Diseño de una plataforma para el desarrollo de sistemas embebidos

William Alexánder Salamanca Becerra

Oscar Humberto Carrillo Hernández

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, 2008

Diseño de una plataforma para el desarrollo de sistemas embebidos

William Alexánder Salamanca Becerra

Oscar Humberto Carrillo Hernández

Trabajo de grado para optar por el título de Ingeniero Electrónico.

Director:
Mse. Jorge Hernando Ramón Suárez.

Codirector:
Msc. Elkim Felipe Roa Fuentes.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, Enero de 2008

a mis padres por sus esfuerzos
durante todo este tiempo,
a Jenny Sofía por su paciencia,
por ser mi motivación y mi gran apoyo.

Agradecimientos

Un agradeciemiento especial a nuestros padres por su apoyo incondicional durante la carrera

Agradecemos a todas las personas que han contribuido con nuestro proceso de formación como ingenieros, a todas las personas que nos han apoyado; Al profesor Jorge Ramón, por toda la paciencia y el apoyo que nos brindó para poder llevar a cabo este proyecto; Al profesor Carlos Iván Camargo por sus experiencia y sus aportes; y al profesor Elkim Roa por su motivación y orientación, la cual inspiró la realización de este proyecto.

RESUMEN

TITULO: DISEÑO DE UNA PLATAFORMA PARA EL DESARROLLO DE SISTEMAS EMBEBIDOS[1].

AUTORES: Salamanca Becerrra William Alexánder y Carrillo Hernández Oscar Humberto[2].

PALABRAS CLAVES: Sistema Embebido, ARM, SoC, FPGA, PCB

DESCRIPCION:

A lo largo del desarrollo del presente proyecto se logró realizar el diseño y una primera implementación del sistema que se ha denominado Phoenix, el cual es una herramienta para el desarrollo de aplicaciones basadas en sistemas embebidos. Phoenix está basado en un arreglo de compuertas lógicas programables (FPGA por sus siglas en inglés) de la familia Spartan-3E de Xilinx y un microcontrolador integrado (SoC) del fabricante Cirrus Logic con un núcleo ARM9TDMI.

Phoenix permite al diseñador realizar aplicaciones empleando metodologías para el desarrollo de software, programando el SoC; y metodologías para la síntesis de *hardware*, configurando el FPGA. Adicionalmente permite explotar las ventajas de estas dos alternativas empleando los dispositivos con metodologías de co-diseño. La implementación de Phoenix involucró el uso de tecnología de tarjetas de circuito impreso(PCB) de múltiples capas, lo cual representa a nivel local una incursión en el proceso de diseño y fabricación de estos. Además se ha evaluado el uso de herramientas profesionales para el diseño de PCBs, evidenciando los beneficios que se obtienen al usarlas, mejorando la calidad del producto y disminuyendo el tiempo de diseño.

La experiencia obtenida de la implementación del sistema, ha permitido detectar aspectos a mejorar en la herramienta y abre la posibilidad para el desarrollo de proyectos basados en Phoenix y el diseño de futuras versiones de esta.

---

[1] Proyecto de grado

[2] Facultad de Ingenierías fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Director: Jorge Hernando Ramón Suárez. Codirector: Elkim Felipe Roa Fuentes

ABSTRACT

TITLE: DESIGN OF AN EMBEDDED SYSTEMS DEVELOPMENT PLATFORM[3].

AUTORS: Salamanca Becerrra William Alexánder y Carrillo Hernández Oscar Humberto[4].

KEYWORDS: Embedded System, ARM, SoC, FPGA, PCB

DESCRIPTION:

The execution of this project has reached the design and the first implementation of the system called Phoenix. This system is a development tool for embedded system based applications. Phoenix is powered by an Xilinx Spartan-3E Field Programmable Gate Array (FPGA) and an ARM9TDMI based System on a Chip (SoC) manufactured by Cirrus Logic.

Phoenix allows the user to design applications employing software development methodologies, by programming the SoC; and hardware synthesis methodologies, configuring the FPGA; As well as it allows to exploit this two alternatives, using a co-design methodology. The Phoenix implementation involves the use of multilayer Printed Circuit Board (PCB) technology. Locally, this means an introduction to the design and fabrication of this kind of PCBs. This project has also evaluated the use of professional PCB design tools, showing the benefit of using them, improving the product quality and reducing the design time.

The experience obtained with the system implementation, allowed to detect some aspects to improve in the tool designed, also give the chance to begin new Phoenix based projects and the implementation of new versions of Phoenix with new features.

[3] Degree work

[4] Faculty of Phisical-Mechanical Engineering. Electrical and Electronic Engineering School. Director: Jorge Hernando Ramón Suárez. Codirector: Elkim Felipe Roa Fuentes

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y conceptos básicos

1.1. Introducción

Los sistemas embebidos son parte de nuestra vida diaria y se han convertido en elementos fundamentales para la realización de nuestras labores. Por medio de ellos satisfacemos gran parte de nuestras necesidades en diferentes campos como entretenimiento, comunicaciones, administración y automatización de las industrias, etc. Cada día, somos testigos de como los avances tecnológicos permiten aumentar su complejidad y así obtener mayores funcionalidades.

Se define un sistema embebido como un sistema de cómputo desarrollado especialmente para ejecutar una tarea específica, con este fin, es diseñado en forma coordinada en *hardware* y *software* satisfaciendo algunas limitaciones propias de la aplicación.

Algunos sistemas embebidos se presentan directamente como un sistema de cómputo ante los usuarios como es el caso de PDAs y calculadoras científicas. Pero otros pueden hacer parte de un sistema o una infraestructura más grande que hace que los usuarios no los perciban como tales. Los sistemas de cómputo instalados en algunos automóviles o celulares, el sistema que controla los tiempos de un horno microondas, las impresoras son elementos que pueden fácilmente pasar inadvertidos como sistemas de cómputo por las personas que los utilizan.

Cada sistema embebido es muy particular y los diseñadores de estos se enfrentan a diferentes tipos de problemas y limitaciones. Según la aplicación, el sistema posee unas restricciones particulares que deben ser consideradas en el diseño, además de las características ideales que

se buscan en todo sistema de cómputo, como el consumo de potencia, tiempos de latencia bajos, capacidad de procesamiento.

Dada la diversidad de aplicaciones que este tipo de wsistemas desarrollan, es difícil establecer una metodología de diseño generalizada. Una metodología de diseño debe especificar las etapas que permitan el mejor aprovechamiento de los recursos disponibles, que se cumplan de mejor manera los objetivos planteados y que se desarrolle en el menor tiempo posible. Durante el primer capítulo se presenta una metodología que ha sido bastante aceptada y está orientada al concepto del codiseño, en donde se analizan las funciones específicas que debe cumplir el sistema y se distribuyen en *hardware* y/o *software*. El codiseño se puede definir como una metodología en donde diseñadores de *hardware* y *software* realimentan su trabajo para que los dos equipos obtengan beneficios que permitan cubrir sus falencias y debilidades.

Con el pasar de los días las aplicaciones exigen cada vez más recursos y prestaciones. La tecnología evoluciona y hace posible la implementación de sistemas con mayor capacidad de procesamiento, mejores métodos de comunicación y mejor tiempo de respuesta. A la cabeza de este avance tecnológico vemos sistemas de uso masivo como celulares, ordenadores de escritorio y consolas de videojuegos. Estos sistemas evolucionan constantemente e implementan sistemas con procesadores mas rápidos, con más memoria o arquitecturas innovadoras. También vemos como aparecen dispositivos especializados como DSPs y ASICs, procesadores y microcontroladores de 32 y 64 bits y FPGAs que permiten implementar prototipos de sistemas para aplicaciones específicas con un buen desempeño y con un reducido costo.

Paralelamente a la evolución del *hardware*, también se desarrolla *software* que busca obtener el mejor desempeño de estos dispositivos. Vemos entonces la aparición nuevos sistemas operativos con características específicas. Aunque gran parte de estos, se orientan a aplicaciones de Tiempo Real, algunos son muy particulares como por ejemplo los sistemas operativos para sistemas parcialmente reconfigurables.

Mediante este proyecto de grado ha desarrollado una tarjeta llamada Phoenix, la cual se presenta como una alternativa para las personas que requieran incursionar en el desarrollo de sistemas embebidos y para las personas que requieren una plataforma independiente para desarrollar sus proyectos. Phoenix cuenta con gran variedad de recursos, y permite desarrollo de sistemas que requieran el diseño de componentes en *hardware* y/o *software*.

1.2. Historia de los sistemas embebidos

Dada la definición de un sistema embebido como un sistema de cómputo, vemos que estos no pudieron aparecer antes de 1971 cuando intel sacó su primer microprocesador, el 4004, el cual fue diseñado originalmente para la compañía japonesa Busicom para ser usado en sus calculadoras. Éste era un procesador de 4 bits con una velocidad máxima de trabajo de 740 kHz, una arquitectura Harvard y un set de 46 instrucciones, encapsulado en CERDIP de 16 pines. Posterior a él se han desarrollado gran cantidad de microprocesadores empezando por el 8008 también de intel, el cual es de 8 bits, y llegando a los procesadores actuales multicore de 32 y 64 bits. Algunos de estos microcontroladores se han especializado y han sido diseñados según especificaciones y necesidades de algunas áreas en particular.

Una de las áreas en las que se evidencian esos avances ha sido el mercado de las consolas de videojuegos, los cuales hacen su aparición incluso antes de la aparición del primer procesador integrado. Los primeros videojuegos comerciales aparecen con Atari desde el 2600 en adelante. Estos sistemas usaban procesadores de 8 bits y tuvieron su auge en los años 70 y principios de los 80. Posteriormente en los años 80 aparecen Nintendo y el Spectrum los cuales aun estaban basados en procesadores de 8 bits. A medida que la tecnología avanza, movida en gran parte por este campo, se iban acogiendo nuevos procesadores y arquitecturas. Es así como aparece Super Nintendo, Neo Geo y muchas otras consolas que aprovechan los recursos técnicos del momento como los procesadores de 16 bits.

Actualmente vemos consolas como la PS3 o la XBOX 360 que poseen arquitecturas *multicore* únicas con los mejores procesadores del mercado. Además incorporan nuevos sensores y nuevas formas para interactuar con el usuario haciendo mas realista la experiencia de jugar.

1.3. Sistemas de tiempo real

Dentro de las restricciones que hay en el diseño de sistemas embebidos, es muy común ver requerimientos en tiempos de respuesta controlados. Es por esto que aparecen los sistemas de tiempo real, los cuales se definen como aquellos sistemas que son capaces de responder a estímulos del medio, en tiempos establecidos en el diseño según la aplicación para la cual están hechos. Para responder adecuadamente, estos sistemas deben tener completamente caracterizados sus tiempos de latencia internos, por ejemplo, el tiempo de latencia de los sensores, de los canales de comunicación, la velocidad de transferencia de datos entre los diferentes niveles de memoria, el tiempo de procesamiento y el tiempo que tardan sus actuadores y periféricos de salida en responder en el medio.

Es común ver que este tipo de sistemas se clasifican según el nivel de tolerancia en su respuesta: los sistemas "Hard Real-Time" son muy estrictos y es crítico que el sistema responda en los tiempos establecidos, generalmente porque una falla en este tipo de sistemas podría incluso llegar a representar pérdidas de vidas humanas. El otro tipo de sistemas es el "Soft Real-Time" el cual es mas flexible en sus tiempos de respuesta, admitiendo mayor tolerancia debido a que la aplicación no falla completamente debido a ese retraso.

1.4. Metodología de diseño de sistemas embebidos

Como se había mencionado, la metodología para el desarrollo de sistemas embebidos no está generalizada y está estrechamente ligada a las limitaciones del problema que el sistema intenta solucionar y a los dispositivos que se emplearán en el desarrollo de la solución. Según el tipo de dispositivo, a continuación se presentan algunas metodologías de diseño comúnmente

empleadas.

1.4.1. Metodología de diseño con microprocesadores y microcontroladores

La metodología presentada en esta sección, se refiere a sistemas que se implementarán sobre dispositivos que contienen una unidad de procesamiento con unos periféricos y una capacidad de memoria establecida como los microcontroladores y DSP. Partiendo de un problema definido y una variedad de dispositivos disponibles en el mercado, el proceso a seguir para el desarrollo de la aplicación se muestra en la figura 1.1

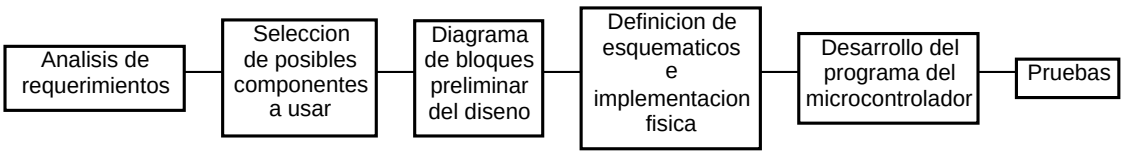


Figura 1.1: Flujo de diseño con microcontroladores

El proceso de desarrollo del programa del microcontrolador, es un subproceso que se resume en el diagrama de flujo de la figura 1.2, donde el diseñador inicia realizando una planificación del código y definiendo algunos parámetros y herramientas como el lenguaje que va a usar para el desarrollo del programa o en algunas rutinas en particular. Posteriormente, el código es compilado y depurado minuciosamente, para finalmente programar el dispositivo dejándolo listo para las pruebas finales de laboratorio y en campo.

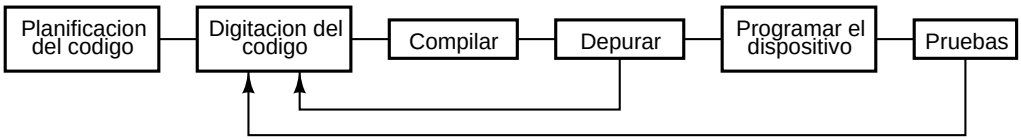


Figura 1.2: Flujo de diseño con microcontroladores

1.4.2. Metodología de diseño con PLDs

Las diferencias entre un PLD y un microcontrolador o DSP, exigen un cambio en la forma como el diseñador concibe el dispositivo, la forma de programación y la metodología de diseño a emplear. El diseño se debe basar en un diagrama de bloques o descripción RTL, el cual es llevado al ordenador por medio de un lenguaje de descripción de hardware (HDL) o por medio de un esquemático, a partir del cual se sigue el flujo de diseño mostrado en la figura 1.3

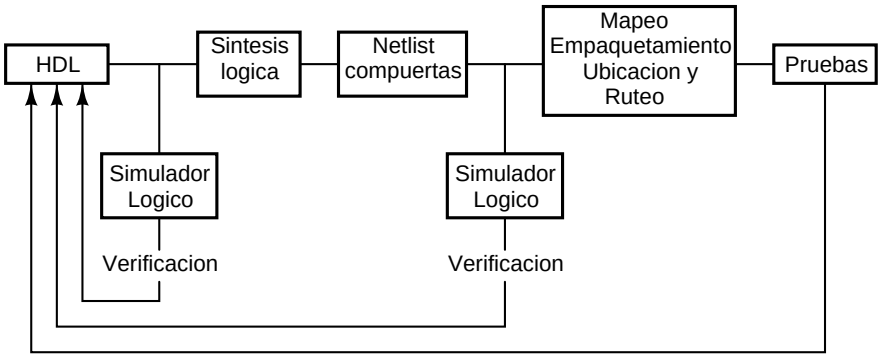


Figura 1.3: Flujo de diseño con PLDs

Los vendedores de los PLDs brindan el software necesario para llevar a cabo este proceso. Este software es generalmente bastante completo y brinda muchas más herramientas y posibilidades que las que se muestran en el diagrama, el cual muestra solo las etapas fundamentales del proceso. Es importante notar que existen verificaciones y simulaciones a diferentes niveles. Se puede iniciar a verificar el diseño en el código o el esquemático, pasada la síntesis lógica, a nivel de compuertas e incluso, si las herramientas de diseño lo permiten, a nivel físico.

1.4.3. Codiseño

Las metodologías basadas en el Codiseño *hardware/software* aparecen en el diseño de sistemas embebidos como una necesidad ante la gran variedad de dispositivos que presenta actualmente la industria para el desarrollo de estos, así como la gran cantidad de restricciones y metas a satisfacer. Las metodologías presentadas anteriormente, se basan en un dispositivo

en particular, el cual tiene una arquitectura fija como en el caso de los microcontroladores o tiene los recursos necesarios para implementar una arquitectura abierta como en un PLD. Las metodologías basadas en el codiseño, tienen en cuenta que las funciones del sistema de procesamiento, comunicación, interacción con el usuario, etc, se pueden desarrollar por medio de *hardware* o *software* y que en cada caso las dos implementaciones tienen diferentes ventajas.

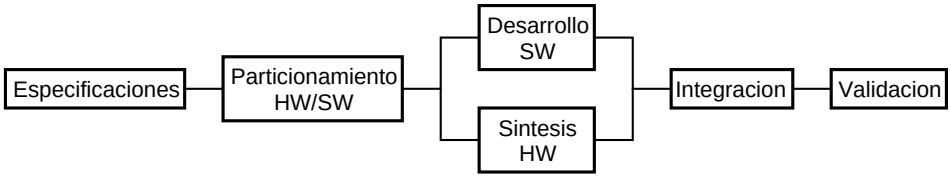


Figura 1.4: Flujo de diseño orientado al codiseño

La figura 1.4 es un esquema simplificado que muestra la forma como las metodologías basadas en codiseño afrontan el diseño de un sistema buscando obtener la mejor solución según los requerimientos y aprovechando las ventajas que ofrece la implementación de funciones en *hardware* o *software*. Para ello define procedimientos para particionar el problema, para comunicar los dos equipos de trabajo y la forma como el trabajo de ambos se debe integrar.

Especificaciones

El objetivo durante esta etapa es desarrollar un modelo general del sistema. En este modelo se deben contemplar características del sistema vistas desde un enfoque funcional a un nivel de abstracción bastante alto. El modelo, ilustrado generalmente como un diagrama de bloques, representará las etapas funcionales necesarias para cumplir el objetivo final del sistema. Cada bloque debe estar especificado por la función que desempeña, si es posible un estimativo de la cantidad de recursos que requiere, mecanismos de comunicación con otros módulos y las restricciones que le impone el medio y la aplicación como el consumo de potencia, el tamaño, velocidades de trasmisión de datos y tiempos de latencia.

