UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Entrenador para FPGAs con microcontrolador

Autor:

D. Julio Díez Tomillo

Tutor:

D. Jesús Manuel Hernández Mangas

VALLADOLID, JUNIO 2020

TRABAJO FIN DE GRADO

TÍTULO:	**Entrenador para FPGAs con microcontrolador.**
AUTOR:	**D. Julio Díez Tomillo**
TUTOR:	**Dr. D. Jesús Manuel Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

TRIBUNAL

PRESIDENTE:	**Jesús Arias**
VOCAL:	**Jesús M. Hernández**
SECRETARIO:	**Ruth Pinacho**
SUPLENTE:	**Luis Marqués**
SUPLENTE:	**Pedro López**

RESUMEN

Las FPGAs (Field-Programmable Gate Array) son dispositivos programables cada vez más usados en el ámbito de la electrónica debido a su gran versatilidad, su capacidad de ser reprogramadas y su bajo coste. En este proyecto se pretende diseñar una placa de circuito impreso de bajo coste basada en una FPGA del fabricante Lattice controlada mediante un microcontrolador LPC1114. La placa está destinada principalmente al ámbito educativo para poder aprender y desarrollar con ella. Se puede usar para aprender nociones de programación de diferentes lenguajes, desde C hasta lenguajes de descripción de hardware (HDL) como Verilog. También se puede usar para iniciarse o profundizar en los circuitos electrónicos digitales y los protocolos de comunicación como, por ejemplo, SPI. Además, a la placa se la puede conectar una pantalla TFT, mediante los conectores ya integrados, para usar la FPGA como un controlador de vídeo.

PALABRAS CLAVE

- *FPGA*
- *Microcontrolador*
- *Placa de Circuito Impreso*
- *ICEStudio*
- *SPI*
- *Software Libre*
- *Educativo*

ABSTRACT

FPGAs (Field-Programmable Gate Array) are programmable devices increasingly used in the field of Electronics due to its versatility, reprogram capability and low cost. This project aims to develop a low-cost printed circuit board (PCB) based on a Lattice FPGA controlled by a LPC1114 microcontroller. The board is mainly intended to provide a learning environment for developing. It performs well as a learning platform for different programming languages, from C to hardware description languages (HDL) such as Verilog. Furthermore, it can be used to learn both basic or advanced digital electronic circuits and communication protocols including SPI. Finally, a TFT display can be attached to the board using the integrated connectors to use the FPGA as a video controller.

KEYWORDS

- *FPGA*
- *Microcontroller*
- *Printed Circuit Board (PCB)*
- *ICEStudio*
- *SPI*
- *Open Source Software*
- *Educational*

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi tutor Jesús porque, a pesar de haber tenido que hacer la mayoría de las comunicaciones de forma telemática, me lograba dar todas las explicaciones necesarias a través del correo electrónico. También por haberme propuesto la realización de este TFG y por todas sus explicaciones, consejos y sugerencias que me fue haciendo durante el proceso que me ayudaron a terminar el proyecto.

En segundo lugar, a mi familia, mis padres Elisa y Carlos, y mis hermanos Patricia y Javier por su apoyo y por haberme aguantado mientras realizaba el proyecto.

También a mis compañeros de clase y profesores de la carrera que me han ayudado directa o indirectamente en el proyecto, habiéndome dado conocimientos necesarios o por haberme ayudado o escuchado mientras les contaba de qué iba el proyecto.

Por último, agradecer a Obi Juan, por haber realizado numerosos cursos sobre FPGAs y dejar a disposición de todos los usuarios las colecciones de ICEStudio que han servido para culminar el software de este proyecto.

A todos ellos, muchas gracias.

Con buenas dosis de ingenio,
cualquiera puede llevar a cabo el
proyecto que se proponga

Giovanni Papini

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO I

INTRODUCCIÓN

En este primer capítulo vamos a efectuar una pequeña introducción al proyecto para que sea más fácil la lectura del resto del documento y su compresión. Resolveremos diferentes preguntas como el porqué de realizarlo, qué objetivos pretendemos conseguir con su realización y en qué consiste. También veremos cómo se divide el presente documento para facilitar su lectura y que el lector tenga una visión general del proyecto antes de profundizar en el mismo.

Motivación

Las FPGA (*Field-programmable gate array*) están aumentando notablemente su presencia en el mercado debido a la capacidad de ser reprogramables, su reducido precio, una gran flexibilidad para diseñar y el aumento progresivo de su velocidad. Por ello es recomendable que, de cara al futuro, las FPGA sean estudiadas más a fondo en los colegios y en las universidades, además de su facilidad para ser programadas y la amplia variedad de posibilidades que nos ofrecen para implementar circuitos hardware de una manera muy sencilla.

Por esta razón principal surge este proyecto, para el diseño de una placa de pruebas que pueda ser usada en las clases. Con esto conseguiríamos enseñar de forma más entretenida y práctica las FPGAs y su funcionamiento, además de poder implementar circuitos lógicos hardware combinacionales y secuenciales, desde los más sencillos hasta circuitos complejos. Además, la FPGA seleccionada se puede programar mediante un software abierto y gráfico, mediante bloques, muy sencillo de usar basado en *Verilog* (un lenguaje de descripción de hardware). Así reduciríamos la complejidad de la programación y que se pueda utilizar esta placa para niveles y cursos muy diversos. Por ejemplo, para cursos muy básicos sin experiencia se pueden usar los bloques hechos y que la única tarea sea interconexionarlos para producir circuitos. Y, por otro lado, a cursos y niveles más altos, se les puede pedir que ellos mismos diseñen algunos bloques propios usando *Verilog*.

Lo que tiene de especial este proyecto frente a otras placas de desarrollo con FPGAs, como el iCEStick [1] o la IceZum Alhambra [2], es que en nuestro proyecto la FPGA se programa mediante un microcontrolador en vez de directamente mediante el software, como ocurre con las dos placas anteriores. Por ello, esto nos permite también poder diferenciar varios niveles para el uso de esta placa; en los cursos inferiores se puede dar el software hecho y que solo sea ejecutarlo para programar la FPGA, mientras que en cursos superiores se puede pedir escribir el protocolo mediante el cual se programará y configurará la FPGA y el microcontrolador.

Además, en la placa se han instalado los componentes necesarios para montar una pantalla TFT compatible con los conectores y poder usar la FPGA como controlador de video para

mostrar cosas por pantalla. También tenemos cuatro LEDs conectados directamente a la FPGA que nos permiten hacer programas muy sencillos para iluminarlos. Y, por último, el microcontrolador también se puede programar y usar sus pines para realizar otras tareas, aparte de poder utilizar la FPGA.

Por ello, la principal motivación de este proyecto fue querer crear una placa polivalente para las aulas que permita aprender sobre las FPGAs y su programación de forma práctica, entretenida y amena, con la capacidad de ser usada en muchos niveles. Desde el más básico con poco conocimiento, como podrían ser últimos cursos de secundaria hasta niveles muy altos, incluidos cursos universitarios.

Objetivos

Con la creación de este proyecto hemos pretendido cumplir una serie de objetivos, algunos de los cuales ya han sido explicados en el apartado anterior sobre la motivación del proyecto.

El primer y principal objetivo del proyecto ha sido crear una placa de desarrollo mediante FPGA para ser utilizado en el ámbito de la educación y la enseñanza. Con la capacidad de ser usado en numerosos niveles debido a la sencillez del programa de código abierto usado para la programación de esta. También queríamos que esta placa pudiera tener un componente visual, por lo que se han instalado en la placa los conectores necesarios para conectar a la placa una pantalla TFT controlada mediante la FPGA.

El segundo objetivo ha sido crear una placa de desarrollo hardware de bajo coste. Las placas existentes en el mercado suelen ser de un precio alto, por ejemplo, la iCEZum Alhambra tiene un coste de 50€ o el iCEStick con un precio de 30€ pero con unas posibilidades muy reducidas debido a los pocos pines de salida que posee.

Otro objetivo del proyecto ha sido ver las posibilidades que ofrecen las FPGAs para realizar tareas varias, desde lo más sencillo como es iluminar unos LEDs hasta ser capaz de usarse como un controlador de video. Además de explorar la posibilidad de que sea programada mediante un software de código abierto como ICEStudio [3] que permite una programación sencilla mediante bloques.

El último objetivo del proyecto, y que nos diferencia de otros existentes ha sido investigar y desarrollar la placa para que la FPGA no se programe directamente, sino que lo haga a través de un microcontrolador, que escriba en las memorias el programa y que active la FPGA para que se programe. Esto nos abre la posibilidad, además, de poder usar también el microcontrolador, y no solo la FPGA, ya que también existe un conector con pines provenientes del microcontrolador, por lo que también puede ser utilizado y programado y que no se use únicamente para programar la FPGA.

Por todo ello podemos concluir que el principal objetivo del proyecto ha sido crear una placa de desarrollo hardware destinada a la educación, que nos permita programar la FPGA mediante un microcontrolador, a la vez que intentamos que sea una placa de coste reducido.

Organización del documento

El documento está organizado en seis capítulos diferentes, además de contener al final de este siete anexos. En páginas anteriores, además, encontramos un resumen junto con unas palabras clave en inglés como en español. También podemos consultar el índice para encontrar alguna sección concreta, además de la lista de figuras que aparecen a lo largo del presente documento.

En el primer capítulo, en el que nos encontramos, hemos realizado una visión general de en qué consiste el Trabajo de Fin de Grado junto con la motivación de este, los objetivos que se pretendían alcanzar y su planificación. Además, en este mismo apartado estamos mostrando como se organiza el documento.

En el segundo capítulo explicaremos ligeramente algunos conocimientos previos necesarios y útiles para la lectura posterior del resto del documento.

En el capítulo tres entraremos a explicar más a fondo el hardware del proyecto. Explicando primero los componentes principales que hemos utilizado y por qué hemos elegido esos. Analizaremos los esquemáticos y la PCB, junto con el consumo eléctrico y la disipación térmica. Y finalmente hablaremos brevemente sobre otros añadidos a la placa, como la caja protectora.

En el cuarto capítulo hablaremos sobre el software necesario para que funcione correctamente el circuito. También hablaremos sobre el programa de código abierto utilizado para programar la FPGA (ICEStudio) así como el procedimiento para hacerlo.

En el quinto capítulo realizaremos un breve estudio económico del proyecto. Podremos ver el presupuesto realizado, así como el cambio de precio en función de las unidades producidas.

Por último, en el sexto capítulo, podremos leer unas pequeñas conclusiones acerca del proyecto y todo el trabajo realizado, así como futuras mejoras que se podrían realizar en el proyecto, es decir, como se podría continuar éste en el futuro.

Finalmente, encontramos la bibliografía, en la cual podemos consultar las referencias que han ido apareciendo a lo largo del documento y también los siete anexos, los cuales se irán comentando y explicando a lo largo del documento, ya que son un apoyo y un añadido a las explicaciones realizadas a lo largo de los seis capítulos.

Vista general del proyecto

En este apartado vamos a ver una visión general del proyecto, para entender su funcionamiento general y como se interconexionan los componentes principales. En la **figura 1** tenemos un diagrama de bloques muy sencillo de la placa. En esa figura podemos ver que la alimentación y programación se realiza a través del ordenador, mediante el puerto USB. Después mediante un convertidor USB-UART, programamos el microcontrolador y podemos comunicarnos con él. El microcontrolador puede escribir y leer de las tres memorias Flash ubicadas en la placa, mediante el protocolo SPI. También tiene unos pines de salida que pueden ser usados y configurados. La labor principal del microcontrolador es escribir el programa de la FPGA en una de las memorias y activarla

utilizando correctamente el pin *Reset* que va directamente a la FPGA. Esta tiene también acceso a las tres memorias Flash, y se configura cargando el programa que deberá contener la memoria número 3 (está configurado de esa manera). Al acabar de configurarse activará el pin *Done* notificando al microcontrolador que se ha configurado correctamente y está en funcionamiento. Por último, la FPGA también tiene una tira de pines configurables para poder ser usados, además de estar conectado a 4 LEDs que pueden servir de prueba para comprobar que se ha programado correctamente.

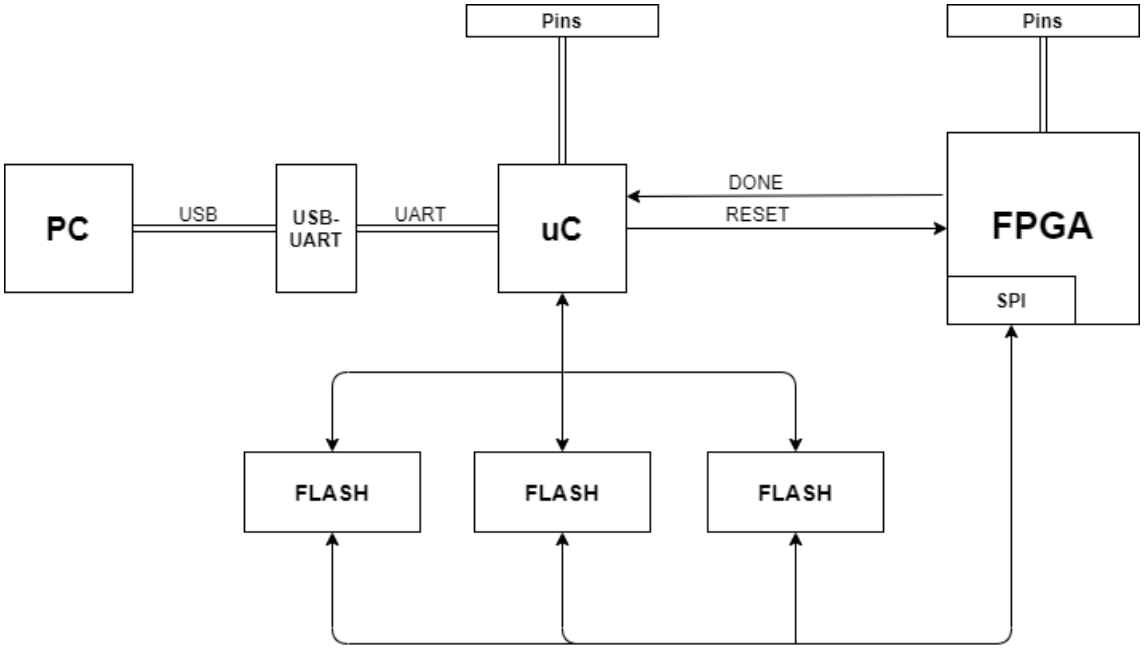


Figura 1.- Diagrama de bloques del proyecto

Aparte de los elementos que podemos ver dentro del diagrama de bloques de la **figura 1** hay muchos otros de los que se ha prescindido para esta pequeña visión general del proyecto, ya que en sí su labor es únicamente lograr que funcionen las partes principales, como por ejemplo los reguladores lineales, que nos permiten alimentar los diferentes elementos al voltaje adecuado.

Otros elementos que no se ha incluido en el diagrama de bloques son los conectores de la pantalla TFT y la propia pantalla. Esto se debe a que en este proyecto no se ha llegado a trabajar con ella, ya que el principal objetivo era la creación de la placa y su correcta configuración. En la placa se han incluido los conectores necesarios para poder conectar la pantalla y que funcione correctamente, pero al no haber sido utilizada ni probada, se ha prescindido de ella en el diagrama de bloques, aunque se podría ver dentro de los pines que salen de la FPGA, ya que es ésta la que tiene la labor de controlador de video para la pantalla.

A lo largo de la lectura de este documento no hay que perder de vista este diagrama de bloques, que nos facilitará notablemente el entender el funcionamiento del sistema y los pasos y decisiones que se han ido tomando durante el desarrollo del proyecto. En el documento iremos viendo más en profundidad cada una de las partes, cómo se comunican entre ellas y el método de configuración y programación de la FPGA y el microcontrolador. En cualquier momento se puede volver a observar el anterior diagrama

de bloques para no perder la visión general del proyecto a medida que profundizamos en cada una de las partes.

Planificación del Proyecto

En el **anexo 1** se puede ver el diagrama Gantt de este proyecto. En él se recoge la planificación, con las diferentes tareas que se han ido haciendo y los plazos para cada una de ellas [4]. A lo largo de su desarrollo, este diagrama se ha ido variando, ya que algunas tareas se han acortado y para otras se ha tenido que alargar los plazos, pues no dio tiempo a completarlo en el plazo fijado inicialmente.

En este diagrama Gantt se recogen seis bloques importantes. El primer bloque es el de planificación inicial, en esta tarea se diseñó un diagrama de Gantt inicial para el desarrollo del proyecto, fijando los plazos y creando las tareas que iba a haber. El segundo bloque corresponde a las tareas relacionadas con la captura esquemática, que dura aproximadamente un mes. El tercer bloque corresponde a las tareas del diseño de la PCB, con todo lo que implica, y se determinaron tres semanas de trabajo para terminarlo.

Después tenemos otro bloque correspondiente a la fabricación y el montaje. Este bloque tiene una duración más larga de la necesaria ya que había que incluir en el plazo las dos semanas que tarda el pedido de la placa en llegar. Durante este periodo se estuvo trabajando en el diseño del software para el proyecto. Una vez terminado el montaje, se pasó al quinto bloque, que corresponde al testeo Hardware y Software, es decir, comprobar que la placa funciona correctamente a nivel hardware y que el firmware creado también.

Una vez se comprobó que el funcionamiento del proyecto se procedió a realizar el último bloque de la planificación. Este bloque, que es el más largo, consiste en la realización de un presupuesto económico del proyecto y de esta memoria. En este bloque, aunque la parte más importante se realizó al finalizar la placa, se estuvo trabajando desde el inicio del proyecto.

CAPÍTULO II

TEORÍA PREVIA

En este segundo capítulo vamos a analizar brevemente la situación del mercado actual de FPGAs, así como sus previsiones futuras. También explicaremos el funcionamiento de los dos protocolos de comunicación usados en nuestra placa que son el SPI y la UART. Este capítulo es muy útil para poder realizar una buena lectura de esta memoria y entender el funcionamiento de nuestra placa, además de poder conocer en profundidad las FPGAs y su mercado actual.

FPGAs en la Actualidad

Una FPGA (*Field-programmable gate array*) es un circuito integrado que puede ser configurado por el usuario para implementar un sistema hardware concreto. Está formado por una matriz de bloques lógicos (LAB) interconectados entre sí. La programación se realiza mediante lenguajes de descripción de hardware (HDL), como puede ser por ejemplo Verilog. [5]

Actualmente, el valor del mercado de las FPGAs está estimado en 9.800 millones de dólares [6]. Y se calcula que para el año 2026 suba hasta un valor de 13.000 millones de dólares, lo que lo convierte en un mercado al alza en el mundo de electrónica [7]. Dentro de este mercado encontramos a tres empresas que tienen la mayor parte de las ventas en el sector, como podemos ver en la gráfica de la **figura 2.**

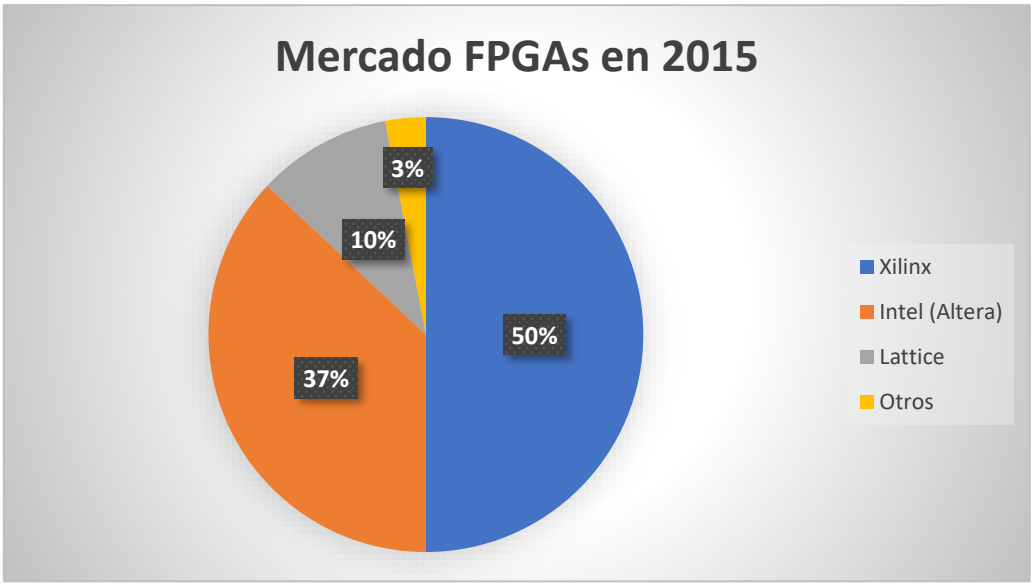


Figura 2.- Reparto del mercado de FPGAs en 2015 por su fabricante[1]

[1] Fuente: IHS Markit (2015) [6]

Según los datos de 2015, la compañía Xilinx posee la mitad del mercado de las FPGAs, siendo la empresa que más ventas realiza. Después nos encontramos con Intel, con su marca de FPGAs llamada Altera, que posee el 37% de las ventas de FPGAs. Luego tenemos la empresa Lattice, fabricante de la FPGA usada en este proyecto. Sus ventas suponen un 10% del mercado, pero es una marca al alza ya que datos anteriores ofrecían que solo vendía el 6%. Finalmente nos encontramos que otras empresas como MicroSemi o Atmel poseen solo el 3% de las ventas de FPGAs

La razón de este incremento de ventas de FPGAs se debe principalmente a la reducción progresiva de su coste, y a sus mejoras técnicas frente a sus competidores como las ASIC (*Application-specific integrated circuit*) o las CPLD (*Complex Programmable Logic Devices*). El principal problema de las ASIC es su alto precio y su limitada versatilidad, ya que no son programables porque son diseños hardware específicos para una aplicación concreta, frente a las FPGAs que podemos implementar todos los circuitos que deseemos, programando cada vez uno diferente. Además, las FPGA tienen un alto rendimiento, y pueden integrar muchas funciones como bloques de memoria o multiplicadores. [8]

Dada la gran versatilidad de las FPGAs se encuentran muy extendidas dentro del mundo de la electrónica y se dedican a múltiples aplicaciones. En la **figura 3**, podemos ver que las FPGAs se dedican principalmente para el mercado de las Telecomunicaciones, aplicaciones industriales y automovilísticas. También se usan en el sector militar y aeroespacial, aunque en menor medida debido a que se suelen utilizar habitualmente ASIC ya que suelen ser funciones muy específicas, por lo que la principal ventaja de las FPGAs, que es su capacidad de reprogramarse, no es útil. [9]

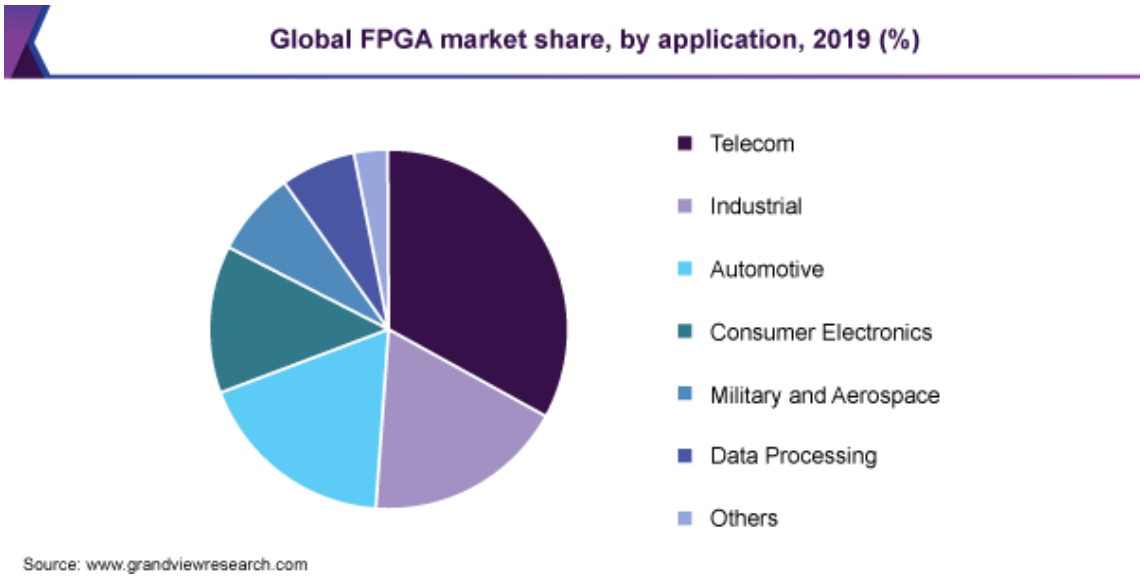


Figura 3.- Reparto del mercado de FPGAs por su aplicación

Como hemos visto, las FPGAs están actualmente al alza y cada vez se venden más y se dedican a más sectores. Por ello es imprescindible su aprendizaje y conocimiento en las aulas, ya que seguramente sea uno de los componentes fundamentales en el futuro de la electrónica.

Protocolo UART

En este nuevo apartado vamos a explicar el protocolo que usamos para la comunicaciones entre la interfaz USB-UART y nuestro microcontrolador. Lo primero que cabe destacar es la necesidad que tenemos de esa interfaz. Nuestro ordenador se conecta a la placa mediante un cable Micro-USB, y envía los datos a través del protocolo USB, mientras que nuestro microcontrolador se conecta para su programación mediante el protocolo UART. Por ello, nos hace falta esta interfaz que realiza la conversión de los datos provenientes de nuestro ordenador mediante el protocolo USB al protocolo UART para poder transmitirlos a nuestro microcontrolador. También realiza la conversión en sentido contrario, permitiendo realizar una comunicación bidireccional. [10]

El protocolo UART se corresponde con las siglas *Universal Asynchronous Receiver/Transmitter*. Este protocolo implementa una comunicación serie y asíncrona. Para realizar la comunicación solamente hacen falta dos pines: TXD (transmisor) y RXD (receptor) y solo permite conectar dos dispositivos para que se comuniquen entre ellos. La conexión de estos pines sigue la técnica "Boca-Oreja", es decir, el transmisor de uno se conecta al receptor del otro y viceversa, como podemos ver en la **figura 4**.

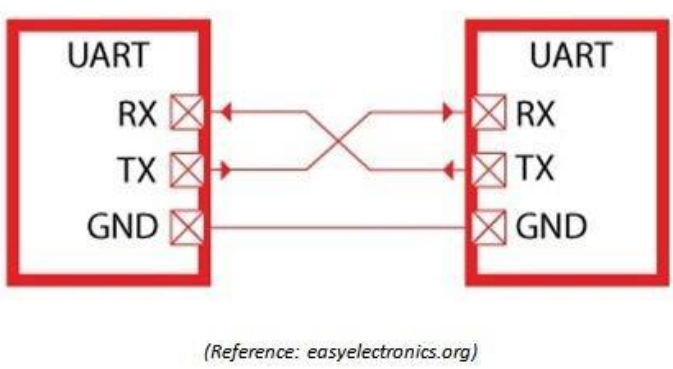


Figura 4.- Esquema de conexión del protocolo UART

El protocolo utiliza para comunicarse una trama configurable. La trama más básica consiste en 8 bits de datos junto con uno de parada y otro de inicio. A mayores se puede añadir un bit de paridad para poder corregir y detectar errores en la comunicación. Además, antes de iniciar la comunicación, ambos dispositivos tienen que decidir una velocidad de comunicación dada en baudios, siendo lo más común 115200 baudios. En la **figura 5** podemos ver un ejemplo de trama, cuando no tenemos bit de paridad y se transmiten 8 bits con un bit de inicio y otro de parada.

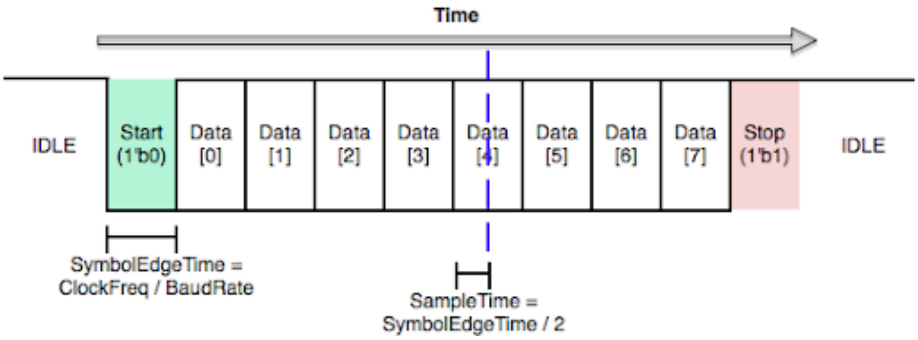


Figura 5.- Ejemplo de trama del protocolo UART

Protocolo SPI

El protocolo que más usamos en nuestra placa es el protocolo SPI (*Serial Peripherial Interface*), ya que con él comunicamos la FPGA y el microcontrolador con las memorias. Es un protocolo que funciona de forma síncrona y en modo *full-duplex*, es decir, puede transmitir y recibir datos simultáneamente. En este protocolo se diferencian dos tipos de componentes. Tenemos el maestro, que es el que selecciona al resto de componentes y transmite la señal de reloj. Luego tenemos al esclavo, que cuando es activado puede transmitir la información pedida por el maestro. Para comunicarnos con este protocolo necesitamos como mínimo 4 pines. El primero que necesitamos es SCLK, que es el reloj con el que se transmiten los datos entre el maestro y el esclavo. Después tenemos los pines a través de los cuales se transmite información. El MISO (*Master In Slave Out*), es la entrada de datos al maestro, es decir, lo que transmite el esclavo. Y MOSI (*Master Out Slave In*), que es lo contrario, la salida de datos del maestro. Por último, tenemos el pin de selección (/SS), activo en baja. Este pin es con el que activamos el componente con el que queremos comunicarnos. Si queremos comunicarnos con más de un componente esclavo, necesitaremos un pin de selección por cada uno de ellos. Pero los pines de reloj y de datos serán comunes a todos ellos, como podemos ver en la **figura 6**, que representa un ejemplo de conexión de un componente maestro con tres esclavos. [11]

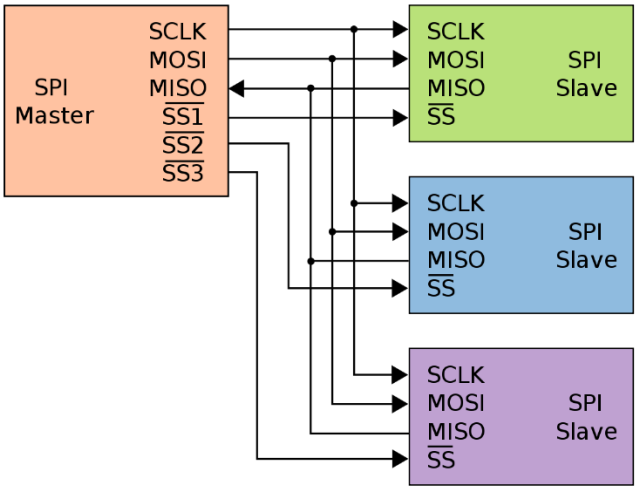


Figura 6.- Esquema de conexión de un dispositivo maestro con 3 esclavos mediante SPI

El proceso de comunicación es muy sencillo como podemos ver en la **figura 7**, que representa una comunicación de datos mediante SPI. Lo primero que hace el maestro es seleccionar y habilitar el componente esclavo con el que queremos comunicarnos poniendo en baja el pin de selección correspondiente. Una vez seleccionado el componente, se procede al intercambio de bytes entre el maestro y el esclavo mediante los buses MISO y MOSI. La comunicación habitualmente se realiza en ráfagas de un byte como máximo. Una vez terminada la comunicación, el maestro tiene que volver a poner en alta el pin de selección, para deshabilitar el esclavo. Aunque en esta figura no se represente, también puede ocurrir que solo uno de los dos esté transmitiendo datos y el otro solo escuche.

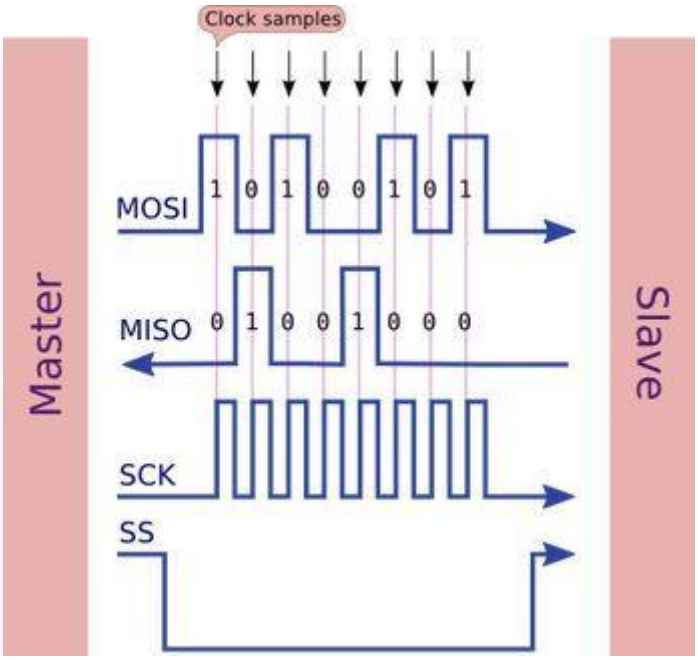


Figura 7.- Ejemplo de comunicación entre dos dispositivos mediante el protocolo SPI

Este protocolo es muy usado actualmente en la electrónica, en gran parte por la alta velocidad de transmisión que se pueda alcanzar y también a los pocos pines que hacen falta para realizar la comunicación. Es verdad que si solo queremos comunicarnos con un componente hacen falta 4 pines, pero si queremos comunicarnos con más, solo hace falta añadir un pin nuevo de selección, ya que los buses de MISO, MOSI y el reloj son compartidos por todos los componentes.

CAPÍTULO III

DISEÑO Y ANÁLISIS HARDWARE

En este tercer capítulo vamos a exponer la parte del proyecto correspondiente al hardware. Aquí trataremos primero los componentes electrónicos utilizados en el circuito impreso. En segundo lugar, comentaremos el análisis térmico realizado del circuito, junto con un análisis de consumo de potencia. Además, analizaremos los esquemáticos del proyecto realizados con el programa Proteus.

Posteriormente pasaremos a describir y mostrar el PCB que se ha fabricado y cómo fue su proceso de diseño. Por último, comentaremos brevemente el modelo 3D realizado también con el programa Proteus del circuito impreso y mostraremos el proceso de creación de la caja imprimida mediante una impresora 3D.

El proceso de diseño y fabricación del hardware corresponde a la mayor parte de tiempo del proyecto, como podemos ver en el diagrama Gantt del **anexo 1**. Dentro de esta fase no solo se fabrica el hardware si no que hay que realizar un proceso de búsqueda de todos los componentes que van a ser utilizados en función de los requerimientos del sistema, así como pensar cómo va a ser su interconexión con el resto del sistema: que protocolos se van a usar, cuantos pines necesito para cada bloque, que alimentación utiliza cada elemento.

ESTUDIO DE LOS COMPONENTES

En este primera apartado del tercer capítulo vamos a profundizar sobre los elementos más importantes del proyecto. Cada uno de estos componentes se ha elegido siguiendo una serie de requisitos que debía cumplir cada elemento y se ha elegido el que mejor satisfacía las necesidades.

En alguna ocasión como se puede ver en el esquemático (**anexo 2**) y en el listado de materiales o BOM (**anexo 3**) puede haber alguna diferencia entre el componente elegido en el proveedor Farnell y plasmado en el diseño frente al componente realmente soldado e implementado en el prototipo del circuito impreso. Esto se debe a que, por abaratar costes, se ha primado el montar componentes ya disponibles en stock en vez de comprar componentes nuevos. Estos componentes montados siempre son totalmente compatibles con los elegidos para el diseño, exceptuando alguna ligera modificación, pero que no afecta al funcionamiento general del circuito ni a su programación.

Todos los componentes han sido buscados en la web de un proveedor de confianza, en nuestro caso Farnell. Al realizar cualquier proyecto de diseño hardware, es necesario tener un distribuidor de confianza para tener siempre el material para el montaje rápidamente y con un gasto no excesivamente alto. En el BOM (*Bill of Materials*) del **anexo 3** todas

las referencias de los productos están obtenidos del proveedor Farnell, previamente mencionado, cuya web es "https://es.farnell.com/".

A continuación, pasamos a explicar detalladamente cada uno de los componentes elegidos para el proyecto.

a) FPGA ICE40HX1K-TQ144

En primer lugar, tenemos el componente más importante de nuestro circuito impreso, la FPGA. Es de la marca Lattice, como hemos visto en capítulos anteriores, una de las grandes empresas de FPGAs. Esta FPGA en concreto pertenece a la familia de las iCE40 que están fabricadas mediante CMOS de 40nm, de ahí el nombre de esta familia. A continuación, vamos a ofrecer una pequeña descripción de la arquitectura de esta familia para después concretar en la FPGA elegida para el proyecto. [12]

La arquitectura de la familia iCE40 se basa en una matriz de bloques lógicos programables (PLB) rodeados de pines programables (PIO). Las celdas de los PIO están ubicadas en la periferia del dispositivo para poder salir al exterior y se encuentran distribuidos en 4 bancos diferentes, uno en cada lado del componente. Además, esta arquitectura posee dos PLLs para poder multiplicar, dividir y desplazar en fase los relojes, pudiendo variar su frecuencia y su fase. También posee un banco SPI para la programación del dispositivo, aunque también posea una *Non volatile Configuration Memory* (NVCM) para poder meter un programa que se ejecute siempre, sin que se borre al apagar el dispositivo. Por último, tiene varios bloques de memoria RAM dentro del dispositivo, conocidos como EBR (*sysMEM Embedded Block RAM*). En la siguiente **figura 8** podemos observar una visión general del interior de la FPGA, pudiendo apreciar todas las partes mencionadas anteriormente.

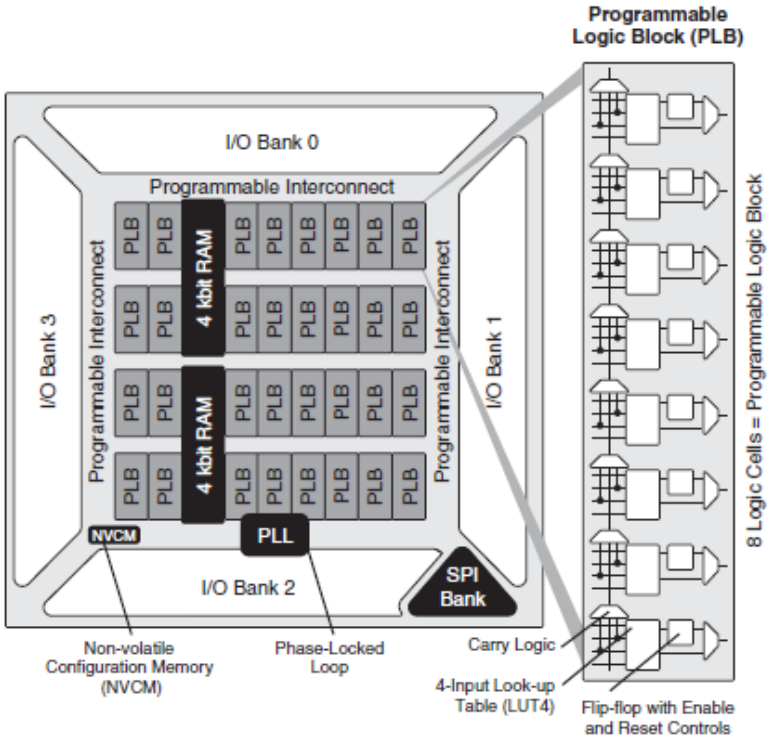


Figura 8.- Visión general de la arquitectura de la familia de FPGAs iCE40

A continuación, vamos a profundizar un poco en la estructura de los bloques lógicos programables (PLB). Estos se componen de tres partes fundamentales que podemos ver en la **figura 9**, sacada del *datasheet* de la familia de FPGAs iCE40 de la que estamos hablando. [12] La primera parte ubicada a la izquierda es una Look-Up Table (LUT) de 4 entradas. Esta nos permite realizar cualquier función lógica combinacional, permitiendo como máximo 4 entradas. Funciona como si fuera una ROM (*Read-Only Memory*) de 16x1, es decir, 16 palabras de solo 1 bit cada una. Combinando y poniendo en cascada varias LUT podremos crear funciones lógicas mucho más complejas y permitiendo un mayor número de entradas.

