VNIVERSIDAD NACIONAL AVTENOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ACTUALIZACIÓN DEL HARDWARE DEL SUBSISTEMA DE CONTROL DE ORIENTACIÓN DEL SATÉLITE EDUCATIVO SATEDU

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO ELÉCTRICO ELECTRÓNICO

P R E S E N T A :

VIZCAÍNO TORRES EDUARDO

DIRECTOR:

DR. ESAÚ VICENTE VIVAS

México D.F. 2012

Agradecimientos

A Dios por haberme permitido llegar hasta donde hoy he llegado, y por haberme dado la fuerza para romper todos los obstáculos que se me han presentado en camino.

A la Universidad Nacional Autónoma de México y a la facultad de Ingeniería, por haberme permitido ser parte de este recinto que es la máxima casa de estudios en México, donde me formé como profesionista y como una persona íntegra.

A mi familia por sus consejos, su interés y su apoyo

A mis padres por el gran apoyo brindado con muchísimo amor y esfuerzo, ya que sin él no podría haber llegado hasta donde hoy en día me encuentro. Por todos los consejos, la confianza y el interés prestado en mí y por el gran patrimonio que un padre pudiera heredar a su hijo, que es una buena preparación. Por haberme enseñado a luchar por mis ideales con fuerza, coraje, conciencia, responsabilidad y amor, y por todas esas grandes cosas que solo ellos pudieron haberme entregado. A mi padre por haberme enseñado que hasta en las cosas, por más insignificantes que pudieran parecer, es posible encontrar la felicidad y por enseñarme a ser fuerte y de gran carácter. A mi madre, por ser mi ejemplo a seguir, por enseñarme a imprimir el máximo en todos y cada uno de los proyectos de mi vida, y sobre todo, por demostrarme que en los momentos más adversos de la vida se puede salir a delante, ya que no existen imposibles. LOS AMO PADRES.

A mis hermanos, por el apoyo moral que me han brindado, por todos esos viernes de fiestas VIP muy relajantes después de una semana pesada en la universidad (en esta parte se incluye Julio ya que al igual él ha compartido esos tan esperados viernes), y sobre todo el gran cariño de hermanos que siempre nos hemos tenido a pesar de las diferencias y las peleas.

A Rita Miranda Zenón, por su confianza y apoyo incondicional en esta travesía, por esos momentos de relajación y diversión en los momentos más difíciles, y por todo ese cariño con el que actuó hacia mí. Por creer en mí, por todas esas palabras de aliento y sobre todo, por ser esa gran compañera con la cual he recorrido gran parte de mi vida.

A mis amigos CCH-eros, por todos los momentos de relax que me hacían olvidar por momentos las presiones de la carrera.

A mis amigos de la facultad de Ingeniería Mayra, Alejandro, Javier, Luis, Emma, Citlalli, Josymar, y por toda la demás banda de ingeniería con la cuál compartí clases (el estrés), proyectos (más estrés jajaja…) y por todos esos momentos los cuales hicieron amena mi estancia en la carrera.

A mis compañeros de trabajo y amigos del Instituto de ingeniería Mario, Emilio, Rodrigo, Ignacio, Enkor, Alejandro, Miguel y todo el grupo de trabajo de tecnología satelital con los cuales compartí una etapa nueva y fenomenal en mi vida. Sobre todo a Mario e Ignacio los cuales me asesoraron y ayudaron en todo el proceso de desarrollo de esta tesis.

A mi tutor Esaú Vicente Vivas, por hacerme permitido ser parte de un grupo de trabajo dentro del instituto de ingeniería donde encontré mi verdadera pasión y aumenté exponencialmente mis conocimientos.

**El que posee las nociones más exactas sobre las causas de las cosas y es capaz de dar perfecta cuenta de ellas en su enseñanza, es más sabio que todos los demás en cualquier otra ciencia.*
**La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica.*
Aristóteles

**Es mejor saber después de haber pensado y discutido que aceptar los saberes que nadie discute para no tener que pensar.*
Fernando Savater

ÍNDICE

PRÓLOGO

La inclusión de novedosas herramientas tecnológicas es cada vez más habitual en los ámbitos académicos y de desarrollo. El valor que tienen las nuevas tecnologías en relación con la enseñanza resulta indiscutible y su incorporación en el proceso enseñanza-aprendizaje se hace cada vez más indispensable, principalmente en el momento de generar propuestas innovadoras orientadas principalmente a difundir y mejorar el desarrollo de la tecnología.

El uso de herramientas de aprendizaje se vuelve particularmente importante en un área relativamente joven como la tecnología espacial. Sin lugar a duda, en los últimos cuarenta años este desarrollo incipiente de la tecnología espacial ha tenido un incremento impresionante principalmente en su nivel de complejidad y sofisticación. Especialmente en el campo satelital, las misiones actuales demandan sistemas que cuenten con un mayor nivel de robustez y confiabilidad, en virtud de prolongar su tiempo de vida una vez que son puestos en órbita. De igual forma, los campos técnicos en los cuales son utilizados los satélites son cada vez más numerosos: telecomunicaciones, percepción remota, validación tecnológica, investigación, etc. De acuerdo con la aplicación, un satélite es totalmente diferente a otro, y de igual forma sus requerimientos técnicos y constructivos.

En el campo del uso de herramientas de aprendizaje en el campo satelital, a nivel mundial existen diversas plataformas que permiten la experimentación sobre tecnología espacial. Muchos de ellos, creados principalmente por universidades y centros de investigación para propósitos específicos de desarrollo tecnológico. Comercialmente existe también un producto que permite además de la experimentación y el desarrollo en el área satelital, la capacitación en ésta área de desarrollo científico.

Basados en el paradigma de crear una herramienta de aprendizaje de múltiple propósito, que permita el desarrollo, evaluación y entrenamiento e inducción de recursos humanos en el área satelital, surge SATEDU en el año 2008.

SATEDU es un sistema de entrenamiento totalmente diseñado, desarrollado y validado en el Instituto de Ingeniería de la UNAM. Es un sistema formado por tarjetas electrónicas que integran un modelo de satélite de laboratorio. Cuenta con todos los recursos que integra un satélite real pequeño, lo que lo convierte en una potente herramienta de enseñanza para el campo de desarrollo tecnológico espacial.

En los últimos años, se han realizado esfuerzos por actualizar parte de los subsistemas de SATEDU. Uno de ellos, el subsistema de estabilización y control de apuntamiento por medio de ruedas inerciales. Este subsistema, dentro de una plataforma satelital de vuelo, es de vital importancia, ya que permite el apuntamiento específico del satélite hacia objetivos específicos. En el caso de SATEDU, su sistema de control actual, permite la realización de maniobras básicas en un eje, aunque cuenta con los recursos en hardware para la realización de pruebas en tres ejes, utilizando ruedas inerciales y bobinas de torque magnético como actuadores activos.

Sin embargo, el avance de la tecnología particularmente en dispositivos programables, hacen atractiva la propuesta de evolucionar el módulo de cómputo a bordo del subsistema de control de orientación de SATEDU hacia una plataforma mayormente versátil y

totalmente flexible, en comparación con la que tenía originalmente. Lo cual permitirá la realización de pruebas cada vez más elaboradas y la evolución de esquemas de control.

En particular, en esta tesis se propuso diseñar y construir un nuevo subsistema de control de orientación para SATEDU, utilizando un sistema embebido basado en un dispositivo de tecnología FPGA. Esta elección se respalda por el hecho de aprovechar las ventajas del dispositivo en términos de la construcción de sistemas totalmente a la medida. En procesamiento, los FPGAs permiten construir arquitecturas de cómputo que aprovechan al máximo los recursos para la ejecución de operaciones en paralelo, y también cuentan con la posibilidad de implementar sistemas que sean reconfigurables totalmente de forma remota. Esta es una característica altamente útil en pruebas de validación de esquemas de control de orientación en un simulador de vuelo satelital, que base su operación en una mesa suspendida en aire, tal como la que se usa actualmente en el Instituto de Ingeniería de la UNAM.

La propuesta plantea el desarrollo del subsistema de control de orientación a nivel de hardware y software.

INTRODUCCIÓN

En el presente trabajo de tesis se describe el análisis, diseño y validación de hardware de procesamiento en tiempo real basado en una plataforma reconfigurable FPGA (sistema mínimo), el cual es la base para el diseño y desarrollo del nuevo hardware del subsistema de control de orientación en tres ejes del satélite educativo SATEDU el cual residirá en una de sus tarjetas electrónicas. Dentro de las ventajas que ofrecerá esta nueva versión de subsistema satelital, se incluye la implementación de sistemas embebidos flexibles basados en núcleos de cómputo tanto personalizados como de pago, así como la capacidad para realizar la reconfiguración remota del FPGA utilizando un microcontrolador de gestión y control de carga.

El esquema general que plantea la tesis es el diseño y la integración de una tarjeta electrónica de diseño propio, la cual integra módulos de potencia, comunicaciones, reconfiguración, entradas y salidas digitales, así como componentes de soporte. El elemento central de gestión y procesamiento de datos, es un dispositivo lógico programable FPGA XC3S100E de la gama baja de la familia Spartan 3E de Xilinx, en el cual se implementarán arquitecturas de control sencillas descritas en VHDL.

El trabajo se encuentra dividido en 8 capítulos.

En el primer capítulo se presenta una revisión histórica sobre la incursión de México en el desarrollo de tecnología satelital hasta llegar a la creación del satélite educativo SATEDU y la actual participación de construcción de un Nano-satélite para la constelación internacional HUMSAT.

En el segundo capítulo se abordan las generalidades de los dispositivos FPGA, describiendo su arquitectura y los recursos internos con los cuales cuenta, al igual que el proceso de diseño y la creación de arquitecturas de cómputo para ellos.

En el tercer capítulo se describe la tarjeta de desarrollo Spartan 3E-Starter Kit, dando un preámbulo a los diferentes modos de configuración de FPGAs de la familia Spartan, enfocándose sólo en aquellos que maneja la tarjeta de desarrollo.

En el cuarto capítulo se analizan cuatro esquemas propuestos para llevar a cabo el proceso de reconfiguración remota de un FPGA de la familia Spartan 3E, explicando las ventajas y desventajas que tienen cada uno de ellos y la definición del esquema que integra la tarjeta de sistema mínimo.

En el quinto capítulo se presenta el esquema completo de la tarjeta de sistema mínimo, el cual integra el esquema de reconfiguración previamente analizado y seleccionado.

En el sexto capítulo se realiza una descripción detallada de cada una de las herramientas en hardware y software, utilizadas para llevar a cabo una serie de pruebas, que permitieron definir la lógica de control para la reconfiguración remota del FPGA.

En el séptimo capítulo se presentan las pruebas experimentales que se desarrollaron para la validación de la tarjeta de sistema mínimo.

En el octavo capítulo se describe el diseño de la nueva tarjeta del subsistema de control de orientación en tres ejes del satélite SATEDU.

En el noveno capítulo se presentan las conclusiones obtenidas del proyecto, que lleven a la generación de observaciones y correcciones que realizar en versiones futuras del desarrollo de este proyecto, junto con una serie de recomendaciones para futuros desarrollos o mejoras sobre la base del sistema presentado.

En la sección de bibliografía se indican las diferentes fuentes (libros, manuales, páginas de internet, notas de aplicación) que fueron consultados durante el desarrollo de este trabajo de tesis.

Adicionalmente se presenta una sección de apéndices, en la cual se podrán consultar extractos de código fuente utilizados.

CAPÍTULO 1

Introducción a los medios de instrumentación y automatización en sistemas espaciales, con énfasis en proyectos nacionales

En los últimos años la tecnología espacial ha recobrado gran importancia en numerosos ámbitos de las actividades de la sociedad en diversas naciones, principalmente en el rubro de las telecomunicaciones. México no ha sido la excepción dentro de esta marcha mundial; si bien los esfuerzos de la incipiente tecnología e industria espacial mexicana han logrado posicionar a nuestro país como un importante manufacturador de la industria aeroespacial a nivel internacional, también se han venido realizando esfuerzos por posicionar al país como un semillero de desarrollos tecnológicos e innovaciones en el área, que datan desde la década de los años 40’s.

En este capítulo se mencionan algunos de los esfuerzos realizados en México en materia de desarrollo de sistemas espaciales y que muestran el deseo constante de ubicar a nuestro país como uno de los referentes en el desarrollo de ciencia y tecnología en el área.

1.1 SATEX

El proyecto SATEX se plantea en el marco de los programas de transferencia de tecnología que existieron con la compañía lanzadora *Ariane Space*, así como con *Hughes*, ésta última fabricante de los satélites mexicanos "Morelos" y "Solidaridad". En 1994 el proyecto SATEX se pone en marcha como un proyecto multi-institucional en el que participan las instituciones públicas de mayor prestigio en México, entre las cuales se encuentra la Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME) del Instituto Politécnico Nacional (IPN), el Instituto de Ingeniería de la Universidad Nacional Autónoma de México (UNAM), el Instituto de Geografía de la UNAM, el Centro de Investigación y Desarrollo de Tecnología Digital (CITEDI), el Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), el Centro de Investigaciones Matemáticas (CIMAT), el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), entre otras, figura 1.1.

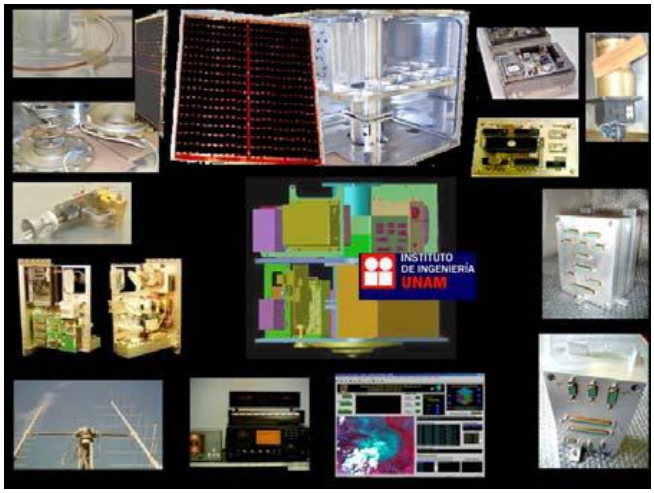


Figura 1.1. Productos desarrollados para el proyecto SATEX.

1.1.1 Objetivo

El objetivo de este proyecto fue diseñar, construir y validar un Microsatélite experimental de bajo costo, de 50 Kg de masa y 50 cm por lado; un Microsatélite de órbita baja (entre 400 y 800 Km), con estabilización en 3 ejes utilizando medios pasivos (gradiente gravitacional) y medios activos (bobinas de torque magnético). La carga útil de este satélite sería una cámara que permitiría validar mediante la captura de fotografías la funcionalidad del satélite, en tanto que la transferencia de datos se realizaría con una estación terrena por enlaces de VHF y UHF que oscilan entre los 150 y 400MHz. La vida útil aproximada del satélite sería aproximadamente de dos años, llevando además un mástil en la parte superior (gradiente gravitacional), que ya en despliegue orbital se extendería hasta seis metros y

tendría un peso al extremo de dos kilos y medio aproximadamente, lo cual mantendría estabilizado al satélite.

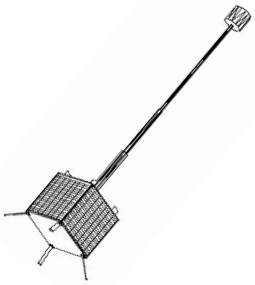


Figura 1.2. Esquema artístico del satélite SATEX.

Cada institución participó en el diseño y desarrollo de subsistemas para la plataforma satelital. Como se puede observar en la figura 1.3, la labor sinérgica de las instituciones permitiría la construcción de una misión satelital de gran relevancia para el país.

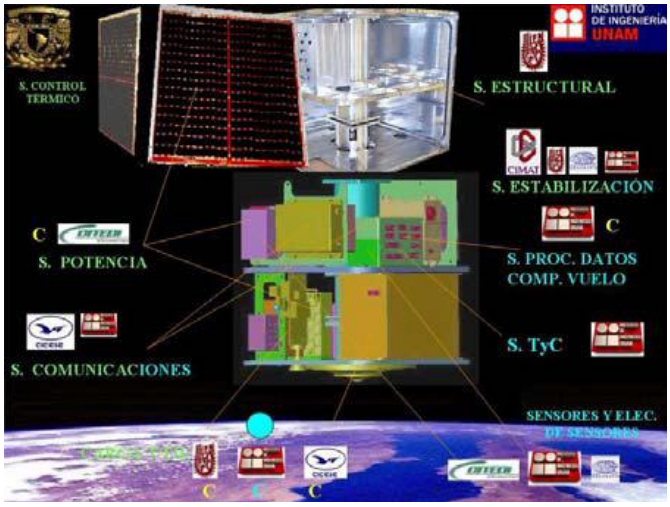


Figura 1.3. Participación de las instituciones en cada subsistema.

1.1.2 Arquitectura de SATEX

Estructura

Para la estructura se utilizó aluminio AL 7075T6, la cual integra un cilindro central que sostiene tres bandejas intermedias, como se muestra en la figura 1.4. Para validar la estructura se realizaron análisis y simulaciones en computadora sobre los esfuerzos mecánicos y carga contenida en la estructura.

Figura 1.4. Estructura de aluminio de SATEX.

Computadora de vuelo

El diseño de la computadora de vuelo se basó en la arquitectura de un sistema redundante, utilizando para su implementación un microcontrolador RISC SAB80C166, diseñado bajo el esquema de una arquitectura de cómputo tolerante a fallas y con un arreglo de procesadores físicos y virtuales. La computadora de vuelo se encargó de ejecutar los algoritmos de estabilización, la adquisición de datos de sensores, el control de la activación de los experimentos, la gestión de la red de datos interna del satélite, el establecimiento y control de la comunicación con estación terrena para subir nuevo programa y la recepción de comandos, así como la generación y envío de señales de telemetría a estación terrena, junto con otras tareas adicionales.

El desarrollo e implementación de los algoritmos de estabilización para el satélite se basó en un análisis matemático. Contemplaba el uso de bobinas de torque magnético y un gradiente gravitacional como elementos actuadores. Los algoritmos se validaron operativamente, mediante simulaciones digitales, las cuales incluyeron todas las etapas de operación en órbita. Adicionalmente, se pretendió emplear una mesa suspendida instrumentada electrónicamente para validar los algoritmos de control y para recrear la condición de cero fricción presente en el espacio, la cual fue diseñada en el instituto de ingeniería de la UNAM.

Subsistema de potencia

El subsistema de potencia contenía dos bancos de baterías recargables, un microcontrolador 87C51, convertidores DC-DC y una red interna de datos que permitía a la computadora de vuelo enviar los comandos necesarios para el control y activación de los diferentes subsistemas y equipos que alimentaba. El microcontrolador permitía conocer el estado de carga de las baterías y poner el satélite en modo de bajo consumo, de ser necesario, al igual que regresarlo a su estado de consumo normal en el momento que las baterías se encontraran cargadas.

Subsistema de comandos y telemetría

El subsistema de comandos y telemetría fue diseñado con el propósito de monitorear y controlar de forma remota los comandos y las señales de telemetría del satélite, por medio de un enlace de comunicación digital bidireccional entre el satélite y la estación terrena. Este enlace permitiría cargar la información de los parámetros operativos en el satélite, así como leer y hacer mediciones de los experimentos de la carga útil. El equipo utilizado dentro de este subsistema, consideraba el uso de un esquema de radio redundante para transmisión y recepción, para el satélite y para la estación terrena, con un ancho de banda de 148 a 400MHz. Este subsistema contenía además un sistema computacional mínimo (MCS), que actuaba como el último recurso en el caso que ambos equipos fallaran, manteniendo la comunicación en un modo de supervivencia con el satélite. Esto permitía al sistema mantener en operación las funciones vitales del satélite en caso de una falla grande o fatal en la computadora de vuelo.

Este satélite permitiría llevar a bordo tres cargas útiles:

- Un receptor en banda K, para estudiar fenómenos de atenuación atmosférica.
- Una cámara que obtenía imágenes de la tierra en espectro visible.
- Un receptor óptico para un enlace experimental desde la estación terrena hasta el satélite.

1.1.3 Resultados del proyecto

Diversos factores impidieron terminar este proyecto, uno de ellos, quizá el más importante, fue que se contó con un bajo monto de financiamiento. En 10 años las instituciones recibieron cerca de 10 millones de pesos, un financiamiento significativamente menor al requerido para este tipo de proyectos en otros países. La figura 1.5 muestra una comparativa del monto aproximado en millones de dólares que se requiere para el desarrollo de satélites de diversa escala ante el presupuesto considerado para el proyecto SATEX.

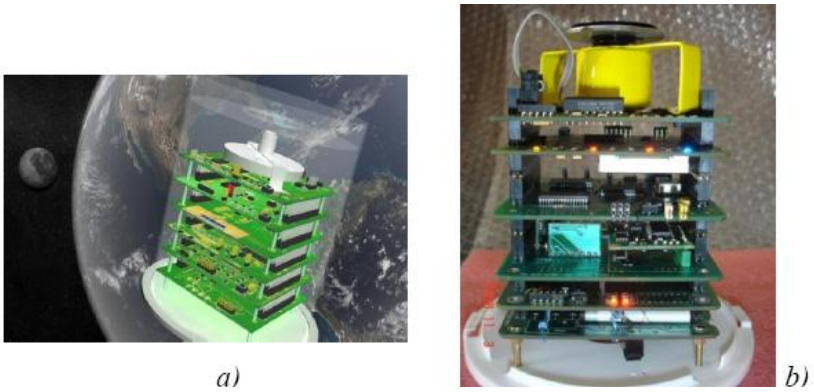
Figura 1.5. Costo del desarrollo de satélites en dólares.

1.2 Satélite educativo (SATEDU)

Tras la gran experiencia obtenida en la participación del proyecto SATEX y considerando el amplio nicho de oportunidades en el desarrollo de sistemas espaciales en México, el grupo de desarrollo de sistemas aeroespaciales del Instituto de ingeniería de la UNAM, tomó la iniciativa y puso en marcha en 2007 el diseño y construcción de un satélite de laboratorio.

La idea inicial de este satélite de laboratorio, fue el contar con una plataforma satelital con fines didácticos, de enseñanza y de desarrollo de tecnología satelital, atendiendo a la gran necesidad de transmitir de una forma sencilla y clara, los principales conceptos de la tecnología satelital a alumnos de niveles desde medio superior hasta posgrados. De esta manera surge SATEDU, acrónimo de SATélite EDUcativo, el cual es una plataforma de enseñanza para capacitación en tecnología espacial, única en su género en México y en el mundo.

Este satélite fue diseñado, construido y validado totalmente por un grupo multidisciplinario de estudiantes y académicos en el Instituto de ingeniería de la UNAM. La figura 1.6 muestra a SATEDU.



a) b)

Figura 1.6. a) Diseño virtual en 3D de SATEDU, b) SATEDU.

1.2.1 Objetivo

El objetivo que persigue SATEDU es ser empleado como herramienta de enseñanza en instituciones de diferentes niveles académicos, desde educación media superior hasta posgrados, así como centros de investigación y desarrollo. Esto permitirá introducir y entrenar a las nuevas generaciones dentro del mundo de la ciencia y la tecnología espacial, también para conocer, desarrollar y validar diversos subsistemas satelitales, así como evaluar nuevas tecnologías para su posterior transferencia hacia pequeños satélites reales.

1.2.2 Arquitectura de SATEDU

Los subsistemas de SATEDU se diseñaron conforme al estándar CubeSat. Las tarjetas electrónicas que integran a cada subsistema tienen una dimensión de 9 x 9 cm, formando un módulo que tiene las dimensiones de un contenedor de CDs, con un peso aproximado a 1 kg, como se muestra en la figura 1.7. El sistema SATEDU cuenta con todos los subsistemas que integran a un satélite pequeño real de vuelo espacial, incluso integra esquemas de protección contra el efecto *latch-up*, que se genera en ambiente espacial.

SATEDU constituye una arquitectura de cómputo distribuido, en la cual cada tarjeta electrónica incluye un microcontrolador que se encarga de interactuar con la computadora de vuelo para administración de tareas, suministro de energía y comunicación los demás subsistemas.

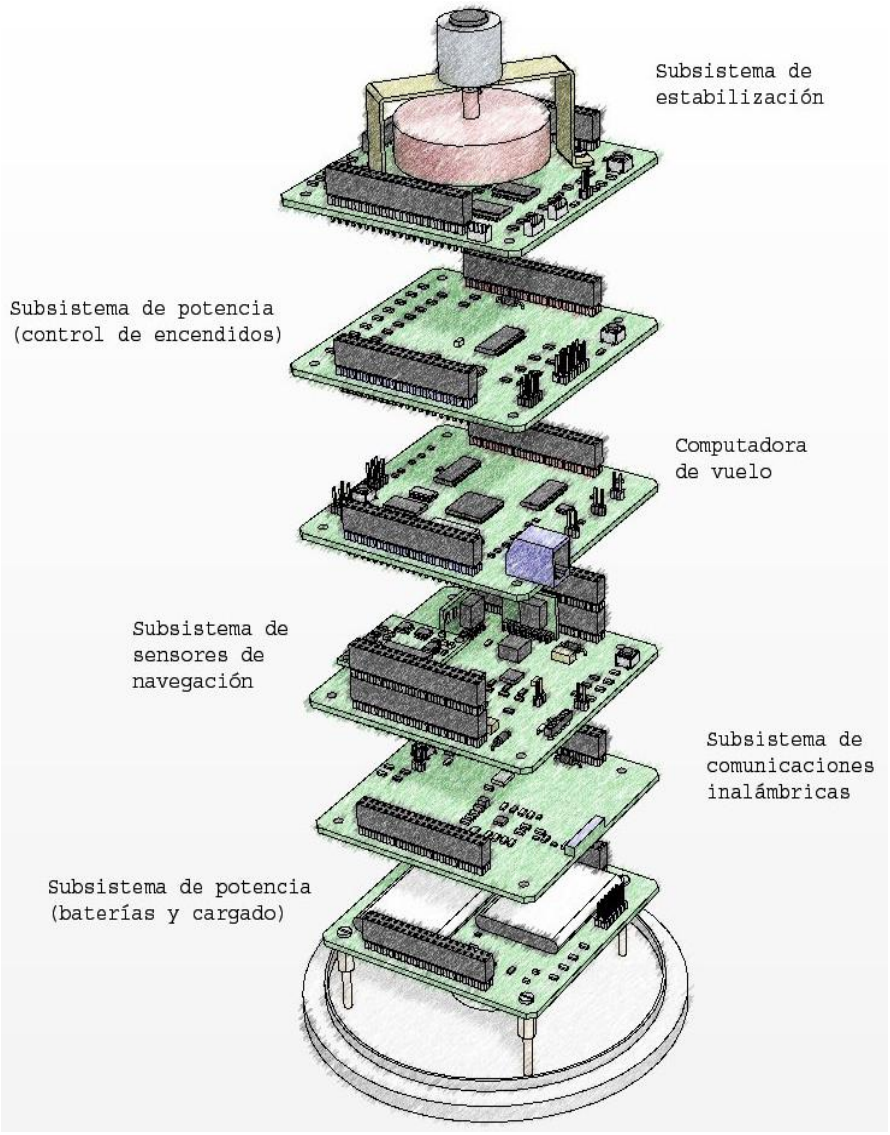


Figura 1.7. Arquitectura de SATEDU.

Subsistema de computadora de vuelo (CV)

La Computadora de vuelo es una tarjeta electrónica que cuenta con un microprocesador SAB80C166 de Siemens, el cual es el encargado de coordinar las tareas con los demás subsistemas del satélite. Cuenta también con un microcontrolador PIC16F876A, una Memoria Flash de 32MB, 3 Sensores de Temperatura y 3 Arreglos de protección contra efecto *latch-up.* La figura 1.8 muestra la tarjeta diseñada para este subsistema.

Figura 1.8. Computadora de vuelo.

Subsistema de Potencia (SP)

El Subsistema de potencia reside en 2 tarjetas, una de ellas contiene toda la lógica electrónica la cual integra un PIC18F2321 (dispositivo encargado de comunicarse con la CV para controlar el apagado y encendido de las fuentes de energía de los demás subsistemas del satélite); un convertidor de DC-DC que provee energía para controlar el motor de la rueda inercial, así como una serie de reguladores de voltaje para obtener los voltajes requeridos por los subsistemas satelitales. La segunda tarjeta está constituida por un banco de 4 baterías de Litio-ion, así como circuitos electrónicos de carga. La figura 1.9 muestra las tarjetas diseñadas para este subsistema.

Figura 1.9. Subsistema de Potencia.

Subsistema de estabilización (SE)

Está integrado en una tarjeta que maneja dos métodos de estabilización activa para el satélite: rueda inercial y bobinas de torque magnético. La rueda inercial es una pequeña masa sujeta a un motor, mientras que las bobinas de torque magnético son embobinados,

uno en cada eje, para que en conjunto con la rueda inercial realicen maniobras de estabilización de SATEDU. La figura 1.10 muestra la tarjeta diseñada para este subsistema.

Figura 1.10. Subsistema de Estabilización

Subsistema de comunicaciones (SC)

El subsistema consiste de 2 tarjetas que constituyen dos módulos de comunicación bidireccional utilizando circuitos integrados de RF como medio de enlace, para la transferencia de datos de telemetría y comandos entre la estación terrena y el SATEDU. Una tarjeta está conectada a la PC que funciona como estación terrena, y la otra está integrada en SATEDU. La figura 1.11 muestra la tarjeta diseñada para este subsistema.

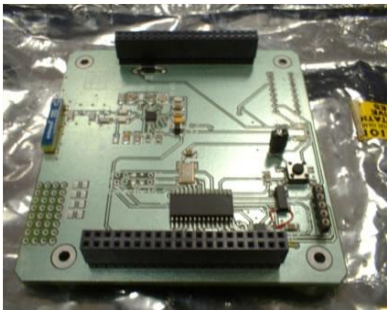


Figura 1.11. Subsistema de Comunicaciones de SATEDU.

Subsistema de sensores de estabilización (SSE)

El SSE es una tarjeta que contiene los sensores necesarios para monitorear el movimiento de SATEDU y algunos de ellos sirven para visualizar su posición en tiempo real en una computadora. Los sensores que la componen son una brújula electrónica, un acelerómetro triaxial y tres giróscopos electrónicos colocados de manera ortogonal para monitorear los tres ejes del satélite. La figura 1.12 muestra la tarjeta fabricada para este subsistema.

Figura 1.12. Subsistema de Sensores de Estabilización

1.2.3 Aplicaciones

SATEDU es una potente plataforma de enseñanza y desarrollo de tecnología satelital, que permite acercar a estudiantes desde nivel medio superior hasta posgrados al campo de desarrollo tecnológico satelital. La flexibilidad de la arquitectura de hardware de SATEDU, permite con relativa facilidad la integración de nuevas tarjetas electrónicas, por lo que además de representar una herramienta de capacitación y enseñanza, se torna además en una potente plataforma de desarrollo y evaluación de subsistemas satelitales en Tierra.

Recientemente se terminó exitosamente la integración de una nueva tarjeta de comunicaciones inalámbricas utilizando el protocolo Bluetooth. De igual forma, se trabaja actualmente en el desarrollo e integración de una carga útil con aplicación en telemedicina.

1.3 HumSat

HumSat es una iniciativa internacional educativa para la construcción de una constelación internacional de nano-satélites que proporcione capacidad de comunicación en áreas con poca infraestructura. En este proyecto participan organizaciones internacionales como la Agencia Espacial Europea (*European Space Agency (ESA)*), la Oficina de Naciones Unidas para Asuntos del Espacio Ultraterrestre (*United Nations Office for Outer Space Affairs (UNOOSA)*), la Administración Nacional de Aeronáutica y del Espacio (*National Aeronautics and Space Administration (NASA)*), y la Federación Internacional de Astronáutica (*International Astronautical Federation (IAF)*). También se cuenta con la participación universidades como la Universidad de Vigo en España (líder del proyecto), la Universidad Politécnica del estado de California en Estados Unidos (CalPoly), la Universidad Nacional Autónoma de México y el Centro Regional de Enseñanza en Ciencia y Tecnología del Espacio para América Latina y el Caribe *(CRECTEALC)*, siendo la *ESA* el gestor del proyecto.

1.3.1 Objetivo

El objetivo principal de HumSat es desarrollar una constelación de nano-satélites para enlazar a un conjunto de usuarios con una red de sensores distribuidos en todo el mundo. Los sensores serán los responsables de la adquisición de los datos del usuario y de transmitir estos datos a los satélites de la constelación a través de una interfaz de radiocomunicaciones. Los usuarios serán capaces de definir sus propios sensores, para monitorear diferentes tipos de parámetros, como por ejemplo la contaminación del agua o datos de operación de redes de petróleo, energía, etc.

La red de estación terrena GENSO será uno de los componentes principales del sistema de distribución de datos y tendrá como tarea recuperar los datos de los satélites. Una vez que los datos hayan sido transmitidos por los satélites HumSat los usuarios autorizados podrán acceder a estos a través de una conexión de internet, de acuerdo con el esquema conceptual del proyecto que se muestra en la figura 1.13.

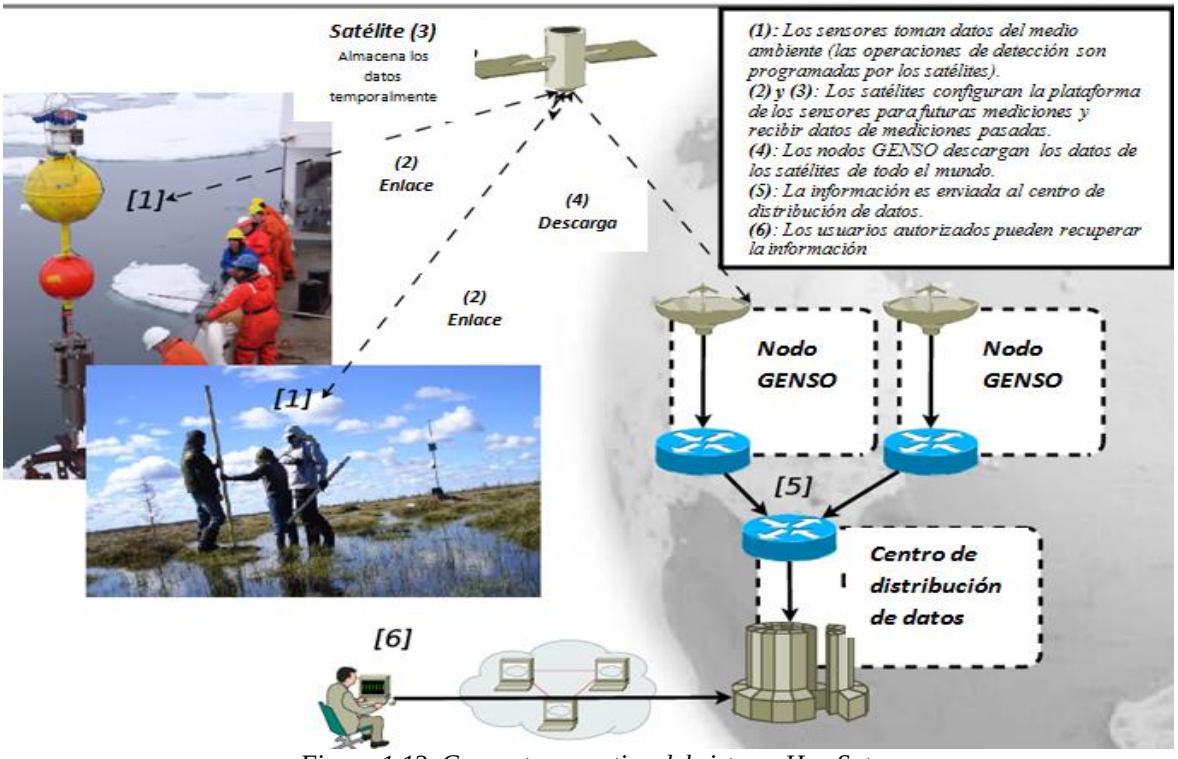


Figura 1.13. Concepto operativo del sistema HumSat.

Este proyecto tiene otros objetivos fuertes en la rama de la educación como lo son:

- Proporcionar experiencia práctica en un proyecto espacial a estudiantes de ingeniería y ciencias.

- Hacer que los estudiantes se familiaricen con las normas y procedimientos para los proyectos espaciales.
- Promover la cooperación internacional entre universidades, instituciones y agencias espaciales a nivel mundial sobre la tecnología espacial.
- Transferencia de tecnología con universidades de países en desarrollo.

1.3.2 Arquitectura de la constelación HumSat

El proyecto HumSat actualmente se encuentra en fase de desarrollo, por lo que no está definida una arquitectura final, sin embargo las primeras propuestas del proyecto consideran las características mencionadas a continuación.

- Los satélites se podrán basar en las normas del satélite CubeSat aproximadamente de 1 a 4.5 Kg, pero se encuentra abierto a otras normas.
- Diseñados para órbita baja, aproximadamente 600 Km.
- Estación terrena de red mundial GENSO para descarga de datos y TTC, proporcionando una visibilidad casi continua de los SATÉLITES, como se muestra en la figura 1.14.
- Uso de nano y micro tecnología para sensores.
- Uso de sistemas como *Argos* y *WEOS* para contrarrestar el efecto *Doppler*.
- Posibilidad de incluir mecanismos pasivos para la distribución optima de los satélites.
- Posibilidad de integrar nuevos satélites HumSat para mejorar el rendimiento general del sistema.
- Posibilidad de incluir carga útil primaria o secundaria en cualquier CubeSat, NanoSat o MicroSat.
- Cada usuario podrá definir sus propios sensores y lugares de monitoreo.
- Una vez que un sensor ha transmitido sus datos, estos datos se descargan a través de una de las estaciones del GENSO y se transmiten al centro de distribución de datos a través de Internet.
- Los usuarios tendrán que registrarse, con el fin de acceder a sus datos.
- El acceso a estos datos se realizará a través de Internet.

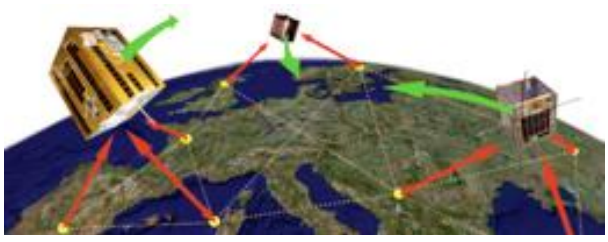


Figura 1.14. Simulación de la constelación HumSat.

1.3.3 Aplicaciones

La constelación HumSat proporcionará un servicio de comunicaciones genérico conocido como de almacenamiento y reenvío, en el cual se espera que múltiples usuarios del sistema puedan construir y definir sus propias aplicaciones. Sin embargo, se puede identificar un conjunto básico de aplicaciones, para las cuales se espera que HumSat preste sus servicios, como:

- Captura de datos de mediciones de cambio climático.
- Apoyo en comunicaciones a una baja tasa de transmisión en aéreas de poca infraestructura, zonas de baja población, etc.
- Prestar apoyo en situaciones de emergencia (desastres naturales, accidentes, etc.).

1.3.4 HumSat-México

México participará dentro del proyecto HumSat al menos con dos satélites para la constelación, uno de ellos del Instituto de ingeniería de la UNAM el cual se encuentra en fase de diseño y desarrollo, que se llamará satélite HumSat-México, cimentándose en la arquitectura de SATEDU. Esto es posible gracias a que SATEDU fue diseñado con el objetivo de tener casi todos los componentes en hardware y software que existirían en un satélite con calificación de vuelo, lo que permitirá volcar la tecnología al nanosatélite, previas adecuaciones para un modelo de vuelo espacial. HumSat-México está planeado con un peso de 3 kg, llevando con él cargas útiles de percepción remota, estabilización en tres ejes, modem de tecnología software radio y comunicación de bajada en banda S.

Una gran aportación tecnológica que se dará para este satélite es que contará con un sistema de control de estabilización y apuntamiento en tres ejes, materia que sólo algunos países a nivel mundial dominan e integran en sus plataformas satelitales.

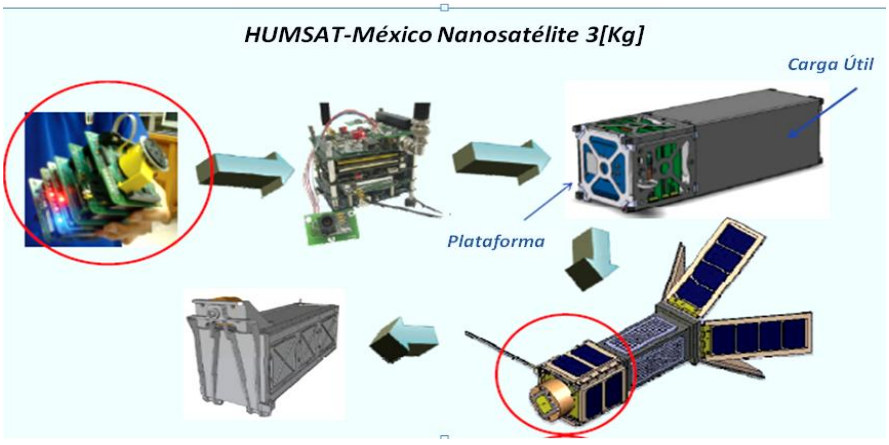


Figura 1.15 Sateélite HumSat-México del Instituto de Ingeniería, UNAM.

Hasta aquí se ha mostrado una breve pero sustanciosa panorámica de los trabajos sobre tecnología satelital realizados en México. Como pudo observarse, algunos de ellos, no llegaron a una completa concreción por diversos factores, sin embargo sentaron las bases para nuevos proyectos, actualmente terminados, como el caso de SATEDU.

