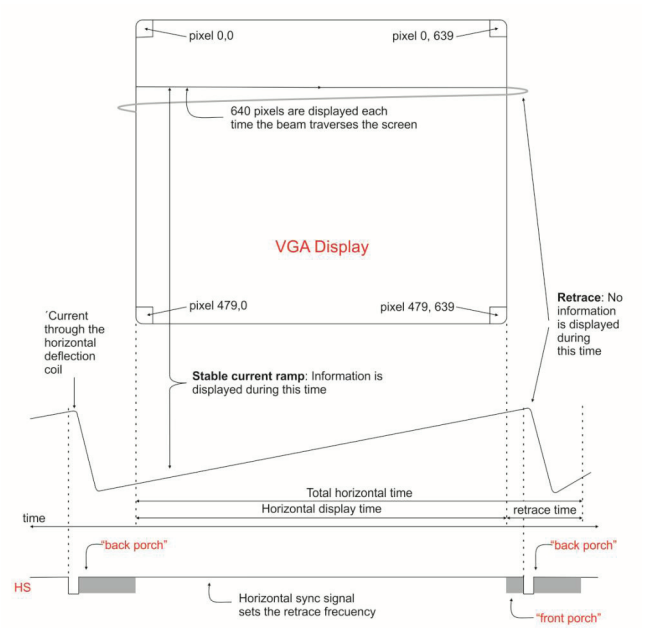


Figura : Temporización de la señal HSync en VGA de 640x480 pixeles [52]

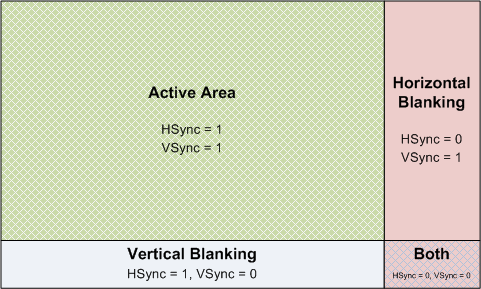


Figura : Regiones activa y no activas de una pantalla VGA [53]

Por último, se muestra en la figura 43 el esquemático del circuito para la salida VGA en KiCad.

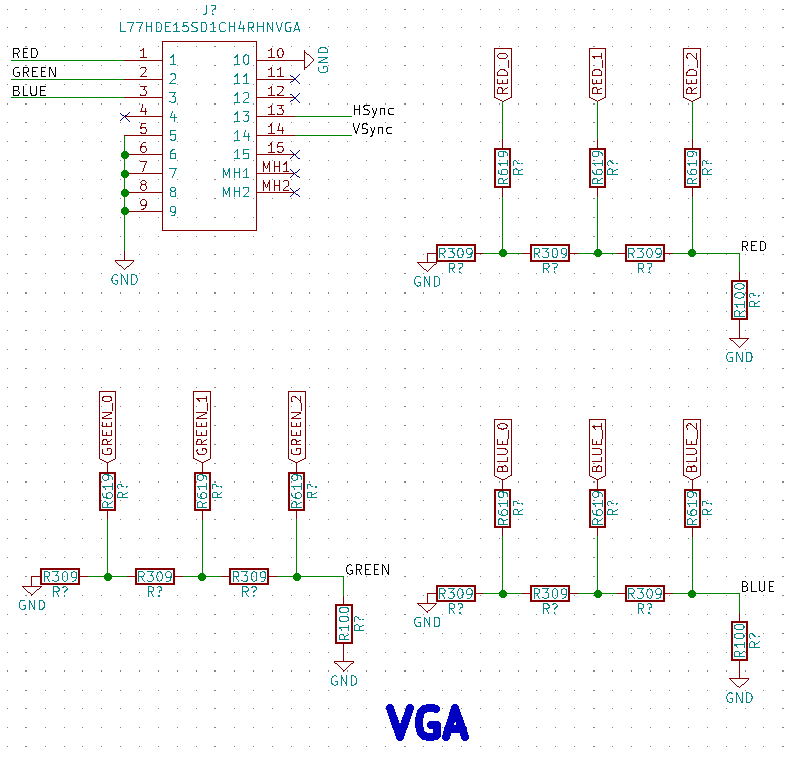


Figura : Esquemático VGA en KiCad

## Diseño del microcontrolador y sus periféricos

### Chip STM32

### Tarjeta SD

SD [55] 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 43.