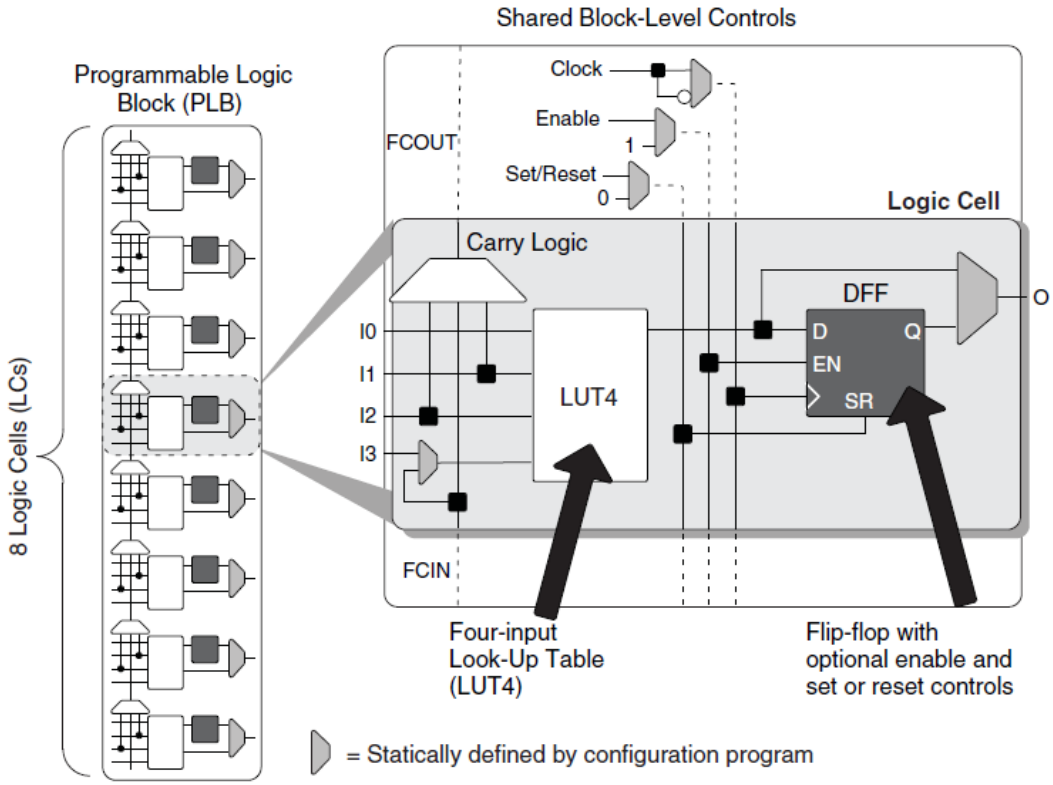


Figura 9.- Diagrama de bloques de los PLBs

El siguiente elemento que podemos apreciar es un *Flip-Flop* de tipo D (DFF), cuya tabla de verdad podemos ver en la **tabla 1**. Este DFF tiene un pin de *clock-enable* opcional y una señal de control de *Reset*. Este *flip-flop* nos permite crear funciones lógicas secuenciales. Además, están conectados todos los DFF de la FPGA a una entrada *Reset* global que los resetea nada más realizarse la configuración de la FPGA. Por último, nos encontramos con la *Carry Logic*, que nos aumenta la eficiencia de las funciones aritméticas, como sumadores, restadores, comparadores…

Tabla 1.- Tabla de verdad de un Flip-Flop tipo D

D Flip Flop Truth Table

CL (Note 1)	D	R	S	Q	$\overline{Q}$
⌐	0	0	0	0	1
⌐	1	0	0	1	0
¬	x	0	0	Q	$\overline{Q}$
x	x	1	0	0	1
x	x	0	1	1	0
x	x	1	1	1	1

No Change
x = Don't Care Case
Note 1: Level Change

Dentro de esta familia de FPGAs para el proyecto se ha seleccionado la iCE40HX1K-TQ144. Esta FPGA en concreto tiene 1280 celdas lógicas, es decir, la LUT más los *Flip-Flops* de tipo D, como hemos explicado anteriormente. Tiene también 16 bloques de memoria RAM4k. Tenemos además un PLL configurable, 96 PIO programables y 12 entradas de par diferencial. La FPGA tiene un voltaje de alimentación de 1V2.

Como podemos ver, el nombre de nuestra FPGA es iCE40HX1K-TQ144. La primera parte (iCE40) hace referencia al nombre de la familia de FPGAs a la que pertenece. El HX nos dice que es un dispositivo de la serie de alto rendimiento. Ya que esta familia de FPGAs tiene otra serie denominada LP, que son dispositivos de muy bajo consumo (*Ultra Low Power*). Después el 1K, nos dice el número de celdas lógicas que tiene el dispositivo seleccionado, en nuestro caso como hemos dicho 1280. Pero también estarían las opciones de 4K (3520 celdas lógicas) y 8K (7680 celdas lógicas). Por último, el TQ144 nos indica el encapsulado elegido, en este caso de tipo TQFP de 144 pines, con un paso de 0,5 mm. Otras opciones de encapsulado que se podrían haber elegido son CB132, VQ100, BG121, entre otros.

En nuestro proyecto hemos seleccionado esta FPGA porque se adapta muy bien a las características del proyecto. También se ha elegido porque es la utilizada en el dispositivo iCEStick, comercializado por Lattice, que es una placa de desarrollo conectada mediante USB. Para algunas partes del diseño hardware de nuestro proyecto nos hemos basado en el diseño de esta placa, por lo que el uso de esta FPGA en concreto facilitaba el proceso de diseño. Además, otra característica que la hacía recomendable para su uso era su bajo precio, ya que costando aproximadamente 5€, es notablemente más barato que otras FPGAs en el mercado. También el número de pines, junto con su gran capacidad de implementar las funciones que deseábamos la hacía muy útil, ya que se podría llegar a usar como controlador de vídeo para la pantalla TFT. Por último, la posibilidad de programar la FPGA mediante el protocolo SPI y poder realizar los programas en un software gráfico de código abierto fue una de las principales razones para seleccionar esta familia de FPGAs. No tener que pagar por un software, y la facilidad de manejo del usado (ICEStudio) hacía esta FPGA ideal para el principal objetivo del proyecto que es su uso para la enseñanza.

En resumen, hemos seleccionado esta FPGA por su reducido precio, por tener un diseño previo en el que podernos basar para la realizar el proyecto y la existencia de un software abierto y gráfico que facilita su programación y su uso para la enseñanza.

b) Microcontrolador LPC1114

El microprocesador elegido para este proyecto ha sido el LPC1114FBD48/302. Esta familia de microprocesadores está basada en ARM Cortex-M0 y son de bajo coste. Puede funcionar a frecuencias de hasta 50 MHz. El precio del microcontrolador elegido ha sido de 1,85€, muy barato frente a otros microprocesadores con características parecidas, lo que lo hacía ideal para nuestro proyecto, buscando siempre componentes lo más asequibles posibles para la creación de nuestra placa de bajo coste. [13]

Entrando dentro de las características del microcontrolador concreto elegido, podemos ver que el encapsulado es un LQFP48. Como dice el propio nombre del encapsulado tenemos 48 pines, y es de tipo montaje superficial (SMD). En nuestro microprocesador tenemos 32kB de memoria Flash y 8kB de memoria SRAM total. Tenemos una UART que usaremos para programarlo, dos controladores SPI y una interfaz I2C. Tenemos además 8 canales ADC y 42 pines GPIO.

En la **figura 10** podemos ver el reparto de pines de nuestro microcontrolador en el encapsulado elegido. Podemos resaltar algunos pines, como por ejemplo el *XTALIN* y *XTALOUT*, en los cuales se conecta el cristal para el funcionamiento del microprocesador. En nuestro caso, el cristal lo hemos seleccionado de 12 MHz. También se aprecian los pines de alimentación (VDD) y los de tierra (VSS), así como los pines correspondientes a los dos controladores SPI. Además, en los pines 47 (TXD) y 46 (RXD), podemos ver la interfaz UART usada para la comunicación con el ordenador a través del integrado CY7C65213-28PVXI. Una cosa apreciable es que muchos pines pueden tener muchas funcionalidades y servir para diferentes usos, por eso, en la programación del microcontrolador habrá que informarle de qué función va a tener cada pin mediante la correcta programación de sus registros.

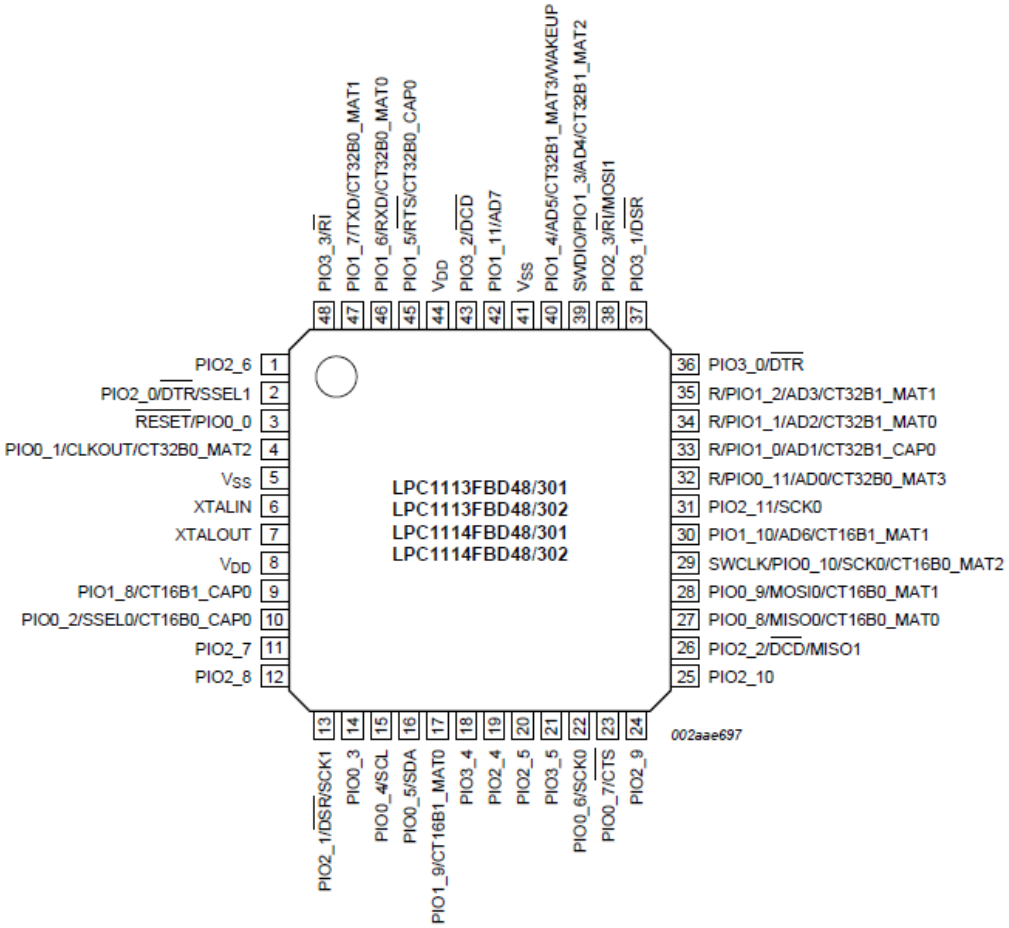


Figura 10.- Reparto de pines en nuestro microcontrolador

Finalmente, en la **figura 11** podemos ver el diagrama de bloques general de nuestra familia de microcontroladores. En él se aprecian los diferentes bloques y pines que hemos visto previamente en esta misma sección. Podemos ver los dos controladores SPI con los 4 pines correspondientes a cada uno de ellos. También a la izquierda vemos la interfaz UART, y justo encima los puertos GPIO. En la parte superior derecha, el bloque de generación del reloj y control de la alimentación. Dentro de este bloque encontramos, por ejemplo, los PLL mediante los cuales aumentaremos la velocidad de funcionamiento de nuestra FPGA, ya que su reloj de entrada será a través del pin *CLKOUT*. Como este pin coincide con el pin de *Bootloader*, una vez iniciado el microcontrolador cambiaremos el uso de este pin para asignarle la utilidad de reloj de salida para hacer funcionar la FPGA, introduciendo esta señal a través de una resistencia en su reloj de entrada.

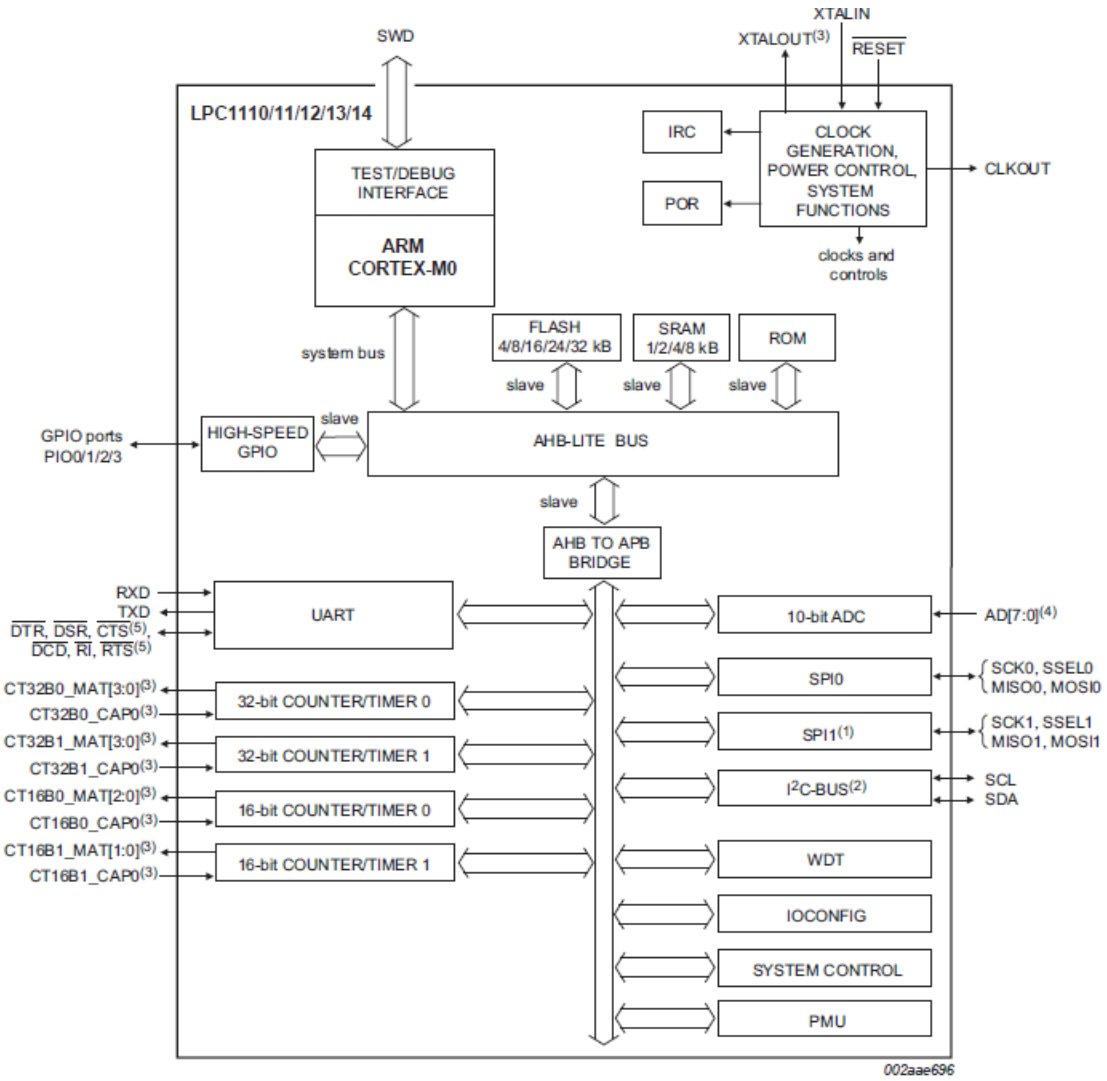


Figura 11.- Diagrama de bloques de nuestro microcontrolador

La principal utilidad del microcontrolador va a ser la de programar las memorias con la información obtenida a través del ordenador. Lo principal que tenemos que grabar en las memorias es el programa correspondiente a la FPGA, para que lo pueda cargar y funcionar correctamente. Además, el microcontrolador manejará a la FPGA, controlando sus pines de control (*ICE_DONE* y *ICE_RESET*), a los que está conectado a través de un pull-up. Regulando el voltaje de estos dos pines el microcontrolador mandará a la FPGA leer el programa de la memoria o que se quede inactiva. Además de esta función clave que es la de controlar la FPGA, tenemos 8 pines GPIO regulables para poderlas dar el uso que queramos. Esto lo podemos apreciar en la **figura 12**, donde se muestra el extracto del microcontrolador del esquemático.

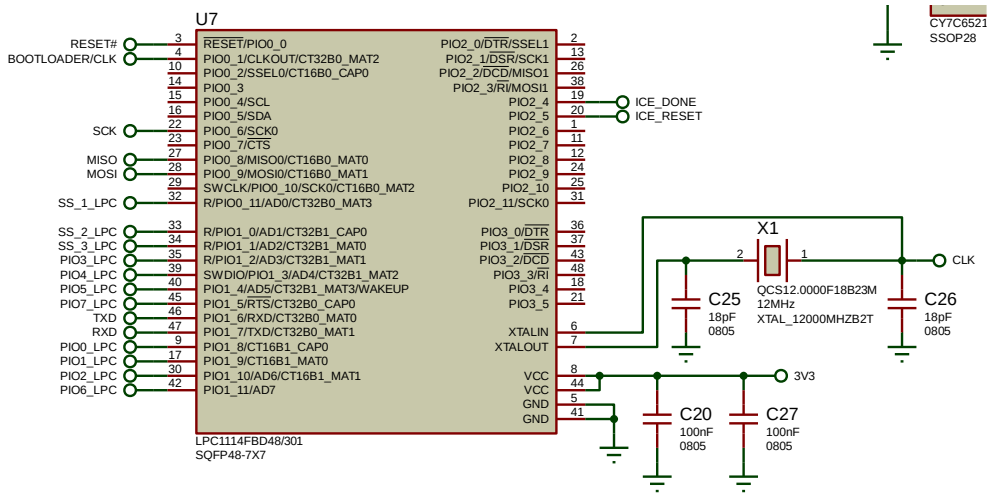


Figura 12.- Extracto del microcontrolador del esquemático

En resumen, podemos decir que el microcontrolador se encarga de la comunicación con el ordenador para su programación y la escritura de las memorias, además del control de la FPGA. Como hemos dicho varias veces a lo largo de este documento, la clave de este proyecto es que el control de la FPGA se hace exclusivamente a través del microcontrolador, lo que lo convierte en la pieza clave del proyecto. La elección de este microcontrolador frente a otros se debe principalmente a su bajo precio, pero que no por ello nos limita las funcionalidades, ya que tenemos suficientes pines para realizar todas las tareas necesarias y a una velocidad de reloj suficientemente alta para que todo funcione rápidamente. Además, mediante los PLL presentes dentro del microcontrolador, como ya hemos dicho, podemos controlar la velocidad de funcionamiento de nuestra FPGA, consiguiendo que funcione más rápido que el propio microcontrolador. Por todo esto, podemos concluir, que el microcontrolador es la pieza más importante de este proyecto, ya que dirige el funcionamiento de la mayoría de los componentes presentes en la placa.

c) Boost Regulator CS5171GDR8G

El siguiente integrado del que vamos a hablar es la fuente elevadora de tensión o *boost regulator*. Lo primero es analizar por qué necesitamos este componente. Según el análisis de consumo de potencia que podemos ver en la página 6 del **anexo 2**, la corriente procedente del USB del ordenador resultaba insuficiente para la alimentación del *backlight* de la pantalla TFT. Del USB podemos obtener como máximo 500 mA, y el *backlight* consume 355 mA, lo que sumado al resto de consumo excedíamos ampliamente la corriente del USB, lo que hacía el diseño inviable. Por eso se optó por que la alimentación del *backlight* se hiciera de forma externa mediante el conector de tornillo presente en la parte izquierda de la placa. [14]

A este conector se le conectan 5V, ya que es un voltaje relativamente fácil de obtener ya que es muy habitual dentro de la electrónica, pero el *backlight* funciona a 10V, por lo que tenemos que aumentar el voltaje de la señal de entrada para lograr el correcto

funcionamiento de la pantalla. Ahí es donde entra en función nuestro elevador de tensión, que subirá los 5V obtenidos del conector a 10V para que funcione el *backlight.*

Como vemos en el esquemático (página 2 del **anexo 2**) para que nuestro integrado funcione correctamente y realice la función asignada necesitamos incorporar un diodo Schottky, además de una inductancia de 22 uH y de una serie de capacidades y resistencias que ahora pasaremos a analizar.

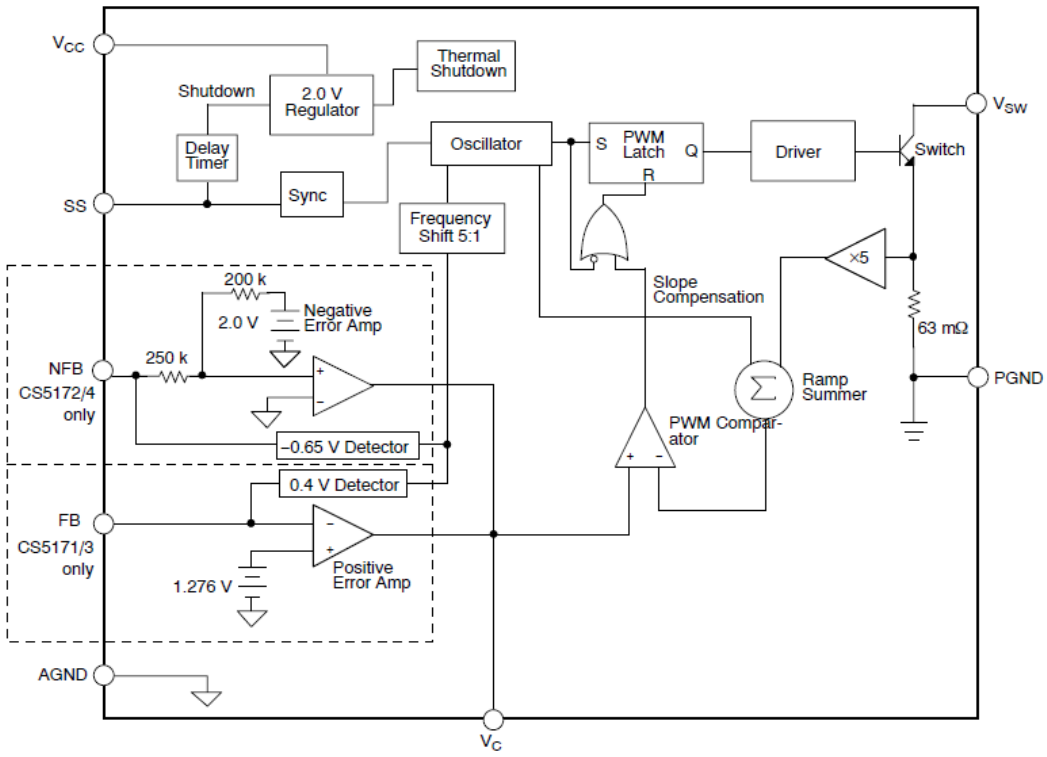


Figura 13.- Diagrama de bloques del CS5171GDR8G

En la **figura 13**, podemos ver el diagrama de bloques interno de este integrado. En nuestro diseño para este componente hemos dejado el pin SS sin conectar. El esquema realizado para este componente se puede apreciar, como ya hemos mencionado, en la parte inferior izquierda de la segunda página del **anexo 2**, que podemos ver en la **figura 14**. El esquema para cualquier voltaje de entrada y de salida es el mismo para todos, lo único que cambia son los valores del divisor de tensión presente en el pin FB. El voltaje de salida se elige mediante este divisor resistivo. El cálculo es inmediato; en el pin FB debemos tener 1,276V, entonces se calcula el valor de las resistencias para, teniendo el valor de voltaje que queremos de salida como alimentación, que en el punto intermedio de las resistencias haya el voltaje de 1,276V. Para entenderlo mejor pongamos de ejemplo nuestro caso.

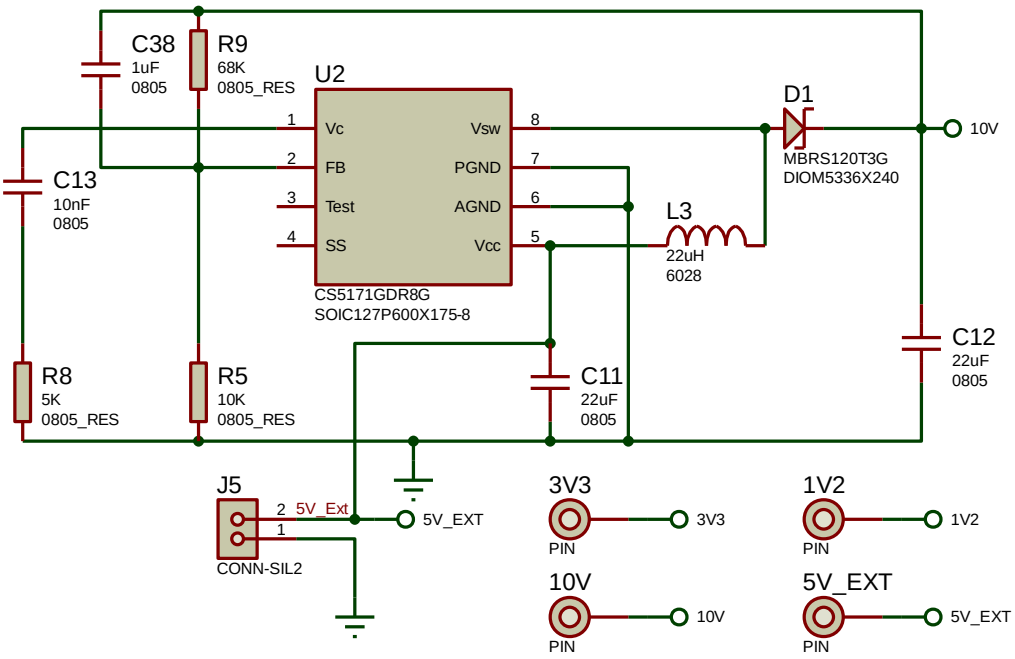


Figura 14.- Extracto del Boost Regulator del esquemáticos

En la salida queremos tener 10V y en el pin FB queremos 1,276V, por lo tanto, suponiendo R1 la resistencia conectada a 10V y a FB, y R2 la resistencia conectada a FB y a tierra, la fórmula a seguir sería:

$$1{,}276 = 10\frac{R_2}{R_1 + R_2}$$

Por todo ello, dado que las resistencias debían ser de la serie E12, siguiendo los cálculos, se llegó a la conclusión de que R1 debía de ser 68k y R2 de 10k, como podemos ver en el esquemático. Finalmente, el voltaje de salida será el que salga de nuestro diodo Schottky, que por el otro lado está conectado al pin VSW de nuestro elevador.

Este integrado elegido funciona a una frecuencia de 280kHz, lo que nos da una salida con un voltaje bastante estable de 10V. Además, tiene un consumo de 8 mA interno como máximo, teniendo un típico de 5,5 mA. El análisis térmico de este componente es un poco complejo, ya que hay que calcular varias potencias pertenecientes al dispositivo, que analizaremos más adelante, en el apartado correspondiente.

Por último, hay que resaltar que este componente no se ha usado mucho ya que en este proyecto no se ha llegado a programar la pantalla TFT. Nos hemos limitado a comprobar su correcto funcionamiento, que daba los 10V necesarios en la salida, y que era suficiente para alimentar el *backlight* de la pantalla TFT. La elección de este integrado se debió a su bajo coste de 1,6€ y a su fácil funcionamiento ya que como hemos visto solo se regula mediante los valores de dos resistencias.

d) CI de interfaz USB CY7C65213-28PVXI

Otro integrado fundamental que podemos encontrar en nuestra placa es la interfaz USB. Esta, de forma simplificada, traduce la información que viene del ordenador para que el microcontrolador lo entienda. La información del ordenador viene mediante el conector USB a través de los pines D+ y D-. Estos pines se conectan en este integrado, que transmite la información a través de una UART al microcontrolador, para programarlo o mandar la información que queramos o necesitamos. [15]

Este integrado no hace falta alimentarlo mediante 3V3 ya que puede funcionar correctamente con 5V. Lo que hay que tener en cuenta es la diferencia de que nuestro microprocesador funciona con 3V3, mientras que este integrado funciona con 5V. Esta diferencia quedará plasmada en que la información que transmita el integrado a nuestro microcontrolador, por ejemplo, el *Reset* o el *Bootloader* tendrá un valor máximo de 5V, mientras que lo que se transmita en sentido contrario, del microprocesador al integrado, tendrá como límite 3V3. No hay problema por esto ya que, como podemos ver en el *datasheet*, nuestro microprocesador tiene unos pines tolerantes a 5V cuando el voltaje de alimentación está presente. Por esto, la comunicación entre los dos integrados se realizará correctamente, y no se estropeará el microcontrolador, a pesar de tener voltajes de alimentación diferentes.

En la **figura 16** podemos ver el diagrama de bloques del integrado que realiza nuestra conversión de USB-UART. Lo primero que observamos son los pines de alimentación, que como añadido tienen un pin con voltaje de salida de 1V8, que no usaremos, por lo que, siguiendo el datasheet hemos conectado un condensador conectado a tierra. El diseño realizado se puede ver en el esquemático realizado, en la página 3 del **anexo 2** y en la **figura 15**.

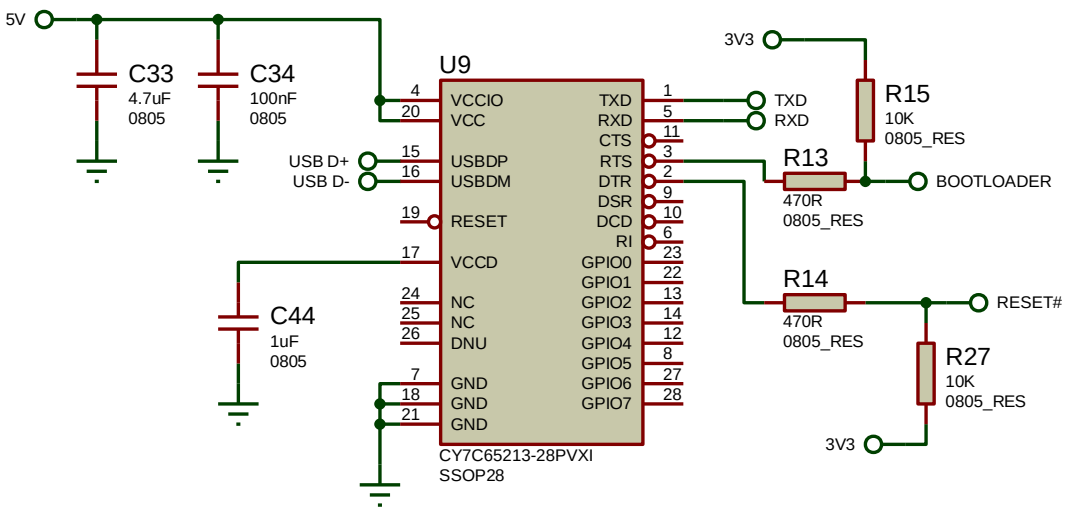


Figura 15.- Extracto de la interfaz USB-UART del esquemático

Procedente del USB conectaremos los pines D- a USBDM y D+ a USBDP, además de los pines de alimentación a los 5V procedentes del USB añadiendo dos condensadores de desacoplo.

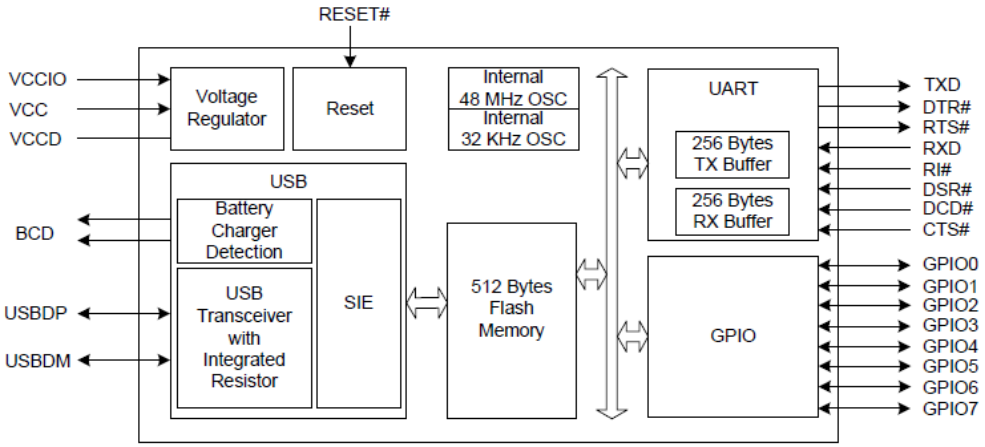


Figura 16.- Diagrama de bloques del integrado CY7C65213-28PVXI

Por el otro lado, conectado al microcontrolador, tenemos 4 pines. Los dos primeros son los correspondientes a la UART: TXD y RXD. La técnica seguida para la interconexión de estos pines es la de *Boca-Oreja*, es decir, conectar el TXD con el RXD del otro y viceversa. En nuestro caso, hemos conectado el TXD de la interfaz USB-UART al RXD del microprocesador y viceversa. Además, tenemos otros dos pines conectados al microprocesador que son los pines de *Reset*, DTR en el integrado y el pin de *Bootloader*, RTS en el integrado. Estos dos pines los conectaremos a nuestro microprocesador mediante dos *pull-up*, a los pines 3 y 4, respectivamente.

Cabe destacar como último dato, como podemos ver en el diagrama de bloques de la **figura 16**, la UART del integrado tiene dos buffers, de transmisión y de recepción cada uno de 256 Bytes. Este dato habrá que tenerlo presente ya que nos limitará a la hora de la programación la cantidad de información que podemos enviar desde el ordenador en cada envío. Por ejemplo, como veremos en el capítulo correspondiente al software, el archivo que tendrá que grabar el microprocesador en las memorias se enviará en *chunks* (trozos) de 256 Bytes, porque es el tamaño límite del buffer de transmisión, como acabamos de ver.

Como podemos ver en el diagrama de bloques, el integrado elegido presenta muchas otras características, además de las usadas en nuestro diseño. Todas estas funcionalidades extras no se han usado ya que no eran necesarias para el funcionamiento del proyecto, y habría sido una complicación añadida innecesaria.

Al igual que los componentes anteriores, la elección de este integrado no es casual, ya que tiene un precio inferior frente a integrados con las mismas características en el mercado. Este integrado cuesta 1,9€ en el proveedor Farnell, mientras que integrados con las mismas funcionalidades suelen estar en torno a los 5€. La elección de este frente al resto se justifica en que los otros suelen aumentar el precio por funcionalidades añadidas, que en nuestro caso son innecesarias ya que el uso que le damos a nuestro integrado es muy básico. Como comentábamos en un principio, la función principal va a ser la traducción de la información que nos llega del ordenador para que lo entienda nuestro microprocesador. Esta tarea, aunque parezca simple, es de las más importantes de nuestra placa ya que es la que nos permitirá programar el microprocesador para que pueda controlar la FPGA y las memorias, función principal del proyecto. Por lo que este integrado es una pieza fundamental e imprescindible de nuestro placa.

e) Regulador de Voltaje LDO MCP1725-ADJE/SN

Uno de los últimos integrados presentes en nuestra placa son dos reguladores lineales de voltaje, que nos realizan la conversión del voltaje de alimentación del USB de 5V, a los voltajes de alimentación necesarios para el funcionamiento de nuestro circuito, que son 3V3 y 1V2. Lo que nos indica que sea un regulador lineal es que la corriente de entrada en el integrado será la misma que la corriente de salida, descontando, obviamente, las pérdidas mínimas que produce el propio integrado. Como el voltaje de salida va a ser inferior al de entrada, la diferencia de potencia presente entre la salida y la entrada se disipará en forma de calor en nuestro integrado por lo que, como veremos en el apartado de la PCB, hemos tenido que añadir una aleta de cobre en cada una para la disipación de este calor. [16] La conexión de estos componentes la podemos ver en la **figura 17** sacada de la página 2 del **anexo 2**.

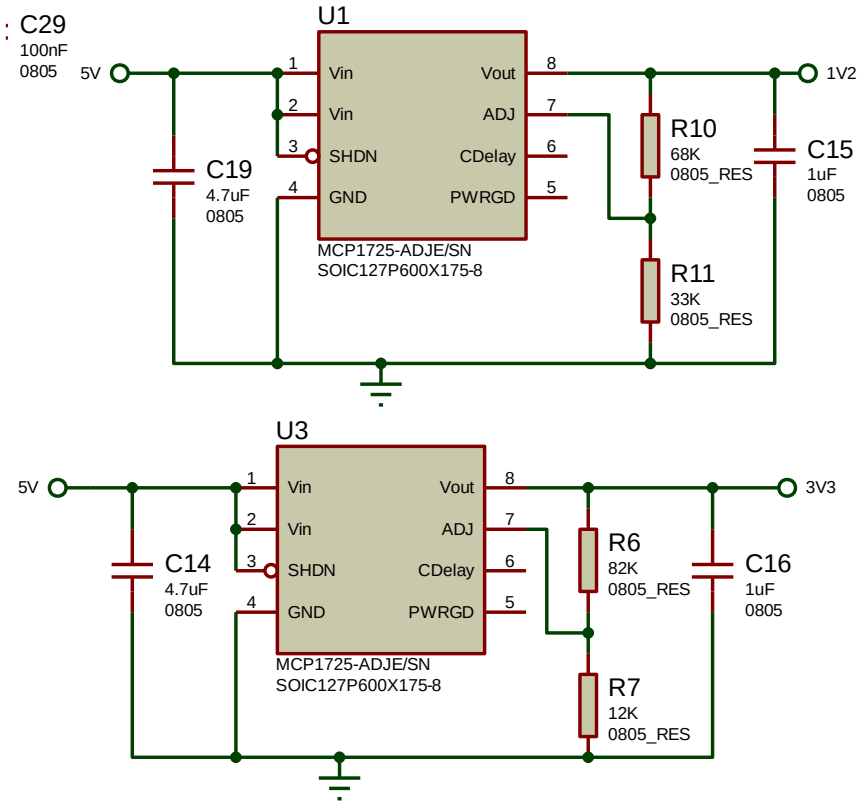


Figura 17.- Extracto de los reguladores lineales del esquemático

Para la implementación de estos dos reguladores lineales se podría haber optado por dos métodos de conexión diferentes. El primer método consiste en una conexión en cadena, es decir, al primer regulador lineal se le conecta el voltaje de alimentación del circuito, y al segundo regulador lineal se le conecta en la entrada la salida del regulador lineal anterior. El segundo método es realizarlo en paralelo, es decir, conectar el voltaje de alimentación del circuito a la entrada de los dos reguladores lineales. Finalmente se optó por usar en la placa el segundo método. Se eligió por diversas razones, la primera es que el segundo regulador lineal, a pesar de tener un gran diferencia de voltaje entre la entrada y la salida (5V y 1V2 respectivamente), como la corriente es muy pequeña, no tendremos

problemas de calentamiento por la potencia disipada. La segunda razón es que al hacerlo en paralelo evitamos que toda la potencia atraviese el primer regulador lineal, con el que ya teníamos un problema de calentamiento, al estar cercano a la temperatura límite del componente, como podemos en los cálculos térmicos de la página 7 del **anexo 2**. Por ello, evitando que la corriente del segundo regulador lineal atraviese el primero, evitamos que se caliente más. A pesar de ello, cabe mencionar, que el primer método suele ser más usado normalmente, aunque como vemos en nuestro caso, depende de las características y necesidades de cada circuito.

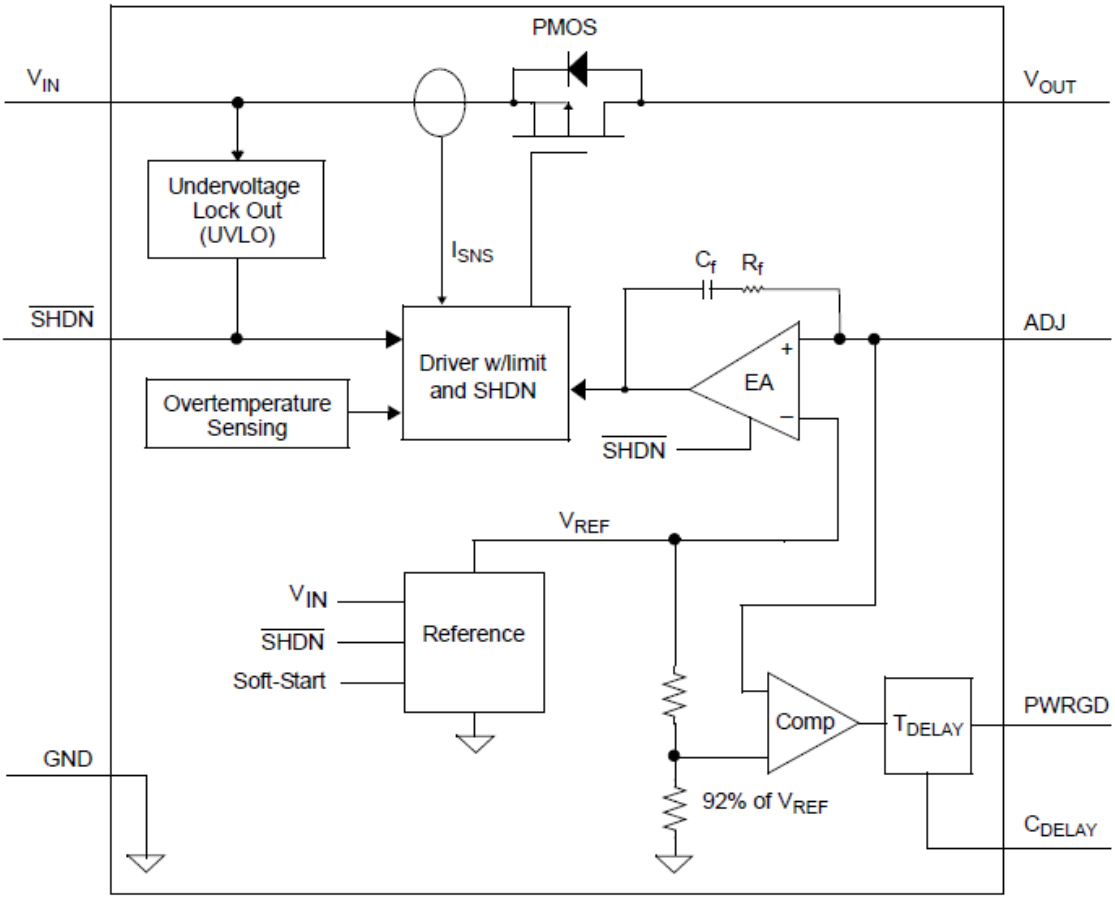


Figura 18.- Diagrama de bloques del regulador lineal MCP1725

En **la figura 18** podemos ver el diagrama de bloques interno de los reguladores lineales y en la página 2 del **anexo 2**, podemos ver los diseños realizados en nuestra placa. Como cabe esperar en Vin tenemos que conectar el voltaje de entrada, en nuestro caso, 5V; y en Vout saldrá el voltaje deseado, 1V2 y 3V3 en nuestro caso. Pero ¿cómo se selecciona el voltaje de salida que queremos? Pues es un método prácticamente igual al seguido en el elevador de voltaje del que hemos hablado en un apartado anterior. La clave reside en que en el pin ADJ debemos tener aproximadamente 0,41V, por lo que, al igual que lo que explicamos previamente, tendremos que realizar un divisor resistivo, con el voltaje de salida, para obtener en este pin el voltaje deseado. Realizando el cálculo con la ecuación vista en el **apartado c)** de esta sección, y suponiendo también que las resistencias deben ser de la serie E12 hemos concluido que para los reguladores lineales necesitamos las siguientes resistencias: para el primer regulador, el de 3V3, necesitaremos una de 82k y

otra de 12k; y para el segundo regulador, el de 1V2, necesitaremos una de 68k y otra de 33k. La primera resistencia es la que va conectada al pin ADJ y al voltaje de salida, mientras que la segunda es la que va conectada al pin ADJ y a tierra.