Sin embargo, como se comentó, SATEDU ofrece una arquitectura flexible que permite la integración de nuevos subsistemas y actualizaciones. Como es el caso que ocupa a este trabajo de tesis, la cual consiste en actualizar la nueva tarjeta del subsistema de control de orientación, basado en una plataforma FPGA, que permitirá realizar maniobras de control en tres ejes.

En el siguiente capítulo se presentan las bases de la tecnología FPGA, la cual se empleó en la actualización del módulo de cómputo a bordo de la tarjeta del subsistema de control de orientación de SATEDU.

CAPÍTULO 2

Introducción a los FPGAs y al cómputo reconfigurable

El rápido crecimiento de capacidad tanto de los dispositivos lógicos programables, como de los circuitos de aplicación específica (ASIC), hacen posible el desarrollo de sistemas de cada vez mayor complejidad. En la actualidad es común observar sistemas muy complejos compuestos por una gran cantidad de elementos en un solo circuito integrado, conocidos como *System on Chip* (SoC). En los últimos años se ha incrementado el uso de dispositivos lógicos programables como los FPGAs, como una opción a los ASICs, aprovechando su aumento de capacidades y funciones así como a la considerable reducción de su precio en el mercado.

Los FPGAs son plataformas tecnológicas que permiten el desarrollo de sistemas donde se potencializan las ventajas de la coexistencia de software y hardware. En los FPGAs se implementa estructuras de hardware que realizan cálculos que requieren millones de operaciones las cuales se realizan en recursos contenidos en el *chip* de silicio. Tales sistemas pueden ser cientos de veces más rápidos que los diseños tradicionales basados en un solo microprocesador. Los sistemas basados en FPGAs son sistemas que pueden ser programados y reprogramados multiples veces.

En este capítulo se abordan las generalidades de los dispositivos FPGA, que permitirán tener una panorámica conceptual de la arquitectura y los recursos internos del dispositivo, para utilizarlo como plataforma de desarrollo para el nuevo sistema de control de orientación de SATEDU.

2.1 Arquitectura general de un FPGA

Un FPGA (*Field Programmable Gate Array*) es un dispositivo semiconductor que está constituido por bloques lógicos programables, cuya interconexión y funcionalidad puede ser configurada y definida mediante un lenguaje de descripción de *hardware*, como VHDL, *Verilog* o bien, de forma esquemática mediante diagramas de estado. Esquemáticamente un FPGA es una matriz de bloques configurables que se pueden conectar por medio de segmentos de pistas de diferentes longitudes (canales de ruteo ó interconexión), además de interruptores programables para enlazar bloques lógicos a pistas, o pistas entre sí, como se muestra en la figura 2.1. De forma práctica podemos afirmar que lo que se programa en un FPGA son los interruptores que permiten las conexiones entre los diferentes bloques más la configuración de los propios bloques.

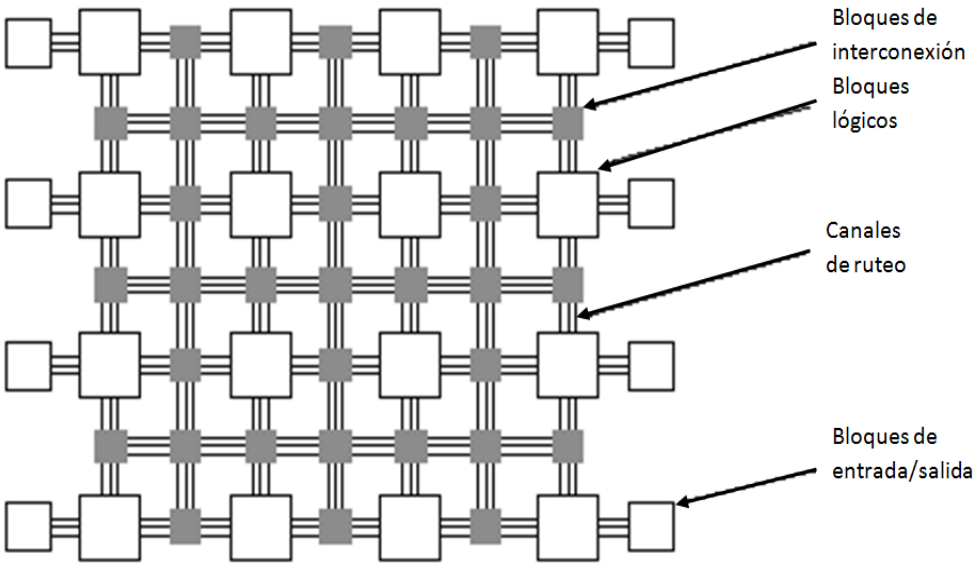


Figura 2.1. Arquitectura demostrativa de un FPGA de Xilinx.

2.1.1 Bloque lógico programable (CLB)

La arquitectura interna de un FPGA está integrada por una serie de bloques que a su vez, agrupan a una serie de elementos básicos tanto combinacionales como de memoria, que permiten la implementación de diferentes esquemas.

Existen varios tipos de arquitecturas de CLBs, los cuales varían en función de su complejidad, familia del dispositivo y fabricante. Por ejemplo, en el caso de Xilinx, los CLBs están compuestos por *slices*, estos a su vez están compuestos de un par de LUTs, un par de *flip-flops*, la cadena de acarreo y demás componentes lógicos.

Existen diferentes números de *slices* para un CLB, dependiendo de la familia del dispositivo se pueden encontrar 1, 2 ó 4 porciones. En el caso de Altera, llama *slices* a un

grupo de LUTs, un *flip-flop* y un LE *(logic element)*. Los LE están agrupados dentro de los bloques de arreglos lógicos LAB (*Logic Array Blocks)*, en grupos de 8 a 10 LEs dependiendo de la familia del dispositivo. Existen dos cadenas dentro de un LAB, una denominada acarreo rápido, similar a la que se encuentra en los dispositivos de Xilinx, además de una cadena cascada, para la realización de funciones grandes. Para efectos de este trabajo, abordaremos únicamente conceptos asociados con FPGAs de Xilinx.

Los CLBs están formados por *slices*, los cuales a su vez están divididos en dos partes llamadas SLICEM y SLICEL como se muestra en la figura 2.2.

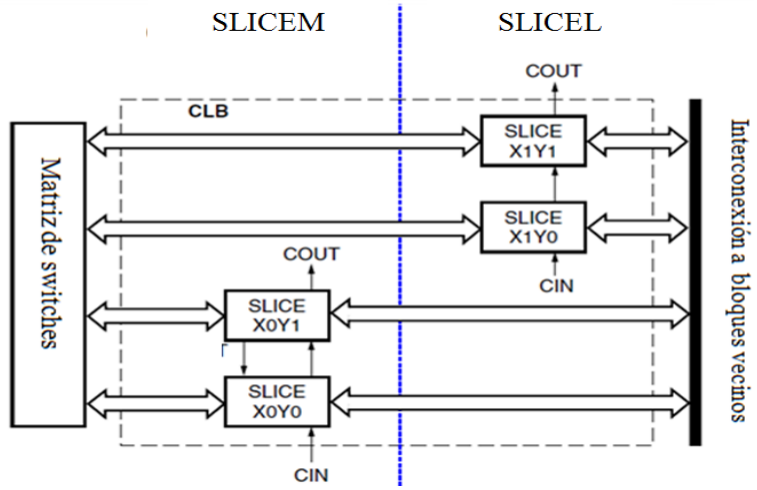


Figura 2.2 Arreglo de slices dentro de un CLB de la familia Spartan-3E de Xilinx.

Los *slices,* realizan la mayor parte de la funcionalidad del FPGA y suelen estar agrupados de 2 en 2 o de 4 en 4. Un *slices* está integrado por celdas lógicas (CL), las cuales a su vez constan de una parte combinacional y una secuencial. La parte combinacional permite implementar funciones booleanas e independientemente del fabricante, este bloque se basa principalmente en Tablas de búsqueda LUTs *(Look-Up Table)*. La parte secuencial está constituida por elementos básicos de memoria, como *flip-flops*, esta parte permite sincronizar la señal a la salida con una señal de reloj externa, lo cual es útil para diseñar circuitos secuenciales e implementar registros. La arquitectura de una CL se muestra en la figura 2.3.

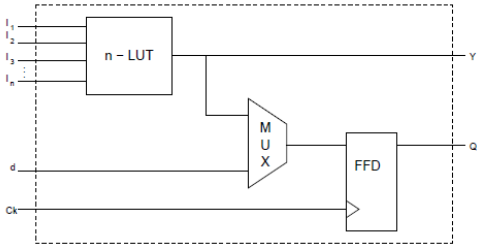


Figura 2.3. Arquitectura sencilla de una celda lógica.

La LUT es una unidad de memoria con una salida de 1 bit que esencialmente implementa una tabla de verdad, donde cada combinación de entrada genera una cierta salida lógica. Cuando se implementa una función lógica, las entradas de ésta son las direcciones de la memoria, y en cada celda se almacena el resultado de cada una de las combinaciones correspondientes como se muestra en la figura 2.4.

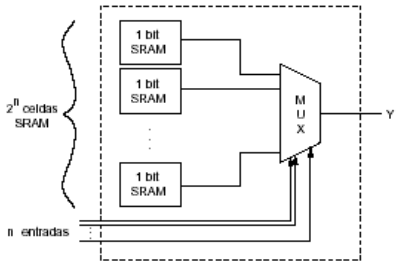


Figura 2.4. Arquitectura básica de una LUT.

2.1.2 Bloques de entrada/salida (IOB)

Estos bloques permiten la entrada y/o salida de señales del exterior al interior y viceversa, dependen de una interfaz eléctrica que debe contar con recursos tales como:

- Salidas configurables como *tri-state* u *open-collector*.
- Entradas con posibilidad de resistencias de *pull-up* o *pull-down* programables.
- Registros de salida.
- Registros de entrada.
- Diodos de protección.

Ya que generalmente los FPGAs trabajan a altas frecuencias y contienen miles de canales de conexión, los IOBs deben ser capaces de adecuar la impedancia en las terminales físicas del FPGA para evitar reflexiones de las señales (la lógica que implementa ésta característica se encuentra integrada dentro del FPGA).

2.1.3 Canales de ruteo o interconexión

Estos canales permiten una interconexión flexible entre los CLBs, además permiten la conexión entre los CLBs y los IOBs para interacción con el exterior.

Estas interconexiones pueden establecerse de múltiples formas. Por ejemplo, en los dispositivos Spartan-3 de Xilinx se realiza la siguiente clasificación:

- Líneas largas: Conectan a uno de cada seis bloques lógicos. Por su baja capacitancia, este tipo de líneas son utilizadas para el tránsito de señales de alta frecuencia con pequeños efectos de carga.

- Líneas Hex: Conectan uno de cada tres bloques lógicos. Cubren el desfasaje entre las líneas largas y las líneas dobles
- Líneas dobles: Conectan uno de cada dos bloques lógicos brindando una gran flexibilidad.
- Líneas directas: Conectan un CLB con sus vecinos contiguos.

Mientras más flexible es la malla de interconexiones el consumo de potencia será mayor y la velocidad de operación del FPGA disminuirá.

2.1.4 Bloque de interconexión

Estos bloques están conformados por interruptores, los cuales realizan la interconexión entre CLBs o los canales mismos.

2.2 Recursos Internos de los FPGAs

2.2.1 Recursos de ruteo o interconexión

Como se ha mencionado anteriormente, las interconexiones se logran gracias a interruptores programables contenidos dentro de los bloques de interconexión. Los recursos de interconexión se pueden clasificar en 5 grupos:

- **Estilo isla:** Consiste en un arreglo de bloques lógicos programables con canales de ruteo horizontales y verticales. Los bloques lógicos pueden acceder a los canales por medio de los bloques de interconexión, ver figura 2.5.

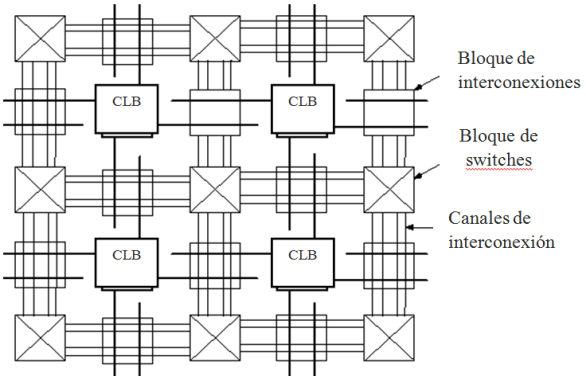


Figura 2.5. Arquitectura estilo isla.

- **Hilera:** Consiste en bloques lógicos programables arreglados en hileras, con canales de ruteo horizontales entre ellas. El ancho de las líneas de ruteo puede variar desde

un módulo hasta la longitud total del canal. Otras líneas corren verticalmente dentro de los bloques lógicos, esto es para conectar los canales horizontales con los bloques lógicos como se observa en la figura 2.6.

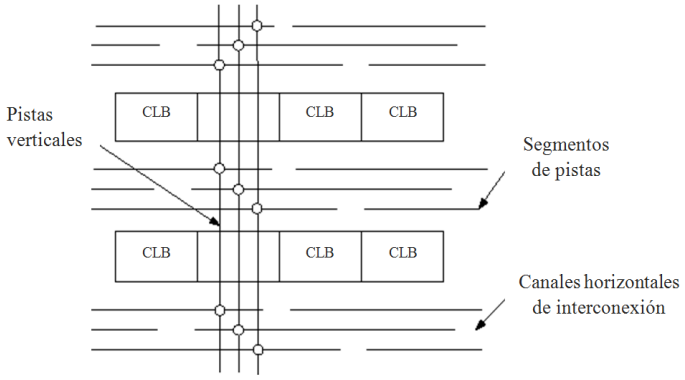


Figura 2.6. Arquitectura en Hilera.

- **Estructuras unidimensionales:** La mayoría de los FPGAs actuales son bidireccionales, lo cual proporciona más flexibilidad en las conexiones, pero exigen una mayor área de ruteo. La estructura unidimensional consiste en colocar los bloques lógicos a lo largo de un eje, lo cual simplifica y agiliza el tiempo de conexión, como se puede observar en la figura 2.7.
  Una desventaja de la arquitectura unidimensional es que si no hay los suficientes recursos de interconexión para un área específica, se vuelve más complicada toda la interconexión a diferencia de una estructura bidimensional ya que existen más alternativas de interconexión.

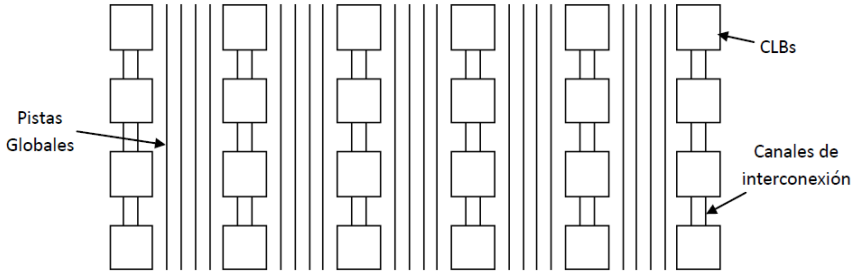


Figura 2.7. Estructura Unidimensional.

- **Mar de compuertas *(sea of gates)*:** Consiste en bloques lógicos que cubren todo el piso del dispositivo. Las conexiones se realizan entre bloques adyacentes y son generalmente más rápidas que los otros recursos de ruteo, ver figura 2.8.

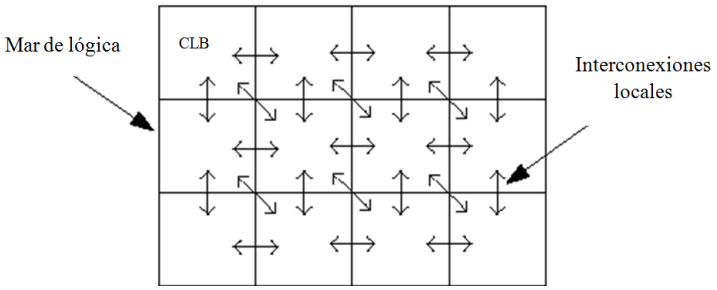


Figura 2.8. Arquitectura Mar de compuertas.

- **Jerárquica:** Consiste en conectar bloques lógicos en grupos, estos grupos están conectados recursivamente para formar una arquitectura jerárquica. La velocidad de la red se determina por el número de interruptores, los cuales permiten la conexión. Esta arquitectura reduce el número de interruptores para conexiones largas y por ende pueden correr a altas velocidades, ver figura 2.9.

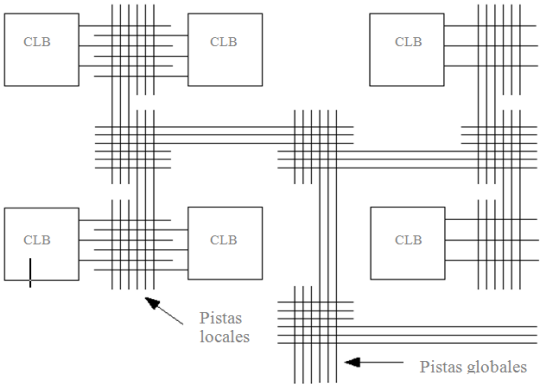


Figura 2.9. Arquitectura jerárquica.

2.2.2 SRAM-LUTs

Se utilizan en la mayoría de los FPGAs comerciales, en algunos dispositivos se permite utilizar las SRAM-LUTs como memorias RAM o registros de desplazamiento, esto como funciones adicionales a la principal de la LUT.

Una LUT de tres entradas puede utilizarse para formar una RAM de 8x1 bits como se muestra en la figura 2.10, al igual que esos bits pueden conectarse en cadena para formar uno o varios registros.

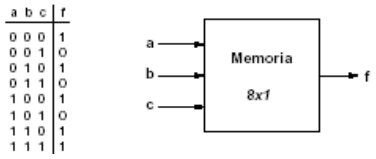


Figura 2.10. LUT utilizado como memoria SRAM.

2.2.3 Cadenas de acarreo rápidas *(fast carry chains)*

Su función es programar de manera rápida señales de acarreo entre dos elementos lógicos. Estas cadenas, figura 2.11, permiten implementar de manera rápida y eficiente multiplicadores, comparadores, contadores y sumadores.

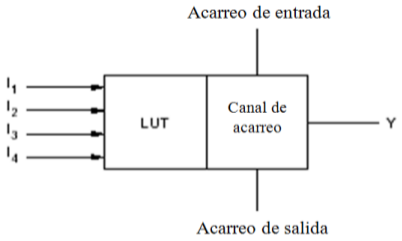


Figura 2.11. Celda de acarreo

2.2.4 RAM embebida

En la actualidad el uso de memorias es indispensable para ser integrado en diversas aplicaciones, por tal motivo, la mayoría de los dispositivos FPGA incluyen dentro de su arquitectura bloques de memoria RAM además de la SRAM-LUT.

Dependiendo del fabricante es la arquitectura que tiene el FPGA, donde además los bloques de memoria pueden estar situados en la periferia del dispositivo, dispersos sobre la superficie o en columnas, figura 2.12. Estos bloques de memoria pueden ser configurados de diferentes maneras, formando memorias de un sólo puerto (*single-port* RAMs), memorias de doble puerto (*dual-port* RAMs), memorias FIFO (*first-in first-out*), etc. También cuentan con un sistema de conexión dedicado, el cual permite una comunicación eficiente entre los bloques de memoria y los bloques lógicos.

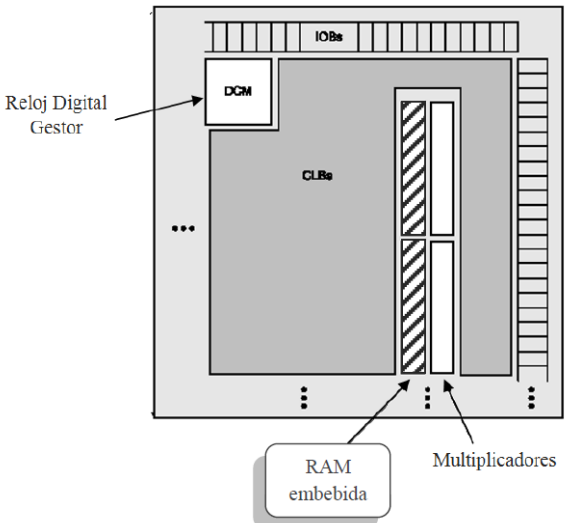


Figura 2.12. Ubicación de la memoria RAM dentro de un FPGA de la familia Spartan-3E de Xilinx.

2.2.5 Multiplicadores

Los multiplicadores se utilizan para aplicaciones de procesamiento digital de señales DSP, donde los algoritmos regularmente se basan en sumas y multiplicaciones, lo que mejora la velocidad de procesamiento, al contrario de lo que sucedería en el caso de decidir utilizar un gran número de CLBs. Generalmente los multiplicadores se encuentran cercanos a los bloques de RAM, ya que se utilizan para procesar datos dentro de dichos bloques, ver figura 2.13.

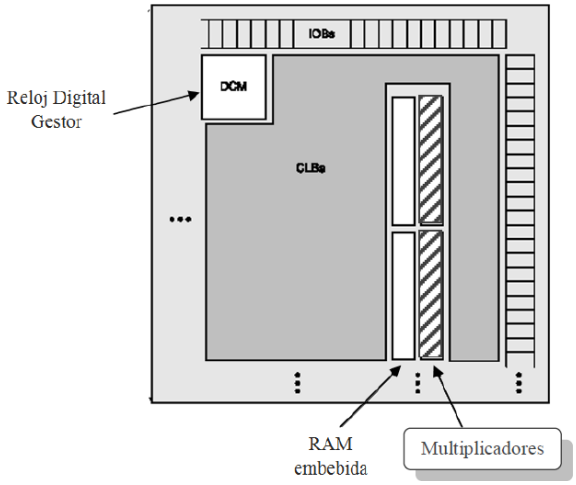


Figura 2.13. Ubicación de los multiplicadores dentro de un FPGA de la familia Spartan-3E de Xilinx.

2.2.6 Reloj Digital Gestor DCM *(Digital Clock Manager)*

Estos bloques pueden proporcionar auto calibración, retardos, multiplicaciones, divisiones, y diferentes señales de reloj. Las señales de reloj generadas por el DCM se conectan a otro bloque de reloj, el cual se encuentra conectado a todos los elementos secuenciales dentro del FPGA, tales como los *flip-flops* de los bloques lógicos, ya que es necesario que todos se encuentren sincronizados a una misma señal de reloj. La señal de reloj es distribuida en forma de árbol para que ésta pueda llegar en fase a todos los elementos secuenciales, con esto podemos asegurar la sincronización entre los elementos. Estas señales también se utilizan para manejar dispositivos externos al FPGA, ver figura 2.14.

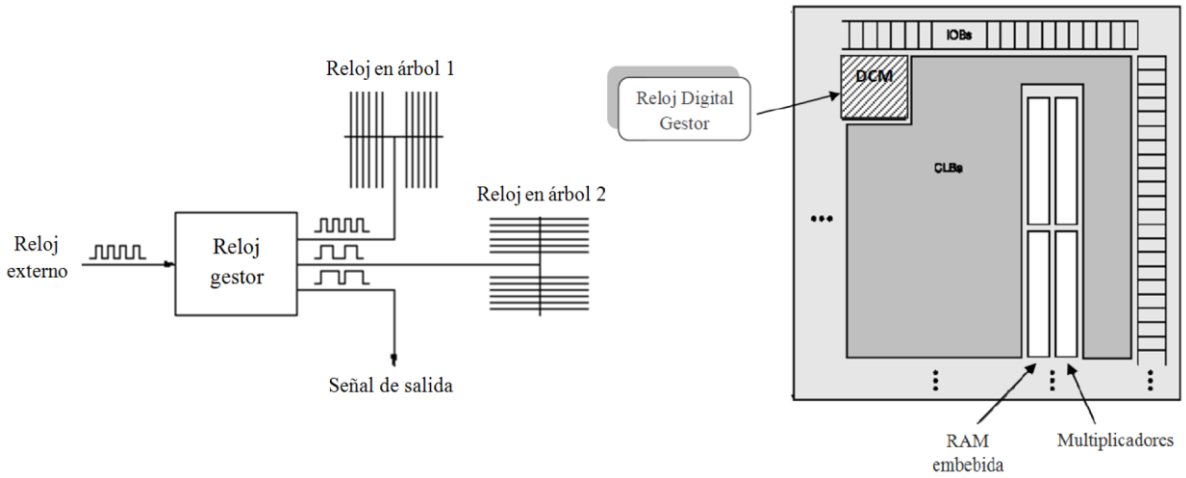


Figura 2.14. Estructura general de un DCM y su ubicación dentro de un FPGA de la familia Spartan-3E de Xilinx.

2.3 Tecnologías de programación

Existen diversas formas y métodos para realizar la carga de programas hacia el FPGA. En esta sección se mencionan algunos de los métodos más utilizados.

2.3.1 SRAM

En este método de programación la información se almacena en celdas de memoria SRAM. Cuando se implementa una red de interconexión por medio de transistores de paso, la SRAM controla si dichos transistores están encendidos o apagados, en cambio la programación de una LUT de un bloque lógico se almacena dentro de la celda de SRAM.

La ventaja de esta tecnología radica en que es posible reutilizar el mismo dispositivo para diferentes aplicaciones más, ya que puede ser reprogramado tantas veces como el semiconductor lo permita. Sin embargo existe una leve desventaja, ya que la memoria de almacenamiento es volátil, esto quiere decir que al apagar el sistema el programa útil se borra de la SRAM. La tecnología basada en SRAM es generalmente utilizada para plataformas reconfigurables.

2.3.2 Antifusible

Se llama antifusible ya que tiene la función inversa a un fusible. Esta tecnología utiliza una conexión programable cuya impedancia varía al aplicarle un voltaje de programación que varía con respecto al dispositivo utilizado Al principio la impedancia es de unos cuantos giga ohms, por lo tanto se podría considerar como circuito abierto, en el momento de que se

suministra un voltaje alto esta impedancia llega al valor de unos cuantos ohms y con esto se puede realizar la transferencia de datos para la reconfiguración del FPGA.

La ventaja de esta tecnología se halla en que el área del elemento de programación es del tamaño de una vía, lo cual reduce significativamente el espacio utilizado en el FPGA, a comparación con la tecnología basada en SRAM. Otra ventaja es que se pueden fabricar dispositivos más resistentes a altas radiaciones, ya que la radiación de partículas de gran energía puede borrar o modificar la información almacenada en las celdas de memoria. La desventaja que presenta es que no es posible volver a restablecer las conexiones entre elementos de la arquitectura del FPGA, esto quiere decir que el dispositivo solo puede ser programado una sola vez.

2.3.3 EPROM, EEPROM y FLASH

Esta tecnología combina la no volatilidad del antifusible y la reprogramación de la SRAM. La resistencia de los interruptores de interconexión es más grande que la del antifusible, mientras que la programación es más compleja que la utilizada con la SRAM, su tamaño es bastante más pequeño que el de una SRAM pero sin llegar a ser tan pequeño como un antifusible. Son reprogramables, aunque la velocidad de programación es mucho más lenta que la de una SRAM.

2.4 Flujo del diseño en FPGAs

El flujo del diseño utilizando dispositivos FPGA es un proceso que comprende desde el proponer una arquitectura, la captura de la lógica utilizando un lenguaje, la inclusión de bancos de prueba para verificación, hasta la implementación física del sistema en un dispositivo. Los pasos que puede llevar el realizar estos diseños se describen en los siguientes párrafos:

1.- **Diseño del sistema y obtención de los archivos HDL**

En esta sección se diseña la arquitectura a implementar en el FPGA utilizando un lenguaje de descripción de hardware. Se realiza la verificación de la sintaxis del programa, para detectar errores durante la captura de la lógica de la arquitectura. Se agrega un archivo de restricciones para definir el ruteo de las entradas y salidas físicas de señales en el dispositivo físico y los niveles de voltaje asociados a ellas.

2.- **Creación del banco de pruebas en HDL y realización de la simulación RTL**
El termino RTL refleja el hecho de que el código HDL se hace a nivel de transferencia de registros.

3.- **Síntesis e Implementación**
El proceso de síntesis es regularmente conocido como síntesis lógico, en donde el software transforma la arquitectura HDL y construye componentes genéricos a nivel de compuerta.
El proceso de implementación consiste en tres pequeños procesos: interpretar, trazar el plano, colocar y rutear. El proceso de interpretación combina varios archivos de diseño en una sola lista de conexiones. El proceso de trazar el plano es regularmente conocido como tecnología de trazado, traza el plano de las compuertas genéricas en la lista de conexiones a las celdas lógicas y los IOBs del FPGA. El proceso de colocación y ruteo, deriva del diseño físico dentro del chip FPGA. Coloca las celdas en lugares físicos y determina las rutas para conectar varias señales.
En el flujo de Diseño, al final del proceso de implementación se realiza el análisis estático de tiempo que determina varios parámetros de tiempo como retardo máximo de propagación y frecuencia máxima de reloj.

4.- **Generación y descarga del archivo de programación**
En este proceso, se genera el archivo de programación de acuerdo a la lista final de conexiones. Este archivo se descarga al dispositivo FPGA en serie para configurar las celdas lógicas y los interruptores de interconexión.

En la figura 2.15 se muestra el diagrama de flujo del diseño antes descrito.

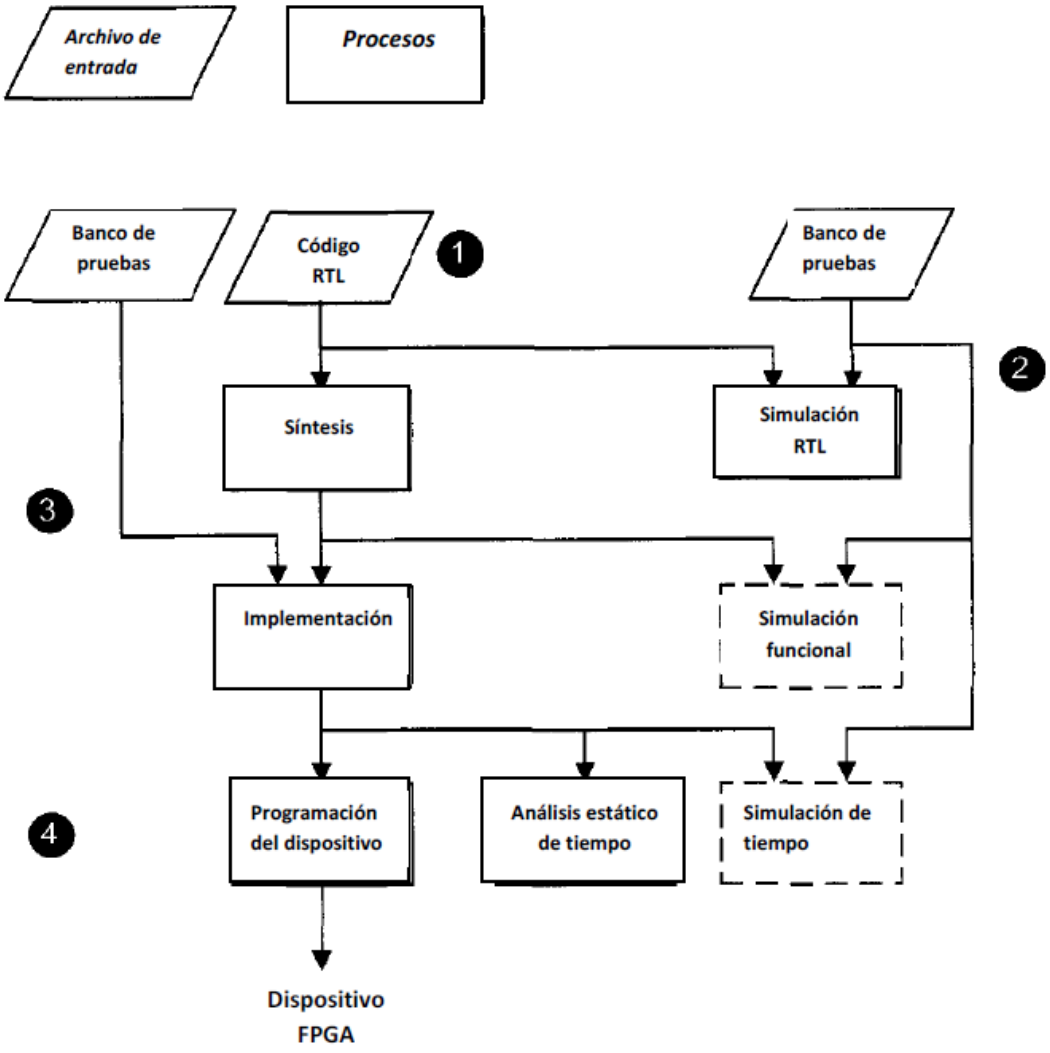


Figura 2.15. Diagrama de flujo del diseño.

En la parte izquierda del diagrama 2.15 se muestra el proceso de programación y refinamiento en el proceso de diseño en un FPGA, en el cual al sistema se le conecta temporalmente un banco de pruebas descrito en HDL. El cual es un bloque que contiene datos de prueba, los cuales se le inyectan a la arquitectura o unidad bajo prueba (UUT) y que permiten la verificación funcional a nivel de software del programa que se ha desarrollado mediante gráficas de tiempo. Una vez realizada esta fase de verificación es posible la implementación confiable de la arquitectura desarrollada en el dispositivo real, descargando mediante algún software adecuado, el *bitstream* de la configuración y datos del FPGA.

En el Flujo de Xilinx, la simulación funcional puede realizarse después del proceso de síntesis y la simulación de tiempo puede realizarse al final del proceso de implementación. La simulación funcional utiliza la lista de conexiones sintetizada para sustituir la descripción RTL y comprobar la exactitud del proceso de síntesis. La simulación de tiempo

utiliza la lista de conexiones final. Ambas simulaciones pueden omitirse del flujo de diseño, constituyéndose sólo una herramienta opcional, pero muy útil, para la verificación preliminar del funcionamiento parcial o total de la arquitectura desarrollada en el FPGA.

En este capítulo se ha hablado de la arquitectura y los recursos internos con los que cuenta un FPGA de Xilinx, donde dependiendo del tipo de la familia con la que se trabaje varían el número de recursos internos y su posición dentro del dispositivo. Con ello se ha logrado dar un panorama amplio sobre este tipo de dispositivos y su funcionamiento, el cual proporciona las herramientas necesarias para la comprensión de este tema de tesis.

En el capítulo siguiente hablaremos de la tarjeta de desarrollo Spartan 3E-starter kit, la cual fue una herramienta primordial para validar algunas pruebas realizadas para esta tesis, además de que cuenta con un FPGA de la misma familia que el utilizado como hardware central de un sistema con recursos mínimos.

CAPÍTULO

3

Descripción de la arquitectura del sistema de desarrollo Spartan 3E-Starter kit

El proceso de diseño de un sistema electrónico, incluye una serie de pasos lógicos que permiten la evaluación y selección de los elementos y componentes más adecuados para la aplicación. Los elementos de diseño que se incluyen se agrupan en dos bloques: *hardware* y *software*. Respecto a la etapa de definición de *hardware*, el uso de plataformas de desarrollo y evaluación (tarjetas comerciales de desarrollo) es una práctica relativamente común que permite la aceleración en la toma de decisiones para integrar la arquitectura final de un sistema.

En el desarrollo de esta tesis, el uso de la tarjeta de evaluación Spartan 3E-Starter kit (SK), permitió acelerar el trabajo de definición de esquemas de reconfiguración y la integración de una plataforma FPGA de diseño propio de acuerdo con las necesidades que requiere el satélite educativo SATEDU que desarrolla el Instituto de ingeniería, UNAM.

En este capítulo se abordan las generalidades que permiten conocer el sistema de desarrollo y evaluación Spartan 3E-SK.

3.1 Sistema de desarrollo Spartan 3E-SK

Los objetivos para utilizar el sistema de desarrollo Spartan 3E-SK fueron en primera instancia realizar y validar algunas pruebas técnicas de reconfiguración total del dispositivo FPGA montado en la tarjeta así como utilizarla como instrumento para acelerar el proceso de definición del diseño de una tarjeta propia, con los recursos de interfaces I/O y electrónica de soporte a la medida.

Finalmente y después de un arduo estudio y experimentación con cada uno de los esquemas e implementando cada una de las técnicas de reconfiguración propuestas en diferentes fuentes bibliográficas, que abordaban el tópico de la reconfiguración de FPGAs de la familia Spartan 3E, se llegó a la conclusión de que el modo a adoptar, por su factibilidad técnica de implantación era aquella que utiliza el protocolo JTAG.

Como se ha mencionado, este sistema de desarrollo y evaluación agrupa las herramientas necesarias para cumplir con nuestros objetivos. A continuación se listan algunos de los componentes más importantes de la tarjeta de desarrollo Spartan 3E-SK, resaltando los módulos que se utilizaron para este trabajo de tesis.

- **FPGA Spartan 3E de Xilinx XC3S500E**.
- **CPLD *CoolRunner* de Xilinx XC2C64A, 64 macroceldas.**
- Memoria *flash* PROM M25P16 de 16Mbits SPI (*STMicro*).
- Memoria *parallel* NOR *flash* PROM Intel de 128 Mbit (*StrataFlash*).
- **Plataforma flash PROM Xilinx XCF04S de 4Mbits**
- Memoria DDR SDRAM MT46V32M16 de 512 Mbits, con interfaz de 16 datos.
- Memoria EEPROM 1-*wire* SHA-1, para copia de protección del *bitstream*.
- Convertidor digital analógico (CDA) LTC2624.
- Convertidor analógico digital (CAD) LTC1407A-1.
- Amplificador LTC6912-1, como preamplificador del CAD.
- Interfaz de *ethernet* de capa física (PHY), SMSC LAN83C185 10/100.
- Conector RJ-45 de *ethernet.*
- Cristal de 25MHz.
- Pantalla LCD de 16x2.
- Puerto para pantalla VGA.
- Dos puertos RS232 de 9 terminales (estilo DTE y DCE).
- Conector tipo B, USB para carga de programa y depuración del FPGA.
- **Oscilador de 50MHz.**
- Socket para oscilador opcional, empaque DIP 8 terminales.
- Conector SMA para entrada de reloj.
- ***Switch Push-button* para programación.**
- **Cuatro *switches* de propósito general.**

- Cuatro *switches push-button*.
- **Ocho LEDs de propósito general.**
- LED de configuración
- Dos entradas para CAD.
- Cuatro salidas de CDA.
- **Conectores para selección de modo de configuración.**
- **Conectores para comunicación JTAG.**
- Conector mini-DIN de 6 terminales, para teclado o *mouse*.
- Conector de 100 terminales Hirose FX2, J3.
- Conectores de acceso, J1.
- **Conectores de acceso, J2.**
- Resistencias y capacitores de soporte.

3.2 Descripción de los componentes utilizados en el sistema Spartan 3E-SK (SP3E-SK)

El sistema de desarrollo SP3E-SK es una plataforma de desarrollo y evaluación que cuenta con recursos en *hardware* que permiten la validación operativa de diversas arquitecturas y estructuras digitales que se describen mediante lenguaje descriptor de *hardware*, utilizando ambientes de *software* que facilitan la captura del código, su síntesis, ruteo y posterior implementación física. Un diagrama general del sistema SP3E-SK, se muestra en la figura 3.1.

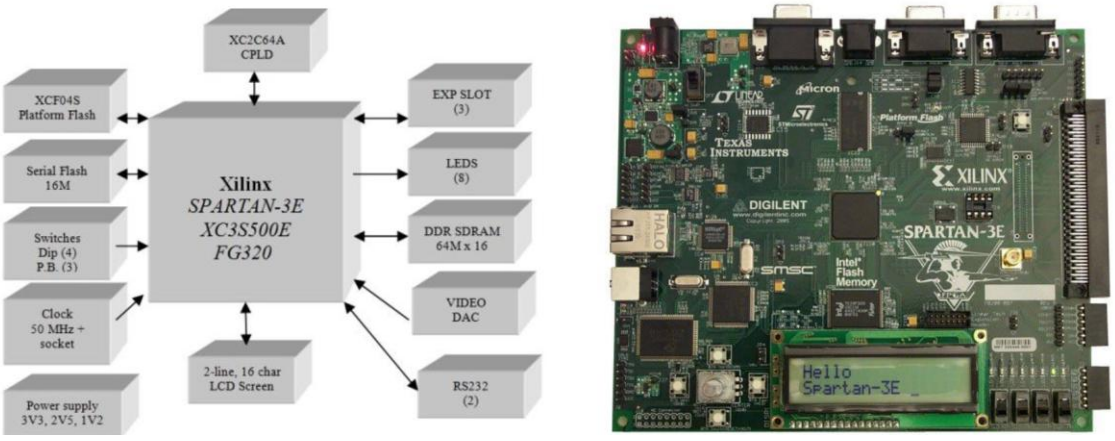


Figura 3.1. Esquema general del sistema SP3E-SK.

A continuación se describen algunos de los bloques que aparecen en la figura 3.1.

FPGA Spartan 3E XC3S500E de Xilinx

Es un dispositivo de bajo costo y alto rendimiento para aplicaciones de gran volumen de lógica. Este dispositivo contiene 320 terminales de las cuales 232 se utilizan como terminales de entrada/salida. Trabaja con señales de 3.3V, 2.5V, 1.8V, 1.5V y 1.2V (dentro de esta tarjeta trabaja con señales de 3.3V, 2.5V y 1.2V).