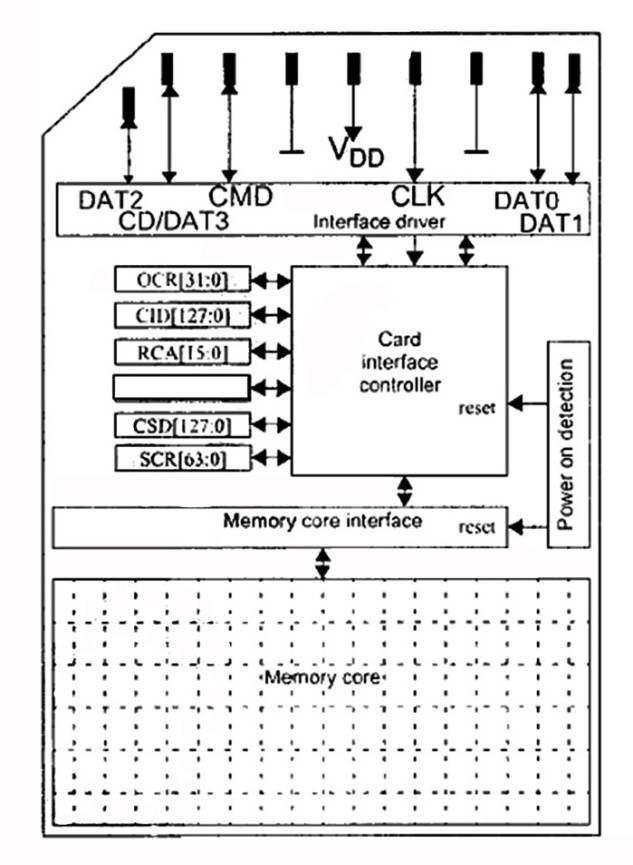


Figura : Arquitectura interna de una tarjeta SD

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 muestra 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 es a 3.3V.

|  |  |  |
| --- | --- | --- |
| PIN NO. | PIN NAME | Descripción |
| 1 | X | Sin usar en modo SPI |
| 2 | CS | Chip Select |
| 3 | DI | Data Input |
| 4 | VDD | Alimentación +3.3V |
| 5 | SCLK | Serial Clock |
| 6 | VSS | Ground |
| 7 | D0 | Data Out |
| 8 | X | Sin usar en modo SPI |

Tabla : Pinout tarjeta SD en modo SPI

Para nuestro diseño utilizaremos el slot de la marca Molex [57] de la figura 44. En la figura 45 podemos ver el esquemático del circuito correspondiente.

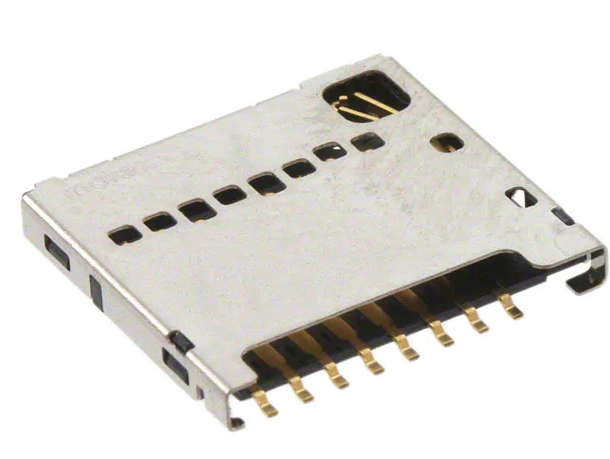


Figura : Slot para tarjetas microSD de la marca Molex

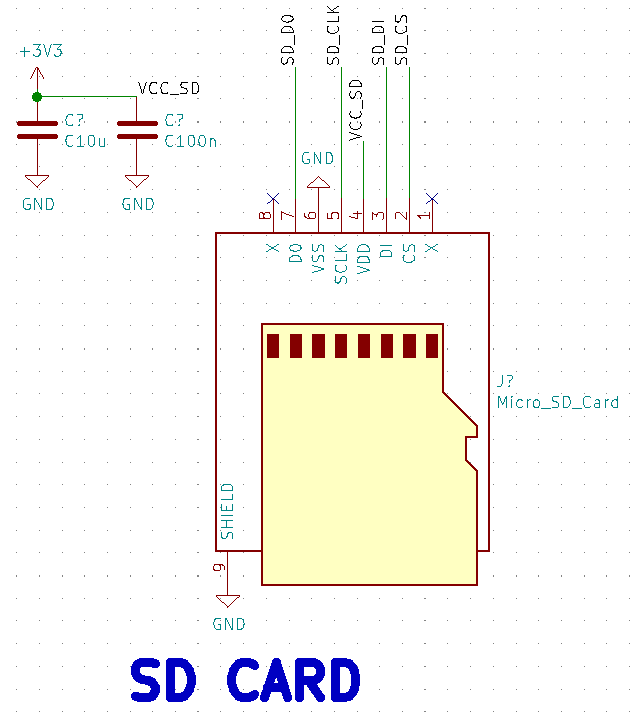


Figura : Esquemático en KiCad para el slot microSD

## 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, por lo que nos basaremos en su implementación.

El chip encargado de este proceso será el FT2232H [58] de la marca FTDI [59]. 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 47.

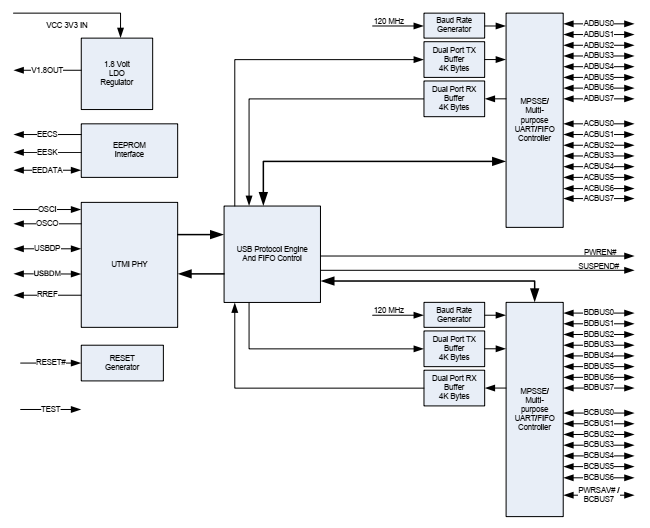


Figura : Diagrama de bloques del FTD2232H

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 el chip de la FPGA leería mediante el banco SPI. Esta configuración se denomina “SPI Master Configuration Interface” [59] ya que la FPGA actúa como máster sobre la memoria Flash como podemos ver en la figura