Particionamiento HW/SW

Una vez conocidas las etapas o bloques funcionales del sistema, se define cuales de esas tareas deben ser implementadas por medio de una secuencia de instrucciones, es decir por *software*; y cuales requieren un elemento especializado basado en funciones lógicas, es decir que realice las tareas por *hardware*.

Para hacer esta partición, se deben tener en cuenta las restricciones del sistema ya que generalmente se generan compromisos entre los aspectos que mejoran cada una de las componentes HW y SW. Por ejemplo la implementación de funciones en *hardware* puede mejorar la velocidad de procesamiento o de respuesta, ya que brinda al sistema un alto grado de concurrencia, pero puede incrementar el consumo de potencia y la complejidad del diseño haciéndolo mas costoso. Por otra parte la implementación de tareas por *software*, puede simplificar el tamaño del sistema, a la vez que lo hace de fácil actualización, pero puede ocupar tiempo valioso de procesamiento ofrecer muy baja concurrencia.

El correcto particionamiento del sistema es muy importante ya que puede definir varias de las características del producto final como la capacidad para ampliar su memoria o para ser actualizado. Por ejemplo si se esta diseñando un reproductor MP3 y se decide codificar y decodificar el audio por *hardware*, será bastante difícil actualizar el sistema para que pueda recibir otro tipo de formato de música.

Desarrollo de código y Síntesis de *hardware*

El desarrollo de los dos componentes de sistema *hardware* y *software*, es un proceso que se debe realizar en forma simultánea para buscar mayor integración entre estos dos componentes, ya que la armonía con la que trabajen determinará las mejores características del sistema. A medida que se va desarrollando cada uno de los componentes, los entes encargados de cada una de las partes deben intercambiar información, ya que es posible trasladar pequeñas funciones de *software* a *hardware* y viceversa. Esta interacción e intercambio de información se conoce

comúnmente con el nombre de co-diseño *hardware* y *software* y es la actual tendencia de los modelos de diseño de sistemas embebidos.

Integración

Una vez se tengan los diferentes bloques funcionales del sistema implementados, se debe proceder a desarrollar los mecanismos que permitan a estos intercambiar información y generar eventos. Según el tipo y la complejidad del sistema, este proceso puede ser muy dispendioso porque aunque el modelo inicial defina algunos de los protocolos a usar, la coordinación e implementación de estos canales no siempre es sencilla.

Verificación

Aunque en cada una de las etapas anteriores se desarrollan mecanismos para verificar los objetivos planteados, esta etapa pretende probar el sistema como un conjunto. Para esto es posible que los diseñadores generen sus propias escenarios de prueba o apliquen pruebas *standard* para poder comparar el producto desarrollado con otros de similares características. A este punto del desarrollo, es posible descubrir errores o fallas de diseño que nos lleven a reconsiderar decisiones en alguna de las etapas anteriores.

1.5. Sistema operativo

Dado que los sistemas embebidos han incrementado la cantidad y variedad de los recursos disponibles, y las aplicaciones han aumentado su complejidad, se ha hecho necesaria la implementación de sistemas operativos embebidos. El sistema operativo tiene como principal función administrar los recursos de un sistema y permitir la interacción con el usuario. Muchos de estos sistemas operativos están basados en los sistemas operativos más comunes empelados en ordenadores como Windows y Linux en varias de sus distribuciones. Es así como encontramos a Windows CE, uCLinux, Embedded Debian, etc. Gracias a esta relación con los sistemas operativos de ordenador, es posible encontrar drivers, aplicaciones y librerías que han sido

compiladas para una arquitectura embebida, o son fácilmente adaptables por medio de compiladores cruzados, si se cuenta con su código fuente. Así mismo existen otros sistemas operativos independientes, desarrollados especialmente para aplicaciones embebidas, algunos de ellos libres, que ya ofrecen su código fuente completo y los hace deseables debido a las posibilidades de adaptación que esto genera.

1.5.1. Sistemas operativos de tiempo real (RTOS)

Dado que la mayoría de restricciones de los sistemas operativos están relacionados con tiempos de respuesta, la mayoría de sistemas operativos se están orientando al concepto de tiempo real, el cual consiste en que estos sistemas operativos garantizan que las tareas que se estén ejecutando se deben desarrollar en un tiempo establecido por el diseñador. Para el manejo de esos tiempos, el diseñador debe establecer prioridades entre las tareas que se ejecutan. La diferencia principal de un sistema operativo de tiempo real con uno normal, está básicamente en su algoritmo de planificación de tareas, el cual hace respetar la prioridad que tienen las tareas y los procesos que se están ejecutando.

1.6. Sistema de archivos

Un sistema operativo se escribe en el medio que lo contiene a través de un sistema de archivos o *filesystem*. Un sistema de archivos es el formato que se da al medio de almacenamiento y que permite escribir y leer información por nombres, y no por direcciones de memoria, como es usual en un medio virgen. Normalmente en un sistema de archivos se crea un registro en el que se organiza la información por directorios y contenidos. Ahí se coloca información importante del archivo como tipo, tamaño, número de partes en que se ha dividido y ubicación de cada parte en el medio de almacenamiento.

Existen características que diferencian los sistemas de archivos, como su estructura y algunas mejoras que brindan mayor velocidad de acceso o garantizan la integridad de la información.

Debido a la constante pérdida de datos como consecuencia de anomalías en la red eléctrica, problemas en los sistemas operativos, y errores humanos, los sistemas de archivos avanzados, cuentan con un sistema *journalist*, que consiste en un registro de sucesos, que se crea al momento de abrir un archivo. De este modo se garantiza que si existe un problema en la escritura, cuando se modifica el archivo, el sistema de archivos sufre un menor daño, evitando grandes pérdidas de información.

1.7. Herramientas de diseño

Las herramientas de las cuales dispone un diseñador o un grupo de diseñadores pueden ser muy variadas dependiendo de los dispositivos a emplear y del presupuesto para el diseño. Pero generalmente un diseñador requiere de un ordenador que actúe como *host* y una tarjeta o sistema de desarrollo que aloje dispositivos y recursos del sistema. Entre el la tarjeta y el *host* pueden existir diferentes canales de comunicación, los cuales tienen como objetivo permitir funciones de depuración, intercambio de datos, programación y configuración del sistema. Por ejemplo, el desarrollo de una aplicación puede requerir comunicación por puerto serie para descargar las aplicaciones que se van a ejecutar en el sistema, JTAG para la depuración o monitoreo de la aplicación y Ethernet para acceso a una red LAN o a un sistema de archivos remoto.

A nivel de software, siempre es posible encontrar herramientas libres para los dispositivos comercialmente disponibles. Si se trata de un procesador, encontramos *toolchain* que contienen compiladores cruzados y herramientas necesarias para compilar y depurar aplicaciones para arquitecturas diferentes a la del *host*. Si se trata de un FPGA, encontramos gran cantidad de herramientas ofrecidas por los fabricantes, las cuales facilitan el desarrollo y la implementación de diseños sobre estos dispositivos.

Capítulo 2

Selección de componentes y diseño de la tarjeta

A lo largo del presente capítulo, se mostrarán los recursos disponibles en la tarjeta, el proceso de selección de los componentes, su forma de conexión y el proceso de diseño e implementación física de la tarjeta.

2.1. Requerimientos de la tarjeta

Para el desarrollo del proyecto, se requirió inicialmente determinar las necesidades que pueda tener un diseñador de sistemas embebidos, para así ofrecerle una solución que le permita desarrollar aplicaciones de manera rápida y con el menor número de contratiempos por implementaciones en *hardware* y *software*. El diseño se desarrolló teniendo en mente una arquitectura abierta que explote al máximo todas las capacidades de los dispositivos empleados y que permita añadir tarjetas de expansión o subsistemas que permitan ampliar las funciones de la tarjeta según lo requiera la aplicación.

Phoenix, la tarjeta desarrollada, tiene implementados recursos y periféricos que son de uso cotidiano en el diseño de sistemas embebidos, además de un gran número de alternativas de expansión y con base en esos recursos, se deja abierta la posibilidad de instalar un sistema operativo.

Durante el estudio de los sistemas embebidos, hemos observado que las dos formas más

comunes de implementar un sistema son por medio de un FPGA, o por medio de un *SoC*[1] o un procesador y componentes discretos[2]. La implementación de todo el sistema en un FPGA, brinda bastante flexibilidad y la posibilidad de generar módulos personalizados dentro del sistema, aunque limitado por los recursos del dispositivo seleccionado, por otro lado la segunda alternativa ofrece recursos limitados, dado que solo se dispone de los periféricos y los recursos que posea el microcontrolador. Aunque para gran parte de los sistemas, estos recursos son suficientes y ofrecen el desempeño necesario para su correcto funcionamiento.

Otra alternativa de implementación es la que se puede obtener de la combinación de las dos anteriores, donde se pueden emplear un SoC y un PLD. El primero aporta su capacidad de procesamiento y recursos implementados, mientras que el PLD puede desarrollar funciones de coprocesamiento, aceleración de procedimientos por *hardware* o implementación de periféricos.

Phoenix ha integrado estas tres formas de implementación, por ello ofrece un procesador de 32 bits, periféricos de interacción con el usuario y de comunicación con otros sistemas; un FPGA y la posibilidad de emplear estos dispositivos independientemente o en forma combinada.

2.2. Organización de la tarjeta

Como se habló en el capítulo 1, el desarrollo de sistemas embebidos está compuesto por elementos y funciones que pueden ser implementados en *hardware* o en *software*. Por eso Phoenix se ha dividido en dos secciones que le permitirán al diseñador desarrollar el sistema empleando esas dos alternativas de diseño. La primera sección está centrada en un procesador de 32 bits, con memoria y algunos periféricos implementados. La segunda sección está destinada al desarrollo de periféricos o al coprocesamiento y está basada en un FPGA. Entre las dos secciones existen varios canales de comunicación y algunos recursos adicionales, para implementar dife-

[1] System on Chip

[2] Existen otras formas de implementación como los ASIC, pero durante este trabajo, se ha hecho énfasis sobre las alternativas más económicas y con orientación al prototipado.

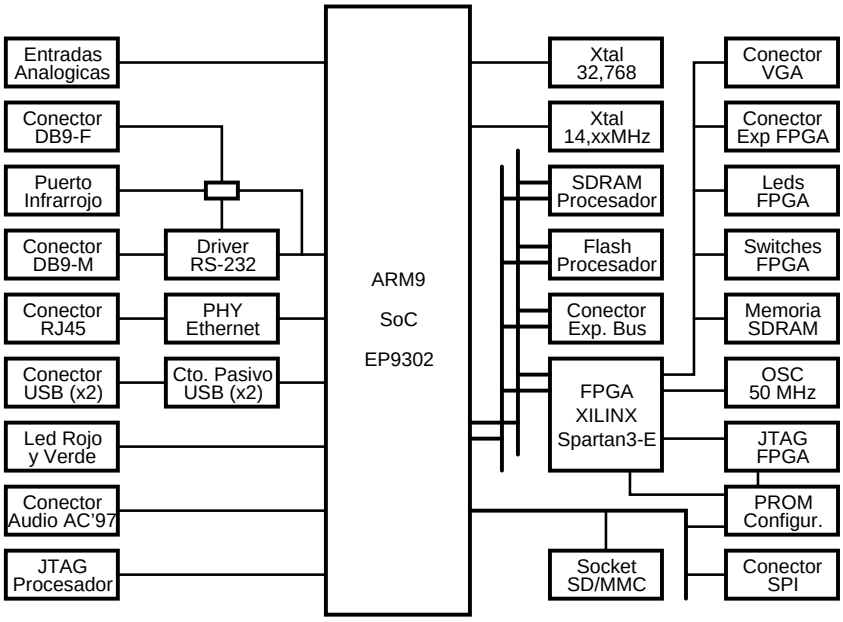


Figura 2.1: Diagrama de bloques de Phoenix

rentes tipos de periféricos. La figura 2.1 muestra el diagrama general de bloques de la tarjeta. En ella se aprecia el procesador, el FPGA, los periféricos y los buses por medio de los cuales se comunican estos dispositivos.

2.3. Sección del procesador

2.3.1. El procesador

Muchas de las aplicaciones que pueden llevar a cabo los sistemas embebidos, requieren de procesamiento de señales, alta transferencia y procesamiento de datos, o requieren atender múltiples tareas simultáneamente con prioridades y restricciones de tiempo, para lo cual se implementa un sistema operativo. Esto supone que Phoenix debe estar dotado de una unidad de cómputo que brinde suficiente capacidad para satisfacer tareas de este tipo. Además debe

estar soportado por los sistemas operativos normalmente usados en este tipo de desarrollos y debe contar con las herramientas de diseño necesarias para llevar a cabo las aplicaciones.

Por ello se estudiaron algunas de las arquitecturas de los procesadores de 32 bits que se encuentran disponibles en el mercado como AVR, MIPS, SPARC, PowerPC, e incluso algunos *soft-processors* como el microblaze, así como los fabricantes que emplean estas arquitecturas en sus chips. A partir de ello, se observa que a diferencia de los ordenadores de escritorio, en arquitecturas embebidas los diseñadores prefieren las arquitecturas RISC, y esto se basa en que los sistemas embebidos poseen muchas veces restricciones de consumo de potencia y espacio.

Como procesador principal de nuestro sistema hemos escogido un procesador ARM. La familia de procesadores ARM, se presenta como una de las arquitecturas más empleadas en sistemas embebidos junto a los procesadores AVR. ARM es una familia de procesadores RISC que ha venido evolucionando desde 1983 cuando Acorn Computers Ltd. inició el desarrollo de las primeras versiones, dando como resultado el ARM1 y ARM2, los cuales fueron presentados en el mercado en 1985 y 1986. Siempre estos procesadores se han caracterizado por su simplicidad; desde sus primeras versiones estos procesadores requieren muy pocos transistores en su implementación. Posteriormente han aparecido nuevas versiones como los ARM3, ARM6, ARM7, ARM9, XScale, etc.

Los procesadores ARM se presentan en el mercado como núcleos, lo cual significa que los fabricantes de chips como Freescale, Cirrus, Atmel, Texas Instruments, etc, desarrollan sus sistemas SoC tomando como base un núcleo ARM y añadiendo algunos periféricos. Para este proyecto se empleó el EP9302 fabricado por la empresa Cirrus y que incorpora un ARM920T. Este núcleo incorpora una MMU[3], lo cual permite instalar sistemas operativos como Windows y Linux. Además del núcleo, el EP9302 tiene implementados una gran variedad de periférícos de comunicación y un coprocesador de punto flotante. La arquitectura del SoC se detallará en

[3]Memory Management Unit

el capítulo 3.

Implementación

Los componentes básicos para el funcionamiento del procesador son básicamente los circuitos de reloj y el puerto de programación. Durante el proceso de diseño de los diagramas esquemáticos de estos componentes, se tuvieron en cuenta las recomendaciones y los diseños mostrados en la página del fabricante.

Osciladores El EP9302, requiere dos osciladores para su funcionamiento: uno de Tiempo Real, para el arranque y operaciones que requieran medir tiempos exactos, que tiene como base un cristal externo de 32.768 kHz. Y un segundo reloj que posee otro cristal externo de 14,7456 MHz, el cual entrega su señal a dos PLLs internos que generan las frecuencias necesarias para el funcionamiento del procesador, las memorias y los periféricos. Ambos relojes son fundamentales para el procesador y el sistema no iniciará si uno de los dos llega a estar ausente. Dado que el reloj de tiempo real es muy suceptible a ruido interno del chip [20] se ha implementado este circuito como un oscilador externo con ayuda de dos compuertas discretas con el objetivo de entregar al procesador una señal de onda cuadrada de muy buena calidad. El esquemático de la figura 2.2 muestra la configuración empleada en los dos osciladores.

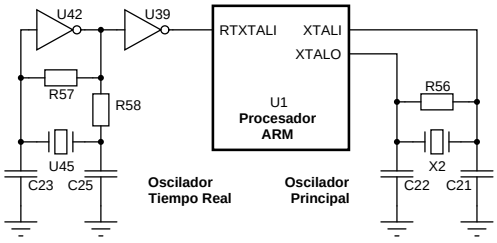


Figura 2.2: Diagrama esquemáticos de los osciladores del procesador

Puerto JTAG Entre sus pines, el procesador *EP9302* tiene 5 pines dedicados para el acceso directo al puerto JTAG del núcleo ARM9TDMI. Aunque este puerto no será muy usado, se

decidió implementarlo dado que permite realizar depuración y pruebas sobre el núcleo. Las señales del puerto son TDK, TDI, TDO, TCK y TRST. En el PCB, cada una de estas señales tiene un resistor pull-up y son llevadas directamente al conector U49 que recibe el cable que conecta el puerto al *host*.

2.3.2. Bus principal

El procesador *EP9302* , posee los controladores para manejar memorias externas dinámicas síncronas y estáticas en el bus principal. Estos controladores comparten los pines del bus de datos y direcciones, pero cada uno maneja su propio bus de control. Las señales del bus principal del sistema se muestran en la tabla 2.1. A este bus también es posible conectar periféricos u otros componentes externos que cumplan el protocolo y los tiempos del bus. La figura 2.3 muestra la organización del bus empleada en Phoenix.