Posee cuatro gestores de reloj digital (DCMs), que trabajan con un rango de frecuencias que van desde 5MHz hasta 300MHz utilizando un oscilador externo (en el caso de este sistema trabaja con un oscilador de 50MHz), con divisores, multiplicadores y sintetizadores de frecuencia, ocho señales de reloj globales y ocho señales de reloj designadas para cada mitad del dispositivo. Está integrada por 360Kbits de memoria RAM rápida y 73Kbits de memoria RAM distribuida. Se compone por puertos designados para configuración por comunicación JTAG *IEEE 1149.1/1532, Master Serial, Slave serial, Master parallel Up y Down, y SPI Serial Flash.*

En la figura 3.2 se muestran los recursos internos del FPGA XC3S500E Spartan 3E.

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92

Figura 3.2 Recursos internos del FPGA.

Oscilador 50MHz

Genera la señal de reloj con la que opera la lógica del FPGA. La tarjeta además posee un conector tipo RSMA y un *socket* para ingresar otras fuentes de reloj.

Botón para programación

Inicia el proceso de reconfiguración del FPGA a través del modo de configuración *Master Serial*. Se encuentra en la parte superior derecha de la tarjeta, como se muestra en la figura 3.3.

Conectores para comunicación JTAG

Estos conectores se utilizan para enlazar las terminales de JTAG externas con las terminales del puerto JTAG del FPGA, para entablar la comunicación y llevar a cabo la configuración del dispositivo. Se encuentran en la parte superior derecha de la tarjeta, como se puede observar en la figura 3.3.

Conectores para modo de configuración y pin LED de configuración

Estos conectores se utilizan para seleccionar el modo de configuración del FPGA. Estos conectores se encuentran en pares y se etiquetan con un logo que va desde M2 a M0, una fila de conectores se encuentran conectados a tierra o en un nivel lógico '0', mientras la siguiente fila se encuentra conectada a las terminales para modo de configuración del FPGA. Para seleccionar los diferentes modos de configuración es necesario conectar los conectores de configuración a sus respectivos conectores de tierra, de no ser así, el sistema tomara esas entradas en un nivel alto '1'.

El sistema SP3E-SK soporta varios modos de configuración, los cuales son:

- *Modo Master Serial*
- *Modo Master SPI*
- *Modo Master BPI*
  - *BPI Up*
  - *BPI Down*
- Modo JTAG

Para fines de verificación, la tarjeta cuenta con un LED indicador (DONE), el cual enciende una vez que la reconfiguración del dispositivo FPGA, por cualquiera de las formas antes descritas, ha sido realizada de manera exitosa. Estos componentes se muestran en la figura 3.3.

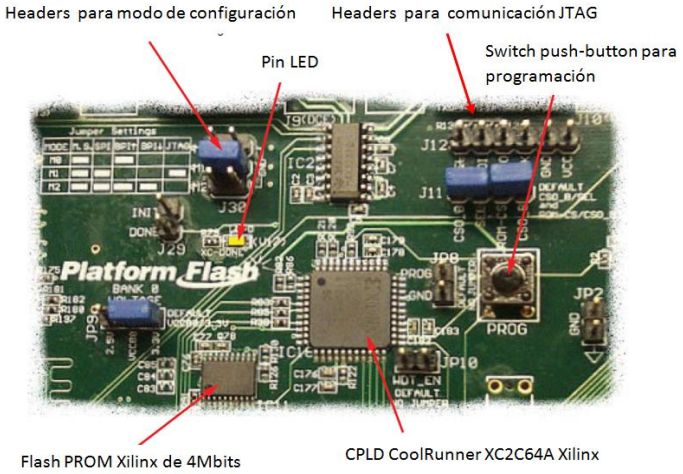


Figura 3.3. Componentes para configuración del sistema SP3E-SK.

Plataforma *Flash* PROM XCF04S y CPLD *CoolRunner* XC2C64A de Xilinx

Memoria XCF04S: Opera con un voltaje de alimentación de 3.3V, con un oscilador externo con una frecuencia máxima de 33MHz. Este tipo de memorias funciona como interfaz serial para la configuración del FPGA; en la figura 3.4 se muestra un diagrama de bloques del funcionamiento de la memoria XCF04S como interfaz serial.

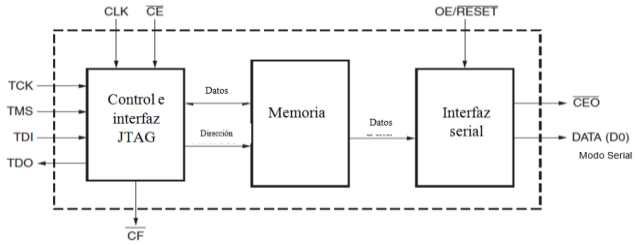


Figura 3.4. Diagrama de la plataforma Flash PROM XCF04S.

Particularmente la memoria XCF04S es necesaria para la implementación de un esquema de reconfiguración del FPGA a través de la comunicación JTAG, ya que sirve como medio de almacenamiento temporal del *bitstream* antes de descargarlo al dispositivo FPGA si se selecciona el modo de reconfiguración *Master Serial*.

CPLD XC2C64A: Opera con un voltaje de alimentación en un intervalo de -0.5V hasta 2V (en el sistema SP3E-SK su voltaje de alimentación es de 1.8V). Utiliza modos de reloj flexibles y gracias a ello puede funcionar a frecuencias de reloj semejantes a las de otros dispositivos programables. Las terminales de entrada/salida son compatibles con niveles de 1.5V, 1.8V, 2.5V, y 3.3V.

Interruptores, LEDs y Conectores de acceso J2

Los interruptores se encuentran en la parte inferior derecha de la tarjeta, y están marcados con una etiqueta que va desde SW3 hasta SW0. Estos interruptores se utilizan como entradas lógicas para validar el funcionamiento de la arquitectura de cómputo cargada al FPGA a través de la comunicación JTAG o por cualquier otro medio de reconfiguración.

Al igual que los interruptores, los LEDs se encuentran en la parte inferior derecha de la tarjeta, y están marcados con un etiquetas que van desde LED7 hasta LED0. Los LEDs se utilizan como salidas lógicas de verificación de una arquitectura que se carga al FPGA.

Los conectores de expansión se encuentran en la misma parte que los LEDs y los interruptores, y son utilizados como entradas y salidas lógicas para validar la arquitectura cargada.

La figura 3.5 muestra la posición y el orden en el que se encuentran los LEDs, interruptores y conectores de expansión o salida/entrada.

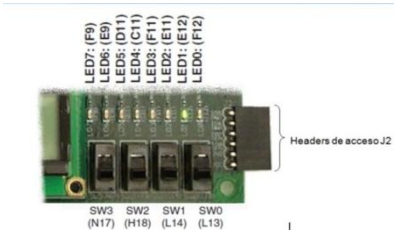


Figura 3.5. Interruptores, LEDs y Conectores de expansión o acceso.

3.3 Modos de configuración del FPGA

Como se ha mencionado dentro del apartado 3.2, existen varios modos de configuración para el FPGA integrado en el sistema de desarrollo SP3E-SK. En este subcapítulo se abordarán brevemente algunos detalles de cada uno de estos modos de configuración.

3.3.1 Modo Master Serial

En este modo de configuración, el *bitstream* de configuración generalmente reside en una memoria no volátil contenida en la misma tarjeta SP3E-SK. El FPGA internamente genera una señal de reloj de configuración llamada CCLK, el cual controla el proceso de configuración.

Particularmente en este modo de configuración, el FPGA utiliza la interfaz serie con que cuentan las memorias de tecnología PROM de las familias XCF**S y XCF**P instaladas en la tarjeta SP3E-SK. La plataforma Flash PROM ofrece las siguientes ventajas al sistema:

- Interfaz sencilla, con pocas terminales empleadas durante la configuración.
- Bajo costo por bit de configuración.
- Mayor ancho de banda entre la PROM y el FPGA, con esto se obtiene un menor tiempo de configuración.
- Programable en sistema y reprogramable por medio de una interfaz JTAG integrada.
- Compatible con el *software* de programación IMPACT de XILINX.
- Múltiples entradas/salidas y niveles de voltaje JTAG, para mayor flexibilidad del sistema.

La forma de seleccionar el Modo *Master Serial* se realiza ubicando los *jumpers* de conexión en el modo de configuración correspondiente en un nivel lógico ‘0’, (M[2:0] = 0:0:0). El esquema de conexión se muestra en la Figura 3.6.

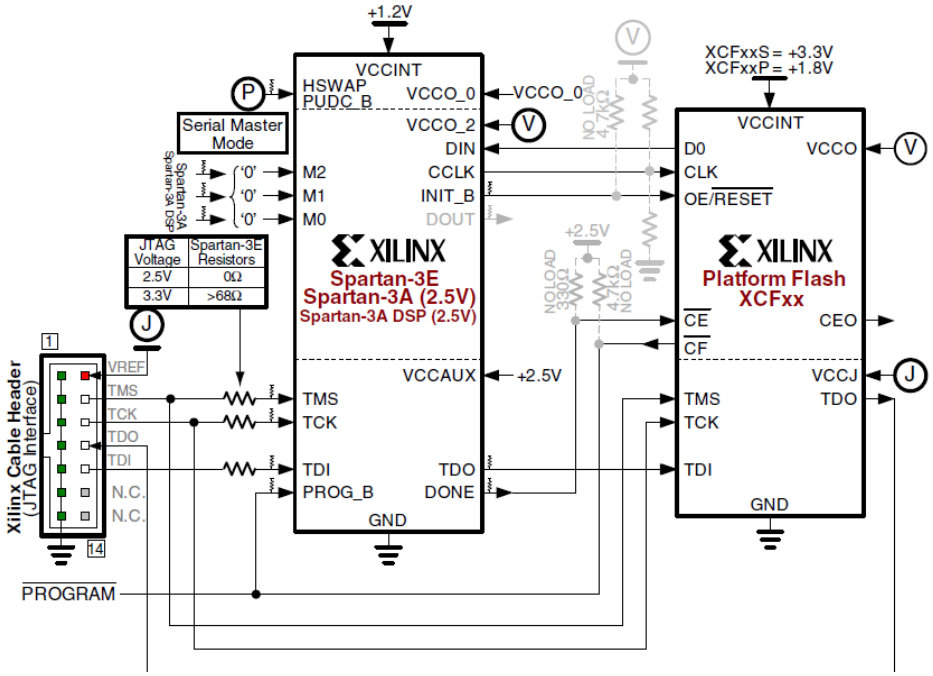


Figura 3.6. Modo de configuración Master Serial.

El FPGA suministra la señal de reloj CCLK, esta señal funciona como oscilador interno de la plataforma *Flash* PROM. Como respuesta, la plataforma *Flash* PROM suministra bits de datos de forma serial al FPGA (DIN); el FPGA acepta estos datos cada vez que la señal de reloj CCLK se encuentra en estado lógico alto. El FPGA restablece la comunicación con la memoria PROM por medio de la terminal INIT_B.

3.3.2 Modo Master SPI

En el modo *Master* SPI se configura a los dispositivos FPGAs de las familias Spartan-3E y Spartan 3A a partir de la memoria estándar SPI *Flash* PROM de las familias M25P**, M25PE** y M45PE** instaladas en la tarjeta de desarrollo SP3E-SK. Al igual que la memoria *Flash* PROM, la memoria SPI *Flash* PROM es también compatible con el software IMPACT de XILINX.

Este modo de configuración es ideal para aplicaciones con las siguientes características:

- Cuando se utilizan memorias *Flash* PROM en el sistema.
- Cuando el FPGA necesita almacenar datos en una memoria no volátil.

La forma de seleccionar el Modo *Master SPI* se obtiene posicionando los *jumpers* de conexión M2 y M1 a un nivel lógico ‘0’, mientras que el conector M0 se mantendrá en nivel lógico ‘1’, (M[2:0] = 0:0:1). El esquema de conexión se muestra en la Figura 3.7.

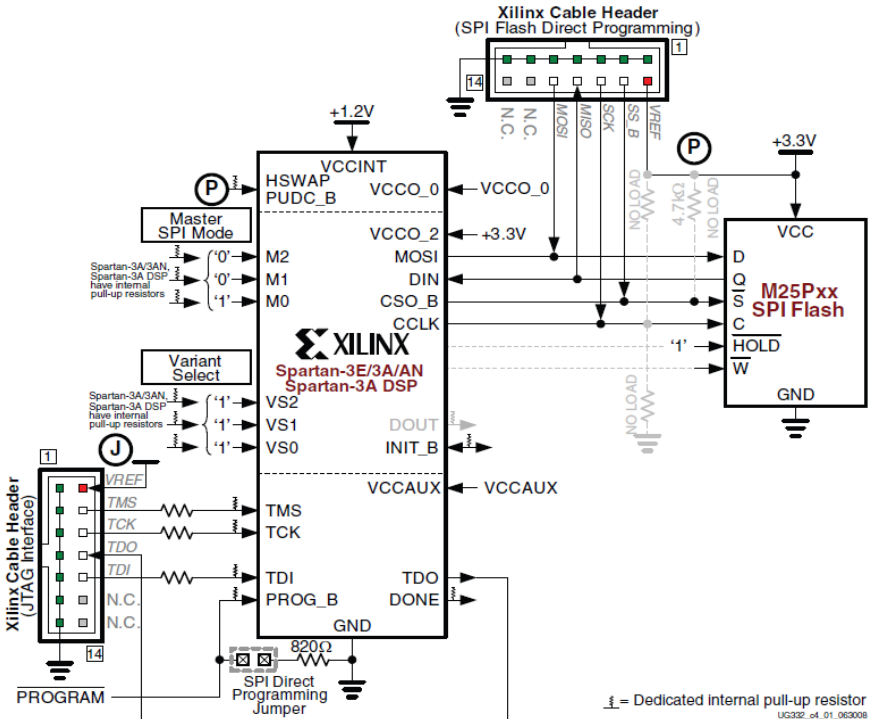


Figura 3.7. Modo de configuración Master SPI.

La comunicación entre el FPGA y memoria SPI *Flash* PROM se establece a través del protocolo de comunicación serie SPI. El bus de este protocolo de comunicación está formado por cuatro señales denominadas MOSI, DIN, SCK y CSO_B. El FPGA suministra la señal de reloj CCLK hacia SCK, que también utiliza la memoria SPI *Flash* PROM como señal de oscilador interno.

Los dispositivos FPGAs no poseen una interfaz dedicada SPI, salvo para la configuración. En consecuencia, para interactuar con los dispositivos SPI, *flash* u otros, después de la configuración, la arquitectura de cómputo debe contener un controlador y recursos para operar el bus SPI como maestro.

3.3.3 Modo Master BPI

El modo de configuración *Master Byte-wide Peripheral Interface BPI* se encuentra disponible para los FPGAs de las familias Spartan 3A, 3AN, 3A DSP y Spartan 3E cuyos dispositivos se configuran a partir de una memoria estándar paralela de tecnología NOR *Flash* PROM.

Cabe señalar que este modo de configuración no es soportado íntegramente por la familia Spartan 3E, utilizando en su lugar el modo de configuración a partir de una memoria paralela *Flash* PROM que opera de forma muy similar.

En el modo BPI, un dispositivo Spartan 3E o cualquier otro de los soportados se configura a sí mismo a partir de una memoria paralela de tecnología NOR *Flash* PROM, como se observa en la figura 3.8.

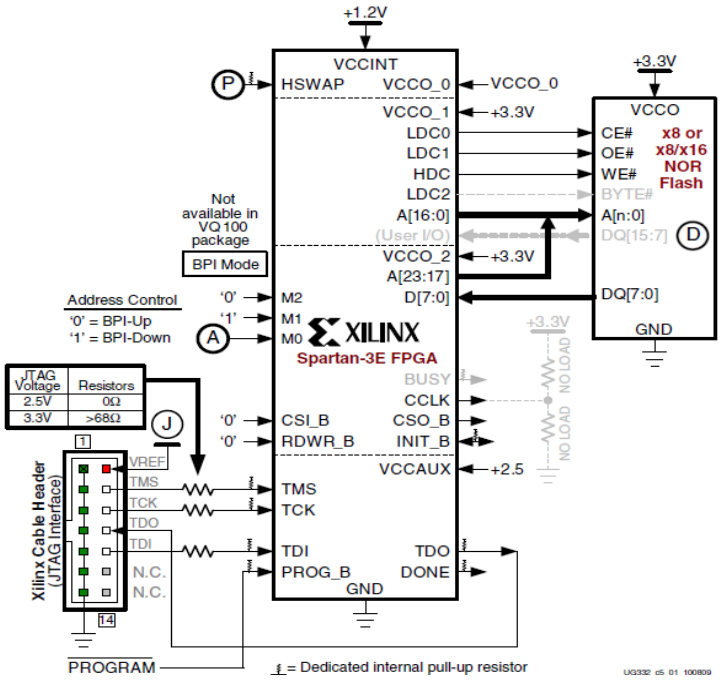


Figura 3.8. Modo Master BPI parallel.

La interfaz de configuración BPI está diseñada básicamente para soportar memorias de tipo estándar paralelas de tecnología NOR *Flash* PROMs. La interfaz soporta anchos de 8 y 16 bytes. Para la configuración del FPGA, la interfaz BPI no fija alguna característica específica de la *Flash* PROM, tal como un bloque de arranque o un tamaño específico de sector.

El FPGA suministra la señal de reloj por medio de la terminal de salida CCLK, el cual controla el flujo de datos. Sin embargo, la señal CCLK no está conectada en aplicaciones de un solo dispositivo. El FPGA controla tres terminales en nivel lógico bajo durante la configuración (LDC[2:0]) y una terminal en nivel lógico alto durante la configuración (HDC) en las entradas de control de la PROM.

3.3.4 Modo JTAG

Los dispositivos de la familia Spartan 3E son compatibles con el estándar IEEE 1149.1/1532 *Test Access Port* (TAP) y la arquitectura *Boundary-Scan* propias del protocolo JTAG. La arquitectura comprende el *Test Access Port* (TAP), el control del TAP, el registro de instrucciones, el registro de identificación, el registro *boundary-scan*, el registro *bypass* y el registro *USERCODE*.

El modo de configuración JTAG tiene una característica que lo mantiene por encima de los demás modos de configuración. Esta característica se debe a que el protocolo JTAG permite realizar la configuración de uno o más dispositivos conectados en cadena utilizando el mismo puerto de configuración. Para este modo de configuración se dedican 4 terminales: TMS, TCK, TDI y TDO donde:

- TDI: Es la entrada de el dispositivo consecutivo en el canal.
- TDO: Señal de salida de un dispositivo.
- TCK: La señal de reloj que sincroniza la comunicación.
- TMS: La señal requerida para llevar a cabo la comunicación.

La salida del último dispositivo de la cadena se conecta a la terminal TDO del puerto.

Para seleccionar este modo de configuración se colocan los conectores M2 y M0 a un nivel lógico ‘1’ y M1 dejarlo en nivel lógico ‘0’, (M[2:0] = <1:0:1) , como se puede observar en la figura 3.9.

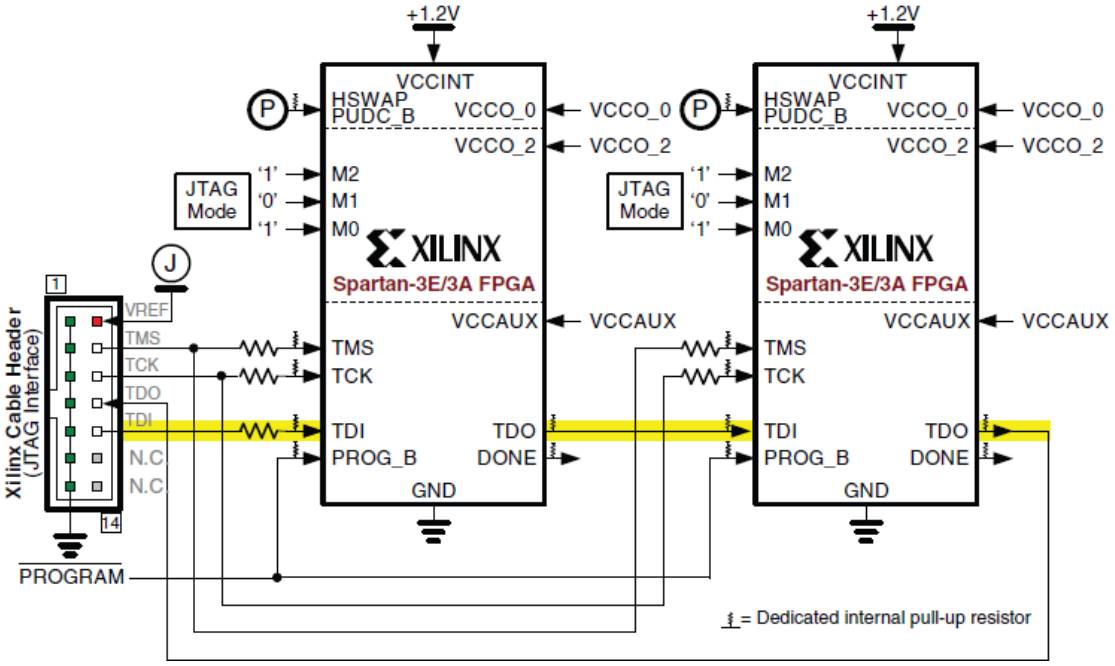


Figura 3.9. Modo de configuración JTAG.

El voltaje de alimentación para configurar el FPGA de este modo se encuentra dado por la terminal VCC_{AUX}, la cual opera con 2.5V, sin embargo pueden variar dependiendo de la familia con la que se esté trabajando.

Una vez descrita la arquitectura de la tarjeta de desarrollo utilizada, y habiendo conocido los diferentes modos de configuración existentes para configurar un dispositivo FPGA de la

familia Spartan 3E, lo siguiente fue proponer diferentes esquemas de reconfiguración que pudieran ser viables para llevar a cabo el proceso de reconfiguración de forma remota.

En el siguiente capítulo se habla de los diferentes esquemas de reconfiguración propuestos, también se mencionan las ventajas y desventajas que presentan cada uno de ellos, y se darán a conocer las razones por las cuales se determinó utilizar el modo de configuración JTAG.

CAPÍTULO 4

Descripción de propuestas para implantar procesos de reconfiguración remota para FPGAs

Dentro de las principales ventajas que ofrecen los sistemas basados en FPGAs, está la flexibilidad para reconfigurar las arquitecturas de cómputo que en ellos se construyen. Sin embargo, muchos de esos sistemas no explotan con rigor dicha característica, o bien lo hacen pero de una forma parcial. Existen varios esquemas en *hardware* y *software* que permiten la carga del *bitstream* de reconfiguración a partir de versiones respaldadas en memorias externas, algunas de ellas mostradas en el capitulo anterior.

En sistemas aeroespaciales el contar con un sistema reconfigurable al vuelo, que permita la actualización remota del *hardware* y *software* en tiempo de ejecución, ya sea de una forma total o parcial, dota al sistema de características superiores en términos de flexibilidad, tolerancia a fallas y eficiencia en la creación y actualización de nuevas arquitecturas e integración de sistemas.

En este capítulo se muestran algunas propuestas de reconfiguración remota y la selección de una de ellas para el FPGA XC3S100E de la familia Spartan 3E de Xilinx, utilizado como unidad de procesamiento del Sistema mínimo

Introducción

Una vez que un satélite ha sido puesto en órbita es sumamente costoso realizar reparaciones físicas o modificaciones posteriores, por lo cual se puede decir que es en la mayoría de los casos prácticamente imposible. Por ello existe una gran necesidad de integrarles recursos necesarios para actualizar de forma remota sus arquitecturas de cómputo.

Los dispositivos FPGA permiten la reconfiguración y evolución de los diseños en *hardware*, integrados a partir de núcleos de procesamiento y control programados en lenguaje descriptor de *hardware* HDL. En la actualidad, los FPGAs son dispositivos idóneos para la implementación de diseños complejos basados en sistemas en un *chip* (SoC).

Por otro lado, la tendencia de los requerimientos de la industria espacial incluye el desarrollo de misiones satelitales en un menor tiempo, reduciendo costos y espacio en *hardware*, cumpliendo tareas cada vez más complejas y con una arquitectura mucho más flexible.

Al considerar a los dispositivos de tecnología FPGA como nicho de oportunidades de desarrollo en el campo espacial, surge en el IINGEN la propuesta de realizar la actualización del subsistema de control de orientación del satélite educativo SATEDU, con el principal objetivo de desarrollar un nuevo subsistema de uso espacial, además de servir como una herramienta didáctica para el entrenamiento y formación de recursos humanos en el área satelital.

SATEDU actualmente cuenta con un subsistema para procesar pequeños algoritmos de control de orientación, el cual está basado en un microcontrolador de propósito general de 8 bits, de arquitectura fija, sobre el cual se programan los algoritmos de control de orientación y la generación de comandos, los cuales se envían hacia *drivers* de potencia que controlan a una sola rueda inercial. Sin embargo, las necesidades de SATEDU contemplan el uso de tres ruedas inerciales, además de algoritmos complejos y pesados de control de orientación satelital en tres ejes. El aspecto de *hardware* de esta solución se aborda en esta tesis.

El esquema completo del nuevo subsistema de control de orientación, contempla el desarrollo de un sistema de procesamiento dedicado y de alto desempeño que aprovecha la ventaja en cuanto a la flexibilidad de la tecnología FPGA para el desarrollo y validación de esquemas de control de orientación. Los cuales constituyen sistemas embebidos que integran núcleos IP (procesadores de alto desempeño y periféricos tanto propios como de pago). De igual forma, el nuevo subsistema persigue la reconfiguración total del FPGA de forma remota vía radio módems, lo cual facilitará la realización de pruebas experimentales sobre una plataforma de simulación de vuelo satelital basada en una mesa suspendida (basada en un cojinete de aire) con la que cuenta actualmente el Instituto de ingeniería de la

UNAM para la validación práctica de estrategias de control de orientación para satélites pequeños.

Las propuestas que se presentan en este capítulo, tienen un microcontrolador de propósito general que funge como elemento de gestión de tráfico de datos y de carga al FPGA de los archivos de reconfiguración enviados desde una estación terrena. En cuanto al medio de comunicación utilizado se optó por módems de RF, debido a la flexibilidad que añade al sistema al instalarse en el simulador satelital de forma inalámbrica y permitir el envío y recepción de datos, así como comandos de control entre la tarjeta electrónica y una PC, que hace las veces de estación terrena y donde es posible realizar posteriores análisis cuantitativos de los procesos realizados y datos obtenidos.

Las siguientes estrategias se consideraron como base para el diseño del sistema desarrollado en esta tesis:

- Uso de un módulo de procesamiento (microcontrolador), el cual llevará a cabo el control del proceso de reconfiguración del FPGA.
- Almacenamiento de la ráfaga de datos (*bitstream*) de reconfiguración en una memoria a bordo. Esto permite tener un respaldo del *bitstream* que fue cargado al FPGA por cualquier eventualidad que se presente.
- Utilizar la UART de un microcontrolador para realizar la comunicación con la Estación terrena e interactuar para la transferencia de datos.

En los siguientes apartados se presentan las generalidades de algunos de los esquemas más comúnmente utilizados para realizar el proceso de carga del *bistream* de configuración al FPGA. Posteriormente a la descripción de cada uno de los esquemas y de los componentes asociados en *hardware* y *software*, se toma la decisión respecto a cuál esquema es el más adecuado para esta aplicación.

4.1 Esquema de reconfiguración a través del protocolo JTAG

El modo de configuración usando el protocolo JTAG se utiliza comúnmente para programar y configurar dispositivos tales como microcontroladores, memorias y dispositivos lógicos programables (CPLDs y FPGAs) respectivamente. En el caso de estos últimos se emplea también como medio de control y gestión de la descarga del *bitstream* de configuración que contendrá unidades de procesamiento o sistemas de prueba, entre otros. La función de los sistemas de prueba es justamente realizar pruebas en cadena de cada uno de los dispositivos programados en el dispositivo. Estas pruebas se realizan para verificar el buen funcionamiento de cada uno de ellos, además de configurarlos de forma

independiente. Este proceso se lleva a cabo conectando uno o varios dispositivos programables en cadena, donde la terminal de salida del primer dispositivo se encuentra conectada a la terminal de entrada del dispositivo siguiente y así sucesivamente, finalizando con la salida del último dispositivo, la cual se conecta a una terminal de entrada del módulo de control. En la figura 4.1 se muestra el esquema general de conexión de la cadena JTAG, utilizando un microcontrolador como dispositivo que aloja el módulo de control y gestión de tráfico de señales JTAG.

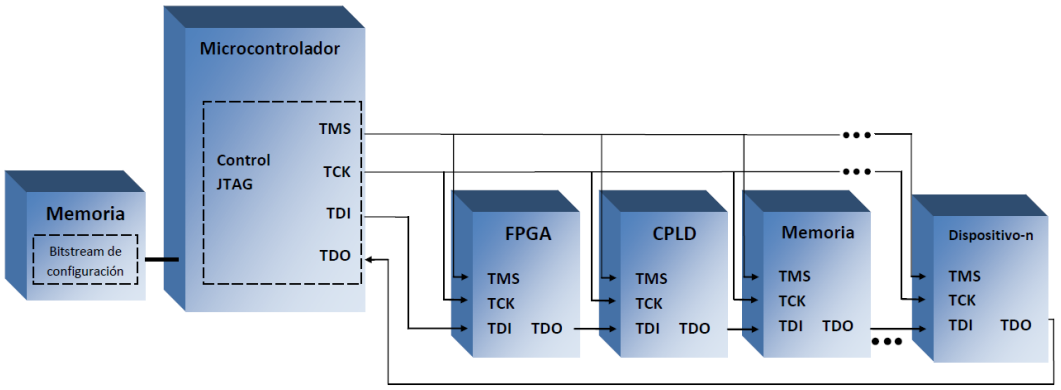


Figura 4.1. Esquema de conexión de la cadena JTAG utilizando un microcontrolador.

Como se puede apreciar en la figura 4.1, el protocolo JTAG está integrado por 4 señales, de las cuales 3 se utilizan como señales de entrada al FPGA, llamadas TDI, TMS y TCK, y una como señal de salida llamada TDO. TMS es la señal de control y TCK es la señal de reloj, ambas comunes a todos los dispositivos conectados a la cadena JTAG. TDI es la señal de entrada de datos y TDO es la señal de respuesta del dispositivo programable, se conectan para cerrar así la cadena JTAG.

Una de las ventajas que se obtiene al llevar a cabo la reconfiguración utilizando el protocolo JTAG, radica en que se pueden conectar varios dispositivos programables usando las mismas líneas de configuración, donde cada dispositivo puede configurarse incluso de forma independiente. Esto reduce significativamente los recursos utilizados del microcontrolador utilizado como gestor de carga y de control de tráfico de señales JTAG, principalmente en el número de terminales de entrada/salida requeridas y el espacio de memoria utilizado para su *firmware*.

También se reduce el espacio que ocupa el *hardware* al disminuir el número de líneas de ruteo que se utilizan en caso de requerir configurar más de un dispositivo dentro de la misma tarjeta. De igual modo, disminuye el número de elementos en *hardware* para la gestión y el cargado del *bitstream* a sólo un dispositivo dentro de la tarjeta. Esto se debe a que todo el control para el flujo de comandos y datos que se cargan al FPGA se encuentra embebido en el *firmware* del microcontrolador.

Otra ventaja que se ha encontrado al usar este modo de configuración, es el manejo de archivos con formato .xsvf, donde se encuentra la arquitectura de cómputo diseñada y donde se tienen además los comandos de control para realizar la configuración por JTAG. Estos archivos son de tamaño reducido en comparación con los archivos .MCS utilizados para almacenar el *bitstream* de reconfiguración en una memoria *Flash* PROM de Xilinx. Cabe señalar que los archivos .xvsf llegan a ser aproximadamente de un tercio de la magnitud de un archivo .MCS (utilizado para almacenar el *bitstream* de reconfiguración en una memoria *Flash* PROM), gracias a lo cual es posible reducir el tiempo de transmisión y de recepción del *bitstream* de forma significativa.

Una vez conocidas las características y tamaño del tipo de archivos que se pueden utilizar para este modo de configuración, se decidió utilizar como dispositivo de almacenamiento una memoria EEPROM, ya que el *bitstream* requerido para configurar al FPGA utilizado en esta tarjeta requería menos de 1Mbit de espacio, por lo tanto este tipo de memoria cubriría las necesidades y requerimientos de la aplicación.

Para el diseño de este esquema de reconfiguración, se requieren los siguientes dispositivos: un FPGA en donde se programan los algoritmos de control que serán validados; un microcontrolador para controlar el flujo de datos cargados en el FPGA que además almacenará el *bitstream* de reconfiguración en una memoria; una memoria EEPROM para almacenar el *bitstream*; un transceptor RS232-TTL que acoplará el nivel de voltaje de los datos enviados de estación terrena de forma serial por medio del radio módem al nivel de voltaje utilizado en las terminales de los puertos del microcontrolador; *hardware* de potencia, para suministrar energía a todos los dispositivos; *hardware* de soporte, para garantizar el buen funcionamiento de todo el sistema en general, al igual que *hardware* de depuración para validar el desempeño del proceso de reconfiguración remota. En la figura 4.2 se muestra el esquema diseñado para la configuración JTAG.

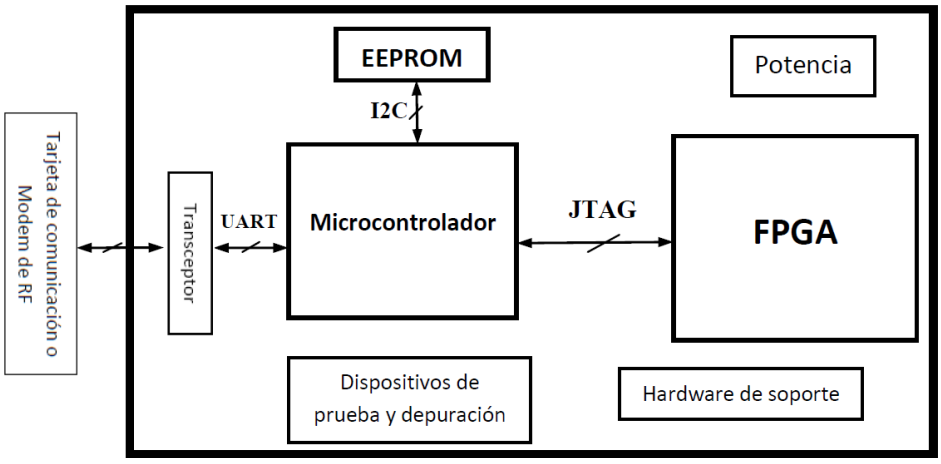


Figura 4.2. Esquema de reconfiguración en modo JTAG.

Hay que señalar que el esquema de la figura 4.2 está basado en la información presentada en la nota de aplicación XAPP058 de Xilinx, la cual además de la información, contiene una propuesta de código en C para configurar al FPGA a través del puerto paralelo de la PC. Esta configuración se validó utilizando el FPGA de la tarjeta de desarrollo Spartan 3E-Starter kit para. Una vez realizada esta experiencia, se propuso nuestro propio *hardware* y una versión adaptada del *software* de control para el *firmware* de nuestro microcontrolador que funciona como gestor de carga del bitstream al FPGA.

4.2 Esquemas de reconfiguración a través del protocolo serie

Otra opción que se usa frecuentemente para la configuración de dispositivos FPGA es por medio del protocolo serie, utilizando como medios de control y gestión de la descarga del *bitstream* de configuración a unidades de procesamiento o el *Flash* PROM de Xilinx.

Este protocolo utiliza 4 señales para realizar la reconfiguración del FPGA, llamadas CCLK, DIN, INIT y PROGRAM. La señal DIN es el dato de configuración y PROGRAM es la señal de control para borrar la memoria de configuración interna, y ambas son terminales de entrada al FPGA. Mientras que INIT es la señal de inicialización y es una terminal de salida. CCLK es la señal de reloj y es una terminal bidireccional dentro del FPGA, la cual puede usarse como terminal de entrada para el modo *Slave serial* o como terminal de salida para el modo *Master serial*. En la figura 4.3 se muestra un diagrama de las señales utilizadas para este modo de configuración dentro del FPGA.

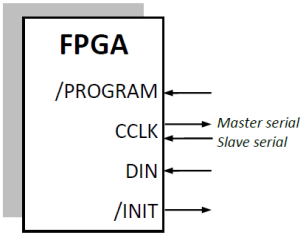


Figura 4.3. Señales de configuración a través del protocolo Serial

4.2.1 Reconfiguración en modo Master Serial

Este modo de configuración se utiliza comúnmente para configurar FPGAs, y utiliza la plataforma *Flash* PROM.

Se llama modo *Master serial* debido a que el FPGA sincroniza las señales de configuración provenientes de la plataforma *Flash* PROM, suministrándole la señal de reloj (CCLK), por

lo tanto se dice que el FPGA se encuentra en modo maestro ya que genera una señal de reloj que sincroniza el proceso de configuración, al igual que indica a la plataforma con la terminal INIT el inicio y el fin de éste.

A las memorias XCFxxS se les denomina plataformas *Flash* PROM ya que contienen embebida la lógica de control, tanto para almacenar el *bitstream* de configuración en formato .MCS, a través de una interfaz JTAG, como para extraer el mismo a través de una interfaz serie. En la figura 4.4 se muestra un diagrama en bloques de la arquitectura interna de la plataforma *Flash* PROM.

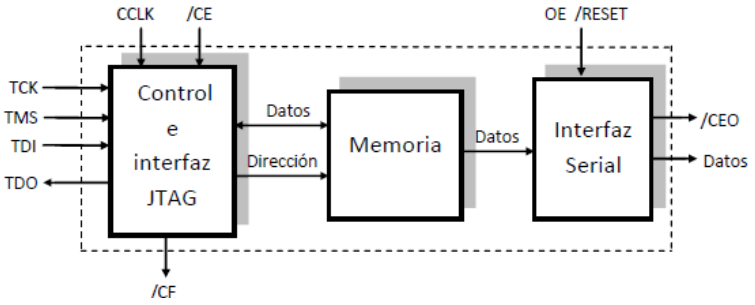


Figura 4.4. Diagrama de bloques de la plataforma Flash PROM.

Gracias a las funciones que caracterizan a este tipo de memorias, es posible realizar la configuración de un FPGA sin utilizar un procesador (microcontrolador) que gestione la descagra del *bitstream* hacia el FPGA. La función del procesador dentro de este esquema de configuración consiste en almacenar el *bitstream* en la memoria *Flash* PROM a través del protocolo JTAG, así como indicar al FPGA el momento en que puede inicializar el proceso de configuración. En la figura 4.5 se muestra un diagrama de bloques de las conexiones necesarias para este proceso, en el cual se utiliza un microcontrolador como dispositivo de almacenamiento del *bitstream* en la memoria.

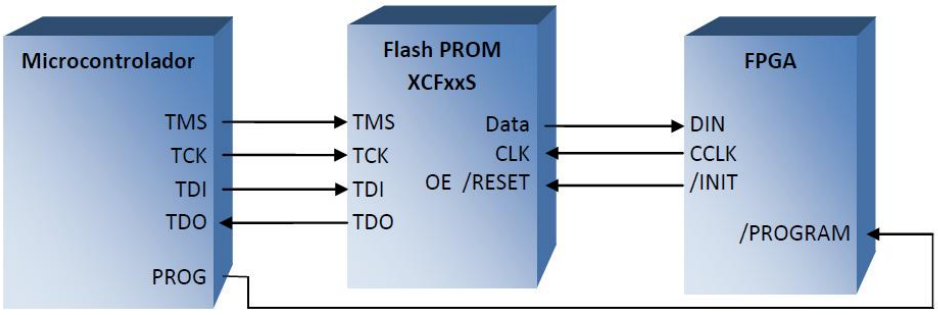


Figura 4.5. Diagrama general de conexiones en modo Master serial.

La desventaja de trabajar con este modo de configuración es el uso de los archivos en formato .MCS, ya que como se mencionó para el esquema de reconfiguración usando el

protocolo JTAG, este archivo es de una magnitud considerablemente mas grande en comparación con los archivos .xsvf, aumentando con esto el tiempo de transmisión y almacenamiento del *bitstream* en la memoria. Un archivo .xsvf es aproximadamente un tercio de la magnitud de un archivo .MCS.

Para el diseño de este esquema de reconfiguración, se requieren los siguientes dispositivos: un FPGA; un microcontrolador que almacenará el *bitstream* dentro de la memoria; una memoria *Flash* PROM para almacenar el *bitstream* y posteriormente para descargarlo hacia el FPGA; un transceptor RS232-TTL*; hardware* de potencia; *hardware* de soporte, al igual que *hardware* de prueba y depuración. En la figura 4.6 se muestra el esquema diseñado para la configuración en modo *Master serial*.

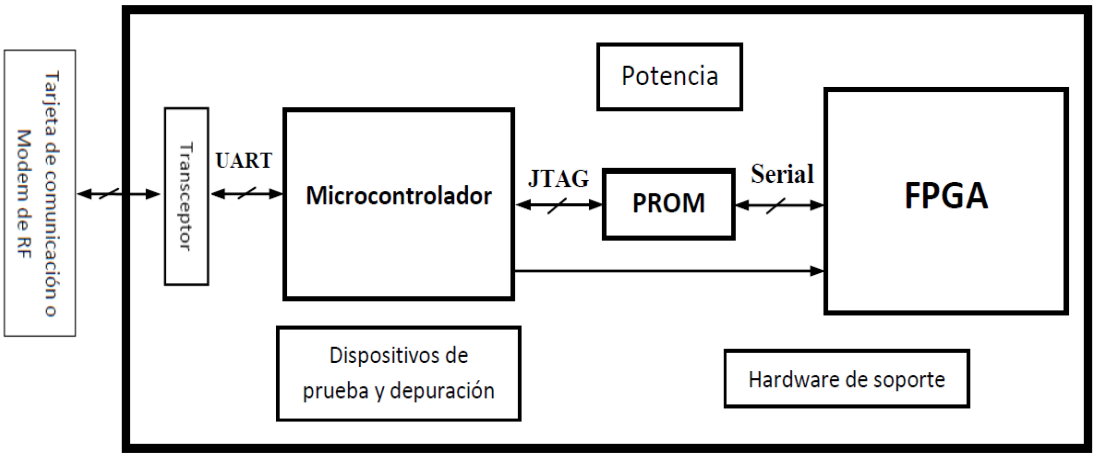


Figura 4.6. Esquema de reconfiguración en modo Slave serial.