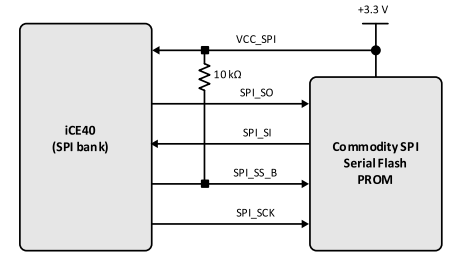


Figura : Configuración SPI Master [59]

La definición de los pines presentes en la figura 48 lo encontramos en la tabla

|  |  |
| --- | --- |
| 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 : 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. Además, 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 [60], 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 podemos ver la conexión en esquemático entre la memoria y la FPGA, así como las señales que salen hacia el FTDI.

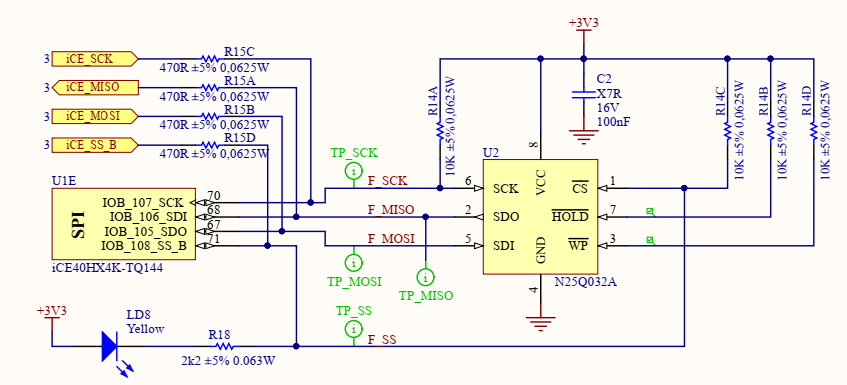


Figura : Esquemático de la memoria Flash en KiCad [61]

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. En la figura 52 se podrá observar que las etiquetas del canal A corresponden a las señales necesarias por el protocolo SPI.

Para programar el microcontrolador 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 50 podemos ver el proceso de selección del bootloader para el microcontrolador.

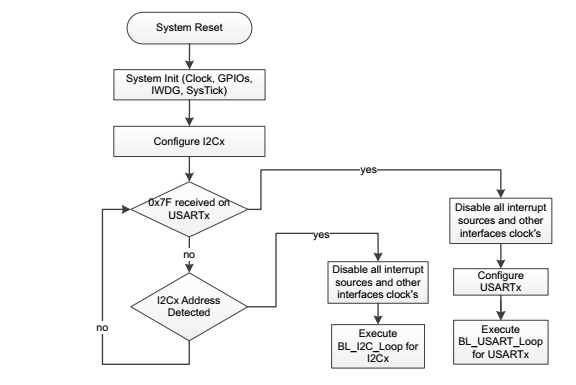


Figura : Diagrama de selección del bootloader para el STM32F091VC

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 : Descripción de pines del canal B para protocolo UART

Cabe destacar que, aunque ambos canales utilicen protocolos diferentes, la información de ambos canales será idéntica por lo que deberemos de 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 51. Así, si colocamos un jumper sobre los pines 1 (iCE\_CS\_B) y 2 (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 3 y 4, el microcontrolador arrancará en modo “empty” 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 ejecutaran el contenido cargado en sus memorias.

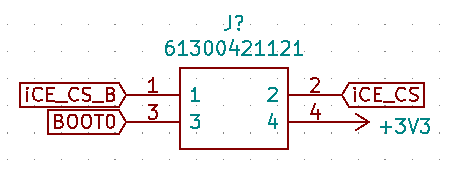


Figura : Pines para la selección del dispositivo a programar

Por último, vamos a comentar el circuito resultante para el FTDI. En la figura 52 podemos ver 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 usamos antes sobre la FPGA.
* 93LC56C es una memoria EEPROM de 2K. Su función es almacenar la configuración deseada para el FDTI. Podemos ver el contenido de la misma en la figura 54.
* El pin TEST se ha conectado a masa como indica el fabricante.
* El pin REF se ha conectado a tierra mediante una resistencia de 12kΩ 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 53 podemos ver las distintas señales de alimentación del dispositivo. Dichas señales son:

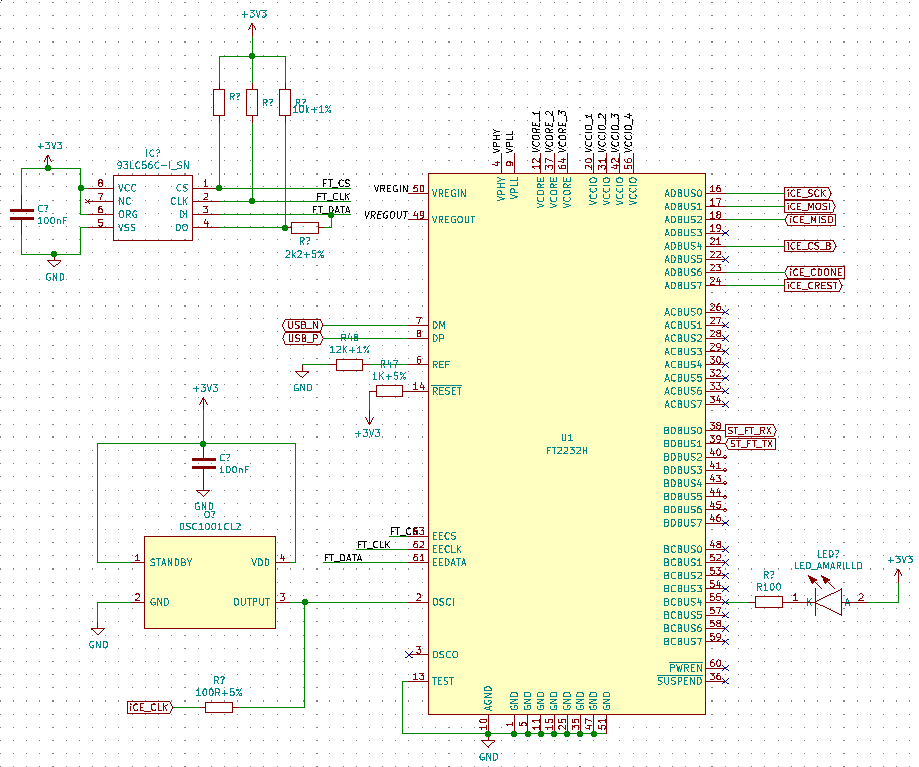
* VREGIN: Entrada de +3.3V del regulador integrado en el FTDI. Desacoplado con 100nF.
* VREGOUT: Salida de +1.8V del regulador integrado. Desacoplado con un condensador de 10uF y conectado a VCORE.
* VCORE: Entrada principal de alimentación de +1.8V. Se ha conectado desde VREGOUT y cada entrada se ha desacoplado con un condensador de 100nF.
* VPLL: 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.

Figura : Esquemático del FTDI en Kicad (1)

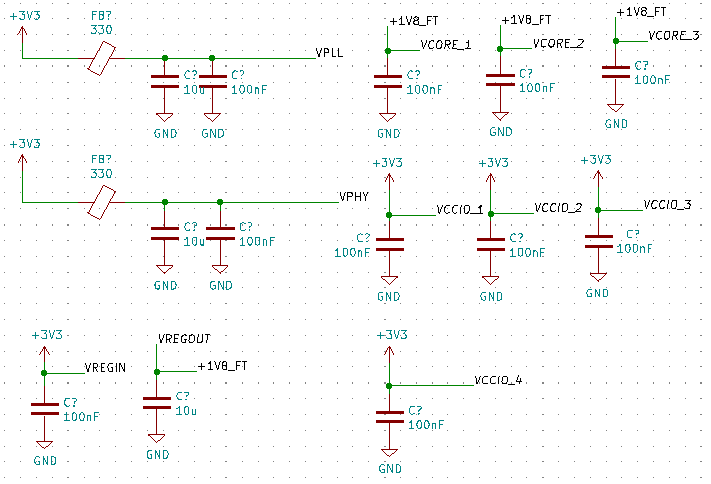


Figura : Esquemático alimentación para el FTDI en KiCad

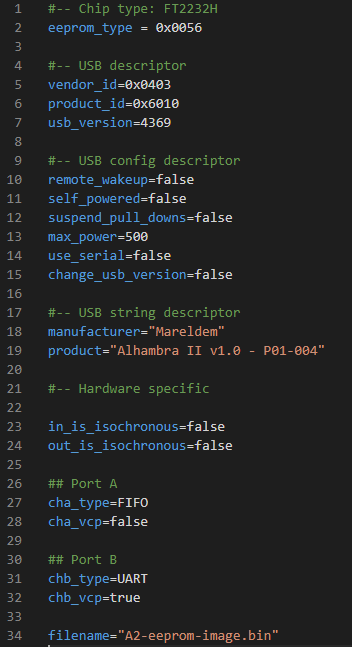


Figura : Contenido de la EEPROM del FTDI

## Diseño SRAM y bus de comunicación

Como se comentaba al principio de este texto, en el apartado *Idea del diseño*, alguno de los principales inconvenientes con lo que se encontraba el estudiante a la hora de implementar su aplicación eran:

* Por una parte, la escasa memoria de alta velocidad de la que disponían el microcontrolador utilizado, Arduino Uno, y 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 entre los dos dispositivos mediante la SRAM.

La memoria SRAM elegida para la implementación es la IS61WV25616EDBLL [62], 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 55.