Señal	Descripción
WRn	Señal Escritura
RDn	Señal Lectura
WAITn	Señal de espera
AD[25:0]	Bus de direcciones
DA[15:0]	Bus de datos
CS[7:6,3:0]	Selectores de dispositivos
DQMn[1:0]	Mascara de datos

Señal	Descripción
SDCLK	Reloj mem. síncronas
SDCLKEN	Habillitador del reloj
SDCS[3:0]	Selector de dispositivos síncronos
RASn	Latch de la fila de la dirección
CASn	Latch de la columna de la dirección
SDWEn	Habilitador de escritura

Tabla 2.1: Señales del bus principal

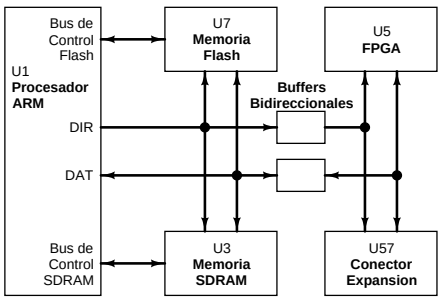


Figura 2.3: Organización del bus principal del sistema

Las líneas de selección se activan según el mapa de memoria del SoC y han sido asignadas a cada componente del bus. Así, la memoria Flash se puede acceder por medio de CS_7 o CS_6. $CS[3:0]$ se destinó a los periféricos del FPGA y el conector de expansión para periféricos adicionales; mientras que $SDCS\_3$ fue asignado a la memoria SDRAM

Memorias

En Phoenix se han seleccionado dos tipos de memoria que se implementaron en el bus principal: Una memoria SDRAM *MT48LC4M16A2* de 8MB, por su alta densidad y bajo costo comparada con las memorias RAM estáticas, y una memoria Flash *TC58FVM6B2A* de 8MB lo cual representa capacidad suficiente para almacenar código de arranque y el kernel de un sistema operativo. En el diseño del PCB, estas memorias tuvieron prioridad para ser ruteadas, dado que ellas manejan la mayor tasa de transferencia de datos y son fundamentales para el funcionamiento del sistema. Su conexión se detalla en los diagramas esquemáticos[37].

FPGA y puerto de expansión

Estos elementos se han implementado con el objetivo de que el diseñador tenga la posibilidad de añadir sus propios periféricos al bus principal. Para estos dos elementos se han dispuesto buffers bidireccionales que tienen como función mejorar la calidad de las señales del bus y proteger de posibles errores en la implementación de los periféricos a las memorias y al procesador.

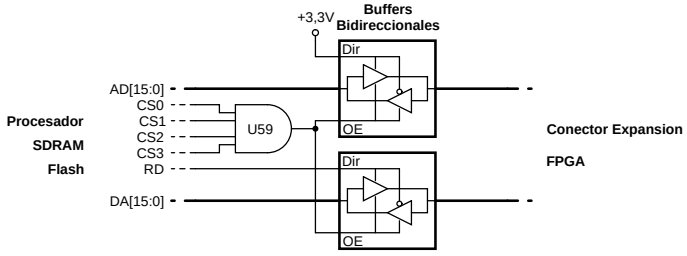
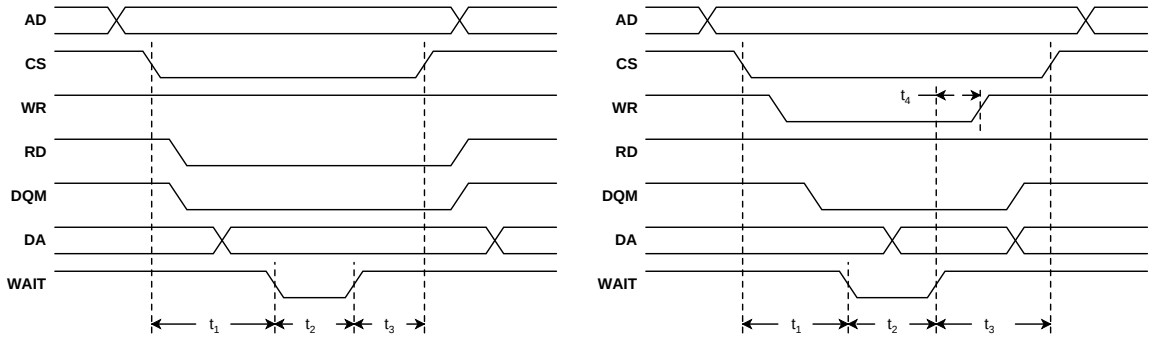


Figura 2.4: Conexión de los buffers del bus principal

La conexión de estos buffers se hizo según lo muestra la figura 2.4, para lo cual se tuvo en cuenta el diagrama de tiempos del procesador que se muestran en la figura 2.5. De esa forma se observó que se puede emplear la señal RD para seleccionar la dirección del bus de datos y que según la asignación realizada de las señales CS, cualquiera de las señales $CS[0:3]$ debe habilitar los buffers.



(a) Ciclo de lectura

(b) Ciclo de escritura

Figura 2.5: Diagramas de tiempo del bus principal

2.3.3. Periféricos

El SoC *EP9302* reúne una variedad considerable de periféricos. La implementación de cada periférico es muy particular y puede o no requerir de componentes externos. A continuación se detallan aspectos técnicos sobre la implementación de cada uno de los periféricos del procesador.

Puertos Serie El SoC *EP9302* posee dos unidades $UART$[4]. $UART_1$ maneja todas las señales del puerto serie estándar. Esta unidad fue dotada de una interfaz RS232 que tiene como puerto el conector U13 de la tarjeta. La segunda unidad $UART_2$ tiene adicionalmente la capacidad para manejar un transmisor/receptor infrarrojo compatible con el estándar IrDA[5] con velocidades baja(115.2 kbps), media(0.576/1.152 Mbps) y alta(4 Mbps). Por ello el diseño de esta

[4]Universal Asynchronous Receiver-Transmitter

[5]Infrared Data Association

unidad se ha desarrollado de modo que pueda ser empleada como puerto serie RS232 o como transmisor/receptor infrarrojo. La figura 2.6 muestra el diagrama de conexiones de los puertos.

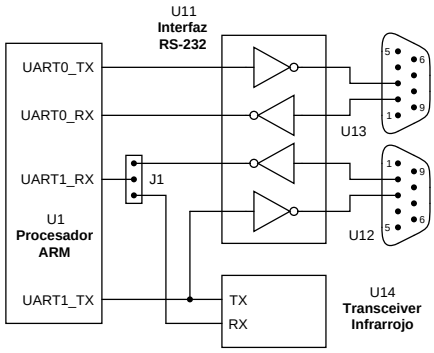


Figura 2.6: Conexión de los puertos seriales

El diseño permite que por medio del jumper J1 se seleccione la interfaz física a usar. Si el jumper se ubica en el lado derecho, la interfaz seleccionada es la RS232, por lo cual se habilita la señal de recepción para que sea recibida desde el conector U12 pasando por el chip *MAX3232* . Si por el contrario el jumper se encuentra en el lado izquierdo, la señal de recepción llegará directamente desde el transmisor/receptor infrarrojo *TFDU4100* . La señal de transmisión es llevada directamente a ambas interfaces.

Puertos USB El *EP9302* tiene dos puertos USB *host* de alta velocidad. Para su puesta en marcha, en el PCB sólo fue necesario añadir el conector e implementar la terminación de línea, para lo cual se empleó el chip *USBDF01W5* .

Puerto Ethernet Ethernet es una tecnología que permite la comunicación por medio de paquetes entre varios sistemas de cómputo. En él están definidas las normas que debe cumplir la implementación de la capa física (PHY) y de transferencia de datos. El *EP9302* posee una MAC[6] implementada, y para su funcionamiento requiere la implementación externa de la capa física. El protocolo que rige la interconexión entre la MAC y la capa física se llama MII (Media

[6]Media Access Control

Indepent Interface) y como su nombre lo dice, admite PHYs que manejan diferentes tipos de medios. Para la tarjeta se ha seleccionado la PHY *LXT972A* del fabricante Intel que maneja par trenzado a una velocidad de 100 Mbps.

La implementación de la PHY requiere la utilización de algunos elementos pasivos, los cuales se seleccionaron de una lista de elementos recomendados por el fabricante.

Entradas Analógicas El *EP9302* posee un conversor analógico digital de 12 bits y de 5 canales de entrada multiplexados orientado al manejo de pantallas táctiles y señales de baja frecuencia como tensión en la batería y temperatura del procesador. La frecuencia máxima de muestreo del conversor es de 3750 muestras por segundo.

En la tarjeta se han llevado los 5 canales a un conector para ser usados por el usuario. Este debe tener en cuenta que el rango de tensión de entrada es de 0 a 3.3V y que estas señales no están acondicionadas por componentes analógicos adicionales.

CODEC de audio El *EP9302* ofrece la posibilidad de conectar a él un CODEC de audio tipo AC'97, ya que el controlador está implementado al interior del chip. Las líneas necesarias para la conexión de este chip han sido llevadas a un conector de 12 pines. De esta forma el usuario tiene la posibilidad de implementar el CODEC que requiera su aplicación en una tarjeta de expansión.

GPIO Los pines de entrada/salida de propósito general en el procesador, son de dos clases: los que tienen capacidad para generar señales de interrupción (EGPIO) y los que no (GPIO). En el *EP9302* existen 7 GPIO y 11 EGPIO, los cuales están organizados por puertos. En la tarjeta se han empleado muchos de estos pines con el objetivo de disminuir el consumo de potencia y para ampliar las funciones del SoC. A continuación se listan los GPIO y EGPIO del SoC y sus funciones.

Señal	Tipo	Descripción
CGPIO[0]	GPIO	Habilitador de la PHY de ethernet
EGPIO[0]	EGPIO	Ready Flash
EGPIO[1]	EGPIO	Detector de protección contra escritura en el socket SD/MMC
EGPIO[2]	EGPIO	Detector de tarjeta en el socket SD/MMC
EGPIO[3]	EGPIO	Habilitador de la fuente de la sección del FPGA
EGPIO[4]	EGPIO	Señal $INIT\_B$ de la configuración del FPGA
EGPIO[5]	EGPIO	Señal $PROG\_B$ de la configuración del FPGA
EGPIO[6]	EGPIO	Señal $DONE$ de la configuración del FPGA
EGPIO[7]	EGPIO	Habilitador del oscilador del FPGA
EGPIO[8:15]	EGPIO	Conector de EGPIO
FGPIO[1]	EGPIO	Interrupción 1 desde el FPGA
FGPIO[2]	EGPIO	Interrupción 2 desde el FPGA
FGPIO[3]	EGPIO	Interrupción 3 desde el FPGA
GRLED	GPIO	Driver del Led verde
RDLED	GPIO	Driver del Led rojo
HGPIO[2]	GPIO	Habilitador del puerto SPI para el periférico 0
HGPIO[3]	GPIO	Habilitador del puerto SPI para el periférico 1
HGPIO[4]	GPIO	Habilitador del puerto SPI para el periférico 2
HGPIO[5]	GPIO	Habilitador del puerto SPI para el periférico 3

Tabla 2.2: Señales del bus principal

Los 8 pines EGPIO[8:15] que no han cumplido una función específica en el diseño, han sido llevados al conector U58 junto el reset del procesador y alimentación para ser usados por medio de una tarjeta de expansión.

2.3.4. Fuente del Procesador

Para su funcionamiento, el procesador requiere de dos tensiones de alimentación: 1,8V para el núcleo y el reloj y 3,3V para los periféricos y los pines de salida; estás dos tensiones se suplieron con reguladores individuales de referencia *511-LD1117SXX* con capacidad de 0,9A cada uno.

2.4. Sección del FPGA

La sección del FPGA, está orientada al desarrollo de periféricos para el procesador, por ello cuenta con algunos recursos que pueden ser empleados en el desarrollo de los mismos, como el FPGA, una memoria SDRAM, un oscilador propio, leds y pulsadores. Además de ellos esta sección cuenta con las conexiones con el procesador necesarias para la comunicación de los periféricos.

2.4.1. FPGA

Dado que en el FPGA se sintetizarán principalmente periféricos entrada salida y/o elementos de coprocesamiento, se seleccionó un FPGA que contara una alta relación recursos lógicos sobre pines entrada salida. Se ha escogido entonces el FPGA de Xilinx *XC3S250E-PQ208* que pertenece a la familia Spartan-3E que posee esta característica. Adicionalmente está dotado de multiplicadores implementados dentro del chip, lo que optimiza la implementación de funciones relacionadas con procesamiento de señales.

Configuración

El FPGA es un dispositivo volátil, lo que implica que debe ser configurado cada vez que el sistema se energice. La configuración del FPGA es el procedimiento por el cual el dispositivo recibe la información de las interconexiones internas necesarias para su funcionamiento. Esta información está almacenada en un archivo .bit el cual es el resultado del proceso de diseño y es generada por *software* a partir de la descripción de *hardware* hecha por el diseñador.

El FPGA puede ser configurado en 6 modos diferentes, según lo especifica la hoja de datos del dispositivo. Cada uno de ellos brinda al FPGA la posibilidad de recibir la información de configuración de diferentes fuentes como memorias paralelas, seriales o dispositivos externos como microprocesadores u ordenadores de escritorio. El proceso de configuración del FPGA inicia cuando el dispositivo es energizado, o cuando la señal $PROG\_B$ toma un nivel bajo de

forma continua durante por lo menos 0,5 us. Posteriormente el FPGA responde poniendo a uno el pin $INIT\_B$, tras lo cual inicia la transferencia de los datos según el modo de configuración seleccionado. El modo de configuración es definido por los pines M_0, M_1 y M_2, los cuales son muestreados al iniciar el proceso de configuración cuando la señal $INIT\_B$ tiene su flanco ascendente.

Se han empleado 3 modos para permitir al sistema configurar el FPGA en diferentes instantes que la aplicación lo requiera y con diferentes archivos de configuración. Los pines de configuración del FPGA fueron conectados según se muestra en el esquemático de la figura 2.7.

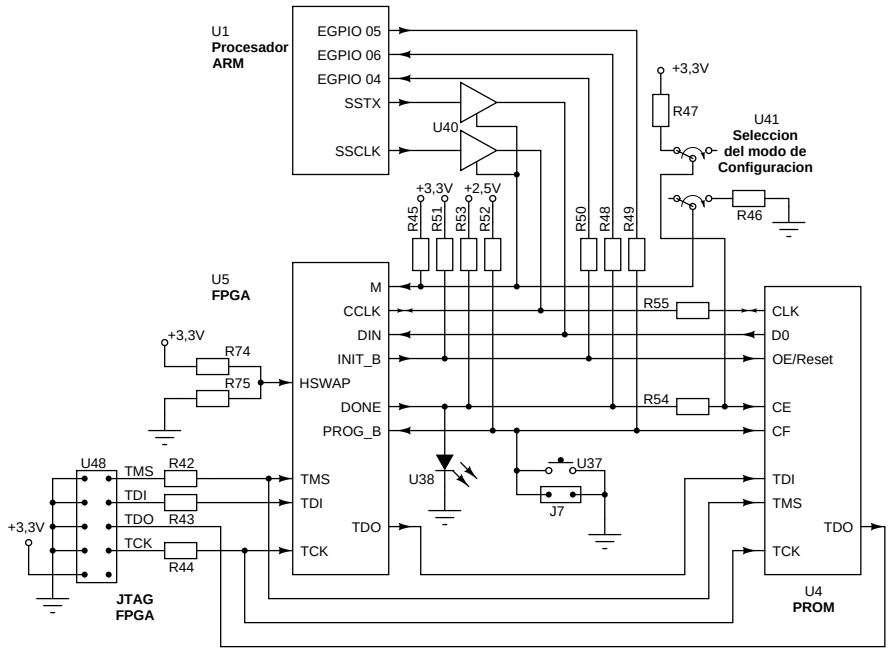


Figura 2.7: Configuración del FPGA

Esta configuración permite configurar o reconfigurar el FPGA desde una memoria de configuración, o desde el procesador, o desde un ordenador por medio del puerto JTAG. El conmutador doble U41 permite al usuario seleccionar entre los dos primeros modos, mientras el puerto

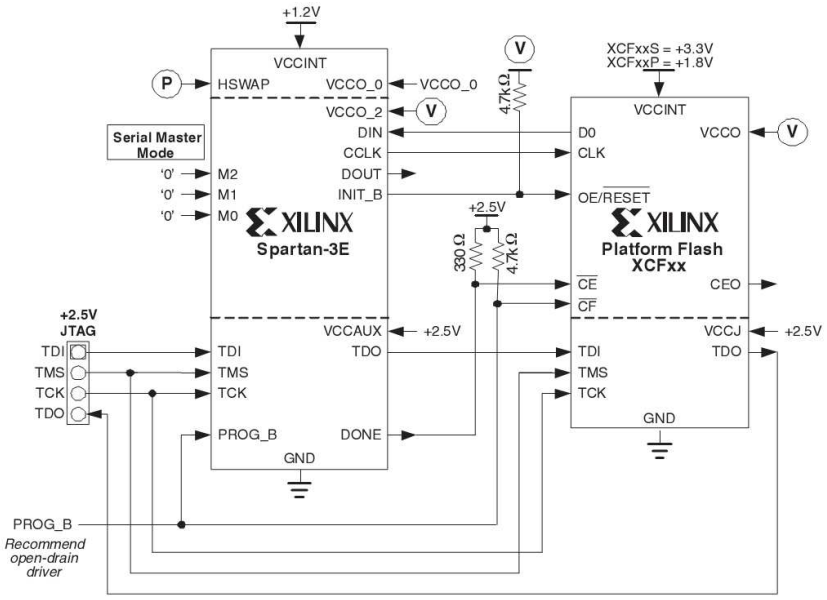
JTAG está en todo momento disponible para iniciar el proceso de reconfiguración. A continuación se mostrarán los detalles de las diferentes alternativas disponibles para reconfigurar el FPGA.

Por medio de la memoria PROM Las memorias PROM son dispositivos desarrollados por el fabricante del FPGA que permiten almacenar uno o varios archivos de configuración para poder inicializar el FPGA al momento de energizar el sistema, sin necesidad de un procesador o de complejas interconexiones. El protocolo de acceso a la memoria es serial y para leer la información de configuración desde la memoria, el FPGA tiene implementado un controlador que inicia el proceso de configuración automáticamente en caso de que el FPGA no se haya configurado aún. Este modo de configuración se llama maestro serial y para que el FPGA entre en este modo, sus tres pines de configuración deben ser muestreados en nivel bajo.