Cabe señalar que en la bibliografia hay muy pocas referencias respecto al uso de este esquema, material que nos pudiera orientar para realizar la programación de la memoria *Flash* PROM y el posterior proceso de configuración al FPGA. Esto dificulta el desarrollo del *firmware* para programar la memoria ya que se requiere diseñarlo por completo, por lo tanto, llevaría un mayor tiempo de desarrollo y validación la creación de un módulo de reconfiguración funcional con este tipo de características.

4.2.2 Reconfiguración en modo Slave Serial

El modo de configuración *Slave Serial* se utiliza comúnmente para configurar FPGAs, utilizando uno o varios procesadores.

Este modo de configuración utiliza archivos en formato .bit, estos archivos contienen la arquitectura de cómputo diseñada y al igual que los archivos .xsvf son del mismo tamaño, por lo que para este esquema se llegó a la determinación de utilizar como dispositivo de almacenamiento una memoria EEPROM.

Explicado lo anterior, en seguida se presentan dos propuestas de reconfiguración para la tarjeta de control de orientación en tres ejes de SATEDU, las cuales presentan ciertas ventajas distintas una de la otra.

4.2.2.1 Esquema de reconfiguración de FPGA basada en CPLD y microcontrolador

En este primer esquema se utiliza como dispositivo principal de procesamiento a un CPLD para reconfigurar al FPGA, además se utiliza un microcontrolador como dispositivo de procesamiento secundario. Este esquema se utiliza principalmente en sistemas donde el microcontrolador realiza múltiples tareas que nada tienen que ver con el proceso de configuración, las cuales ocupan más del 75% de sus recursos internos. En esta arquitectura, el CPLD ocupa un papel muy importante, ya que sirve como interfaz síncrona entre el microcontrolador y el FPGA para enlazar y controlar el flujo de señales y la lógica entre ambos dispositivos.

La interfaz integrada en el CPLD está compuesta por tres registros: el registro de configuración, el registro de programación y el registro de entrada. Estos registros almacenan las señales de configuración cada vez que el microcontrolador realiza la lectura o escritura en el puerto. El diagrama de conexión al igual que los registros internos del CPLD, se muestran en la figura 4.7.

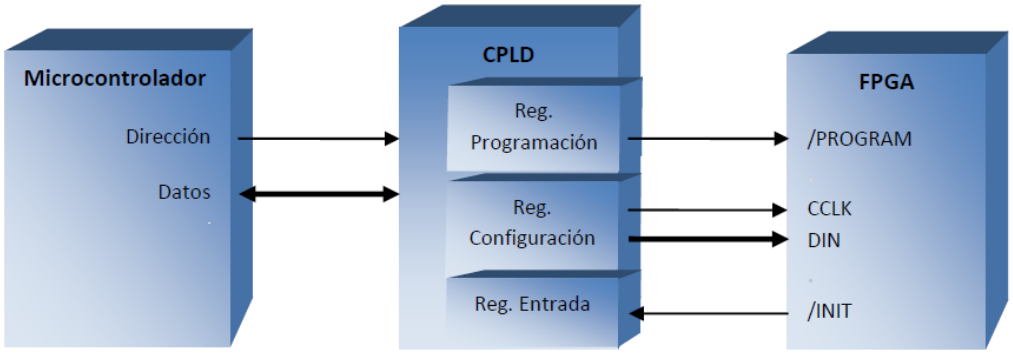


Figura 4.7. Esquema de conexión para configuración en modo Slave serial

Como se puede observar en la figura 4.7, el microcontrolador utiliza pocas terminales de interconexión con el CPLD, ya que tanto las señales de dirección y de datos de transmiten en serie, lo cual libera al microcontrolador de recursos en *hardware* (terminales i/o), las cuales pueden ser utilizados para otras tareas. También puede deducirse, que el CPLD realiza la mayor parte del trabajo, ya que él lleva el proceso de control para la configuración serie, gracias a esto también se reduce significativamente el espacio en el *firmware* del microcontrolador.

Para el diseño de este esquema de reconfiguración, se integran los siguientes dispositivos: un FPGA; un microcontrolador que envía el *bitstream* de reconfiguración con sus respectivas direcciones además de almacenar el *bitstream* de reconfiguración dentro de una memoria; un CPLD, que controla el flujo de datos hacia el FPGA; una memoria EEPROM; un transceptor RS232-TTL; *hardware* de potencia; *hardware* de soporte, al igual que *hardware* de prueba y depuración. En la figura 4.8 se muestra el esquema diseñado para la configuración en modo *Slave serial*.

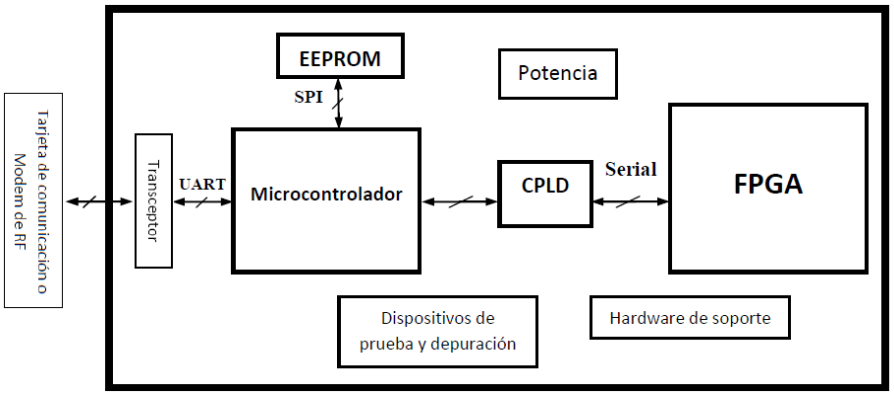


Figura 4.8. Esquema de reconfiguración en modo Slave serial.

Cabe señalar que, no sólo se añade al esquema un dispositivo CLPD, sino que también se agrega su respectivo *hardware* de soporte, lo que incrementa el espacio del *hardware* del sistema de reconfiguración dentro de la tarjeta de sistema mínimo.

Hay que señalar que el esquema mostrado en la figura 4.8 está basado en la información presentada en la nota de aplicación XAPP502 de Xilinx, la cual además de la información, contiene una propuesta de código en C para un microcontrolador, y el código en VHDL para un CPLD, donde los dos en conjunto realizan la reconfiguración del FPGA, a través de terminales de propósito general de dichos dispositivos.

4.2.2.2 Esquema de reconfiguración de FPGA basada un microcontrolador

Este esquema, a diferencia del primero, se utiliza para reconfigurar a un FPGA con un sólo procesador para realizar todo el proceso de configuración. Este esquema se utiliza principalmente en sistemas que requieren que procesador realice muy pocas o ninguna tarea adicional a la de reconfiguración. Aquí el procesador controla todas las señales enviadas al FPGA para su reconfiguración. En la figura 4.9 se muestra el esquema de conexiones para configurar a un FPGA utilizando un microcontrolador como dispositivo de procesamiento.

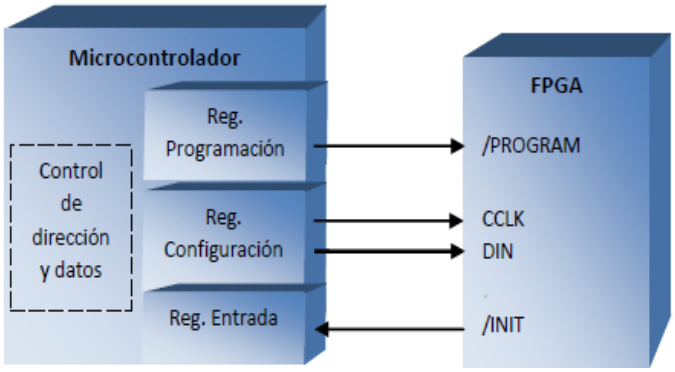


Figura 4.9. Esquema de conexión para configuración en modo Slave serial.

Como se puede observar en la figura 4.9, dentro del microcontrolador se encuentran los registros necesarios para configurar al FPGA, por lo tanto utiliza más recursos en *software* que la propuesta del primer esquema. También podemos observar que se necesitan usar más recursos en *hardware*, ya que requiere de un mayor número de terminales de propósito general para establecer comunicaciones con el FPGA.

Para el diseño del esquema de reconfiguración, se contemplarían los siguientes dispositivos: un FPGA; un microcontrolador, el cual controlará el flujo de datos para reconfigurar el FPGA y el almacenamiento del *bitstream* en la memoria; una memoria EEPROM; un transceptor RS232-TTL; *hardware* de potencia; *hardware* de soporte al igual que *hardware* de prueba y depuración. En la figura 4.10 se muestra el esquema propuesto.

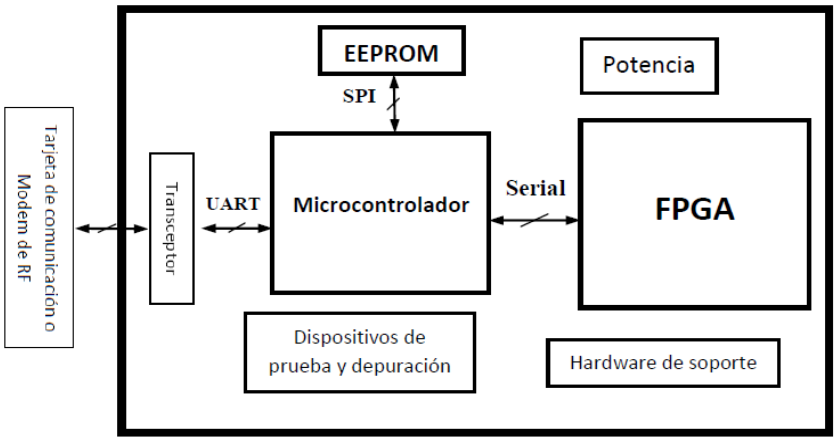


Figura 4.10. Esquema de reconfiguración en modo Slave serial.

Como se observa en la figura 4.10, a diferencia de la primer propuesta, este esquema emplea menos *hardware*, lo que lo hace una buena opción si se requiere un sistema de dimensiones reducidas y a la medida, donde la única función del microcontrolador es almacenar el *bitstream* en memoria y la reconfiguración del FPGA, y este, a su vez, es el encargado de realizar las tareas propias del sistema por desarrollar o automatizar.

Cabe señalar que en la bibliografia hay muy pocas referencias respecto del uso de este esquema, material que nos pudiera orientar para realizar el proceso de configuración al FPGA. Esto dificulta el desarrollo del *firmware* para configurar el FPGA ya que se requiere diseñar el *firmware* por completo, por lo tanto, la creación de un módulo de reconfiguración funcional con este tipo de caracteristicas llevaría un mayor tiempo de desarrollo y validación.

4.3 Análisis de las Propuestas

Después de un análisis en términos costo-beneficio de las opciones expuestas en este capítulo, el esquema de reconfiguración que utiliza el protocolo JTAG resulta el más adecuado para nuestra aplicación. No sólo por la factibilidad técnica que representa el ahorro de espacio físico en la tarjeta de circuito impreso y la disminución del tiempo de transmisión y recepción de la arquitectura de computo, sino también por toda la documentación que se tiene disponible al respecto, particularmente notas de aplicación del fabricante y *software* libre que sirve como base para acelerar el desarrollo de un sistema propio, el cual utiliza un microcontrolador de propósito general como módulo de procesamiento y control para realizar todas y cada una de las tareas asociadas con el proceso de reconfiguración remota del FPGA. Finalmente, una tarea importante en esta vertiente de desarrollo del trabajo de tesis en materia de reconfiguración remota, consistió en realizar la transferencia del *software* diseñado para la PC a un *firmware* que pudiera ser utilizado en un microcontrolador de propósito general.

Una vez elegido el modo de configuración, y habiendo definido una arquitectura de trabajo, el siguiente paso a seguir fue diseñar una propuesta innovadora de una tarjeta de sistema mínimo, basada en un dispositivo FPGA, con los recursos necesarios para evaluar el funcionamiento de diferentes arquitecturas de cómputo. En el siguiente capítulo se aborda el tema sobre el desarrollo del esquema propuesto que contendrá el *hardware* y *software* necesario para levar a cabo el proceso de reconfiguración remota del FPGA.

CAPÍTULO 5

Propuesta de desarrollo de una plataforma reconfigurable, con hardware dedicado.

El desarrollo de sistemas satelitales ha tenido asociados una serie de requerimientos particularmente relacionados con la integración de recursos flexibles, reducción en tamaño, masa y consumo de energía. Las tendencias actuales en el campo de desarrollo de los satélites pequeños están orientadas a la integración de varias funciones en una misma plataforma de cómputo. De igual forma, el avance en la tecnología de circuitos integrados, como los dispositivos lógicos programables, han permitido extender la gama de posibilidades de integración de sistemas completos de diferentes niveles de complejidad.

En este capítulo se expone el planteamiento de una propuesta, en *hardware* y *software*, que integra entre otros, un dispositivo FPGA, que ofrece una alternativa flexible para el desarrollo de sistemas embebidos y cuyo espectro de aplicación requiere una plataforma de desarrollo sumamente versátil.

5.1 Propuesta de sistema FPGA mínimo en una tarjeta (SMIN)

Como se ha tratado en los capítulos anteriores, el contar con un sistema basado en un FPGA dota al sistema de gran flexibilidad en términos de diseño digital y de una capacidad de procesamiento aún mayor que los dispositivos tradicionales de arquitectura fija.

La versión de sistema FPGA mínimo que se propone en este capítulo, está integrado por un FPGA, una serie de módulos tales como transceptores para comunicaciones seriales, alimentación de energía, memoria, interfaces digitales de entrada/salida y otros dispositivos electrónicos de soporte. Adicionalmente, el paradigma bajo el que se diseña el prototipo, contempla la integración de un módulo que permitirá por un lado la reconfiguración del FPGA y por otro, validar dinámicamente la recarga de nuevas arquitecturas de cómputo de forma remota utilizando para ello un microcontrolador de propósito general y una memoria de estado sólido.

El diseño de sistemas basados en FPGAs y, en general, sistemas que contemplen dispositivos de alta escala de integración con un gran número de terminales físicas, y encapsulados de montaje superficial de dimensiones mínimas, representan un gran reto, no sólo en el diseño mismo de la tarjeta de circuito impreso, sino en su ensamble y puesta a punto.

En términos generales de diseño, uno de los principales retos de esta propuesta fue lograr el acoplamiento correcto de las líneas E/S de cada uno de los componentes de la tarjeta electrónica, por las que transitan señales de datos y lógica de control. Particularmente entre FPGA y los módulos de comunicaciones, reconfiguración remota y memoria, básicamente por las restricciones establecidas en los niveles de impedancia de entrada, corriente y niveles de voltaje.

La propuesta que se presenta en este capítulo, considera diagramas de bloques diseñados en dos grandes segmentos: estación terrena y tarjeta SMIN.

El diagrama de la estación terrena incluye cada una de las operaciones que se ejecutan para llevar a cabo la transmisión del *bitstream* de reconfiguración, utilizando una interfaz gráfica de usuario (GUI). Estas operaciones se encuentran divididas en bloques ya que se ejecutan en diferentes plataformas de *software*. Estos bloques son:

- **Diseño y síntesis de la arquitectura de cómputo reconfigurable**
  Este bloque utiliza las herramientas de diseño de la suite ISE de Xilinx para el diseño y síntesis de las arquitecturas de cómputo. Después de realizar la síntesis se genera un archivo en formato .bit, el cual es el formato natural de configuración para FPGAs.

- **Creación de un archivo en formato .xsvf**
  En este bloque se genera un archivo .xsvf a partir del archivo .bit, creado al sintetizar la arquitectura, para ello se hace uso de la herramienta iMPACT de la misma suite ISE. Se crea un archivo en este formato, ya que como se mencionó en el capítulo anterior, el *software* proporcionado por Xilinx lleva a cabo la configuración del FPGA utilizando un archivo con este tipo de formatos.
- **Creación de un archivo en formato Intel HEX**
  En este bloque se crea un archivo en formato Intel HEX a partir del archivo .xsvf creado previamente, este proceso puede efectuarse utilizando cualquier compilador de lenguaje C. La finalidad de utilizar un archivo con este formato, es debido a que este tipo de formatos ayuda a tener un control sobre la transferencia de datos de forma serial y garantiza la transmisión y recepción exitosa de éstos.
- **Interfaz grafica de usuario GUI**
  Este bloque permite tomar el control del puerto serie de la PC para transmitir el archivo de configuración, así como recibir datos para verificar el estado operativo de algunos de los dispositivos a bordo de la tarjeta, como por ejemplo la memoria de estado sólido, e indicar cada una de las tareas que debe ser ejecutada por el microcontrolador para llevar a cabo el proceso de reconfiguración remota del FPGA.

En cuanto al diagrama de la tarjeta SMIN, está integrado por tres bloques principales:

- **Bloque de recepción, transmisión, almacenamiento en memoria y reconfiguración del FPGA (BTRAR)**
- **Bloque de acondicionamiento y potencia (BAP)**
- **Bloque entradas y salidas digitales (BESD)**

En la figura 5.1 se muestra el esquema propuesto, el cual integra los dos diagramas antes mencionados.

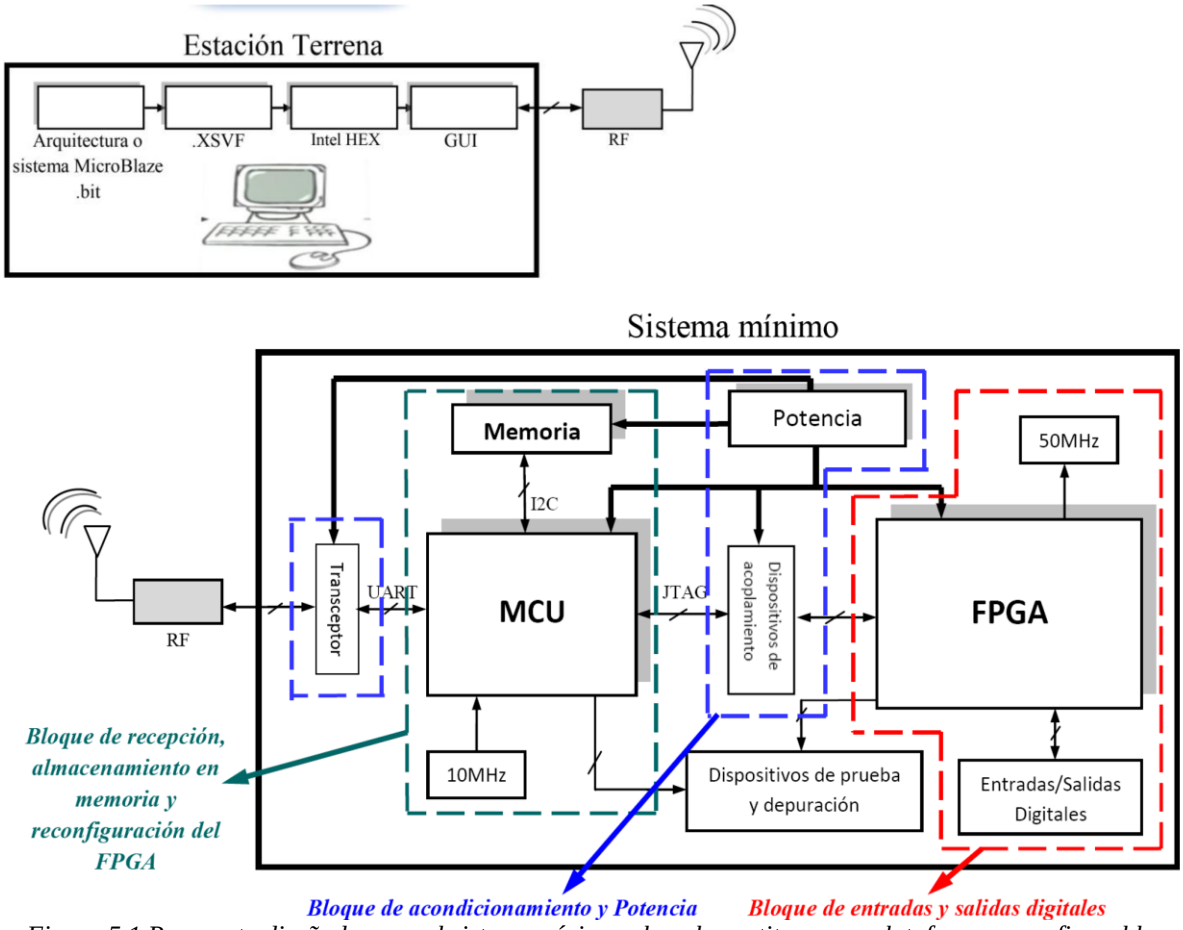


Figura 5.1 Propuesta diseñada para el sistema mínimo, el cual constituye una plataforma reconfigurable

Como se puede observar en la figura 5.1, se integra un módulo donde se incluyen componentes de apoyo a la depuración de *software*, integrado básicamente por LEDs e interruptores deslizables (*dip switch*). Este módulo no es parte de algún bloque específico de la tarjeta, ya que es común para todos los bloques que la integran.

Los componentes del módulo de prueba y depuración para el BAP, indican la activación y desactivación de los reguladores de voltaje responsables de suministrar los diferentes niveles de voltaje para cada uno de los componentes a bordo. Para el BTRAR, indican la ejecución de los procesos de almacenamiento y lectura de la memoria, así como la reconfiguración del FPGA. Para el BESD, se encargan de validar la descarga exitosa del *bitstream* de reconfiguración del FPGA, lo que ayuda a monitorear visualmente si el FPGA ha sido configurado de forma adecuada.

En los siguientes párrafos se describe el funcionamiento y las características de cada bloque integrado en este esquema.

5.2 Bloque de recepción, almacenamiento en memoria y reconfiguración del FPGA

Este bloque integra el esquema de reconfiguración propuesto en el capítulo anterior que utiliza un MCU (PIC de Microchip), una memoria EEPROM para el respaldo del *bitstream* de reconfiguración y un cristal de 10MHz como base de tiempo para el MCU.

El uso de un microcontrolador en la unidad de control obedece a que este dispositivo ya cuenta con interfaces para comunicación como USART, I2C, SPI, etc, los cuales pueden habilitarse con facilidad para permitir el flujo de datos utilizando algunos de éstos protocolos de comunicación, ello reduce el tiempo de desarrollo para el *firmware* que controlará el proceso de reconfiguración remota del FPGA.

Para fines de diseño e integración, el *firmware* interno del PIC fue dividido en dos módulos. La estrategia de formar el *firmware* en bloques permite validar estos módulos individualmente. En caso de existir problemas en el funcionamiento del *firmware*, es más sencillo detectar los errores de programación analizando bloques separados, máxime si el *firmware* contiene menos librerías donde pudiera presentarse un comando erróneo. Una vez validado el funcionamiento de cada uno de estos módulos, se integran para formar un sólo *firmware* y así poder realizar las pruebas finales y validar el proceso de reconfiguración remota en su totalidad. Los módulos en los que se encuentra dividido el *firmware* son:

- Módulo de recepción, transmisión y almacenamiento en memoria.
- Módulo de reconfiguración del FPGA.

El diagrama de bloques del *firmware* interno del MCU se muestra en la figura 5.2.

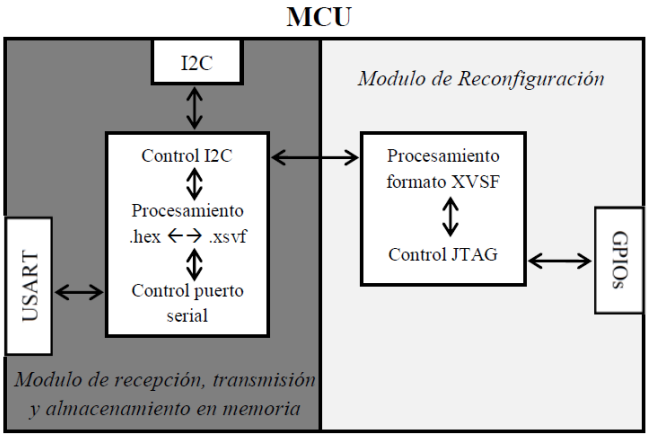


Figura 5.2. Firmware embebido en el PIC.

Como se puede observar en la figura 5.2, cada módulo del *firmware* tiene múltiples funciones, las cuales serán detalladas en las siguientes secciones.

5.2.1 Módulo de recepción, transmisión y almacenamiento en memoria

Este módulo se encarga de recibir comandos de control para seleccionar cada una de las tareas a ejecutar, al igual que recibir y transmitir de en serie, el *bitstream* de configuración que será cargado en el FPGA. Dentro de estos comandos de control se encuentran los siguientes.

Almacenamiento del bitstream en memoria

Aquí el PIC recibe por el puerto serie el *bitstream* transmitido desde la estación terrena en formato Intel HEX, después pasa por una etapa de procesamiento, convirtiendo este archivo a su formato original xvsf. En esta etapa de procesamiento, primero se decodifica el *bitstream* y posteriormente se convierte cada bit a entero, ya que el formato Intel HEX contiene el *bitstream* en código ASCII, este cambio se debe a que el PIC no opera con este tipo de datos. Una vez procesado el *bitstream* a través del puerto I2C del PIC, envía los comandos de control indicados hacia la memoria EEPROM y los almacena.

Lectura del bitstream de memoria

Al recibir este comando, el PIC comienza a extraer el *bitstream* almacenado en la memoria y pasa por un proceso similar al antes mencionado, con la diferencia de que aquí se convierte nuevamente a formato Intel HEX, para transmitirlo hacia estación terrena a través del puerto serie.

Borrado de memoria

Al recibir este comando el PIC envía una trama de bits específica, con la cual se elimina la información almacenada en memoria.

5.2.2 Módulo de Reconfiguración

Este módulo se encarga de extraer el *bitstream* almacenado en la memoria. *Byte* a *byte* procesa y ejecuta las instrucciones integradas en el archivo xsvf, las cuales controlan las terminales de propósito general del PIC para generar las señales JTAG, que se conectan hacia el puerto de configuración del FPGA. También procesa los datos recibidos provenientes del FPGA, con lo que cierra la cadena JTAG para llevar a cabo el proceso de reconfiguración de manera exitosa.

5.3 Bloque de acondicionamiento y potencia

Este bloque se encuentra dividido en 2 etapas, cada etapa por su parte tiene diferente función. Sus funciones van desde suministrar la potencia necesaria para el funcionamiento óptimo de los dispositivos que componen la tarjeta, hasta el acondicionamiento de los niveles de voltaje entre dispositivos para establecer una comunicación y transferencia de datos entre ellos.

5.3.1 Potencia

La etapa de potencia está compuesta por cuatro reguladores de voltaje que suministran energía eléctrica a los dispositivos de la tarjeta, éste bloque de potencia permite que la tarjeta pueda emplearse dentro o fuera de SATEDU. Los niveles de voltaje que suministra son 5V, 3.3V, 2.5V y 1.2V.

Los FPGAs utilizan diferentes niveles de voltaje para operar, de éstos la familia Spartan 3E opera con 2 niveles de voltaje para realizar funciones lógicas internas, donde las terminales VCC_{INT} y VCC_{AUX} indican las entradas de alimentación donde se suministran estos voltajes. La entrada de alimentación VCC_O se utiliza en cada uno de los cuatro bancos de entrada/salida del dispositivo, la cual suministra la energía hacia los *buffers* de salida que están asociados a los IOBs de cada banco y establece el umbral de voltaje de entrada para las señales provenientes de algún otro dispositivo. Estos bancos se encuentran en los bordes del dispositivo, como se muestra en la figura 5.3.

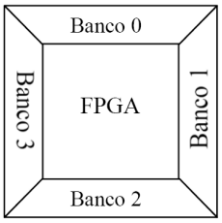


Figura 5.3. Distribución de los bancos del FPGA.

Las terminales VCC_{INT} constituyen la alimentación de la lógica del núcleo interno del FPGA, estas terminales utilizan un voltaje nominal de 1.2V, además, son la fuente de alimentación de todas las funciones lógicas tales como CLBs, Bloques de RAM y multiplexores.

Las terminales VCC_{AUX} son líneas de alimentación auxiliar que utilizan un voltaje nominal de 2.5V, suministran energía a los DCMs, controladores diferenciales, pines dedicados para configuración y para la interfaz JTAG.

Las terminales VCC_O antes mencionadas, están designadas para la alimentación de los IOBs de cada banco, estas terminales pueden utilizar diferentes niveles de voltaje, los cuales comprenden 1.2V, 1.5V, 1.8V, 2.5V y 3.3V. Para cada banco del FPGA se designa un valor numérico en las terminales (VCC_O_0, VCC_O_1 y VCC_O_2) para señalar hacia qué banco se canaliza la energía suministrada. Una vez seleccionado el nivel de voltaje con el que se requiere que los IOBs operen, se conectan todas terminales VCC_O a la misma alimentación. Los FPGAs tienen la capacidad de operar con diferentes niveles de voltaje en cada uno de sus bancos, pero para ello es necesario agregar al diseño una línea más de alimentación, la cual se conecta a las terminales V_{REF} del FPGA.

En el diseño de la tarjeta electrónica que se describe en este trabajo de tesis, se utilizó el mismo nivel de voltaje para todas las líneas de VCC_O, el cual es de 3.3V. Debido a que algunos dispositivos que integran al sistema, operan a niveles de 3.3V, como los sensores de navegación inercial. Por lo tanto, no se requiere de dispositivos externos para acondicionar los voltajes y llevar a cabo la comunicación entre el FPGA y los sensores.

Tanto el PIC utilizado como la memoria operan a un voltaje que puede oscilar entre 4.5 y 5.5V, por lo tanto fue necesario agregar un regulador de 5V, el cual suministra la potencia necesaria para el buen funcionamiento de estos dispositivos. Este regulador también se utiliza para alimentar a otros reguladores que operarán dentro de la tarjeta.

5.3.2 Acondicionamiento

La etapa de acondicionamiento se divide en dos secciones, las cuales resuelven los problemas de acoplamiento de señales entre estación terrena y el PIC, y entre el PIC y el FPGA.

La primera sección está formada por un transceptor, el cual se usa para acondicionar el nivel de voltaje de las señales de comunicación transmitidas y recibidas por el PIC, a través de los módems de RF o el sistema de comunicaciones, desde el puerto serie de la PC de la estación terrena.

La segunda etapa está formada por un *buffer*, el cual se encarga de acondicionar los niveles de voltaje de las señales de configuración que se conectan entre el PIC y el FPGA.

5.4 Bloque de entradas y salidas digitales

Este bloque está integrado por el FPGA como dispositivo principal donde se descargarán las arquitecturas de cómputo diseñadas. Dichas arquitecturas se encuentran descritas como bloques en *hardware* (núcleos IP), como son: núcleos de control, comunicación con dispositivos externos, algunas operaciones internas, etc. Los núcleos IP fueron desarrollados utilizando la suite ISE en lenguaje VHDL.

Dentro de este bloque existen terminales de entrada salida conectadas a terminales del FPGA que se utilizan para interactuar con dispositivos externos. A través de estas terminales se valida el funcionamiento óptimo de las arquitecturas descargadas.

En este capítulo se presentó la propuesta planteada para contar con una plataforma flexible y novedosa, en la cual sea posible llevar a cabo el proceso de reconfiguración remota de un FPGA. Se describió cada uno de los módulos de los cuales se compone todo el esquema planteado, desde el proceso que debe llevarse a cabo en estación terrena hasta las operaciones que serán ejecutadas por cada dispositivo dentro de la tarjeta.

En el siguiente capítulo se hablará de las herramientas de *hardware* y *software* que fueron utilizadas para llevar a cabo las pruebas de validación, para la puesta a punto de la tarjeta SMIN.

CAPÍTULO 6

Descripción de las herramientas utilizadas para llevar a cabo el proceso de reconfiguración remota

El proceso de diseño y desarrollo de la tarjeta SMIN, constó de una serie de pasos que consideraron el desarrollo en forma modular de cada una de los bloques que integran el sistema, su validación funcional, su posterior integración y puesta a punto. El desarrollo se realizó en las vertientes de *hardware* y *software*. *Hardware* de diseño propio en tarjetas de circuito impreso ensambladas de forma manual en el laboratorio, *software* integrado por interfaces gráficas de tipo consola y ventanas desarrolladas en lenguaje de programación de alto nivel.

En este capítulo se presentan algunos detalles del diseño y construcción de herramientas en *software* y *hardware* utilizadas para la integración del diseño final de tarjeta de SMIN basado en un FPGA, el cual tiene como innovación tecnológica la capacidad de reconfigurarlo de forma remota para instalarlo en un simulador de vuelo satelital.

6.1 Introducción al proceso de reconfiguración remota

Un FPGA puede estar en uno de dos estados: modo de configuración o modo usuario. Cuando se inicializa al FPGA, luego de energizarlo, éste se encuentra en modo de configuración, permaneciendo en un estado inactivo (idle) con todas sus salidas inactivas, hasta que se configura.

Configurar un FPGA significa descargar en él una trama de ceros y unos a través de terminales especiales. Una vez que se configura vuelve a modo de usuario y se activa, realizando la ejecución de tareas de acuerdo a la (las) función (es) lógica (s) que le han sido programadas.

Existen 3 formas clásicas de configurar un FPGA, atendiendo a los recursos en *hardware* que se tengan:

- Para un sistema de desarrollo personalizado y a la medida: utilizando el cable *parallel III* desde la PC hacia un sistema de desarrollo FPGA, y correr un *software* en la PC para enviar datos a través del cable.
- Para un sistema de desarrollo personalizado y a la medida: utilizando un microcontrolador en la tarjeta de desarrollo que contiene al FPGA, con el *firmware* adecuado para el envío de datos al FPGA.
- Para un sistema de desarrollo personalizado y a la medida: usando una *boot-PROM* en la tarjeta (conectada al FPGA) que lo configure de forma automática cuando éste es energizado.

Durante el desarrollo de un sistema basado en FPGA, como el que se describe en esta tesis, el primer método es el más fácil y rápido para la definición de la lógica del proceso de configuración y de los componentes necesarios. Una vez que se ha puesto a punto al sistema FPGA, el uso de una PC para los procesos subsiguientes se vuelve innecesario, pudiendo utilizar cualquiera de los dos métodos antes descritos.

El proceso de configuración funciona de forma similar tanto en dispositivos de Xilinx, como en el caso del sistema que se describe en esta tesis, como en dispositivos de Altera que es otro fabricante de FPGAs de gran presencia en el mercado. Las principales diferencias que existen, radican principalmente en los nombres de las terminales de los dispositivos y en los nombres de los modos de operación, que son diferentes para ambos fabricantes siendo sin embargo, similar la funcionalidad.

Con base en el protocolo de transferencia de datos y en los recursos de *hardware* disponibles, la reconfiguración del FPGA se puede realizar por medio de alguno de los siguientes métodos:

- Utilizando la interfaz JTAG, la cual fue seleccionada para la implementación en el sistema que se describe en los siguientes apartados de este capítulo.

- Utilizando la interfaz síncrona serie.

6.1.1 Protocolo JTAG (Estándar IEEE 1149.1/1532)

6.1.1.1 Antecedentes

JTAG es un estándar IEEE (1149.1) desarrollado en la década de los 80s inicialmente para resolver cuestiones de manufactura de tarjetas electrónicas. Sin embargo hoy en día se usa como protocolo de programación, depuración y prueba.

El puerto de acceso a pruebas IEEE 1149.1 (*Test Access port)* TAP y la arquitectura *boundary-scan,* comúnmente referida como JTAG, es un método popular de prueba en la industria. JTAG es un acrónimo de *Joint Test Action Group,* nombre que adoptó del subcomité técnico inicialmente responsable del desarrollo del estándar. Este estándar provee un medio para asegurar la integridad de los componentes individuales a nivel de tarjeta y sus interconexiones. Con el incremento de la densidad de tarjetas multicapa y de las cada vez más sofisticadas técnicas de ensamblado de circuitos y componentes de montaje superficial, las pruebas *boundary-scan* están llegando a ser ampliamente usadas como un importante estándar de depuración.

Los dispositivos que contienen la lógica *boundary-scan* pueden enviar datos a terminales E/S para probar conexiones entre dispositivos a nivel de tarjeta. La circuitería asociada puede también utilizarse para enviar señales internamente que realicen pruebas sobre dispositivos específicos.

Además de servir como herramienta de prueba, el esquema *boundary-scan* ofrece la flexibilidad para que un dispositivo tenga su conjunto propio de instrucciones definidas por el usuario. Las instrucciones comunes añadidas por el fabricante, tales como configurar o verificar, han incrementado el uso de la arquitectura *boundary-scan* y su funcionalidad.

6.1.1.2 Arquitectura JTAG (IEEE 1149.1/1532)

El estándar IEEE 1149.1 está compuesto por el Puerto de Acceso de Prueba (TAP), el controlador del TAP y una arquitectura *Boundary-Scan,* los cuales se muestran en la figura 6.1.

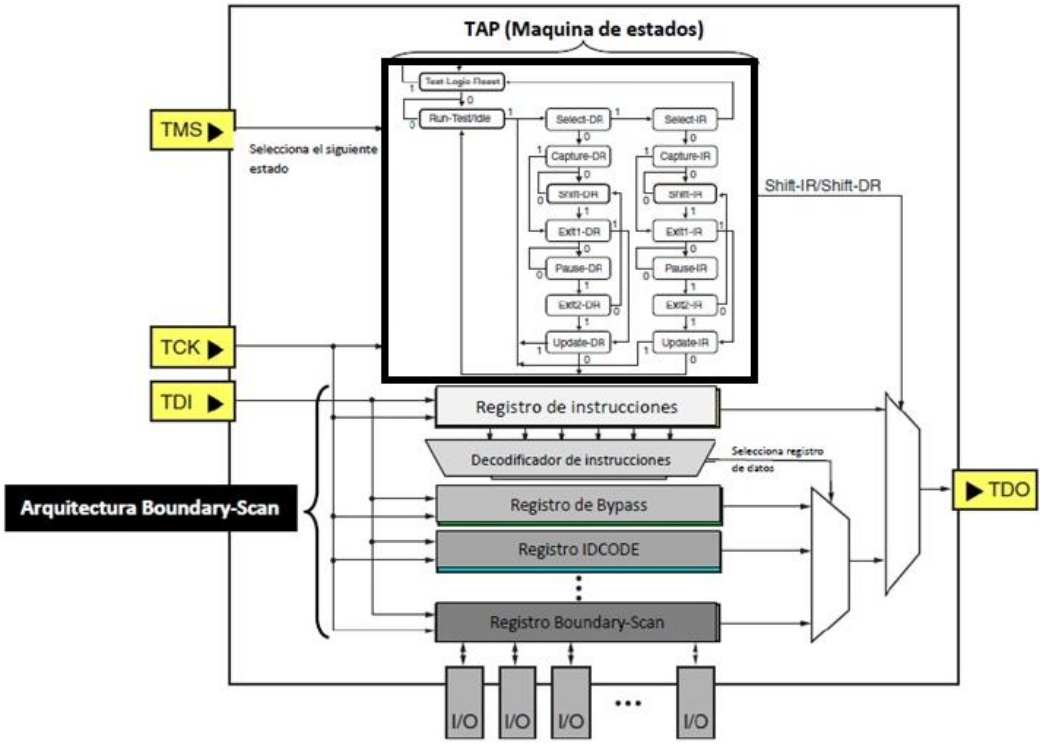


Figura 6.1. Arquitectura JTAG.

El diagrama de la figura anterior muestra el módulo JTAG embebido en un dispositivo programable, cada uno de los bloques, que lo integran, se explican a continuación.

Puerto de acceso de prueba (TAP)

El TAP es un puerto de propósito general, el cual permite el acceso a las funciones integradas en el dispositivo. Se compone de 4 señales: 3 señales de entrada (TMS, TDI y TCK) y una de salida TDO, con una cuarta señal de entrada opcional denominada TRST. Estas señales se describen a continuación.

TDI: Es la señal de entrada para todos los registros de instrucción y datos de JTAG.

TMS: Esta señal determina la secuencia de los estados en el TAP cuando se presenta un flanco de subida en la señal TCK.

TCK: Provee la señal de reloj para el TAP y los registros JTAG.

TDO: Es la señal de salida establecida por los registros de instrucción y datos. Todas las funciones JTAG se llevan a cabo cuando se presenta un flanco de subida de esta señal.

TRST: Esta señal inicializa la comunicación de forma asíncrona, esto quiere decir que cuando el TAP no se restablece correctamente esta señal envía el TAP a su estado inicial.

Controlador del TAP

El controlador del TAP es una máquina de estados (FSM), que se muestra en la figura 6.2, la cual utiliza las señales TCK y TMS para realizar el cambio de estado, mientras que usa TDI para escribir y a TDO para leer datos.