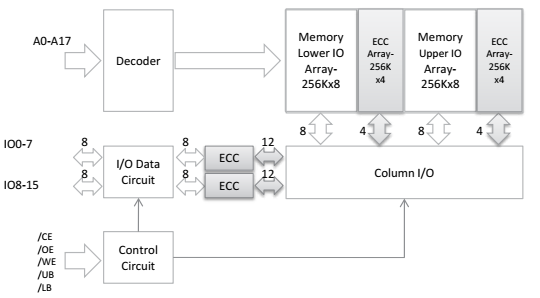


Figura : Diagrama de bloques de la memoria SRAM

En la tabla 10 se muestra la descripción de pines de la memoria y en la figura 56 el pinout del encapsulado.

|  |  |
| --- | --- |
| PIN NO. | Descripción |
| A0 - A17 | Address Input |
| I/O0 - I/O15 | Data Input/Output |
| CE | Chip enable input (activo en valor bajo) |
| OE | Output enable input (activo en valor bajo) |
| WE | Write enable input (activo en valor bajo) |
| LB | Lower-byte control (activo en valor bajo) |
| UB | Upper-byte control (activo en valor bajo) |
| NC | No conectar |
| VDD | Alimentación |
| GBD | Masa |

Tabla : Descripción de pines de la memoria SRAM

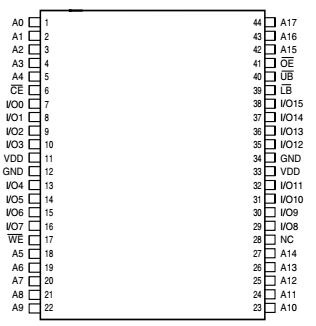


Figura : pinout de la memoria SRAM

Los pines de control “LB” y “UB” no los usaremos y los fijaremos a masa. El de Output enable “OE” tampoco es 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 VDD máxima(mA) |
| Not Selected | X | H | High-Z | ISB=15 |
| Read | H | L | DOUT | ICC=50 |
| Write | L | L | DIN | ICC=50 |

Tabla : Tabla de verdad de la memoria SRAM

Dicho esto, el esquemático queda como en la figura 57 donde sólo se ha puesto las masas, nombres a las líneas y añadido condensadores de desacoplo en las entradas de alimentación.

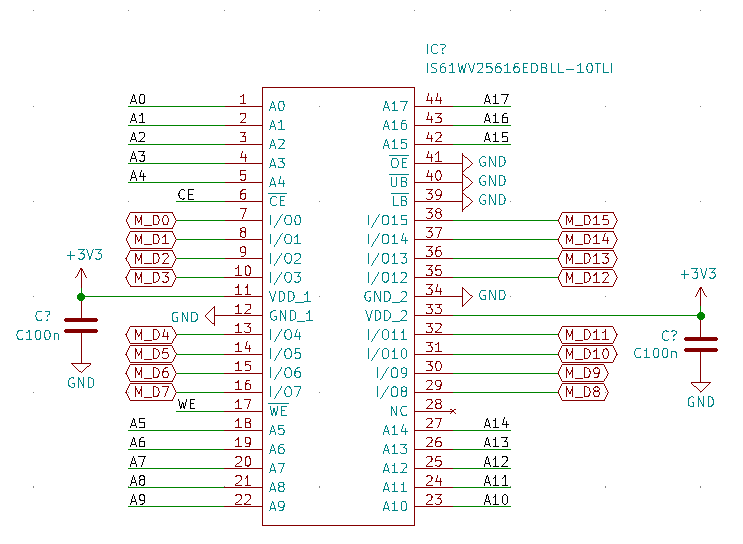


Figura : Esquemático de la memoria SRAM en KiCad

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 [63].

La entrada de un circuito CMOS es un inversor como el de la figura 58 que desacopla el circuito interno de la señal externa.

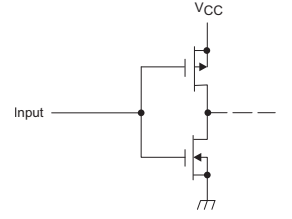


Figura : Etapa de entrada de un circuito CMOS

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 59) como en nuestro caso, las líneas son bidireccionales y se debe prestar especial atención cuándo todas las salidas están inactivas. El 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.

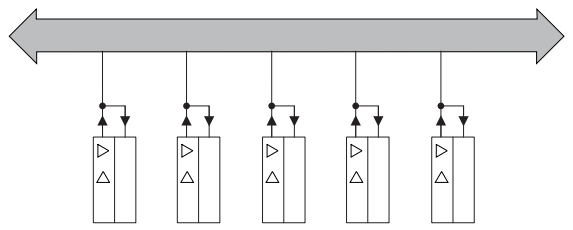


Figura : Transmisión bidireccional en un sistema de buses

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. 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 3:

(3)

Donde tr es el tiempo de subida necesario, Cs 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 Cs 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.

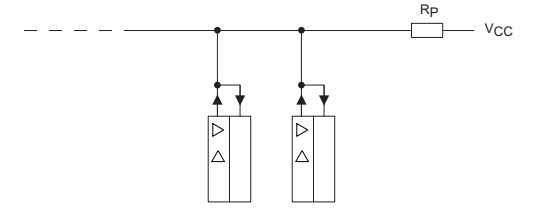


Figura : Resistencia de pull-up en una línea de bus

* Otra alternativa es un circuito de retención de buses como el de la figura 61. 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.

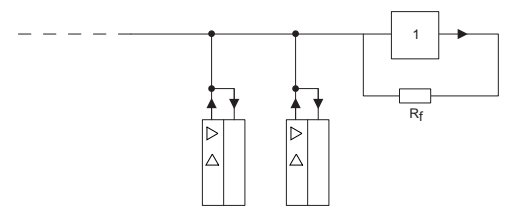


Figura : Circuito de retención de buses

* Por último, y basándose en la solución anterior, existen circuitos integrados de retención de buses. Para nuestra placa utilizaremos el SN74ALVCH16827 [64] 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 puede ver en la figura 62. 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 63 y su tabla de verdad en la tabla 12.