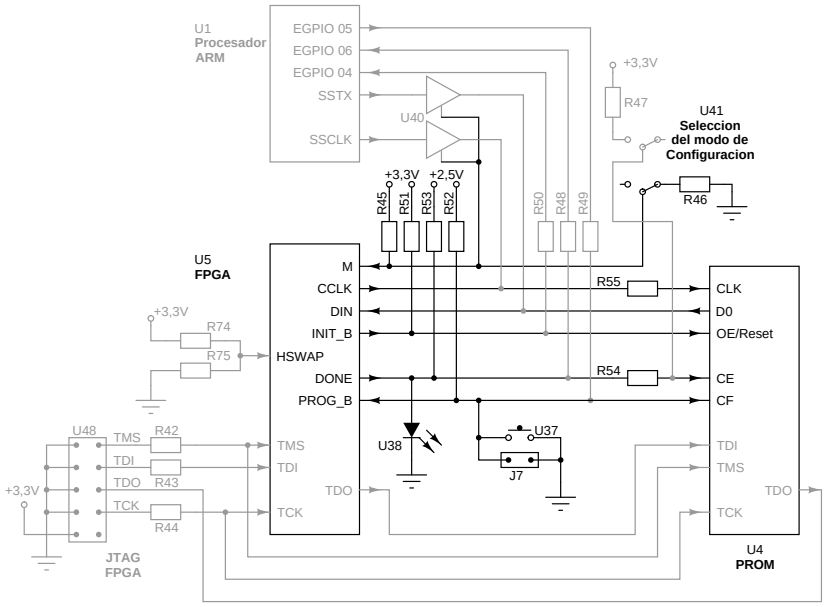
Las conexiones necesarias para que el sistema funcione de la manera indicada anteriormente se detallan en las hojas de datos de los dispositivos y se muestran en la figura 2.8(a). El conmutador doble U41 debe estar en la posición que se muestra en la figura 2.8(b), así el buffer de las señales SSTX y SSCLK del procesador estará inhabilitado y los pines de configuración M_0, M_1 y M_2 estarán en bajo.

En este modo, el FPGA genera la señal de datos y la señal de reloj que requiere la memoria PROM. Dado que la entrada del buffer triestado esta en bajo, la salida de estos se encuentra en alta impedancia y no se generará ningún conflicto a menos que se cambie el estado del conmutador U41 mientras se está realizando la reconfiguración.

Desde el procesador Gracias al arreglo que se implementó el procesador tiene la capacidad de reprogramar el FPGA en cualquier instante y para ello hace uso del modo serial esclavo. Este modo es similar al serial maestro, excepto porque el reloj es generado por el dispositivo externo, como se muestra en la figura 2.9(a). Para entrar en este modo, el conmutador U41 debe estar situado según lo muestra la figura 2.9(b). Esta posición pone en alto los pines de



(a) Diagrama recomendado por Xilinx



(b) Esquemático de la tarjeta

Figura 2.8: Configuración del FPGA desde la memoria PROM

modo de configuración por medio del pull-up R45, a la vez que activa los buffers para que la señal de reloj y de datos del FPGA lleguen desde el procesador.

Para la generación de estas señales, se ha empleado el puerto SSP[7] del procesador, dado que es posible configurarlo para que cumpla con las especificaciones de tiempo necesarias en las señales de configuración. Estos requerimientos se detallan en la figura 2.10.

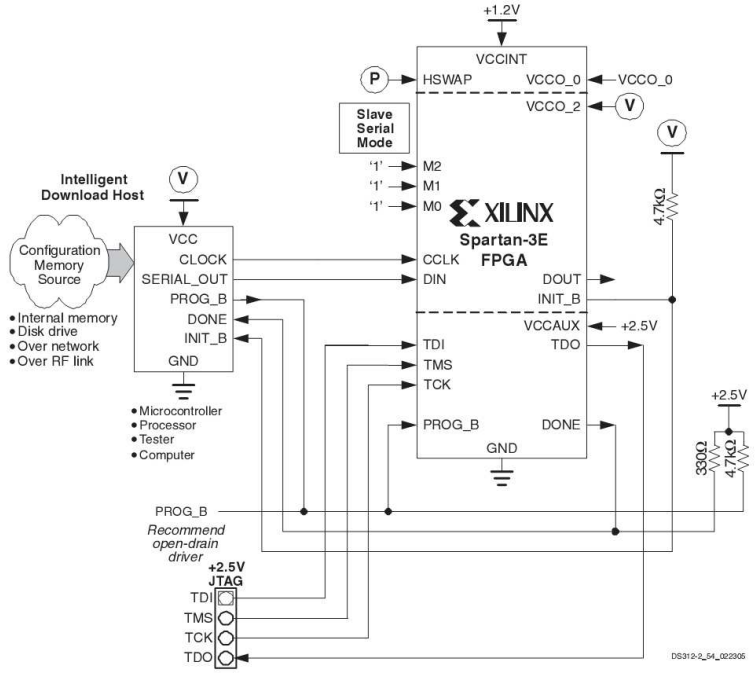
El puerto SPP del procesador comparte los pines con el puerto I2C, pero esto no afecta el funcionamiento normal del circuito dado que este puerto no se empleará en la tarjeta. El puerto tiene la capacidad de manejar interfaces SPI, Microwire o interfaces seriales de TI[8]. Las diferencias entre estas interfaces radican en los flancos activos de reloj, el número de bits por frame y la lógica de las señales de habilitación. Para la configuración del FPGA se ha seleccionado el formato SPI, el cual puede enviar frames desde 4 hasta 16 bits, y es posible configurar el nivel inactivo y la fase de la señal de reloj por medio de los bits de configuración SPO y SPH. La gráfica 2.11 muestra la transmisión de frames por medio de SPI con $SPO = 0$ y $SPH = 0$. Esta configuración busca generar la forma de onda de configuración del FPGA mostrada en la figura 2.10 ya que el estado inactivo de la señal de reloj es bajo y el flanco activo es el primero (subida)[9].

Por medio del puerto JTAG JTAG es la forma más sencilla de configurar el FPGA desde un ordenador, dado que el protocolo empleado para la comunicación es serial y además el FPGA tiene pines dedicados para dicha comunicación. En el host, la herramienta software que permite la utlización de este puerto se llama Impact y pertenece al paquete ISE, el cual puede ser descargado desde la página de Xilinx. JTAG organiza los dispositivos en una cadena, y por medio de él es posible programar los dispositivos que formen parte de la misma. Dentro de esta cadena se han añadido el FPGA y la memoria PROM de configuración *XCF02S* , lo

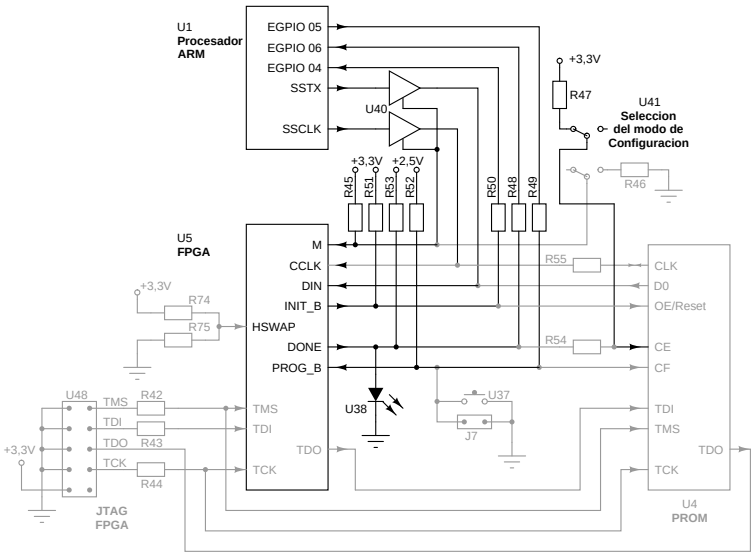
[7] Synchronous Serial Port

[8] Texas Instruments

[9] También es posible emplear la configuración $SPO = 1$ y $SPH = 1$ dado que en ella el estado inactivo de la señal de reloj es alto y el flanco activo es el segundo(subida).

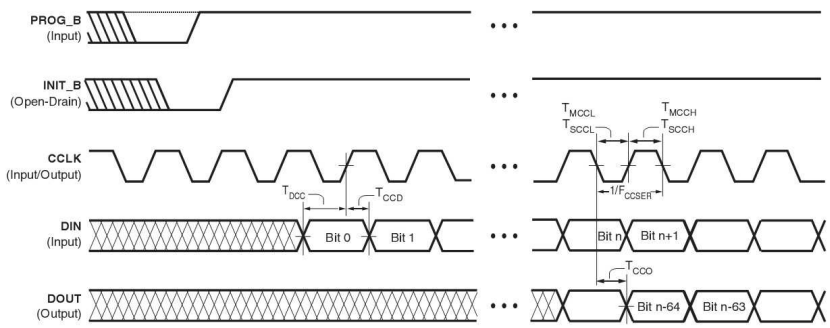


(a) Diagrama recomendado por Xilinx



(b) Esquemático de la tarjeta

Figura 2.9: Configuración del FPGA desde el procesador



Tiempo	Descripción	Min	Max	Unidades
T_{DCC}	Tiempo entre el instante en que se establece el valor de DIN y el flanco de reloj	11	-	ns
T_{CCD}	Tiempo entre el flanco de reloj y el cambio de estado de DIN	0	-	ns
T_{CCH}	Ancho de pulso en estado alto de la señal de reloj	5	∞	ns
T_{CCL}	Ancho de pulso en estado bajo de la señal de reloj	5	∞	ns
F_{CCSER}	Frecuencia de la señal de reloj	0	66/20[a]	MHz

[a] Sin compresión/Con compresión

Figura 2.10: Forma de onda del proceso de configuración del FPGA

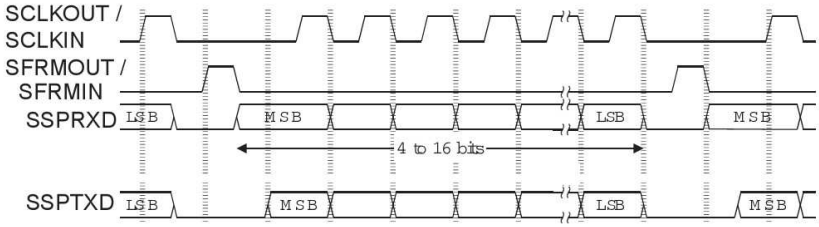


Figura 2.11: Diagramas de tiempos del puerto SPI del procesador con $SPO = 0$ y $SPH = 0$

cual significa que además de configurar directamente el FPGA, se puede grabar en la memoria PROM el archivo de configuración para que posteriormente sea descargado al FPGA por medio del modo maestro serial.

Aunque en la hoja de datos del FPGA sugieren poner los pines de selección de modo de

configuración del FPGA en $M[2.,0] = 101$, esto no es necesario, dado que los dispositivos siempre están disponibles para ser configurados por este modo. Lo único que garantiza este valor en los pines de Modo es que no se va a emplear ningún otro método diferente a JTAG.

La figura 2.12 muestra el diagrama sugerido por el fabricante del FPGA para la configuración de esta por medio de JTAG y las partes del diagrama implementado que están activas durante el proceso de configuración del FPGA o de la memoria PROM.

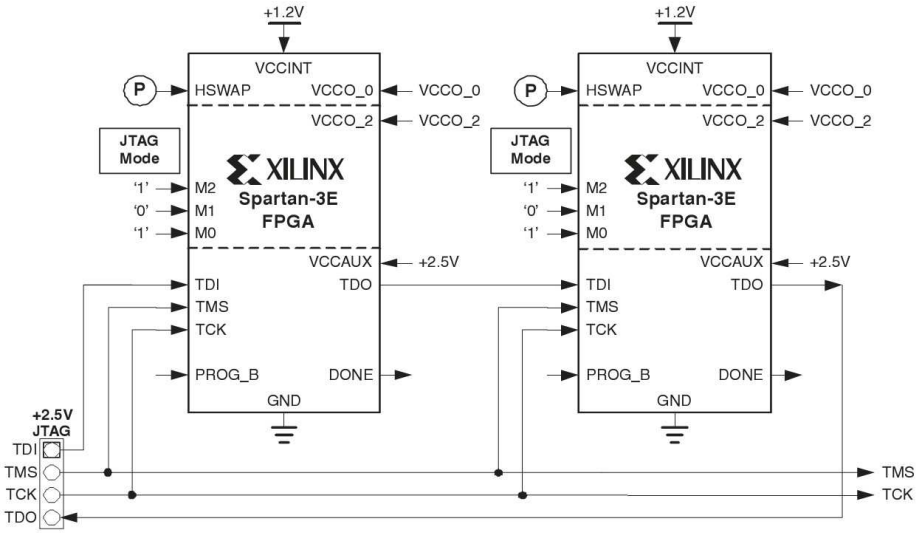
2.4.2. Periféricos FPGA

Con el objetivo de ampliar el número de aplicaciones que se pueden desarrollar en el sistema, el FPGA ha sido dotado de algunos periféricos:

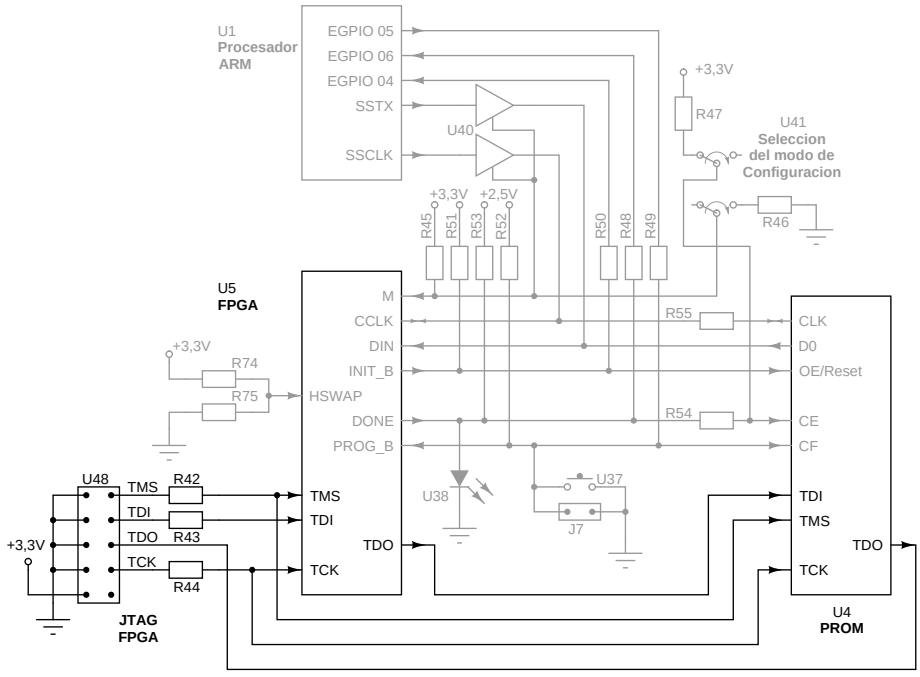
Oscilador Dado que la gran mayoría de las aplicaciones que se implementan en el FPGA son de tipo síncrono, se ha implementado un oscilador de 50 MHz en la tarjeta. La señal de este oscilador puede ser tomada por los módulos DCM internos del FPGA y adaptada según la apilcación lo requiera. El oscilador seleccionado es de referencia *C3392-50.000* y puede ser deshabilitado por el procesador para reducir el consumo de energía por medio del pin EGPIO7. La figura 2.13 presenta la conexión del oscilador.

Leds y Pulsadores Con el objetivo de brindar al FPGA medios sencillos de interacción con el usuario, se ha provisto la tarjeta de 4 pulsadores y 8 Leds. Cada uno de ellos controlado por un pin dedicado del FPGA como se ve en la figura 2.13.

Conector VGA DE-15 Teniendo en cuenta que una de las posibles aplicaciones de la sección del FPGA puede ser el desarrollo de un periférico de video, se ha implementado un conector VGA estándar para monitor, el cual gracias a un arreglo resistivo tiene la capacidad de generar 16 tonalidades en cada uno de los colores primarios RGB. El conector estándar para esa conexión es el DE-15 y para su funcionamiento requiere manejar por lo menos 5 líneas: HS Sincronización horizontal, VS Sincronización vertical, R Rojo, G Verde y B Azul. Las dos



(a) Diagrama recomendado por Xilinx



(b) Esquemático de la tarjeta

Figura 2.12: Configuración del FPGA desde el puerto JTAG

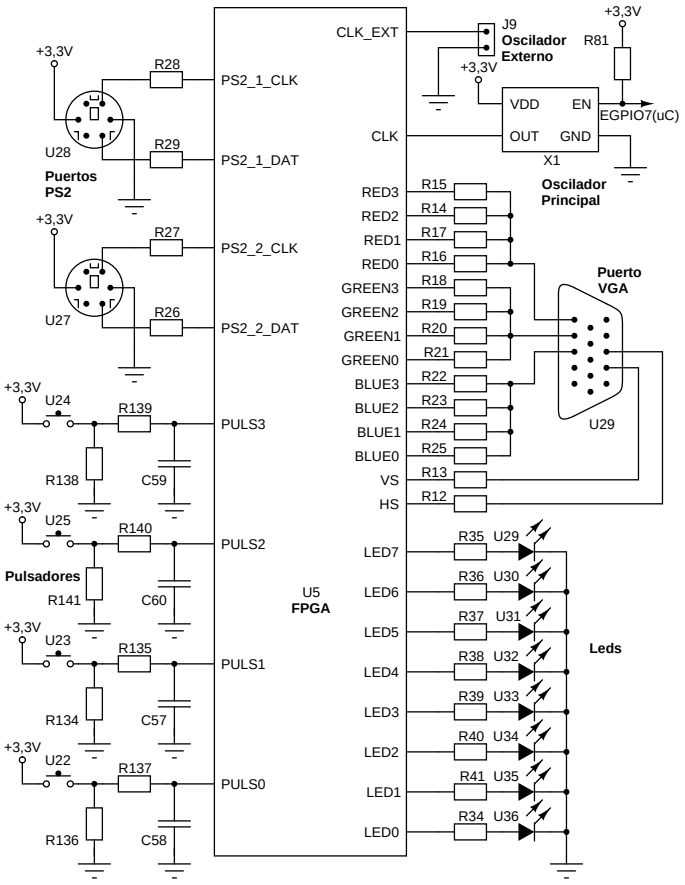


Figura 2.13: Periféricos del FPGA

primeras señales indican el inicio de una fila y un pantallazo respectivamente, mientras las tres últimas llevan la información de color de cada uno de los pixeles. La figura 2.13 muestra el arreglo de resistencias que se implementó.