Como podemos ver en el datasheet de este componente, recomiendan que, por estabilidad del componente y un correcto funcionamiento, habría que añadir un condensador en la entrada y otro en la salida del integrado. Por ello, como podemos ver en el esquemático, se ha añadido en la salida de ambos reguladores un condensador de 1uF, mientras que en la entrada se ha conectado uno de 4,7uF.

La principal razón para la elección de este componente fue primordialmente la necesidad, ya que para que puedan funcionar el resto de integrados necesitamos tener un voltaje adecuado, y eso solo lo podemos obtener mediante unos reguladores de voltajes, que nos bajen los 5V a los voltajes necesitados. Un regulador lineal es muy útil, por su principal característica, ya mencionada, de que la corriente de entrada es igual a la de salida, descontando unas mínimas pérdidas intrínsecas del componente. La primera razón para elegir este componente frente a otros con características similares es la misma que llevamos comentando con el resto de los componentes y es su bajo precio, ya que cada uno cuesta únicamente 0,47€. Por lo que, por menos de 1€ tenemos los dos reguladores. Además, aunque este regulador permita solo como máximo 500 mA, para nuestra aplicación es ideal, ya que nunca el puerto USB nos va a dar más de esa cantidad porque es la máxima corriente que puede dar. Por lo que nuestros integrados no tienen riesgo de rotura por excesiva corriente. Otra razón para su elección es su versatilidad, ya que no nos dan un voltaje definido de salida como otros voltajes, sino que podemos controlar el voltaje que queremos en la salida mediante dos resistencias como hemos comentado previamente. Con esto conseguimos que con el mismo integrado y el mismo voltaje de entrada, logremos los dos voltajes de salida que necesitamos. Esto siempre reduce el precio final, ya que evitamos tener que comprar dos componentes diferentes porque con uno solo obtenemos el mismo resultado. Además, los reguladores lineales con voltajes fijos de salida suelen ser algo más caros que los ajustables, a pesar de ser menos versátiles.

f) Memorias

Finalmente, vamos a hablar de los últimos integrados presentes en nuestra placa, que son las tres memorias que tenemos. En la etapa de diseño se optó por unas memorias llamadas I25LP032D-JNLE, de coste 1,32€, pero posteriormente en la etapa de montaje, descubrimos que las memorias de las que disponíamos eran las llamadas N25Q064. Por ello, para ahorrar costes, y dado que tenían prácticamente las mismas características, se optó por estas segundas. La única modificación que hubo que hacer fue que la huella de la placa era ligeramente más estrecha que esta memoria, por lo que hubo que apretar ligeramente los pads de las memorias para que cupieran en el hueco de las memorias. [17]

Estas memorias seleccionadas están desarrolladas mediante tecnológica NOR de 65nm. Son memorias Flash, no se borran al desconectar la alimentación del circuito, lo que nos será útil para algunas aplicaciones de nuestra placa. Son controladas mediante el protocolo SPI, ya visto en secciones anteriores. Tienen un tamaño de 64Mb, y su memoria está distribuida como podemos ver en la **tabla 2**. Los 64Mb están divididos en 128 sectores, que, a su vez, están divididos en 16 subsectores. Esto nos permite que el borrado

se pueda hacer de un sector (64kB), de un subsector (4kB) o realizar un borrado total. Finalmente, la memoria está dividida en páginas de 256 bytes, que es lo que podemos leer o escribir.

Tabla 2.- Mapa de memoria de las memorias

Sector	Subsector	Address Range	
		Start	End
127	2047	007F F000h	007F FFFFh
	⋮	⋮	⋮
	2032	007F 0000h	007F 0FFFh
⋮	⋮	⋮	⋮
63	1023	003F F000h	003F FFFFh
	⋮	⋮	⋮
	1008	003F 0000h	003F 0FFFh
⋮	⋮	⋮	⋮
0	15	0000 F000h	0000 FFFFh
	⋮	⋮	⋮
	0	0000 0000h	0000 0FFFh

En la **figura 20**, podemos ver el diagrama de bloques interno de la memoria. En este observamos como la memoria está dividida en páginas de 256 bytes. Además, en la parte superior izquierda, podemos ver los pines que vamos a tener que usar, aunque como vienen en una nomenclatura diferente a la habitual, vamos a proceder a traducirlos a la usada normalmente. El primer pin que vemos es el *Hold#*. Este pin es usado para parar cualquier comunicación serie con el dispositivo, sin deseleccionarle, y es activo en baja. A continuación, nos encontramos con el pin *W#/Vpp*, llamado además, en otras páginas del *datasheet*, como DQ2. En nuestro esquemático a este pin le hemos llamado WP y es el protector de escritura (*Write Protect*). El siguiente pin es el S#, que es el pin de selección del protocolo SPI utilizado, activo en baja. Después el pin C es el pin SCK del protocolo SPI, es decir, el reloj. Por último, nos encontramos con otros dos pines que son el DQ0 y el DQ1. Estos dos corresponden a los pines MOSI y MISO del protocolo SPI, respectivamente. Todo esto lo vemos en la **figura 19** correspondiente al extracto de una memoria del esquemático del **anexo 2.**

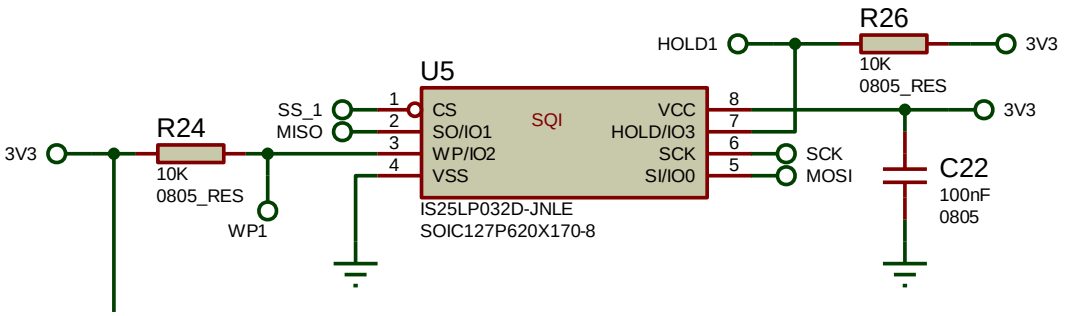


Figura 19.- Extracto de la conexión de una memoria de esquemático

Estas memorias, además de los comandos básicos como la escritura y la lectura, posee otras funcionalidades añadidas como por ejemplo el modo XIP, que permite sacar datos

de la memoria únicamente con una dirección, sin una instrucción. La parte de la programación de la memoria, y los comandos necesarios para hacerla funcionar se estudiará más a fondo en el capítulo IV correspondiente al software desarrollado para su funcionamiento. Ahí profundizaremos en la estructura de la memoria y los procesos para realizar su borrado, escribir en la memoria o leer de esta.

Para concluir, cabe destacar que no podemos decir el precio de la memoria montada ya que no se fabrica, ni se puede encontrar. Por eso en el BOM (**anexo 3**) y en el esquemático tenemos puesta la memoria seleccionada en un principio que tiene un coste de 1,32€ por unidad. He de recordar que esta memoria tiene prácticamente las mismas características que la memoria montada, por lo que no hay grandes diferencias entre ambas, siendo compatibles entre sí.

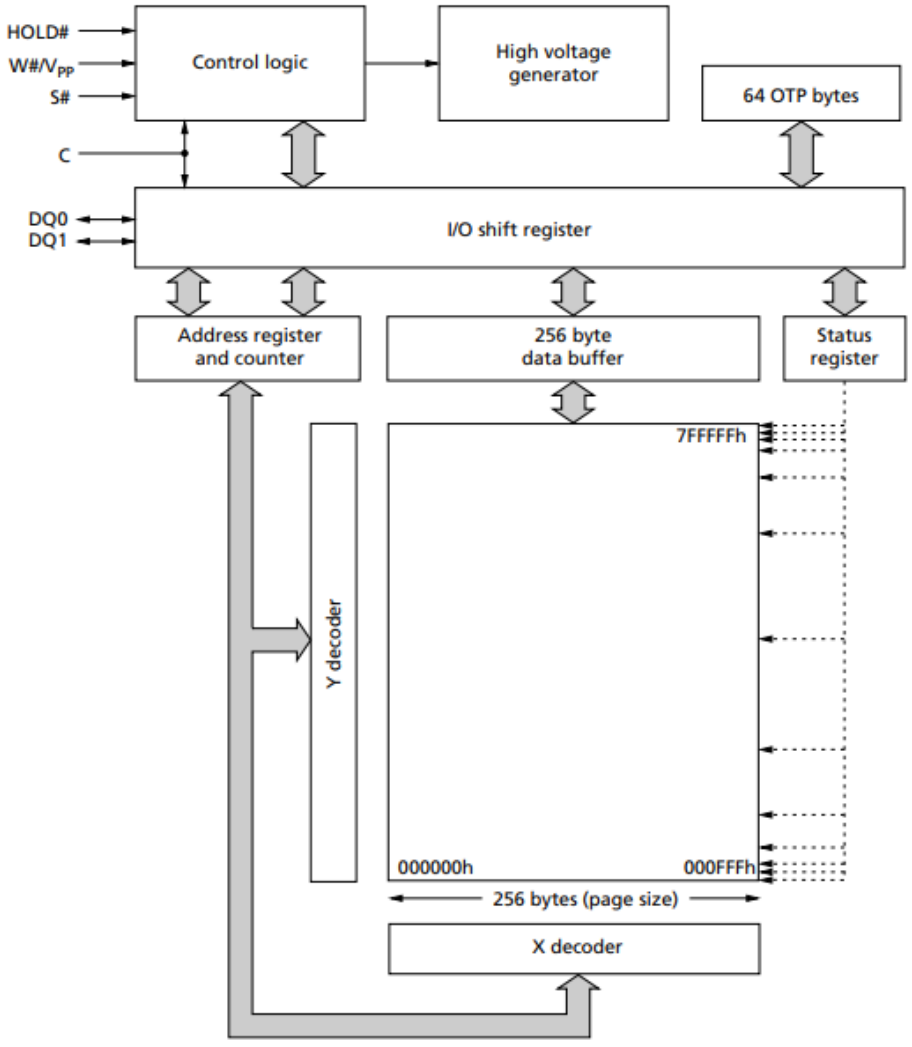


Figura 20.- Diagrama de bloques de la memoria N25Q064

ESQUEMÁTICO

En esta nueva sección vamos a analizar el esquemático diseñado, que encontraremos en el **anexo 2**. En la primera página, **figura 21**, podemos encontrar los bloques correspondientes a 4 de los bancos de la FPGA. En esta página veremos todos los pines que van conectados a la pantalla TFT para su correcto funcionamiento en el banco número 1. En el segundo y tercer banco podemos ver los pines GPIO que van a nuestro conector macho. Estos pines son nuestra conexión de la FPGA con el exterior. Se pueden controlar para usarlos de entrada y salida pudiendo conectar cualquier componente externo. Cabe destacar además que en el segundo banco también tenemos los pines conectados a los 4 LEDs para iluminarlos y controlarlos. Nos pueden servir de prueba o de señalización visual. El programa básico que veremos en el capítulo correspondiente al software consistirá en lograr encender estos LEDs. Además, en este mismo banco, tenemos los pines de control de la FPGA (ICE_DONE y ICE_RESET), que como ya hemos visto, van conectados al microprocesador a través de un *pull-up*. Finalmente, en el banco número 0, tenemos, además del pin con la señal de reloj de entrada, los pines con los que podemos controlar las tres memorias presentes en la placa.

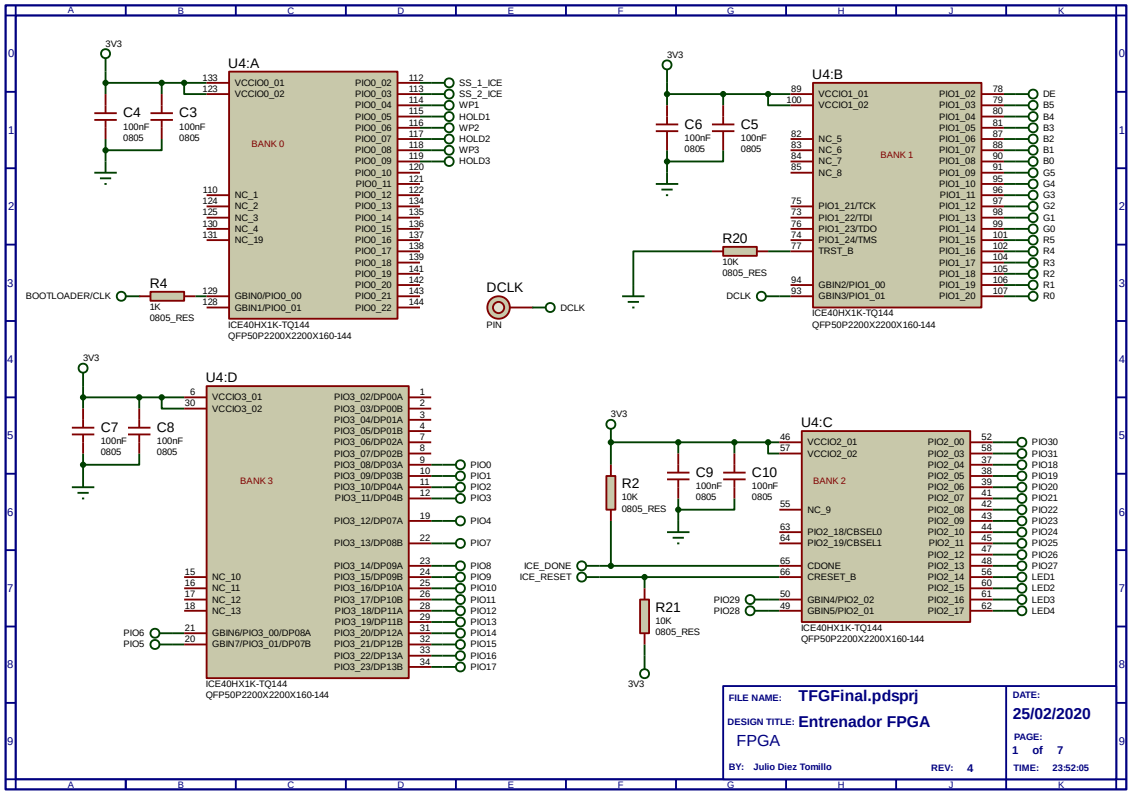


Figura 21.- Página 1 del esquemático

En la segunda página, **figura 22**, podemos ver los componentes correspondientes a la alimentación. En la parte superior derecha encontraremos el conector USB, que nos alimenta el circuito con 5V a través de un núcleo de ferrita. Estos 5V como hemos visto previamente, alimentan nuestros dos reguladores lineales que nos dan los voltajes necesarios (1V2 y 3V3) para el correcto funcionamiento de todo el circuito. En la parte

inferior izquierda tenemos el elevador de voltaje del que también hemos hablado ya. Va alimentado por 5V también, pero estos vienen del exterior mediante el conector que veremos en la parte inferior. Finalmente, en la parte superior izquierda podemos ver el banco de alimentación de la FPGA. Dentro de este bloque tenemos todos los pines de alimentación que le hace falta a la FPGA para funcionar. A cada pin de alimentación le conectamos un condensador de desacoplo para conseguir una alimentación lo más estable posible, evitando en la mayor medida posible los ruidos.

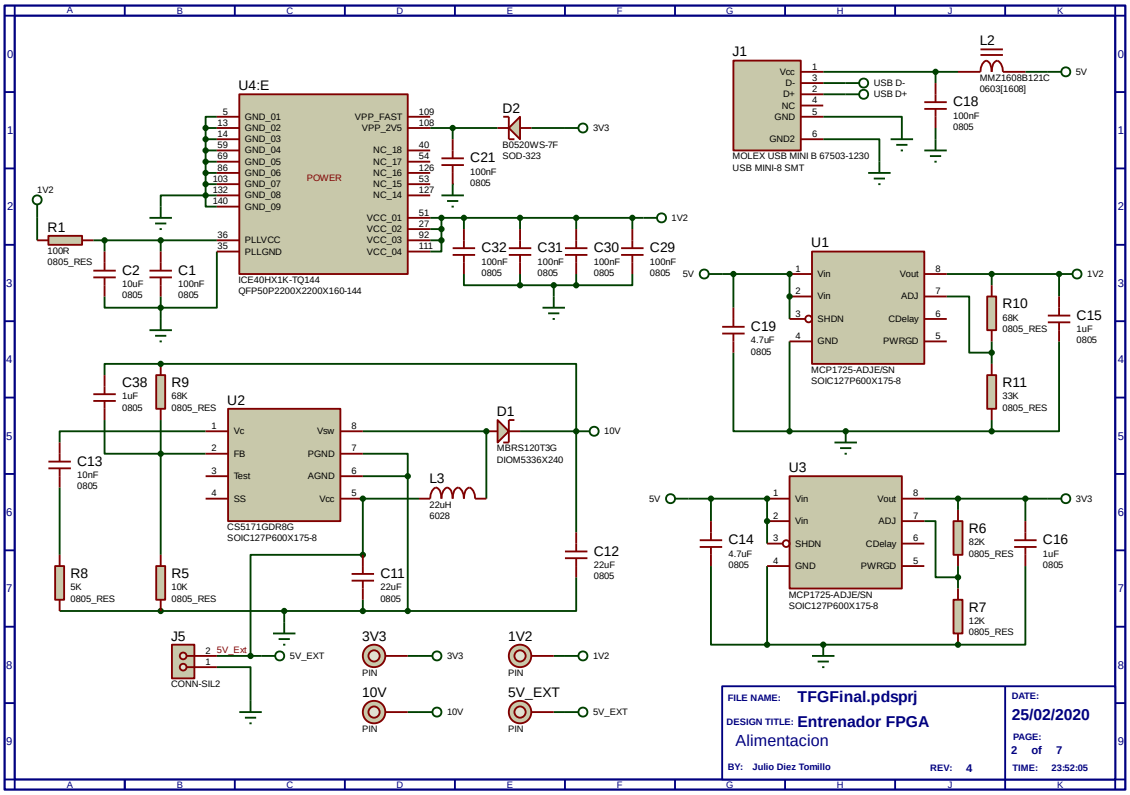


Figura 22.- Página 2 del esquemático

En la tercera página, **figura 23**, nos encontramos con nuestro microprocesador y la interfaz USB-UART. Como hemos visto anteriormente, la interfaz se conecta al microprocesador mediante 4 pines diferentes: *TXD*, *RXD*, *Bootloader* y *Reset*; estos dos últimos conectados mediante un *pull-up*. En el microprocesador podemos ver como tiene estos 4 pines conectados. Además, veremos los dos pines de control de la FPGA. También, nos encontramos con el cristal de 12 MHz, al que le tenemos que conectar dos condensadores de 18 pF por las especificaciones de su *datasheet*. Finalmente, en nuestro microprocesador podemos ver los pines SPI para controlar las memorias, además de los pines GPIO que están conectados al conector macho de la parte inferior. Estos pines, al igual que los de la FPGA nos sirven de conexión del microprocesador con el exterior, pudiendo conectar algún conector para controlarlo, o para obtener alguna medida exterior. Sus posibles usos son inmensos y variados.

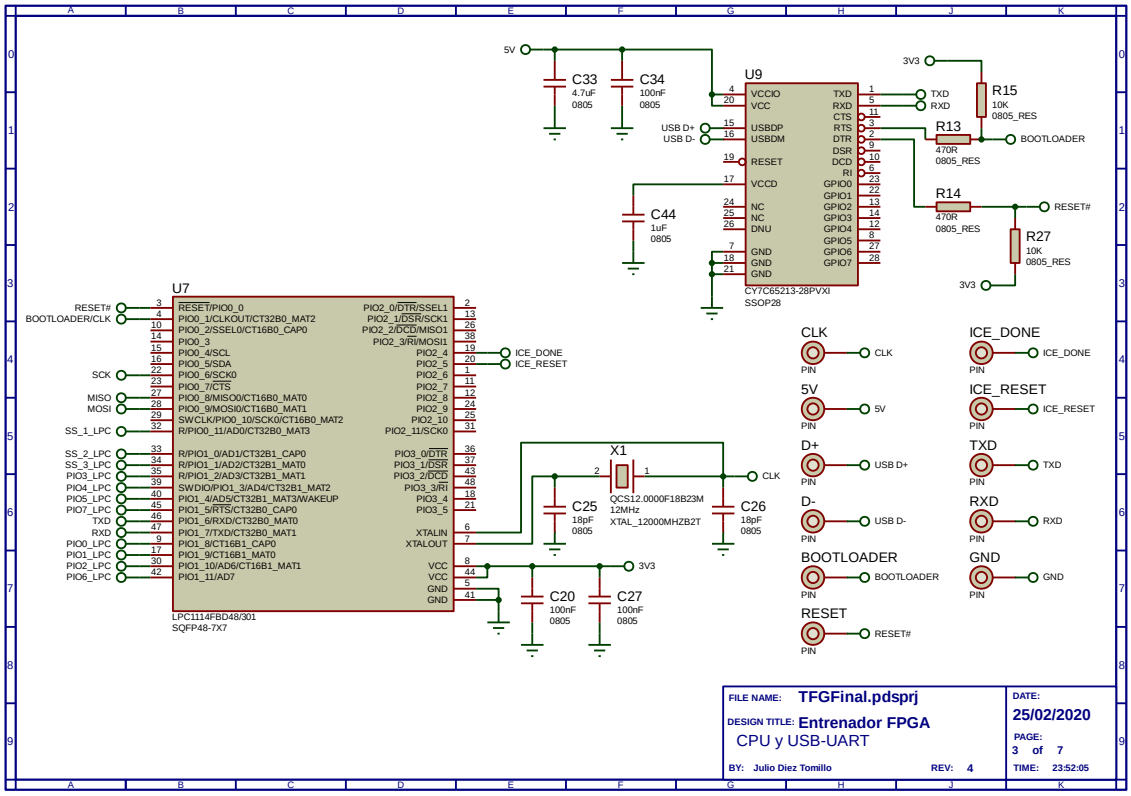


Figura 23.- Página 3 del esquemático

En la cuarta página, **figura 24**, nos encontramos con la mayoría de los conectores presentes en nuestra placa. En los conectores macho de la FPGA y el microprocesador, tenemos conectados nuestros pines GPIO, y además tenemos pines de salida de alimentación (3V3) y de tierra. El conector de la FPGA es de 36 pines, teniendo dos de tierra y dos de alimentación, mientras que el del microprocesador es de 10, teniendo solo uno de cada. A la derecha del todo nos encontramos primero con el conector de la TFT, donde vemos todos los pines conectados a la FPGA correspondientes a los colores (RGB), además del *Data Enable* (DE) y el reloj (DCLK). También tiene dos pines de alimentación a 3V3 y múltiples pines conectados a tierra. Debajo nos encontramos con el conector del *backlight* de la TFT, al que conectamos tierra y los 10V salientes del elevador de voltaje, conectado a través de una resistencia de 10R. La razón de esta resistencia es que el *backlight* está compuesto por unos LEDs puestos en serie; si no se pone la resistencia, pasará toda la corriente que pueda dar la fuente de alimentación, por lo que, si no está limitada la corriente los LEDs se fundirán. Al poner esta resistencia limitamos la corriente que pasa a través de ella, y por ello la que pasa a través de los LEDs, evitando que estos se puedan fundir. Finalmente, en la parte inferior se encuentran los 4 LEDs presentes en la placa, dos rojos y dos verdes. Por la misma razón que hemos comentado previamente hemos añadido una resistencia a cada LED para limitar la corriente que pasa a través de ellos.

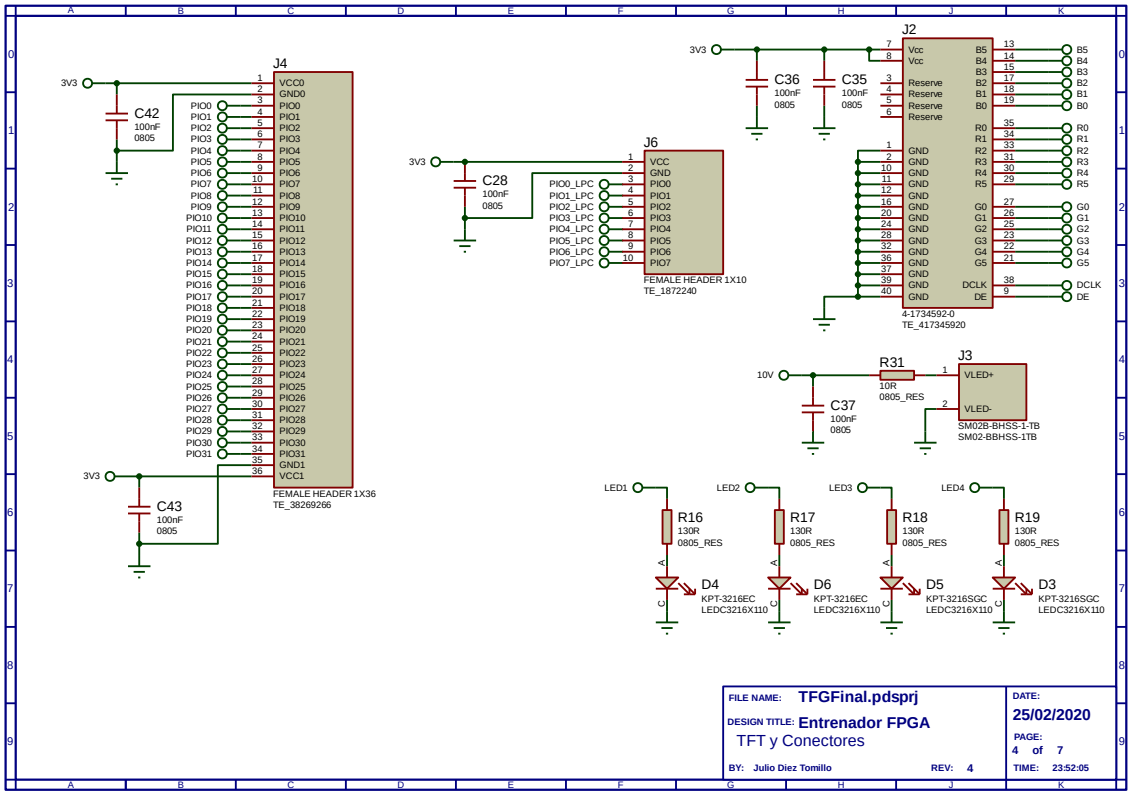


Figura 24.- Página 4 del esquemático

Por último, en la quinta página, **figura 25**, se recoge con lo correspondiente a las memorias y su control. Por ello, nos encontramos con los bloques correspondientes a las tres memorias, además del banco SPI de la FPGA. Como hemos visto en el segundo capítulo, en la explicación del protocolo SPI, todos los pines con la misma función van conectados en el mismo bus. Por ello tenemos un bus correspondiente al MISO, otro al MOSI… Además, tenemos los pines de selección, que nos permiten elegir en que memoria queremos elegir. La memoria predeterminada para que lea y escriba la FPGA es la memoria número tres, que es de donde cargará su programa de funcionamiento. Esto lo podemos ver en el banco SPI de la FPGA, que en el pin de selección está conectado al pin de selección de la memoria número tres. Como podemos ver, además, los pines correspondientes a la selección (SS), los de *Hold* y los de *Write Protect* (WP), tienen un *pull-up* incorporado.

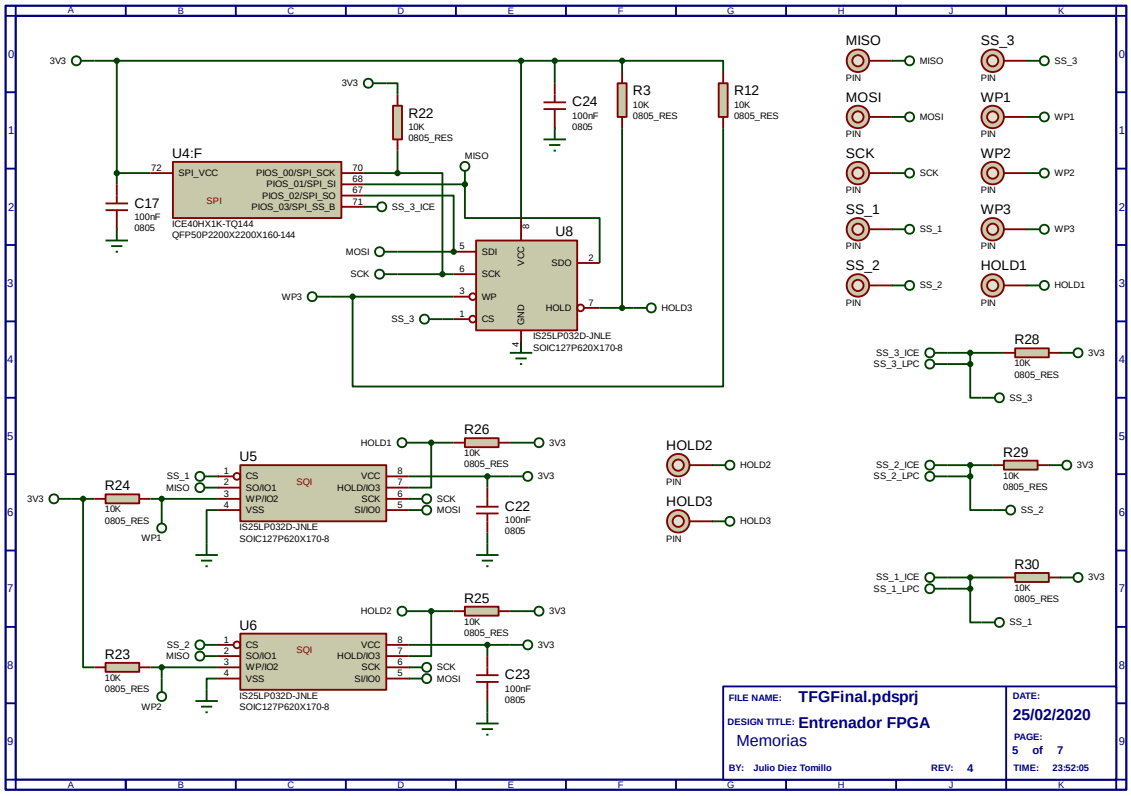


Figura 25.- Página 5 del esquemático

Como último comentario hay que añadir que en todas las hojas se pueden ver unos pines conectados a diferentes nodos del esquemático. Estos son puntos de test que nos servirán, una vez montada la placa, para acceder fácilmente a nodos importantes en el circuito que son interesantes saber su valor en cada momento. Estos puntos de test resultarán muy útiles en la etapa de depuración hardware o software para comprobar que todo funciona correctamente, y que los voltajes tienen los valores necesarios.

En las siguientes secciones vamos a tratar las dos últimas páginas del esquemático, correspondientes al análisis de consumo de potencia del circuito y al análisis térmico de todos los componentes. Esto es muy necesario a la vez que útil para ver, antes de fabricar el circuito, si disponemos de suficiente potencia para hacer funcionar el circuito. También sirve para analizar que ningún componente se va a calentar excesivamente provocando algún fallo, incluso la ruptura. Además, con estos cálculos podemos prevenir problemas, incluyendo, por ejemplo, aletas de cobre que nos permitan disipar este calor excesivo.

a) Consumo de Potencia

Como hemos adelantado en el anterior apartado, realizar un análisis de consumo de potencia de nuestro circuito es muy importante ya que sabremos, mediante unos sencillos cálculos, antes de su fabricación, la alimentación que necesitaremos para nuestro circuito. Mediante estos cálculos iniciales sabremos si nuestro circuito una vez montado va a disponer de suficiente corriente para todos los componentes, o hará falta otra alimentación.

El análisis de consumo de potencia realizado en nuestro circuito lo podemos ver en la página 6 del **anexo 2** de esta memoria y en la **figura 26**. Como ya hemos mencionado en apartados anteriores, inicialmente se iba a disponer únicamente de una alimentación, la proveniente del conector USB. Pero después de realizar una serie de cálculos, descritos en el anexo, se concluyó que la alimentación del USB no era suficiente para todo el circuito. Por ello, se añadió la alimentación externa de 5 V, mediante el conector de tornillo. Esta alimentación externa se usa únicamente para alimentar el backlight de la pantalla TFT.

```
@1V2
FPGA  ICCpeak                        6.9  mA
-----------------------------------------------------
                            TOTAL   6.9  mA

@3V3
TFT  Pantalla      150 mA typ.    200.00 mA máx.
x3 IS25LP032D  25 mA ->            75.0  mA
x4 LEDs        10 mA ->            40.00 mA
x4 FPGA ICCbank_peak 6.8->         27.2  mA
FPGA ICC_pll_peak                   1.8  mA
LPC1114 @ 48 MHz                    9.0  mA
-----------------------------------------------------
                            TOTAL 343    mA

@5V
Corriente @1V2                      6.9  mA   regulador lineal
reguladror 1V2 (interno)            0.12 mA
Corriente @3V3                    343    mA   regulador lineal
regulador 3V3 (interno)             0.12 mA
CY7C65213                          18.0  mA
-----------------------------------------------------
                           TOTAL  368.14 mA


@10V (Proceden de 5V_Ext)
TFT  LEDs
 160 mA typ.
 200 mA máx.

Si ponemos 200 mA -> potencia = 2.0  W
eficiencia ~90%   -> 2.0/0.9 -> 2.22 W
que salen de los 5V, luego 2.22W/5V -> 444 mA

si ponemos 160mA  -> potencia = 1.6W
                  -> 1.6W/0.9 = 1.77W
que a 5V -> 1.77W/5V -> 355 mA


@5V_Ext
reguladror CS5171                   8.00 mA   consumo interno
Corriente @10V                    355.0  mA   fuente conmutada
-----------------------------------------------------
                           TOTAL  363.0  mA

Si solo tuvieramos la alimentacion del USB (500 mA a 5V), no llegaría para
alimentar todo el sistema. Por ello, si queremos usar el TFT BL (355 mA),
tenemos que alimentarlo externamente.
```

Figura 26.- Página 6 del esquemático con el cálculo de consumo de potencia

Para realizar el cálculo de potencia consumida hace falta consultar la hoja de especificaciones (*datasheet)* de todos los componentes. Ahí tendremos que buscar el consumo interno de cada componente, y poner el valor máximo. Se coge este valor máximo ya que al hacer el cálculo de consumo de potencia queremos realizar una aproximación de cuánto podría consumir nuestro circuito en el peor caso posible. En el siguiente cuadro, **figura 27**, tenemos un ejemplo de dónde podemos encontrar el consumo eléctrico de un componente: la interfaz USB-UART. De esta figura cogeremos el valor de corriente de alimentación cuando el componente esté en funcionamiento, y cogeremos el valor máximo, que en este caso serán 18 mA. [15]

Parameter	Description	Min	Typ	Max	Units	Details/Conditions
V_{CC}	V_{CC} supply voltage	3.15	3.30	3.45	V	Set and configure correct voltage range using the configuration utility for V_{CC}.
		4.35	5.00	5.25	V	
V_{CCIO}	V_{CCIO} supply voltage	1.71	1.80	1.89	V	Used to set I/O voltage. Set and configure the correct voltage range using the configuration utility for V_{CCIO}.
		2.0	3.3	5.5	V	
V_{CCD}	Output voltage (for core logic)	–	1.80	–	V	Do not use this supply to drive the external device. • 1.71 V ≤ VCCIO ≤ 1.89 V: Short V_{CCD} pin with the V_{CCIO} pin • V_{CCIO} > 2 V – connect a 1-µF capacitor (Cefc) between the V_{CCD} pin and ground
Cefc	External Regulator voltage bypass	1.00	1.30	1.60	µF	X5R ceramic or better
I_{CC1}	Operating supply current	–	13	18	mA	USB 2.0 FS, UART at 1-Mbps single channel, no GPIO switching at V_{CC} = 5 V, V_{CCIO} = 5 V
I_{CC2}	USB Suspend supply current	–	5	–	µA	Does not include current through the pull-up resistor on USBDP In USB suspend mode, the D+ voltage can go up to a maximum of 3.8 V.

Figura 27.- Características de consumo de la interfaz USB-UART

La FPGA es un caso concreto, ya que a través de la alimentación de 1V2 consume una fija de 6,9 mA. [12] Pero a este consumo hay que añadirle otros 3 mA. El primero es el correspondiente al PLL ya que posee una alimentación diferente. Después tenemos la corriente que consume para alimentar los 4 bancos presentes en el dispositivo. Y, por último, tenemos la corriente que recibe a través de los puertos de 3V3. Esta corriente entrante será la misma que la saliente, por lo tanto, hay que analizar el consumo que tienen los componentes a los que están conectados estos puertos. Los correspondientes a la pantalla TFT tienen un consumo en total, según la hoja de especificaciones de la pantalla, de 150 mA de valor típico y 200 mA de valor máximo. En el caso de los puertos conectados a los LEDs consumirán lo que estos demanden, que con nuestra selección supone 10 mA por componente. Y otro caso particular es nuestro microcontrolador, ya que la corriente de alimentación depende de la frecuencia a la que vaya a funcionar. Como hemos elegido una frecuencia de funcionamiento de 48 MHz, observando su *datasheet* podemos concluir que su corriente de alimentación será de 9 mA. [12]

La forma de dividir este análisis se realiza según las diferentes alimentaciones presentes en nuestro circuito, que en nuestro caso son 5: 1V2, 3V3, 5V, 10V y 5V_EXT. Los reguladores lineales tienen dos consumos diferentes para realizar el cálculo, tienen un consumo propio, muy pequeño, pero también, como la corriente entrante es igual a la

saliente en estos componentes, consumirán la misma potencia que tengan que otorgar a su salida. Por ello, para hacer el cálculo tendremos que sumar todo lo que se consume a la salida de cada regulador lineal y sumarlo a la corriente que consumimos procedente de los 5V de alimentación. Y con la otra alimentación de los 5V externos realizamos la misma operación, pero con un cambio principal, que es que en el *Boost Regulator*, la corriente de salida no es igual a la entrante, por lo que hay que realizar más operaciones para calcular la corriente consumida. Lo primero que hay que hacer es calcular la corriente que necesitamos que salga del componente; en nuestro caso son 200 mA que es lo que consume como máximo el *backlight* de la TFT. A continuación, calculamos la potencia de salida de nuestro circuito sabiendo que el voltaje de salida son 10V. Después calculamos la potencia que necesitaremos a la entrada suponiendo una eficiencia del componente del 90%. Y finalmente, con la potencia que necesitamos en la entrada calculamos la corriente que consume el componente sabiendo que tenemos 5V en la entrada. A continuación, mostramos los cálculos realizados:

$$P_{out} = V_{out} I_{out} \; ; \; P_{out} = 10V * 200mA \; ; P_{out} = 2W$$
$$P_{in} = \frac{P_{out}}{\eta} \; ; \; P_{in} = \frac{2W}{0{,}9} \; ; \; P_{in} = 2{,}23W$$
$$I_{in} = \frac{P_{in}}{V_{in}} \; ; \; I_{in} = \frac{2{,}23W}{5V} \; ; \; I_{in} = 445mA$$

Con estos cálculos podemos ver por qué no podemos alimentar todo nuestro circuito mediante el conector USB, ya que este solo nos proporciona 500mA a 5V, por lo que, visto el cálculo anterior, prácticamente toda la corriente iría destinada a alimentar el *backlight*. Mediante estos cálculos pudimos concluir que la mejor opción de alimentación de nuestro circuito era alimentar el *backlight* mediante una alimentación externa de 5V exclusiva para él y alimentar el resto del circuito mediante el conector USB. Según los cálculos, con el circuito actual se consumirán 368,28 mA del USB, lo que nos deja margen para conectar a los pines de la FPGA o del microcontrolador otros componentes externos que consuman algo de potencia, como un zumbador o algún LED más, por ejemplo.

b) Análisis Térmico

Para concluir el apartado correspondiente al esquemático vamos a comentar de otros cálculos realizados, como es el análisis térmico del circuito, que podemos ver en la página 7 del **anexo 2** y en la **figura 28**. Este cálculo es fundamental para comprobar que ningún componente se va a calentar por encima o de forma cercana a su temperatura máxima de funcionamiento. Estas altas temperaturas pueden afectar al rendimiento y correcto funcionamiento de los componentes y provocar la rotura si se excede la temperatura límite. Esto nos permite, de forma previa a la fabricación, el añadido de soluciones si algún componente se sobrecalienta, como añadir una aleta de cobre para disipar el calor, o cambiar de encapsulado.