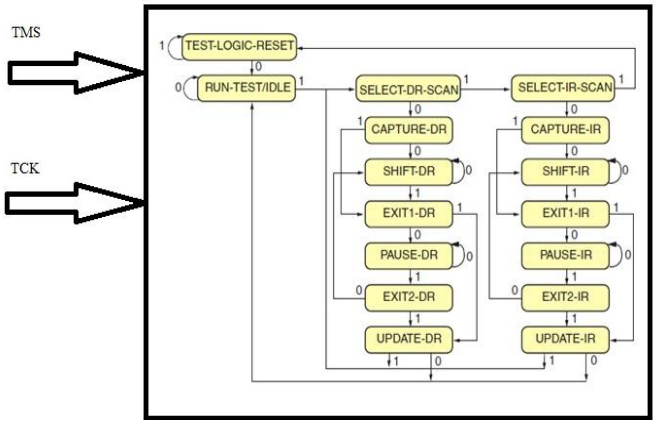


Figura 6.2. Diagrama de estados del TAP.

Las dos columnas verticales que se observan en la figura 6.2, al interior del bloque, representan, la ruta de instrucciones del lado derecho y la ruta de datos del lado izquierdo.

La FSM está integrada por 16 estados diferentes los cuales son:

- *Test Logic Reset*
  En este estado se deshabilita toda la lógica JTAG y el TAP regresa al estado inicial, sin importar cual sea el estado actual del controlador.
- *Run test/IDLE*
  En este estado se activa la lógica JTAG solo si se presentan ciertas instrucciones. El controlador permanecerá en este estado mientras TMS se mantiene bajo. La instrucción no cambia mientras el TAP se encuentre en este estado.
- *Select IR Scan/Select DR Scan*
  El controlador entra a la línea de instrucciones o a la línea de datos. Este es un estado temporal en el que el registro de datos seleccionados por la instrucción actual conserva su estado anterior. La instrucción no cambia mientras el TAP se encuentre en este estado.

Ruta de datos

- *Capture DR*
  En este estado el dato del banco de registro de datos cambia, mientras que en paralelo se cargan los siguientes valores en el registro, en un flanco de subida de TCK.

- *Shift DR*
  En este estado el registro de datos conectado entre TDI y TDO, como resultado de la instrucción actual, desplaza los datos hacia la salida de forma serial en cada flanco de subida de TCK. La instrucción no cambia mientras el TAP se encuentre en este estado.
- *Exit1 DR*
  El controlador pasa al estado de *Update_DR* y termina el proceso de escaneo o al estado de *Pause_DR*. Este es un estado temporal en el que el registro de datos seleccionados por la instrucción actual conserva su estado anterior. La instrucción no cambia mientras el TAP se encuentre en este estado.
- *Pause DR*
  Este estado permite el cambio del registro de datos que se encuentra en la trayectoria entre TDI y TDO y es detenido temporalmente.
- *Exit2 DR*
  El controlador pasa al estado de *Update_DR* y termina el proceso de escaneo, o bien, pasa al estado de *shift_DR*. Este es un estado temporal en el que el registro de datos seleccionados por la instrucción actual conserva su estado anterior. La instrucción no cambia mientras el TAP se encuentre en este estado.
- *Update DR*
  El dato se almacena en el registro de datos en cada flanco de bajada de TCK.

Ruta de Instrucciones

Los estados *Shift IR, Exit1 IR, Pause IR* y *Exit2 IR* que se presentan en la línea de instrucciones, llevan un proceso similar a los estados de la línea de datos.

- *Capture IR*
  En este estado la instrucción en el banco de registro de instrucciones cambia, mientras que en paralelo se carga la siguiente instrucción en el registro en un flanco de subida de TCK. Los últimos dos bits menos significativos deben ser siempre "01".

Arquitectura *Boundary-Scan*

Cuando se habla de la arquitectura *Boundary-Scan*, se habla de los tipos de registros que componen la arquitectura, esto es: registro de Instrucciones, de identificación, de *BYPASS*, *Boundary-Scan*, de configuración JTAG, etc., sólo por mencionar a los más relevantes. A continuación se mencionan algunos detalles de cada uno de los registros antes mencionados.

- Registro de Instrucciones
  Permite que una instrucción pueda ser cambiada dentro del diseño. La instrucción define la tarea a realizar, el registro de datos a acceder o ambos. Este registro se

conecta entre TDI y TDO durante el escaneo de las instrucciones. La instrucción se envía a través de TDO, al mismo tiempo que se carga la siguiente instrucción a través de TDI.

- Registro de Identificación

  También llamado Registro IDCODE, este registro permite identificar a través del TAP datos de: fabricante, número de parte y versión del dispositivo conectado en la cadena.

- Registro de *BYPASS*

  Proporciona una ruta corta en serie para los datos transmitidos, consiste en *flip-flops* conectados entre TDI y TDO. Cuando se presenta una instrucción de *BYPASS* el *bitstream* pasa inadvertido de TDI a TDO mientras se encuentra en esta instrucción.

- Registro *Boundary-Scan*

  Es un registro integrado por celdas que permiten la interacción entre las terminales externas del dispositivo y la lógica embebida en él, cuando se encuentra en modo de prueba.

- Registro de Configuración JTAG

  Este registro permite el acceso al bus de configuración y a las operaciones de *readback*. El bus de configuración, permite cargar el *bitstream* de configuración en el FPGA. Las operaciones *readback* permiten verificar que los datos de configuración actuales dentro del dispositivo son correctos, leyendo los datos desde la memoria de configuración interna.

6.1.2 Arquitectura interna del firmware de configuración

Antes de hablar de los tipos de archivos de reconfiguración, es necesario explicar la arquitectura interna del *Firmware* utilizado para configurar un FPGA a través del protocolo JTAG, utilizando como medio de control y gestión de datos, a un procesador, figura 6.3.

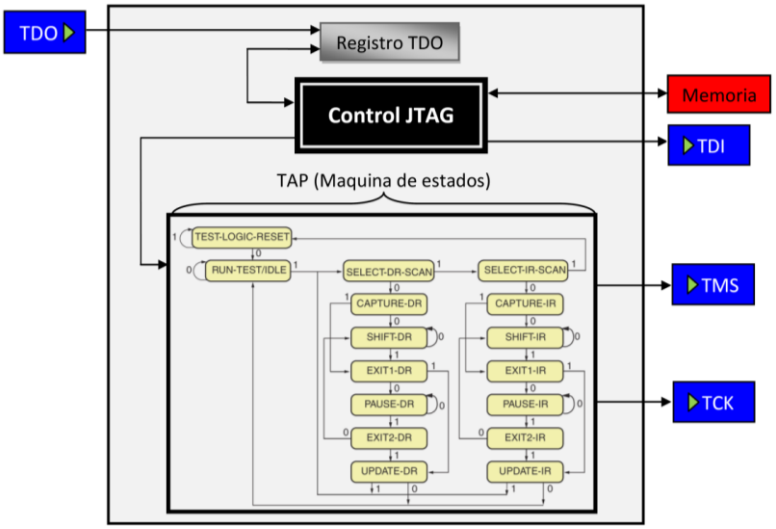


Figura 6.3. Arquitectura del software de reconfiguración.

Como se muestra en la figura anterior, la arquitectura del *firmware* fue dividida en tres bloques, donde cada uno de ellos se encuentra ligado al otro. Estos bloques son:

Control JTAG

Este bloque controla todos los procesos que se llevan a cabo internamente, los cuales realizan las siguientes tareas:

- Obtiene la información almacenada en el registro TDO y la compara con un paquete de datos generado con anterioridad.
- *Byte* a *byte* extrae el *bitstream* almacenado en la memoria EEPROM, para: ejecutar las instrucciones propias del protocolo JTAG, generar la señal TDI y transmitirla al FPGA.
- Una vez que se ejecutan las instrucciones de este mismo bloque, controla la Maquina de estados del TAP.

Registro TDO

En este bloque se almacena momentáneamente la información recibida desde FPGA.

TAP

Dentro de cada uno de los estados del TAP se generan las señales TCK y TMS, y se transmiten al FPGA para controlar el TAP del bloque JTAG embebido en el FPGA.

6.1.3 Archivos de reconfiguración para los dispositivos FPGA

Los dispositivos FPGA de Xilinx pueden reconfigurarse a partir de archivos de datos que contienen arquitecturas de cómputo diseñados previamente en la suite de desarrollo ISE. Estos archivos pueden cargarse en un FPGA en tres formatos diferentes: archivos con formato bit, formato svf o formato xsvf. Cada formato posee características propias y ventajas, las cuales se describen en los siguientes párrafos:

Formato bit

Es la representación binaria de la arquitectura de cómputo. Este archivo se utiliza para configurar los dispositivos de Xilinx, también puede utilizarse para generar archivos para memorias PROM, realizando algunas adecuaciones del formato. Los archivos en este formato frecuentemente se generan en el navegador de proyectos de la suite ISE de Xilinx.

Formato svf

Es un archivo vectorial en serie, y es una versión del formato .bit, el cual se utiliza para registrar las instrucciones del protocolo JTAG con la descripción de la arquitectura de cómputo que será descargada en el FPGA (*bitstream*). Este archivo es una representación ASCII de los comandos del protocolo JTAG junto con el *bitstream*, y es generado por medio del *software* IMPACT, incluido también en la suite ISE, o bien por medio de un programador JTAG comercial. Al integrar las instrucciones del protocolo JTAG, la magnitud de este archivo es aproximadamente tres veces mayor que el .bit.

Formato xsvf

Este formato reúne las características de los dos formatos anteriores, es un archivo que integra las instrucciones del protocolo JTAG, como el formato svf y binario, o como el formato .bit, con la ventaja de que este es un formato comprimido del formato svf, por lo tanto a pesar de que integran las instrucciones JTAG son de la misma magnitud que un archivo .bit. Es un formato propietario de Xilinx. Los archivos con este formato están optimizados para ejecutar los comandos del protocolo JTAG en los dispositivos de Xilinx y están destinados para uso en aplicaciones embebidas.

Para realizar la reconfiguración del FPGA se optó por utilizar un archivo de reconfiguración en formato **xsvf** debido a las ventajas que ofrece en términos de un tamaño compacto, optimización de uso con el protocolo JTAG y flexibilidad para manejarlo en ambientes de desarrollo en *software* de Xilinx y de otros fabricantes.

6.1.4 Instrucciones propias de los archivos XSVF

El formato del archivo xsvf está formado por un *byte* de instrucción seguido por una variable numérica como argumento. Estas instrucciones describen las operaciones del estándar IEEE 1149.1, similares a las del formato svf, con la diferencia del que el archivo xsvf es más compacto. Las instrucciones, con su respectivo valor hexadecimal al que responden junto con la descripción de la tarea a efectuar en el proceso de reconfiguración, son las siguientes:

- **XCOMPLETE→ 0x00**
  Indica que el archivo xsvf ha finalizado.
- **XTDOMASK→ 0x01**
  El argumento de esta instrucción establece una máscara para los valores de TDO.
- **XSIR → 0x02**
  Envía al controlador del TAP al estado *Shift-IR* y cambia el valor de TDI. Si el tiempo determinado por la instrucción XRUNTEST no es cero, envía el controlador

del TAP al estado *Run-test/idle* y espera el tiempo especificado. En otro caso, va al estado XENDIR.

- **XSDR → 0x03**
  Envía al controlador del TAP al estado *Shitf-DR* y cambia el valor de TDI. Compara el valor TDO recibido en la instrucción XSDRTDO con el valor establecido por la instrucción XTDOMASK. Si no coinciden los valores, la operación falla.
- **XRUNTEST → 0x04**
  Cuando en la instrucción XENDIR o XENDDR el estado del controlador es *Run-Test/Idle*, el argumento de XRUNTEST define el valor mínimo del ciclos de reloj (TCK) que deben transcurrir y el mínimo valor en microsegundos que el dispositivo debe permanecer en este estado, en cada ejecución de las instrucciones SDR o SIR.
- **XREPEAT → 0x07**
  El argumento de esta instrucción indica el número de veces que el valor TDO recibido es comparado antes de que se considere que ha fallado la operación.
- **XSDRSIZE → 0x08**
  El argumento de esta instrucción determina la longitud de los registros de XSDR y XSDRTDO.
- **XSDRTDO → 0x09**
  Envía al controlador del TAP al estado *Shift-DR* y cambia el valor de TDI. Compara el valor TDO recibido con el valor TDO generado por la instrucción XTDOMASK. Si no coinciden los valores, la operación falla.
- **XSETSDRMASKS → 0x0A**
  Aquí se indican las máscaras de los datos y la dirección de la instrucción XSDRINC que se presente después de esta instrucción.
- **XSDRINC → 0x0B**
  Realiza sucesivamente instrucciones XSDR.
- **XSDRB → 0x0C**
  Envía al controlador del TAP al estado *Shift-DR* y cambia el valor de TDI. Permanece en este estado hasta que se indique que la operación debe finalizar
- **XSDRC → 0x0D**
  Cambia el valor de TDI y sigue permaneciendo en el estado *Shift-DR* hasta que se indique que la operación debe finalizar.
- **XSDRE → 0x0E**
  Cambia el valor de TDI. Va al estado XENDDR y finaliza la operación.
- **XSDRTDOB → 0x0F**
  Envía al controlador del TAP al estado *Shift-DR* y cambia el valor de TDI. Compara el valor TDO recibido con el valor TDO generado.
- **XSDRTDOC → 0x10**
  Cambia el valor de TDI y compara el valor TDO recibido con el valor TDO generado (permanece en el estado *Shift-DR*).

- **XSDRTDOE → 0x11**
  Cambia el valor de TDI y compara el valor TDO recibido con el valor TDO generado, al finalizar la operación envía el controlador del TAP al estado XENDDR y finaliza la operación.
- **XSTATE → 0x12**
  Cuando el argumento de la instrucción es cero (0x00) envía al controlador del TAP al estado *Test Logic Reset*, garantizando el restablecimiento del TAP. Para valores distintos a cero, el TAP se mantiene en su estado actual y no realiza ninguna modificación.
- **XENDIR → 0x13**
  Esta instrucción indica el final de la instrucción XSIR. El argumento indica en qué estado se posicionará el controlador del TAP. Si el valor es 0x00 envía al controlador al estado *Run-Test/Idle*, pero si el valor es 0x01 envía al controlador al estado *Pause IR*.
- **XENDDR → 0x14**
  Esta instrucción indica el final de la instrucción XSDR. El argumento indica en qué estado se posicionará el controlador del TAP. Si el valor es 0x00 envía al controlador al estado *Run-Test/Idle*, pero si el valor es 0x01 envía al controlador al estado *Pause DR*.
- **XSIR2 → 0x15**
  Esta instrucción se utiliza para cambio de instrucciones con una longitud de más de 255 bits, en otro caso se utiliza la instrucción XSIR.
- **XCOMMENT → 0x16**
  Especifica una longitud arbitraria de caracteres que terminan con un byte cero.
- **XWAIT → 0x17**
  Envía al controlador del TAP al estado *wait_state* y permanece en el estado, al finalizar envía al controlador del TAP a estado *end_state*.

Una vez descritas las tareas que cumplen cada una de las instrucciones del set de JTAG, junto con el funcionamiento del TAP, es posible comprender con más claridad el proceso de reconfiguración a través del protocolo JTAG.

6.2 Diseño y construcción del hardware

Por supuesto que gran parte del éxito en el proceso de reconfiguración del sistema que se describe en este trabajo de tesis recae en el *hardware*. El desarrollo del *hardware* se realizó en dos etapas: la primera consideró el desarrollo de interfaces propietarias y propias, mientras que en la segunda se realizó la integración de los resultados parciales alcanzados en la primera etapa, para concretar un diseño final propio. Como parte de las pruebas parciales que se realizaron, se encuentran pruebas de comunicación utilizando radio módems para la escritura de una memoria utilizando un microcontrolador (PIC) como elemento gestor de carga. Los resultados de estas pruebas permitieron definir la integración de bloques fundamentales para el sistema, tales como transceptor para comunicaciones seriales, PIC gestor y memoria de almacenamiento del *bitstream*, cuyas pruebas se detallan en el capítulo posterior.

El diseño está integrado por una tarjeta electrónica totalmente diseñada, ensamblada de forma manual y validada en el Instituto de ingeniería de la UNAM. Esta tarjeta está integrada a su vez por varios módulos que permiten la comunicación exterior y transferencia de datos por puerto RS232, PIC a bordo para gestión del proceso de reconfiguración, una memoria EEPROM para el almacenamiento del *bitstream* de configuración del FPGA, adquisición de datos digitales, herramientas de depuración a bordo con entradas y salidas digitales, además de fuentes de alimentación propias que permiten realizar pruebas aisladas fuera de la plataforma SATEDU. En los siguientes apartados se describen cada uno de estos bloques.

6.2.1 Cable parallel III

En la primera etapa de validación de la reconfiguración del FPGA (utilizando el protocolo JTAG en la tarjeta de desarrollo Spartan 3E-SK) se utilizó una tarjeta propia, fabricada, ensamblada y validada en el laboratorio del Instituto de ingeniería, basada en el *hardware* libre desarrollado y proporcionado por Xilinx, denominado cable *Parallel III* o DLC5 que se muestra en la figura 6.4.

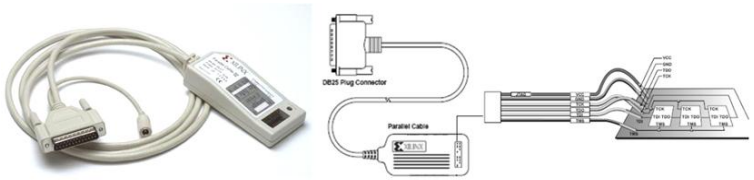


Figura 6.4. Cable DLC5 comercializado por Xilinx.

La tarjeta electrónica asociada a este cable está compuesta principalmente por *buffers* de tres estados que permiten interactuar con las terminales de puerto JTAG del FPGA. Estos *buffers* ajustan las señales de entrada/salida del puerto paralelo de la PC con las señales del puerto JTAG del FPGA, asegurando una correcta comunicación y protegiendo al dispositivo de algún daño que pueda presentarse a causa de variaciones de voltaje durante el proceso de reconfiguración.

El diagrama esquemático de la versión propia del DLC5 se muestra en la figura 6.5. Para el desarrollo del proyecto, tanto para el diagrama esquemático como para el circuito impreso, se utilizó el *software* de desarrollo Protel DXP.

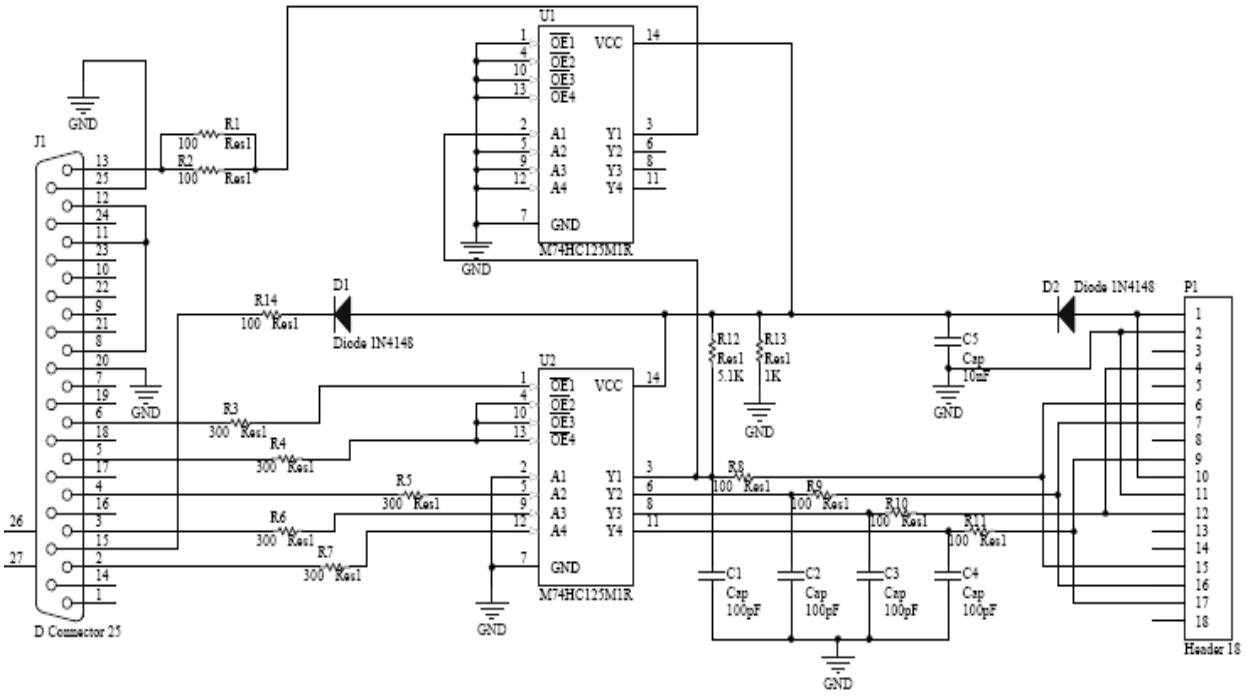


Figura 6.5. Diagrama esquemático del cable parallel III.

La electrónica asociada con el cable fue implantada en una tarjeta fenólica de 3x3cm, en una sola capa y utilizando *buffers* de montaje superficial. El ensamble y soldado de los dispositivos, junto con las pruebas eléctricas, se realizaron a mano en el laboratorio del Instituto de ingeniería. En las figura 6.6 se muestra parte del proceso de ensamble de la tarjeta que contiene la electrónica asociada al cable.

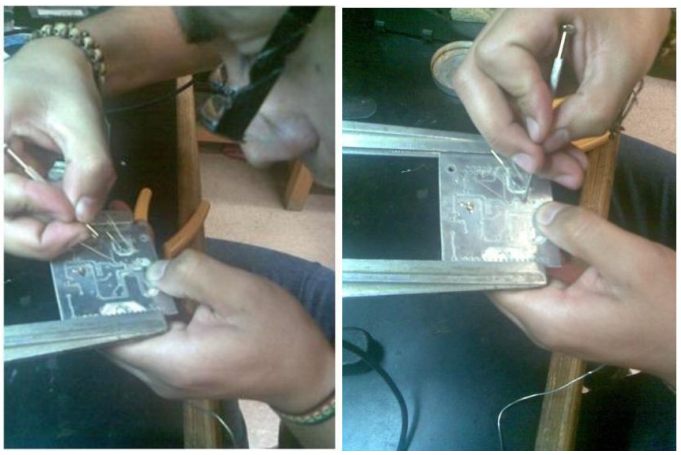


Figura 6.6. Ensamble y validación de la tarjeta propia parallel III.

La versión del DLC5 fabricada en el Instituto de ingeniería, ofreció una potente herramienta que facilitó el establecimiento de un primer esquema en *hardware* de los elementos que se requerirían en la estructura del bloque de reconfiguración del FPGA, además de que permitió la validación de *hardware* de desarrollo propio de muy bajo costo. Comercialmente este cable tiene un precio en el mercado de Estados Unidos de $8.00 USD, y tomando en cuenta impuestos y gastos de envío se añaden unos $13.00 USD, por lo que el costo por adquirir un cable de este tipo es de aproximadamente $21.00 USD. El prototipo que se fabricó tuvo un costo de $90.00 MXN. En la figura 6.7 se muestra el prototipo del cable compatible con el DLC5.

Figura 6.7. Versión propia del cable DLC5.

6.2.2 Tarjeta FPGA de diseño propio (tarjeta SMIN)

Una vez que fueron probados algunos de los esquemas de reconfiguración del FPGA en la tarjeta de desarrollo Spartan 3E, y en virtud de los requerimientos para realizar la actualización de la tarjeta electrónica de SMIN, se procedió a realizar el diseño de una

nueva tarjeta electrónica que permitiera, la implementación del esquema de reconfiguración remota del FPGA.

El proceso de diseño de la tarjeta, comenzó con el planteamiento del *layout* del sistema completo, considerando las entradas/salidas digitales necesarias para validación de arquitecturas de cómputo, comunicaciones con una estación terrena para intercambio de telemetría y comandos, medios de depuración y prueba a bordo (interruptores y leds). De acuerdo con el esquema que se muestra en la figura 6.8.

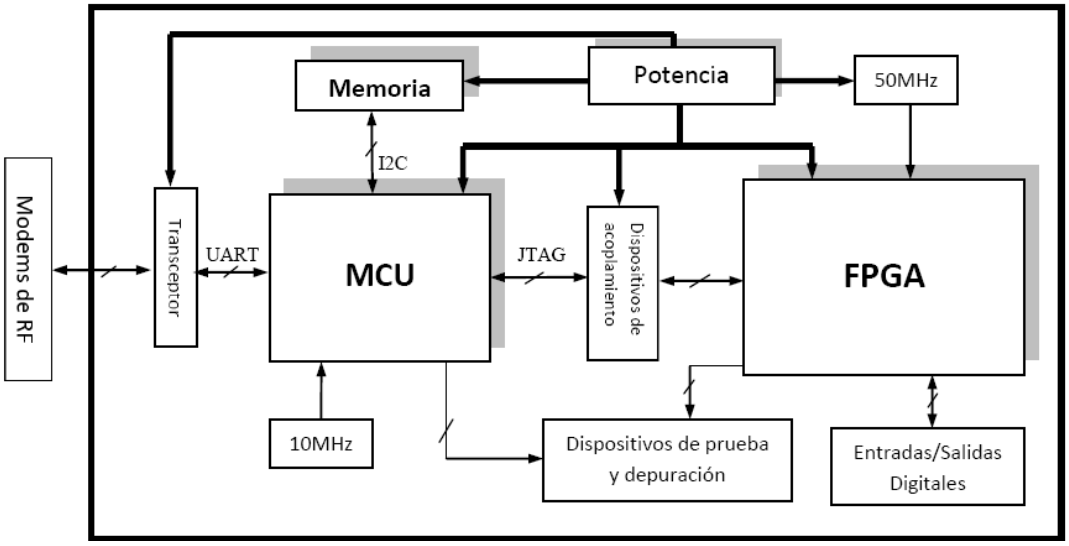


Figura 6.8. Diagrama de bloques de la tarjeta de SMIN.

El proceso de desarrollo del sistema partió de realizar un análisis de disponibilidad de componentes electrónicos en el mercado y de la factibilidad técnica para el montaje de algunos de los elementos que integra el sistema sobre la tarjeta de circuito impreso, principalmente el FPGA. En el instituto de ingeniería UNAM, se han soldado componentes electrónicos de montaje superficial de hasta 250 terminales. Debido a esto, se seleccionó un dispositivo que tuviera un empaquetado tal que fuera posible soldar a mano sobre el circuito impreso. Los demás componentes podrían ser integrados en versiones de empaquetados *through hole*, lo que aceleraría el tiempo de diseño y ensamble.

Como se ha comentado, la tarjeta electrónica desarrollada contiene dispositivos para comunicación, reconfiguración del FPGA y medios de prueba para validar las arquitecturas de cómputo cargadas en el FPGA. Los dispositivos mencionados a continuación tienen un papel trascendente en el funcionamiento de la tarjeta.

FPGA XC3S100E

Basados principalmente en la factibilidad técnica de montaje se eligió al FPGA XC3S100E con empaquetado TQG144C. Es un dispositivo de bajo costo y alto rendimiento para aplicaciones de alta escala de integración. Este dispositivo cuenta con 144 terminales, de las cuales algunas cumplen con funciones de entrada/salida de propósito general, otras sirven para la polarización del dispositivo y para la configuración del mismo. Opera con señales de voltaje de 3.3V, 2.5V y 1.2V.

Posee dos gestores de reloj digital DCMs, que operan en un intervalo de frecuencias que van desde 5MHz hasta 300MHz utilizando un oscilador externo de 50MHz, con divisores, multiplicadores y sintetizadores de frecuencia, ocho señales de reloj globales y ocho señales de reloj designadas para cada mitad del dispositivo. Integra una memoria RAM rápida de 72Kbits y 15Kbits de memoria RAM distribuida. Se compone por puertos designados para la configuración por comunicación JTAG IEEE 1149.1/1532, *Master Serial, Slave serial, Master parallel Up y Down, y SPI Serial Flash.* En la figura 6.9 se muestran otros recursos internos del dispositivo.

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40

Figura 6.9. Recursos internos del FPGA XC3S100E.

Microcontrolador PIC

Como unidad de procesamiento se utilizó un microcontrolador PIC18F4520 de 8 bits y tecnología RISC, con puertos dedicados para comunicación sere síncrona I2C y SPI para ser utilizado en modo maestro y esclavo. Trabaja con una frecuencia de oscilación máxima de 40MHz (frecuencia dada por un reloj externo y un PLL interno) y a ocho diferentes frecuencias del oscilador interno que van de 31KHz hasta 8MHz, con una memoria interna de 256Kbytes y módulo con 13 bits dedicados para convertidor analógico/digital. El PIC es suficiente para integrar el *firmware* de recepción, transmisión y adquisición de datos y el *firmware* de reconfiguración.

Memoria I2C EEPROM

La razón principal por la que se utilizó esta tecnología de memorias fue ahorrar tiempo de desarrollo y darle más prioridad a las pruebas de reconfiguración del FPGA, otra razón es por que utiliza un protocolo de comunicación fácil de manejar y el proceso de adquisición de datos hacia la memoria es sencillo.

Se utilizó una memoria EEPROM 24LC1025, que tiene una capacidad de almacenamiento de 1Mbit, con un *buffer* de datos de 128bytes. Diseñado para aplicaciones de baja potencia como comunicaciones y adquisición de datos. Este dispositivo es capaz de leer datos de forma aleatoria y secuencial. Las líneas de dirección de la memoria A0 y A1 permiten conectar hasta 4 dispositivos en el mismo *bus*, permitiendo almacenar hasta 4Mbits de información (estas líneas se configuran de forma física).

Utiliza además el protocolo de comunicación serie I2C para entablar comunicación con otros dispositivos, este protocolo utiliza un *bus* que consta de 2 señales, SCL y SDA, siendo la primera una señal de reloj de entrada que sincroniza la transferencia de datos desde y hacia la memoria, en tanto que la segunda es una señal bidireccional que sirve para transferencia y recepción de datos. Las terminales físicas SDA y SCL del dispositivo se encuentran en *open-drain*, por lo tanto, requieren de resistencias de *pull-up*.

En las figuras 6.10 y 6.11, se muestra el flujo de comunicación de lectura y escritura en la memoria.

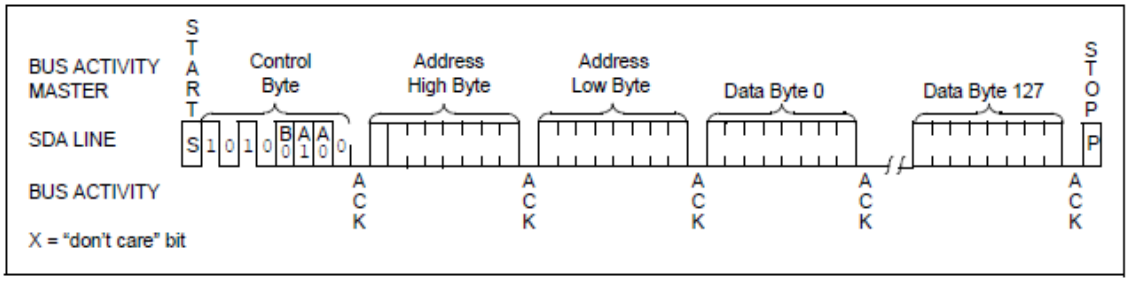


Figura 6.10. Proceso de escritura de la memoria.

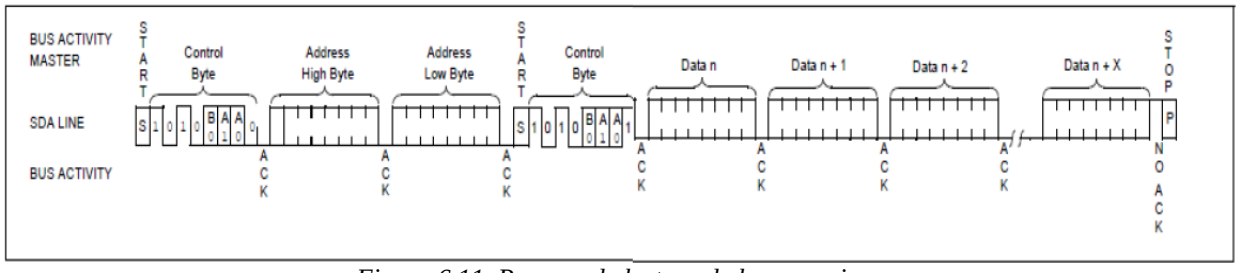


Figura 6.11. Proceso de lectura de la memoria.

Buffer 74HC125

Este dispositivo se utiliza para compatibilizar las señales provenientes de las terminales del PIC con las terminales del FPGA. Es un dispositivo tres estados, de alta velocidad, sus entradas son compatibles con el estándar CMOS y TTL de baja potencia (LSTTL), con un rango de operación de 2 hasta 6 volts y alta inmunidad al ruido.

Conector DB9 y transceptor MAX233

El dispositivo MAX233 acondiciona el nivel de voltaje de las señales provenientes del puerto serie de la estación terrena con las señales del puerto serie USART del PIC. El conector DB9 sirve para compatibilizar las terminales del modem con las terminales de la USART del PIC.

Oscilador 50MHz

Genera la señal de reloj con la que opera la lógica del FPGA. La frecuencia de operación del oscilador va de 500kHz hasta 165MHz y es controlada a través de la corriente de entrada. La frecuencia de operación para la tarjeta es de 50MHz.

Diseño de la tarjeta electrónica

Para el diseño de la tarjeta de circuito impreso de sistema mínimo se tomaron como principales fuentes de referencia algunos sistemas comerciales como: el mismo Spartan 3E-SK, del cual Xilinx ofrece parcialmente sus diagramas esquemáticos; *Sparkfun* que ofrece en su sitio los diagramas esquemáticos de la tarjeta de desarrollo DEV-08458 usando el FPGA XC3S500E; algunas tarjetas desarrolladas en universidades así como notas de aplicación y hojas de especificaciones de productos de Xilinx. Con base en las ideas presentadas en las fuentes citadas y atendiendo a los requerimientos de nuestra aplicación, se integró el sistema que se describe en el diagrama esquemático de la figura 6.12.

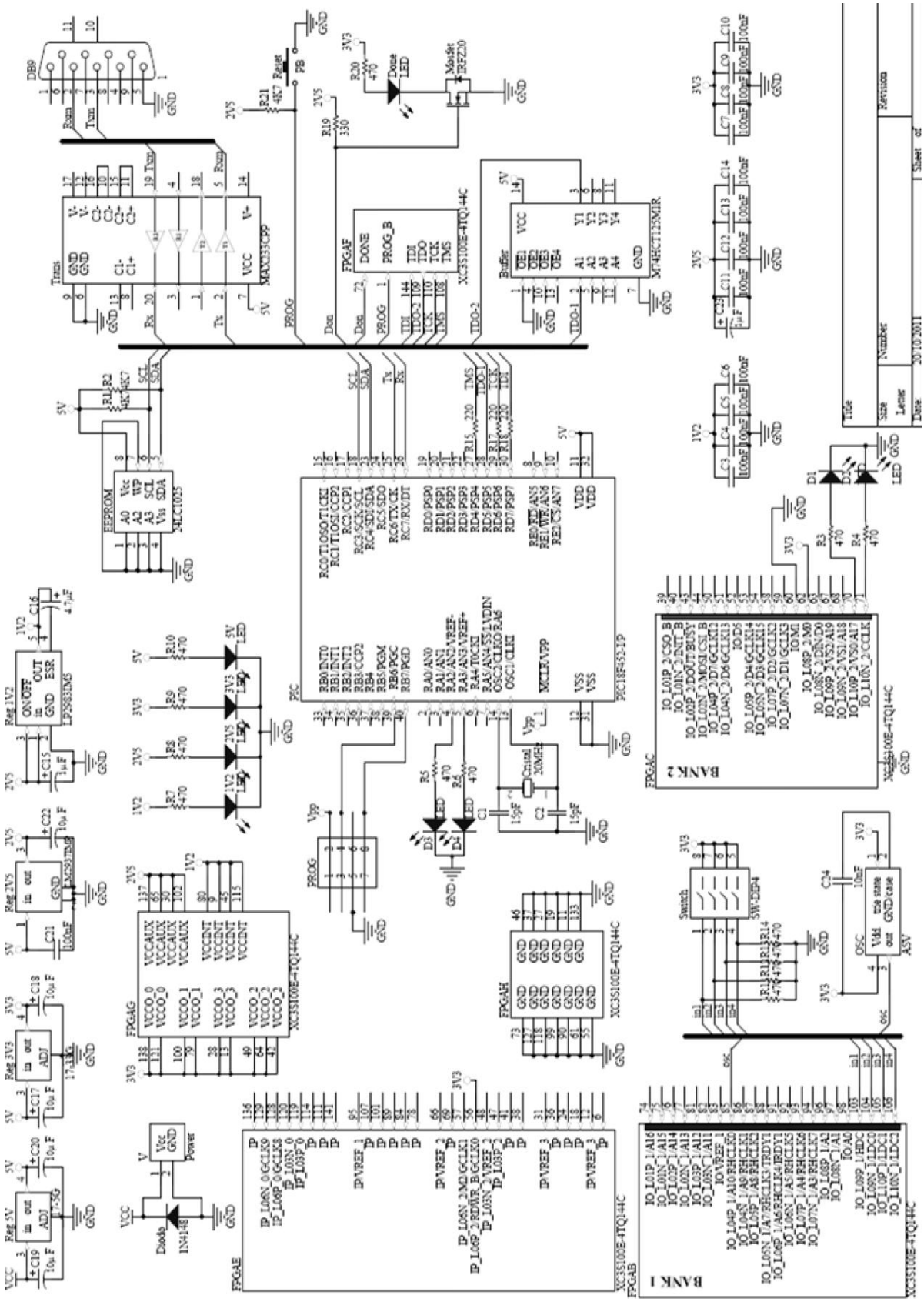


Figura 6.12. Diagrama esquemático de la tarjeta SMIN.

El montaje de los componentes electrónicos junto con las pruebas de validación de la tarjeta, se realizaron en el laboratorio del Instituto de ingeniería. Debido al espacio reducido que existe entre terminales del FPGA, fue necesario utilizar un microscopio para realizar el montaje y soldado a mano de los componentes más complicados de la tarjeta, como el FPGA. La figura 6.13 muestra el proceso de montaje y soldadura del FPGA sobre la tarjeta de circuito impreso.

Figura 6.13. Montaje de FPGA en la tarjeta de circuito impreso usando microscopio.

La tarjeta desarrollada tiene una dimensión de 9x9 cm, ajustando estas dimensiones al estándar de SATEDU, que será el sistema final al que se integrará luego de varias pruebas de desempeño y depuración de errores y omisiones. El diseño de la tarjeta, para esta primera versión, se estableció a una cara por varios factores, principalmente para abatir los costos de construcción, también se incluyeron algunos dispositivos con empaquetado *through hole* para acelerar los tiempos de ensamble de los circuitos integrados y componentes pasivos en la tarjeta. En la figura 6.14 se muestra la tarjeta de circuito impreso con los componentes electrónicos ya instalados.

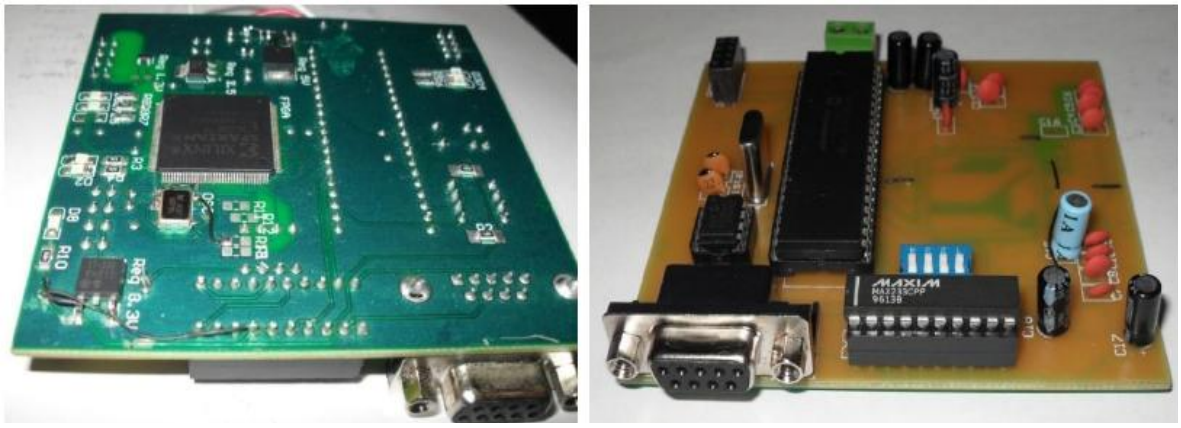


Figura 6.14. Tarjeta SMI, lado superior y lado inferior.

6.3 Descripción del software

Para realizar las pruebas durante el proceso de desarrollo modular del sistema, se utilizaron las mismas herramientas en *software* para cada bloque en *hardware*. Gracias a esto, se logró validar la operación correcta, por separado, de la estación terrena y de la tarjeta electrónica de SMIN.

El *software* utilizado en la estación terrena fue desarrollado en lenguaje de alto nivel, integrando una interfaz gráfica que permite al usuario, de manera amigable e intuitiva, el inicio del envío del *bitstream* al FPGA. En el caso de la tarjeta electrónica, su *software* está integrado básicamente en el *firmware* del PIC. Este *firmware* fue desarrollado en lenguaje C++ y contiene modularmente toda la lógica del TAP del protocolo JTAG para efectuar la transferencia de los archivos de reconfiguración, desde la memoria a bordo al FPGA, así como el almacenamiento del *bitstream* en la misma memoria.

El *software* de estación terrena está integrado por diversos módulos, ya que para transmitir el *bitstream* de reconfiguración, éste debe pasar por diferentes procesos una vez que se sintetiza la arquitectura que describe el sistema digital que se quiere cargar en el FPGA.