El cálculo de estas resistencias se hizo teniendo en cuenta que la impedancia vista desde el terminal del cable VGA es de 75Ω y que los niveles de tensión de las señales RGB deben estar en el rango de 0 a 0,7V. El análisis por superposición muestra que el aporte de cada una de los pines del FPGA es de $0,35V$, $0,175V$, $0,0875V$ y $0,4375V$ asumiendo un nivel de 3,3V en alto y 0V en bajo. Esto permite generar tensiones entre 0V y 0,65625V en pasos de $0,4375V$ por medio de las 16 posibles combinaciones digitales.

Conectores PS/2 Dos conectores PS/2 similares al que aun se encuentra en algunas board de ordenador para la conexión de mouse y/o teclado, se han dispuesto en la tarjeta. El FPGA es el encargado de manejar esos periféricos, su protocolo es serial síncrono y solo requiere de dos líneas, una de reloj unidireccional y una de datos bidireccional. Su conexión se detalla en la figura 2.13. Las resistencias R26, R27, R28, R29 tienen como objetivo proteger al FPGA, dado que este tipo de periféricos manejan señales de 5V.

Conector de expansión Se ha añadido un conector de 40 pines que posee líneas E/S y alimentación para desarrollar periféricos que requieran recursos adicionales a los que ofrece la tarjeta.

Memoria SDRAM Algunas aplicaciones que están relacionadas con procesamiento masivo de datos o con datos de gran tamaño como imágenes, requieren grandes cantidades de memoria de rápido acceso. Por ello se implementó junto al FPGA una memoria SDRAM de 8 Mb idéntica a la del procesador. Esta memoria puede servir de memoria de almacenamiento temporal y de procesamiento para una pequeña tarjeta de video, o de datos de un coprocesador o acelerador de funciones en *hardware*. Algunos de los pines destinados a la memoria SDRAM, están compartidos con el conector de expansión del FPGA, se han seleccionado para ello líneas

del bus de direcciones dado que corresponden a pines de entrada para la memoria, lo cual no representa peligro para esta. El Apéndice A del manual de usuario[] muestra los pines usados por la memoria y los compartidos con el conector de expansión.

2.4.3. Fuente FPGA

Para el diseño de la fuente del FPGA se tuvo en cuenta algunos requerimientos exigidos por el fabricante del FPGA respecto a las corrientes y la forma como se debe inicializar la fuente, principalmente en lo relacionado con la secuencia de inicialización y el control de la pendiente de la rampa desde 0V a la tensión de alimentación. Debido a estos requerimientos tan específicos, se decidió emplear el regulador recomendado por el fabricante del FPGA, el *TPS75003* , el cual está diseñado para esta familia de FPGAs. Este dispositivo contiene integrados tres reguladores: uno de baja capacidad de corriente (300mA), el cual es empleado para la tensión de configuración y fue ajustado a 2,5V y dos que pueden manejar hasta 3A con ayuda de algunos componentes externos y fueron empleados para las tensiones de 1,2V del núcleo y 3,3V de los pines entrada/salida. El esquemático y los componentes adicionales que requiere esta fuente fueron tomados de la hoja de datos del regulador.

2.5. Comunicación Procesador-FPGA

El FPGA y el procesador son el corazón de la tarjeta, cada uno por separado representa una gran cantidad de recursos para cualquier diseñador. Según la aplicación, cada dispositivo puede sacar el mejor provecho de sus cualidades y de los recursos adicionales que posee, pero la tarjeta permite además emplear conjuntamente estos dos dispositivos gracias a los medios de comunicación existentes entre ellos. Estos medios se presentan a continuación:

2.5.1. Bus SPI

El protocolo SPI es una de las formas más sencillas de conectar un periférico; con solo 4 líneas se puede establecer una comunicación bidireccional con el *host*. El *EP9302* tiene imple-

mentado un *host* SPI y en la tarjeta diseñada se ha buscado sacar el mayor provecho a este protocolo. La arquitectura del bus en la tarjeta se muestra en la figura 2.14.

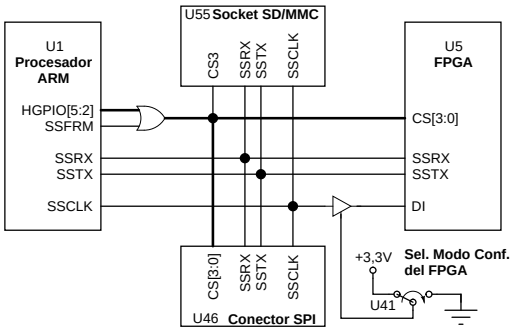


Figura 2.14: Arquitectura del bus SPI

El host SPI implementado en el *EP9302* , maneja las 4 líneas del SPI:

SSTX y SSRX: corresponden a las líneas de transmisión y recepción de datos al dispositivo.

SSCLK: dado que SPI es un protocolo síncrono requiere la transmisión de una señal de reloj. Esta es generada desde el *host*.

SFRM: empleada por algunos dispositivos como línea de habilitación.

El arreglo implementado en la tarjeta permite la conexión de hasta 4 periféricos por medio de SPI. El puerto H del procesador, de 4 bits, es el encargado de manejar los habilitadores que permiten la llegada de la señal SFRM a cada uno de los periféricos conectados. Todo esto se hace sin afectar la capacidad que tiene el SoC de reconfigurar el FPGA.

Como se aprecia en la figura 2.14 las líneas del bus SPI llegan al FPGA, lo cual permite a este alojar hasta 4 periféricos en él. La línea SSCLK no llega directamente al FPGA dado que debe pasar por medio del buffer U40, el cual se habilita poniendo el conmutador del modo de configuración del FPGA en la opción donde el procesador lo puede configurar. SSCLK llega al

mismo pin del FPGA por la cual se transmite la señal de reloj de configuración de este. Este pin puede ser empleado en el diseño circuito sintetizado luego de ser configurado el dispositivo.

En el Bus SPI también se ha dispuesto de un conector de 10 pines, que pone a disposición del usuario las líneas de datos SSTX y SSRX, las 4 líneas de selección de chip, el reloj SSCLK y líneas de alimentación para la implementación de tarjetas de expansión.

Por último se ha empleado el bus SPI para manejar memorias de tipo SD/MMC. Las memorias SD/MMC poseen entre 7 y 9 terminales y varios modos de comunicación, El modo básico de comunicación es el SPI, lo cual las hace compatibles con muchos dispositivos como microcontroladores. Además de este poseen protocolos de 1 y 4 bits de datos, los cuales tienen capacidad para una mayor tasa de transferencia de datos, aunque estos protocolos tienen restricciones privativas. Para la tarjeta ha sido adoptado el modo SPI implementado sobre la línea SSCS_3 del bus SPI.

2.5.2. Bus Principal

En el bus principal, como ya se ha mencionado, el FPGA recibe todas las líneas del bus de datos, direcciones y control necesarias para la implementación de hasta 4 periféricos en el FPGA asignados a las líneas de selección CS_0, CS_1, CS_2 y CS_3.

2.5.3. GPIOS y líneas de interrupción

Finalmente la última forma de comunicación entre el procesador y el FPGA corresponde a un grupo de pines de propósito general que tienen conexión directa entre estos dispositivos. Se ha seleccionado en el procesador el puerto F para esta función dado que cada uno de los pines de este puerto genera una señal de interrupción diferente en el procesador.

2.6. Diseño del PCB

Una vez definidos los esquemáticos y los componentes a utilizar en la tarjeta, se procedió con el diseño del PCB. Esta tarea tiene como objetivo definir físicamente las conexiones en el circuito impreso, garantizando la calidad de las señales del diseño según sus funciones y características en el circuito. Los fenómenos que se estudian en este tipo de diseño están relacionados con capacitancias e inductancias parásitas e interferencias que pueden ser generadas por fuentes externas al circuito o entre las mismas señales de este. Los análisis se hacen basados en principios de electromagnetismo y líneas de transmisión y determinan principalmente los retardos de propagación y el ruido añadido a las señales. Estos efectos se estudian en las etapas de diseño, dado que para prototipos, la fabricación de los circuitos impresos puede llegar a ser costosa, además la medición de estos efectos físicamente sobre el circuito es difícil, debido a los efectos de carga que generan los instrumentos.

En caso de trabajar en circuitos que manejan señales de alta frecuencia o con altas corrientes, los diseñadores siguen algunas reglas prácticas que ayudan a minimizar los efectos indeseados; y si lo requiere, el diseñador se puede apoyar en simuladores, que basados en parámetros de los materiales empleados y parámetros de los componentes empleados, son capaces de dar una predicción del estado de las señales en cada punto del circuito.

2.6.1. Requerimientos del PCB

Aunque los diagramas esquemáticos evidencian una gran cantidad de circuitos integrados y conectores, uno de los principales objetivos que se tuvo en su diseño fue buscar el menor tamaño posible de la tarjeta y la fácil adaptación de los módulos de expansión. También se puede observar en estos diagramas que existen ciertas secciones con características particulares las cuales definen los requerimientos en la tecnología del PCB que se va a fabricar. El bus principal es la sección que requirió mayor atención, debido a que maneja las frecuencias más altas del circuito (hasta 100 MHz) y tiene la mayor complejidad en las conexiones, dado

que está compuesto por cuatro buses de 16 bits que conectan 4 dispositivos diferentes cada uno.

Otro aspecto que define algunas exigencias en el circuito es la variedad y la distribución de las tensiones de alimentación. El diseño de la fuente muestra que se deben manejar tensiones de 1,2 2,5 y 3,3V para la sección del FPGA, 1,8 y 3,3V para el procesador y 5,0V para algunos periféricos, por lo que se asignó una capa para la distribución de las tensiones en la tarjeta. También se optó por emplear una capa dedicada para un plano de tierra con el objetivo de mantener la integridad de las señales del circuito y facilitar el flujo de las corrientes de retorno. Basados en todo ello se decidió estudiar la posibilidad de implementar el diseño en un PCB de 4, 6 o mas capas.

Para la definición del número de capas del PCB, se inició un layout con 4 capas donde dos de ellas, contenían señales, una de ellas tierra y la otra planos de poder. Este diseño requería un número considerable de vías principalmente en las señales del bus principal y por ende una separación adicional entre los componentes. Todas estas dificultades en el ruteo evidenciaron la necesidad de emplear un PCB de 6 capas.

2.6.2. El fabricante

Con conocimiento de los requerimientos del PCB, se inició la búsqueda de un fabricante con buenas especificaciones de tecnología, confiabilidad y bajo costo. Inicialmente se miraron las alternativas locales y nacionales, pero debido a que los fabricantes nacionales no cumplen con los requisitos establecidos inicialmente, debimos buscar fabricantes en otros países. Fue así como se seleccionó MyroPCB para fabricar el PCB diseñado, basados en la excelente relación tecnología/costo y en la experiencia de nuestros compañeros[38] fabricando con ellos.

MyroPCB permite fabricar los circuitos impresos con vías de 0.35 mm y 6 mils[10] de resolución en separaciones y anchos de pista. Todos sus PCBs son en material FR4(Flame Retardant

[10] 1 mil equivale a una milésima de pulgada

4), el cual es un material muy usado en la industria de los circuitos impresos, dado que es económico y ofrece buen desempeño incluso para aplicaciones de alta frecuencia. Adicionalmente ofrece buena resistencia mecánica y resistencia al fuego. Existen otros materiales usados en la industria como el Pértinax que es más económico y fácil de manejar pero menos rígido, el Rogers 4000 o Rogers Duroid que son empleados en aplicaciones de radio frecuencia de alta potencia por su baja permitividad, etc. El material FR4 cumple con los requerimientos del diseño, dado que su comportamiento es estable por debajo de 1GHz.

2.6.3. Herramientas de trabajo

Para el desarrollo del PCB, se buscó una herramienta que permitiera el manejo organizado de componentes y que contara con un entorno gráfico muy bueno, dado que el diseño maneja gran cantidad de líneas y componentes. Adicionalmente debe soportar diseños multicapa posiblemente software de analisis de integridad de señal. Se estudió la posibilidad de emplear Allegro, Mentor, Orcad y Eagle como herramientas de trabajo.

Algunas herramientas de este tipo están diseñadas para ser empleadas por un gran grupo de trabajo, donde sus integrantes asumen funciones muy específicas dentro del proceso de diseño del PCB. Por ello están distribuidas en varios tipos de programas:

Base de Datos: Organiza la información de todos los símbolos, pads y footprints de cada uno de los componentes que se emplearán en el diseño, esto permite modificar y reutilizar desde un componente completo hasta un tipo de hueco con una medida y forma específica. Es importante decir que estos programas no poseen sus propias librerías, dado que los desarrolladores del programa asumen que el equipo de creación de footprints del usuario se encargará de la tarea de generar los pads y footprints que se emplearán en el diseño.

Captura del esquemático: La captura del esquemático consiste en definir la cantidad de elementos que se emplearan de cada uno de los componentes y las interconexiones que hay entre ellos. Este tipo de programas generan un archivo netlist el cual es posible exportar para que otras herramientas puedan llevar a cabo la tarea del layout.

Layout: Este tipo de programas permiten definir físicamente las interconexiones de los componentes, por medio de él se define la ubicación de los componentes en la tarjeta, grosor y distancias entre las pistas, etc. Los programas que generan los layout, disponen de algunas herramientas complementarias como "autorouting" que permiten generar todo el layout de manera automática basados en unas restricciones que define el usuario.

Análisis de integridad de señal: Los programas que hacen este tipo de tareas se basan en la información que suministra el programa que genera el Layout y con los modelos IBIS[11] de los pines de entrada y salida de los circuitos integrados, son capaces de predecir los retardos de propagación de una onda en la línea de transmisión, así como la distorsión de la señal debido a otras líneas agresoras.

Después de manejar un poco estas herramientas, hemos seleccionado las de Mentor Graphics, dado que cumple con las caracteristicas nombradas. Con respecto al manejo de la interfaz gráfica, preferimos Mentor respecto a los demás programas, dado que permite fácilmente mover los trazos, rutear múltiples líneas simultáneamente y trabajar simultaneamente desde dos estaciones sobre el mismo proyecto; respecto a la base de datos, el manejo mas organizado lo hacen Mentor y Allegro, igualmente solo estos dos programas permiten hacer análisis de integridad de señal.

Las herramientas de Mentor están organizadas en *workflows*, es decir, están agrupadas en paquetes que en cualquier caso permiten llevar a cabo todo el proceso desde la captura del esquemático y la generación de footprints hasta el diseño del layout del PCB. El workflow empleado para el diseño del PCB se llama DxDesigner-Expedition, dado que así se llaman las herramientas de captura de esquemático y layout respectivamente. Las herramientas que se pueden emplear en este workflow son:

Library Manager: Permite el manejo de la librería de componentes símbolos y pads. Esta herramienta establece una jerarquía de componentes así: en el nivel mas bajó se encuentran

[11] I/O Buffer Information Specification

los *pads* que son la huella de un pin en el PCB y que conforman los pines de las *celdas*, las cuales son la huella de todo un chip sobre el PCB; por otro lado están los *símbolos* que representan los puntos de conexión del chip, los cuales son asociados a pines en la celda por medio de una estructura que se conoce como *componente*. Para la creación y generación de cada uno de estos elementos el library manager se apoya en subprogramas que ayudan a acelerar procesos, por ejemplo el *SymbolGenerator* tiene la opción para crear símbolos automatáticamente por medio de archivos cvs[12] y *Expedition* que permite la creación de celdas basado en algunos encapsulados y arreglos de pines normalmente usados por los circuitos integrados como SOIC, BGA, QFP, etc.

Constraint Manager: Esta herramienta administra las restricciones que posee cada señal del diseño, las agrupa y les asigna propiedades para que sean tratadas adecuadamente en el proceso de generación del layout. Estas restricciones están relacionadas con tiempos de propagación, longitud del trazo, pares diferenciales y distancias con otros elementos.

HyperLynx: Es una herramienta que permite hacer análisis de integridad de señal, se integra al flujo de diseño tomando archivos exportados de Expedition PCB que contienen la información de las líneas que se van a analizar. Esta información consiste básicamente en la localización de los trazos, distancias de separación, espesor del cobre, separación de las capas, etc. Igualmente HyperLynx puede trabajar independientemente dado que permite introducir manualmente esta información.

Expedition PCB: Esta herramienta permite el trazado de los caminos en el PCB, admite diseños multicapa y tiene una interfaz y un manejo gráfico muy bueno. A lo largo del diseño revisa interactivamente las reglas de diseño relacionadas con distancias que el usuario ha configurado directamente por medio de este programa o por medio del constraint manager. Genera los archivos *.gdo que se envían a los fabricantes de PCB y permite exportar el diseño para que pueda ser analizado por *HyperLynx* para análisis de integridad de señal.

[12]CVS (Comma Separated Values) es un tipo de archivo que contiene datos en forma de tabla donde los separadores son comas y saltos de línea

DxDesigner: Es la herramienta para la captura del esquemático. Emplea los símbolos almacenados en la base de datos creada por medio del *Library Manager* para determinar las conexiones entre los componentes. Tiene algunas funcionalidades adicionales como la interacción con el constraint manager, la creación de conexiones de manera organizada por medio de hojas de cálculo y herramientas de sincronización que permiten seleccionar y modificar señales simultáneamente en el *DxDesigner* y en *Expedition PCB*. Una alternativa a esta herramienta que no pertenece a este *workflow* pero que también es creada por Mentor Graphics se conoce como *DesignView*.

I/O designer: Dado que en los circuitos que involucran FPGAs, el diseñador puede seleccionar los pines que empleará según las funciones requeridas por la aplicación, esta herramienta permite optimizar esta distribución de pines para que se facilite el trazado de las líneas en el PCB.

Xtreme: Se presenta como una modalidad de *Expedition PCB* que permite el trabajo simultáneo sobre un mismo layout de hasta 16 diseñadores que comparten una conexión por medio de una red LAN. Para emplear esta modalidad, uno de los ordenadores debe contiener los archivos del diseño y actúa como host, mientras los demás deben iniciar una "sesión Xtreme" que les permitirá observar y realizar cambios en el diseño.

DashBoard: Esta herramienta permite acceder fácilmente a todas herramientas anteriormente nombradas, visualiza los archivos de los proyectos y ayuda a visualizar los documentos de ayuda de las todas las herramientas. Es empleada como aplicación de bienvenida al usuario.