Figura 28.- Página 7 del esquemático correspondiente al análisis térmico

Para realizar los cálculos térmicos vamos a necesitar dos parámetros fundamentalmente; el primero es la potencia disipada por cada componente, que ya ha sido calculada en el apartado previo correspondiente al análisis de potencia. El segundo parámetro es la resistencia térmica entre la unión y el ambiente (θ_{JA}), que podemos encontrar en los *datasheet* de cada componente. Esta variable se da en ºC/W, por lo que nos dará el aumento de la temperatura del componente en función de la potencia. El cálculo realizado es muy básico. Inicialmente supondremos que la temperatura ambiente es de 25ºC, por lo que para calcular la temperatura en el interior del semiconductor (T_j) tendremos que multiplicar la resistencia térmica por la potencia disipada por el componente y sumar la temperatura ambiente. Este valor finalmente lo compararemos con la temperatura máxima soportada en el interior del semiconductor y si es próxima o superior supondremos que el componente no funcionará correctamente [18].

$$T_j = T_a + P * \theta_{ja}$$

También hay que tener en cuenta que la resistencia térmica depende del encapsulado, por lo que hay veces que un mismo componente, con un encapsulado no supera la temperatura límite, mientras que con otro sí que lo hace. Además, no todos los *datasheet* nos ofrecen el valor de la resistencia térmica, por lo que no se ha podido realizar el cálculo térmico.

Vamos a poner un ejemplo del cálculo que se ha realizado para obtener la temperatura de un componente, en este caso el regulador lineal de 3V3.

Mirando el *datasheet* observamos que su resistencia térmica es de 163 ºC/W, como podemos ver en el siguiente cuadro, **figura 29**, ya que nuestro componente tiene un encapsulado de tipo SOIC. [16]

TEMPERATURE SPECIFICATIONS

Electrical Specifications: Unless otherwise indicated, all limits apply for V_{IN} = 2.3V to 6.0V.						
Parameters	**Sym**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
Temperature Ranges						
Operating Junction Temperature Range	T_J	-40	—	+125	°C	Steady State
Maximum Junction Temperature	T_J	—	—	+150	°C	Transient
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 8LD 2x3 DFN	θ_{JA}	—	76	—	°C/W	4-Layer JC51-7 Standard Board with vias
	θ_{JC}	—	26	—	°C/W	
Thermal Resistance, 8LD SOIC	θ_{JA}	—	163	—	°C/W	4-Layer JC51-7 Standard Board
	θ_{JC}	—	38.8	—	°C/W	

Figura 29.- Características térmicas del MCP1725

Luego la potencia disipada por este componente será la diferencia entre el voltaje de entrada de salida y el de entrada, multiplicada por la corriente que atraviesa el componente:

$$P = (5V - 3V3)343{,}12mA\,;P = \ 0{,}5833W$$

$$T_j = T_{a+}P * \theta_{ja}\ ;\ T_j = 25ºC + \ 0{,}5833W * 163\frac{ºC}{W};\ T_j = 25ºC + 95{,}1ºC$$

$$T_j = 120{,}1ºC$$

Como vemos en el *datasheet*, la temperatura máxima es de 125 ºC, por lo que según nuestro cálculo nos acercamos excesivamente al valor límite, a pesar de estar dentro de los límites. Por esto, para disipar mejor, colocaremos una aleta de cobre de disipación alrededor del componente. Otro caso especial es el *boost regulator*, ya que para calcular la potencia disipada por el componente tenemos que calcular tres potencias diferentes siguiendo las indicaciones recogidas en su *datasheet*. La potencia disipada se compone de la suma de la potencia de saturación (P_{sat}), la potencia de polarización o *bias* (P_{bias}) y la potencia del *driver* del *switch* (P_{driver}). Cada una de estas potencias se define de la siguiente manera: [14]

$$P_{sat} = V_{CE(SAT)} * I_{SW} * D$$

$$P_{driver} = V_{in} * I_{SW}\frac{I_{CC}}{\Delta I_{SW}}D$$

$$P_{bias} = V_{in}I_Q$$

Una vez calculadas todas estas potencias y habiendo calculado la potencia disipada total, procederemos de la misma manera que hemos visto antes con el regulador lineal para calcular la temperatura de la unión y ver si nuestro componente se calienta de forma excesiva. Como podemos ver en la página del análisis térmico no se calentará excesivamente por lo que no hace falta añadir ningún elemento para disipar.

Después de haber realizado todos los cálculos, concluimos que únicamente el regulador lineal de 3V3 se acercaba excesivamente a la temperatura límite por lo que en la PCB se le añadió una aleta de disipación de cobre. Respecto a los componentes que no pudimos obtener su resistencia térmica, no nos hemos preocupado de que se calienten en exceso ya que la potencia consumida por cada uno de ellos no es muy elevada, por lo que no tendremos riesgo de sobrecalentamiento.

c) Análisis con cámara térmica

Después de la fabricación y montaje del circuito hemos querido comprobar el calentamiento real que tiene cada uno de nuestros componentes y el circuito en general. Para ello, hemos cargado en la FPGA un programa simple que enciende cada uno de los LEDs secuencialmente. El programa cargado en el microcontrolador lo único que hace es recibir el programa del ordenador, guardarlo en una memoria y activar la FPGA, después se queda en estado *IDLE*. También hay que tener en cuenta que no se ha conectado la pantalla TFT, por lo que tampoco hemos conectado los 5 V externos. Además, al no tener conectada la pantalla, la corriente que atraviesa el regulador lineal será muy inferior a la máxima supuesta en los cálculos.

Para comprobar cuanto se ha calentado el circuito en estas condiciones hemos usado una cámara térmica. Con ella, se apunta a la zona a la que quieres tomar la temperatura y mediante un mapa de colores te dice las zonas más calientes (amarillo) y las zonas más frías (morado). Además, te dice la temperatura del punto central al que estás apuntando. En la **figura 30** tenemos una captura mediante la cámara del circuito total, como podemos ver la zona más caliente es la correspondiente al conector USB y la interfaz USB-UART (zona superior derecha), y la zona de la FPGA y el microcontrolador (zona central en azul) no está nada caliente, estando a una temperatura aproximada de 21,8ºC. En los cálculos térmicos realizados se supuso una temperatura ambiente de 25ºC, por lo que tampoco son comparables al no coincidir esta temperatura. Podemos ver también que la zona de los reguladores lineales (zona superior izquierda) está menos caliente que la zona del conector USB. Esto se puede justificar como hemos dicho previamente, al no tener conectada la pantalla TFT, la corriente que atraviesa el regulador lineal de 3V3 disminuye notablemente, por lo que también disminuirá la potencia disipada por este componente, procediendo a un calentamiento mínimo.

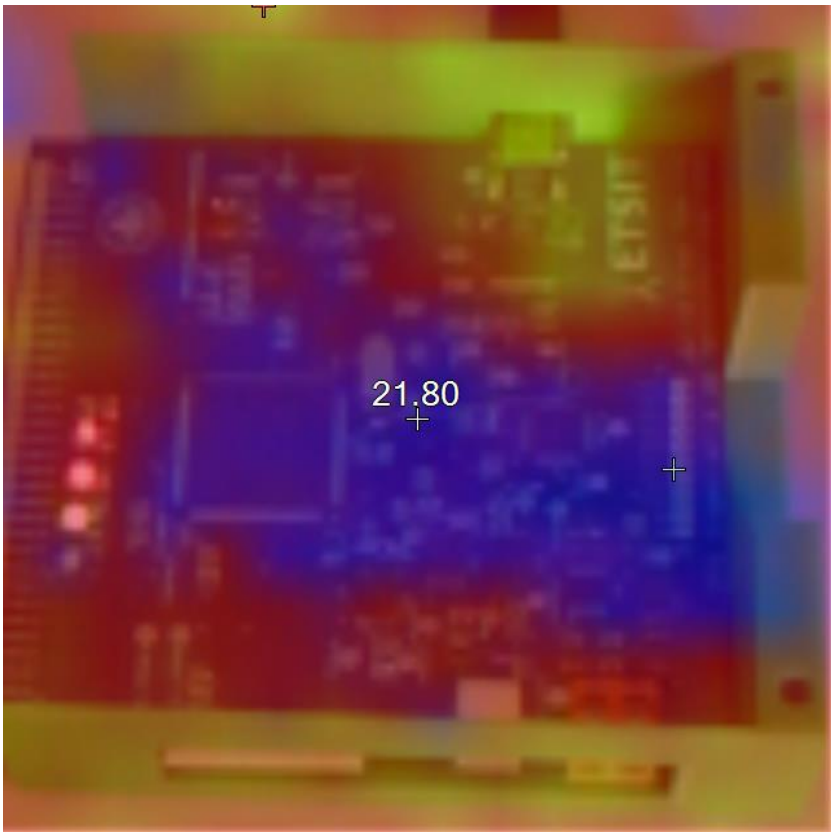


Figura 30.- Fotografía con cámara térmica de todo el circuito

En la **figura 31** podemos ver otra imagen hecha con la cámara térmica, pero esta vez centrándonos únicamente en la zona de la FPGA y los LEDs. Como podemos ver la FPGA no se calienta prácticamente nada, estando a una temperatura de 22,77ºC. Esto cumple los cálculos realizados de forma aproximada, ya que nos daba que solo se calentaría 4ºC respecto a la temperatura ambiente, lo que es un calentamiento ínfimo. En la imagen podemos ver además que hay dos zonas que se calientan más, por un lado, tenemos los LEDs que tienen un calentamiento normal, ya que además de disipar potencia en forma de luz también lo hacen en forma de calor, por lo que su calentamiento es normal. Por otro lado, el elemento más caliente de esta imagen es el cristal de cuarzo, que podemos ver a la izquierda de la FPGA en color amarillo. Su calentamiento también es totalmente normal y dentro de los límites, ya que tiene que estar oscilando continuamente a una velocidad de 12 MHz (frecuencia elegida de nuestro cristal), lo que supone un ligero calentamiento del componente. Los colores ofrecidos por la cámara térmica son relativos a cada imagen, por lo que el color amarillo del oscilador no es preocupante porque lo único que nos dice es que está más caliente que la zona central (que es la FPGA), y como el color tampoco es de un amarillo muy fuerte no es motivo de preocupación.

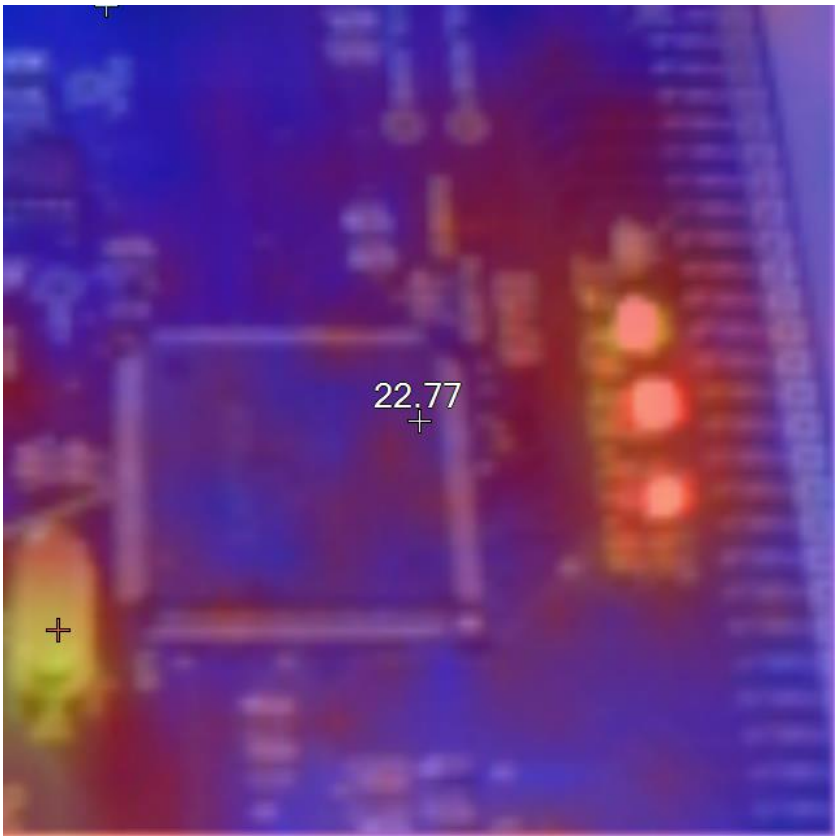


Figura 31.- Fotografía con cámara térmica de la FPGA y los LEDs

Por último, hemos hecho una última foto con la cámara térmica a la zona del microcontrolador y las memorias (parte superior). Como podemos ver en la **figura 32** la zona más caliente de esta imagen es la correspondiente al microcontrolador, con una temperatura de 23,73ºC. Por otro lado, la zona de las memorias está más fría, ya que únicamente usamos una de las memorias para la escritura del programa de la FPGA y su posterior lectura, y después no se vuelve a usar.

Como hemos dicho previamente, los colores que se pueden ver en cada figura son relativos a esta, por ello, no se pueden comparar los colores entre las imágenes. El funcionamiento de la cámara es que detecta la temperatura máxima de la imagen y la mínima y traza una escala de colores entre medias asignando a toda la imagen. Por ello, no podemos comparar las imágenes entre sí, porque tendrán los mismos colores, pero la escala estará definida de una manera diferente.

Finalmente, como conclusión, para los cálculos teóricos realizados en el análisis térmico suponíamos las peores condiciones posibles, con los máximos consumos de corriente. Al definir unos programas muy básicos, es imposible que el calentamiento de los componentes se acerque al valor teórico calculado. Además, respecto a los reguladores lineales, se les añadió en la PCB, como veremos en el siguiente apartado, una aleta de disipación de cobre a cada uno de ellos, por lo que el calentamiento teórico calculado será siempre superior al calentamiento real del componente.

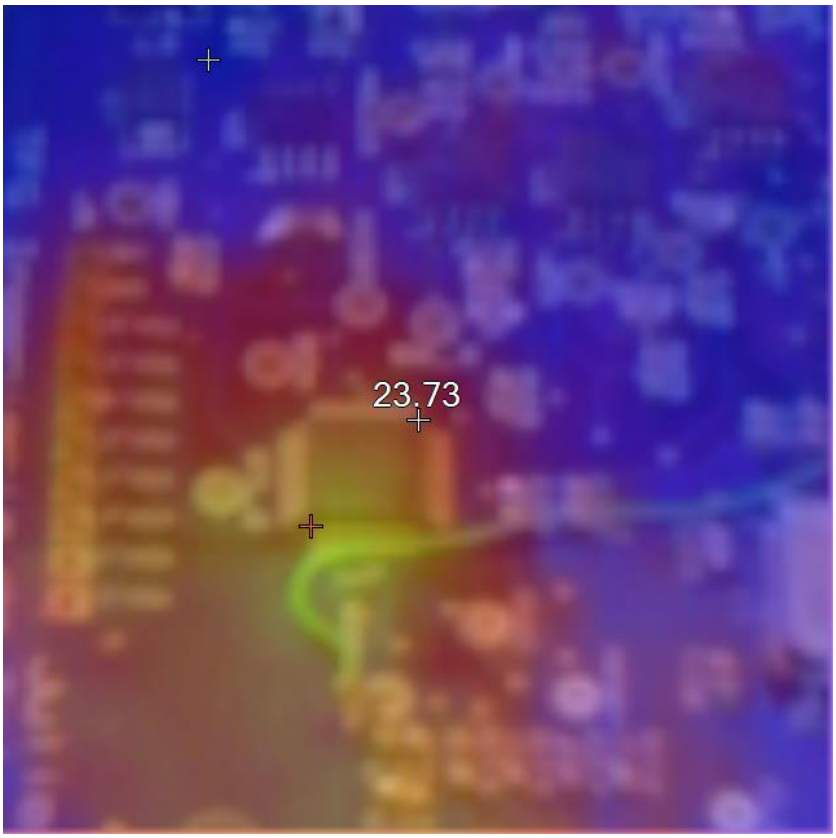


Figura 32.- Fotografía con cámara térmica del microcontrolador

Lo único que podemos afirmar con las fotografías térmicas realizadas es que el circuito nunca se nos calentará de forma excesiva ni peligrosa. Esto se debe, como ya hemos dicho, a que se han añadido aletas de disipación de cobre en los componentes que más se calientan, y que el análisis térmico se ha realizado con los valores máximos, que son poco habituales en comparación con los valores típicos de los *datasheets*. Además, en el diseño de la PCB, que veremos en el siguiente apartado, hemos aumentado de anchura las vías de alimentación para que se produzca menos calentamiento y más disipación a través de estas. Por ello podemos concluir que nuestro diseño es correcto respecto al análisis térmico, ya que ningún componente se va a calentar de forma peligrosa.

PLACA DE CIRCUITO IMPRESO (PCB)

Una vez se ha terminado el diseño del esquemático y se han realizado todos los cálculos de potencia y térmicos, para comprobar que nuestro diseño va a funcionar correctamente, se procede a realizar la placa de circuito impreso o PCB, por sus siglas en inglés, *Printed Circuit Board.* Para ello, lo primero que hay que hacer es acceder a la pestaña de PCB *Layout* de Proteus. El propio programa te sirve de ayuda durante el proceso de diseño, ya que tiene muchas funcionalidades para facilitar el proceso. Por ejemplo, al colocar un componente te dice con qué pin se tiene que conectar cada uno de sus pines, también te notifica si hay errores de diseño.

Para empezar el diseño tenemos que configurar primero las reglas de diseño. En la **figura 33** podemos ver la selección realizada para nuestro circuito. Estas reglas tienen que ser las mismas que las proporcionadas por el fabricante con el que se fabricará la placa, ya que dependen de la maquinaria usada para la fabricación.

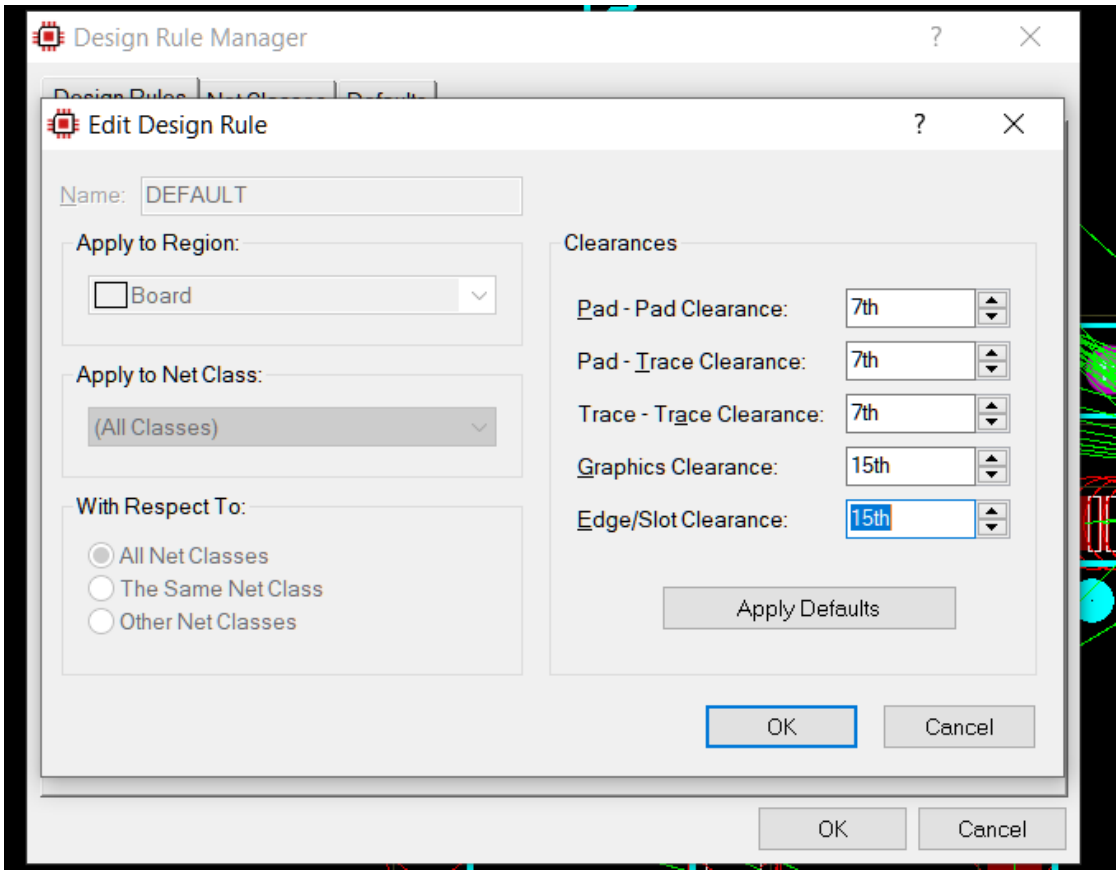


Figura 33.- Configuración en Proteus para realizar la PCB

Para el posicionamiento de los componentes y su rutado existen dos opciones en Proteus, la primera consiste en pulsar el botón de rutado automático y posicionado de componentes automático. Con estas dos selecciones, el propio Proteus posicionará todos los componentes en la placa y los rutará, por lo que Proteus hará todo el trabajo por nosotros, aunque suele ser habitual tener que retocar muchas cosas ya que suele generar numerosos fallos de diseño esta opción. La segunda opción es hacerlo de forma manual, colocando

uno mismo los componentes y procediendo a su rutado. Para la realización de esta placa se ha optado por esta segunda opción.

Para el posicionado de los componentes se ha optado por seguir de forma aproximada los esquemáticos, para que sea más fácil su colocación y encontrarlos posteriormente. En la **figura 34** podemos ver la capa de *Top Silk* de nuestra placa, en ella podemos ver aparte de añadidos, como el escudo de la UVA y de la ETSIT, la colocación de cada uno de los componentes en nuestra placa. Con esto en la fase de montaje podremos cómodamente ubicar donde va cada uno de nuestros componentes. Esto también lo podemos ver en la **figura 35**, correspondiente al *Assembly Plot* que nos dice dónde va ubicado cada uno de los componentes.

Como podemos ver en estas dos imágenes, en la parte inferior derecha se han ubicado los componentes correspondientes a la comunicación entre el microcontrolador y el ordenador, como son la interfaz USB-UART y el conector USB, junto a todos los elementos pasivos necesarios. En la parte derecha tenemos los dos reguladores lineales de nuestro circuito. En la parte inferior tenemos nuestro microcontrolador junto con los pines de salida. Justo a su izquierda tenemos ubicadas las tres memorias Flash junto con los elementos pasivos necesarios para su correcto funcionamiento, descrito todo en los esquemáticos. En la zona izquierda, tenemos todo lo correspondiente a la pantalla TFT, tenemos el conector del *backlight* y el conector principal, junto con el conector de tornillo para alimentarlo externamente, junto con *Boost Regulator*.

Finalmente, en la zona central y la superior tenemos todos los componentes asociados a la FPGA. Casi, justo en el centro tenemos la FPGA, encima tenemos los 4 LEDs conectados a esta junto con las 4 resistencias asociadas a cada uno de ellos. Y en el borde superior tenemos el conector con los pines de salida seleccionadas de la FPGA. Por último, a lo largo de toda la placa se han ubicado puntos de test de nodos que son interesantes de medir para el proceso de búsqueda de errores, o de comprobar el correcto funcionamiento del circuito. Alguno de estos puntos de test, por ejemplo, se corresponden con las alimentaciones (5V, 3V3…) para poder comprobar que sus valores son correctos.

Una vez ubicados todos los componentes se procedió al rutado, que se realizó manualmente a través de las dos caras disponibles, la superior y la inferior. Para el rutado, se usó el mismo tipo de pista para todas las conexiones, exceptuando las correspondientes a la alimentación, que se usaron unas de una anchura superior (20th o 15th si va a pines de componentes) para que tengan menos resistencia y por lo tanto se calienten menos. Después del rutado se procedió al añadido del plano de tierra superior e inferior, además de añadir las dos aletas de disipación de cobre alrededor de los dos reguladores lineales.

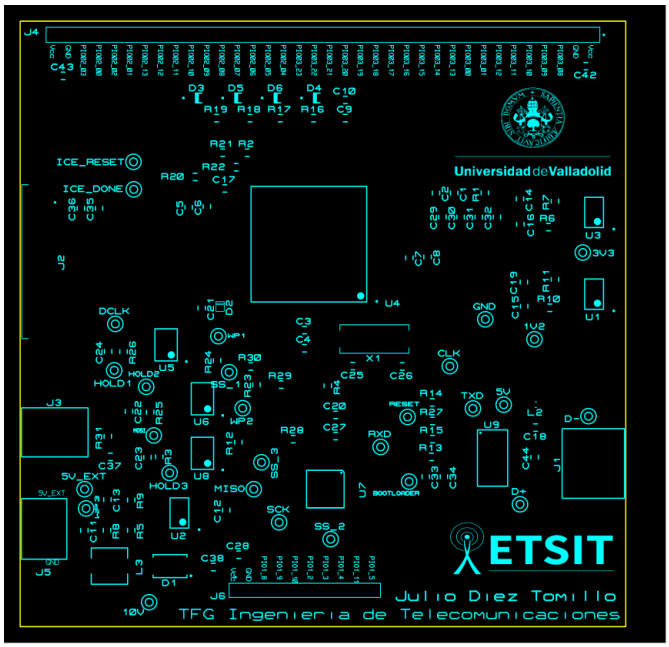


Figura 34.- Capa de "Top Silk"

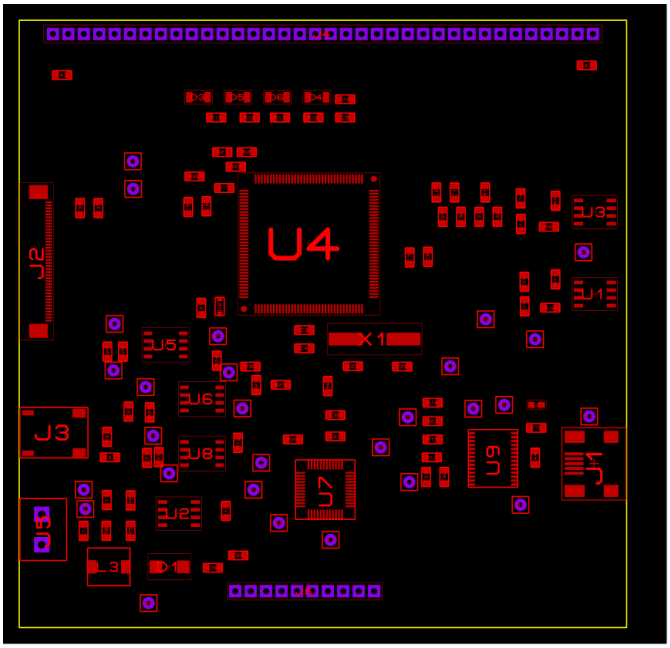


Figura 35.- Diseño de la PCB generado con el "Assembly Plot"

Una vez terminado el diseño de toda la PCB se procede a generar los ficheros GERBER. Estos ficheros contienen lo que necesita nuestro fabricante para la fabricación de nuestra placa, ya que en ellos viene toda la información correspondiente a cada una de las capas para que las máquinas de fabricación puedan realizar nuestro diseño correctamente. Proteus nos los genera automáticamente pulsando el botón correspondiente, después de haber realizado un análisis completo buscando problemas o fallos de diseño. En las siguientes figuras se pueden apreciar algunas de las capas producidas en el diseño de nuestra placa. La **figura 36** corresponde a la capa de *Top paste* de nuestra placa. Las **figuras 37 y 38** corresponden a las capas de *Top Resist* y *Bottom Resist* respectivamente. Y finalmente en las **figuras 39 y 40** tenemos las capas de cobre superior e inferior (*Top and Bottom Cupper)*, respectivamente.

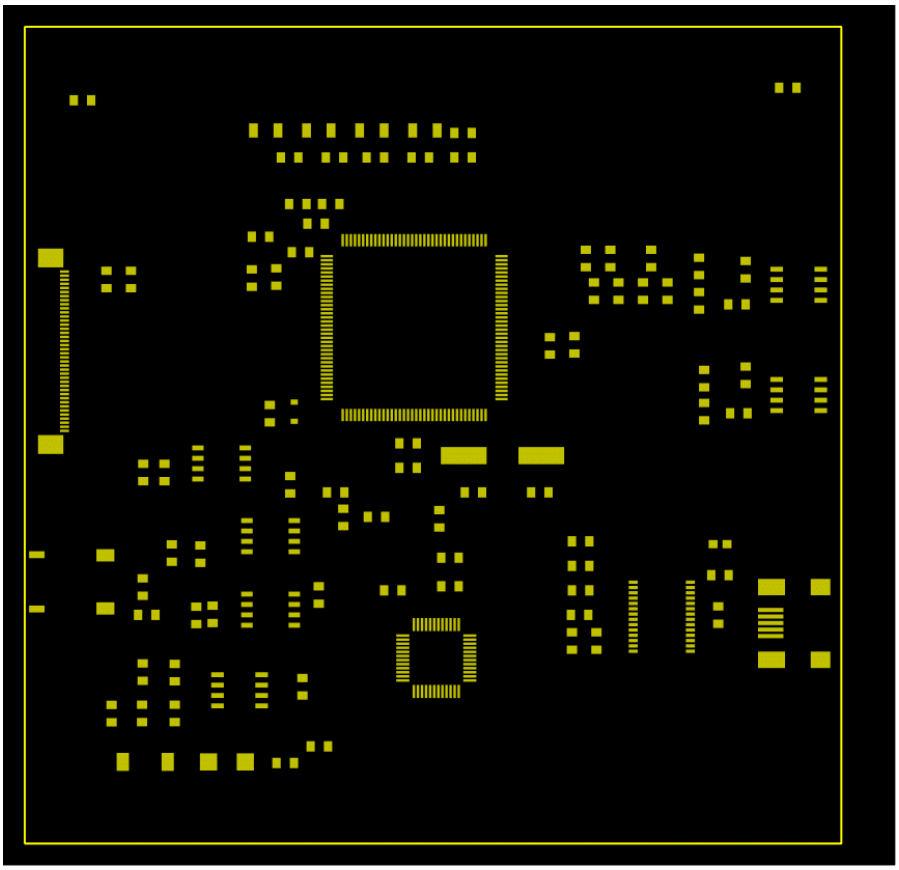


Figura 36.- Capa de "Top Paste"

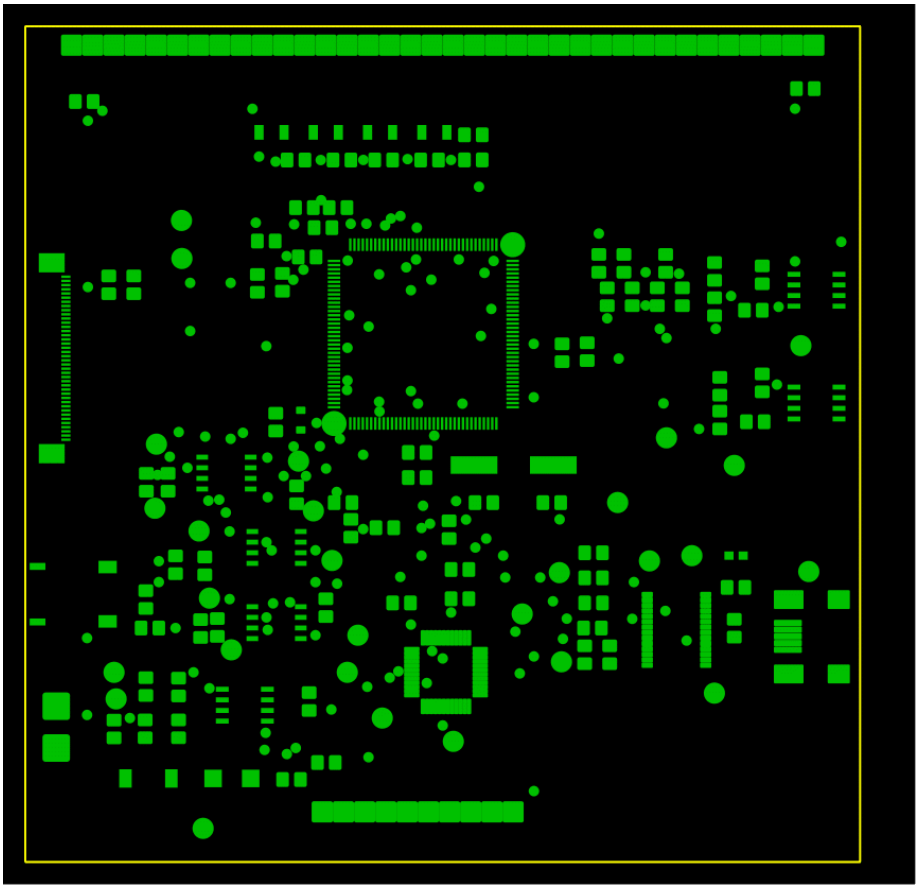


Figura 37.- Capa de "Top Resist"

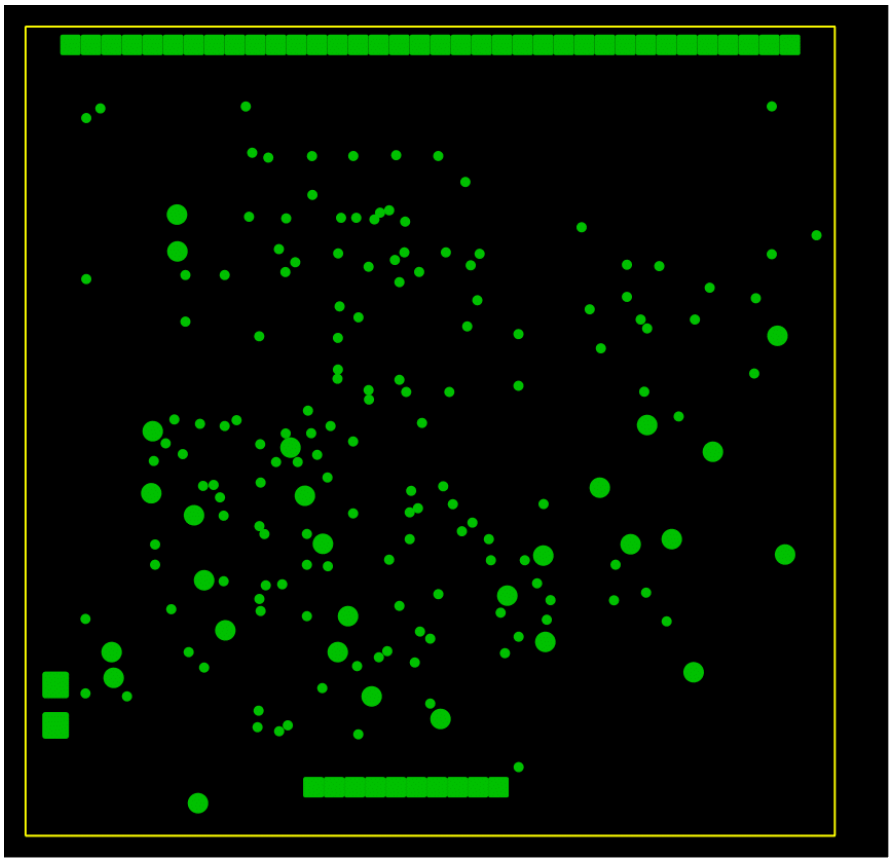


Figura 38.- Capa de "Bottom Resist"

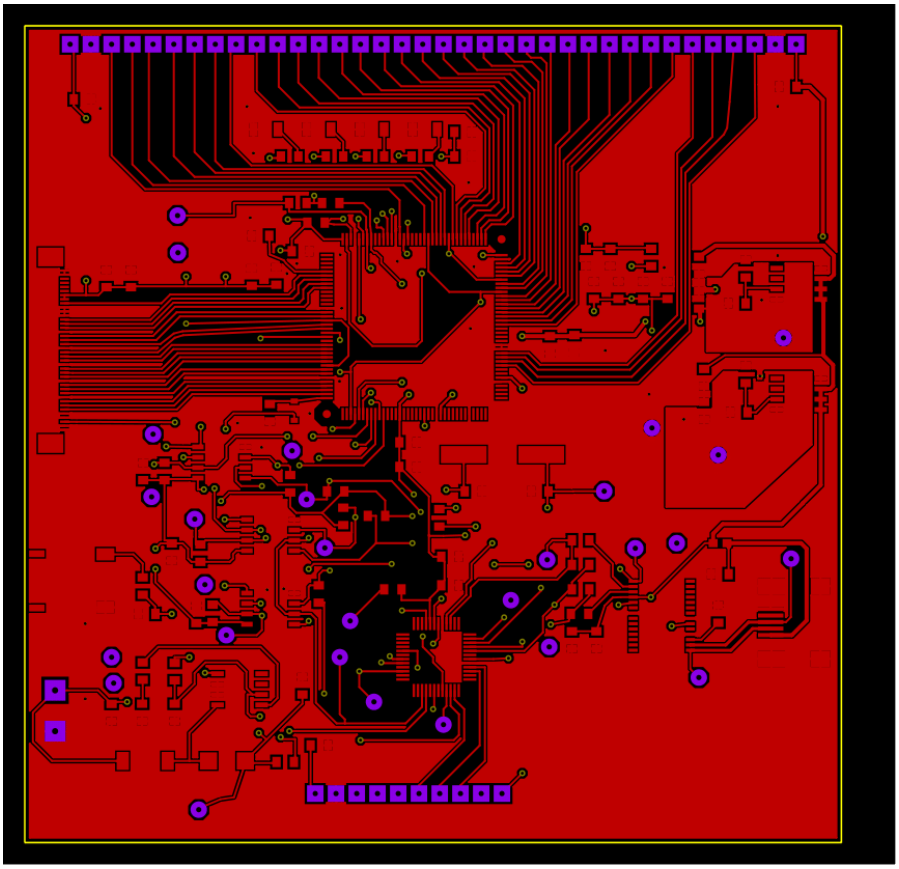


Figura 39.- Capa de cobre superior ("Top Copper")

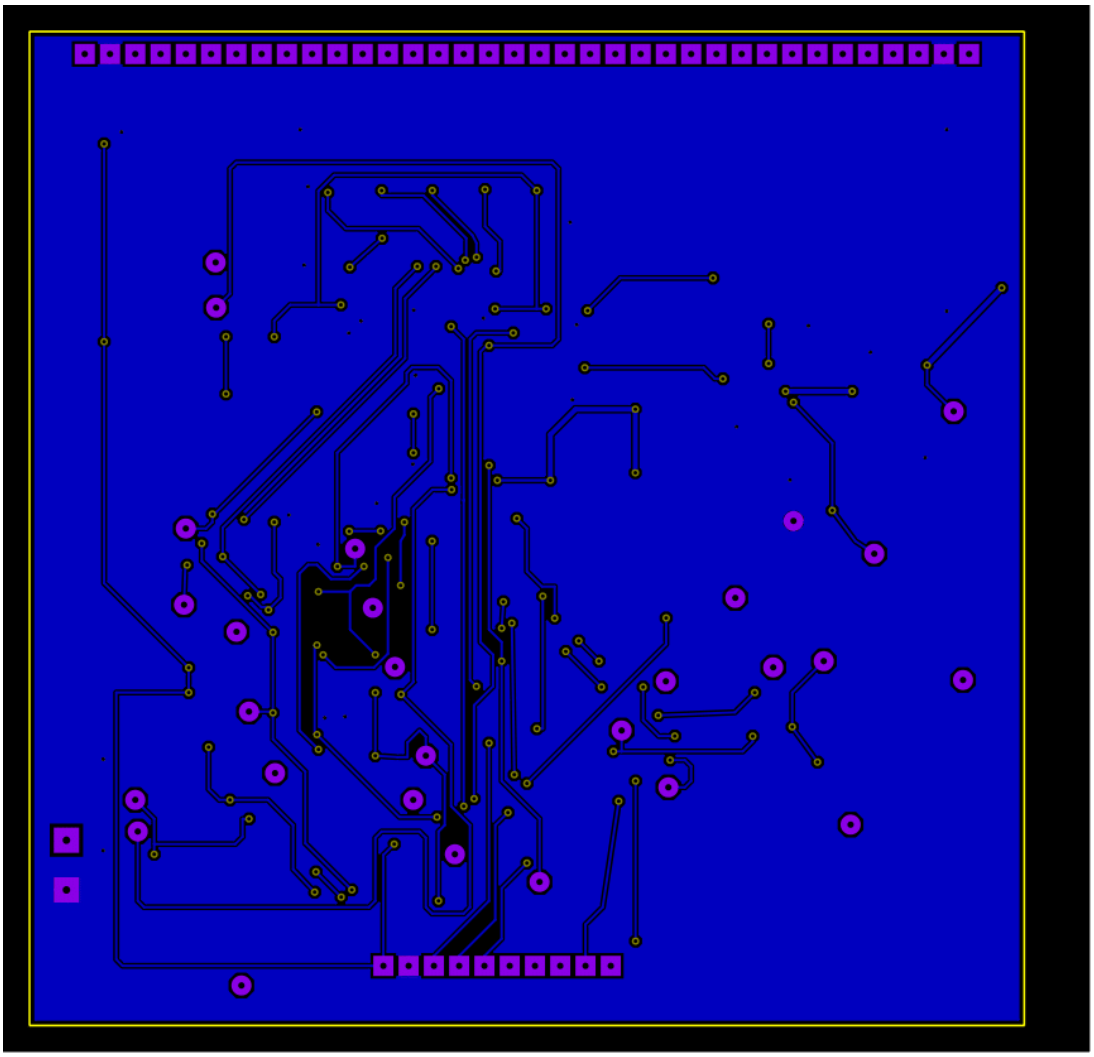


Figura 40.- Capa de cobre inferior (Bottom Copper)

Finalmente, en la **figura 41** podemos ver la superposición de todas estas capas, dando lugar a nuestra placa final diseñada. En ella podemos ver todo lo mencionado anteriormente, como los planos de tierra, las pistas y la ubicación de cada uno de los componentes.