Para realizar las primeras pruebas de reconfiguración fue indispensable utilizar un *software* ejecutado en la PC, este *software* extrae un archivo en formato xsvf de alguna carpeta del disco duro de la PC y realiza las operaciones necesarias para llevar a cabo la reconfiguración del FPGA a través del protocolo JTAG, usando el puerto paralelo de la PC. El *software* nos proporcionó resultados exitosos, necesarios para definir con un mayor nivel de confianza el esquema de reconfiguración final, ya que la lógica de este proceso fue extrapolada al *firmware* de reconfiguración montado en el PIC.

Las herramientas en *software* utilizadas para la estación terrena son:

- **IMPACT de la suite ISE de Xilinx:** utilizado para convertir el archivo bit generado a un archivo xsvf.
- **Intel_Hex:** realiza la conversión al formato Intel Hex. Este *software* es fácil de utilizar, con él se convierte un archivo binario a un archivo ASCII Hexadecimal.
- **Interfaz Grafica de Usuario (GUI):** transmite y recibe el *bitstream* de configuración de forma serial, almacena y extrae el mismo de la memoria, al igual que indica el inicio del proceso de reconfiguración, donde el usuario visualiza en cada momento el proceso de cada operación, haciendo sencillo el procedimiento de comunicación y transferencia de datos entre estación terrena y el subsistema de control de orientación.

El *firmware* del PIC del sistema mínimo está integrado por 2 bloques, que al finalizar las pruebas de validación, fueron integrados en un solo *firmware* para programarlo en el PIC. Los módulos de *firmware* utilizados para el sistema son:

- **Firmware de recepción, transmisión y almacenamiento en memoria:** el cual realiza las tareas relacionadas con transmisión y recepción de datos seriales, así como el almacenamiento, extracción y borrado de la memoria.
- **Firmware de reconfiguración:** donde se realiza la reconfiguración del FPGA a través de puertos de propósito general del PIC, utilizando el protocolo JTAG.

El *software* fue diseñado bajo el concepto de flexibilidad, de manera tal que pueda actualizarse con facilidad, para mejorar o agregar tareas al sistema. Se diseñó de esta manera por practicidad, ya que la tarjeta que contiene al sistema desarrollado tiene el *hardware* de soporte necesario para operar tanto dentro como fuera de la plataforma de SATEDU. Al utilizar la tarjeta en SATEDU como plataforma para evaluar arquitecturas de cómputo, se efectúan las comunicaciones por medio de tarjeta de comunicaciones inalámbricas, en tanto que al usar la tarjeta fuera de SATEDU se emplearía como herramienta didáctica para el aprendizaje de tecnología FPGA.

6.3.1 Software de reconfiguración diseñado para la PC

El *software* de reconfiguración está diseñado para configurar al FPGA utilizando como interfaz al puerto paralelo de la PC, usando el protocolo de transferencia de datos JTAG. El proceso de configuración se lleva a cabo extrayendo de alguna carpeta en la PC el *bitstream* de reconfiguración en formato xsvf, y byte a byte se ejecutan los comandos necesarios para realizar la reconfiguración del FPGA. El programa original obtenido de Xilinx, se encuentra en lenguaje de programación C++, y para este trabajo se desarrolló una versión en Visual C++.

La arquitectura del *software* contiene los elementos obligatorios definidos en el estándar IEEE 1149.1 para el protocolo de configuración JTAG. Estos elementos sólo comprenden al TAP, al controlador del TAP y el registro que contiene las instrucciones .xsvf.

Está estructurado en un programa principal llamado “micro.c” y varios subprogramas auxiliares llamados “ports.c” y “lenval.c”, que hacen uso de algunas bibliotecas propietarias de Xilinx como “micro.h”, “ports.h” y “lenval.h”, los cuales se describen en los siguientes párrafos:

- **micro.c:** Contiene las funciones para interpretar los comandos del archivo xsvf, al igual que para procesar los datos del mismo. Este programa llama a los subprogramas “ports.c” y “lenval.c”.
- **micro.h:** Contiene las funciones prototipo para la interfaz primaria que reproduce el archivo xsvf.

- **ports.c:** Contiene las rutinas que generan los valores de salida hacia los puertos JTAG del FPGA, al igual que las rutinas para leer el valor de salida TDO, y para leer un byte del archivo xsvf almacenado en el disco duro de la PC.
- **Ports.h:** Contienen las declaraciones externas para proporcionar estímulos en los puertos JTAG.
- **Lenval.c:** Contiene las rutinas para utilizar la estructura de datos lenVal definida en "micro.c" (esta estructura es un tipo de byte de orientación utilizado para almacenar un valor binario de longitud arbitraria).
- **Lenval.h:** Contiene la descripción de la estructura de datos LenVal.

Estos programas también utilizan bibliotecas generales tales como: "conio.h", "stdio.h", "stdlib.h", "string.h", "time.h", "io.h" y "dos.h", las cuales habilitan el manejo de funciones estándar de entrada/salida, manejo de cadenas de caracteres alfanuméricos, manejo de puertos de la PC, etc.

En la figura 6.15 se muestra el flujo del proceso de reconfiguración que realiza el *software*, donde:

El proceso "A" realiza los siguientes pasos en el TAP:

- Select DR
- Select IR
- Capture IR
- Exit1 IR
- Update IR

Mientras que el proceso "B" realiza los siguientes pasos:

- Select DR
- Capture DR
- Exit1 DR
- Update DR

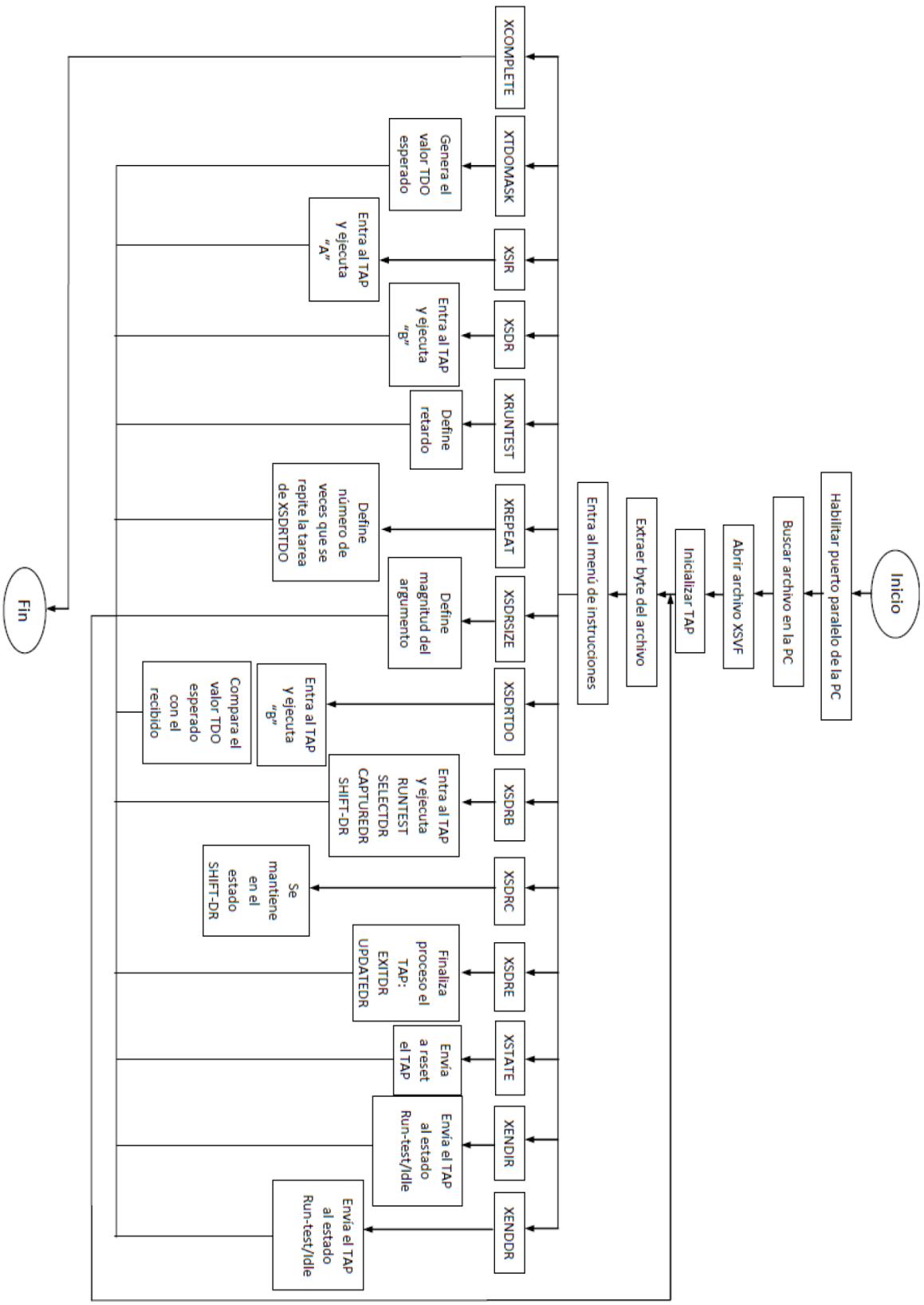


Figura 6.15. Diagrama de flujo del proceso de reconfiguración de FPGA utilizando la PC.

6.3.2 Estación terrena

6.3.2.1 Herramienta iMPACT

Una de las herramientas más importantes que se integró dentro del *software* de estación terrena es iMPACT, la cual es una herramienta desarrollada por Xilinx que permite básicamente dos tareas: la configuración de dispositivos (FPGAs, CPLDs y PROMs) y la generación de archivos de configuración.

La utilidad de configuración de dispositivos de iMPACT permite configurar directamente FPGAs o programar CPLDs y PROMs de Xilinx, utilizando interfaces alámbricas como *Mutipro Desktop Tool*, cable *parallel III* o el Platform Cable USB en varios modos, los cuales se describen a continuación:

- Modo *Boundary Scan*: En este modo pueden configurarse o programarse FPGAs, CPLDs y PROMs de Xilinx.
- Modo de configuración *Slave Serial* o *Select MAP*: en esto modo sólo pueden configurarse directamente a los FPGAs.
- Modo de configuración de escritorio (DESKTOP): en este modo se pueden programar CPLDs y PROMs de Xilinx.
- Modo de configuración directa SPI: Sólo se pueden programar dispositivos *Flash* seriales con interfaz SPI.

La opción de generación de archivos permite la creación de diversos tipos de archivos de configuración y programación para los dispositivos que soporta iMPACT, tales como: ACECF, MCS, SVF, STAPL y XSVF.

Adicionalmente iMPACT permite realizar las siguientes funciones:

- *Readback* y verificación de datos de configuración del diseño.
- Depuración de problemas de configuración.

Para el desarrollo de este trabajo de tesis la herramienta iMPACT se integró como parte de los programas de soporte externo, particularmente en una etapa de conversión previa de archivos de .bit a .xsvf, antes de realizar el envío del *bitstream* hacia la tarjeta electrónica que contiene al FPGA.

6.3.2.2 Software Intel_Hex

El *software* Intel_Hex tiene la función de convertir cualquier archivo binario a un archivo ASCII hexadecimal (.hex). Este *software* se utilizó para transmitir y almacenar el archivo de reconfiguración, utilizando el protocolo de comunicación serie RS232. La ventaja en el uso de este formato radica en que los archivos convertidos contienen datos que ayudan a la depuración de errores producidos durante el proceso de transmisión, y garantizan el

correcto almacenamiento de los datos de reconfiguración en la memoria. Para comprender más a fondo el funcionamiento del *software* de estación terrena, en los siguientes párrafos se habla brevemente de las partes que integran a este tipo de archivos.

Un archivo Intel HEX es un archivo de texto ASCII que codifica y representa un archivo binario. Cada línea en un archivo Intel HEX contiene números hexadecimales que representan código en lenguaje máquina o datos binarios. Por ejemplo: dado un valor binario de 8 bits 0011 1111 es 3F en hexadecimal, y su representación ASCII es un byte de 8 bits que contiene el carácter '3' el cual es 0011-0011 ó 033H y un byte de 8 bits que contiene el carácter 'F' el cual es 0100 0110 ó 046H. Como se observa en el ejemplo anterior, la representación hexadecimal ASCII requiere el doble de bytes de lo que requiere la representación binaria.

El formato Intel HEX está formado por registros, cada registro contiene el tipo de registro, la longitud del dato, la dirección de almacenamiento en memoria, los datos y un *checksum*. En el siguiente ejemplo se muestra parte de un archivo en formato Intel HEX donde se observan grupos de caracteres agrupados y subrayados, la descripción de cada grupo subrayado se presenta en los párrafos siguientes.

: 10 0010 00 06090800000020010FFFFFFF09000000 93

: 02 0000 04 0001 F9

: 00 0000 01 FF

Cada registro comienza con un código de inicio que contiene (**':'**), y su representación en código ASCII es 03AH.

Los primeros dos dígitos hexadecimales son el campo de ***Longitud del Registro*** y contienen la cantidad de bytes del registro, usualmente esta cantidad es 16 o 32 bytes.

Los siguientes cuatro dígitos hexadecimales son el campo de ***Dirección*** y como su nombre lo dice contienen la dirección donde se encuentra localizado cada dato en el archivo Intel HEX.

Los siguientes dos dígitos hexadecimales son el campo de ***Tipo de Registro***, estos dígitos van del 00 hasta el 05 y definen el tipo de datos que contiene el registro, hay 6 tipos de registros:

- **00** → Registro de Datos, contiene direcciones de 16 bits y sus correspondientes datos
- **01** → Fin de archivo, no contiene datos y se coloca como el último registro del archivo.

- **02 →** Dirección Extendida de Segmento, este tipo de registros se utilizan para acceder a direcciones con más de 16 bits. Este valor se desplaza 4 bits a la izquierda y se suma a la dirección proporcionada por los registros de datos. Su campo de longitud debe valer 02 y el de dirección 0000.
- **03 →** Dirección de Comienzo de Segmento, especifica los valores iníciales de los registros, para procesadores 80x86. El campo de dirección es 0000, longitud 04 y los datos contienen dos bytes para el segmento de código y otros dos para el *instruction pointer.*
- **04 →** Dirección Lineal Extendida, permite dirigirse hasta 32 bits de memoria una vez superados los 16 bits de dirección. Su campo de dirección vale 0000 y el de longitud 02.
- **05 →** Comienzo de Dirección Lineal. Contiene 4 bytes que se cargan en el registro EIP de los procesadores 80386 y superiores. Su campo de dirección vale 0000 y el de longitud 04.

Los tipos de registro 00, 01 y 04 son los únicos utilizados en el desarrollo del *firmware.*

Los siguientes 16 pares de dígitos hexadecimales son el campo de ***Datos***, en este campo se encuentra la información del archivo binario.

Los últimos 2 dígitos hexadecimales son el campo ***Checksum***, estos 2 dígitos son el complemento a dos de la suma de todos los campos anteriores, excepto ':'. Al sumar el valor de la longitud de registro, la dirección, el tipo de registro y los datos, con el valor del *Cheksum* el resultado siempre debe ser cero.

En la figura 6.16 se muestra el diagrama de flujo implementado.

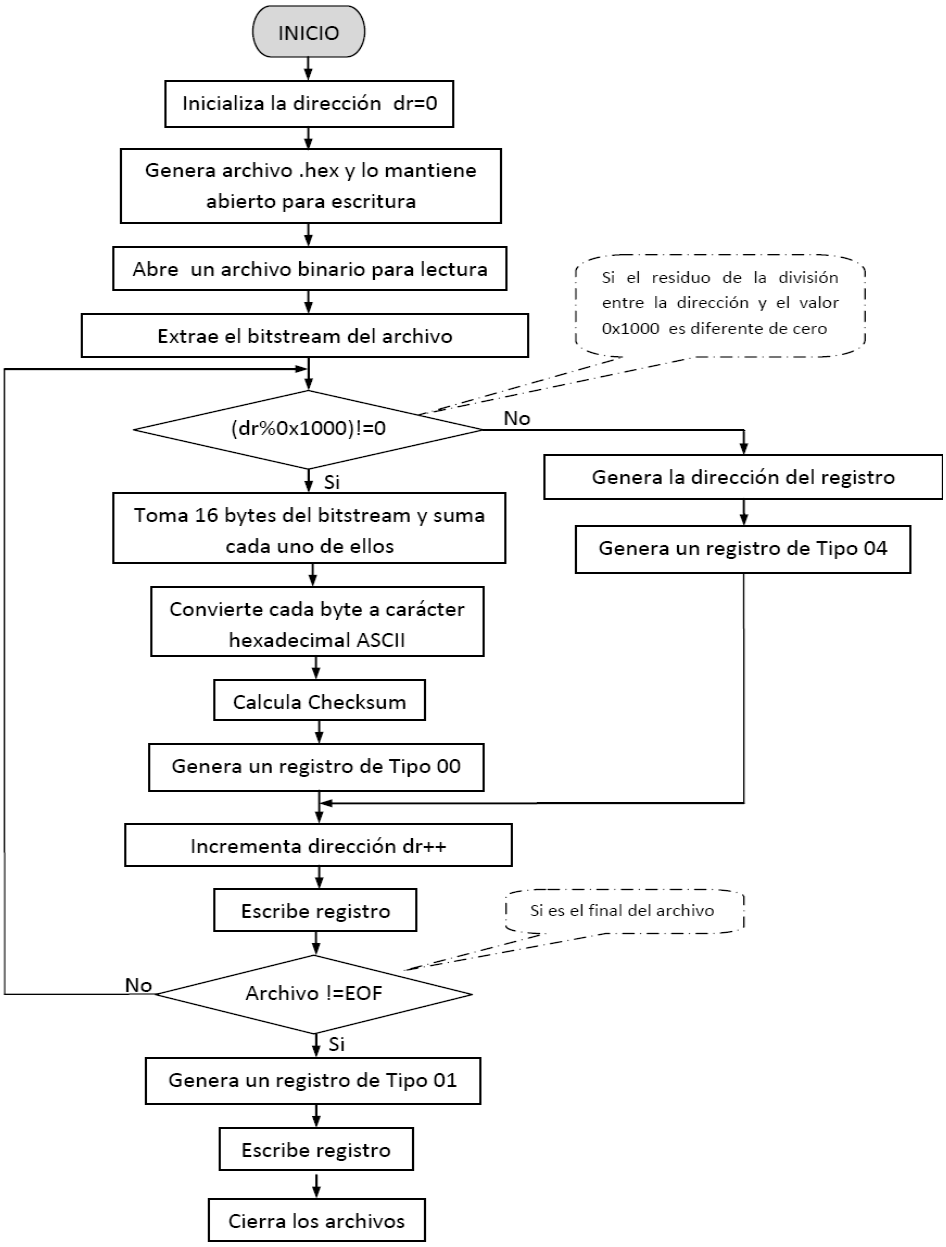


Figura 6.16. Diagrama de flujo para generar el formato Intel HEX.

6.3.2.3 Interfaz gráfica de usuario

Con objeto de facilitar el proceso para establecer comunicación entre la estación terrena y la tarjeta electrónica, así como otras tareas asociadas con la transferencia remota del *bitstream* de reconfiguración, se desarrolló en la PC un *software* que funciona como una interfaz con el usuario (GUI). La interfaz fue desarrollada utilizando el ambiente de programación Visual C#. Tiene una estructura modular, distribuida en cajas de selección, botones de comando y ventanas de despliegue de datos, que integran una herramienta altamente amigable con el usuario, permitiendo de forma sencilla e intuitiva desde la

selección del puerto serial de la computadora, la selección de parámetros de comunicación serie, hasta la selección del archivo de reconfiguración que ha de ser transmitido hacia la memoria EEPROM a bordo de la tarjeta, para posteriormente descargarlo en el FPGA. Una breve descripción de algunas de las tareas que se pueden realizar con la GUI, se describen en los siguientes párrafos.

- Tarea: verificar comunicación
  - Envía el bit correspondiente para verificar comunicación
  - Espera bit de respuesta
- Tarea: escritura a la memoria a bordo de la tarjeta
  - Envía bit de escritura.
  - Extrae el archivo intel Hex de alguna carpeta dentro de la PC.
  - Transmite el archivo por puerto serie.
  - Si existe un error en la comunicación, recibe bit de error y detiene la transmisión. Indica que se presentó un error y espera nuevas instrucciones.
  - Espera bit que indica que se terminó la transmisión.
- Tarea: lectura de la memoria a bordo de la tarjeta
  - Envía bit de lectura.
  - Recibe el archivo extraído de la memoria, si detecta algún error detiene la transmisión e indica que se presentó un error.
  - Imprime en pantalla el archivo recibido.
  - Termina la transmisión al recibir el bit que indica que el *bitstream* de la memoria se extrajo completamente.
- Tarea: borrado de la memoria a bordo de la tarjeta
  - Envía bit de borrado.
  - Espera el bit que indica que se completo la transmisión.
- Tarea: reconfiguración del FPGA
  - Envía el bit de reconfiguración y espera a que finalice la misma.
  - Si existe algún error en la reconfiguración. Recibe el bit de error y espera nuevas instrucciones.

6.3.3 Tarjeta del subsistema de control de orientación

6.3.3.1 Firmware de recepción, transmisión y almacenamiento en memoria

El *firmware* ha sido desarrollado usando el lenguaje de programación C, con el uso del compilador CCS del *software* MPLAB de Microchip. El *firmware* permite que el PIC, residente en la tarjeta de SMIN, sea controlado desde la estación terrena, desde donde se envían, por un lado el *bitstream* de reconfiguración para el FPGA y por otro, comandos que permiten interactuar con la GUI de la estación terrena y ejecutar diversas tareas de

control y verificación de comunicación, así como la escritura y lectura de datos a la memoria a bordo. En el caso, por ejemplo de la escritura, el *bitstream* de reconfiguración, una vez que ha sido transmitido desde la estación terrena, en el PIC se convierte a un archivo binario, para posteriormente almacenarlo en una memoria EEPROM utilizando el bus de comunicación I^2C. Para la lectura, utilizando la opción correcta en la GUI, se envía el comando para extraer el *bitstream* de la memoria y convertirlo nuevamente a un archivo Intel HEX para después transmitirlo a la estación terrena y verificar los datos contenidos de la memoria. En la figura 6.17 se muestra el diagrama de flujo con el cual opera el PIC.

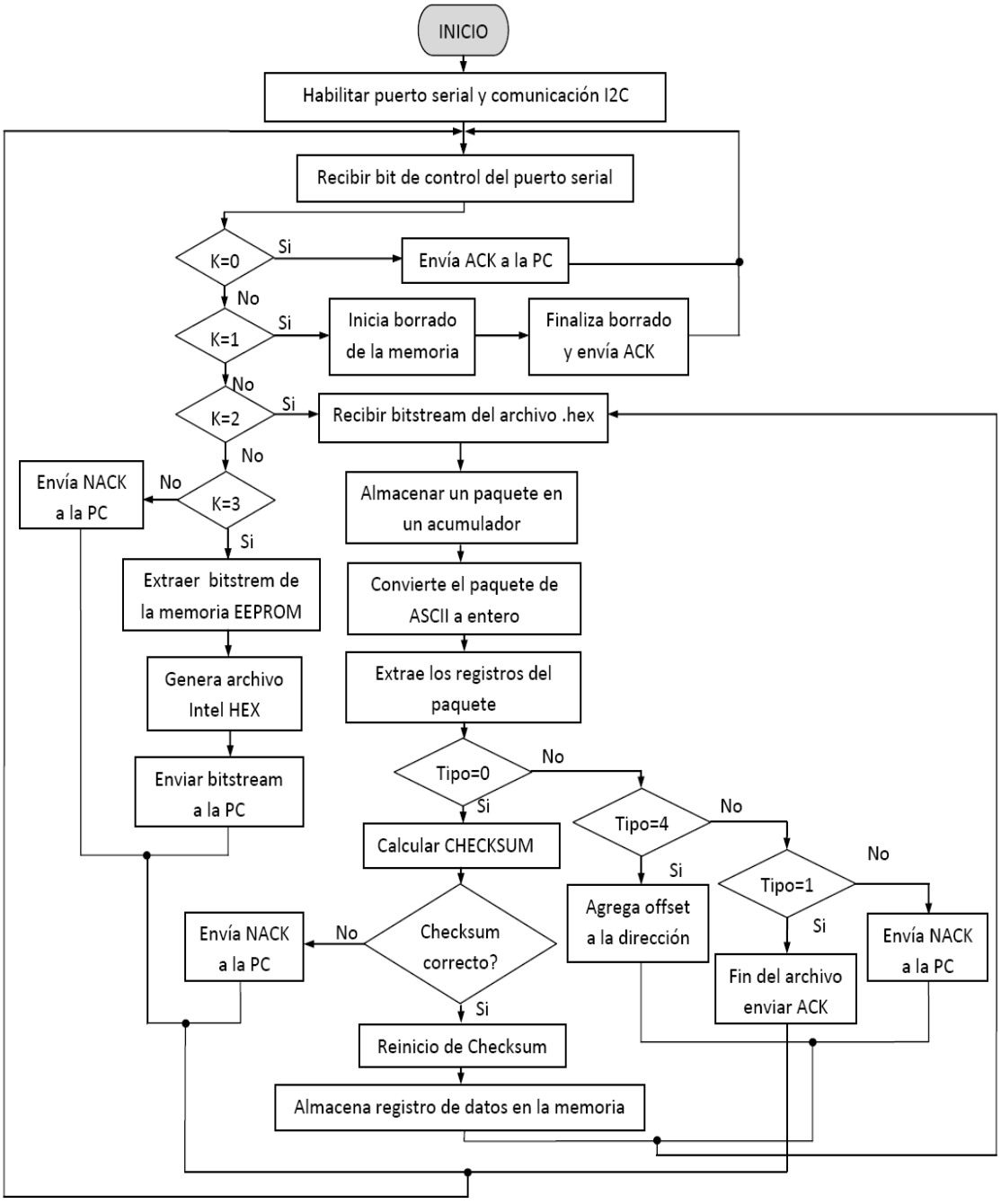


Figura 6.17. Diagrama de flujo del firmware de transmisión y adquisición de datos del PIC.

6.3.3.2 Firmware de reconfiguración ajustado para el PIC

El *firmware* de reconfiguración permite configurar al FPGA utilizando un PIC. La lógica de reconfiguración es similar a la lógica que lleva a cabo el *software* de reconfiguración ejecutado en la PC, a diferencia de que en el PIC se extrae el *bitstream* de reconfiguración de la memoria EEPROM en serie, para después transmitirla a los puertos de configuración JTAG del FPGA utilizando terminales de un puerto de propósito general del PIC.

El proceso de diseño del *firmware* de reconfiguración en el PIC tuvo como punto de arranque la transferencia del *software* que permite la reconfiguración desde una PC, a un PIC. El tiempo de diseño resultó largo y el proceso complicado, debido a los recursos limitados con los que cuenta un PIC, en comparación con los que ofrece una PC. Otra serie de retos que implicó esta transferencia, y que se resolvieron exitosamente, estuvieron relacionados en primer lugar, con el hecho de que si bien, el *software* de configuración de referencia en la PC se encuentra desarrollado en lenguaje C++, los compiladores utilizados tanto para la PC como para el PIC emplean diferentes comandos y definición de tipos de variables. Otro más estuvo relacionado con las velocidades de procesamiento diferentes de la PC y del PIC; mientras que para la PC su velocidad de procesamiento es de 1 GHz, para el PIC ronda alrededor de los 40 MHz, lo cual impacta en el desempeño de las tareas que se ejecutan en ambos sistemas. Debido a esta serie de retos que se presentaron fue necesario comprender las tareas que realiza cada función que integra el *software* de reconfiguración, casi en su totalidad.

El desarrollo del *firmware* se realizó en la plataforma de *software* MPLAB con el compilador CCS de Microchip. En la Figura 6.18 se muestra el flujo del proceso en bloques del *firmware*, se observa que este flujo es similar al de la PC como ya se había mencionado antes.

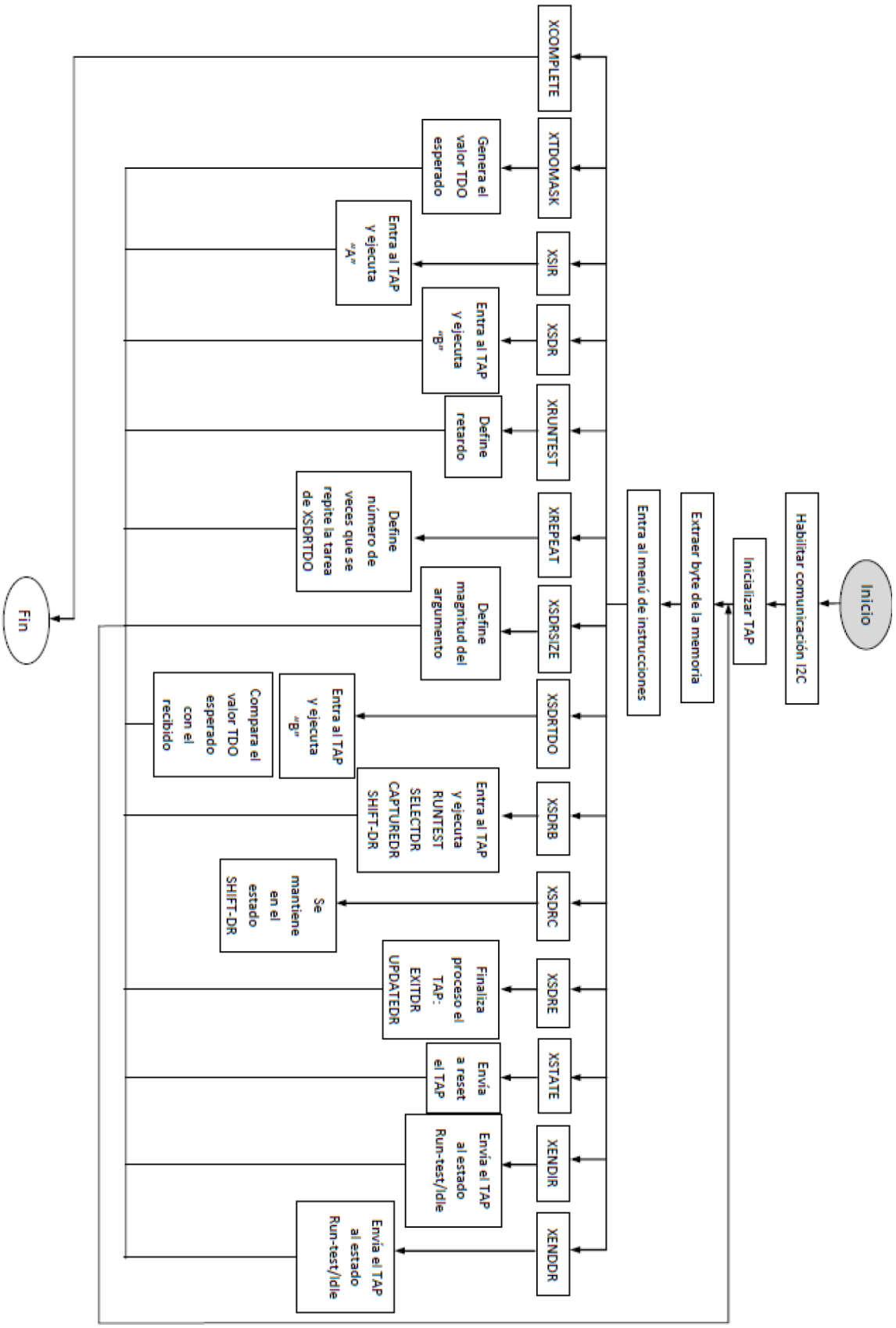


Figura 6.18. Diagrama de flujo del proceso de reconfiguración utilizando el PIC.

Una vez validando el funcionamiento del *firmware*, tanto de recepción, transmisión y almacenamiento en memoria, así como el de reconfiguración cada uno por separado, fueron integrados para realizar todas las tareas en conjunto dentro del PIC. Este proceso fue sencillo ya que sólo se requirió agregar una función más al menú programado en el *firmware* de transmisión y adquisición de datos, y añadir cada uno de los programas y bibliotecas que integran el *firmware* de reconfiguración. El flujo de proceso de los *firmwares* sigue siendo el mismo.

Una vez descritas las herramientas en *software* y *hardware* utilizadas para las pruebas de validación del proceso de reconfiguración remota del FPGA que integra la tarjeta de SMIN, el siguiente paso a seguir fue diseñar una propuesta innovadora de la nueva tarjeta del subsistema de control de orientación para el satélite educativo SATEDU, con los recursos necesarios para evaluar el funcionamiento de diferentes algoritmos de control de apuntamiento satelital. En el siguiente capítulo se aborda el tema sobre el desarrollo del esquema propuesto que contendrá el *hardware* y *software* necesario para operar dentro y fuera de SATEDU.

CAPÍTULO 7

Desarrollo de pruebas de validación integrando hardware y software, para procesos de reconfiguración total de FPGAs de forma remota

Un sistema electrónico es un grupo de componentes o elementos que tienen tareas particulares y que en grupo realizan tareas más complicadas. Un sistema que posee una arquitectura compleja se puede conceptualizar de forma modular, dividiéndolo en bloques o en subsistemas más sencillos, los cuales al ser ensamblados al final del proceso, realizan las tareas del sistema completo. En este sentido, el diseño del bloque de reconfiguración remota para el FPGA, en la tarjeta del sistema mínimo, requirió para su integración final de una serie de desarrollos y subsecuentes pruebas en las dos principales vertientes que lo conforman: *hardware* y *software*.

En este capítulo se describirán algunas de las pruebas parciales más importantes que se realizaron para la validación e integración final del bloque de reconfiguración remota del FPGA.

Introducción

Para validar la tarjeta de sistema mínimo fue necesario dividir el proceso de desarrollo en seis pruebas básicas. En cada una de ellas, se validó el funcionamiento óptimo de los módulos que componen al bloque, al igual que el funcionamiento de cada una de las herramientas diseñadas en *hardware* y *software* asociadas a él.

Las pruebas comenzaron con la validación del módulo de recepción, transmisión y almacenamiento en memoria, utilizando una GUI, la cual permitió el envío de un archivo binario hacia un MCU, utilizando un protocolo de comunicación serie. Las siguientes pruebas realizadas incluyeron por un lado, la validación del cable *parallel III*, construido *ex profeso* para realizar la configuración del FPGA de una tarjeta de desarrollo Spartan 3E-Starter Kit, utilizando el puerto paralelo de la PC. Estas pruebas permitieron comenzar a definir el esquema final que tendría el bloque de reconfiguración remota, en términos de los componentes que requeriría, tanto en *hardware* como en *software*, basados en la técnica seleccionada para realizar el proceso de reconfiguración del FPGA documentado en hojas de especificaciones de los dispositivos utilizados y notas de aplicación del fabricante. La siguiente prueba se realizó para validar el funcionamiento del FPGA, una vez que éste fue integrado dentro de la tarjeta SMIN, para finalmente validar el funcionamiento del *software* de reconfiguración obtenido de la nota de aplicación del fabricante XAPP058.

Una vez validado el de reconfiguración y analizados los primeros resultados experimentales sobre la tarjeta de desarrollo comercial, fue posible tener la certeza de que la propuesta planteada como tema central de esta tesis era totalmente viable en términos técnicos. Basados en esta experiencia, se pudo seguir el desarrollo con mayor confianza, realizando enseguida el desarrollo del *firmware* del MCU que gestionaría las tareas asociadas con el proceso de reconfiguración remota del FPGA. Las pruebas de validación que se consideraron en esta fase de desarrollo, incluyeron pruebas del *firmware* de reconfiguración para el MCU, el cual contiene la misma lógica de control que el *software* de reconfiguración utilizado en la PC, finalizando con la validación de todo el sistema de reconfiguración remota con herramientas en *hardware* y *software* propios.

7.1 Pruebas de validación del módulo de recepción, transmisión y almacenamiento en memoria

7.1.1 Objetivo y descripción

El objetivo de esta primera ronda de pruebas fue verificar operativamente el *firmware* de recepción, transmisión y almacenamiento en memoria de un archivo en formato Intel HEX, utilizando para ello módems de RF como medio de comunicación.

La prueba consistió en que el MCU, el cual gestiona todos los procesos realizados en este módulo, recibiera un archivo transmitido desde el puerto serie de la PC (estación terrena), lo decodificara para obtener el *bitstream* del archivo origen y posteriormente lo almacenara en una memoria EEPROM por medio del protocolo de comunicación I2C. Una vez almacenado el archivo en memoria, al recibir el comando de lectura de memoria, el MCU procederá a extraer el *bitstream* de la memoria, para después generar un archivo Intel HEX a partir del *bitstream* que ha sido leído de la memoria. Finalmente, el archivo Intel HEX será transmitido de regreso hacia la PC. Una vez recibido el archivo desde el MCU en la PC, los datos se desplegarán en la pantalla de la interfaz gráfica de la estación terrena donde se podrán visualizar para verificar que el archivo fue almacenado en la memoria de forma exitosa

En la figura 7.1 se muestra un diagrama de bloques del módulo montado para realizar esta prueba.

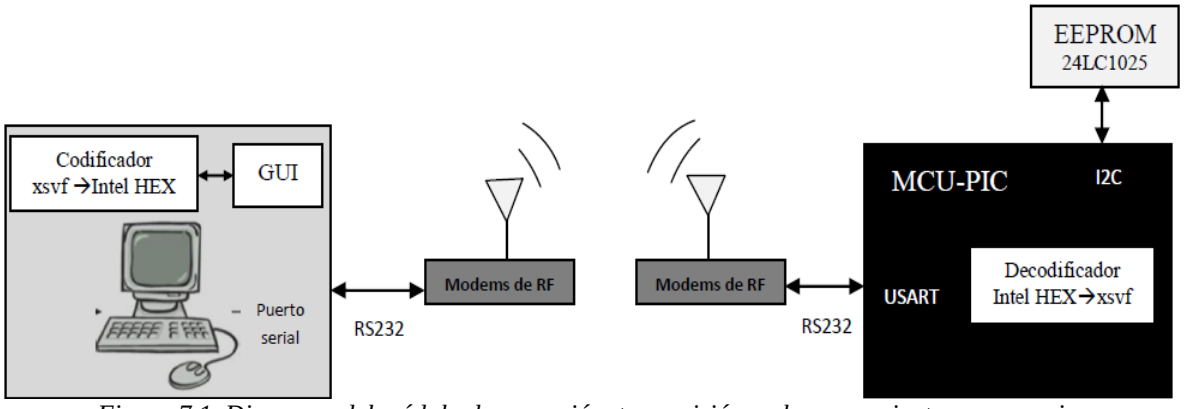


Figura 7.1. Diagrama del módulo de recepción, transmisión y almacenamiento en memoria.

Para llevar a cabo esta prueba se utilizaron herramientas en *software* y se alambró una versión de prueba de este módulo en una tarjeta de prototipos. El esquema desarrollado en la tarjeta prototipo, con la lógica de control y la transferencia de datos sirvieron como base para el diseño final de la tarjeta SMIN.

7.1.2 Hardware utilizado

A continuación se listan algunos de los equipos y componentes que fueron utilizados para la realización de esta prueba.

- **Módems de RF**
- **Microcontrolador PIC18F4520**
- **Memoria EEPROM 24LC1025**
- **Transceptor MAX232 y conector DB9**
- **Tarjeta de prueba**
  En esta tarjeta se integran e interconectan todos los dispositivos: el MCU, la memoria EEPROM y el transceptor MAX232, con su respectivo *hardware* de soporte (resistencias, capacitores, cristal, etc.). En la figura 7.2 se muestra este módulo integrado en la tarjeta de prototipos.

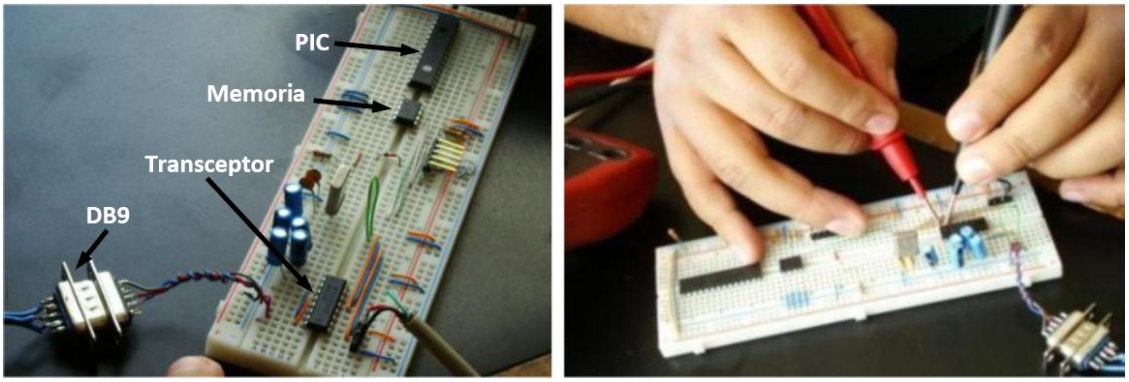


Figura 7.2. Integración del hardware dentro de la tarjeta de prueba.

7.1.3 Software utilizado

A continuación se listan los diferentes programas que fueron utilizados para la realización de esta prueba.

- ***Software* Intel_Hex**
  Utilizado para generar un archivo Intel HEX a partir de algún archivo binario existente en la PC.
- **GUI de estación Terrena**
  Utilizado para interactuar con el MCU y controlar cada tarea que se ejecuta.
- ***Firmware* de recepción, transmisión y almacenamiento en memoria**
  Utilizado para llevar a cabo el almacenamiento, lectura y borrado de la memoria a bordo, así como la recepción y transmisión del *bitstream* en serie.