2.6.4. Captura del esquemático y Creación de la base de datos

Para realizar la captura del esquemático, fue necesario disponer de la base de datos dado que ella contiene los símbolos que representan los componentes. Este proceso de captura y creación de la base de datos se hizo paralelamente. En total, se instanciaron 352 símbolos y la base de datos se alimentó con 46 *pads*, 45 *símbolos* y 39 *celdas* que dieron lugar a 43

componentes diferentes. Esta fase requirió de un tiempo considerable, dado que se debe prestar atención a las medidas y distancias de los pads de los componentes, la distribución de pines y la correspondencia entre el símbolo y su distribución física.

El diseño de los símbolos se ha hecho de modo que tengan la misma distribución que el chip físico, igualmente el diagrama esquemático de la tarjeta, se ha hecho sobre una sola hoja de trabajo con el objetivo de facilitar el diseño de la estrategia de ruteo en el PCB.

2.6.5. Generación del Layout

La generación del Layout se puede definir como el proceso de definir físicamente los caminos establecidos en el diagrama esquemático. Es un proceso que requiere principalmente de experiencia de parte del diseñador para establecer una estrategia que permita que las señales no se degraden al pasar por las pistas de cobre del PCB. Para llevar a cabo este proceso, existen unas subetapas, inicialmente se definen algunos parámetros del PCB, posteriormente se realiza la ubicación de los componentes, el ruteo y finalmente se evalúa y optimiza el resultado obtenido.

Entre las primeras decisiones tomadas sobre el PCB, se estudió la distribución de las capas. En diseños que manejan 6 capas con un plano de tierra y un plano de poder, existen 2 configuraciones normalmente usadas, la primera de ellas asigna las capas 3 y 4 para tierra y poder respectivamente y mientras la segunda dispone de la capa 2 para tierra y la capa 5 para poder como se muestra en la figura 2.15. Los planos contiguos al de tierra y al de poder son los más inmunes a interferencias. Dado que el diseño tiene solo unas pocas líneas críticas y la mayoría de estas serían ruteadas por las capas 1 y 6, se decidió usar la segunda opción, que permite guardar de mejor forma la integridad de señal de las capas externas.

Otra decisión fundamental, tuvo que ver con el método de ruteo del PCB. Inicialmente se optó por rutear el PCB de manera automática, configurando la opción de *autorouting* de Expedition PCB, para ello se indicaron algunas de las restricciones de las señales del diseño. El

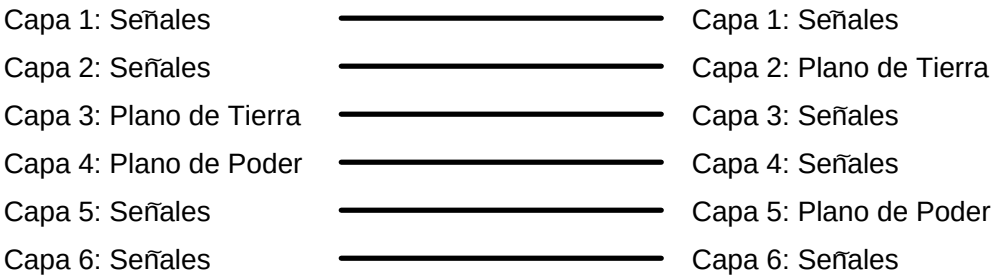


Figura 2.15: Alternativas para la distribución de las capas del PCB

manejo del *autorouting* se hace por medio de varios comandos que pueden actuar sobre grupos de líneas o sobre todo el diseño. El empleo de estos comandos permite un diseño interactivo que permite al usuario evaluar el desempeño del programa después de la ejecución de cada uno de ellos. Algunos de estos comandos son: *Fanout* que genera vías en cada uno de los componentes del sistema para que pueda ser llevado a otros dispositivos por medio de las capas internas. *No via* que realiza conexiones por medio de trazos horizontales y verticales sin emplear vias adicionales. *Tune Delay* que añade a las líneas ruteadas trazos en forma de serpentina con el objetivo de cumplir con los requerimientos de tiempos de propagación y longitud de las líneas, etc.

Para probar las capacidades de la herramienta y ganar experiencia con ella, se ruteó el bus principal del procesador con el *autorouting*. Pero los resultados no fueron satisfactorios, ya que comparado con el ruteo manual, el *autorouting* requiere espacio adicional de separación entre los componentes para llevar a cabo el ruteo. Por ello se decidió hacer el ruteo completo del PCB manualmente.

Una vez definida la distribución de las capas y el método de ruteo, se inició el proceso de localizar los componentes. En el se buscó encontrar una disposición de los elementos que permita el ruteo de las señales de manera cómoda, al tiempo que se minimiza el tamaño del circuito impreso. Como primera observación, se notó que el tamaño de la tarjeta iba a estar definido principalmente por la cantidad de conectores y no por la densidad de caminos. Se

decidió que los conectores de expansión podrían ir en el borde de la tarjeta o en el interior, dado que estos conectores tienen como función albergar otro PCB, mientras los conectores que por lo general reciben cables como Ethernet, USB, Seriales, etc. irían al borde de la tarjeta. La ubicación de componentes inicial no es definitiva y puede variar significativamente a medida que se avanza con el ruteo, pero ayuda a definir algunas secciones del circuito que pueden requerir un tratamiento especial debido a las características en sus señales. Las secciones identificadas dentro del circuito fueron la fuente de poder, el bus principal, la PHY de ethernet, el bus SPI incluyendo las señales de configuración del FPGA y los periféricos digitales del FPGA y el SoC.

El ruteo del PCB se desarrolló teniendo en cuenta algunas reglas prácticas que se pueden emplear en esta etapa del proceso. Estas buscan disminuir los efectos de interferencia entre las señales del circuito, a la vez que proteger estas señales de interferencias externas. Algunas de estas reglas son:

Regla 1: Reducir la longitud de las líneas

Como regla general en el ruteo de PCBs, se debe reducir al máximo la longitud de las líneas para evitar los efectos de resistencias, capacitancias e inductancias parásitas. En las líneas de potencia, las resistencias parásitas añaden resistencia a la fuente, mientras en las líneas de alta frecuencia, se generan retardos y alteraciones significativas debido a la magnitud de las capacitancias e inductancias parásitas generadas.

Regla 2: Evitar llevar líneas en forma paralela a otras líneas ruidosas

Debido a que el acople magnético entre dos conductores se maximiza al estar estos en forma paralela, se debe evitar este tipo de arreglos con líneas ruidosas que puedan afectar las adyacentes. Si se requiere pasar sobre una línea ruidosa, es aconsejable hacerlo en forma perpendicular, especialmente si las dos líneas están en capas adyacentes.

Regla 3: Evitar ángulos rectos en las líneas

Matemáticamente y sustentado por las leyes de Maxwell, se ha demostrado que existen emisiones electromagnéticas y reflexiones de la señal generadas por ángulos rectos en las líneas de transmisión. Esto debido a la variación en el ancho de la pista, lo cual genera discontinuidades en la impedancia de la línea. En la práctica algunos autores han observado que estas emisiones no son significativas a menos que se trabaje con muy altas frecuencias, cercanas a los 3 GHz[26][1]. Aunque el diseño no posea señales de esa frecuencia, es aconsejable evitar el uso de ángulos rectos en los trazos del PCB.

Regla 4: Impedancia de línea constante

Los cambios de medio y las terminaciones de las líneas son los principales factores inherentes a la línea que pueden generar problemas de integridad de señal. Por ello es importante que la impedancia de la línea se mantenga constante desde su punto de inicio hasta su llegada, para ello es necesario evitar en lo posible los cambios de capa en las líneas, así como cruzar los bordes de los planos de tierra y poder.

Regla 5: Ubicación de los condensadores de desacople

Los condensadores de desacople tienen como función filtrar ruido proveniente de la fuente que puede ser generado por la misma fuente o por otros componentes del circuito que también son alimentados por la misma fuente. De este hecho se deduce que estos capacitores deben estar ubicados lo mas cerca posible a cada una de las cargas del circuito. En los circuitos integrados que poseen diferentes tensiones de alimentación y que poseen varios pines de alimentación, se recomienda poner un capacitor de desacople cerca a cada uno de estos pines. Los capacitores normalmente usados son de 0.1uF de tipo cerámicos, para el caso del FPGA se debería seleccionar el valor del capacitor según la frecuencia de trabajo de la aplicación, pero como el sistema diseñado esta abierto a cualquier aplicación, se usó el valor por defecto de 0,1uF.

Regla 6: Ruteo de pares diferenciales

Los pares diferenciales son dos líneas de transmisión que se emplean para transmitir una sola señal, donde el valor de esta corresponde a la diferencia de tensión de las dos líneas. Este tipo de líneas son ampliamente usadas debido a la alta inmunidad al ruido en modo común, es decir, si existe una fuente de ruido externa que afecte las líneas de transmisión, ambas se verán afectadas de igual manera manteniendo la diferencia de tensión constante. El ruteo de estas líneas de transmisión se debe realizar llevando las dos líneas paralelamente manteniendo en lo posible la longitud y la impedancia de las líneas constante.

Para iniciar el ruteo de un PCB se recomienda localizar los capacitores de desacople de los circuitos integrados. En nuestro caso se inició haciendo el ruteo del bus principal por la alta complejidad de este. El ruteo se realizó en dos etapas que involucran las líneas antes y después del buffer. Antes del buffer se encuentran el procesador, y las dos memorias RAM y Flash, por lo que se ha buscado la menor longitud en estos trazos. Para ello fue necesario ubicar el procesador en la capa superior y las memorias en la capa inferior del PCB. Por su parte los 4 buffers *SN74LVC245a* se han distribuido en las capas superior e inferior del PCB, lo cual permitió un ahorro considerable de espacio en esa zona, aunque incrementó considerablemente la densidad de componentes y trazos. El ruteo de esa etapa se ha realizado por las capas externas, las cuales contienen impedancias similares y son las más inmunes al ruido de otras capas.

Las líneas ubicadas después del buffer deben llegar al FPGA y al conector externo. Dado que el conector externo posee pines de hueco pasante, las líneas pueden llegar a él desde cualquiera de las capas, inclusive las internas, por eso se han empleado estas capas para este propósito. La distancia entre el buffer y el FPGA ha sido minimizada.

Con el bus principal ruteado, se procedió a ubicar los capacitores de desacople del procesador, las memorias, los buffers y el FPGA procurando disminuir las inductancias parásitas de los caminos. Posteriormente se han ruteado dispositivos que no tienen grandes requerimientos

en el ruteo como los periféricos del FPGA, periféricos del procesador como el UART y los conectores de expansión. El ruteo del puerto USB requerió solamente la ubicación de los filtros *USBDF01W5* cerca al conector, según recomendacion encontrada en la hoja de datos. Las señales de los dos puertos fueron llevadas como pares diferenciales desde el procesador hasta el filtro y posteriormente al conector de la misma forma.

A continuación se inició el ruteo de las líneas relacionadas con el bus SPI. Estas líneas no son tan delicadas, excepto la línea de reloj de configuración del FPGA. Aunque la línea de reloj del FPGA provenga de la línea de reloj del puerto SPI, el buffer U40 permite mejorar la integridad de la señal en la sección donde se encuentra el FPGA, dado que este se encuentra muy cerca del FPGA.

Posteriormente se ruteó la PHY de ethernet. Se ubicó este componente en la parte inferior de la tarjeta porque era necesario invertir el orden de algunas líneas que provienen del procesador, de esta forma solo es necesario emplear una via por línea. El conector seleccionado es de monatje superficial, lo cual perimtió que se ubicara en la capa superior parcialmente sobre el chip de la PHY.

Finalmente se ruteó la fuente del sistema. El espacio destinado para estos componentes está ubicado sobre el socket de memorias SD/MMC y entre el conector VGA y el FPGA. Los switches de encendido están en la parte superior de la tarjeta y el plug de entrada está ubicado en el borde derecho de la tarjeta. La fuente del procesador está compuesta por tres reguladores fijos, a los cuales se les ha dotado de un pequeño plano en los pines de salida dado que este ayuda a reducir la resistencia térmica del regulador. Por otro lado la fuente del FPGA se ruteó basados parcialmente en el diseño que aparece en la hoja de datos del *TPS75003* . Se tuvo en cuenta que por el circuito integrado no circulan grandes cantidades de corriente, por lo cual no requiere trazos gruesos. Los trazos que requieren mayor atención, son los de los dispositivos por los que circula la corriente de salida, es decir, los transistores mosfet y las bobinas. El

mínimo ancho de camino para estos trazos asumiendo una corriente de 2A y un cambio de temperatura de 10°C al paso de la corriente se ha calculado según información tomada del estándar IPC-2221[16]. En la gráfica 2.16 se observa la relación entre la sección del conductor y la corriente que circula por él para diferentes variaciones de temperatura.

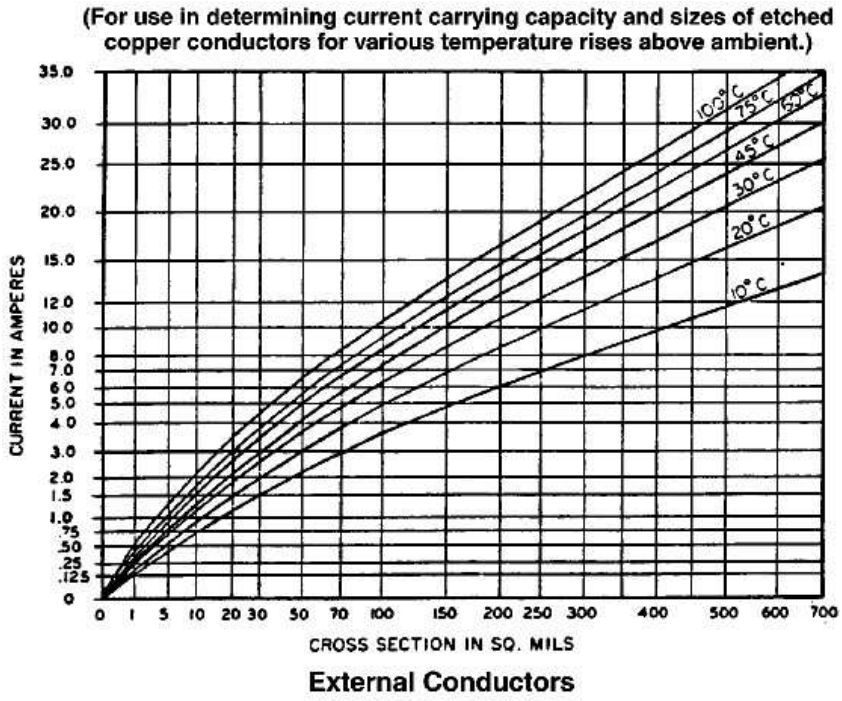


Figura 2.16: Corriente Vs Sección del conductor. (Tomada del estándar IPC-2221[16]).

Para las condiciones del diseño, se observa que la sección del conductor es aproximadamente de $50mils^2$, y teniendo en cuenta que el espezor de los conductores del pcb es 1oz. Se deduce que para esas condiciones, el mínimo ancho del camino debe ser de 0.70126 mm.

Posteriormente se hizo la distribución de las tensiones de alimentación por la capa destinada para esto. Dado que son 6 tensiones de alimentación las que se deben distribuir, se trató de disminuir al máximo los cambios de plano en las líneas críticas. En la zona del FPGA, se hi-

cieron dos anillos de 3,3V y 2,5V rodeando a un núcleo de 1,2V. En la figura 2.17 se aprecia la distribución de estos planos.

Finalmente se hicieron retoques en el ruteo de las líneas, se organizó el *silkscreen* de las capas superior e inferior que contienen la ubicación y las referencias de los componentes. Se movieron algunas para que no cruzaran por cambios de planos y se optimizaron algunas distancias para evitar interferencia entre algunas líneas. Las figuras 2.18 y 2.19 muestran las vista superior e inferior del PCB diseñado.

Los elementos que allí se muestran son los siguientes:

1. Jack de la entrada de alimentación.
2. Fuente de la sección del FPGA.
3. Fuente de la sección del procesador.
4. Pulsadores de encendido de la tarjeta.
5. Procesador.
6. Jumpers para la selección del modo de arranque del procesador.
7. Conector JTAG del procesador.
8. Pulsadores de Reset.
9. Memoria Flash.
10. Memoria SDRAM del procesador.
11. Socket para memorias SD/MMC.
12. Conector DB-9 hembra.
13. Conector DB-9 macho.
14. Transmisor/Receptor infrarojo.
15. Puertos USB.
16. Conector RJ45 de ethernet.