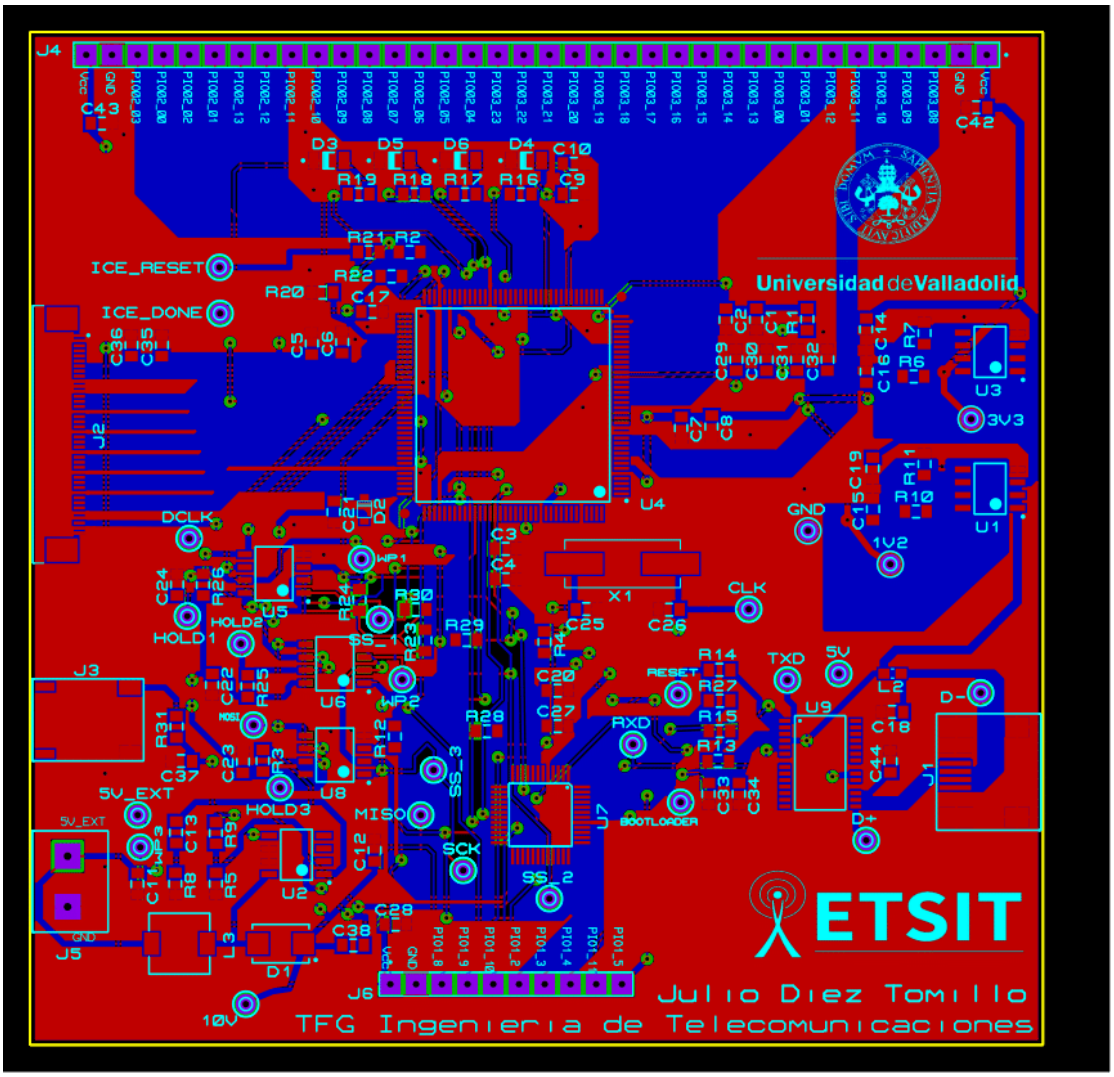


Figura 41.- Superposición final de todas las capas previas

Una vez que se tiene el diseño de la PCB terminado, y los archivos GERBER generados, se procede a su fabricación. Para ello hemos usado una página web denominada PCBWay. En esta web hay que seleccionar las características de fabricación que queremos que tenga nuestra placa. Para la fabricación de la nuestra, hemos configurado primero el tamaño de la placa, que será de 100x100mm, y hemos pedido 10 placas. Únicamente tendremos dos capas, la superior y la inferior; y la placa la hemos fabricado en color azul. El resto de los parámetros se han dejado predeterminados, ya que cambios a mayores suponen un incremento en el precio de las placas, y como llevamos repitiendo desde el principio, el principal objetivo es realizar una placa de bajo precio. Además, con los parámetros predeterminados nuestra placa cumplirá las condiciones requeridas perfectamente, ya que en las reglas de diseño de la PCB de Proteus hemos seleccionado estas. En la **figura 42** podemos ver la configuración elegida para la fabricación de nuestras placas. [19]

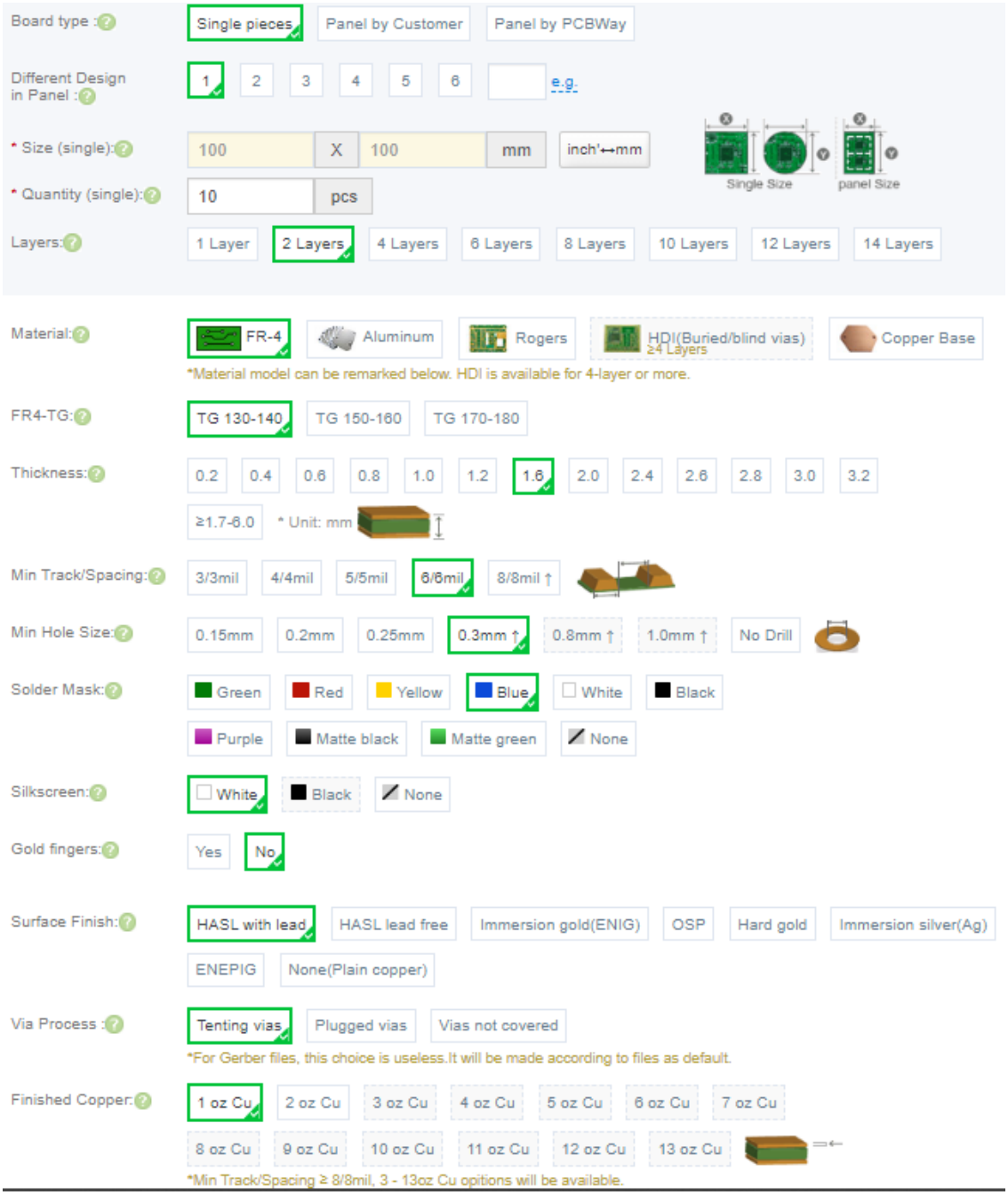


Figura 42.- Características de fabricación elegidas en la web PCBWay

Finalmente, una vez recibidas las placas se procedió a su montaje con todos los componentes necesarios para nuestra placa. En la **figura 43** podemos ver una fotografía de un prototipo de nuestra placa, montado totalmente y preparado para funcionamiento. Como podemos ver en la placa hay algunos cables de color verde, estos se deben a errores de diseño en el esquemático que hubo que solucionar. Todos estos errores y sus soluciones vienen recogidos en el **anexo 7**, correspondiente a los errores de diseño de nuestra placa.

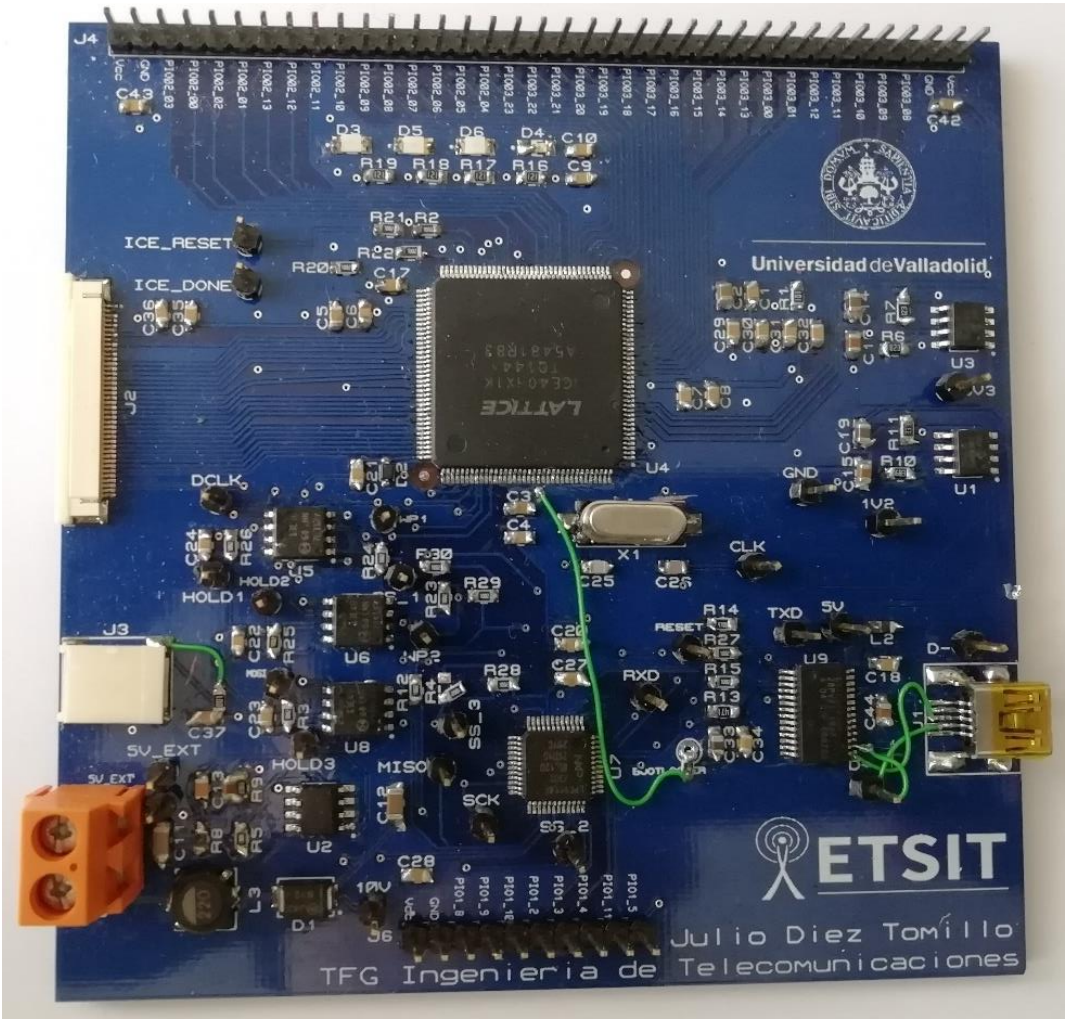


Figura 43.- Circuito final fabricado y montado

MODELO 3D

Después de la realización del esquemático y de toda la PCB se procedió a realizar el modelo 3D de nuestra placa en Proteus. Para ello hubo que buscar los diferentes componentes e irlos asociando a cada uno de los componentes de nuestro circuito, y ubicándolos correctamente. La realización del modelo 3D tiene una doble motivación, la primera es para poder ver cuál va a ser el resultado final de nuestra placa antes de ser fabricada, para poder ver si estéticamente nos convence, o hasta para poder detectar algún fallo.

La segunda motivación es de venta, ya que un modelo 3D siempre es muy útil para poder enseñar a posibles clientes previamente a la fabricación. Con esto se puede enseñar como quedaría la placa finalmente después de todo el desarrollo, por ello, el diseño del modelo 3D final es muy importante para poder enseñar como sería la placa una vez fabricada.

El modelo 3D de la placa se puede ver en la **figura 44**. Si comparamos esta figura con la **figura 43**, en la que veíamos la placa ya fabricada, podemos ver unas pequeñas diferencias pero que se corresponden básicamente a colores diferentes. Esto se debe a que la placa se puede fabricar en muchos colores, pero nosotros para la fabricación elegimos el color azul, frente al verde que se ve en el modelo 3D. Después algunos componentes también pueden ser de color diferente, pero esto ya depende del modelo seleccionado, pero el componente será el mismo, lo único que cambia es su color.

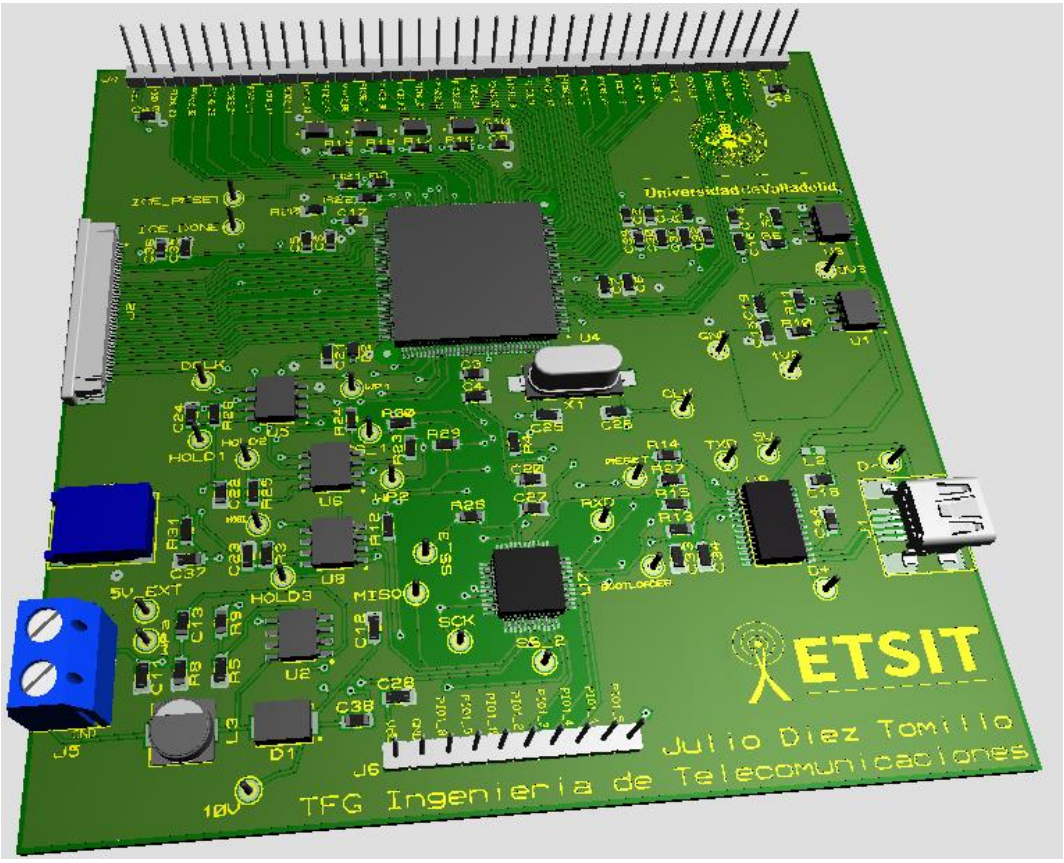


Figura 44.- Modelo 3D de nuestro circuito

CAJA

Finalmente, tras todo el desarrollo previo de la placa se diseñó como añadido final una caja para la protección de la placa. La caja tenía que cumplir una serie de requerimientos como tener diferentes aperturas para las conexiones. Por eso, en el lateral derecho tenemos una pequeña apertura para la conexión con el conector Micro-USB, en la parte inferior tenemos una apertura para la conexión con los pines del microcontrolador. En la parte izquierda tenemos una gran apertura para la conexión de los elementos para la pantalla TFT, como el conector de tornillo, el conector del *backlight* o el conector general de la pantalla. Y por último tenemos la parte delantera totalmente abierta para poderse usar fácilmente los pines de la FPGA. Además de la caja se diseñó una tapa, que se conecta a la caja a través de dos orificios como podemos ver en la **figura 45**.

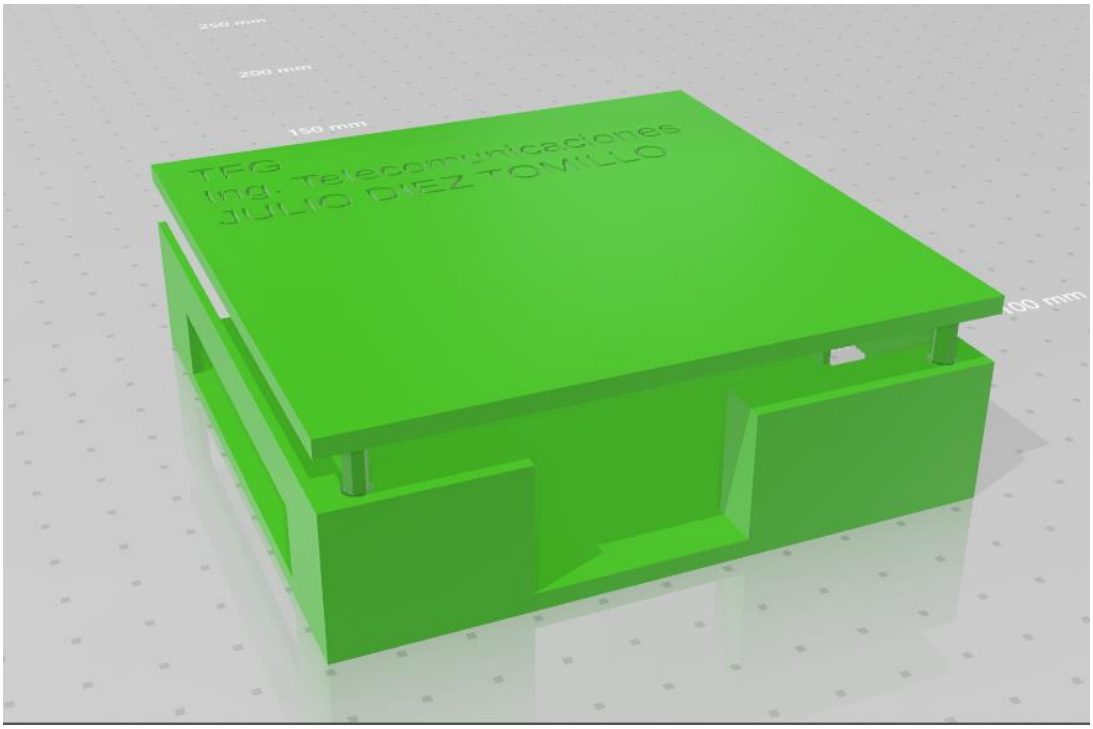


Figura 45.- Modelo 3D de la caja para la placa

Después se diseñó una segunda caja en la que se cambiaba la tapa, que en este caso se convertía en una plataforma para poder sujetar ahí la pantalla TFT. Con esta caja podemos llevar la placa junto con la pantalla y tenerlos interconectados en todo momento. Esta segunda caja la podemos ver en la **figura 46**.

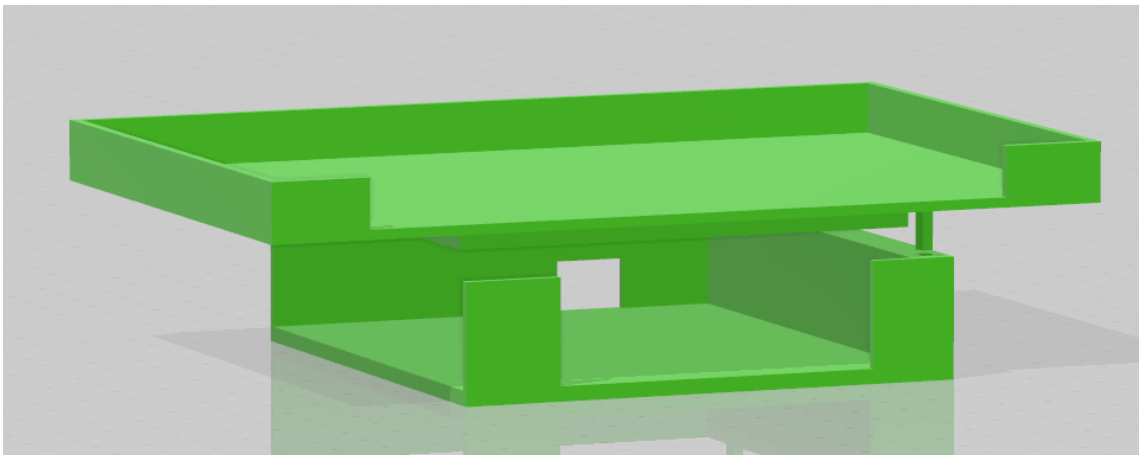


Figura 46.- Modelo 3D de la caja para la placa incluyendo una plataforma para la pantalla TFT

Finalmente, para cerrar este capítulo presentamos en las **figuras 47 y 48** la placa final desarrollada, incluyendo la caja básica fabricada con una impresora 3D con y sin tapa.

Figura 47.- Resultado final del proyecto con caja y la tapa puesta

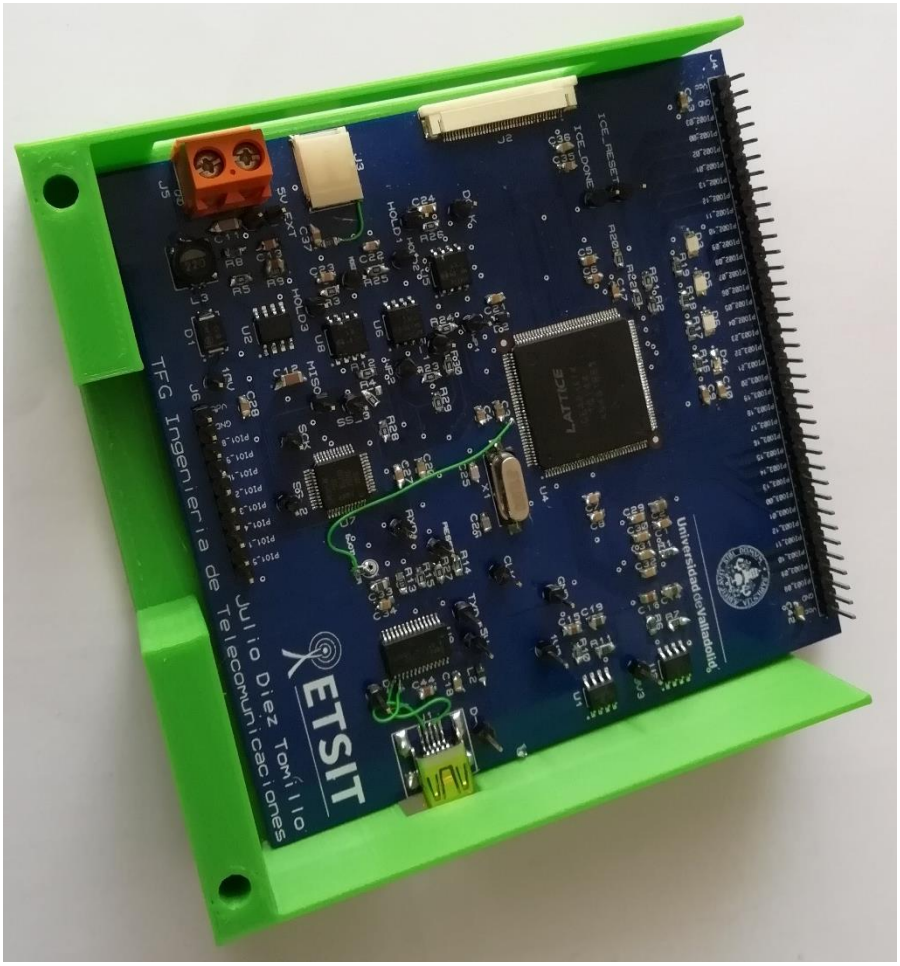


Figura 48.- Resultado final del proyecto con la caja sin la tapa

CAPÍTULO IV

DESARROLLO SOFTWARE

Después de haber terminado todo el desarrollo hardware y habiendo montado nuestra placa, solucionando los problemas y fallos encontrados, el siguiente paso es implementar el firmware. Dentro de nuestra placa tenemos que programar dos componentes, el microcontrolador desde el ordenador a través del conector USB, y la FPGA, que carga su programa de una de las memorias presentes en nuestro circuito.

Este capítulo se compone de dos secciones principales. En la primera hablaremos sobre el procedimiento para la programación de los componentes, es decir, el protocolo que hemos desarrollado para programar el microcontrolador, y como el microcontrolador logra grabar el programa en la FPGA para que funcione esta.

Y en la segunda parte hablamos sobre el desarrollo del programa para la FPGA, esta se realiza mediante un programa de software libre en el que podemos programar de forma gráfica, mediante bloques, cualquier circuito que queramos e implementarlo después con nuestra FPGA.

PROCEDIMIENTO

Para realizar la programación de la FPGA hemos tenido que desarrollar un protocolo. Para ello, hemos tenido que desarrollar dos códigos diferentes. El primero de ellos se graba en el microcontrolador y será el programa que ejecute este. El otro código se ejecutará en el propio ordenador. El diagrama de flujo de este protocolo lo veremos más adelante en este capítulo.

Lo primero que tenemos que hacer es cargar el programa main.c que podemos ver en el **anexo 4** en el microcontrolador. Después ejecutamos el código del ordenador, main.cpp, que vemos en el **anexo 5**. Para ejecutar este código podemos modificar el valor de varios parámetros. Esto se hace añadiendo una letra clave en la línea de comandos y seguido el valor de la variable. Los parámetros que se puedan cambiar son:

- **-m:** Indicamos la memoria en la que queremos que se grabe lo que vamos a enviar, es un entero. Por defecto se graba en la memoria número 3.
- **-a:** Indicamos la dirección de la memoria en la que queremos que se empieza a grabar lo que enviamos, es un entero. Por defecto se empieza en la dirección 0.
- **-f:** Indicamos el archivo que queremos grabar en la memoria, es una cadena. Por defecto se graba el archivo de nombre “hardware.bin”.

Un ejemplo de ejecución del programa en el ordenador sería: `upload_fpga.exe -m 1 -f "temp.bin" -dir 5000.` Donde hemos ejecutado el programa seleccionando la memoria 1, la dirección 5000 y el fichero “temp.bin”. En la **figura 49** podemos ver el

resultado de la ejecución del programa en el que solo hemos cambiado el fichero a escribir.

```
C:\Users\julio\OneDrive\Escritorio\TFG\SOFTWARE\SW_PC>upload_fpga.exe -f "temp.bin"
Archivo a enviar: temp.bin
```

Figura 49.- Resultado ejecución del programa upload_fpga.exe cambiando el archivo a leer

Como podemos ver lo primero que hace el ordenador es abrir el puerto Serie al que está conectada la placa, si no se consigue, se finaliza el programa. Después abrimos el fichero y leemos el número de caracteres que contiene y su contenido. Después enviamos una trama al microcontrolador definida como: Imdddnnn. El primer carácter es la letra I, correspondiente a la inicial de la palabra Inicio o *Init*. El segundo carácter es el número de memoria en el que queremos escribir el programa. Los siguientes tres caracteres nos dicen la dirección de memoria en la que queremos empezar a escribir. Por último, los últimos tres caracteres nos dicen el número de bytes que tenemos que escribir en la memoria, es decir, el número de bytes que vamos a enviar al microcontrolador. El microcontrolador lee esta trama, y si lo ha leído correctamente envía la letra O, de *OK*, al ordenador para notificar que todo ha ido bien. A continuación, el microcontrolador borra el espacio necesario en la memoria para poder escribir el programa. Hay tres formas de borrar la memoria, borrando subsectores, sectores o toda la memoria. Se realiza una de las tres operaciones en función del tamaño del fichero que vayamos a escribir.

Una vez preparada la memoria se envía desde el ordenador el fichero en bloques de 256 B ya que es el tamaño de las páginas de nuestra memoria. Cada vez que el microcontrolador recibe un bloque, lo escribe en la memoria y si ha sido exitoso lo notifica al ordenador mandando la letra O para que envía los siguientes 256 B. Una vez escrito todo en la memoria, el microcontrolador comprueba que se ha escrito todo correctamente leyéndolo. Después activa la FPGA activando el pin de *Reset* y finalmente enviamos al ordenador el carácter ESC para notificarle que se ha realizado todo correctamente por lo que puede cerrar el programa. Después de esto el microcontrolador continuaría ejecutando el resto de su código (si lo tuviera) pudiendo realizar las funciones que le indicáramos.

En la **figura 50** tenemos lo obtenido por consola cuando grabamos un programa en la FPGA denominado *Contador.bin*

```
Archivo a enviar: Contador.bin
Abierto
Memoria: 3
Archivo: Contador.bin
Direccion: 0
Conectado
Tamano archivo: 32220
Trama enviada
.
Recibido el OK se envia archivo
................................................................................................................
......Se ha terminado de enviar el archivo
Checking.
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
................................................................................................................
.....................................................
Xored checksum: fe

Run FPGA0
```

Figura 50.- Resultado obtenido por consola al cargar un programa en la FPGA

Como hemos mencionado antes, en las **figuras 51 y 52** tenemos dos diagramas de flujo que nos muestran el funcionamiento de este protocolo. El primero es el diagrama de flujo del código ejecutado en el ordenador. Mientras que la segunda figura nos muestra el diagrama de flujo del programa que se ejecuta en el microcontrolador.

La principal función de este protocolo es grabar el programa que va a cargar la FPGA en la memoria número 3, pero con él podemos grabar cualquier fichero en las tres memorias de nuestra placa, pudiendo darnos incontables posibilidades a la hora de implementar cosas con nuestra placa. Si lo que queremos es grabar el programa de la FPGA usaremos los parámetros por defecto, pudiendo cambiar el nombre del fichero que queremos grabar si lo hemos llamado de una manera diferente. Este programa se grabará en la memoria número 3, empezando en la dirección 0, que es de donde lee la FPGA el programa una vez la activamos.

Para que la FPGA lea el programa de la memoria, lo primero que tenemos que hacer es quitar el bloqueo en el que se encuentra levantando el pin de *Reset* de esta. Después la FPGA intentará leer el programa de la memoria 3 y si lo consigue y carga correctamente el programa activará el pin *CDone* al acabar. Con esto sabrá el microcontrolador que la FPGA ha sido configurada correctamente y podremos continuar ejecutando el resto del código.

Mediante el código que cargamos al microcontrolador (**anexo 4**), también controlamos el reloj que usa la FPGA para funcionar. Ya que la señal de entrada de reloj de la FPGA es la señal *CLKOut* de nuestro microcontrolador. Esta señal es configurable, por lo que podemos modificar la velocidad a la que funciona el FPGA, pudiendo modificarla, según la velocidad a la que queremos que funcione.

Una vez visto cómo funciona el protocolo para grabar el programa de la FPGA en las memorias, vamos a ver ahora como podemos generar el fichero que contiene el programa y que características tiene este.

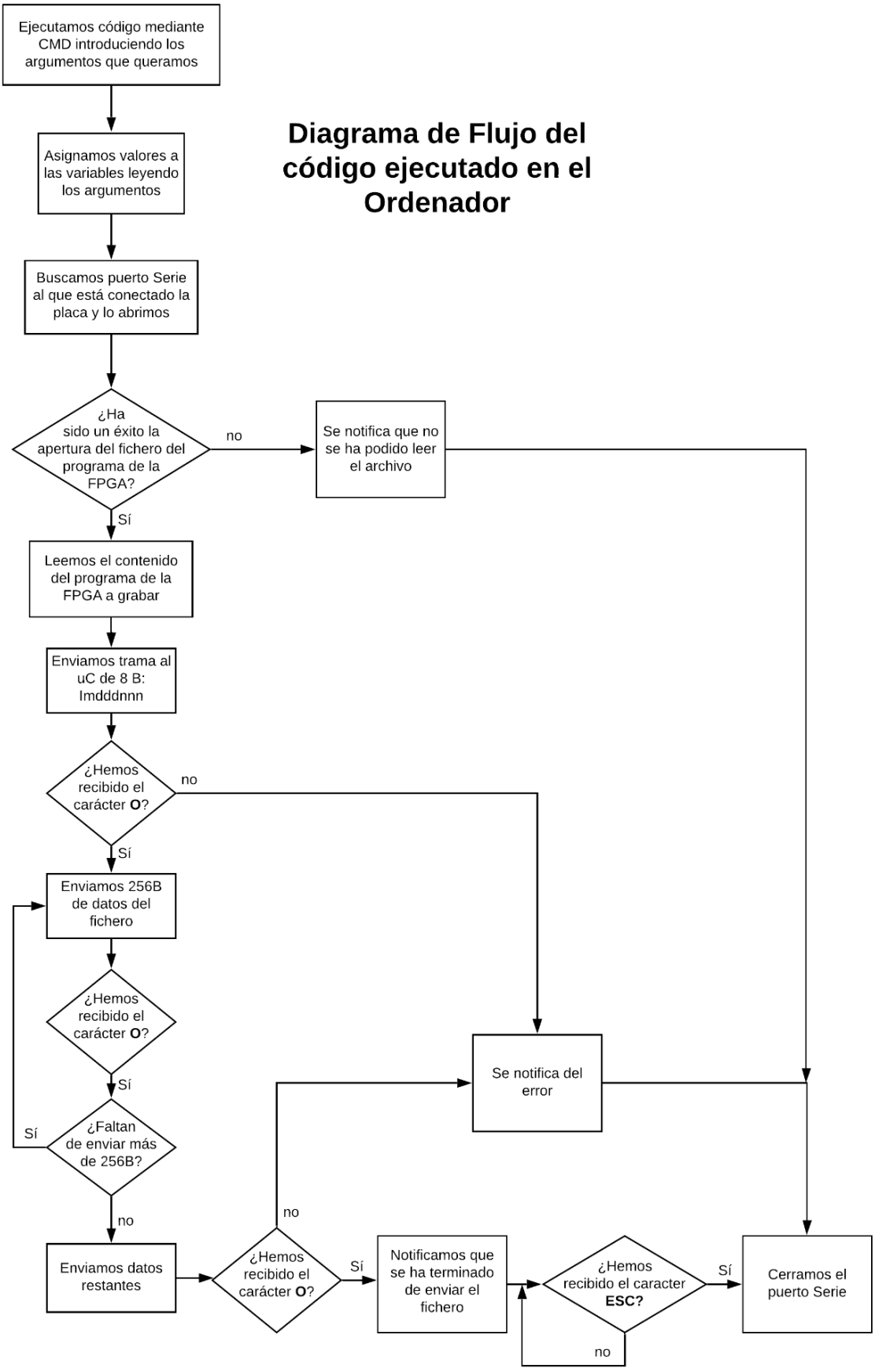


Figura 51.- Diagrama de flujo del código ejecutado en el ordenador

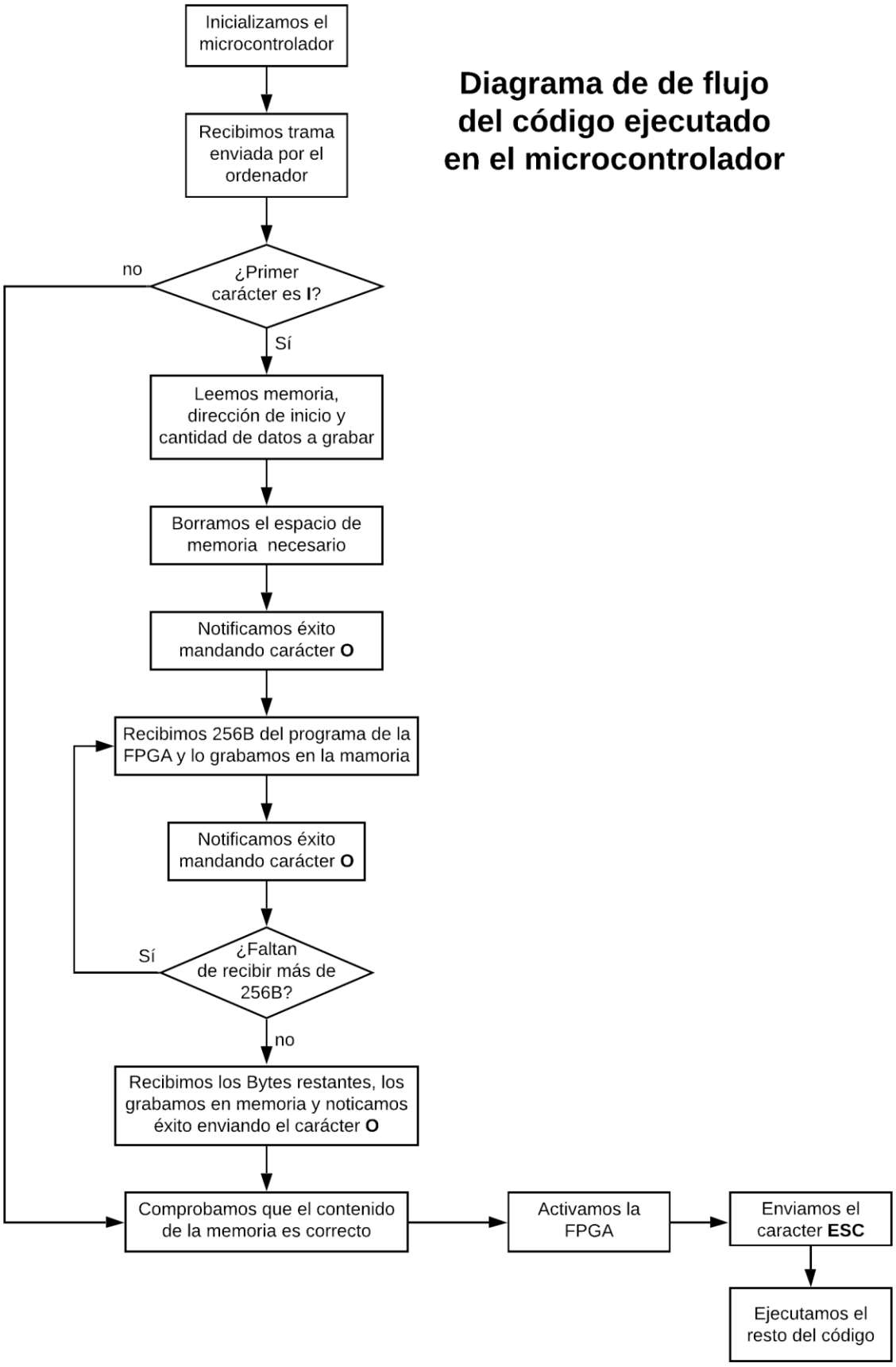


Figura 52.- Diagrama de flujo del código ejecutado en el microcontrolador

PROGRAMACIÓN FPGA: ICESTUDIO

En esta sección vamos a hablar sobre el programa que grabamos en la memoria para que lo ejecute la FPGA. Este programa lo creamos mediante el programa de software libre ICEStudio. A continuación, vamos a ver un vistazo general de este programa y las ventajas que tiene. Luego veremos cómo podemos añadir nuevas placas al programa, para poder introducir la nuestra. Después veremos los componentes disponibles en este programa, y que podemos diseñar con ellos. Y finalmente veremos el programa que hemos creado para comprobar el correcto funcionamiento de la FPGA.

a) Visión General de IceStudio

ICEStudio es un programa para diseño y síntesis de circuitos digitales en FPGAs libres. El programa es de software libre, por lo que no cuesta nada la descarga del programa. En la **figura 53**, podemos ver la interfaz del programa. En la parte inferior izquierda podemos ver el nombre del archivo que estamos editando, como no le hemos puesto nombre todavía sale: "Sin título". En la parte inferior derecha sale la placa que estamos utilizando. Para poder usar nuestra placa tendremos que introducirla en el programa, que veremos cómo hacerlo en el siguiente apartado. [3]

En la parte superior derecha tenemos los bloques digitales que podemos añadir para diseñar nuestro circuito. Inicialmente vienen muy pocos bloques y muy simples, a partir de ellos, se pueden hacer módulos más complejos y guardarlos. Otra forma fácil es añadir colecciones como la que podemos encontrar en internet creada por Obi Juan [20]. En ella podemos encontrar muchas módulos nuevos para no tenerlos que crear de inicio. También se pueden crear módulos escribiendo el código en HDL, como por ejemplo Verilog.