7.1.4 Desarrollo y resultados

Desarrollo

La prueba inicia con la generación de un archivo Intel HEX a partir de un archivo de texto, que contiene caracteres aleatorios, creado previamente en la aplicación de *Windows block* de notas. Para tal efecto, se abre el archivo ejecutable del *software* Intel_Hex y se escribe el nombre del archivo que se va a utilizar, agregando la extensión del mismo. De forma automática el *software* comienza a generar el formato Intel HEX, almacenándolo en un archivo con extensión .hex dentro de la misma carpeta donde se ubica el archivo ejecutable. Este proceso se muestra en la figura 7.3.

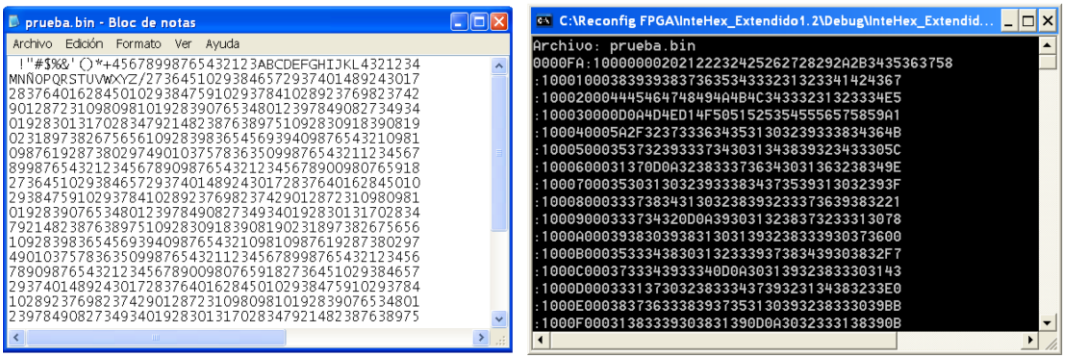


Figura 7.3. Generación de un archivo Intel HEX.

Una vez finalizado el procedimiento descrito anteriormente, se programa el MCU con el *firmware* de recepción, transmisión y almacenamiento en memoria. Una vez programado, se conectan los modems de RF, uno en la terminal DB9 de la tarjeta de prueba y el otro en el puerto serie de la PC (RS232 o USB por medio de un cable convertidor) y son separados a un metro de distancia aproximadamente. Después, se energizan todos los componentes, y en el lado de la PC se ejecuta la GUI para comenzar a interactuar con el MCU. Se inicia la prueba seleccionando el puerto a utilizar y definiendo los parámetros de la comunicación serie, como se muestra en la figura 7.4.

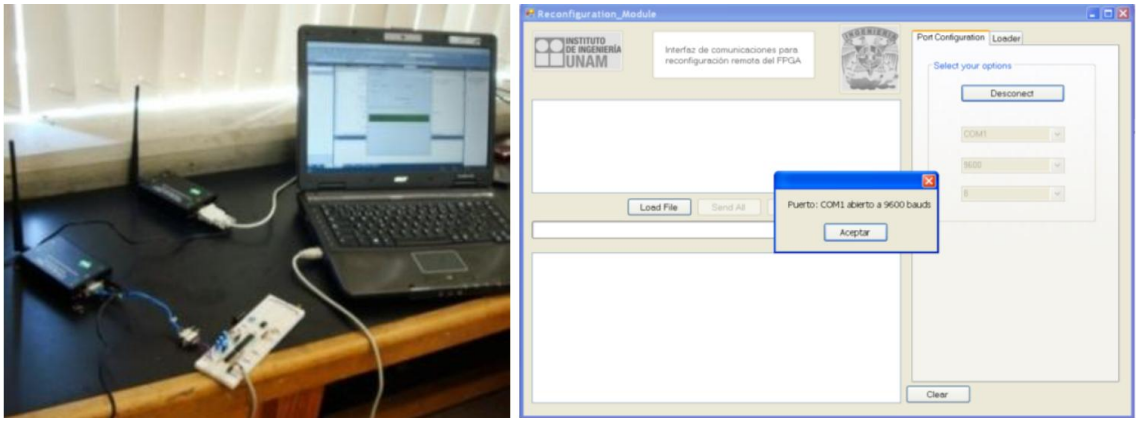


Figura 7.4. Conexión física entre la tarjeta de pruebas y la PC e inicialización de la GUI.

Una vez que el puerto fue abierto, se verifica que la comunicación entre el MCU y la PC se realizó de manera correcta, lo cual se comprueba pulsando el botón *Check* en la GUI, el cual le envía un comando al MCU para que éste regrese una respuesta pre-designada, la cual es posible visualizar en la pantalla de la GUI. Una vez que fue entablada la comunicación, se presiona el botón *Write* para indicar al MCU que debe prepararse para recibir el *bitstream* y realizar la escritura de datos en la memoria. Después al pulsar el botón *Load File* se abre una ventana en la cual seleccionamos el archivo que debe ser transmitido. Por último se teclea el botón *Send All* para comenzar a transmitir el *bitstream*. Al finalizar esta tarea, automáticamente la GUI envía un mensaje que indica que la programación terminó y detiene la transmisión. Estos procesos se muestran en la figura 7.5.

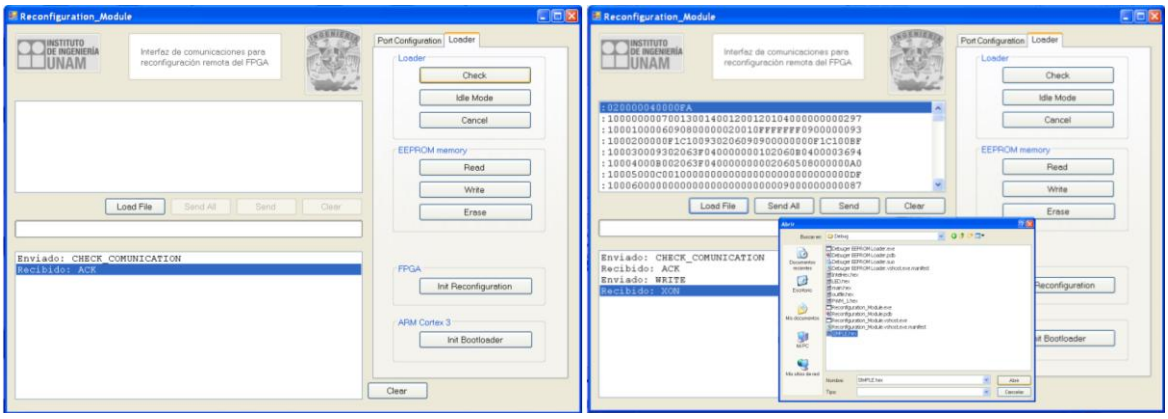


Figura 7.5. Prueba de comunicación y proceso de escritura en la memoria EEPROM.

Una vez que el *bitstream* fue almacenado en la memoria, se vuelve a verificar la comunicación por cualquier anomalía que se hubiera presentado y después se pulsa el botón *Read*, también en la GUI, para indicar al MCU que se desea leer el *bitstream* de la memoria. Inmediatamente después de que se pulsa el botón, se comienza a visualizar en pantalla el *bitstream* almacenado, como se puede observar en la figura 7.6

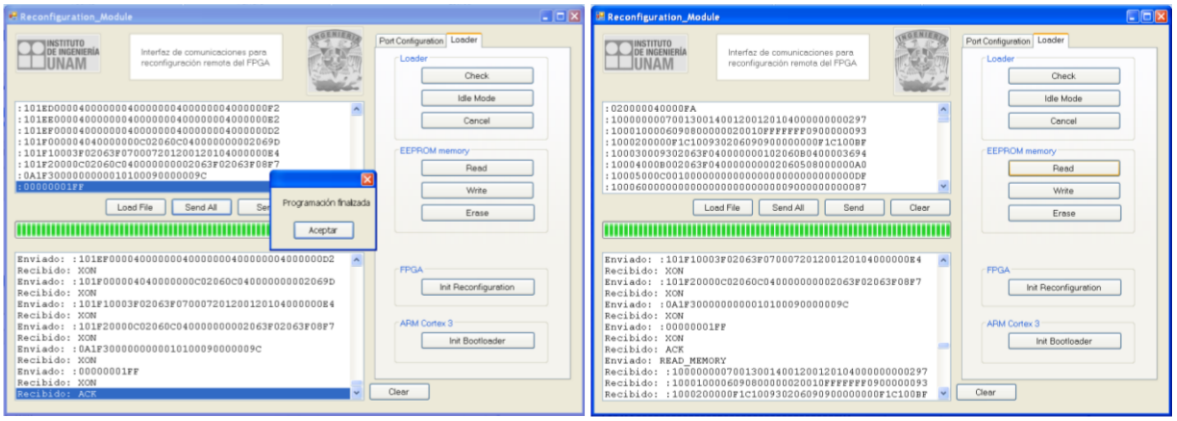


Figura 7.6.Fin de la transmisión del bitstream y proceso de lectura de la memoria.

Resultados

Para las primeras pruebas se eligió una velocidad de transmisión de 9600 baudios y se generó un *bitstream* parcial en formato Intel HEX, con un tamaño aproximado de 20 KBytes, el cual fue transmitido de la PC hacia el microcontrolador, donde después se almacenó en la memoria EEPROM. Después se envió el comando para realizar la lectura de la memoria y se recibió el *bitstream* almacenado, verificando que fue recibido y almacenado correctamente. Posteriormente se realizaron más pruebas pero ahora con un *bitstream* aún más extenso, con un tamaño de 100 KBytes, obteniendo resultados favorables, ya que en el 100% de las pruebas que se llevaron a cabo, los procesos se efectuaron de forma exitosa. El tiempo del proceso de recepción, almacenamiento del *bitstream* en memoria y lectura de la misma, para estas últimas pruebas, fue de aproximadamente 10 minutos.

Posteriormente se aumentó la velocidad de transmisión a 19200 baudios con el mismo *bitstream* de 100 KBytes, sin embargo, a esta velocidad se presentaron errores durante la transmisión y almacenamiento del *bitstream* en algunos casos, lo que llevó a la cancelación de la etapa de transmisión. Se continuaron realizando más pruebas, de las cuales aproximadamente un 90% resultaron exitosas. Cabe señalar que en el caso de los procesos de ejecución de tareas diversas, así como en la lectura del *bitstream* de la memoria, el 100% de las pruebas realizadas resultaron exitosas. Es necesario también mencionar que el tiempo de duración de los procesos de almacenamiento y lectura de datos, para las pruebas efectuadas con esta nueva tasa de velocidad fue de aproximadamente 5 minutos.

Se finalizó realizando una última serie de pruebas, ahora con una tasa de 38400 baudios, en las cuales, a diferencia de las pruebas antes descritas, aproximadamente sólo un 60% resultaron exitosas. En el caso de los procesos de ejecución de tareas diversas, así como en la lectura del *bitstream* de la memoria, el 95% de las pruebas realizadas resultaron exitosas. Cabe mencionar que en esta ocasión el tiempo de duración de los procesos de almacenamiento y lectura fueron de aproximadamente de 3 minutos para escribir y leer un archivo de 100 KBytes.

Gracias a estos resultados se pudo dar por terminada esta primera prueba, concluyendo lo siguiente:

- Para trabajar a velocidades mayores de 38400 baudios es necesario robustecer aún más el *software* de la GUI y el f*irmware* del microcontrolador, es decir, diseñarlo a prueba de cualquier falla que pueda presentarse al momento de ejecutar alguna tarea, particularmente en las de comunicación serie.
- Es necesario trabajar a una velocidad de transmisión mayor a 38400 baudios, para disminuir el tiempo al transferir el *bitstream*, ya que el tamaño típico de un archivo de reconfiguración en formato Intel HEX para el FPGA XC3S100E de la familia Spartan3E, utilizado en la tarjeta de SMIN, es de 203 KBytes, y para el FPGA

XC3S1600E que será utilizado para la tarjeta del subsistema de control de orientación, es de 2.01 MByte.

- En términos generales, se validó operativamente el funcionamiento óptimo de este módulo, con algunos detalles presentes que, sin embargo no interfirieron con el flujo de las pruebas realizadas.

7.2 Pruebas de reconfiguración utilizando la tarjeta de desarrollo Spartan3E-SK y la herramienta iMPACT

7.2.1 Objetivo y descripción

El objetivo de esta prueba fue validar el funcionamiento del cable *parallel III*, utilizado en la realización de pruebas posteriores.

La prueba consistió en reconfigurar el FPGA de la tarjeta de desarrollo Spartan 3E-SK a través de las terminales dedicadas para configuración por medio del protocolo JTAG, utilizando para ello el cable *parallel III* y la herramienta iMPACT de la suite ISE de Xilinx.

Para esta prueba fue necesario utilizar las herramientas en *hardware* y *software* que se describen a continuación.

7.2.2 Hardware utilizado

A continuación se listan las herramientas en hardware que fueron utilizados para la realización de esta prueba.

- **Cable *parallel III***
  Utilizado para acondicionar el nivel de voltaje y la lógica de las señales provenientes del puerto paralelo de la PC al nivel de voltaje de las señales dedicadas para la configuración en el FPGA.
- **Tarjeta de desarrollo Spartan-3E**
  Utilizada para probar la reconfiguración del FPGA utilizando el cable *parallel III* por medio de las terminales JTAG de la propia tarjeta, validando posteriormente las arquitecturas de cómputo descargadas por este medio, usando los recursos de depuración a bordo de la tarjeta de desarrollo (*switches* y *leds*).

7.2.3 Software utilizado

A continuación se listan los diferentes programas que fueron utilizados para la realización de esta prueba.

- **Suite ISE de Xilinx**
  Utilizado para diseñar y sintetizar la arquitectura de cómputo que será montada en el FPGA.
- **Herramienta iMPACT de Xilinx**
  Utilizado para generar un archivo en formato .MCS, necesario para ser almacenado en la memoria PROM integrada en la tarjeta de desarrollo, para llevar a cabo el proceso de almacenamiento en esta memoria.

7.2.4 Desarrollo y resultados

Desarrollo

El primer paso para llevar a cabo esta prueba fue diseñar y sintetizar una arquitectura que describe un sistema digital sencillo, utilizando el navegador de proyectos en ISE, logrando con ello generar un archivo de salida .bit. Después, con la herramienta iMPACT generamos un archivo con formato MCS. Hay que tomar en cuenta que para reconfigurar el FPGA a través del protocolo JTAG en esta tarjeta de desarrollo, no es posible hacerlo directamente. Es necesario almacenar el archivo de reconfiguración en una memoria PROM, la cual se encuentra integrada en la misma tarjeta, para posteriormente llevar a cabo el proceso de configuración del FPGA utilizando el modo de configuración *Master Serial*, el cual se selecciona por medio de unos *jumpers* que también están montados en la tarjeta de desarrollo.

Por tal motivo, fue necesario hacer un cambio de un archivo.bit a formato MCS, ya que en este último formato es como se almacena el archivo de reconfiguración en la memoria PROM. Más adelante se hablará sobre el proceso para generar este tipo de archivos y así llevar a cabo el proceso de configuración del FPGA.

Posteriormente se ejecuta la herramienta iMPACT, en ella se genera un proyecto nuevo, lo más recomendable es almacenarlo dentro de la carpeta donde se ubica el archivo de reconfiguración que se va a utilizar. Una vez hecho lo anterior, se selecciona la tarea a realizar en este proyecto, la cual consistirá en la generación de un archivo para una memoria PROM. Una vez seleccionada esta tarea, se elige la memoria PROM de Xilinx y el formato MCS que se va a generar. Los procesos antes descritos se pueden observar en la figura 7.7.

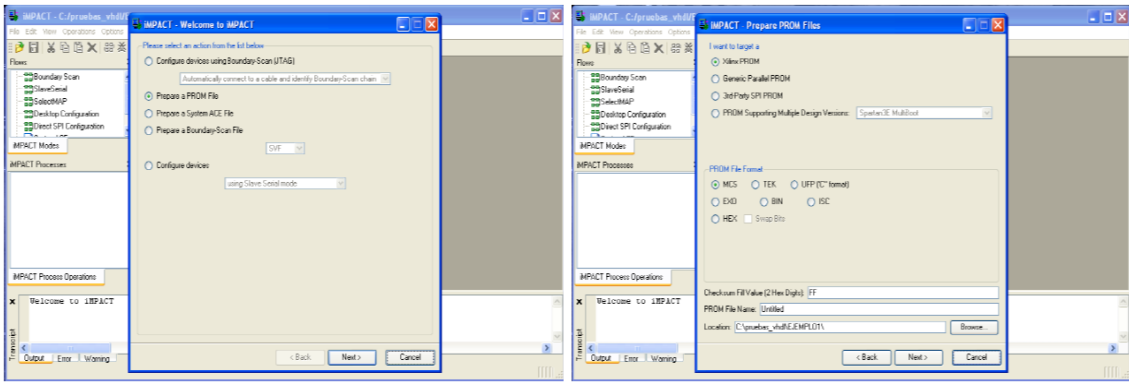


Figura 7.7. Proceso para generar un archivo MCS.

Posteriormente se despliega una ventana donde se selecciona el archivo .bit a utilizar, una vez que ha sido elegido el archivo, se selecciona la opción programar con lo que iMPACT automáticamente comienza a generar el archivo .MCS. Este proceso se muestra en la figura 7.8.

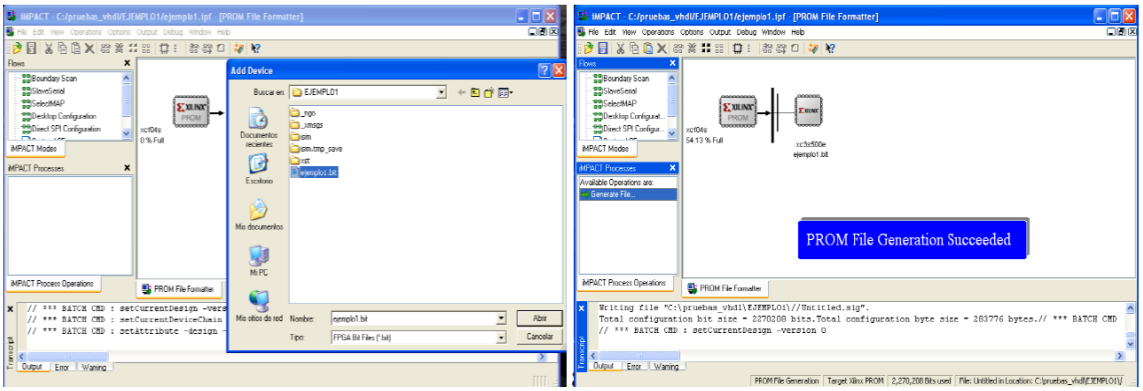


Figura 7.8. Proceso para generar un archivo MCS.

Una vez generado el archivo MCS, se conectan las terminales dedicadas para el puerto JTAG de la tarjeta Spartan3E-KS con el puerto paralelo de la PC a través del cable *parallel III* y se energizan tanto la tarjeta de desarrollo como el módulo de control del cable. En la figura 7.9 se muestra la imagen de la conexión.

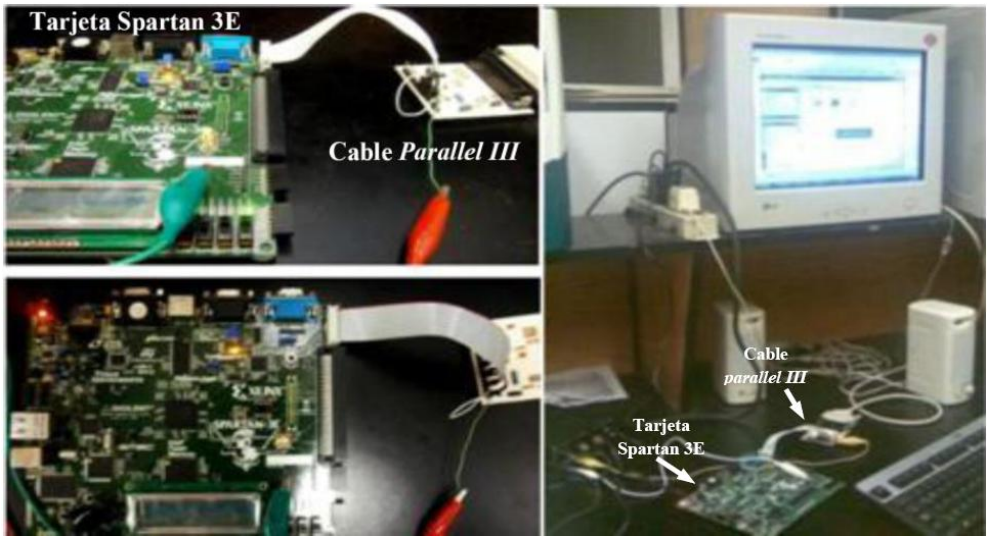


Figura 7.9. Enlace entre la tarjeta de desarrollo y la PC usando el cable parallel III.

Después, sin cerrar iMPACT seleccionamos un icono ubicado en la parte superior derecha de la pantalla, el cual inicializa la cadena JTAG, la cual automáticamente reconoce los dispositivos conectados a ella, mostrándolos en pantalla. En la tarjeta Spartan 3E-SK se reconocen los dispositivos FPGA XC3S500E, una memoria *Flash* XCF04S y un CPLD XC2C64A. Enseguida se abre una ventana para descargar el archivo al dispositivo seleccionado, para la prueba se seleccionó la memoria PROM dentro de la cadena y el archivo MCS utilizado para posteriormente programar la memoria. Este proceso puede observarse en la figura 7.10.

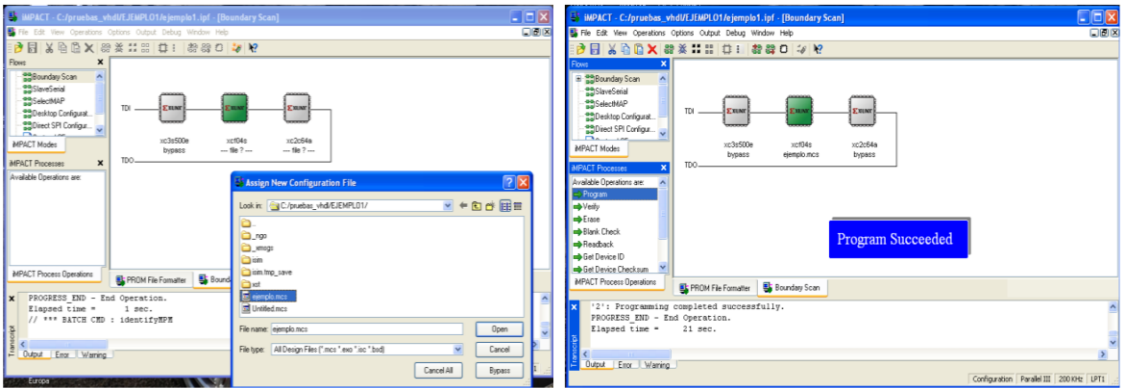


Figura 7.10. Proceso de programación de la memoria.

Una vez almacenado el archivo en la memoria, la reconfiguración del FPGA se realiza mediante el cambio de modo de configuración JTAG al modo *Master Serial*, lo cual se logra ubicando unos *jumpers* que definen del modo de configuración del FPGA de la tarjeta de desarrollo (ver capitulo 3). Después de realizar este cambio se presiona el botón PROG para reconfigurar al FPGA.

Resultados

Cabe señalar que uno de los primeros inconvenientes técnicos que se enfrentaron durante la realización de esta prueba y que tuvo que ser solucionado rápidamente para la obtención de resultados, estuvo relacionado con el reconocimiento por parte del *software* de los dispositivos conectados por medio de la cadena JTAG. Habitualmente al momento de ejecutar el *software* iMPACT éste realiza un reconocimiento automático de cada uno de los dispositivos conectados a la cadena JTAG, sin embargo en las primeras pruebas de este sistema, el *software* no reconocía a los dispositivos de la tarjeta Spartan 3E (FPGA, CPLD y memoria *Flash* PROM).

Finalmente, se detectó que el problema se debió a la incompatibilidad de los niveles de voltaje entre el puerto de reconfiguración JTAG del FPGA y el cable *parallel III*. Este problema se solucionó agregando un divisor de voltaje, lo cual resultó como una solución adecuada ya que no afectó el flujo de datos con lo que se logró entablar la comunicación satisfactoriamente, y por lo tanto fue posible llevar a cabo el proceso de reconfiguración del FPGA sin mayores problemas.

Una vez que fue solucionado el problema de la detección de los dispositivos conectados en la cadena JTAG, se realizó la reconfiguración del FPGA, almacenando en la memoria PROM a bordo de la tarjeta de desarrollo una arquitectura de cómputo que describía el funcionamiento de algunas compuertas lógicas. Se asignaron algunos *switches* y *leds* de la misma tarjeta como entradas y salidas lógicas respectivamente para validar el funcionamiento de las arquitecturas descargadas al FPGA. Esta prueba fue realizada siete veces, construyendo diferentes arquitecturas de cómputo en cada una de ellas. De las pruebas efectuadas, el 100% resultaron exitosas, con ello logramos concluir esta segunda etapa de pruebas, validando el funcionamiento del cable *parallel III*. Con lo que además se obtuvo una herramienta importante para validar algunas pruebas en etapas de desarrollo posteriores.

7.3 Prueba de reconfiguración utilizando la tarjeta de SMIN y la herramienta iMPACT

7.3.1 Objetivo y descripción

El objetivo de esta prueba fue validar el funcionamiento óptimo del FPGA que está integrado en la tarjeta de SMIN. La prueba consistió en configurar el FPGA de forma aislada través de su puerto de configuración JTAG. La prueba aislada se efectuó para validar individualmente cada uno de los dispositivos del sistema desarrollado, ya que de

presentarse algún problema, éste podría ser detectado y resuelto de una manera más rápida y sencilla. Para esto utilizamos el cable *parallel III*, cuya descripción y uso fue presentado en el apartado anterior.

Esta prueba fue muy similar a la anterior, a diferencia de que el proceso de configuración se llevó a cabo de forma más directa, ya que sólo se empleó el archivo de salida .bit generado luego de sintetizar la arquitectura de cómputo en el navegador de proyectos de ISE y descargado al FPGA por medio de la herramienta iMPACT.

Para esta prueba fue necesario utilizar las herramientas en *hardware* y *software* que se describen a continuación.

7.3.2 Hardware utilizado

A continuación se listan las herramientas en *hardware* utilizadas en esta prueba.

- **Tarjeta de desarrollo propia (SMIN)**
  Utilizada para montar y validar el funcionamiento de arquitecturas de cómputo, verificando el funcionamiento del FPGA una vez que ha sido descargado el *bitstream* en él.
- **Cable *parallel III***
  Utilizado para acondicionar el nivel de voltaje y la lógica de las señales provenientes del puerto paralelo de la PC al nivel de voltaje de las terminales dedicadas para la configuración JTAG de FPGA.

7.3.3 Software utilizado

A continuación se listan las herramientas en *software* que fueron utilizadas para la realización de esta prueba.

- **Navegador de proyectos de ISE - Xilinx**
  Utilizado para diseñar y sintetizar las arquitecturas de cómputo que serán montadas en el FPGA.
- **Herramienta iMPACT de Xilinx**
  Utilizada para llevar a cabo el proceso de reconfiguración del FPGA.

7.3.4 Desarrollo y resultados

Desarrollo

Para la realización experimental de esta prueba se diseñaron tres arquitecturas de cómputo, las cuales fueron sintetizadas por medio de las herramientas de *software* antes descritas y se

generaron los archivos de salida .bit correspondientes para el modelo del FPGA montado en la tarjeta SMIN (XC3S100E).

Después, se conecta físicamente el puerto JTAG del FPGA con el puerto paralelo de la PC a través del cable *parallel III* y se energizó el sistema. En la figura 7.11 se muestran las conexiones que se realizaron para la verificación de esta prueba.

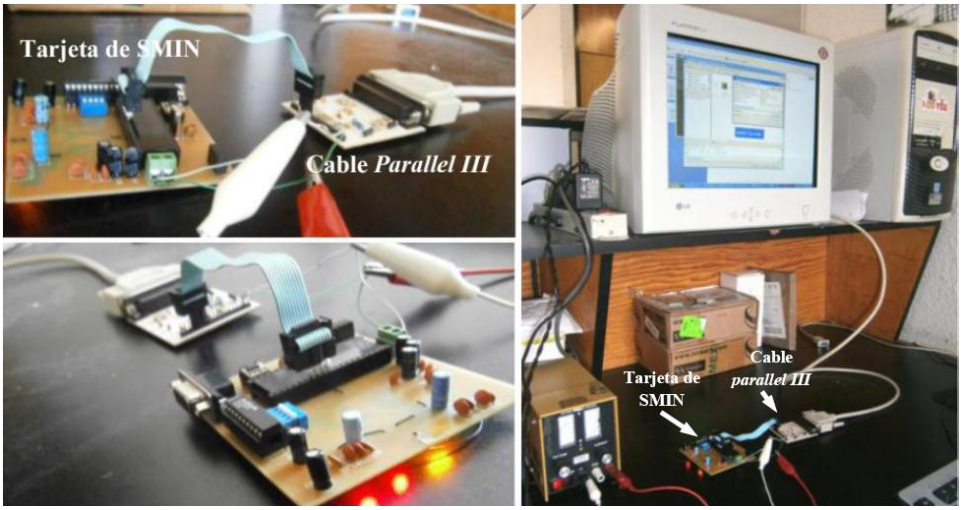


Figura 7.11. Enlace entre la tarjeta del sistema mínimo y la PC.

Una vez sintetizada la arquitectura, dentro del mismo navegador de proyectos de ISE se ejecuta la herramienta iMPACT, se selecciona la opción para configurar dispositivos a través de JTAG, luego de esto iMPACT automáticamente detecta el dispositivo conectado en la cadena. Después, como hay un solo dispositivo conectado en la cadena, se elige en el menú contextual, el archivo que se desea cargar y posteriormente se configura el FPGA seleccionando la opción a programar. En la figura 7.12 se muestra este proceso.

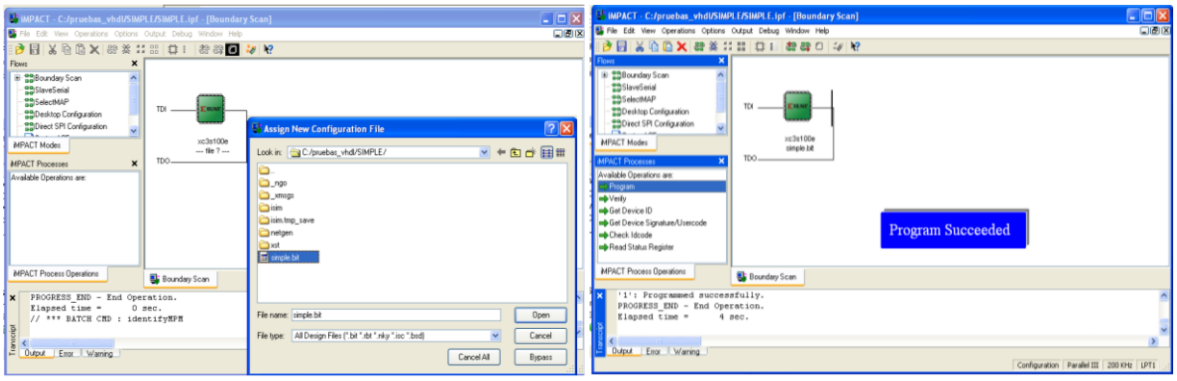


Figura 7.12. Proceso de configuración del FPGA que integra el sistema mínimo.

Resultados

Para llevar a cabo y concluir de forma satisfactoria esta etapa de pruebas, fue necesario solucionar un problema de enorme importancia. Este problema se debió a que el voltaje de rizo, a la salida de uno de los reguladores de alimentación al FPGA, afectaba su funcionamiento, ya que la lógica interna del FPGA es sensible a los cambios de voltaje que se presentan en las líneas de suministro de energía. El problema fue resuelto agregando un banco de capacitores en paralelo a la salida del regulador que se identificó como problemático, y con ello tratar de disminuir su voltaje de rizo. Una vez solucionado este problema se llevó a cabo la reconfiguración del FPGA de forma exitosa, realizando una serie de pruebas en repetidas ocasiones, en las cuales fueron cargadas tres diferentes arquitecturas de cómputo, dos de las cuales consistían en estructuras lógicas sencillas y la tercera, el núcleo de un generador de señales PWM.

Estas pruebas fueron realizadas sin ningún contratiempo, con lo cual se verificó el buen funcionamiento del FPGA una vez integrado en la tarjeta de SMIN, y con ello poder continuar con la siguiente etapa de prueba.

7.4 Prueba de reconfiguración utilizando la tarjeta de evaluación y el software de reconfiguración

7.4.1 Objetivo y descripción

Uno de los objetivos de esta prueba, fue validar el funcionamiento del *software* de reconfiguración para la PC. Esta prueba se realizó también con la finalidad de transferir la lógica del *software* que se ejecutaba en la PC al *firmware* que sería montado en el MCU, y con ello tener una relativa seguridad de que si se presentaba algún error al realizar las pruebas de validación para el MCU, éste pudiera ser atribuido a los cambios efectuados en el *software* de origen o por el *hardware* de acondicionamiento entre las terminales del MCU y las terminales del FPGA.

La prueba consistió en llevar a cabo el proceso de reconfiguración del FPGA utilizando por un lado, un archivo de reconfiguración en formato .xsvf y por otro, el *software* de reconfiguración de la PC. El *software* de la PC se utilizó básicamente para controlar el flujo de datos y transmitirlos por el puerto paralelo mediante el cable *parallel III* hacia el puerto de configuración JTAG del FPGA.

Para esta prueba se utilizaron las siguientes herramientas en *hardware* y *software*:

7.4.2 Hardware utilizado

A continuación se listan las herramientas en *hardware* que fueron utilizadas para la realización de esta prueba.

- **Tarjeta SMIN**
  Utilizada para montar arquitecturas de cómputo dentro del FPGA de forma directa.
- **Cable *parallel III***
  Utilizado para acondicionar el nivel de voltaje y la lógica de las señales provenientes del puerto paralelo de la PC al nivel de voltaje de las terminales dedicadas para la configuración JTAG de FPGA.

7.4.3 Software utilizado

A continuación se listan las herramientas en *software* que fueron utilizadas para la realización de esta prueba.

- ***Software* de reconfiguración para la PC**
  Utilizado para llevar a cabo el proceso de reconfiguración del FPGA, extrayendo el *bitstream* de reconfiguración de alguna carpeta existente en la PC.
- **Herramienta iMPACT**
  Utilizada para generar el archivo .xsvf.

7.4.4 Desarrollo y resultados

Desarrollo

El primer paso a realizar en esta prueba fue generar los archivos .xsvf a partir de los archivos .bit de las arquitecturas digitales diseñadas en la prueba anterior, utilizando para ello la herramienta iMPACT.

Se ejecuta la herramienta iMPACT, cuya interfaz permite seleccionar la generación de un proyecto o bien la carga de un proyecto existente. Se selecciona generar un nuevo proyecto, lo más recomendable es almacenarlo dentro de la carpeta donde se ubica el archivo de reconfiguración que será utilizado debido a la cantidad de archivos que genera. Luego de ello, en el siguiente menú contextual se selecciona la opción de generar un archivo .xsvf. La figura 7.13 muestra *grosso* modo este proceso.

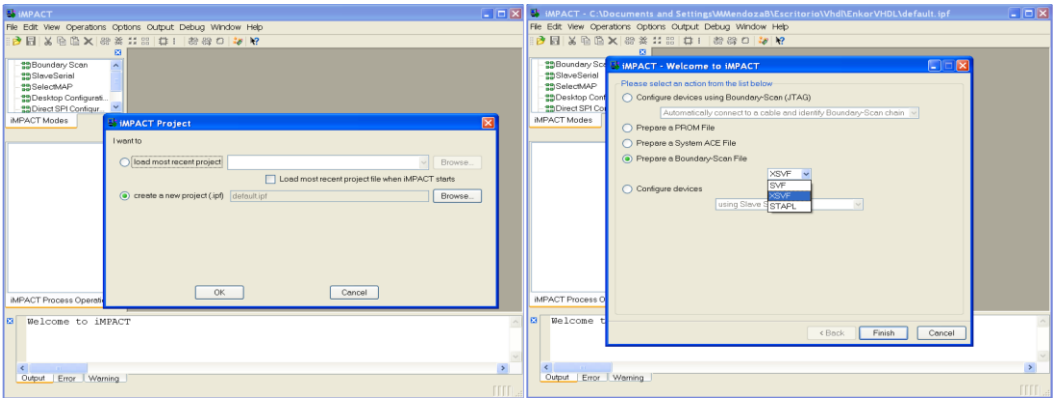


Figura 7.13. Generación de un archivo .xsvf.

Una vez seleccionada esta opción, se elige el archivo que va a ser utilizado para este proceso y finalmente se selecciona la opción programar para que se comience a generar el archivo .xsvf. Al terminar este proceso iMPACT envía un mensaje de que se generó el archivo satisfactoriamente.

Una vez creado el archivo, conectamos físicamente la tarjeta de SMIN y la PC a través del cable *parallel III*, de la misma forma que se conectó en la prueba anterior, como se muestra en la figura 7.14.

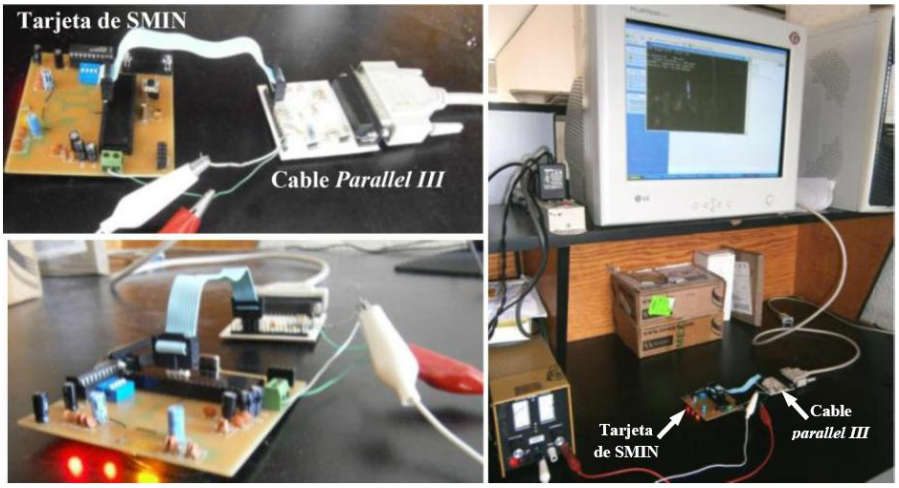


Figura 7.14. Enlace entre la tarjeta del sistema mínimo y la PC.

Posteriormente se copia el archivo .xsvf generado, a la carpeta donde se encuentra el *software* de reconfiguración y abrimos el archivo ejecutable del mismo.

Se abre una ventana de comando, la cual solicita que se ingrese el nombre del archivo a utilizar para llevar a cabo el proceso de reconfiguración. Una vez ingresado el nombre, comienza a ejecutarse todo el proceso para reconfigurar al FPGA. Después de configurar al FPGA, se muestra en la pantalla que el proceso se ejecutó de manera exitosa, desplegando también el tiempo que se tomó para realizar dicho proceso. Esto se puede observar en la figura 7.15.

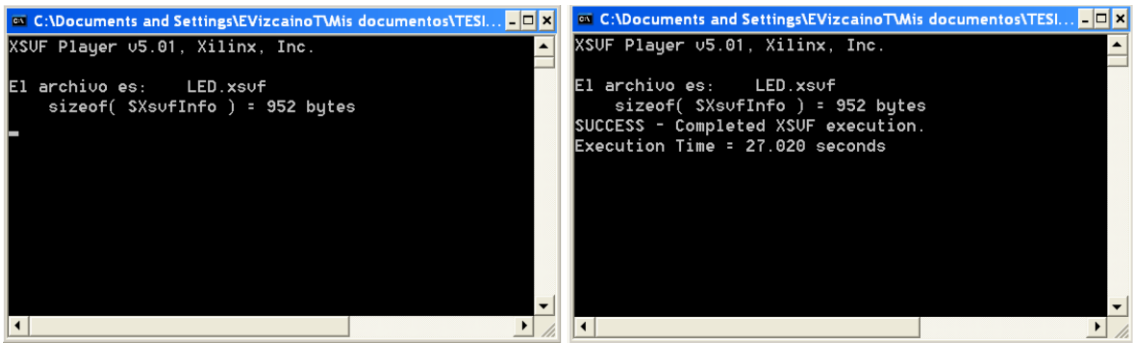


Figura 7.15. Proceso de configuración del FPGA.

Resultados

El *software* de reconfiguración que fue utilizado en la realización de esta prueba en particular, sólo permitía la configuración de dispositivos FPGA conectados en cadena, ya que opera únicamente con archivos en formato xsvf. Por ello, no se realizaron más pruebas para validar el *software* de reconfiguración utilizando la tarjeta de desarrollo Spartan3E-SK.

Para esta etapa de pruebas, los problemas que se presentaron no fueron de gran complejidad, lo cual aceleró el tiempo de validación ya que la mayoría de estos problemas se ocasionaron principalmente por malas conexiones entre la PC y el FPGA.

Se utilizaron las mismas arquitecturas digitales mencionadas en la etapa anterior, las cuales fueron cargadas exitosamente en el FPGA, pudiendo validar con ello el *software* de reconfiguración.

7.5 Prueba de reconfiguración alámbrica

7.5.1 Objetivo y descripción

El objetivo de esta prueba fue validar el funcionamiento del *firmware* de reconfiguración del MCU a bordo de la tarjeta SMIN.