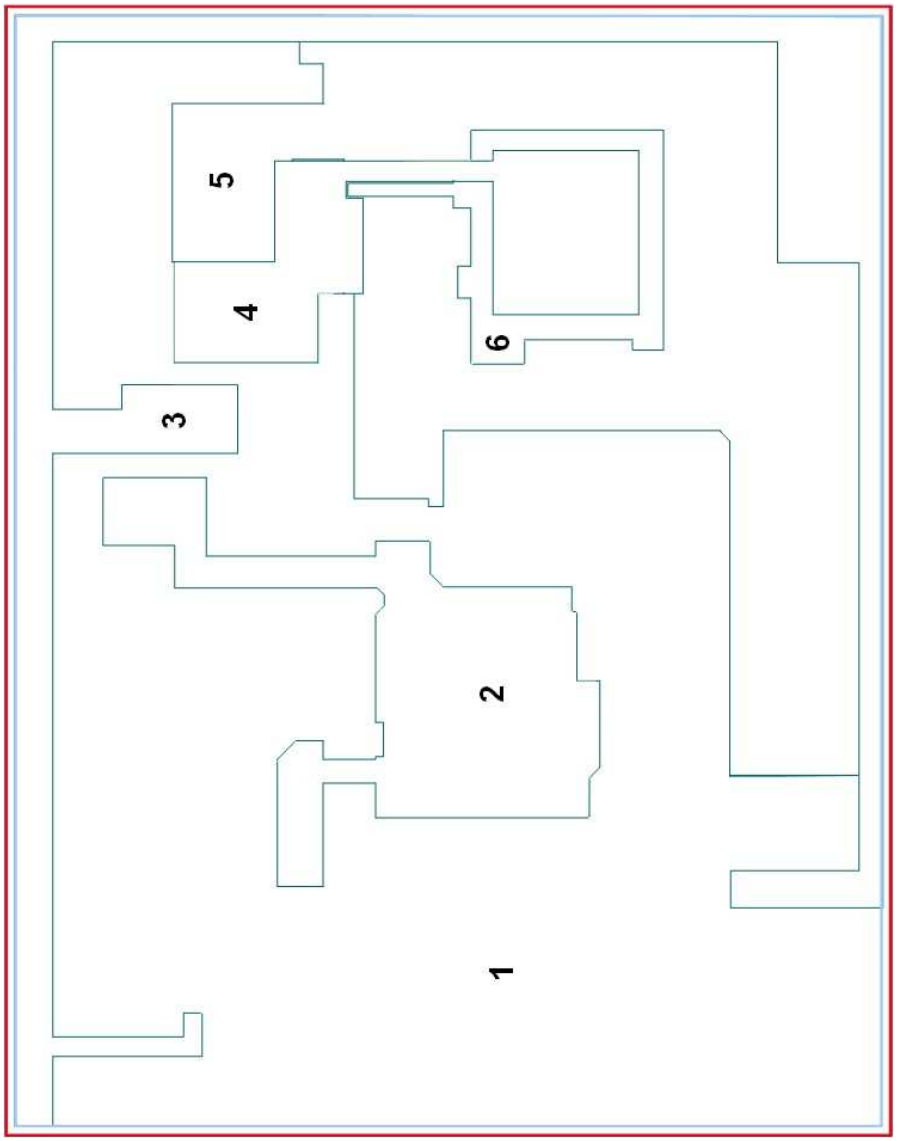


Figura 2.17: Planos de poder. **1.** 3,3V(SoC) - **2.** 1,8V - **3.** 5,0V - **4.** 1,2V - **5.** 3,3V(FPGA) - **6.** 2,5V

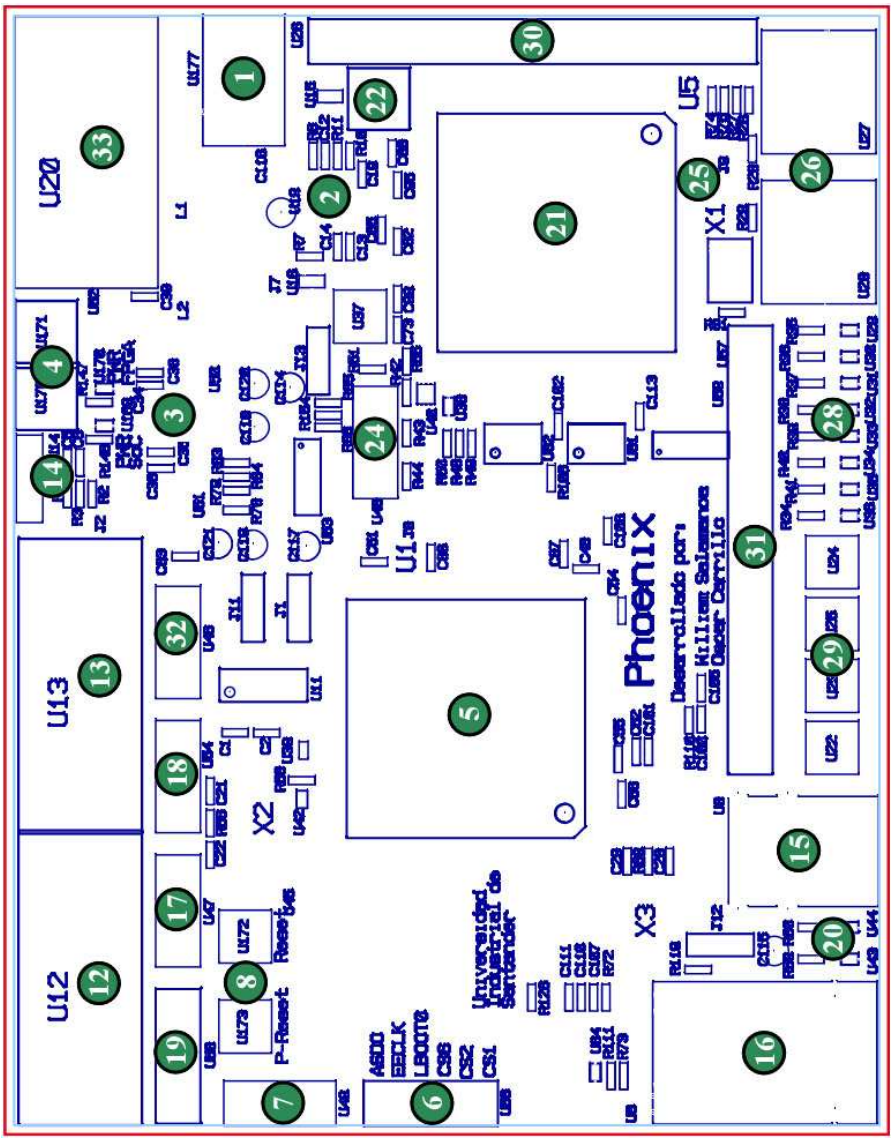


Figura 2.18: Ubicación de los componentes de la tarjeta (Vista Superior)

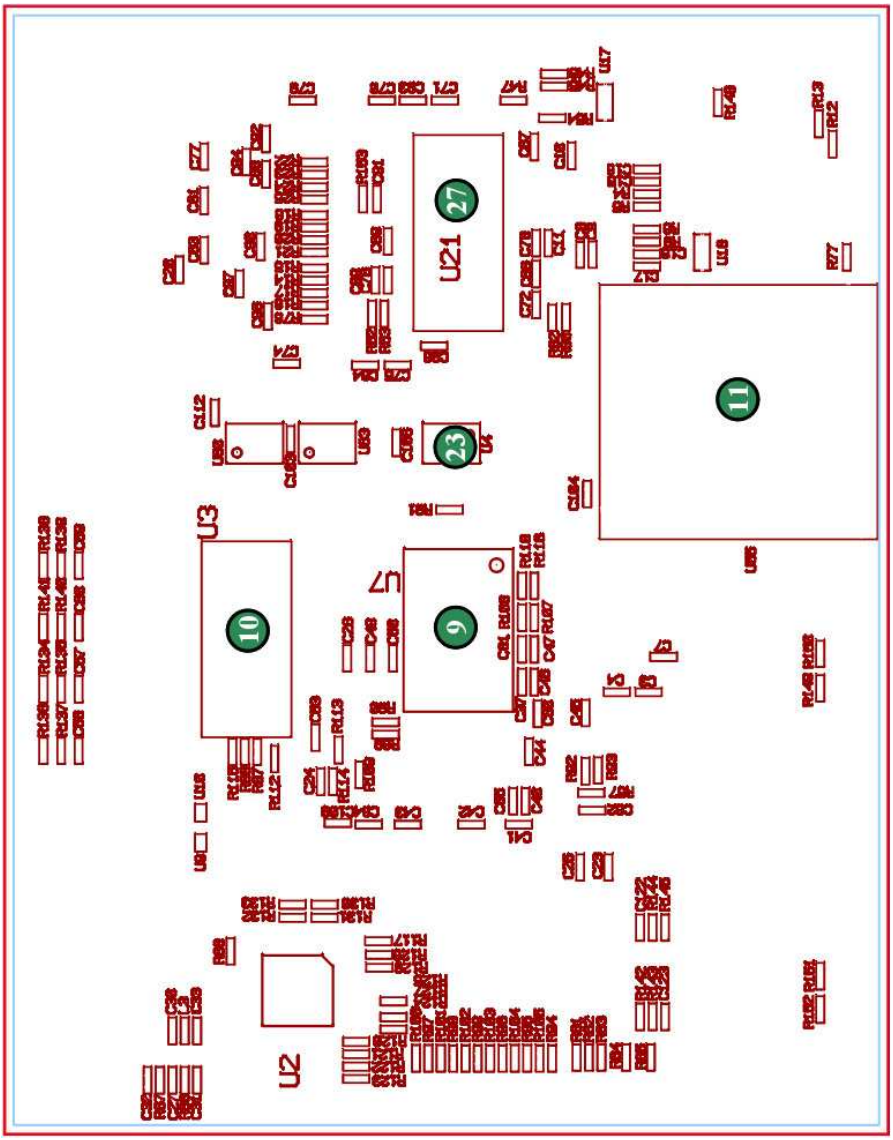


Figura 2.19: Ubicación de los componentes de la tarjeta (Vista Inferior)

17. Conector de entradas analógicas.
18. Conector para tarjeta con CODEC AC'97.
19. Conector con GPIO del procesador.
20. Leds de propósito general.
21. FPGA.
22. Conmutador para definir el modo de configuración del FPGA.
23. Memoria PROM de configuración del FPGA.
24. Conector JTAG del FPGA.
25. Oscilador del FPGA.
26. Puertos PS2.
27. Memoria SDRAM del FPGA.
28. Leds.
29. Pulsadores.
30. Conector de expansión del FPGA.
31. Conector de expansión del procesador.
32. Conector de expansión SPI.
33. Puerto VGA.

2.7. Montaje de componentes

Para el montaje de la tarjeta, se ha usado una estrategia que permite probar los componentes a medida que se instalan. Basados en ella, se soldó por secciones funcionales, empezando por las fuentes de alimentación, ya que son fundamentales para el funcionamiento del sistema.

Capítulo 3

Procesador

3.1. Procesador

3.1.1. Arquitectura

El EP9302 es un SoC de alto desempeño (ver figura 3.1[1]) que integra un procesador ARM9 de 200MHz y un gran número de periféricos, diseñados para aplicaciones de alto rendimiento, sus principales características son:

Procesador ARM920T de 200MHz.

16KB de memoria de datos.

16KB de memoria de instrucciones.

MMU.

Bus del sistema de 100MHz.

Interfaz JTAG para depuración.

Unidad matemática $MaverickCrunch^{TM}$**.**

Algoritmos optimizados para la compresión de audio digital.

Instrucciones con tipos de dato en punto flotante, enteros y para procesamiento de señal.

Periféricos Integrados.

Conversor A/D con 5 entradas y resolución de 12 bits.

MAC de Ethernet de 1/10/100 Mbps.

2 Puertos USB 2.0.

[1]Imagen tomada de www.cirrus.com

2 UARTs.

Interfaz IrDA.

Puertos SPI.

Interfaces $AC'97$ e I^2S.

Driver para memoria externa.

Hasta 2 bancos de memoria SDRAM de 16 bits.

Interfaz para memorias SRAM/FLASH/ROM de 8 y 16 bits.

Interfaz para EEPROM serial.

Periféricos internos.

Reloj de tiempo real con ajuste por software.

12 canales DMA para optimizar transferencia de datos.

Memoria ROM de arranque.

2 timers de 16 bits de propósito general.

1 timer de 32 bits de propósito general.

1 timer de 40 bits para depuración.

Pines de entrada y salida de propósito general.

16 GPIOs con uso de interrupciones.

8 GPIOs opcionales para manejo de periféricos.

3.2. Mapa de memoria del EP9302

El mapa de memoria puede variar ligeramente según el modo de arranque del procesador. La tabla 3.1 muestra como se modifica el mapa de memoria si el arranque es en modo síncrono.

Cuando el procesador arranca, el mapa se puede ver ligeramente modificado en sus primeras direcciones si el arranque es interno o externo, es decir si se ejecuta o no el código interno de la memoria ROM.

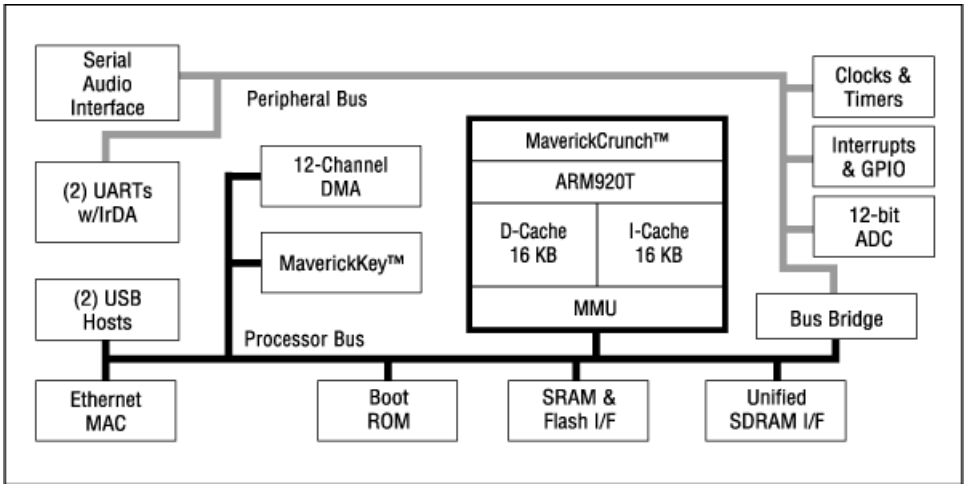


Figura 3.1: Características del EP9302

3.3. Arranque del procesador

Cuando se inicializa el procesador, después de ser energizado o después de un reset, su modo de arranque queda determinado por el estado que tengan algunos pines del chip. Estos pines establecen cada uno de los siguientes items:

Arranque interno/externo Determina si el primer código que se ejecutará en el procesador, está contenido en la memoria ROM interna o esta contenido en una memoria externa. Esto puede llegar a modificar el mapa de memoria durante el arranque.

Arranque con memoria síncrona/asíncrona Esta opción permite modificar el mapa de memoria del procesador para que en el primer sector del mapa de memoria, se ubique una memoria síncrona o asíncrona.

Ancho del bus Permite variar el ancho del bus de la memoria entre 8 o 16 bits.

La figura 3.2 muestra la secuencia de inicio del procesador y la tabla 3.2 muestra como se deben configurar estos pines para que el procesador inicie en el modo deseado.

Rango de direcciones	Arranque Síncrono	Arranque Asíncrono
0xF000_0000 - 0xFFFF_FFFF	Mem. asíncrona (nCS0)	Mem. síncrona (nSDCE3)
0xE000_0000 - 0xEFFF_FFFF	Mem. síncrona (nSDCE2)	Mem. síncrona (nSDCE2)
0xD000_0000 - 0xDFFF_FFFF	Mem. síncrona (nSDCE1)	Mem. síncrona (nSDCE1)
0xC000_0000 - 0xCFFF_FFFF	Mem. síncrona (nSDCE0)	Mem. síncrona (nSDCE0)
0x9000_0000 - 0xBFFF_FFFF	No usada	No usada
0x8080_0000 - 0x8FFF_FFFF	Mapeados en el APB	Mapeados en el APB
0x8010_0000 - 0x807F_FFFF	Reservada	Reservada
0x8000_0000 - 0x800F_FFFF	Mapeados en el AHB	Mapeados en el AHB
0x7000_0000 - 0x7FFF_FFFF	Mem. asíncrona (nCS7)	Mem. asíncrona (nCS7)
0x6000_0000 - 0x6FFF_FFFF	Mem. asíncrona (nCS6)	Mem. asíncrona (nCS6)
0x5000_0000 - 0x5FFF_FFFF	Reservada	Reservada
0x4000_0000 - 0x4FFF_FFFF	Reservada	Reservada
0x3000_0000 - 0x3FFF_FFFF	Mem. asíncrona (nCS3)	Mem. asíncrona (nCS3)
0x2000_0000 - 0x2FFF_FFFF	Mem. asíncrona (nCS2)	Mem. asíncrona (nCS2)
0x1000_0000 - 0x1FFF_FFFF	Mem. asíncrona (nCS1)	Mem. asíncrona (nCS1)
0x0001_0000 - 0x0FFF_FFFF	Mem. síncrona (nSDCE3)	Mem. asíncrona (nCS0)

Tabla 3.1: Mapa de memoria del *EP9302* en sus dos modos de arranque

EECLK	EEDAT	LBOOT0	LBOOT1	ASDO	CSn[7:6]	Configuración de boot
0	1	0	0	1	0 0 0 1	Arranque de memoria externa síncrona, SROM o SyncFLASH... valor de CSn[7:6] SFLASH de 16 bits SROM de 16 bits
0	1	0	0	0	0 0 0 1	Arranque de memoria externa asíncrona... valor de CSn[7:6] SRAM de 8 bits SRAM de 16 bits
1	1	0	1	X	0 1	Arranque serial de 16 bits
1	1	0	0	1	0 0 0 1	Arranque desde la ROM interna. El ancho del bus esta dado por el valor de CSn[7:6] 16 bits 16 bits
1	1	0	0	0	0 0 0 1	Arranque desde la ROM interna. El ancho del bus esta dado por el valor de CSn[7:6] 8 bits 16 bits

Tabla 3.2: Configuración de los pines de arranque

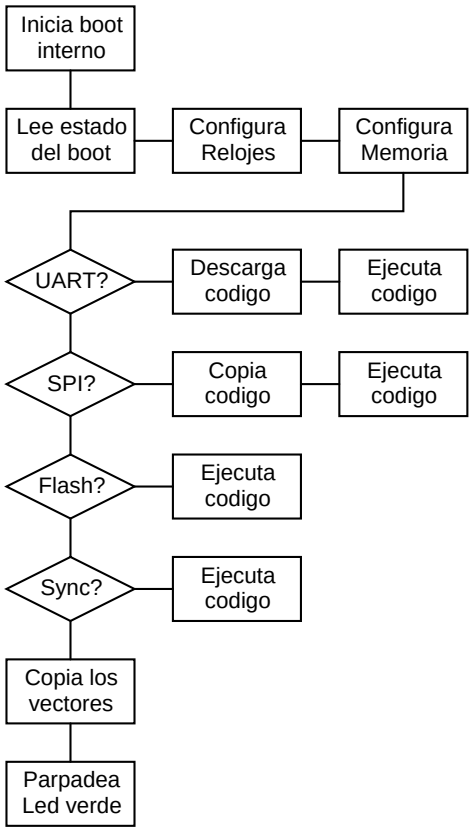


Figura 3.2: Secuencia de arranque

3.3.1. Arranque por UART

En caso de seleccionar el arranque serial de 16 bits, se puede arrancar el dispositivo desde el UART principal del procesador, es decir UART0, este arranque sigue los siguientes pasos:

1. El procesador envía un < por el puerto serial.
2. El procesador debe recibir la secuencia $SURC$ o $CRUS$, para aceptar el arranque.
3. El procesador lee 2048 caracteres a través del UART0 y los copia al buffer de ethernet situado en la dirección de memoria 0x80014000.
4. El procesador salta inmediatamente a ejecutar las instrucciones a partir de la dirección de memoria 0x80014000.