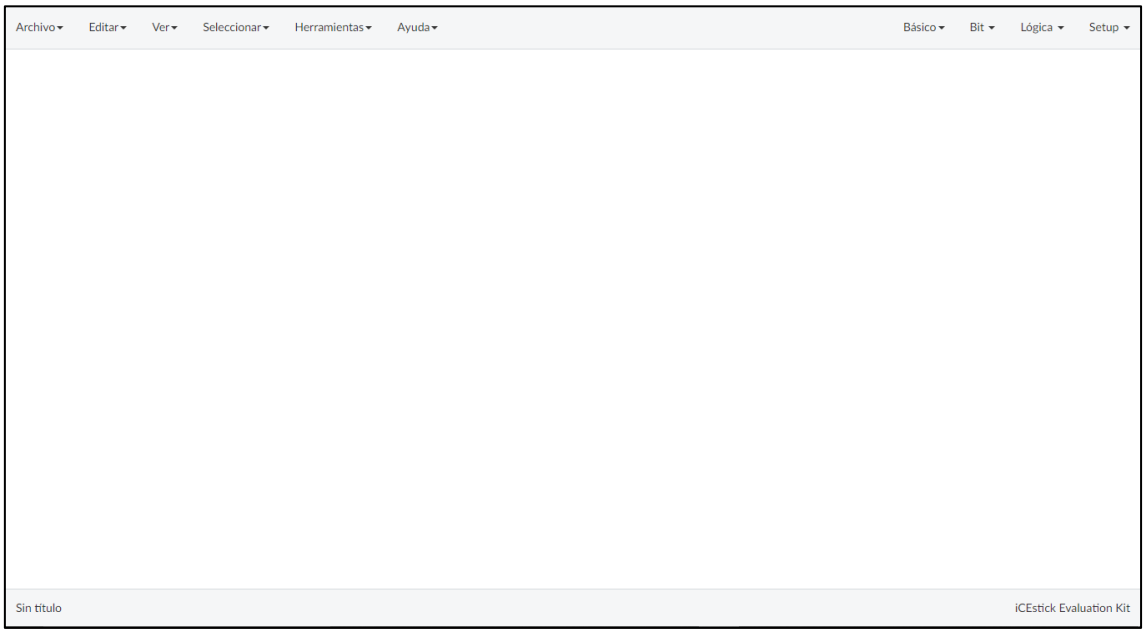


Figura 53.- Interfaz de IceStudio

En la parte superior izquierda tenemos otros menús diferentes. Por ejemplo, en *Archivo* podemos guardar el proyecto en el que estamos trabajando o exportarlo en diferentes

formatos, como Verilog o *BitStream*. Después en *Ver*, podemos encontrar las características de nuestra placa o los pines que posee. También podemos ver las propiedades que tiene la colección que estamos usando. En *Seleccionar*, podemos seleccionar la placa y la colección que queremos usar. Finalmente, en *Herramientas* podemos Verificar, Sintetizar o Grabar nuestro proyecto, aunque esta última función no la usaremos. También en esta pestaña se puede añadir el *Toolchain*, los *Drivers* o nuevas colecciones.

ICEStudio funciona sobre HDL, es decir, todos los bloques y módulos que usamos están escritos en ese lenguaje, aunque los veamos de forma gráfica. Así que cuando diseñamos un circuito en el programa lo único que hacemos al pulsar el botón de sintetizar es convertir ese circuito en un código de HDL. Lo que haremos nosotros, es exportar el fichero como BitStream y será lo que grabaremos en la memoria para que cargue la FPGA.

b) Añadir nueva placa

Una vez visto de forma general el programa, el siguiente paso es introducir nuestra propia placa dentro del programa, ya que inicialmente solo están las placas comerciales como la IceZUM Alhambra o el iCEstick. Introducir nuestra propia placa es muy importante ya que podremos poner a cada pin el nombre que hemos usado y así facilitar su programación. Por ejemplo, es mucho más cómodo iluminar un LED sabiendo que su nombre es D2, que tenerse que aprender el número de pin al que está conectado.

Para ello, lo primero que tenemos que hacer es acceder a los ficheros del programa. Allí accederemos a una carpeta denominada *Boards*, allí hay un fichero *.json* que contiene el menú de FPGAs, por ello, introducimos el nombre de nuestra placa en la categoría de FPGAs HX1K. El siguiente paso es crear allí una nueva carpeta con el nombre de nuestra placa. En esta carpeta tiene que haber 4 archivos diferentes. El primero se denomina *info.json*, y contiene la información general de nuestra placa, como por ejemplo, el nombre, el número de LUTs, PIOs… y se podría incluir un enlace a su datasheet. El siguiente fichero es *rules.json*, que contiene las reglas de la placa. Estas reglas por ejemplo es decir en que pin está conectado el reloj de entrada o que pines deben tener siempre un valor determinado. Después tenemos otro fichero denominado *pinout.pcf* que podemos ver en el **anexo 6**. En este fichero como podemos ver, se establece que pines son de entrada o de salida, que número de pin es y el apodo o nombre que le ponemos. Por ejemplo, veamos esta línea: `set_io --warn-no-port D3 56  # output.` En ella estamos declarando un pin como salida. Y a este pin que tiene el número 56, le ponemos de apodo D3 para que sea más fácil su identificación. Finalmente, el último fichero que tenemos que crear es *pinout.json*. Pero este fichero no hay que escribirlo, ya que mediante el ejecutable generator.py, que requiere tener instalado *Python*, generamos automáticamente este fichero *json* a partir del fichero previamente creado *pcf.*

Una vez hemos seguido estos pasos, podemos iniciar el programa ICEStudio, y si vamos a la pestaña Seleccionar y ahí a Placa, podemos ver que ya está incluida nuestra placa, como se ve en la **figura 54**.

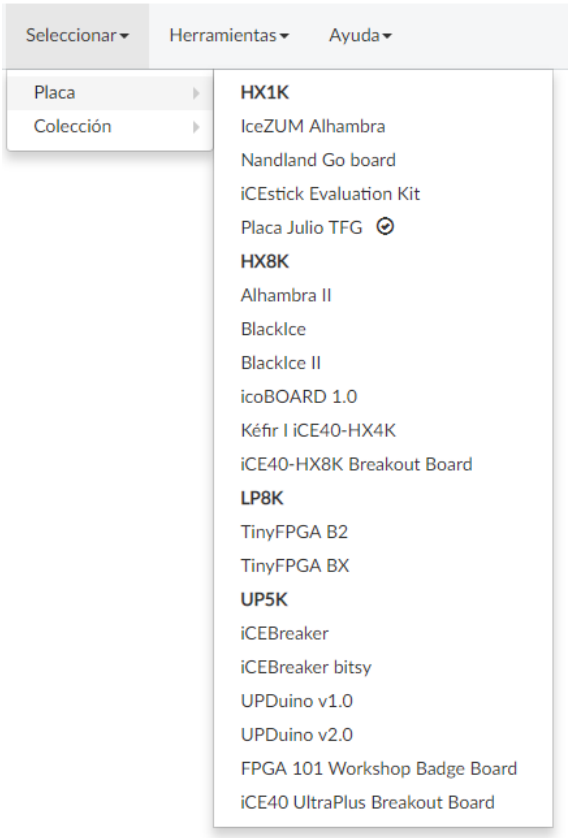


Figura 54.- Selección de nuestra placa en el programa

Finalmente, una vez seleccionada nuestra placa, podemos ver en la **figura 55**, como en la parte inferior derecha ya sale el nombre y sus características. Además, hemos introducido un bloque de entrada llamado Reloj, al que hemos asignado el pin de reloj (CLK) y un bloque de salida denominado LED, al que hemos asignado el pin que hemos configurado antes denominado D4.

Figura 55.- Interfaz de IceStudio habiendo seleccionado nuestra placa

c) Componentes Disponibles

Una vez que ya tenemos nuestra placa añadida al programa para poder trabajar con ella, pasamos a ver que componentes o módulos podemos añadir a nuestro programa. Para ello, lo primero que hacemos es descargar una colección creada por ObiJuan denominada "Collection-Jedi-Master". En esta colección tenemos añadidos numerosos módulos que nos simplifican y facilitan el trabajo, ya que si no tendríamos que haberlos creado y añadido nosotros mismos. [21]

En esta colección tenemos numerosos componentes. Podemos ver todas las categorías que contiene en la **figura 56**.

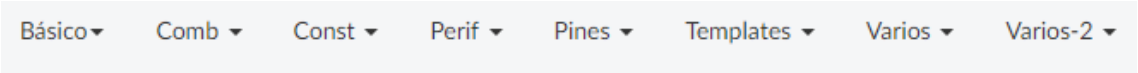


Figura 56.- Categorías que tiene la colección usada

A continuación, vamos a analizar más a fondo cada uno de estos menús para ver las posibilidades que nos otorga esta colección en concreto.

En la **figura 57**, tenemos seleccionado el menú con los componentes denominados básicos, ahí por ejemplo podemos marcar algunos pines concretos de nuestra placa como entradas o como salidas, para sacar valores o leer valores. También podemos añadir valores de constantes o crear una memoria. Para estas memorias tendremos que introducir a mano los valores que va a tomar cada una de las direcciones, pudiendo seleccionar introducir los valores en binario, decimal o hexadecimal. También podemos añadir código para crear un módulo o un bloque escribiéndolo en un lenguaje de HDL como puede ser Verilog. Finalmente podemos añadir texto a nuestro proyecto para mostrar información relevante que queramos añadir.

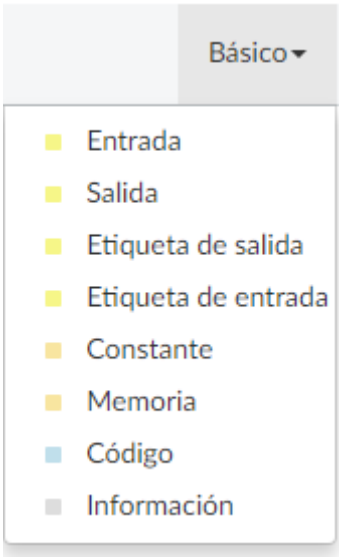


Figura 57.- Componentes del menú Básico

En la **figura 58** tenemos el menú que contiene los bloques combinacionales. Aquí por ejemplo podemos añadir codificadores y decodificadores de diferentes tipos. También tenemos multiplexores y demultiplexores de múltiples bits. Finalmente nos encontramos con una categoría en la que nos encontramos con muchas puertas lógicas, incluyendo desde las más simples (OR, AND, NAND…), hasta otras más complejas con más de dos entradas. Finalmente, podemos crear tablas y acceder a ellas en hexadecimal o en binario con cantidad de bits a elegir.

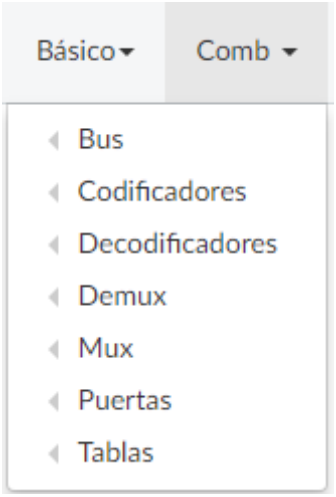


Figura 58.- Componentes del menú Combinacional

En la **figura 59**, podemos ver el menú que contiene las constantes. Aquí podemos meter un bit constante (1 o 0), o crear buses con valores concretos de datos. Además, también tenemos la posibilidad de añadir componentes de 7 segmentos, que nos dan su valor como salida, por ejemplo, para poder añadir letras o dígitos.

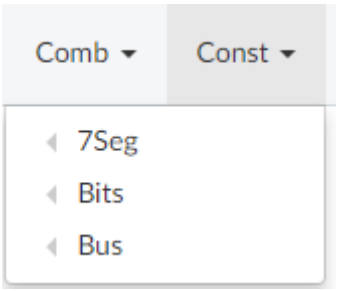


Figura 59.- Componentes del menú Constantes

En la **figura 60**, podemos ver el menú que contiene los periféricos. Aquí solo nos encontramos con dos subcategorías, la primera es la de DHTxx que es para introducir el control de este tipo de componentes externos que son sensores de temperatura y de humedad. Segundo tenemos para poder controlar Displays y poder mostrar por pantalla lo que queramos, hay varios tipos de Displays de diferentes marcas y características.

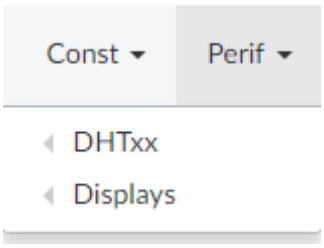


Figura 60.- Componentes del menú Periféricos

En la **figura 61** tenemos el menú que contiene Pines. En este solo hay dos bloques iguales, modificando únicamente la ubicación de cada una de las salidas. Estos pines se basan en puertas triestado para seleccionar si van a ser entradas o salidas. En la **figura 62** podemos ver ambos bloques para comprobar que son iguales y que solo se modifica la ubicación del pin.

Figura 61.- Componentes del menú Pines

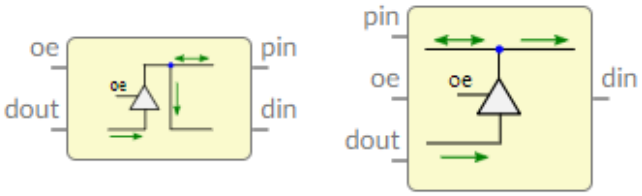


Figura 62.- Módulos de la categoría de Pines

En la **figura 63** vemos el menú *Templates*, en él como su nombre dice tenemos plantillas de varios componentes. Tenemos la plantilla para añadir un componente nuevo, un tipo de constante nueva o nueva documentación para nuestro proyecto.

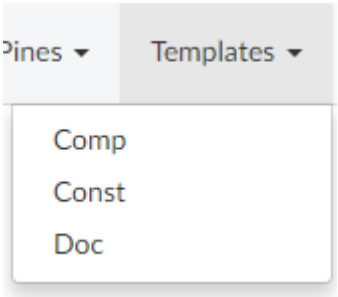


Figura 63.- Componentes del menú Templates

Después nos encontramos con la carpeta principal, denominada Varios, como podemos ver en la **figura 64**. Aquí podemos encontrar múltiples componentes que nos pueden ayudar a realizar circuitos digitales complejos. La primera categoría que encontramos es la de accesorios, que contiene múltiples módulos para hacer añadidos a nuestro circuito. Luego nos encontramos con un ADC, componente muy útil para circuitos digitales. También nos encontramos con módulos aritméticos, como, por ejemplo, sumadores, multiplicadores o desplazadores. Después tenemos biestables, como, por ejemplo, *Flip-Flops* de tipo D o de tipo T.

Debajo tenemos señales de bombeo, es decir nos crea señales cuadradas a frecuencias configurables. Después podemos añadir módulos para trabajar con buses, como por ejemplo agregadores para convertir dos buses en uno más grande o disgregadores para hacer lo opuesto. También tenemos comparadores y contadores, componentes muy útiles para circuitos digitales complejos. Después nos encontramos con detectores de flancos de reloj (de subida y bajada), para poder realizar acciones en función de ellos.

Después nos encontramos con una categoría con componentes para comunicarnos mediante el protocolo serie I2C. También nos encontramos con una librería con la que podemos configurar los pines de entrada a la FPGA añadiendo por ejemplo *pull-ups* o puertas triestado. A continuación, tenemos otros módulos que nos permiten controlar LEDs pudiendo hacer por ejemplo un *fade-out*, es decir, que se apague poco a poco el LED atenuando el voltaje que le llega.

En la siguiente categoría tenemos módulos para controlar máquinas. Con estos podemos crear máquinas de estados finitos, que son modelos computacionales que operan sobre una señal de entrada para dar una salida concreta. Una forma habitual de representar esto es mediante diagramas de estado, y podemos implementarlo con estos módulos. Después nos encontramos con el *prescaler*, que en sí es un PLL, es decir, nos permite dividir la frecuencia de una señal de reloj para ajustarla a lo que necesitamos.

También tenemos módulos que implementan pulsadores para realizar acciones en función de si se ha pulsado o no, o cuantas veces se ha pulsado. Después tenemos módulos que nos permiten crear señales PWM en función de los datos que metamos de entrada y la señal de reloj que queramos. Después tenemos registros de diferentes tamaños y tenemos bloques que nos provocan retardos de señales.

Finalmente, tenemos una última categoría llamada Serial, que nos permite implementar protocolos de comunicación serie como, por ejemplo, el UART. Tenemos bloques que permiten recibir datos, transmitirlos o realizar las dos acciones simultáneamente.

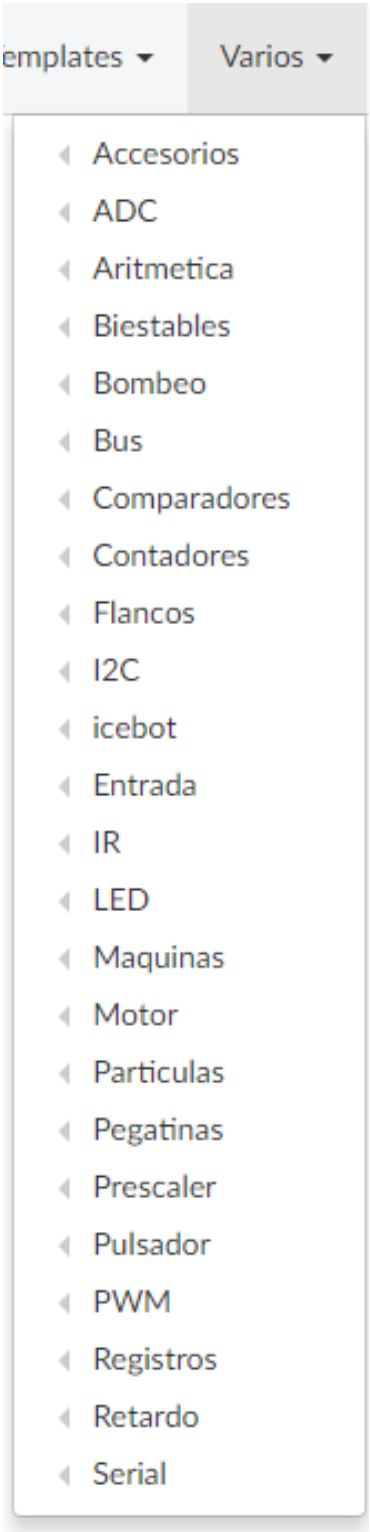


Figura 64.- Componentes del menú Varios

Por último, tenemos, como vemos en la **figura 65**, otro menú denominado Varios-2, que igual que el anterior es una colección de diversos módulos útiles para la implementación de circuitos digitales. La primera categoría nos permite incluir un módulo que actúa como si fuera un *Buffer*. La siguiente, nos permite introducir diferentes tipos de memoria, como por ejemplo RAM o ROM de diferentes tamaños de memoria y de páginas. Después nos

encontramos con otra categoría que nos permite controlar servos o motores de diferentes marcas y tipos.

A continuación, tenemos otro módulo que nos permite reproducir diferentes notas musicales, en función del valor de entrada. También tenemos una serie de módulos que nos permiten implementar mediante la FPGA una comunicación SPI como la que hemos visto a lo largo del proyecto. Permite implementar como si fuera maestro o esclavo y en diferentes modos.

También podemos implementar una pila o *stack* como podemos ver en la figura. Después podemos implementar temporizadores, muy útiles para poder hacer circuitos secuenciales o que se activen determinados componentes pasado un tiempo. Finalmente tenemos unos últimos bloques que nos permiten usar una conexión VGA y transmitir por ella.

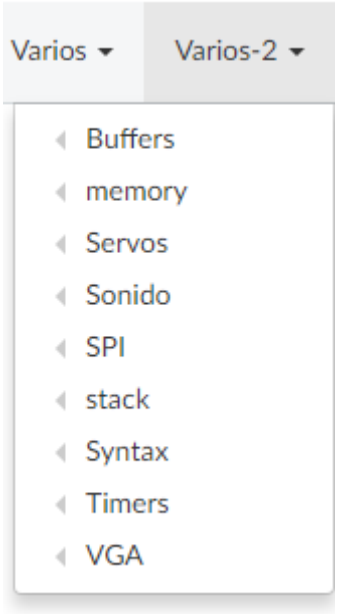


Figura 65.- Componentes del menú Varios-2

Como hemos podido ver en esta sección, esta colección es tremendamente útil para poder realizar los circuitos digitales. Mediante todos estos módulos se puede implementar cualquier tipo de circuito o de componente digital. Además, como se pueden añadir nuevos módulos, programándolos mediante Verilog, el uso de este programa es tremendamente útil para la aplicación de la educación, ya que se puede implementar cualquier circuito y simular su funcionamiento, sin necesidad de montarlo en la realidad, ya que la propia FPGA monta el circuito internamente. Mediante su utilización, también se puede aprender el funcionamiento de diferentes protocolos de comunicación como el SPI o el I2C y se podría implementar hasta los módulos que usa por ejemplo una UART y comunicarnos a través de ella.

d) Programa de prueba

Una vez está todo configurado en el programa y hemos visto que bloques podemos usar, procedemos a crear un programa de prueba para comprobar el correcto funcionamiento. El programa que hemos implementado lo podemos ver en la **figura 66**. El funcionamiento es muy simple ya que es un contador de 4 bits que podemos ver iluminando los 4 LEDs presentes en nuestra placa.

Lo primero que vemos es un bloque en el que entra nuestro reloj en un bloque que tiene un corazón. Este bloque realiza un bombeo de un bit con valor 1 con frecuencia de 1 Hz. La salida de este bloque irá a nuestro contador de 4 bits. Por ello, el contador aumentará de valor cada 1 segundo. La salida del contador es un bus de 4 bits por lo que necesitamos un disgregador que nos lo separe en cuatro cables diferentes. Cada uno de los cables irá conectado a cada uno de los LEDs de la placa, siendo el LED D4 el bit de menor orden y el LED D3 el de mayor orden.

Además, con este contador, una vez se alcanza el límite, es decir el valor 15, se vuelve al valor 0, por lo que este código se ejecutará continuamente hasta que se desconecte la placa de la alimentación.

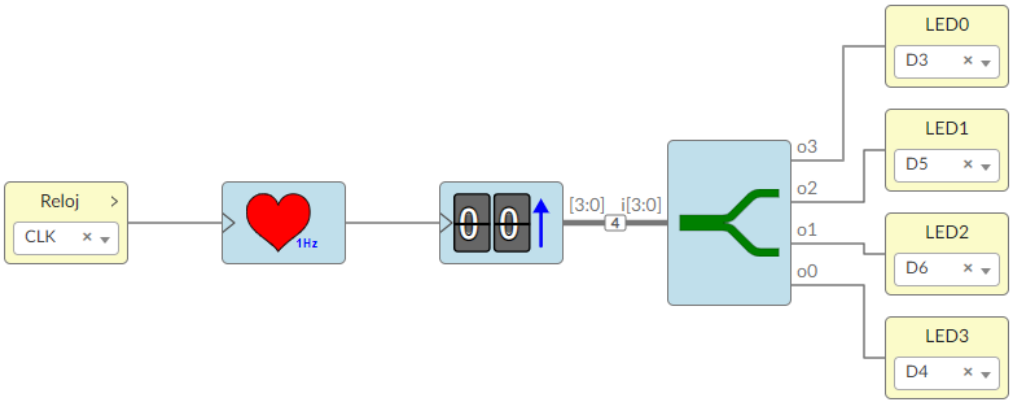


Figura 66.- Programa de prueba realizado

En la **figura 67**, podemos ver una imagen de nuestra placa en un instante mientras se está ejecutando el programa que hemos visto anteriormente. Como podemos ver, están 3 de los 4 LEDs encendidos de nuestra placa, dando a entender que el contador se encuentra en el estado 1011, es decir, el número 11 en decimal.

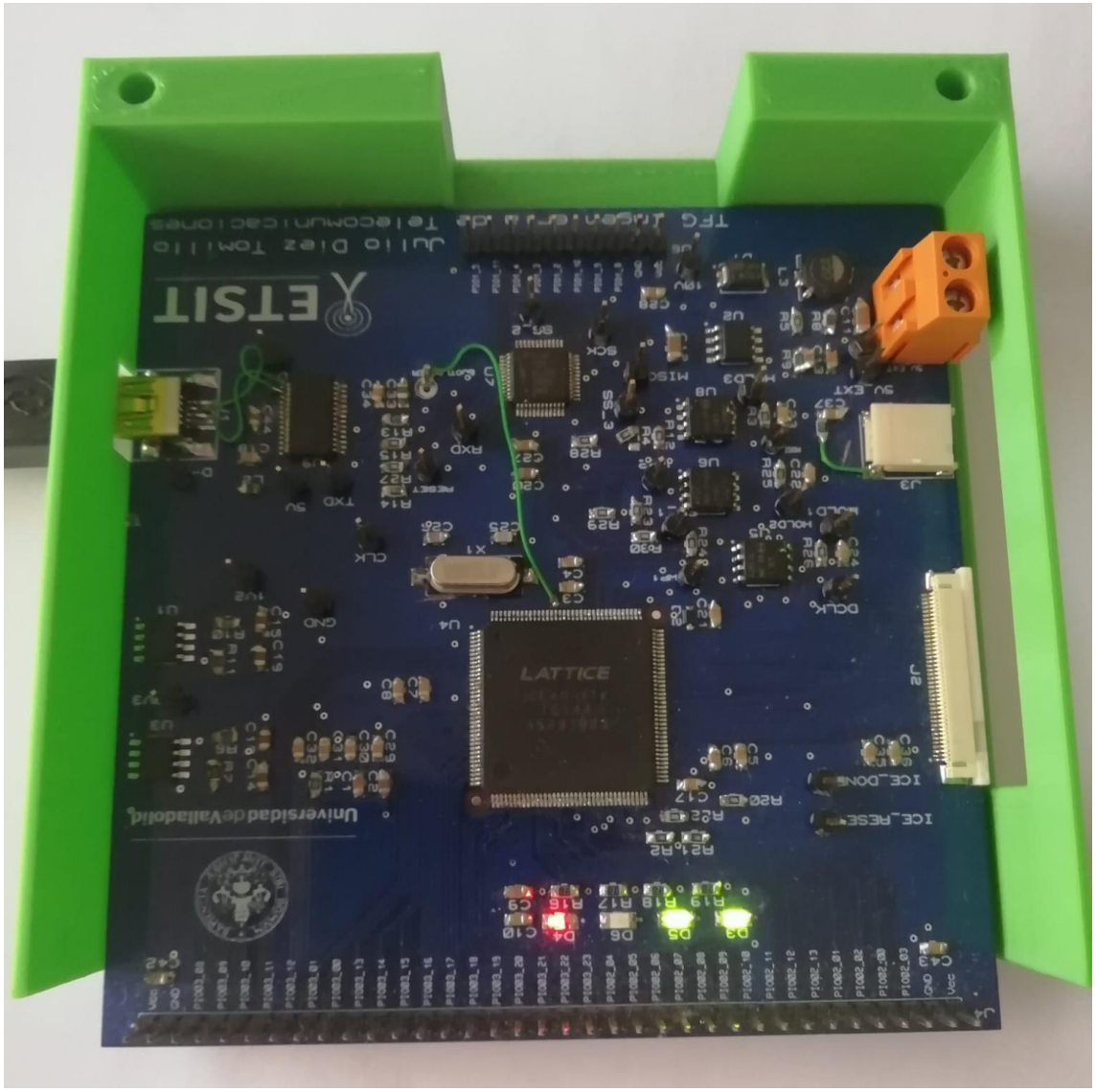


Figura 67.- Imagen de la placa en un instante de la ejecución del programa

Una vez tenemos el programa creado, el siguiente paso es pulsar el botón verificar para comprobar que no hay ningún error en el diseño. Después pulsaremos el botón sintetizar, esto transformará el programa creado en bloques en un archivo de Verilog, y lo compilará. Con esto, podremos ya exportar nuestro programa al programa que queramos.

Para poder enviar el programa al microcontrolador y que lo guarde en la memoria necesitaremos un archivo binario (*.bin*), por lo que tendremos que seleccionar la opción de exportar como *BitStream* siguiendo los pasos que podemos ver en la **figura 68.**

Lo exportaremos al mismo directorio en el que tenemos el ejecutable *upload_fpga.exe*, para facilitar el comando. Y procederemos a ejecutar el comando para grabar el programa en nuestra FPGA para que se pueda ejecutar. O, como alternativa, se puede modificar el menú de ICEStudio para poder descargarlo automáticamente, sin tener que ejecutar el ejecutable desde línea de comandos.

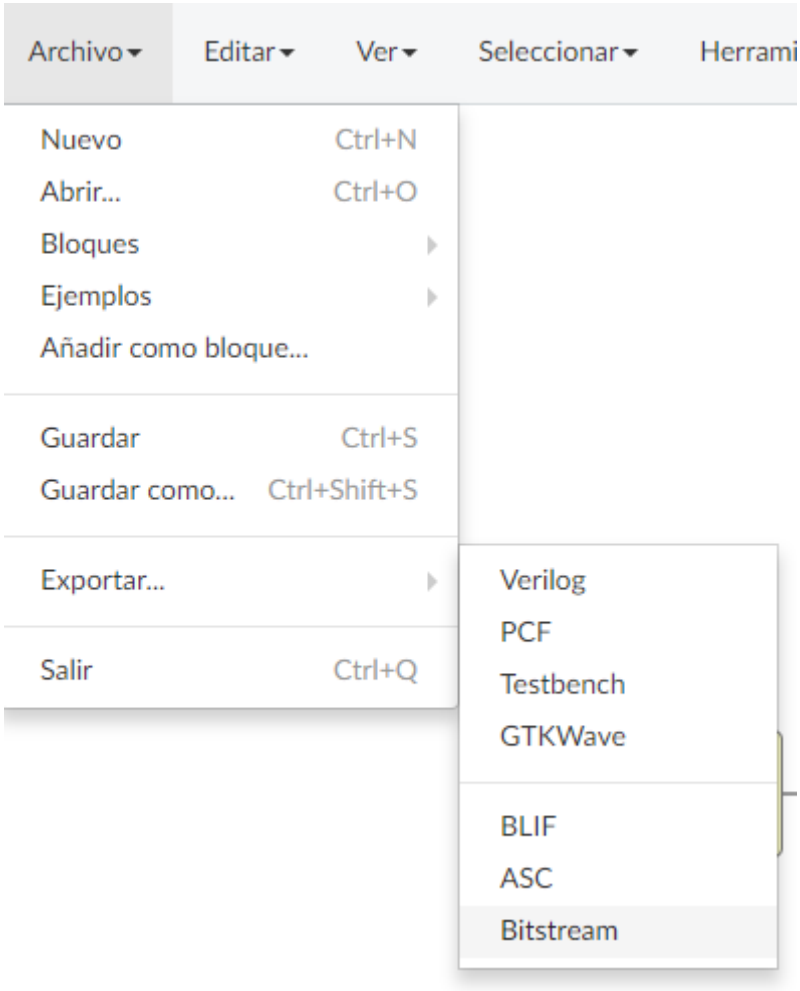


Figura 68.- Selección para exportar el fichero

CAPÍTULO V

ESTUDIO ECONÓMICO DEL PROYECTO: PRESUPUESTO

El siguiente paso en nuestro proyecto es realizar el estudio económico. Como hemos expuesto durante todo el documento, el principal objetivo es realizar una placa de desarrollo de precio asequible que pueda destinarse a la educación. Por ello es muy importante la realización de un estudio económico y un presupuesto, para comprobar si nuestra placa finalmente se podría vender a un precio asequible o supera el precio de otras placas en el mercado. Por ello, en este apartado inicialmente realizaremos el presupuesto económico de nuestro proyecto para compararlo a continuación con otras placas de desarrollo similares en el mercado.

Presupuesto

Lo primero que vamos a realizar, como ya hemos comentado, es elaborar el presupuesto de nuestra placa. En él vemos que hay varios apartados que encarecen nuestro proyecto. El primero son los componentes que componen nuestra placa, como resistencias, condensadores, integrados… Por cada placa fabricada habrá que comprar todos los componentes. Los componentes son adquiridos a través del distribuidor Farnell. En el **anexo 3** podemos ver el *Bill of Materials* (BOM), en el que viene detallado el precio de cada uno de los componentes de nuestra placa con este distribuidor. Para cada componente podemos ver diferentes precios en función de la cantidad que compremos, abaratándose a mayor número. Lo que vemos en el BOM se corresponde con el precio la compra de más de 10 componentes.

En el presupuesto también aparece incluido el coste que tiene la fabricación de las placas. Las placas se fabrican en una empresa denominada PCBWay que tiene unos precios muy baratos y ofrece una buena calidad. Dentro de este coste se incluye el que tienen las placas en sí, como los gastos de envío que hay que pagar. En tercer lugar, tenemos coste de diseño a amortizar que es el coste que ha tenido el desarrollo de toda la placa. Se ha estado trabajando en el hardware aproximadamente 73 horas, mientras que en el software básico unas 40 horas, dando un tiempo total de desarrollo de 113 horas. A esto hay que sumar también el coste de la mano de obra que supone el montaje de todos los componentes del circuito. Se ha supuesto que se tarda aproximadamente 2 horas en realizarlo. Además, el coste tanto de la mano de obra como del diseño se ha valorado en 15 €/h, precio aproximado de un desarrollador novel en una empresa. [22]

Finalmente, a nuestro presupuesto sumamos de forma opcional la pantalla TFT, ya que se puede trabajar sin ella perfectamente. Es un elemento accesorio que puede llegar a ser muy útil pero que puede encarecer el precio, por lo que no sería obligatoria su compra

junto con la placa, pero si recomendable. Además, en este proyecto no se ha realizado la programación del control de la pantalla, por lo que ofrecemos en el presupuesto el precio total de la placa cuando se incluye la pantalla TFT y cuando no. En todos los precios está incluido el IVA.

Finalmente, hemos realizado dos presupuestos: uno cuando solo se producen 10 placas y otro cuando producimos 100, para comparar la diferencia de precio. En la **tabla 3** presentamos el presupuesto cuando se producen únicamente 10 placas. Como podemos ver, la placa sin la pantalla tiene un coste aproximado de 225€, lo que consideramos un precio excesivo. El principal elemento que encarece el coste es el diseño, que es demasiado caro para producir únicamente 10 placas. Por ello, el precio de las placas aumenta notablemente ya que habrá que amortizarlo entre pocas placas. Además, tenemos que añadir otro coste, la mano de obra por el montaje de la placa, de aproximadamente dos horas, que encarece cada placa en 30€. Por eso, hemos realizado un nuevo presupuesto fabricando 100 placas para averiguar si el precio disminuye.

Tabla 3.- Presupuesto de producir 10 placas

	PRECIO	**UNIDADES**	**TOTAL**
COMPONENTES (FARNELL)	25,06 €/u	10 u	250,6 €
PLACAS (PCBWAY)	26,39 €	10 u	26,39 €
COSTE DISEÑO A AMORTIZAR	15 €/h	113 horas	1695 €
MANO DE OBRA (MONTAJE)	15 €/h	2 horas/placa	300 €
PANTALLA TFT	30 €	10 u	300 €
TOTAL (Sin Pantalla)	**227,2 €/u**	10 placas	2272 €
TOTAL (Con Pantalla)	**257,2 €/u**	10 placas	2572 €

En la **tabla 4** podemos ver el presupuesto realizado cuando se fabrican 100 placas. Como se puede apreciar, el precio final por placa, sin incluir la pantalla, pasa a ser aproximadamente 70€, que es una disminución de más de 3 veces el precio de cuando se producen 10 placas. Esto se debe a que lo más caro es el coste del diseño, que es un precio fijo, es decir, independiente del número de la placas que fabriquemos. Por ello, a mayor número de placas que fabriquemos más bajará el precio ya que se reparte el coste de la mano de obra entre el número de placas. Con los componentes y la fabricación ocurre lo mismo, y es que a mayor número que se solicite, más disminuirá el precio. No ocurre esto con el coste que supone la mano de obra del montaje de la placa, ya que esto supone un añadido de 30€ por cada placa que fabriquemos, independientemente del número de placas que mandemos fabricar.

Tabla 4.- Presupuesto de producir 100 placas

	PRECIO	**UNIDADES**	**TOTAL**
COMPONENTES (FARNELL)	21,81€/u	100 u	2181 €
PLACAS (PCBWAY)	181,09 €	100 u	181,09 €
COSTE DISEÑO A AMORTIZAR	15 €/h	113 horas	1695 €
MANO DE OBRA	15 €/h	2 horas/placa	3000 €
PANTALLA TFT	30 €/u	100 u	3000 €
TOTAL (Sin Pantalla)	**70,57€/u**	100 placas	7057,09 €
TOTAL (Con Pantalla)	**100,57€/u**	100 placas	10057,09 €

Con todo lo expuesto anteriormente podemos concluir que nos sale más barato producir un mayor número de placas. Pero tampoco se pueden fabricar demasiadas ya que el mercado tampoco es muy grande, es decir, no hay una gran cantidad de posibles compradores de las placas, ya que el objetivo principal son universidades. Un buen número de placas a fabricar serían 100, ya que se consigue un precio bastante barato, y es una cantidad fácilmente distribuible.

Comparación con otras Placas: Estado del Arte

En este apartado vamos a comparar placas similares a la nuestra basadas en FPGAs, para poder ver si nuestra placa supone una mejora frente a las placas actuales en el mercado. Para ello, vamos a comentar las características de cada una de las placas existentes en el mercado junto a su precio actual, y compararlas con nuestra placa. Todas ellas, como la nuestra, tienen en común que usan una FPGA de la familia Ice40, explicada en apartados anteriores, del fabricante Lattice.

a) Icezum Alhambra [2]

- **Fabricante:** Mareldem Technologies
- **FPGA usada:** ICE40HX1K
- **Frecuencia:** 12 MHz
- **Características:** Posee 8 LEDs, 2 pulsadores, 4 canales A/D y numerosos pines de acceso a la FPGA. Es de hardware libre y es compatible con Arduino.
- **Precio:** 65€

Figura 69.- Icezum Alhambra

b) Lattice Icestick [1]

- **Fabricante:** Lattice
- **FPGA usada:** ICE40HX1K
- **Frecuencia:** 12 MHz

- **Características:** Tiene 4 LEDs rojos y uno verde. Tiene unos pocos pines de acceso a la FPGA.
- **Precio:** 18,25€

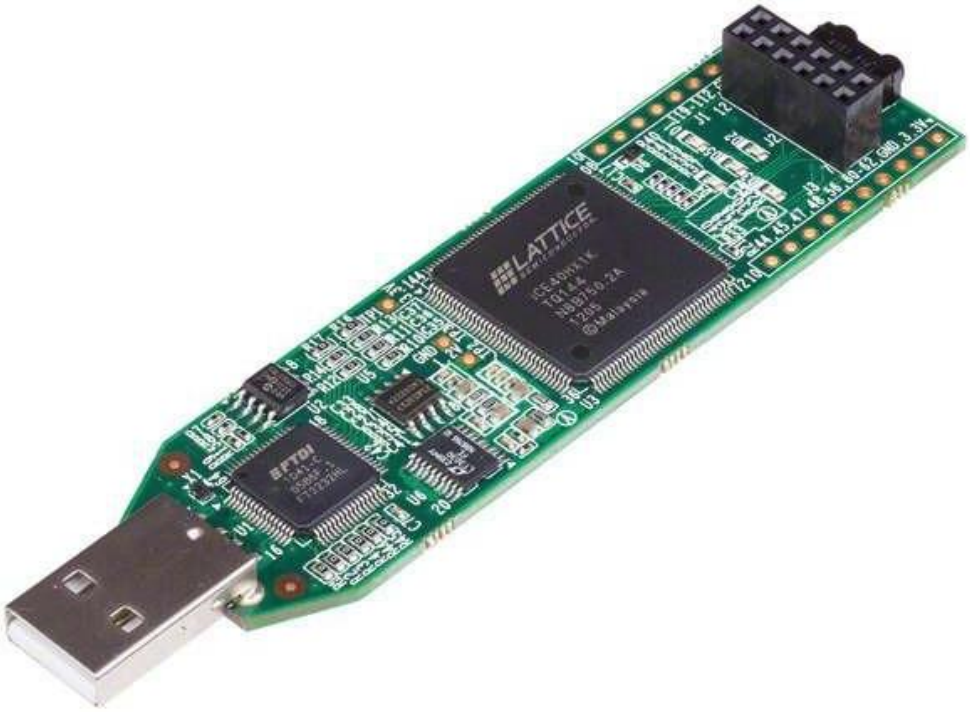


Figura 70.- Lattice Icestick

c) Kéfir I [23]

- **Fabricante:** Instituto Nacional de Tecnología Industrial (INTI), Argentina
- **FPGA usada:** ICE40HX4K-TQ144
- **Frecuencia:** 24 MHz
- **Características:** Tiene 4 sensores capacitivos y 4 LEDs. Es de hardware libre y es compatible con Arduino.
- **Precio:** No se fabrica a nivel industrial

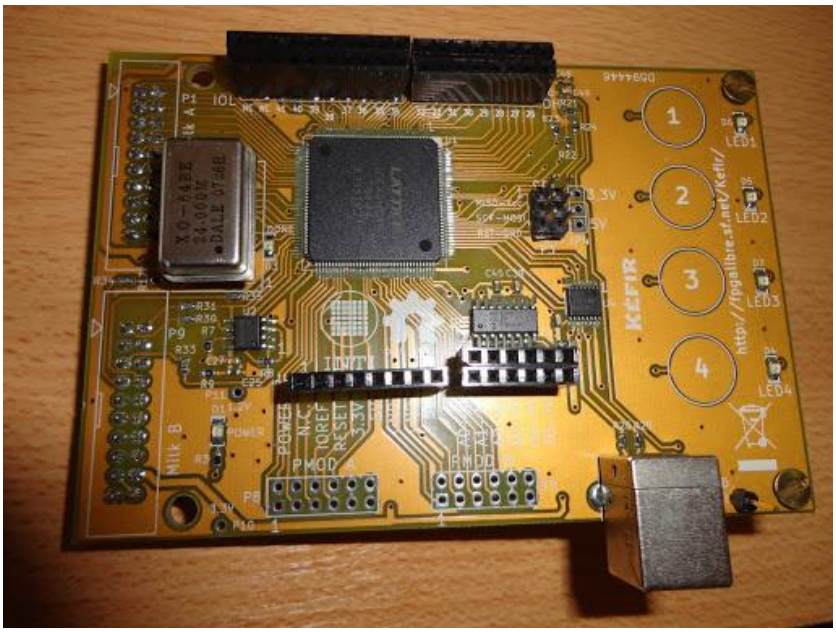


Figura 71.- Placa Kéfir I

d) Lattice Breakout Board [24]

- **Fabricante:** Lattice
- **FPGA usada:** ICE40HX8K CT256
- **Frecuencia:** 12 MHz
- **Características:** Tiene 8 LEDs verdes y tenemos acceso a 40 pines de la FPGA
- **Precio:** 40 €

Figura 72.- Placa Lattice Breakout Board

e) Nandland Go Board [25]

- **Fabricante:** Nandland (Crowdfunding)
- **FPGA usada:** ICE40HX1K
- **Frecuencia:** 12 MHz
- **Características:** Tiene 4 LEDs verdes, 2 displays de 7 segmentos, 4 pulsadores y 1 conector VGA.
- **Precio:** 60 €

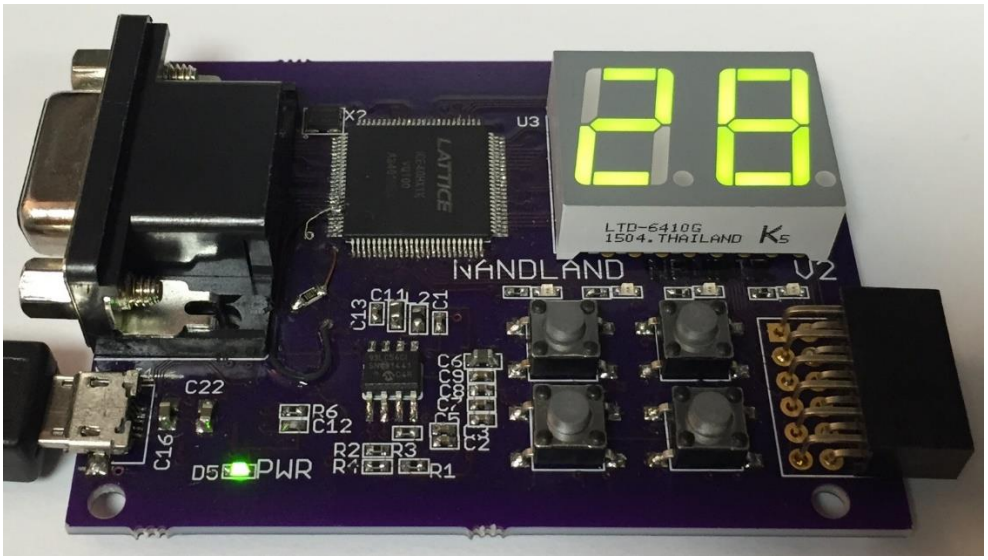


Figura 73.- Placa Nandland Go Board

Como hemos podido ver, nuestra placa es diferente respecto a las placas examinadas respecto a sus características. Esto se debe principalmente a que la filosofía de nuestro diseño es radicalmente diferente al del resto que hemos visto. La programación de nuestra FPGA se hace mediante un microcontrolador que también es programable, lo que supone un cambio radical con el resto de los diseños. Además, la posibilidad de añadir una pantalla supone otro añadido, respecto al resto de las placas (aunque implementable mediante los pines). Otra gran característica que diferencia a nuestro proyecto del resto es que podemos cambiar la velocidad de funcionamiento de nuestra FPGA, ya que su reloj se obtiene del microcontrolador, y este reloj de salida es configurable, por lo que podemos decidir a qué velocidad queremos que funcione la FPGA. Finalmente, al tener nuestra placa un precio reducido, podemos concluir que sí tendríamos mercado dentro de las placas de FPGAs ya que supone un cambio radical en comparación con el resto. Además, nuestra placa está enfocada a la educación para el aprendizaje de sobre FPGAs y microcontroladores, por lo que es más útil que el resto de las existentes en el mercado.