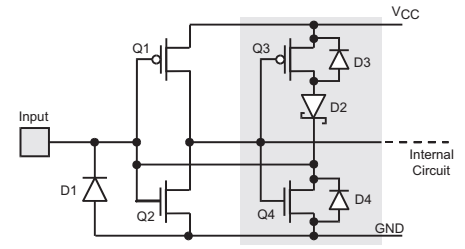


Figura : Circuito simplificado del integrado de retención de buses

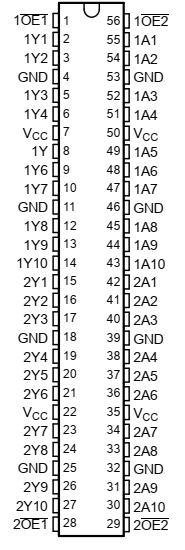


Figura : Pinout del SN74ALVCH162827

|  |  |  |  |
| --- | --- | --- | --- |
| OE1 | OE2 | A | Y |
| L | L | L | L |
| L | L | H | H |
| H | X | X | Z |
| Z | H | X | Z |

Tabla : Tabla de verdad del integrado de retención de buses

## 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.

### Análisis de consumo

En la tabla 10 se muestran los principales componentes que se alimentan a +3.3V, así como sus máximos consumos de corriente permitidos. Esto no quiere decir 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 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) |
| FT2232H | Ireg | Corriente del regulador | 150 |
|  | Iccphy | Corriente PHY en operación | 60 |
| DSC1001 | IDD | Corriente de alimentación | 6 |
| 93LC56C | ICC write | Corriente al escribir | 2 |
| STM32F091VC | IVDD | Corriente máxima en el total de pines VDD | 120 |
| iCE40HX4K | ICCIOPEAK | Corriente de pico en arranque para todos los bancos IO | 6.8mA\*8=54.5 |
|  | ISPI\_PEAK | Corriente de pico en arranque para el banco SPI | 6.8 |
| W25132JV | ICC | Corriente en escritura | 25 |
| ADS7924 | IDVDD | Corriente en DVDD | 0.01 |
|  | IAVDD | Corriente en AVDD | 0.08 |
| NCP114 | IGND | Corriente en Vin y EN para una salida de 0.1mA | 0.1+0.1=  0.2 |
| IS62WV51216BLL | ICC | Corriente en operación | 30 |
| SN74ALVCH162836 | ICC | Corriente en operación | 0.75 |
| **TOTAL** | | | 455.34 |

Tabla : 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.

Además, la FPGA debe alimentarse a +1.2V con un consumo 22.3mA por cada VCC, 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.

Además, interesa sacar varios pines de alimentación tanto de 5V como de 3.3V al ser dos valores muy utilizados en distintos módulos y sensores.

### 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 55) 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.

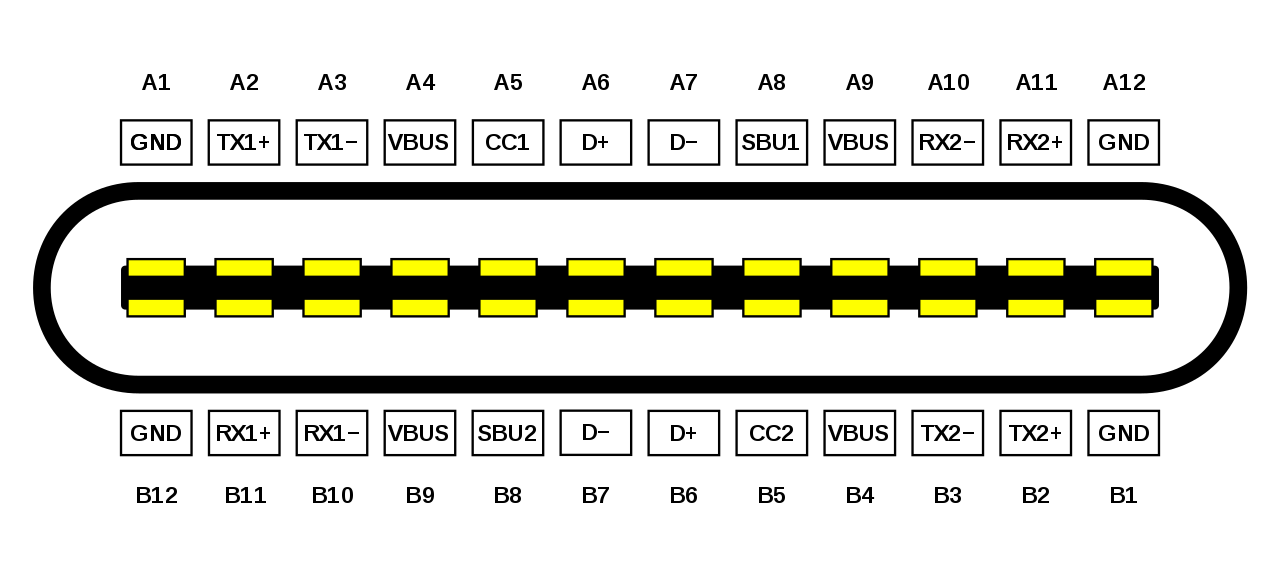


Figura : Pinout USB tipo C [62]

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 4.8A, sin embargo, esta con un solo conector admitirá 5V con una corriente de 5A.