3.3.2. Arranque por SPI

Este caso sucede cuando la señal EEDAT esta en alto, LBOOT0 y LBOOT1 están en bajo, para ello en las primeras direcciones de la memoria SPI debe estar escrita la sentencia $SURC$ o $CRUS$, el código, que seguirá a estas sentencias se copiará en el buffer de Ethernet en la dirección 0x80014000. El procesador estará en modo ARM SVC. Así el usuario puede usar el código en la memoria de la MAC para cargar el resto de datos desde la memoria SPI.

3.3.3. Arranque por FLASH

Para arrancar desde una memoria FLASH el dispositivo debe estar en modo de arranque normal, y debe aparecer la sentencia $SURC$ o $CRUS$ en alguna de las siguientes direcciones de memoria en la FLASH:

0x10001000
0x20001000
0x30001000

0x40001000

0x50001000

0x60001000

0x70001000

Es decir la dirección (Flash-base + 0x1000), y el inicio del código debe estar en la dirección (FLASH-Base + 0x0). El procesador estará en modo ARM SVC.

3.3.4. Arranque por SDRAM o SyncFLASH

Cuando se arranca en este modo desde una memoria SDRAM se siguen los siguientes pasos:

1. Arranque interno con un dispositivo asíncrono.
2. Re-configuración de la SDRAM para acceso con 16 bits.
3. Salto a la memoria SDRAM deseada.

Para arrancar desde un dispositivo SyncFLASH se debe primero colocar la sentencia $SURC$ o $CRUS$ en las primeras cuatro direcciones de la memoria, en seguida de esto debe ir el código que se ejecutará. El procesador estará en modo ARM SVC.

Capítulo 4

Instalación y configuración del sistema operativo

El sistema operativo es un conjunto de programas que permiten la administración eficiente de los recursos del sistema, permitiendo además la interacción con el usuario. Para lograr un óptimo acople entre el sistema operativo y el hardware que se maneja se debe tener en cuenta las compatibilidades y ventajas que pueden ofrecer estos a la hora de implementarse en la tarjeta. Teniendo en cuenta que el procesador cuenta con MMU, usado en sistemas operativos avanzados como Linux y Windows CE, se tomó la decisión de instalar Linux debido a que es de código abierto, y ofrece mayor compatibilidad con los dispositivos del sistema. Además en la página de soporte del fabricante[1] está disponible el software necesario para trabajar con este sistema operativo. Allí encontramos el código fuente de Linux y todos los programas soportados por los procesadores Cirrus, junto con las herramientas de desarrollo necesarias para trabajar con estos procesadores.

Para iniciar la ejecución del sistema operativo, primero es necesario que el procesador ejecute un *bootloader* o cargador de arranque, que es un software con interfaz de usuario, que contiene los drivers y funciones necesarias para cargar un sistema operativo, desde el medio en que se encuentra almacenado.

[1] http://arm.cirrus.com

4.1. Instalación del bootloader

Para instalar un *bootloader* en la tarjeta, se usan las herramientas proporcionadas por Cirrus Logic, ya que han sido probadas durante largo tiempo, por eso es el método mas conveniente para trabajar con SoCs de dicho fabricante.

Entre las herramientas que ofrece Cirrus existen dos *bootloaders*: *uBoot* y *RedBoot*, entre las cuales se optó por la segunda opción, debido a que tiene mas aplicaciones y drivers que la primera. En la página de Cirrus se encuentra el código fuente y el *toolchain* necesario para realizar las modificaciones pertinentes, para que el sistema funcione correctamente. Para instalarlo se debe iniciar la tarjeta con un arranque por UART, y cargar *RedBoot* en la memoria FLASH de la tarjeta, a través del cargador o *downloader* elaborado para los procesadores ARM de Cirrus.

Para iniciar el proceso de instalación, se debe conectar la tarjeta a un puerto serie del *host*, donde se debe ejecutar el cargador. Este queda en espera a que el procesador inicie en arranque serial. Al momento que el procesador solicite código de arranque por el puerto serie, el *host* envía el dato SURC o CRUS. Esto es interpretado por el programa que se esta ejecutando desde la ROM interna del procesador, e inmediatamente se comienza a recibir un código de 2048 bytes por el puerto serie, que es almacenado en el buffer del puerto de ethernet. Una vez almacenado ahí, este código es ejecutado. La función de este código es descargar un programa mas elaborado que se almacena en la memoria SDRAM externa y que tiene como función principal escribir el bootloader en la memoria Flash, para que posteriormente el sistema pueda arrancar directamente desde esta.

4.2. Sistema de archivos

Primero que todo hay que tener en cuenta que el sistema operativo se debe instalar en una memoria que pueda leer la tarjeta directamente solo al conectarla, y para ello lo mejor es hacerlo

en memorias extraíbles como tarjetas SD, MMC, Memory Stick y memorias USB. En nuestro caso se usa una tarjeta de memoria SD de 512MB, suficiente para contener una distribución de Linux adecuada para el sistema. La manera mas sencilla de dar formato a esta memoria es montarla en un PC (de preferencia con sistema operativo Linux), mediante un lector de tarjetas, y dar formato con las herramientas del sistema operativo. Los sistemas de archivos que puede manejar el *bootloader* son EXT2, EXT3 y JFFS. Los dos primeros son nativos de Linux y el último es un sistema de archivos diseñado para para aprovechar al máximo la capacidad de memorias FLASH, y dar una excelente velocidad de acceso a los archivos allí grabados, además de contar con sistema *journalist*, de ahí su nombre que significa $journalistFLASHfilesystem$. Por compatibilidad, estabilidad y soporte, lo mejor es usar ext2 o ext3, luego de dar formato a la memoria se pueden copiar el SO en la misma.

4.3. Instalación del sistema operativo

La manera mas sencilla de instalar el sistema operativo es primero compilar las aplicaciones necesarias, usando el *toolchain*, para después copiarlas a la memoria previamente formateada. Otra forma de instalar el sistema operativo es, teniendo montada la memoria en el PC, especificar al *toolchain* que instale allí las aplicaciones compiladas, esta manera es la mas adecuada y rápida. Existe una tercera forma de instalar un sistema operativo desde la misma tarjeta, con ayuda del *bootloader*, se arranca un sistema de archivos remoto por NFS *(network file system)*, y desde allí se copia el sistema en la memoria local. Este último método es el más lento de todos, y solo se utiliza para instalar aplicaciones pequeñas.

Capítulo 5

Conclusiones y Observaciones

A lo largo del presente proyecto se ha logrado realizar el diseño y una primera implementación del sistema que se ha denominado Phoenix que busca convertirse en una herramienta para el desarrollo de aplicaciones que requieren sistemas embebidos. Se ha demostrado la viabilidad para construir esta herramienta que ofrece lo necesario para diseños que involucren desarrollo de *hardware* por medio de HDL, desarrollo de software basados en un procesador ARM o estas dos variantes reunidas en una metodología de codiseño; todo en un solo sistema de desarrollo. Hasta el momento a nivel local, el desarrollo de aplicaciones de este tipo solo era posible sobre sistemas de desarrollo individuales. El presente proyecto ha buscado desarrollar una herramienta que integre estos campos.

Adicionalmente se ha hecho un primer acercamiento al diseño de circuitos impresos de múltiples capas, complementando la experiencia de fabricación de PCBs en el exterior aportada por nuestros compañeros [38]. Se ha ratificado la confiabilidad, el nivel tecnológico, excelente costo y la atención al cliente del fabricante MyroPCB, esta vez en un diseño de 6 capas.

Esta primera implementación, ha permitido ver algunos aspectos a mejorar en el diseño y plantea algunas reformas para futuras versiones. Los principales problemas se discuten a continuación:

Memoria Flash: La memoria Flash *TC58FVM6B2A* seleccionada para este proyecto no presentó compatibilidad en los tiempos del protocolo de acceso. Se llegó a esta conclusión tras no obtener respuesta alguna al ejecutar algunos comandos internos como lectura y escritura de datos, lectura del ID del fabricante y de la memoria, aún cuando el driver

de esta memoria aparece listado en la aplicación de instalación del bootloader. Posteriormente se trabajó con la memoria *M29W320DT* , también de arquitectura AMD, compatible en su distribución de pines y comandos, pero su respuesta fue idéntica a la de la *TC58FVM6B2A* . A diferencia de las memorias flash Intel, en estas memorias el pin 0 del bus de direcciones, debe ser conectado al bus del procesador para que se puedan transmitir los códigos de los comandos de la memoria. La recomendación para solucionar este problema es emplear memorias como la J828F128J3C120 fabricada por Intel o una Micron-Q fabricada por Micron las cuales ya han sido empleadas en otros diseños como el CS-EP9301 y EDB930X respectivamente con buenos resultados. Es necesario rutear de nuevo esa sección del PCB dado que el empaquetado y la distribución de pines es diferente a la usada en el proyecto. Otra posible solución, que no obliga a fabricar nuevamente el PCB, es implementar en un pequeño PCB una memoria serial y conectarla al bus SPI por medio del conector destinado para ello. De esta manera, cambiando la configuración del bootloader, se puede instalar este programa en la memoria.

Pulsadores de Reset: Las señales de reset del procesador RST y PRST son de lógica negativa. Durante la captura del esquemático se implementaron estos pulsadores con lógica positiva, lo cual se corrigió en el PCB con un puente, pero debe ser corregido para versiones futuras de Phoenix.

Habilitador de la fuente del FPGA: El chip *TPS75003* , seleccionado como fuente de la sección del FPGA, posee un pin de habilitación de lógica alta controlado por el pin $EGPIO_3$ del procesador, además de tener una resistencia pull-up en caso de que la sección del procesador esté inhabilitada. En el esquemático, esta resistencia es llevada a la tensión 3,3V del procesador, por lo cual no puede cumplir su función. Es necesario llevarla a la tensión de entrada del regulador, de esta manera cuando la sección del procesador este inactiva, los pines de habilitación del *TPS75003* estarán a V_{IN}[1] y la fuente estará activa; cuando ambas secciones están activas el pin $EGPIO_3$ se encargará de imponer el nivel a los pines de habilitación.

[1] El regulador permite en este pin niveles de tensión desde $-0{,}3V$ hasta $V_{IN} + 0{,}3V$.

Fuente del FPGA: Debido a las dificultades que se tuvieron al soldar el chip *TPS75003* , se recomienda que en diseños posteriores, se acuda al servicio de ensamblaje que ofrecen algunos fabricantes de PCBs, o se emplee reguladores individuales para cada una de las tres tensiones que requiere el FPGA. Aunque la hoja de datos del FPGA exige ciertas condiciones en la forma como el FPGA se debe encender, principalmente en la rata de ascenso de las tensiones, es posible usar reguladores independientes siempre y cuando cumplan con los requisitos de corriente del FPGA.

Señal SSCLK del SPI: Según el diseño actual, para implementar un periférico SPI dentro del FPGA, es necesario que la señal SSCLK pase a través del buffer que permite la configuración desde el procesador (ver figura 2.14). Esto no sería ningún inconveniente si el diseñador planea configurar el FPGA desde el procesador, pero si el archivo de configuración se encuentra en el PC o en la memoria PROM, el usuario debe realizar un switch mecánico para configurar inicialmente el FPGA y luego para que pueda recibir los datos por el bus SPI. Para evitar este inconveniente, se puede emplear uno de los pines de entrada libres del FPGA para que actúe como entrada de reloj del bus SPI.

Led del FPGA: El led 4 es controlado directamente desde el FPGA por el pin 194 del FPGA, pero dado que este pin solo puede ser configurado como entrada, entonces este Led no puede encender. La solución a este problema es asignar otro pin que pueda ser configurado como salida y volver a rutear este trazo.

5.1. Posibilidades a futuro

Con el objetivo de apoyar y orientar los trabajos futuros que busquen complementar el presente proyecto, presentamos algunos puntos que surgen como resultado de la experiencia en la realización de este trabajo.

Sobre los empaquetados

Durante la definición de los requerimientos, el diseño de los esquemáticos y la selección de componentes se tuvo presente los empaquetados de los dispositivos. Debido a las dificultades que presenta el manejo de algunos de estos empaquetados como el BGA, se decidió no emplearlos. Esto limitó bastante la selección de componentes principalmente el FPGA y el procesador, donde solo los modelos mas bajos de las familias tienen empaquetados diferentes al BGA.

La recomendación para futuros trabajos es emplear estos empaquetados y si no es posible soldarlos con los equipos de la universidad, se puede acudir a servicios particulares de ensamblaje, por ejemplo MyroPCB[2].

Sobre el trabajo con el software para diseño de PCBs

El uso de software profesional para el manejo de un proyecto de mediana y gran magnitud es necesario, y más aun si el grupo de trabajo es numeroso. Software como el desarrollado por Mentor Graphics y Cadence permite mayor coordinación en el trabajo de grupo. Igualmente es necesario que los integrantes del grupo establezcan un conjunto de reglas internas que les permitan manejar un lenguaje común y facilitar el intercambio de información. Algunas de estas reglas pueden ser:

- Asignar sufijos y prefijos a los nombres de las señales y componentes del circuito, para resaltar algunas de sus características como si pertenecen a un par diferencial o a una sección del circuito.
- Determinar desde un comienzo la organización de los diagramas esquemáticos, definiendo el contenido de cada uno de ellos según las secciones del circuito. De esta manera es mas fácil la edición y la revisión de errores.
- Establecer una metodología que permita hacer copias de seguridad y organizar versiones

[2] US$0.0125 por pin de montaje superficial y US$0.025 por pin de hueco pasante.

del PCB de manera eficiente, posiblemente con ayuda de scripts o *software* especializado para el control de versiones.

Bibliografía

[1] Altera. *Guidelines for Designing High-Speed FPGA PCBs.* . 2004.

[2] Ken Arnold. *Embedded Controller Hardware Design.* . LLH Technology Publishing, 2000.

[3] Stuart R. Ball. *Embedded Microprocessor Systems - Real World Design.* . Newnes, 2 edition, 2000.

[4] Michael Barr. *Programming Embedded Systems in C and C++.* . O'Reilly, 1999.

[5] C. Coombs. *Printed circuits handbook.* . McGraw-Hill, 2001.

[6] Lewin A. R. W. Edwards. *Embedded System Design on a Shoestring.* . Newnes, 2003.

[7] Fairchild. *74LCX32 Low Voltage Quad 2-Input OR Gate with 5V Tolerant Inputs.* .

[8] Fairchild. *NC7SPU04 TinyLogic (C) ULP Unbuffered Inverter Datasheet.* .

[9] Fairchild. *NC7SZ14 TinyLogic (C) UHS Inverter with Schmitt Trigger Input.* .

[10] Fairchild. *NC7WZ126 TinyLogic (C) UHS Dual Buffer with 3-STATE Outputs.* .

[11] Craig Hollabaugh. *Embedded Linux®: Hardware, Software, and Interfacing.* . Addison Wesley, 2002.

[12] Texas Instruments. *MAX3232 3V to 5.5V Multichannel RS-232 Line Driver/Receiver.* .

[13] Texas Instruments. *SN74LVC245A Octal bus Transceiver with 3-state outputs.* .

[14] Texas Instruments. *Triple-Supply Power Management IC for Powering FPGAs and DSPs.* .

[15] Intel. *Intel LXT972A Single-Port 10/100 Mbps PHY Transceiver.* .

[16] IPC. *IPC-2221 Generic Stndard on Printed Board Design.* . 1998.

[17] Melanie Thelen Jürgen Sauermann. *Realtime Operating Systems.* .

[18] Cirrus Logic. *EP9301 User's guide.* .

[19] Cirrus Logic. *EP9302 Data Sheet.* .

[20] Cirrus Logic. *EP93xx RTC Oscillator Circuit.* .

[21] Cirrus Logic. *EP93xx Silicon Rev E Design Guidelines.* .

[22] Peter Marwedel. *Embedded System Design.* . Springer, 2006.

[23] Anthony J. Massa. *Embedded Software Developmant with eCOS.* . Prentice Hall, 2003.

[24] C. Maxfield. *The design warrior's Guide to FPGAs.* . Newnes, 2004.

[25] Micron. *Synchronous DRAM MT48LC4M16A2 1 Meg x 16 x 4 banks.* .

[26] M. Montrose. *Right angle corners on printed circuit board traces, time and frequency domain analysis.* . 2000.

[27] Bart Broekman Edwin Notenboom. *Testing Embedded Software.* . Addison-Wesley, 2003.

[28] Michael J. Pont. *Embedded C.* . Addison-Wesley, 2002.

[29] Vishay Semiconductors. *Serial Infrared Transceier SIR, 115.2 kbit/s 2.7V to 5.5V Operation.* .

[30] ST. *LD1117 Low drop fixed and adjustable positive voltage regulators.* .

[31] ST. *USBDFxxW5 EMI filter and line termination for USB downstream ports.* .

[32] S. Thierauf. *High-Speed Circuit Board Signal Integrity.* . Artech House, Inc, 2004.

[33] Toshiba. *Toshiba TC58FVM6(T/B)2A(FT/XB)65 Flash memory.* .

[34] Xilinx. *Platform Flash In-System Programmable Configuration PROMS.* .

[35] Xilinx. *Spartan-3E FPGA Family. The Complete Data Sheet.* .

[36] H. Johnson y M. Graham. *High-Speed Digital Design. A handbook of Black Magic.* . Prentice Hall, 1993.

[37] W. Salamanca y O. Carrillo. *Guia de usuario de Phoenix.* . UIS, 2008.

[38] M. Erazo y S. Banguero. *Plataforma de desarrollo para sistemas embebidos basada en el game boy advance.* . UIS, 2007.

[39] Ahmed Anime Jerraya y Sungjoo Yoo y Diederik Verkest y Norbert Wehn. *Embedded Software for SoC.* . Kluwer Academic Publishers, 2004.

[40] Qing Li y Carolyn Yao. *Real-Time Concepts For Embedded Systems.* . CMP Books, 2003.