CAPÍTULO VI

CONCLUSIONES Y MEJORAS FUTURAS

Para finalizar esta memoria, vamos a efectuar unas breves conclusiones sobre el trabajo realizado y sobre nuestra placa. También mencionaremos brevemente algunas líneas posibles de trabajo futuro sobre este proyecto, para ampliarlo y hacer posibles mejoras o añadidos.

CONCLUSIONES

Como dijimos en la introducción de esta memoria, el principal objetivo del proyecto era realizar una placa de bajo coste para poder dedicarla a la educación y aprendizaje sobre FPGAs y microcontroladores. Como hemos podido ver, el proyecto ha cumplido todas las expectativas porque se ha logrado realizar la placa y como hemos analizado ver en el anterior capítulo, tiene un coste reducido frente a las otras placas existentes actualmente en el mercado.

La clave de esta placa es que no sirve únicamente para aprender sobre FPGAs, sino que, al estar controlada mediante un microcontrolador, también hace falta programar este para poder controlar la FPGA, por lo que trabajando con esta placa se puede aprender sobre dos componentes muy importantes en la electrónica actual, como son los microcontroladores y las FPGAs.

También, como hemos podido ver en el apartado en el que hablábamos sobre el programa ICEStudio, este nos ofrece muchísimas posibilidades de aprendizaje y de trabajo. Con él podemos crear cualquier circuito digital y comprobar su funcionamiento cargándolo en la FPGA. Además, también se puede combinar con el aprendizaje de lenguajes HDL, pudiendo crear módulos programados, y posteriormente simularlos cargando el programa en la FPGA.

Las posibilidades que ofrece este programa son infinitas pudiendo dedicarse principalmente al sector educativo para el aprendizaje de electrónica digital. Además, como el programa es de software libre, no hace falta que la escuela o universidad tenga que gastar en la compra licencias del programa ya que cualquier alumno puede instalarlo gratuitamente.

Podemos concluir diciendo que las FPGAs es un mercado que está en alza, y se están haciendo un hueco dentro de la electrónica, por lo que es y será fundamental estudiar su funcionamiento y saber sus usos y aplicaciones. Por ello, esta placa puede suponer un avance revolucionario en el modo de explicar el funcionamiento de las FPGAs o de cualquier circuito digital a los estudiantes.

TRABAJO FUTURO

Para finalizar todo el proyecto es interesante comentar y cómo se puede seguir avanzando en el proyecto en el futuro.

En este proyecto, a pesar de haber diseñado el esquemático con los conectores para la pantalla TFT, y haberlo preparado para solo tener que conectarlo, no se ha trabajado con la pantalla. Es obvio que en el futuro se puede trabajar sobre la ella, procediendo a su conexión y funcionamiento. Para ello, el objetivo principal sería construir mediante ICEStudio un programa que convierta la FPGA en un controlador de video, es decir, que sea la propia FPGA la que controle lo que salga por pantalla.

También se podrían implementar mediante nuestra FPGA programas de ejemplo más complejos. Por ejemplo, se podrían conectar los pines de salida de la FPGA o del microcontrolador a algún sensor, procesar estos valores y dar una salida por la pantalla TFT. Se podría dar a la placa infinitos usos, algunos de ellos también dedicados al IoT (*Internet of Things*), el internet de las cosas.

Otra línea de trabajo sería realizar añadidos a la placa para hacerla más versátil y con más utilidades. Para ello se podrían conectar diversos periféricos como, por ejemplo, botones, zumbadores o un altavoz. Con esto aumentarían las posibilidades que daría la placa para aplicarse a la educación, ya que la misma nos podría dar realimentación acústica y no solo visual con los LEDs o con la pantalla TFT.

Como podemos ver, este proyecto no es ni mucho menos un proyecto cerrado sobre el que no se pueda trabajar más. Es un proyecto abierto con mucho futuro, que puede ser ampliable y desarrollado por muchos sitios, y dedicar su aplicación a muchos sectores, aunque como hemos expuesto previamente, el fundamental es la educación.

BIBLIOGRAFÍA

[1] Lattice Semiconductor, "ICEstick Evaluation Kit, User's Guide", Manual de Usuario del ICEstick, Agosto 2013

[2] ObiJuan, "La IceZum Alhambra y otras placas con FPGAs libres", https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/V%C3%ADdeo-3:-La-Icezum-Alhambra-y-otras-placas-con-FPGAs-libres

[3] Página web oficial de ICEStudio, https://icestudio.io/

[4] Carmen Villanueva, "¿Qué es y para qué sirve un diagrama de Gantt", Diciembre 2018, https://blog.teamleader.es/diagrama-de-gantt

[5] Xilinx, "What is an FPGA?", xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[6] Jeff Johnson, "List and comparison of FPGA companies", Julio 2011, http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html

[7] Global Market Insights, Inc., "Field Programmable Gate Array (FPGA) Market Value to Hit USD $13 Billion by 2026: Global Market Insights, Inc.", Noviembre 2019, https://www.prnewswire.com/news-releases/field-programmable-gate-array-fpga-market-value-to-hit-usd-13-billion-by-2026-global-market-insights-inc-300961294.html

[8] Apuntes Asignatura Diseño de Circuitos Digitales para Comunicaciones, "Tema 1 – Lógica Programable", Departamento Electricidad y Electrónica, Universidad de Valladolid (UVA), Febrero 2019

[9] *Grand View Research*, "Deep Learning Market Size, Share & Trends Analysis Report By Solution, By Hardware (CPU, GPU, FPGA, ASIC), By Service, By Application, By End-Use, By Region, And Segment Forecasts, 2018 – 2025", Mayo 2017

[10] Enrique Gómez, "Cómo funciona el Puerto Serie y la UART", Octubre 2017, https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/

[11] Kiara Navarro, "¿Cómo funciona el protocolo SPI?", Octubre 2014, http://panamahitek.com/como-funciona-el-protocolo-spi/

[12] Lattice Semiconductor, "ICE40™ LP/HX Family Data Sheet", ICE40HX1K *Datasheet*, Octubre 2017

[13] NXP Semiconductors, "LPC1110/11/12/13/141/15", LPC 1114 *Datasheet*, Marzo 2014

[14] ON Semiconductor, "1.5 A 280 kHz/560 kHz Boost Regulators", CS5171 *Datasheet*, Enero 2010

[15] Cypress, "USB-UART LP Bridge Controller", CY7C65213 *Datasheet*, Febrero 2018

[16] Microchip, "500 mA, Low Voltage, Low Qquiescent Current LDO Regulator", MCP1725 *Datasheet*, Diciembre 2007

[17] Micron, "Micron Serial NOR Flash Memory", N25Q064A *Datasheet*, Febrero 2018

[18] Trujillo, F. D.; Pozo, A.; Triviño. A, "Disipación de calor", OCW – Universidad de Málaga, 2011

[19] Página web oficial de la empresa PCBWAY, https://www.pcbway.com/

[20] Obijuan Academy, Obi Juan, Marzo 2019, http://www.iearobotics.com/wiki/index.php?title=Obijuan_Academy

[21] Repositorio de GitHub con la colección creada por Obijuan llamada Collection-Jedi, https://github.com/FPGAwars/Collection-Jedi

[22] Sueldos para Ingeniero de Telecomunicaciones, Mayo 2020, https://www.glassdoor.es/Sueldos/ingeniero-de-telecomunicaciones-sueldo-SRCH_KO0,31.htm

[23] FPGA Libre, http://fpgalibre.sourceforge.net/Kefir/, 2017

[24] Lattice Semiconductor, "MachXO2 Breakout Board Evaluation Kit, User's Guide" Manual de Usuario de Breakout Board, Enero 2014

[25] NandLand, "Go Board- The Best FPGA Development Board for Begineers", https://www.nandland.com/goboard/introduction.html, 2016

APÉNDICES

ANEXO 1

DIAGRAMA GANTT

Un diagrama de Gantt es una herramienta gráfica para planificar y programar las diferentes tareas de un proyecto. A cada una se la asigna una fecha de inicio, que puede estar ligada a la finalización de otra tarea, y una fecha de finalización. El diagrama se ha modificado a lo largo del proyecto en función de las tareas que iban sufriendo retraso o se terminaron prematuramente.

TFG Entrenador FPGA - Julio Díez

26-feb-2020

ETSIT - UVA

https://www.tel.uva.es/

Encargado del proyecto	Julio Diez
Fechas de inicio y fin del proyecto	10-oct-2019 - 18-mar-2020
Progreso	100%
Tarea	15
Recursos	1

Planificación del TFG de Ingeniería de Telecomunicaciones en el que se realizará un entrenador de FPGAs mediante un microcontrolador.
Realizado por Julio Díez Tomillo.

TFG Entrenador FPGA - Julio Díez

26-feb-2020

2

Tarea

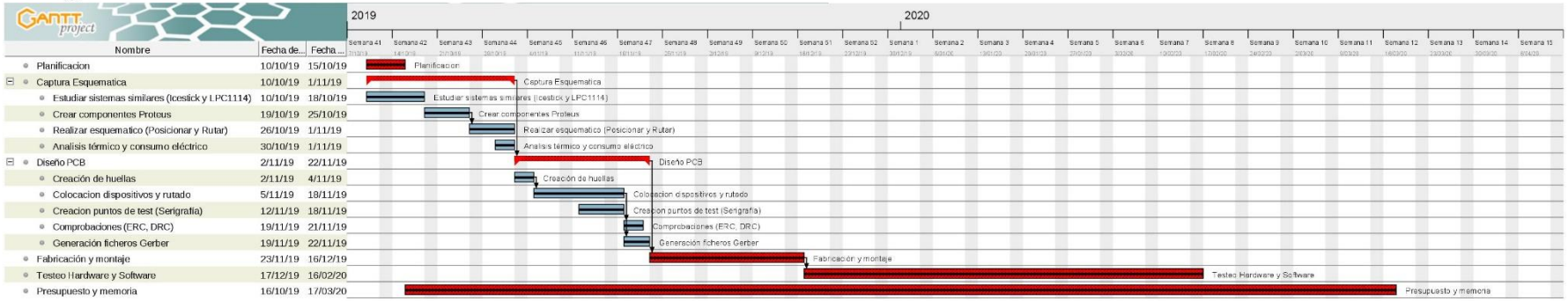
Nombre	Fecha de inicio	Fecha de fin
Planificacion	10/10/19	15/10/19
Captura Esquematica	10/10/19	1/11/19
Estudiar sistemas similares (Icestick y LPC1114)	10/10/19	18/10/19
Crear componentes Proteus	19/10/19	25/10/19
Realizar esquematico (Posicionar y Rutar)	26/10/19	1/11/19
Analisis térmico y consumo eléctrico	30/10/19	1/11/19
Diseño PCB	2/11/19	22/11/19
Creación de huellas	2/11/19	4/11/19
Colocacion dispositivos y rutado	5/11/19	18/11/19
Creacion puntos de test (Serigrafía)	12/11/19	18/11/19
Comprobaciones (ERC, DRC)	19/11/19	21/11/19
Generación ficheros Gerber	19/11/19	22/11/19
Fabricación y montaje	23/11/19	16/12/19
Testeo Hardware y Software	17/12/19	16/02/20
Presupuesto y memoria	16/10/19	17/03/20

TFG Entrenador FPGA - Julio Díez

26-feb-2020

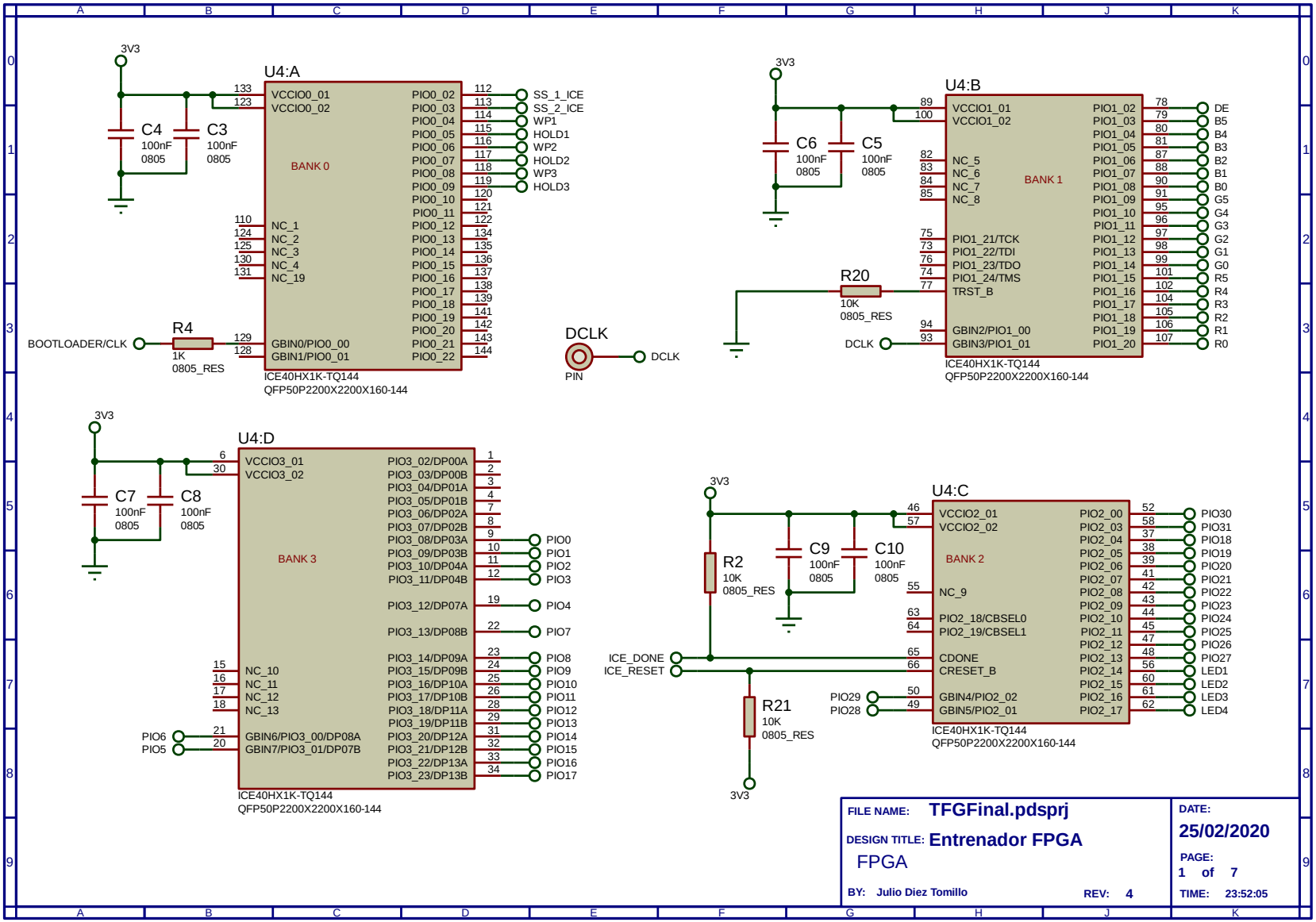
Diagrama de Gantt

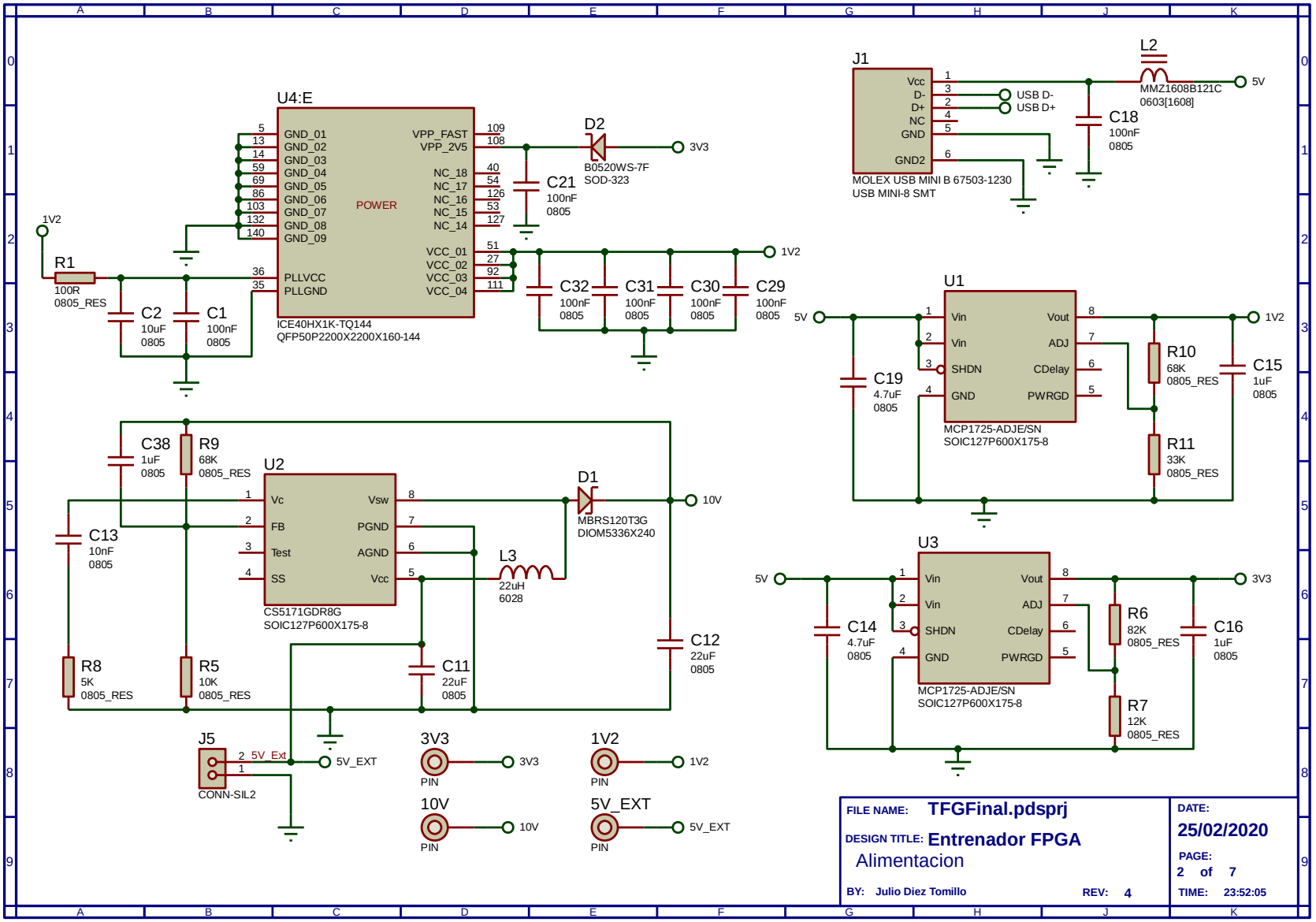
4

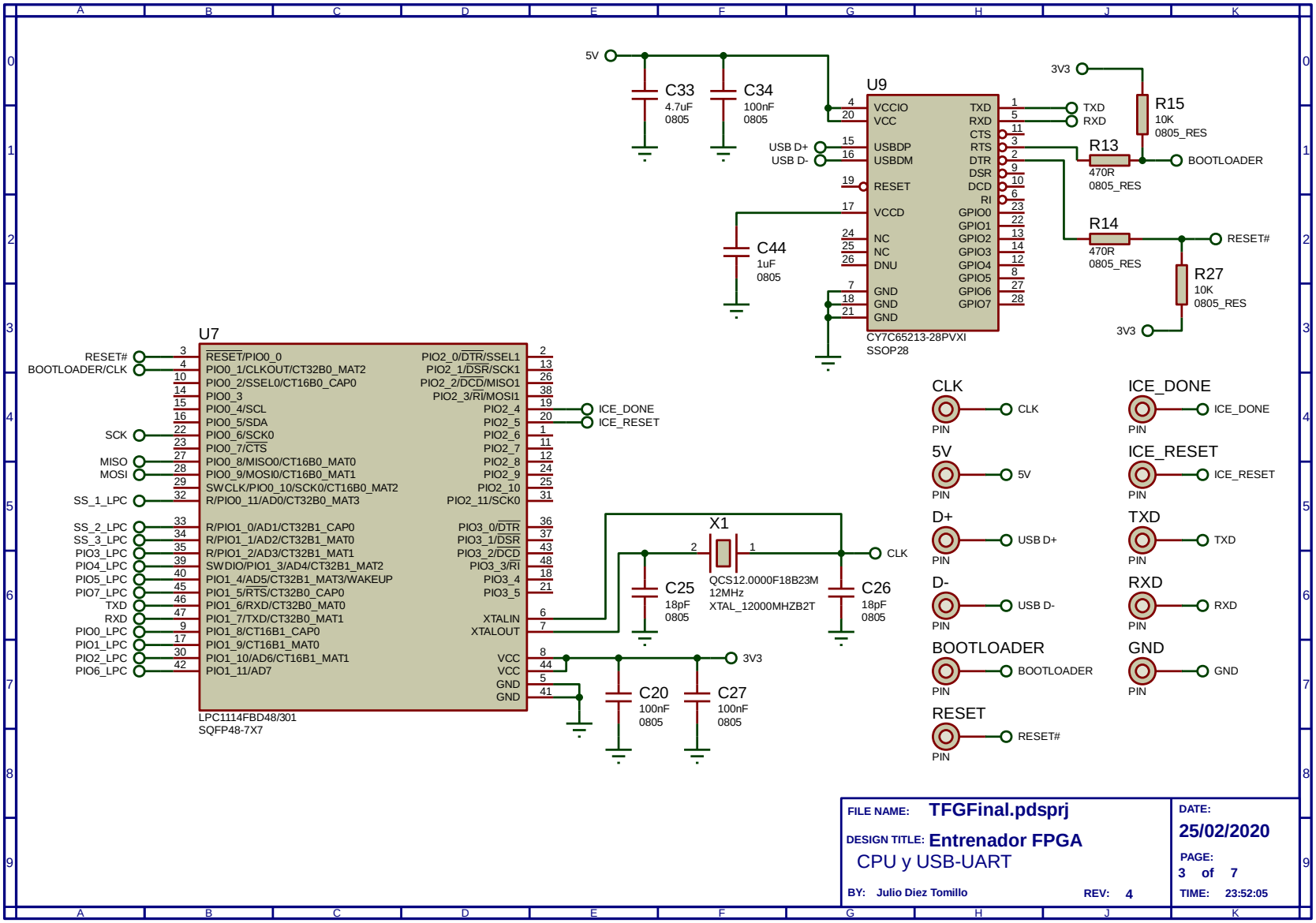


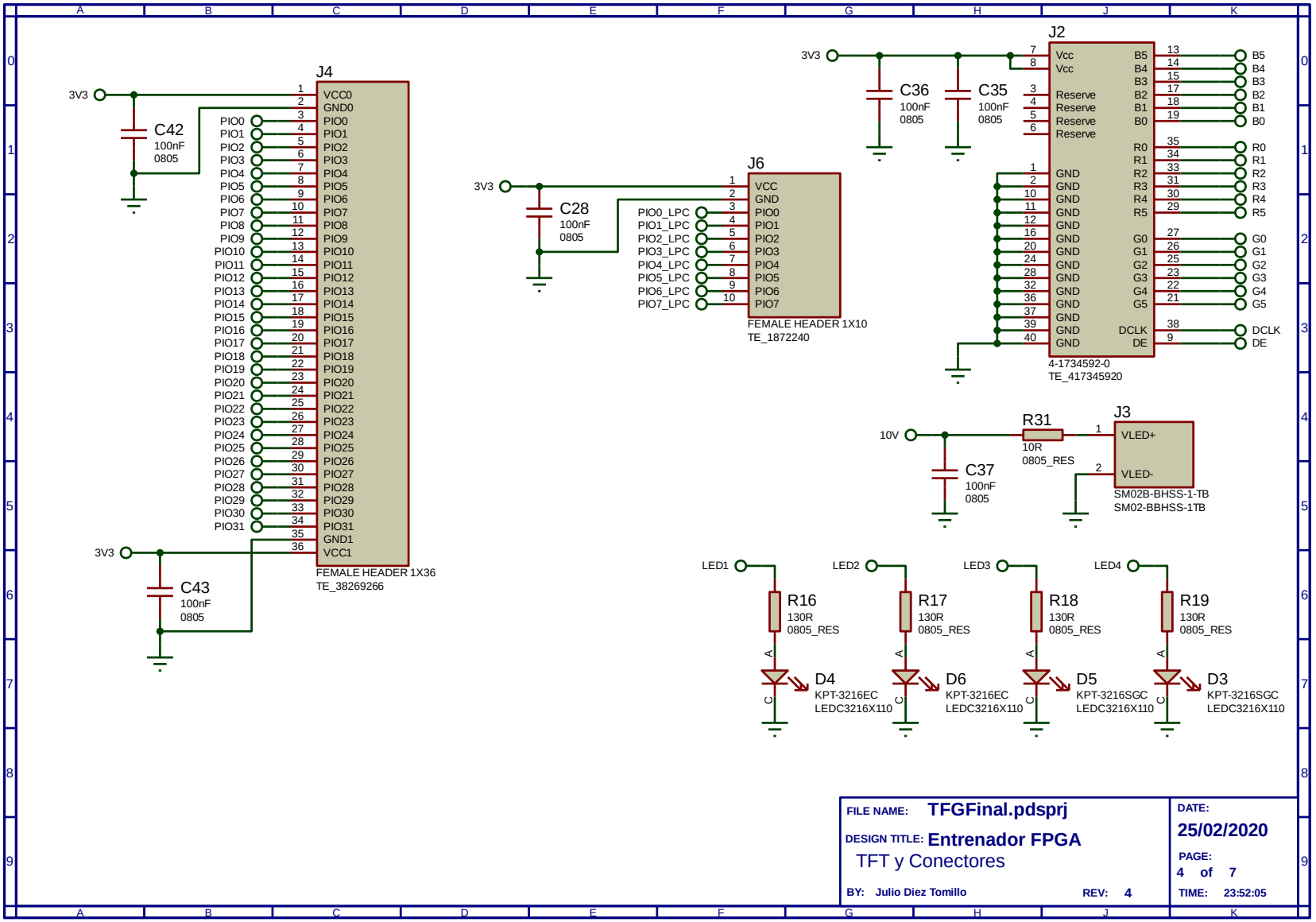
ANEXO 2

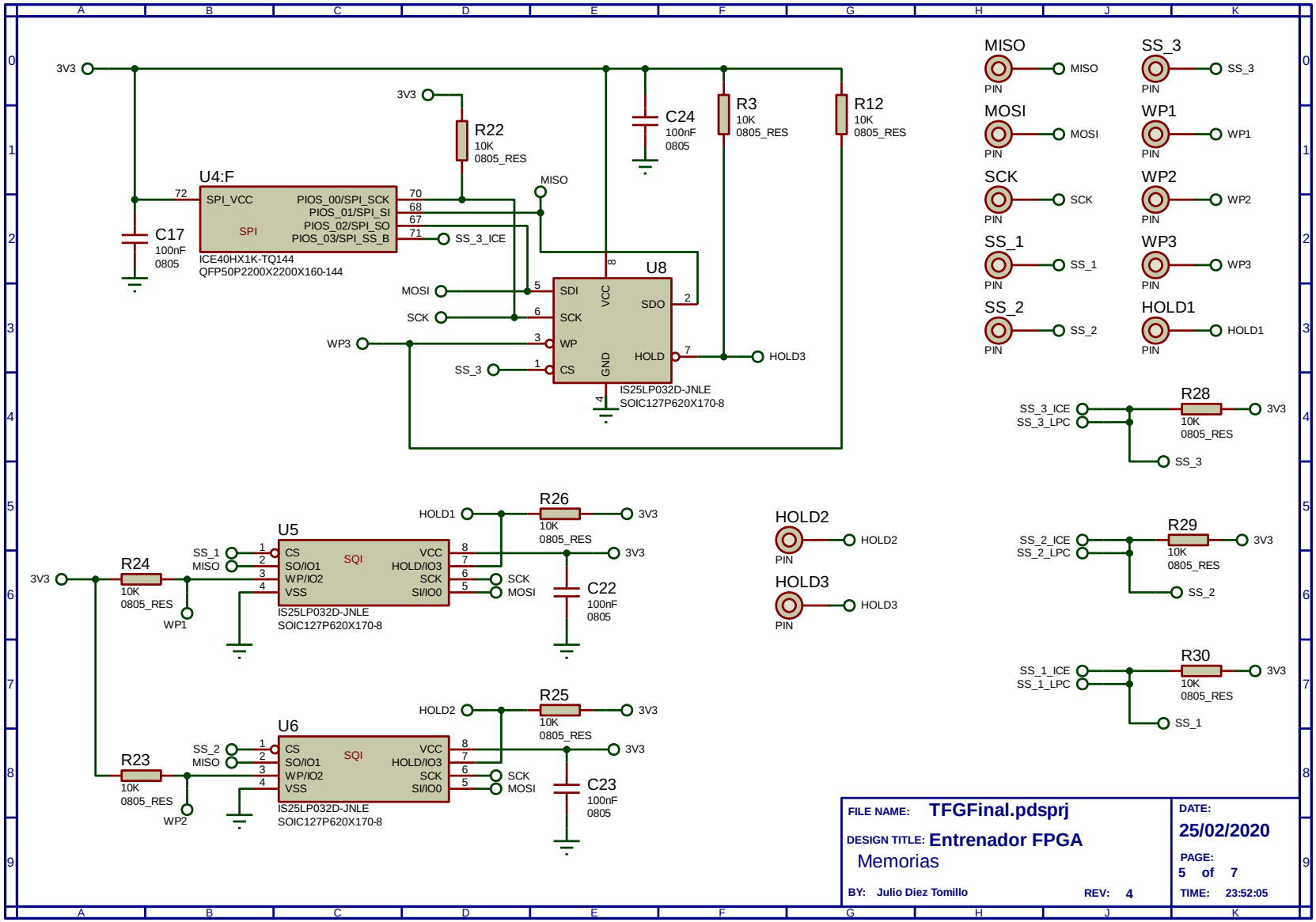
ESQUEMÁTICO DEL CIRCUITO IMPRESO


U4:A
U4:B
U4:C
U4:D
3V3
C4 100nF 0805
C3 100nF 0805
C6 100nF 0805
C5 100nF 0805
C7 100nF 0805
C8 100nF 0805
C9 100nF 0805
C10 100nF 0805
R4 1K 0805_RES
R20 10K 0805_RES
R2 10K 0805_RES
R21 10K 0805_RES
BANK 0
BANK 1
BANK 2
BANK 3
BOOTLOADER/CLK
DCLK
PIN
ICE_DONE
ICE_RESET
ICE40HX1K-TQ144
QFP50P2200X2200X160-144
FILE NAME: TFGFinal.pdsprj
DESIGN TITLE: Entrenador FPGA
FPGA
BY: Julio Diez Tomillo
REV: 4
DATE: 25/02/2020
PAGE: 1 of 7
TIME: 23:52:05


U4:E
POWER
GND_01
GND_02
GND_03
GND_04
GND_05
GND_06
GND_07
GND_08
GND_09
PLLVCC
PLLGND
VPP_FAST
VPP_2V5
NC_18
NC_17
NC_16
NC_15
NC_14
VCC_01
VCC_02
VCC_03
VCC_04
ICE40HX1K-TQ144
QFP50P2200X2200X160-144
1V2
R1
100R
0805_RES
C2
10uF
0805
C1
100nF
0805
D2
B0520WS-7F
SOD-323
3V3
C21
100nF
0805
C32
C31
C30
C29
J1
Vcc
D-
D+
NC
GND
GND2
USB D-
USB D+
MOLEX USB MINI B 67503-1230
USB MINI-8 SMT
L2
MMZ1608B121C
0603[1608]
5V
C18
100nF
0805
U1
Vin
Vin
SHDN
GND
Vout
ADJ
CDelay
PWRGD
MCP1725-ADJE/SN
SOIC127P600X175-8
C19
4.7uF
0805
R10
68K
0805_RES
C15
1uF
0805
R11
33K
0805_RES
C38
1uF
0805
R9
68K
0805_RES
U2
Vc
FB
Test
SS
Vsw
PGND
AGND
Vcc
CS5171GDR8G
SOIC127P600X175-8
D1
MBRS120T3G
DIOM5336X240
10V
L3
22uH
6028
C13
10nF
0805
C12
22uF
0805
C11
22uF
0805
R8
5K
0805_RES
R5
10K
0805_RES
U3
C14
4.7uF
0805
R6
82K
0805_RES
C16
1uF
0805
R7
12K
0805_RES
J5
5V_Ext
CONN-SIL2
5V_EXT
PIN
FILE NAME: TFGFinal.pdsprj
DESIGN TITLE: Entrenador FPGA
Alimentacion
BY: Julio Diez Tomillo
REV: 4
DATE:
25/02/2020
PAGE:
2 of 7
TIME: 23:52:05


5V
C33
4.7uF
0805
C34
100nF
0805
U9
VCCIO
VCC
USBDP
USBDM
RESET
VCCD
NC
NC
DNU
GND
GND
GND
TXD
RXD
CTS
RTS
DTR
DSR
DCD
RI
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
CY7C65213-28PVXI
SSOP28
USB D+
USB D-
C44
1uF
0805
3V3
R15
10K
0805_RES
R13
470R
0805_RES
BOOTLOADER
R14
470R
0805_RES
RESET#
R27
10K
0805_RES
3V3
U7
RESET#
BOOTLOADER/CLK
SCK
MISO
MOSI
SS_1_LPC
SS_2_LPC
SS_3_LPC
PIO3_LPC
PIO4_LPC
PIO5_LPC
PIO7_LPC
TXD
RXD
PIO0_LPC
PIO1_LPC
PIO2_LPC
PIO6_LPC
RESET/PIO0_0
PIO0_1/CLKOUT/CT32B0_MAT2
PIO0_2/SSEL0/CT16B0_CAP0
PIO0_3
PIO0_4/SCL
PIO0_5/SDA
PIO0_6/SCK0
PIO0_7/CTS
PIO0_8/MISO0/CT16B0_MAT0
PIO0_9/MOSI0/CT16B0_MAT1
SWCLK/PIO0_10/SCK0/CT16B0_MAT2
R/PIO0_11/AD0/CT32B0_MAT3
R/PIO1_0/AD1/CT32B1_CAP0
R/PIO1_1/AD2/CT32B1_MAT0
R/PIO1_2/AD3/CT32B1_MAT1
SWDIO/PIO1_3/AD4/CT32B1_MAT2
PIO1_4/AD5/CT32B1_MAT3/WAKEUP
PIO1_5/RTS/CT32B0_CAP0
PIO1_6/RXD/CT32B0_MAT0
PIO1_7/TXD/CT32B0_MAT1
PIO1_8/CT16B1_CAP0
PIO1_9/CT16B1_MAT0
PIO1_10/AD6/CT16B1_MAT1
PIO1_11/AD7
PIO2_0/DTR/SSEL1
PIO2_1/DSR/SCK1
PIO2_2/DCD/MISO1
PIO2_3/RI/MOSI1
PIO2_4
PIO2_5
PIO2_6
PIO2_7
PIO2_8
PIO2_9
PIO2_10
PIO2_11/SCK0
PIO3_0/DTR
PIO3_1/DSR
PIO3_2/DCD
PIO3_3/RI
PIO3_4
PIO3_5
XTALIN
XTALOUT
VCC
VCC
GND
GND
ICE_DONE
ICE_RESET
LPC1114FBD48/301
SQFP48-7X7
X1
QCS12.0000F18B23M
12MHz
XTAL_12000MHZB2T
C25
18pF
0805
C26
18pF
0805
CLK
C20
100nF
0805
C27
100nF
0805
3V3
CLK
5V
D+
D-
BOOTLOADER
RESET
ICE_DONE
ICE_RESET
TXD
RXD
GND
PIN
FILE NAME: TFGFinal.pdsprj
DESIGN TITLE: Entrenador FPGA
CPU y USB-UART
BY: Julio Diez Tomillo
REV: 4
DATE: 25/02/2020
PAGE: 3 of 7
TIME: 23:52:05


J4
3V3
C42
100nF
0805
VCC0
GND0
PIO0
PIO1
PIO2
PIO3
PIO4
PIO5
PIO6
PIO7
PIO8
PIO9
PIO10
PIO11
PIO12
PIO13
PIO14
PIO15
PIO16
PIO17
PIO18
PIO19
PIO20
PIO21
PIO22
PIO23
PIO24
PIO25
PIO26
PIO27
PIO28
PIO29
PIO30
PIO31
GND1
VCC1
FEMALE HEADER 1X36
TE_38269266
3V3
C43
100nF
0805
J6
3V3
C28
100nF
0805
PIO0_LPC
PIO1_LPC
PIO2_LPC
PIO3_LPC
PIO4_LPC
PIO5_LPC
PIO6_LPC
PIO7_LPC
VCC
GND
PIO0
PIO1
PIO2
PIO3
PIO4
PIO5
PIO6
PIO7
FEMALE HEADER 1X10
TE_1872240
J2
3V3
C36
100nF
0805
C35
100nF
0805
Vcc
Vcc
Reserve
Reserve
Reserve
Reserve
GND
B5
B4
B3
B2
B1
B0
R0
R1
R2
R3
R4
R5
G0
G1
G2
G3
G4
G5
DCLK
DE
4-1734592-0
TE_417345920
10V
R31
10R
0805_RES
C37
100nF
0805
J3
VLED+
VLED-
SM02B-BHSS-1-TB
SM02-BBHSS-1TB
LED1
LED2
LED3
LED4
R16
R17
R18
R19
130R
0805_RES
D4
D6
D5
D3
KPT-3216EC
KPT-3216SGC
LEDC3216X110
FILE NAME: TFGFinal.pdsprj
DESIGN TITLE: Entrenador FPGA
TFT y Conectores
BY: Julio Diez Tomillo
REV: 4
DATE: 25/02/2020
PAGE: 4 of 7
TIME: 23:52:05


3V3
R22
10K
0805_RES
C24
100nF
0805
R3
10K
0805_RES
R12
10K
0805_RES
MISO
U4:F
SPI_VCC
SPI
PIOS_00/SPI_SCK
PIOS_01/SPI_SI
PIOS_02/SPI_SO
PIOS_03/SPI_SS_B
SS_3_ICE
C17
100nF
0805
ICE40HX1K-TQ144
QFP50P2200X2200X160-144
U8
SDI
SCK
WP
CS
VCC
GND
SDO
HOLD
MOSI
SCK
WP3
SS_3
HOLD3
IS25LP032D-JNLE
SOIC127P620X170-8
MISO
SS_3
MOSI
WP1
SCK
WP2
SS_1
WP3
SS_2
HOLD1
PIN
R28
SS_3_ICE
SS_3_LPC
10K
0805_RES
R26
HOLD1
U5
SQI
CS
SO/IO1
WP/IO2
VSS
VCC
HOLD/IO3
SCK
SI/IO0
SS_1
MISO
R24
WP1
C22
100nF
0805
HOLD2
HOLD3
R29
SS_2_ICE
SS_2_LPC
SS_2
R25
U6
SS_2
R23
WP2
C23
R30
SS_1_ICE
SS_1_LPC
SS_1
FILE NAME: TFGFinal.pdsprj
DESIGN TITLE: Entrenador FPGA
Memorias
BY: Julio Diez Tomillo
REV: 4
DATE:
25/02/2020
PAGE:
5 of 7
TIME: 23:52:05

```
@1V2
FPGA  ICCpeak                       6.9  mA
-----------------------------------------------------
                            TOTAL   6.9  mA

@3V3
TFT  Pantalla      150 mA typ.    200.00 mA máx.
x3 IS25LP032D  25 mA ->            75.0  mA
x4 LEDs        10 mA ->            40.00 mA
x4 FPGA ICCbank_peak 6.8->         27.2  mA
FPGA ICC_pll_peak                   1.8  mA
LPC1114 @ 48 MHz                    9.0  mA
-----------------------------------------------------
                            TOTAL 343    mA

@5V
Corriente @1V2                      6.9  mA   regulador lineal
regulador 1V2 (interno)             0.12 mA
Corriente @3V3                    343    mA   regulador lineal
regulador 3V3 (interno)             0.12 mA
CY7C65213                          18.0  mA
-----------------------------------------------------
                          TOTAL  368.14 mA


@10V (Proceden de 5V_Ext)
TFT  LEDs
 160 mA typ.
 200 mA máx.

Si ponemos 200 mA -> potencia = 2.0  W
eficiencia ~90%   -> 2.0/0.9 -> 2.22 W
que salen de los 5V, luego 2.22W/5V -> 444 mA

si ponemos 160mA  -> potencia = 1.6W
                  -> 1.6W/0.9 = 1.77W
que a 5V -> 1.77W/5V -> 355 mA


@5V_Ext
regulador CS5171                    8.00 mA   consumo interno
Corriente @10V                    355.0  mA   fuente conmutada
-----------------------------------------------------
                          TOTAL  363.0  mA

Si solo tuvieramos la alimentacion del USB (500 mA a 5V), no llegaría para
alimentar todo el sistema. Por ello, si queremos usar el TFT BL (355 mA),
tenemos que alimentarlo externamente.
```

FILE NAME: TFGFinal.pdsprj		DATE: 25/02/2020
DESIGN TITLE: Entrenador FPGA Consumo eléctrico		PAGE: 6 of 7
BY: Julio Diez Tomillo	REV: 4	TIME: 23:52:05

ANALISIS TERMICO

Para todos los cálculos suponemos que la temperatura ambiente
es de 25ºC.
La temperatura Tj la calculamos como: Tj = Ta + P*Theta_JA.