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, VBUS y D+ y D-. Utilizaremos el conector de 16 pines DX07S016JA1R1500 [63] cuyo pinout podemos ver en la tabla 11.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pad NO. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| PIN NO. | A1/  B12 | A4/  B9 | A5 | B8 | B7 | A6 | B6 | A7 | B5 | A8 | A9/  B4 | A12/  B1 |
| Uso | GND | VBUS | NC | NC | D- | D+ | D+ | D- | NC | NC | VBUS | GND |

Tabla : Pinout del conector DX07S016JA1R1500

Los pines del 13 al 17 son de protección y van conectados a masa. En la figura 56 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 VBUS 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 para las líneas de datos USB. En la figura 57 podemos ver 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.

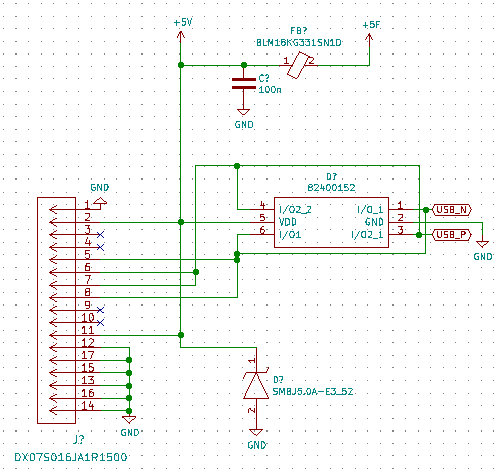


Figura : Esquemático del conector USB C en KiCad

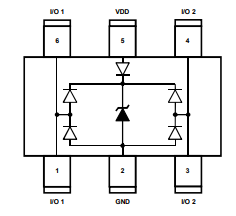


Figura : Interior del circuito de protección ESD para USB [64]

### Regulador de tensión

Como ya se comentó en el apartado *Análisis de consumo,* 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.* 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 transformar los 5V de entrada a los 3.3V y 1.2V. Para ello utilizaremos el PAM2306 [65], un conversor DC-DC de dos canales, con eficiencia de hasta 96% y corriente de salida de 1A por canal. Podemos ver el pinout del dispositivo en la figura 58, su diagrama de bloques para un canal en la figura 59 y una implementación típica en la figura 60.

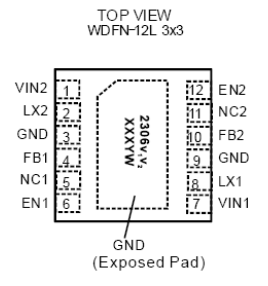


Figura : Pinout del PAM2306

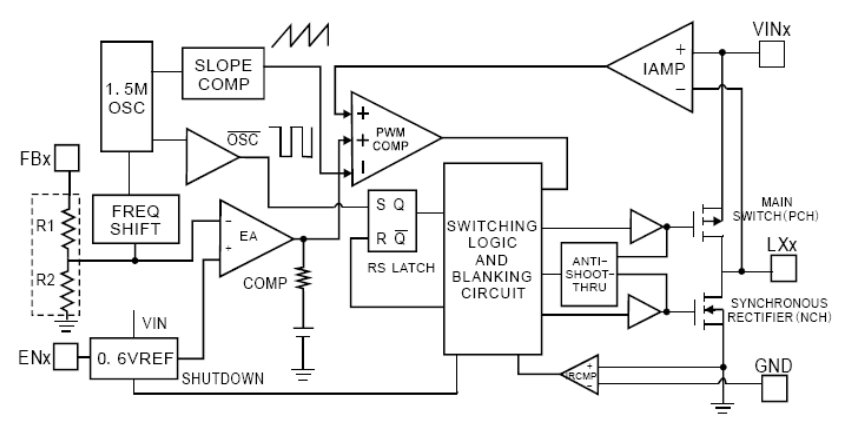


Figura : Diagrama de bloques PAM2306

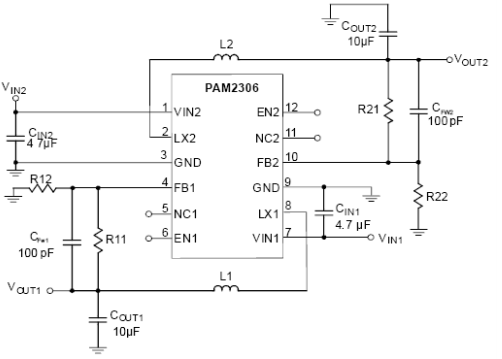


Figura : Implementación típica PAM2306

El valor de las inductancias decidirá el rizado presente en la corriente de salida según la ecuación 1:

(1)

Optaremos por una inductancia de valor típico L=4.7µH para ambos canales, así como con una baja resistencia DC para mayor eficiencia.

Para CIN y COUT optaremos por condensadores de 10uF 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 2:

(2)

Por lo tanto, para el canal que necesitamos 3.3V optaremos por R1=45kΩ y R2=10kΩ. Para 1.2V optaremos por R1=10kΩ y R2=1m0kΩ.