-CY7C65213 (SSOP)
Theta_JA = 62 ºC/W
P = 5V*18 mA = 0.09 W

Tj = 30.58 ºC
Tjmax = 85 ºC

-CS5171 (SOIC-8)
Theta_JA = 165 ºC/W
P = Pbias + Pdriver + Psat
Pbias = 0.0275 W
Pdriver = 0.0148 W
Psat = 0.0197 W
P = 0.062 W

Tj = 35.23 ºC
Tjmax = 150 ºC

-ICE40HX1K-TQ144
Theta_JA = 38.5 ºC/W
P = 1.2V*6.9 mA + 3.3V*29 mA = 0.104 W

Tj = 29 ºC
Tjmax = 125 ºC

-MCP1725-ADJE/SN (SOIC-8) @1V2
Theta_JA = 163 ºC/W
P = (5-1.2)V*6.9 mA = 0.026 W

Tj = 29.27 ºC
Tjmax = 125 ºC

-MCP1725-ADJE/SN (SOIC-8) @3V3
Theta_JA = 163 ºC/W
P = (5-3.3)V*343.12 mA = 0.5833 W

Tj = 120 ºC
Tjmax = 125 ºC

-LPC1114FBD48/301
Theta_JA = 82.1 ºC/W (JEDEC)
P = 3.3V*9 mA = 0.0297 W

Tj = 27.44 ºC
Tjmax = 125 ºC

-IS25LP032D-JNLE (SOIC-8)
P = 3.3*25 mA = 0.083 W

Tjmax = 105 ºC
No hemos podido obtener informacion sobre su Resistencia térmica (Theta_JA).
Por lo que no hemos podido realizar el análisis térmico.

-MBRS120T3G
Theta_JL = 12 ºC/W
P = 0.6V*160 mA = 0.096 W

Tj = 26.15 ºC
Tjmax = 125 ºC

-SLF6028T-220MR77-PF
P = 0.104*1.3^2 A = 0.176 W

Tjmax = 85 ºC
No hemos podido obtener informacion sobre su Resistencia térmica (Theta_JA).
Por lo que no hemos podido realizar el análisis térmico.

Únicamente con el MCP1725 de 3V3 nos acercamos a la temperatura
límite del elemento, por lo que tendremos que hacer una aleta
de disipación.

FILE NAME: TFGFinal.pdsprj	DATE: 25/02/2020
DESIGN TITLE: Entrenador FPGA Analisis Termico	PAGE: 7 of 7
BY: Julio Diez Tomillo REV: 4	TIME: 23:52:05

ANEXO 3

BILL OF MATERIALS (BOM)

TFG - INGENIERÍA DE TELECOMUNICACIONES

Bill Of Materials for Entrenador FPGA

Design Title Entrenador FPGA
Author Julio Diez Tomillo
Document Number
Revision 4
Design Created domingo, 16 de febrero de 2020
Design Last Modified lunes, 17 de febrero de 2020
Total Parts In Design 124

41 Capacitors

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
28	C1,C3-C10,C17-C18,C20-C24,C27-C32, C34-C37,C42-C43	0805	100nF	1650866	€0,05
1	C2	0805	10uF	1463362	€0,16
2	C11-C12	0805	22uF	1108326	€0,21
1	C13	0805	10nF	1650861	€0,04
3	C14,C19,C33	0805	4.7uF	2409055	€0,12
4	C15-C16,C38,C44	0805	1uF	2409048	€0,06
2	C25-C26	0805	18pF	2392190	€0,02
Sub-totals:					€2,64

31 Resistors

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
1	R1	0805_RES	100R	2447552	€0,01
16	R2-R3,R5,R12,R15,R20-R30	0805_RES	10K	2447553	€0,01
1	R4	0805_RES	1K	2447587	€0,01
1	R6	0805_RES	82K	2447729	€0,01
1	R7	0805_RES	12K	2447563	€0,01
1	R8	0805_RES	5K	2616759	€0,07
2	R9-R10	0805_RES	68K	2447706	€0,01
1	R11	0805_RES	33K	2447639	€0,01
2	R13-R14	0805_RES	470R	2447662	€0,01
4	R16-R19	0805_RES	130R	2447568	€0,01
1	R31	0805_RES	10R	2447556	€0,01
Sub-totals:					€0,27

9 Integrated Circuits

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
2	U1,U3	SOIC127P600X175-8	MCP1725-ADJE/SN	1834878	€0,47
1	U2	SOIC127P600X175-8	CS5171GDR8G	2534179	€1,60
1	U4	QFP50P2200X2200X160-144	ICE40HX1K-TQ144	2362849	€4,97
3	U5-U6,U8	SOIC127P620X170-8	IS25LP032D-JNLE	2787053	€1,32
1	U7	SQFP48-7X7	LPC1114FBD48/301	1786289	€1,85
1	U9	SSOP28	CY7C65213-28PVXI	2767866	€1,90
Sub-totals:					€15,21

6 Diodes

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
1	D1	DIOM5336X240	MBRS120T3G	1459069	€0,24
1	D2	SOD-323	B0520WS-7F	1773478	€0,28
2	D3,D5	LEDC3216X110	KPT-3216SGC	2099247	€0,10

2	D4, D6	LEDC3216X110	KPT-3216EC	2099245	€0,13
Sub-totals:					€0,98

6 Connectors

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
1	J1	USB MINI-8 SMT	MOLEX USB MINI B 67503-1230	2313554	€0,99
1	J2	TE_417345920	4-1734592-0	1816399	€0,92
1	J3	SM02-BBHSS-1TB	SM02B-BHSS-1-TB	2399341	€1,04
1	J4	TE_38269266	FEMALE HEADER 1X36	1022264	€0,84
1	J5	CONNSIL2	CONN-SIL2	2527547	€0,32
1	J6	TE_1872240	FEMALE HEADER 1X10	1022261	€0,33
Sub-totals:					€4,43

2 Inductors

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
1	L2	0603[1608]	MMZ1608B121C	2133951	€0,07
1	L3	6028	22uH	1670015	€1,11
Sub-totals:					€1,18

1 Crystal

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
1	X1	XTAL_12000MHZB2T	QCS12.0000F18B23M	2508484	€0,35
Sub-totals:					€0,35

28 Miscellaneous

Quantity	References	PCB Package	Value	Stock Code	Unit Cost
28	10V, 1V2, 3V3, 5V, 5V_EXT, BOOTLOADER, CLK, D+, D-, DCLK, GND, HOLD1-HOLD3, ICE_DONE, ICE_RESET, MISO, MOSI, RESET, RXD, SCK, SS_1-SS_3, TXD, WP1-WP3	PIN	PIN		
Sub-totals:					€0,00

Totals:	€25,06

lunes, 17 de febrero de 2020 0:10:31

ANEXO 4

CÓDIGO DEL PROGRAMA main.c

Este código se carga en el microcontrolador y va a ser el programa que ejecute este. La función principal es la de recibir el programa de la FPGA desde el ordenador y grabarlo en la memoria y dirección que se le diga. Después levantará el bloqueo de la FPGA para que lea el programa de la memoria y se configure correctamente. Una vez realizado esto se quedará el microcontrolador en modo *IDLE*.

```
/* Main.c file generated by New Project wizard
 *
 * Autor:           Julio Díez Tomillo
 * Created:         Enero. 28 2020
 * Processor:       LPC1114FBD48/301
 * Compiler:        GCC for ARM
 */

#include <LPC11xx.h>
// ----------------------------------------------------------------------------
#define P1_0    (unsigned short)(1<<0)
#define P1_1    (unsigned short)(1<<1)

#define P0_11   (unsigned short)(1<<11)

// ----------------------------------------------------------------------------
extern uint32_t SystemCoreClock;
extern int _printf(const char *format, ...);
// ----------------------------------------------------------------------------
//////////////////////////////////////////////////////////////
Retardo Activo
// Ojo, cuando se ejecuta desde Flash el alineamiento influye
// mucho en la velocidad de ejecuci? del bucle
void __attribute__ ((naked, aligned(8))) delay_loop(unsigned int d)
{
  asm volatile ("1: sub r0,#1\n" "   bne 1b\n" "   bx  lr\n");
}
#define _delay_us(n) delay_loop((n*(SystemCoreClock/1000)-6000)/4000)
#define _delay_ms(n) delay_loop((n*(SystemCoreClock/1000)-6)/4)
// ----------------------------------------------------------------------------
// UART putch & getch
void U0putch(int d)
{
 while ((((signed int) LPC_UART->LSR) << (31 - 5)) >= 0) ;
 LPC_UART->THR = d;
}
// ----------------------------------------------------------------------------
int U0getch()
{
```

```
 while(!(LPC_UART->LSR&1));
 return LPC_UART->RBR;
}
// ------------------------------------------------------------------------
// La memoria 23LC1024 tiene los siguientes comandos
// Las direcciones son de 24 bits, las p?inas de 32 bytes
#define READ        (0x03)
#define WRITE       (0x02)
#define WRITE_ENABLE  (0x06)
#define WRITE_DISABLE (0x04)
#define SUBSECTOR_ERASE (0x20)
#define SECTOR_ERASE  (0xD8)
#define BULK_ERASE    (0xC7)
#define READ_ID       (0x9E)
#define READ_STATUS   (0x05)
// ------------------------------------------------------------------------
int SPI_EnviaRecibe(int d)
{
 int i;
 LPC_SSP0->DR = d;
 while((LPC_SSP0->SR) & 0x10); // Espera a transmitir bit BSY
 i = LPC_SSP0->DR;
 return i;
}
// ------------------------------------------------------------------------
#define SS2_OFF LPC_GPIO1->DATA &= ~P1_0;
#define SS2_ON  LPC_GPIO1->DATA |=  P1_0;
#define SS3_OFF LPC_GPIO1->DATA &= ~P1_1;
#define SS3_ON  LPC_GPIO1->DATA |=  P1_1;
#define SS1_OFF LPC_GPIO0->DATA &= ~P0_11;
#define SS1_ON  LPC_GPIO0->DATA |=  P0_11;
#define ICERESET_OFF LPC_GPIO2->DATA &= ~(1<<5);
#define ICERESET_ON  LPC_GPIO2->DATA |=  (1<<5);

int memoria = 3;

void SPI_Exit_XIP()
{
 SS_OFF();
```

```
 SPI_EnviaRecibe(0xFF); // Salimos del modo XIP
 SPI_EnviaRecibe(0xFF);
 SPI_EnviaRecibe(0xFF);
 SPI_EnviaRecibe(0xFF);
 SS_ON();
}

// Deselecciona las memorias SPI
void SS_OFF()
{
 if(memoria==1)
  {
    SS1_OFF;
  }
 else if(memoria==2)
  {
    SS2_OFF;
  }
 else if(memoria==3)
  {
    SS3_OFF;
  }
}
// ---------------------------------------------------------------------------

// Selecciona la memoria elegida
void SS_ON()
{
 if(memoria==1)
 {
    SS1_ON;
 }
 else if(memoria==2)
 {
  SS2_ON;
 }
 else if(memoria==3)
  {
    SS3_ON;
  }
```

```
}

//Leemos la memoria activa
int SPI_Read(int dir, int n, char *data)
{
 int i;
 SS_OFF();
 SPI_EnviaRecibe(READ);
 SPI_EnviaRecibe((dir>>16)&0xFF);
 SPI_EnviaRecibe((dir>> 8)&0xFF);
 SPI_EnviaRecibe((dir>> 0)&0xFF);
 for(i=0;i<n;i++) data[i] = SPI_EnviaRecibe(0);
 SS_ON();
 return n;
}

//Leemos el estado de la memoria seleccionada
int SPI_Read_Status()
{
 int d;
 SS_OFF();
 SPI_EnviaRecibe(READ_STATUS);
 d = SPI_EnviaRecibe(0);
 // Bit 7 Status Register write enable (Non volatile)
 // Bit 5 Protected memory at top/bottom (0/1) (Non volatile)
 // Bit 6,4,3,2 Block protected (1/0) (Non volatile)
 // Bit 1 Write enable latch set/clear (1/0) (Volatile)
 // Bit 0 Write in progress (Volatile)
 SS_ON();
 return d;
}

// Borra la memoria seleccionada
// Se puede borrar un subsector (4kB), un sector (64kB)
// o hacer un Bulk Erase (8MB)
void SPI_Erase(int cmd, int d)
{
 SPI_Write_Enable();
 SS_OFF();
```

```
 SPI_EnviaRecibe(cmd);
 if(cmd==SUBSECTOR_ERASE || cmd==SECTOR_ERASE)
 {
  SPI_EnviaRecibe(d>>16);
  SPI_EnviaRecibe(d>> 8);
  SPI_EnviaRecibe(d&0xFF);
 }
 SS_ON();

 if(cmd==BULK_ERASE)
 {
  while(SPI_Read_Status()&0x01)
  {
       _printf(".");
       _delay_ms(1000);
  }
 }
 else
  while(SPI_Read_Status()&0x01);
}

//Activamos la memoria para escribir
void SPI_Write_Enable()
{
 SS_OFF();
 SPI_EnviaRecibe(WRITE_ENABLE);
 SS_ON();
}

void SPI_VolatileConf()
{
 SPI_Write_Enable();
 SS_OFF();
 SPI_EnviaRecibe(0x81); // Write Volatile Conf
 SPI_EnviaRecibe(0x8F); // 8 dummy cyles, no XIP,continuous  sequential read
mode
 SS_ON();
}

// ---------------------------------------------------------------------------
```

```
// Escribimos en la memoria
int SPI_Write(int dir, int n, char *data)
{
 // Tamaño de la pagina (256B)
 int i,direccion;
 SPI_Write_Enable();
 SS_OFF();
 SPI_EnviaRecibe(WRITE);
 SPI_EnviaRecibe((dir>>16)&0xFF);
 SPI_EnviaRecibe((dir>> 8)&0xFF);
 SPI_EnviaRecibe((dir>> 0)&0xFF);
 for(i=0;i<n;i++) SPI_EnviaRecibe(data[i]);
 SS_ON();
 return n;
}


int main(void)
{
 int dir,num,i,carac,direccion;
 char D[32]="Hola esto es una prueba";
 char Mensaje[256];

 //Configuramos los pines del uC con la configuracion deseada

 //Activamos el pin clkOut y lo configuramos
 LPC_SYSCON->CLKOUTCLKSEL = 3;
 LPC_SYSCON->CLKOUTDIV = 1;
 LPC_SYSCON->CLKOUTUEN = 0;
 LPC_SYSCON->CLKOUTUEN = 1;

 LPC_IOCON->PIO0_1 = 1;      // Función CLKOUT

 LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16); //BIT16;
 // Release all peripheral resets
 LPC_SYSCON->PRESETCTRL=0x0F;


 // Configura UART
```

```
 // UART. See "Enabling sequence for UART clock" on LPC111x User Manual
 // first assign pin functions for UART
 LPC_IOCON->PIO1_6 = 1|(2<<3); // RXD, pull-up
 LPC_IOCON->PIO1_7 = 1; // TXD
 // then enable the UART clock
 LPC_SYSCON->UARTCLKDIV = 1;     // then enable the UART clock
 LPC_SYSCON->SYSAHBCLKCTRL = 0x7FFFF;  // Clock for all
//

 #define BAUD 115200
 #define UDIVI ((SystemCoreClock/BAUD+8)/16)
 LPC_UART->LCR = 0x80;
 LPC_UART->DLM = UDIVI>>8;
 LPC_UART->DLL = UDIVI&255;
 LPC_UART->LCR = 0x3;    // 8-bit
 LPC_UART->FCR = 7;      // con FIFOS, nivel IQR en 1

 // Configura SPI
 LPC_IOCON->PIO0_6 = 2;   // SCK
 LPC_IOCON->SCK_LOC = 2;  // SCK en PIO_0.6
 LPC_IOCON->PIO0_8 = 1;   // MISO
 LPC_IOCON->PIO0_9 = 1;   // MOSI

 LPC_SYSCON->SSP0CLKDIV = 1;

 LPC_SSP0->CR0 = 7|(0<<6)|(0<<8); // 8 bit, SPI,CLK/1, CPOL=0
 LPC_SSP0->CR1 = 2;       // SSP enabled, master
 //LPC_SSP0->CPSR = 48;     // 1 MHz, entre 2 y 254, par, divide PCLK entre
este valor
   LPC_SSP0->CPSR = 2;
 // Configurar pines como GPIO
 LPC_IOCON->JTAG_TMS_PIO1_0  = 0x01 | (1<<7); //Configure as Digital IO
 LPC_IOCON->JTAG_TDO_PIO1_1  = 0x01 | (1<<7); //Configure as Digital IO
 LPC_IOCON->JTAG_TDI_PIO0_11 = 0x01 | (1<<7); //Configure as Digital IO
 LPC_IOCON->PIO2_4 = 0;   //ICEDONE
 LPC_IOCON->PIO2_5 = 0;   //ICERESET

 // Configurar direccion
 LPC_GPIO1->DIR  |= P1_0 | P1_1; //Configure as Output
 LPC_GPIO0->DIR  |= P0_11;       //Configure as Output
```

```
 LPC_GPIO2->DIR  |= (1<<5);
 _delay_ms(100);
 //_printf("Mensaje por la consola\n");
_delay_ms(20);
 ICERESET_ON;
 SS1_ON;
 SS2_ON;
 SS3_ON;
 ICERESET_OFF;
 SPI_Exit_XIP();; //Bloqueamos la FPGA para que no interfiera

 // Recibimos la trama enviada por el ordenador
 carac = U0getch();
 if(carac == 'I')
 {
       // Guardamos la memoria, la direccion y el tamaño recibido
       memoria = U0getch();
       dir = (U0getch()<<16);
       dir|= (U0getch()<<8);
       dir|= U0getch();
       num = (U0getch()<<16);
       num|= (U0getch()<<8);
       num|= U0getch();

  direccion = dir;


       //Iniciamos el borrado. La memoria tiene 128 sectores de
       //64kB cada uno y 2048 subsectores de 4kB. Memoria total
       //es de 8MB
       if(num<4096)
       {
             SPI_Erase(SUBSECTOR_ERASE,dir);
       }
       else if(num<=65536)
       {
             SPI_Erase(SECTOR_ERASE,dir);
       }
       else if(num<=127*65536)
```

```
	{
		for(i=0;i<(num/65536);i++)
		{
			SPI_Erase(SECTOR_ERASE,dir+i*65536);
		}
	}
	else
	{
		SPI_Erase(BULK_ERASE,0);
	}

	//Avisamos que todo ha funcionado correctamente
	U0putch('O');

	//Pasamos a recibir el archivo y a grabarlo en la memoria
	//El mensaje lo recibiremos en 256B por cada envio
do
	{

		if(num>=256)
		{
			for(i=0;i<256;i++)
			{
				Mensaje[i] = U0getch();
			}

		}
		else
		{
			for(i=0;i<num;i++)
			{
				Mensaje[i] = U0getch();
			}
		}
  num-= 256;
		SPI_Write(dir,256,Mensaje);
		//Esperamos mientras la memoria esta escribiendo
		//el mensaje (el ultimo bit nos dice si esta escribiebdo)
		while(SPI_Read_Status()&1)
```

```
            {
                  _delay_ms(1);
            }

            //Anunciamos al ordenador que hemos acabado y le decimos
            //que nos mande el proximo paquete del archivo
            U0putch('O');
            dir = dir + 256;

 }while(num>0);
}

 _printf("Checking.\n");

// Realizamos comprobación para ver que se ha escrito adecuadamente
// en la memoria
num=0;
do
{
 SPI_Read(direccion,32,D);
 while(SPI_Read_Status()&0x01); {_printf("."); _delay_ms(1);};
 for(i=0;i<32;i++)
 {
  num^=(unsigned int)D[i];
 };
 //_printf("\n");
 direccion+=32;
 }while(direccion<dir);

_printf("\nXored checksum: %02x\n",num);
 SPI_VolatileConf(); // Preparo la memoria

// Desactivo pines SPI master
LPC_IOCON->PIO0_6 = 0;   // SCK
LPC_IOCON->PIO0_8 = 0;   // MISO
LPC_IOCON->PIO0_9 = 0;   // MOSI
LPC_GPIO0->DIR &= ~((1<<6)|(1<<8)|(1<<9)); //Configure as Inputs
SCK,MISO,MOSI
LPC_GPIO1->DIR &= ~P1_1; //Configure as Input SS_3
```

```
 _delay_ms(10);
 // Activo FPGA
 ICERESET_ON;

 // Esperamos a CDONE
 _delay_us(100);

 while(!(LPC_GPIO2->DATA & (1<<4))); // Espera a que suba

 #define ESC 27
 _printf("\nRun FPGA%c\n",ESC);
 while(1);
 return 0;
}
```

ANEXO 5

CÓDIGO DEL PROGRAMA main.cpp

Este código se ejecuta en el ordenador y va a realizar una comunicación con el microcontrolador. El objetivo de este código es enviar el programa de la FPGA al microcontrolador para que lo grabe en la memoria. Una vez enviado el archivo completo y comprobado que el microcontrolador lo ha recibido correctamente se cerrará el puerto y se terminará el programa.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "windows.h"
// Variables necesarias
HANDLE idComDev;
DCB dcb = {0};
#define ESC "\x1B"
//---------------------------------------------------------------------
//Funcion para abrir los puertos COM
bool AbrirCOM(char * COM_PORT)
{
  idComDev = CreateFile(COM_PORT, GENERIC_READ | GENERIC_WRITE,
                  0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if(idComDev ==INVALID_HANDLE_VALUE)   {
     //printf("Can't open %s port\n",COM_PORT);

     return false;
  }

  dcb.DCBlength=sizeof(dcb);

  if(!GetCommState(idComDev, &dcb))
  {
     printf("Can't obtain state from serial port\n");
     return false;
  }
//   dcb.BaudRate = (DWORD) CBR_115200;
//   dcb.BaudRate = (DWORD) 230400;
  dcb.BaudRate = (DWORD) 115200;
  dcb.ByteSize = (BYTE) 8;
  dcb.Parity   = (BYTE) NOPARITY;
  dcb.StopBits = (BYTE) ONESTOPBIT;

  if(!SetCommState(idComDev, &dcb))
  {
```

```
      printf("Can't set state, SetCommStatus\n");
      return false;
   }

   COMMTIMEOUTS timeouts={0};
   timeouts.ReadIntervalTimeout = 50;
   timeouts.ReadTotalTimeoutConstant = 50;
   timeouts.ReadTotalTimeoutMultiplier = 10;
   timeouts.WriteTotalTimeoutConstant = 50;
   timeouts.WriteTotalTimeoutMultiplier = 10;

   if(!SetCommTimeouts(idComDev,&timeouts))
    {printf("Can't set timeouts\n");return false;};

   return true;
 }
//---------------------------------------------------------------------
 void CerrarCOM()
 {
   CloseHandle(idComDev);
 }
 //---------------------------------------------------------------------
 void EscribeCOM(char *Buffer, int longitud)
 {
   DWORD l, n, p = 0;
   if(longitud==0)   l = strlen(Buffer);
   else              l = longitud;
   while(l)
   {
    if(!WriteFile(idComDev, &Buffer[p], l, &n, NULL)) return;
    p += n; l -= n;
   }
 }
//---------------------------------------------------------------------
 void LeeCOM(char *buf, int m=1) // Recibe datos
 {
   DWORD p =0,n = 0;
```

```
   while( m ) // Hasta que no recibas algo no retorna
    {
     ReadFile(idComDev, &buf[p], m, &n, NULL);
     p += n;
     m -= n;
     //L_Delay(10);
    }
    if(m!=0) buf[0]=0xf0; // Escape pulsado
 }
//---------------------------------------------------------------------
bool GetCTS()
{
 LPDWORD V;
 GetCommModemStatus(idComDev ,V);
 if(*V & MS_CTS_ON) return true;
 else               return false;
}
//BOOL GetCommModemStatus (HANDLE hFile, LPDWORD lpModemStat);
//
//returns the status of the modem control signals in the variable
pointed to
// by lpModemStat. The flags returned can be any of the following:
//
//    * MS_CTS_ON Clear to Send (CTS) is active.
//          * MS_DSR_ON Data Set Ready (DSR) is active.
//    * MS_RING_ON Ring Indicate (RI) is active.
//          * MS_RLSD_ON Receive Line Signal Detect (RLSD) is active.

//---------------------------------------------------------------------
unsigned char EnviaComando( char* Buffer, int n=0)
{
 char D;
 EscribeCOM(Buffer,n);
 LeeCOM(&D,1);
 return D;
}
```

```
int   enviarArchivo(char* archivo, int m, int dir)
{
 FILE *f;
 unsigned int tamMax = 8*(1<<20);
 char Contenido[800000];
 char Trama[9] = "Imaaannn";
 int i,n,num;
 char Rec[3];
 char p;

 //Mostramos por pantalla en que memoria y direccion junto con el
archivo a escribir
 printf("Memoria: %d\n", m);
 printf("Archivo: %s\n", archivo);
 printf("Direccion: %d\n", dir);

//Intentamos abrir el archivo seleccionado
 if(f=fopen(archivo,"rb"))
 {
      //Leemos el fichero y guardamos en n el numero de elementos
leidos
      n = fread(Contenido,1,800000,f);
      //Cerramos el fichero
      fclose(f);
      //Notificamos el tamaño del archivo
      printf("Conectado\n");
      printf("Tamano archivo: ");
      printf("%d\n", n);

      /* Mostramos por pantalla el contenido del fichero leido
      for(i=0;i<n;i++)
            printf("%02x ", Contenido[i]);*/

      //Creamos la trama a enviar al uP
      Trama[1] = m;                              //Memoria
      Trama[2] = (dir&0xFF0000)>>16;     //Direccion 8 bits mas
significativos
```

```
      Trama[3] = (dir&0x00FF00)>>8;//Direccion 8 bits centrales
      Trama[4] = dir&0x0000FF;           //Direccion 8 bits menos
significativos
      Trama[5] = (n&0xFF0000)>>16; //Tamaño 8 bits mas significativos
      Trama[6] = (n&0xFF00)>>8;          //Tamaño 8 bits centrales
      Trama[7] = (n&0xFF);               //Tamaño 8 bits menos
significativos

      //Mandamos la trama
      EscribeCOM(Trama,8);

    printf("Trama enviada\n");

    //Esperemos la respuesta del micro
      Rec[0] = '.';
      while(Rec[0]=='.')
      {
            LeeCOM(Rec,1);
            printf(".");
      }
      //Si es una O proseguimos, si no notificamos y salimos
      if(Rec[0]=!'O')
      {
            printf("Ha habido un error\n");
            exit(1);
      }

      printf("\nRecibido el OK se envia archivo\n");
      //Procedemos a enviar el archivo en bloques de 256B
      i=0;
      do
      {
            //Enviamos el bloque
            if(n>256)
            {
                  EscribeCOM(&Contenido[i],256);
                  //printf("Enviado paquete\n");
            }
```

```
            else
            {
                  EscribeCOM(&Contenido[i],n);
                  //printf("Enviado ultimo paquete\n");
            }
            n-=256;
            i+=256;

            //Esperamos la respuesta del micro
            Rec[0] = '.';
            while(Rec[0]=='.')
            {
                  LeeCOM(Rec,1);
                  printf(".");
            }
            //Si es una O proseguimos, si no notificamos y salimos
            if(Rec[0]=!'O')
            {
                  printf("Ha habido un error\n");
                  exit(1);
            }

      } while(n>=0);

      //Notificamos que se ha enviado el archivo correctamente
      printf("Se ha terminado de enviar el archivo\n");

      //Esperamos a que el micro nos envie informacion final, y cuanto
termina volvemos a la rutina principal
      Rec[0]=' ';
      do
      {
            LeeCOM(Rec,1);
            if(Rec[0]==27) break;
            printf("%c", Rec[0]);

            Rec[0] = '.';
```

```
      }while(Rec[0]=='.');
 }
 else
 {
      printf("No se puede abrir el archivo\n");
      exit(1);
 }
 return 0;
}


//---------------------------------------------------------------------
---

int main(int argc, char**argv)
{
 char B[256];
 int memoria=3;
 int direccion=0;
 char archivo[256]="hardware.bin";
 int i,opt;
 char COM[16];
 int resultado;

//Guardamos los argumentos que se han seleccionado al ejecutar el
programa
 while((opt = getopt(argc, argv, "maf")) != -1)
 {
      switch(opt)
  {
      case 'm':
            memoria = atoi(argv[optind]);
            printf("Memoria seleccionada: %d\n", memoria);
            break;
      case 'a':
            direccion = atoi(argv[optind]);
            printf("Direccion seleccionada: %d\n", direccion);
            break;
```

```
      case 'f':
            strcpy(archivo,argv[optind]);
            printf("Archivo a enviar: %s\n", archivo);
            break;
      case '?':
            printf("Opcion desconocida: %c\n", optopt);
            break;
      case ':':
            printf("La opcion necesita un valor\n");
            break;
      default:
            printf("Ha habido un fallo\n");
  }
 }
 //Intentamos abrir los puertos COM
 for(i=40;i>0;i--)
 {
 sprintf(COM,"Com%d", i);
  if(AbrirCOM(COM))
  {
      //Si hemos logrado abrir uno llamamos a la funcion
enviarArchivo()
    printf("Abierto\n");
      resultado = enviarArchivo(archivo, memoria, direccion);
      printf("%d\n", resultado);
      CerrarCOM();      break;
  }
 }
 //Finalizamos notificando si ha sido un exito o no
 if(resultado) return 0;
 return 1;
}
```

ANEXO 6

ARCHIVO pinout.pcf

```
# ---------------------------------------------------------------------------
#- TFG Julio constraint file (.pcf)
#- By Julio Díez
#- December - 2019
#- GPL license
# ---------------------------------------------------------------------------
set_io --warn-no-port D3 56  # output
set_io --warn-no-port D5 60  # output
set_io --warn-no-port D6 61  # output
set_io --warn-no-port D4 62  # output

# ------------ System 12 MHz clock ------------------------------------------
set_io --warn-no-port CLK 129  # input

# ------------ PIO0 as outputs                ---------------------------------
set_io --warn-no-port O0_02  112  # output
set_io --warn-no-port O0_03  113  # output
set_io --warn-no-port O0_04  114  # output
set_io --warn-no-port O0_05  115  # output
set_io --warn-no-port O0_06  116  # output
set_io --warn-no-port O0_07  117  # output
set_io --warn-no-port O0_08  118  # output
set_io --warn-no-port O0_09  119  # output
# Sinonimos ----------------------------------------------------------------
set_io --warn-no-port O_SS_1     112  # output
set_io --warn-no-port O_SS_2     113  # output
set_io --warn-no-port O_WP1      114  # output
set_io --warn-no-port O_HOLD1    115  # output
set_io --warn-no-port O_WP2      116  # output
set_io --warn-no-port O_HOLD2    117  # output
set_io --warn-no-port O_WP3      118  # output
set_io --warn-no-port O_HOLD3    119  # output
# ------------ PIO0 as inputs                 ---------------------------------
set_io --warn-no-port I0_02  112  # input
set_io --warn-no-port I0_03  113  # input
set_io --warn-no-port I0_04  114  # input
set_io --warn-no-port I0_05  115  # input
set_io --warn-no-port I0_06  116  # input
set_io --warn-no-port I0_07  117  # input
set_io --warn-no-port I0_08  118  # input
set_io --warn-no-port I0_09  119  # input
# Sinonimos inputs ---------------------------------------------------------
set_io --warn-no-port I_SS_1     112  # input
set_io --warn-no-port I_SS_2     113  # input
set_io --warn-no-port I_WP1      114  # input
set_io --warn-no-port I_HOLD1    115  # input
set_io --warn-no-port I_WP2      116  # input
set_io --warn-no-port I_HOLD2    117  # input
set_io --warn-no-port I_WP3      118  # input
set_io --warn-no-port I_HOLD3    119  # input

# ------------ J4 connector PIO3 as outputs ---------------------------------
set_io --warn-no-port O3_00 21  # output
set_io --warn-no-port O3_01 20  # output
set_io --warn-no-port O3_08  9  # output
set_io --warn-no-port O3_09 10  # output
set_io --warn-no-port O3_10 11  # output
set_io --warn-no-port O3_11 12  # output
set_io --warn-no-port O3_12 19  # output
```

```
set_io --warn-no-port O3_13 22  # output
set_io --warn-no-port O3_14 23  # output
set_io --warn-no-port O3_15 24  # output
set_io --warn-no-port O3_16 25  # output
set_io --warn-no-port O3_17 26  # output
set_io --warn-no-port O3_18 28  # output
set_io --warn-no-port O3_19 29  # output
set_io --warn-no-port O3_20 31  # output
set_io --warn-no-port O3_21 32  # output
set_io --warn-no-port O3_22 33  # output
set_io --warn-no-port O3_23 34  # output

# ------------ J4 connector PIO3 as inputs ----------------------------------
set_io --warn-no-port I3_00 21  # input
set_io --warn-no-port I3_01 20  # input
set_io --warn-no-port I3_08  9  # input
set_io --warn-no-port I3_09 10  # input
set_io --warn-no-port I3_10 11  # input
set_io --warn-no-port I3_11 12  # input
set_io --warn-no-port I3_12 19  # input
set_io --warn-no-port I3_13 22  # input
set_io --warn-no-port I3_14 23  # input
set_io --warn-no-port I3_15 24  # input
set_io --warn-no-port I3_16 25  # input
set_io --warn-no-port I3_17 26  # input
set_io --warn-no-port I3_18 28  # input
set_io --warn-no-port I3_19 29  # input
set_io --warn-no-port I3_20 31  # input
set_io --warn-no-port I3_21 32  # input
set_io --warn-no-port I3_22 33  # input
set_io --warn-no-port I3_23 34  # input

# ------------ J4 connector PIO2 as outputs       ---------------------------
set_io --warn-no-port O2_00 52  # output
set_io --warn-no-port O2_01 49  # output
set_io --warn-no-port O2_02 50  # output
set_io --warn-no-port O2_03 58  # output
set_io --warn-no-port O2_04 37  # output
set_io --warn-no-port O2_05 38  # output
set_io --warn-no-port O2_06 39  # output
set_io --warn-no-port O2_07 41  # output
set_io --warn-no-port O2_08 42  # output
set_io --warn-no-port O2_09 43  # output
set_io --warn-no-port O2_10 44  # output
set_io --warn-no-port O2_11 45  # output
set_io --warn-no-port O2_12 47  # output
set_io --warn-no-port O2_13 48  # output
set_io --warn-no-port O2_14 56  # output
set_io --warn-no-port O2_15 60  # output
set_io --warn-no-port O2_16 61  # output
set_io --warn-no-port O2_17 62  # output

# ------------ J4 connector PIO2 as inputs       ----------------------------
set_io --warn-no-port I2_00 52  # input
set_io --warn-no-port I2_01 49  # input
set_io --warn-no-port I2_02 50  # input
set_io --warn-no-port I2_03 58  # input
set_io --warn-no-port I2_04 37  # input
set_io --warn-no-port I2_05 38  # input
```

```
set_io --warn-no-port I2_06 39  # input
set_io --warn-no-port I2_07 41  # input
set_io --warn-no-port I2_08 42  # input
set_io --warn-no-port I2_09 43  # input
set_io --warn-no-port I2_10 44  # input
set_io --warn-no-port I2_11 45  # input
set_io --warn-no-port I2_12 47  # input
set_io --warn-no-port I2_13 48  # input
set_io --warn-no-port I2_14 56  # input
set_io --warn-no-port I2_15 60  # input
set_io --warn-no-port I2_16 61  # input
set_io --warn-no-port I2_17 62  # input

# ------------ J2 connector PIO1       ---------------------------------------
#set_io --warn-no-port O1_01 93  # output
#set_io --warn-no-port O1_02 78  # output
#set_io --warn-no-port O1_03 79  # output
#set_io --warn-no-port O1_04 80  # output
#set_io --warn-no-port O1_05 81  # output
#set_io --warn-no-port O1_06 87  # output
#set_io --warn-no-port O1_07 88  # output
#set_io --warn-no-port O1_08 90  # output
#set_io --warn-no-port O1_09 91  # output
#set_io --warn-no-port O1_10 95  # output
#set_io --warn-no-port O1_11 96  # output
#set_io --warn-no-port O1_12 97  # output
#set_io --warn-no-port O1_13 98  # output
#set_io --warn-no-port O1_14 99  # output
#set_io --warn-no-port O1_15 101 # output
#set_io --warn-no-port O1_16 102 # output
#set_io --warn-no-port O1_17 104 # output
#set_io --warn-no-port O1_18 105 # output
#set_io --warn-no-port O1_19 106 # output
#set_io --warn-no-port O1_20 107 # output
# Sinonimos
set_io --warn-no-port DCLK 93 # output
set_io --warn-no-port DE   78 # output
set_io --warn-no-port B5   79 # output
set_io --warn-no-port B4   80 # output
set_io --warn-no-port B3   81 # output
set_io --warn-no-port B2   87 # output
set_io --warn-no-port B1   88 # output
set_io --warn-no-port B0   90 # output
set_io --warn-no-port G5   91 # output
set_io --warn-no-port G4   95 # output
set_io --warn-no-port G3   96 # output
set_io --warn-no-port G2   97 # output
set_io --warn-no-port G1   98 # output
set_io --warn-no-port G0   99 # output
set_io --warn-no-port R5  101 # output
set_io --warn-no-port R4  102 # output
set_io --warn-no-port R3  104 # output
set_io --warn-no-port R2  105 # output
set_io --warn-no-port R1  106 # output
set_io --warn-no-port R0  107 # output

# ------------ PIOS                    ---------------------------------------
#set_io --warn-no-port OS_00 70  # output
#set_io --warn-no-port OS_01 68  # output
```

```
#set_io --warn-no-port OS_02 67  # output
#set_io --warn-no-port OS_03 71  # output
#Sinónimos
set_io --warn-no-port I_SCK 70  # input
set_io --warn-no-port I_MOSI  68  # input
set_io --warn-no-port O_MISO  67  # output
set_io --warn-no-port I_SS_B 71  # input

set_io --warn-no-port O_SCK 70  # output
set_io --warn-no-port O_MOSI  68  # output
set_io --warn-no-port I_MISO  67  # input
set_io --warn-no-port O_SS_B 71  # output
```

ANEXO 7

ERRORES DE DISEÑO

En este anexo vamos a analizar los errores hardware que se encontraron en el diseño de la placa y que hubo que arreglar una vez montada la placa:

- Los pines D- y D+ del conector USB se conectaron de forma invertida en el diseño, por lo que hubo que invertirlos.
- Hay redundancia entre la resistencia R4 y R29, por lo que se procedió a eliminar la resistencia R4.
- La amplitud de la señal de salida del oscilador es insuficiente para la FPGA, por lo que se cortó la vía que lo conectaba. Se conectó una resistencia de 1K al pin de *Bootloader* del microprocesador y se conectó al pin 129 de la FPGA, que es la entrada del reloj. El pin de *Bootloader* tiene también la función de ser señal de reloj de salida del microprocesador, por lo que tenemos que activarlo y configurarlo en la programación.
- En la parte de la fuente conmutada se cortó la pista que conectada el condensador C37 con el conector J3. Se puso una resistencia de 10R de tamaño 1206, conectada al condensador y el otro extremo al pin de 10V del conector J3.
- Se puso un condensador de 1uF en paralelo a R9 para que la fuente conmutada tenga un arranque lento.

Todos estos errores están corregidos en los esquemáticos presentados en esta memoria.