En la figura 61 podemos observar el esquemático final del convertidor. Como se indicó antes, la entrada VIN del convertidor es la línea filtrada +5VF además de desacoplarlo con condensadores de 10uF. Los pines EN se han puesto también a +5V ya que se activan en valor alto. Por último, comentar que se ha añadido un diodo Zener en la tensión de +3.3V para fijar la tensión ya que muchos dispositivos se conectarán a esa línea de tensión.

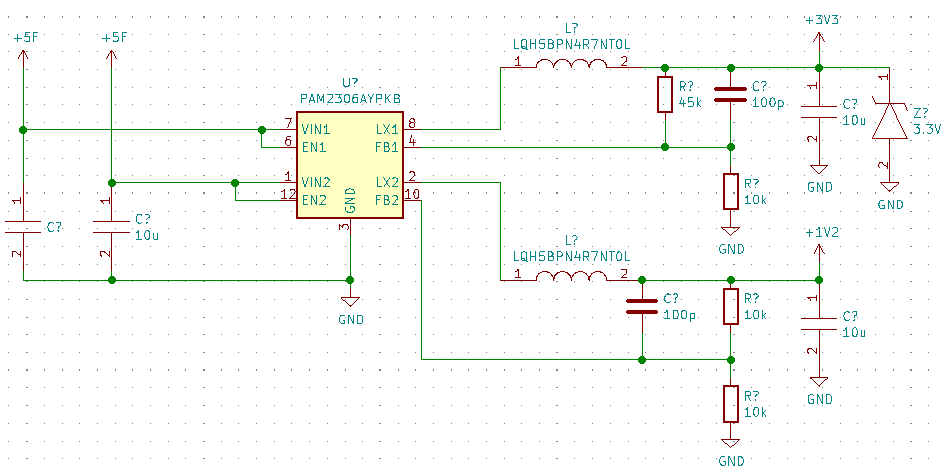


Figura : Esquemático del PAM2306 en KiCad

Además, 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 ha implementado un interruptor que permite activar o desactivar los niveles de tensión para los periféricos que se conecten a la placa como podemos ver en la figura 72. En el podemos apreciar un switch a la izquierda que permitirá al transistor entrar en conducción o corte y así controlar la tensión de +5VP. Dicha tensión esta filtrada mediante un condensador de aluminio de 220µF 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 [69] que permite proporcionar hasta 1A en la salida y consumiendo en reposo solamente 200µA. El esquemático típico del dispositivo lo encontramos en la figura 72. A la entada 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 220µF 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.

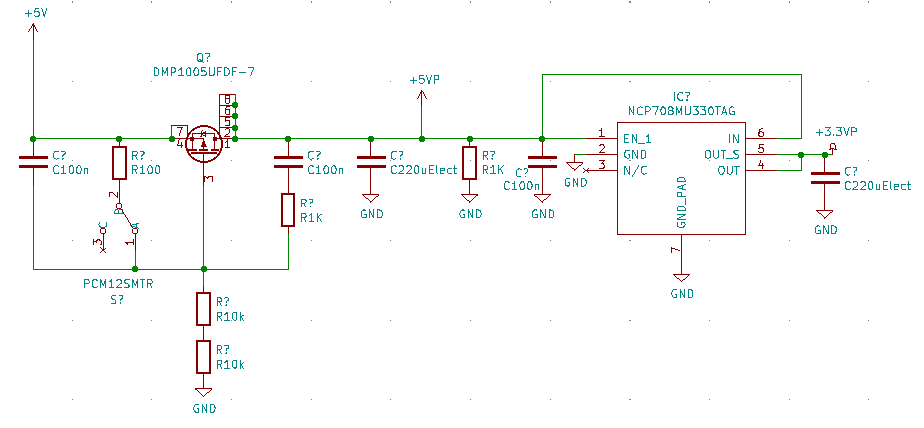


Figura : Interruptor de la tensión para periféricos

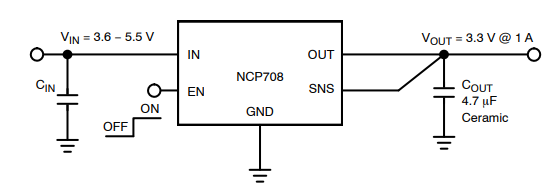


Figura : Aplicación tipica del LDO

Por último, 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 externos están activados. Podemos ver sus esquemáticos en la figura 73.

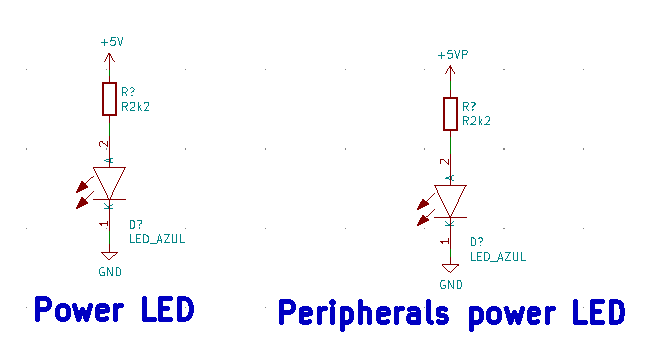


Figura : LEDs indicadores de alimentación

## Placa de circuito impreso

## Producto final

### Especificaciones

### Pinout

### BOM

# Conclusiones y trabajo futuro

# Bibliografía