La prueba consistió en reconfigurar al FPGA utilizando el MCU como medio de procesamiento para generar las señales JTAG. Se usó la GUI de estación terrena para controlar las tareas a ejecutar por el MCU y como enlace entre el puerto serie de la PC y el modulo de comunicaciones de la tarjeta SMIN se utilizó un cable serie.

Para esta prueba fue necesario utilizar las herramientas en *hardware* y *software* que se describen a continuación.

7.5.2 Hardware utilizado

A continuación se listan las herramientas en *hardware* que fueron utilizadas para la realización de esta prueba.

- **Tarjeta SMIN**
  Utilizada para montar arquitecturas de cómputo dentro del FPGA y con ello validar el funcionamiento del MCU como dispositivo de reconfiguración.

7.5.3 Software utilizado

A continuación se listan las herramientas en *software* que fueron utilizadas para la realización de esta prueba.

- ***Firmware* de reconfiguración para el MCU**
  Utilizado para llevar a cabo el proceso de reconfiguración del FPGA, extrayendo el *bitstrem* de reconfiguración de la memoria a bordo.
- ***Firmware* de recepción, transmisión y almacenamiento en memoria**
  Utilizado para llevar a cabo el almacenamiento, lectura y borrado de la memoria a bordo, así como la recepción y transmisión del *bitstream* en serie.
- ***Software* Intel_Hex**
  Utilizado para generar un archivo Intel HEX a partir de un archivo .xsvf generado previamente.
- **GUI de Estación terrena**
  Utilizada para interactuar con el MCU y controlar cada tarea que se ejecuta.

7.5.4 Desarrollo y resultados

Desarrollo

Al inicio de esta etapa de pruebas, lo primero que se realizó fue el almacenamiento del archivo de reconfiguración en formato Intel HEX en la memoria EEPROM. Para ello, se programa al MCU con el *firmware* de recepción, transmisión y almacenamiento en memoria. Después, aprovechando los archivos .xsvf generados en la etapa anterior y para cada uno de ellos, se genera un archivo en formato Intel HEX utilizando el *software* Intel_Hex, como se realizó en la primera etapa de pruebas.

Luego de generar estos archivos, se conectó físicamente la tarjeta de SMIN con la PC a través del cable serial y se energizó el sistema. Una vez energizado, se almacenó el archivo generado dentro de la memoria EEPROM, proceso similar al realizado en la primera prueba, con la diferencia de que en esta ocasión se utilizó un cable serial como se muestra en la figura 7.16.

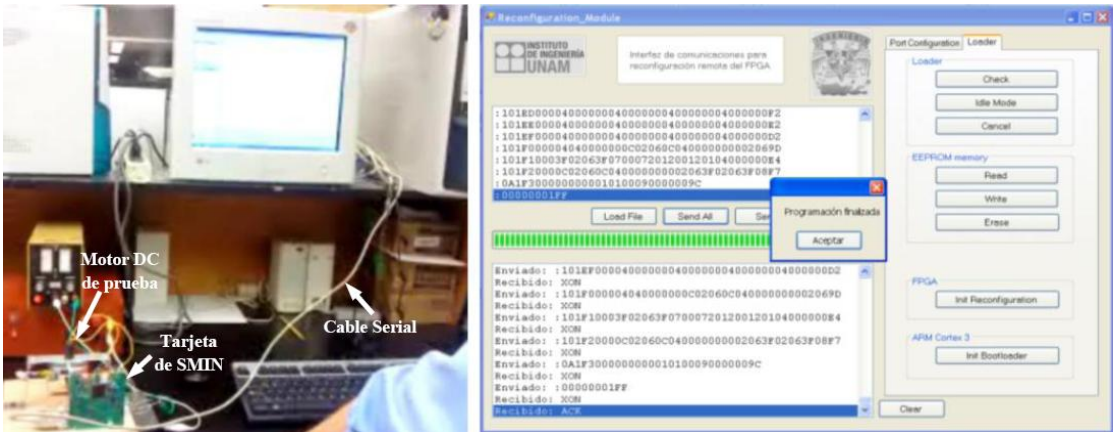


Figura 7.16. Proceso de transmisión y almacenamiento del bitstream en la memoria EEPROM.

Una vez verificado que el archivo de configuración fue almacenado de forma exitosa dentro de la memoria, mediante la lectura del archivo y el despliegue de su contenido en el espacio de trabajo superior de la GUI, se programa al MCU con el *firmware* de reconfiguración y utilizando la misma GUI se envía el comando de control para que el MCU realice la configuración del FPGA. Este proceso se muestra en la figura 7.17.

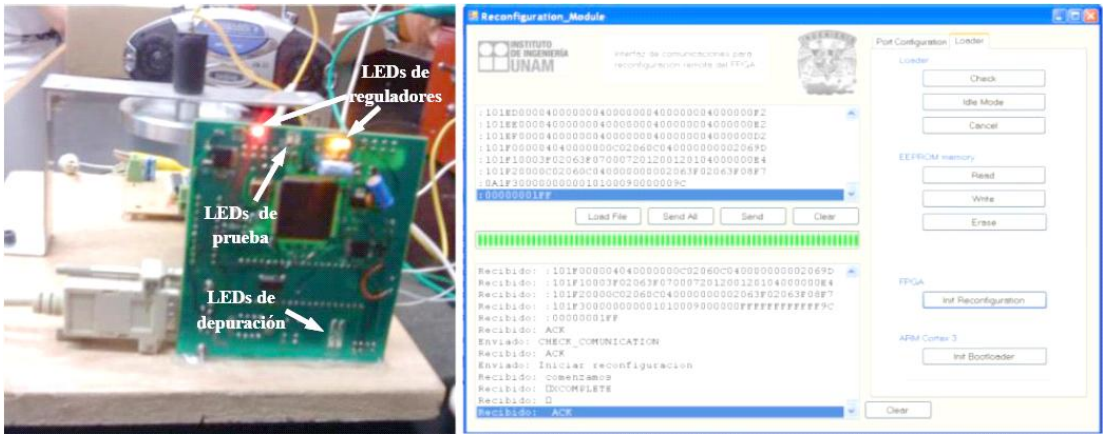


Figura 7.17. Proceso de reconfiguración del FPGA de SMIN de manera alámbrica utilizando al MCU.

Una vez terminado el proceso de configuración, se verifica que el *bitstream* de configuración se haya cargado satisfactoriamente en el FPGA, utilizando el *hardware* de prueba (*Leds* y *switches*).

Resultados

Al realizar la transmisión y almacenamiento en memoria del archivo de reconfiguración se utilizó una velocidad de comunicación de 57600 baudios, ya que esta tarea se realizó de forma alámbrica y por lo tanto no se presentaron pérdidas al transferir datos.

Al llevar a cabo la reconfiguración del FPGA con la primera arquitectura de cómputo, se presentaron una serie de problemas, de los cuales la mayoría se debieron a errores en el funcionamiento del *firmware* de reconfiguración. Los problemas que se presentaron fueron:

- Algunas variables fueron declaradas con un tamaño inferior al requerido. Esto se debió a que no todas las variables fueron adaptadas al compilador CCS para el MCU de forma correcta.
- Los retardos y tiempos de ejecución no fueron adaptados al *firmware*, por lo que estos eran muy grandes. El MCU tiene una velocidad de procesamiento mucho menor a la de una PC.
- Algunas funciones se ejecutaban de forma errónea, por lo que fue necesario replantearlas con los comandos propios para el compilador utilizado.

La solución a estos problemas requirió de meses de estudio y pruebas, debido a que aún no se conocía a fondo el funcionamiento del *software* de reconfiguración de Xilinx.

De lado del *hardware*, uno de los problemas más fuertes que se enfrentaron fue el acoplamiento en voltaje y corriente entre las terminales JTAG del MCU y las del FPGA, por lo que fue necesario ajustar el valor para las resistencias entre estos dos dispositivos.

Una vez solucionados estos problemas, se configuró exitosamente al FPGA con algunas arquitecturas de cómputo de prueba. Finalizando con el diseño y carga de un núcleo PWM para controlar la velocidad y el sentido de giro del motor que se mostró en la figura 7.18. Con estos resultados fue posible dar por concluida esta prueba satisfactoriamente, llevándonos a realizar la última prueba programada que fue la reconfiguración total remota del FPGA.

7.6 Pruebas de reconfiguración remota

7.6.1 Objetivo y descripción

Para esta prueba el objetivo fue validar el funcionamiento del *firmware* de recepción, transmisión, almacenamiento en memoria y de reconfiguración, integrados en uno solo, utilizando los módems de RF para validar el control a distancia.

La prueba consistió en integrar las dos piezas de *firmware* (comunicaciones y reconfiguración) en una sola, la cual fue programada en el MCU y con ello reconfigurar el

FPGA utilizando módems de RF como medio de comunicación serial para la transferencia de archivos de reconfiguración así como el control de tareas del MCU.

7.6.2 Hardware utilizado

A continuación se listan las herramientas de *hardware* que fueron utilizadas para la realización de esta prueba.

- **T**

  arjeta SMIN

  Utilizada para montar arquitecturas de cómputo y con ello validar el funcionamiento del MCU como dispositivo tanto de control y gestión de datos como de reconfiguración del FPGA, llevando a cabo la comunicación a través de módems de RF.

7.6.3 Software utilizado

A continuación se listan las herramientas en *software* que fueron utilizadas para la realización de esta prueba.

Firmware de reconfiguración remota

Integra la lógica del *firmware* de reconfiguración y el de recepción, transmisión y almacenamiento en memoria. Utilizado para realizar todos los procesos necesarios para llevar a cabo la reconfiguración del FPGA de forma remota

GUI de Estación terrena

Utilizada para interactuar con el MCU y controlar cada tarea que este ejecuta.

7.6.4 Desarrollo y resultados

Desarrollo

El primer paso para realizar esta prueba fue la transmisión y almacenamiento en memoria de uno de los archivos de reconfiguración en formato Intel HEX generados en pruebas anteriores. Para esto, se carga el *firmware* de reconfiguración remota en el MCU a bordo de la tarjeta SMIN y posteriormente se conecta en el puerto serial de la PC un modem de RF y por el conector serie de la tarjeta otro modem de RF, ambos separados a una distancia de un metro aproximadamente. Por último, se energizan el sistema y los dispositivos de comunicaciones. Ver figura 7.18.

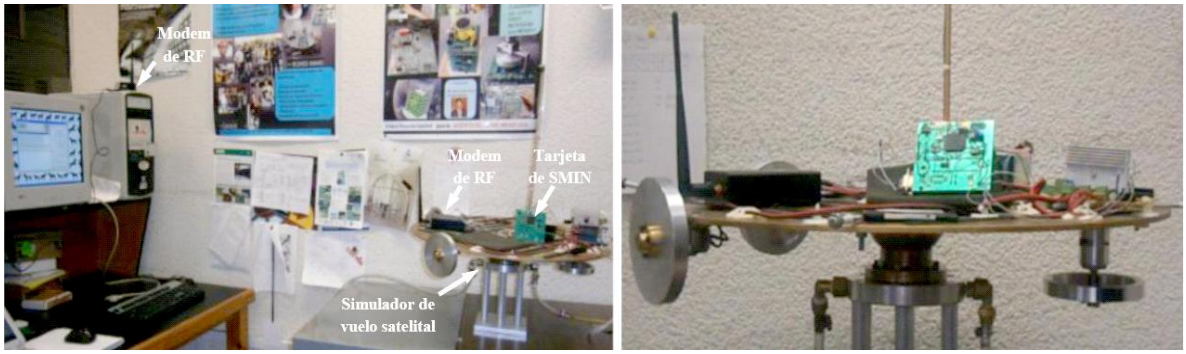


Figura 7.18. Enlace entre la tarjeta SMIN y la PC.

Después se ejecuta la GUI en la estación terrena para interactuar con el MCU y al igual que en la primer prueba, transmitimos el archivo de reconfiguración hacia el MCU para que este lo almacene en la memoria EEPROM.

Una vez almacenado el *bitstream* de reconfiguración, se envía el comando de control para inicializar la reconfiguración del FPGA. Una vez que se finaliza este proceso, el MCU envía un byte que indica que el FPGA fue configurado de forma exitosa. Este proceso se visualiza en la figura 7.19.

Figura 7.19. Proceso de reconfiguración del FPGA de forma remota.

Resultados

Debido a que la validación del módulo de reconfiguración de la tarjeta SMIN requirió un tiempo de desarrollo muy grande, no fue posible robustecer adecuadamente la GUI de estación terrena y el *firmware* de recepción, transmisión y almacenamiento en memoria del MCU (propuesta dada en la primer etapa de pruebas) por lo tanto se llevó a cabo todo el proceso de reconfiguración de forma remota a una velocidad de transmisión de 19200

baudios, lo que nos llevó un tiempo aproximado de 15 minutos, siendo el proceso de transmisión y almacenamiento en memoria el que toma un tiempo de 11 minutos.

Se realizaron seis pruebas de reconfiguración, utilizando los mismos archivos de reconfiguración en formato Intel HEX generados y utilizados en pruebas anteriores. Estas pruebas se ejecutaron de forma exitosa en su totalidad.

Se llevó a cabo una última prueba, la cual consistió en cargar un archivo de reconfiguración que integraba una arquitectura computacional que describía un módulo de control para el movimiento de una rueda inercial. Para esta prueba se colocó la tarjeta SMIN y su respectivo modem de RF sobre el simulador de vuelo satelital (basado en un cojinete de aire) del instituto de ingeniería de la UNAM.

Gracias a los resultados obtenidos en estas etapas de prueba, fue posible concluir la validación de la tarjeta de SMIN con éxito, sin embargo, debido a que el FPGA integrado en la tarjeta SMIN no cuenta con los recursos internos suficientes como para soportar algoritmos de control de orientación en tres ejes para el satélite educativo SATEDU, se comenzó con el diseño y desarrollo de la nueva tarjeta del subsistema de control de orientación de SATEDU, con la confianza de que será una tarjeta totalmente funcional, con un porcentaje mínimo de errores y que podrá validarse en corto de tiempo.

En el siguiente capítulo se hablará de la propuesta para la nueva tarjeta del subsistema de control de orientación de SATEDU mostrando los avances en el diseño de la misma, todo basado en la experiencia obtenida al desarrollar la tarjeta SMIN.

CAPÍTULO

8

Propuesta y desarrollo del subsistema de control de orientación, basado en una plataforma reconfigurable

Es en el inicio del siglo XXI cuando se observa que los circuitos integrados de lógica programable tienen amplias ventajas competitivas en relación con los circuitos integrados de arquitectura y/o funciones fijas. La lógica programable permite tomar un circuito con funciones básicas y definir la forma en que han de interactuar dichas funciones para conformar una aplicación específica. Al mismo tiempo, la programación de la configuración del circuito puede limitarse a pruebas de escritorio, sin tener la necesidad de fabricar un circuito de aplicación específica. Finalmente, otorga al diseñador la flexibilidad para modificar su diseño hasta conseguir la función y desempeño requeridos por la aplicación, a través de pruebas parciales y totales del prototipo.

Con base en las ventajas advertidas de la filosofía de diseño en plataformas reconfigurables, utilizando circuitos lógicos programables, en este capítulo se propone una arquitectura base para la implementación del subsistema de control de orientación del satélite educativo SATEDU en una plataforma FPGA.

8.1 Propuesta de la nueva tarjeta del subsistema de control de orientación basado en un FPGA

El diseño que se propone en este capítulo contempla el uso de un dispositivo FPGA como núcleo central de procesamiento del subsistema de control de orientación para SATEDU, donde serán implementadas las arquitecturas computacionales para el soporte de los algoritmos de control de orientación. Se ha considerado un FPGA de la misma familia, pero con mayores recursos lógicos que el que fue utilizado en la tarjeta del SMIN, debido a que la tarjeta del subsistema de control de orientación integrará esquemas de control de orientación satelital en tres ejes. Esto permitirá validar diferentes esquemas de control, con los cuales cuenta el Instituto de ingeniería y que se han validado exitosamente por medio de simulaciones numéricas y visualizaciones en modelos de realidad virtual de *Simulink*, y que actualmente se encuentran en fase de adaptación para su validación física. Cabe señalar que los recursos de esta nueva plataforma FPGA permitirán adicionalmente utilizarla para montar y validar el funcionamiento de un sistema de comunicaciones *software* radio, otra de las líneas de investigación dentro del grupo de sistemas aeroespaciales del Instituto de ingeniería y que forma parte de una tesis doctoral en fase de desarrollo. Esta tarjeta contará con recursos suficientes que le permitirá operar dentro y fuera de la plataforma educativa SATEDU. Esta característica tiene la finalidad de que la tarjeta electrónica pueda ser utilizada también, de forma separada, como una herramienta didáctica para el entrenamiento y experimentación en tópicos relacionados con la tecnología FPGA.

Actualmente el subsistema de control de orientación de SATEDU tiene como elemento central de procesamiento a un microcontrolador RISC de 8 bits de arquitectura en su computadora a bordo que coordina las operaciones del subsistema. El diseño que se propone en este capítulo, añadirá al subsistema de control de orientación de SATEDU un alto valor agregado, por la inherente flexibilidad del dispositivo FPGA en torno al que se diseñará y que facilitará computacionalmente la implementación de diferentes esquemas y algoritmos de control. Adicionalmente, al integrar el nuevo sistema FPGA a la plataforma SATEDU aportará una herramienta invaluable para validar en tierra de diversas estrategias de control y realización de maniobras en satélites pequeños, las cuales que serían muy difícil de anticipar al tener un satélite en el espacio, salvo por los resultados que pudieran adelantarse obtenidos a partir de simulaciones y visualizaciones en modelos de realidad virtual.

Con la experiencia obtenida luego de validar el funcionamiento de la tarjeta SMIN, fue posible en un breve lapso de tiempo proponer un nuevo diseño con un margen de error mínimo, lo cual ayudará a acelerar el proceso de desarrollo y validación del subsistema de control de orientación de forma completa.

El sistema que se propone en este capítulo, estará integrado por módulos tales como transceptor para comunicaciones seriales, alimentación de energía, interfaces de entrada/salida para señales digitales y dispositivos electrónicos de soporte, mismos que fueron utilizados en la tarjeta SMIN. Sin embargo, la tarjeta cuyo diseño se propone, cuenta con variantes significativas, las cuales dotan al sistema de una mayor capacidad para la construcción de esquemas de control más complejos. Uno de los primeros cambios fue el dispositivo FPGA, el cual cuenta con un mayor número de recursos lógicos internos, así mismo, se incluyó el uso de una memoria *flash* de 64Mbits, reemplazando a la anterior memoria EEPROM de 1 Mbit. En esta nueva versión se incluye también un mayor número de terminales físicas, lo cual permitirá la interacción con más dispositivos externos tales como una unidad de mediciones inerciales (IMU), señales de control para actuadores así como señales de datos y control para el uso de un sistema de *software* radio.

La propuesta que se presenta para el subsistema de control de orientación, considera diagramas de bloques diseñados en dos grandes segmentos: estación terrena y tarjeta electrónica del subsistema de control de orientación. En la figura 8.1 se muestra el esquema propuesto.

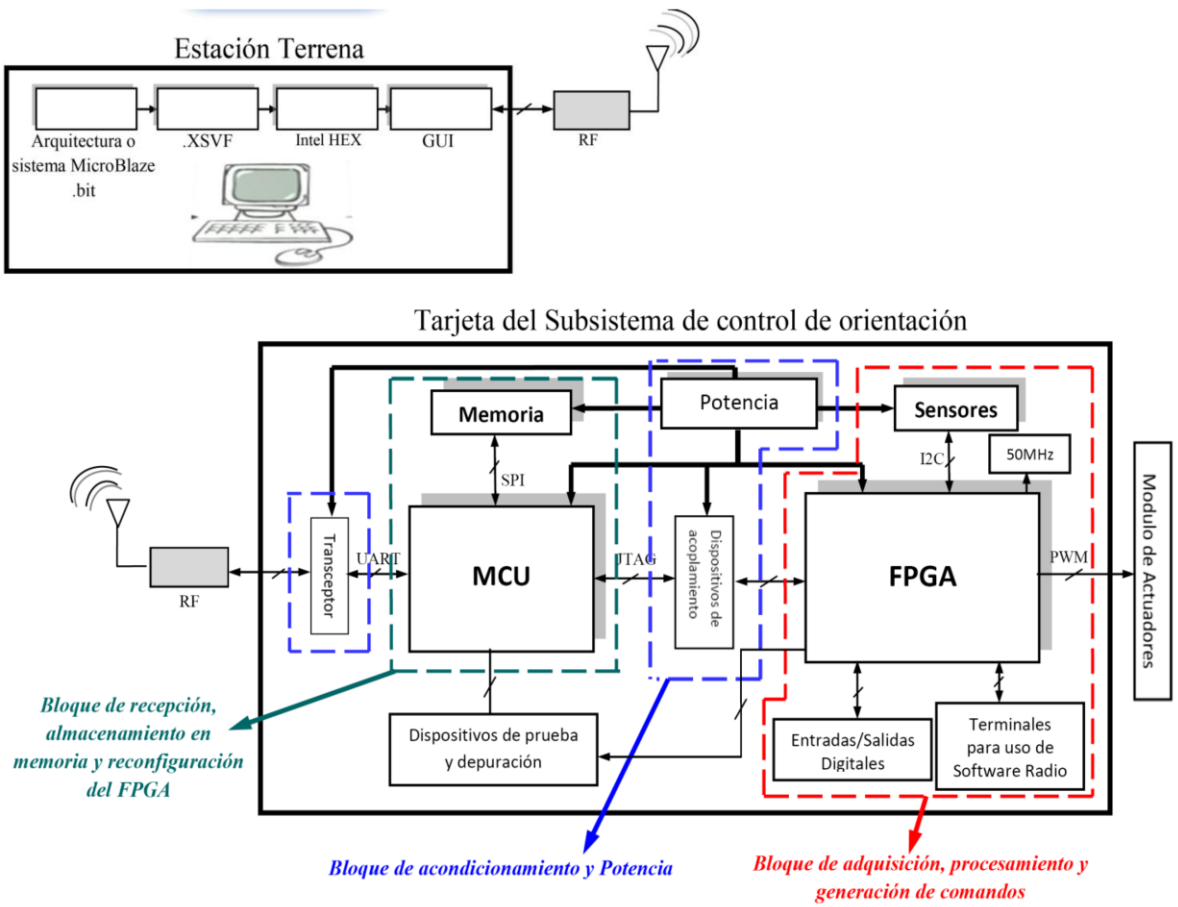


Figura 8.1 Propuesta del subsistema de control de orientación basado en FPGA.

Los dos bloques mostrados en la figura anterior se describen enseguida.

8.2 Diagrama de estación terrena

El diagrama de la estación terrena integra las mismas tareas que la versión anterior, las cuales son:

- **Diseño y síntesis de la arquitectura de cómputo reconfigurable**
  Este bloque utiliza las herramientas de diseño de la suite ISE de Xilinx para el diseño y síntesis de las arquitecturas de cómputo.
- **Creación de un archivo en formato .xsvf**
  En este bloque se genera un archivo .xsvf a partir de un archivo .bit, creado al sintetizar la arquitectura.
- **Creación de un archivo en formato Intel HEX**
  En este bloque se crea un archivo en formato Intel HEX a partir del archivo .xsvf creado previamente.
- **Interfaz Gráfica de Usuario GUI**
  Este bloque permite tomar el control del puerto serie de la PC para interactuar con el microcontrolador a bordo de la tarjeta SMIN.

8.3 Diagrama de la tarjeta del subsistema de control de orientación

El diagrama de la tarjeta del subsistema de control de orientación está integrado por tres bloques principales, donde se mantienen los dos primeros bloques, que son similares a los propuestos para la tarjeta SMIN. Estos bloques se describen enseguida.

8.3.1 Bloque de recepción, almacenamiento en memoria y reconfiguración del FPGA

Este bloque integra un MCU (en nuestro caso un PIC de Microchip de bajo consumo) y una memoria *Flash* para el respaldo del *bitstream* de reconfiguración del FPGA. En esta ocasión se omitió el cristal como base de tiempo del MCU, ya que al realizar varias pruebas dentro de la tarjeta SMIN, concluimos que no afecta significativamente bajar la base de tiempo al utilizar el oscilador interno del MCU.

En este caso utilizamos el mismo *firmware* que fue validado para la tarjeta de SMIN. El *firmware* se encuentra dividido en dos módulos, de los cuales a uno de ellos se le realizó una serie de cambios y adecuaciones para almacenar datos en una memoria *Flash*. Los módulos en los que se encuentra dividido el *firmware* se describen a continuación:

Módulo de recepción, transmisión y almacenamiento en memoria

Este módulo será el elemento responsable de recibir los comandos de control para seleccionar cada una de las tareas a ejecutar, al igual que recibir y transmitir en serie, el *bitstream* de configuración que será cargado en el FPGA. Dentro de estos comandos de control se encuentran:

- **Almacena el *bitstream* en memoria**
  Al recibir este comando, el MCU recibe por el puerto serie el *bitstream* transmitido desde la estación terrena y a través del puerto SPI del MCU lo almacena en la memoria *Flash*.
- **Lee el *bitstream* de la memoria**
  Al recibir este comando el MCU comienza a extraer el *bitstream* almacenado en la memoria y lo transmite hacia la estación terrena a través del puerto serie.
- **Borra memoria**
  Al recibir este comando el MCU envía una trama de bits específica, con la cual se eliminan todos los datos almacenados en la memoria.

Módulo de reconfiguración del FPGA

Este módulo extrae el *bitstream* almacenado en la memoria y posteriormente ejecuta el proceso de reconfiguración del FPGA.

8.3.2 Bloque de acondicionamiento y potencia

Este bloque se encuentra dividido en 2 etapas, de igual forma que en la tarjeta SMIN. Las funciones que desempeña abarcan desde el suministro de energía eléctrica necesaria para el funcionamiento de los dispositivos que componen al sistema, hasta el acondicionamiento de los niveles de voltaje lógicos entre dispositivos. A continuación se presenta la descripción de cada una de las etapas que integran a este bloque.

Potencia

La etapa de potencia está compuesta por cuatro reguladores de voltaje. Dichos dispositivos suministran energía eléctrica a los niveles requeridos, a todos los dispositivos dentro de la

tarjeta. La finalidad de integrar estos reguladores permitirá operar a la tarjeta como un sistema independiente de SATEDU. Los niveles de voltaje que suministra este módulo son 5V, 3.3V, 2.5V y 1.2V; el regulador de 5V tiene la función de ajustar el voltaje suministrado por una fuente externa al operar fuera de SATEDU, para después distribuirlo a los demás reguladores.

Tanto el MCU como la memoria *Flash* utilizadas operarán a un nivel de voltaje de 3.3V, por lo que es necesario hacer uso de un regulador de 3.3V, el cual suministrará la potencia necesaria para el buen funcionamiento de estos dispositivos.

Los FPGAs de la familia Spartan 3E operan con tres niveles de voltaje principales VCC_{INT}, VCC_{AUX} y VCC_{O} donde:

Las terminales VCC_{INT} están designadas para la alimentación de la lógica del núcleo interno del FPGA, estas terminales utilizan un voltaje nominal de 1.2V. Además, son la fuente de alimentación de todas las funciones lógicas tales como CLBs, Bloques de RAM y multiplexores.

Las terminales VCC_{AUX} están designadas para la alimentación auxiliar, estas terminales utilizan un voltaje nominal de 2.5V. Suministran energía a los DCMs, controladores diferenciales, pines dedicados para configuración y para la interfaz JTAG.

Las terminales VCC_{O} están designadas para la alimentación de los IOBs de cada banco, estas terminales son energizadas con un voltaje de 3.3V, para ofrecer compatibilidad con los dispositivos externos con los cuales interactúa.

Acondicionamiento

La etapa de acondicionamiento se divide en dos secciones, las cuales resuelven los problemas de acoplamiento de señales entre estación terrena y el MCU, y entre el MCU y el FPGA. La primera sección estará formada por un transceptor, el cual es utilizado para acondicionar el nivel de voltaje de las señales de comunicación entre el MCU y la PC de la estación terrena. La segunda etapa estará formada por una serie de resistencias, las cuales se encargarán de acondicionar la impedancia de las señales de configuración que se conectan entre el MCU y el FPGA.

8.3.3 Bloque de adquisición, procesamiento y generación de comandos (BAPGC)

El BAPGC se encuentra integrado en una arquitectura de cómputo embebida, la cual está dividida en tres módulos principales. En dicha arquitectura algunos de sus componentes se encuentran descritos como bloques en *hardware* (núcleos IP), mientras que la lógica de control, controladores de los núcleos, comunicación con dispositivos externos así como la

comunicación entre los mismos módulos y algunas operaciones aritméticas internas se encuentran desarrolladas en *software*.

Los núcleos IP fueron desarrollados utilizando la *suite ISE* en lenguaje VHDL y simulados previamente dentro de bancos de prueba, en tanto que las secciones de *software* fueron diseñadas utilizando el compilador en C integrado en el ambiente de desarrollo EDK, dentro de un procesador *Microblaze* de 32 bits y arquitectura RISC, donde se integra finalmente tanto el *hardware* como el *software* en un sistema embebido para la plataforma FPGA.

Los módulos en los cuales se encuentra dividido este bloque se describen a continuación:

Módulo de adquisición de datos

Este módulo se encarga de adquirir la información de cada uno de los sensores de navegación y/o de otros dispositivos externos. Todo el proceso de adquisición de datos se encuentra descrito en *hardware*, desarrollando en *software* la interfaz I2C para comunicación con la IMU y la comunicación con el módulo de procesamiento de datos.

Módulo de procesamiento de datos

Este módulo se encarga de realizar etapas de preprocesamiento y procesamiento de datos, ya sea para la obtención de la señal de compás magnético en esquemas de control de orientación en un eje, o bien, para la integración de bloques de algoritmos para la determinación de la orientación en esquemas de control en tres ejes. Gran parte del módulo se encuentra descrito en *hardware*, particularmente operaciones aritméticas matriciales de uso recurrente en varias partes de los algoritmos de control, lo cual implica un ahorro de recursos, utilizando el principio de reutilización de código en bloques. También hay parte del módulo que se resuelve en *software*, básicamente operaciones aritméticas que no requieran gran cantidad de recursos así como la lógica de control para la transferencia de datos con bloques adyacentes como sensores y generación de comandos.

Módulo de generación de comandos

Este módulo se encarga de generar las señales de control hacia los actuadores, con base en las señales resultantes del esquema de control generado en el módulo anterior. Se encuentra descrito en *hardware* con interfaces en *software* para su manejo y lógica de control por parte del microprocesador.

8.4 Diseño de la tarjeta electrónica

8.4.1 Análisis y descripción de los componentes utilizados

Antes de comenzar a realizar un esquema de conexiones fue necesario analizar los requerimientos presentes para esta tarjeta, y con base en ellos se seleccionaron los dispositivos a utilizar para el diseño de la tarjeta del subsistema de control de orientación. En los siguientes párrafos se listan los dispositivos utilizados, describiendo aquellos que tienen un papel más importan dentro de la tarjeta. Cabe señalar que se conservan algunos componentes usados en la tarjeta SMIN.

- FPGA XC3S1600E
- PIC18LF2520
- Memoria SST25VF064C
- Oscilador 50MHz
- MAX 233
- Reguladores de 5, 2.5, 3.3 y 1.2 Volts
- Diodos de protección
- LEDs
- Capacitores
- Resistencias

FPGA XC3S1600E

Se utilizó un dispositivo con empaquetado FG320 que tiene 320 terminales y opera con señales de voltaje de 3.3V, 2.5V, 1.8, 1.5 y 1.2V.

Posee ocho gestores de reloj digital DCMs, que operan en un intervalo de frecuencias que van desde 5MHz hasta 300MHz utilizando un oscilador externo (en el caso de este sistema opera con un oscilador de 50MHz) con divisores, multiplicadores y sintetizadores de frecuencia, ocho señales de reloj globales y ocho señales de reloj designadas para cada mitad del dispositivo. Integra una memoria RAM rápida de 648Kbits y 231Kbits de memoria RAM distribuida. Se compone por puertos designados para la configuración por comunicación JTAG IEEE 1149.1/1532, *Master Serial, Slave serial, Master parallel Up y Down y SPI Serial Flash*. Otros recursos internos del dispositivo se muestran en la tabla de la figura 8.2.

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Figura 8.2. Recursos internos del FPGA XC3S600E

PIC18LF2520

Con base en los resultados obtenidos en la serie de pruebas realizadas para la tarjeta SMIN se determinó utilizar este dispositivo ya que al pertenecer a la misma familia del PIC18F4520 cuentan con recursos similares. Puede ser utilizado para operar a bajo nivel de potencia (para este caso se trabajará con un voltaje de 3.3V) además de tener dimensiones más reducidas al contener menos terminales de entrada/salida.

Memoria SST25VF064C

Se utilizó esta tecnología en memorias porque al utilizar un FPGA con mayores recursos que el utilizado en la tarjeta SMIN es necesario usar un dispositivo de almacenamiento proporcionalmente mayor.

Es una memoria *Flash* con capacidad de 64Mb, con *buffer* de datos de 256 bytes, la cual puede trabajar con frecuencias de reloj de hasta 80MHz. Diseñado para aplicaciones de baja potencia como comunicaciones y adquisición de datos.

Utiliza el protocolo de comunicación SPI para establecer comunicación con otros dispositivos, este protocolo utiliza un bus que consta de 4 señales, SCK, MOSI, MISO y CS, que en este dispositivo son designadas como SCK, SI, SO y CE respectivamente. Siendo SCK la señal de reloj de entrada que sincroniza la transferencia de datos desde y hacia la memoria; SI, la señal de entrada para recepción de instrucciones y datos; SO, la señal de salida para transmisión de datos y CE, la señal de entrada para habilitar el dispositivo y así realizar la escritura y lectura del mismo.

Para llevar a cabo la programación de esta memoria, después de que el dispositivo es habilitado se transmite el comando de control, con el cual se le indica el estado en que se debe colocar el dispositivo, posteriormente se transmite la dirección desde la cual se desea comenzar a programar y finalmente se comienza a trasmitir el *bitstream* que será almacenado. Este flujo de comunicación para el almacenamiento en memoria se muestra en la figura 8.3.

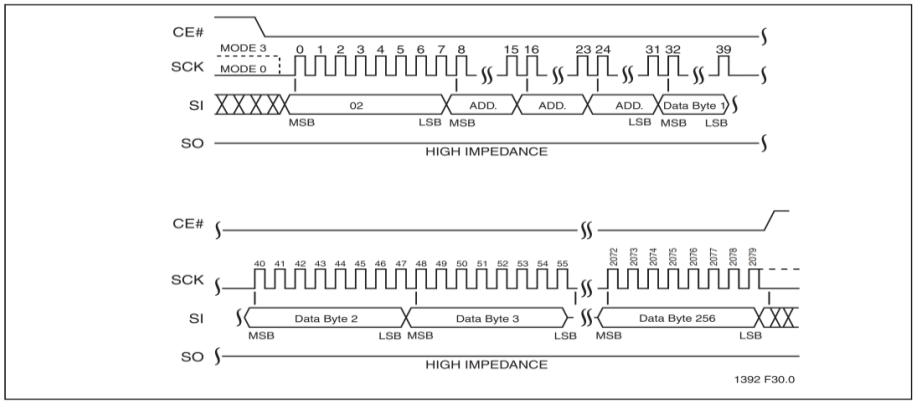


Figura 8.3. Programación de la memoria.

Para llevar a cabo la programación de esta memoria, después de que el dispositivo es habilitado se transmite el comando de control, con el cual se le indica el estado en el que se debe colocar el dispositivo, posteriormente se transmite la dirección desde la cual se desea comenzar a programar y finalmente se comienza a trasmitir el *bitstream* que será almacenado. En las figuras 8.4, se muestra el flujo de comunicación para lectura en memoria.

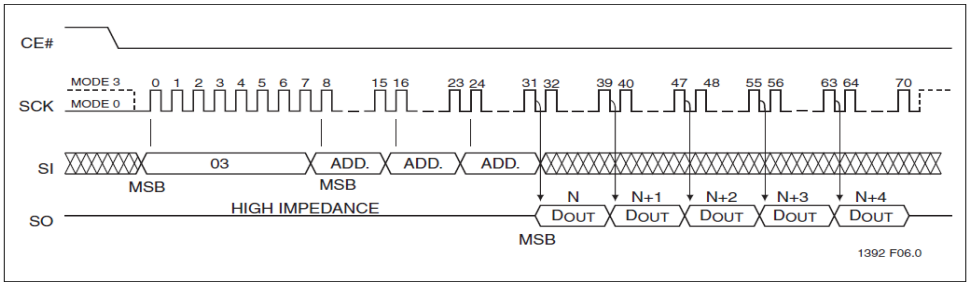


Figura 8.4. Proceso de lectura de la memoria.

El principal reto de esta propuesta es lograr el ensamble correcto del FPGA ya que este se encuentra fabricado en un encapsulado FG320, donde sus terminales se encuentran distribuidas en la superficie de la cara inferior del encapsulado, esto complica por un lado el diseño al interconectar sus terminales con dispositivos externos y por el otro el proceso de soldado del FPGA en la tarjeta.

8.4.2 Avance de diseño de la tarjeta de circuito impreso

Para el diseño de la tarjeta de circuito impreso, además de tomar como fuentes de referencia a algunos sistemas comerciales como el mismo Spartan 3E-SK, notas de aplicación y hojas de especificaciones de productos de Xilinx. Se aprovechó la experiencia obtenida del desarrollo y pruebas de validación de la tarjeta SMIN. Con base en las ideas presentadas y atendiendo a los requerimientos de nuestra aplicación final, se integró el sistema que se describe en los diagramas esquemáticos de las figuras 8.5 y 8.6, el cual lleva un avance del 75% de su diseño total.

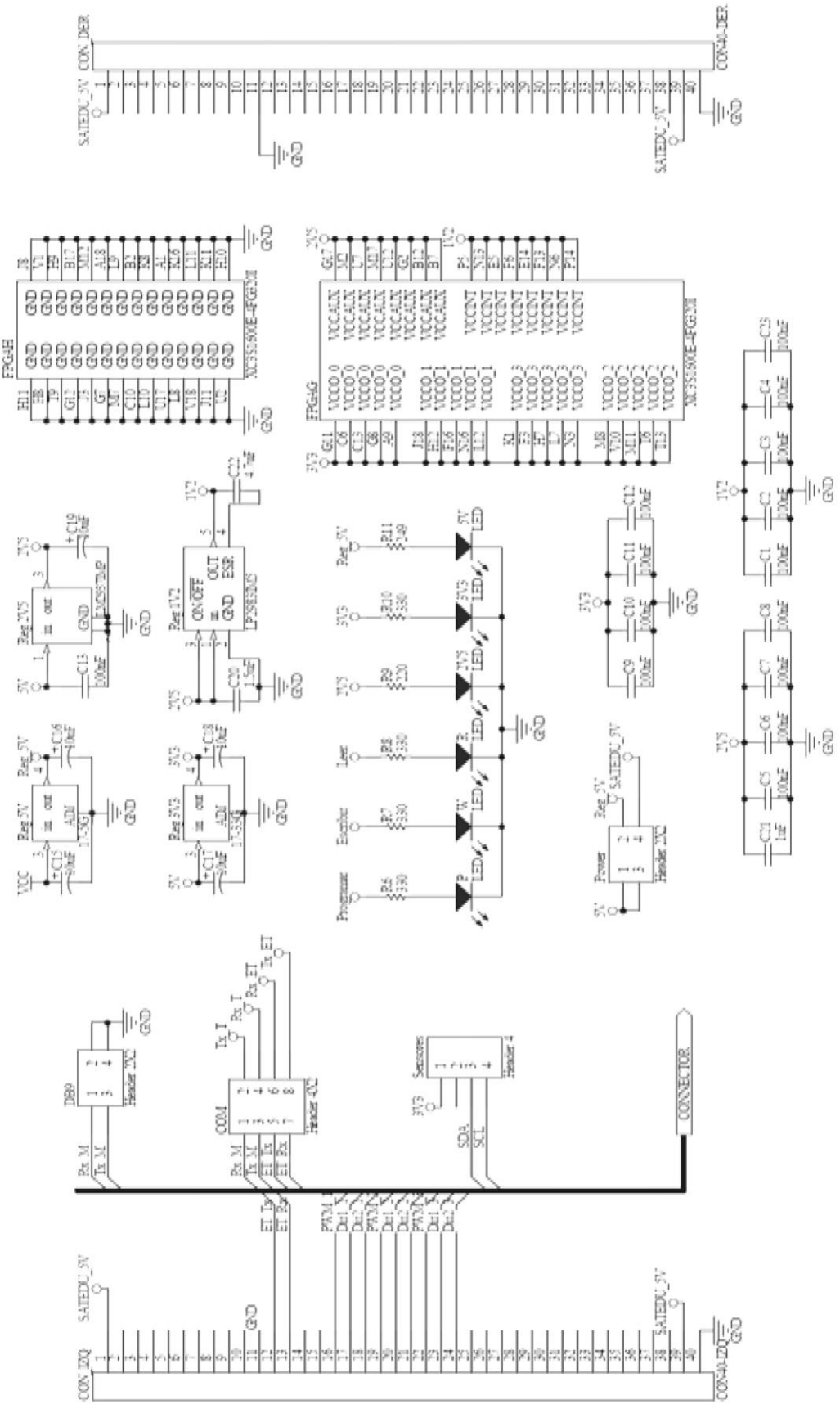


Figura 8.5. Sección de potencia, depuración y sensores, del subsistema de control de orientación.

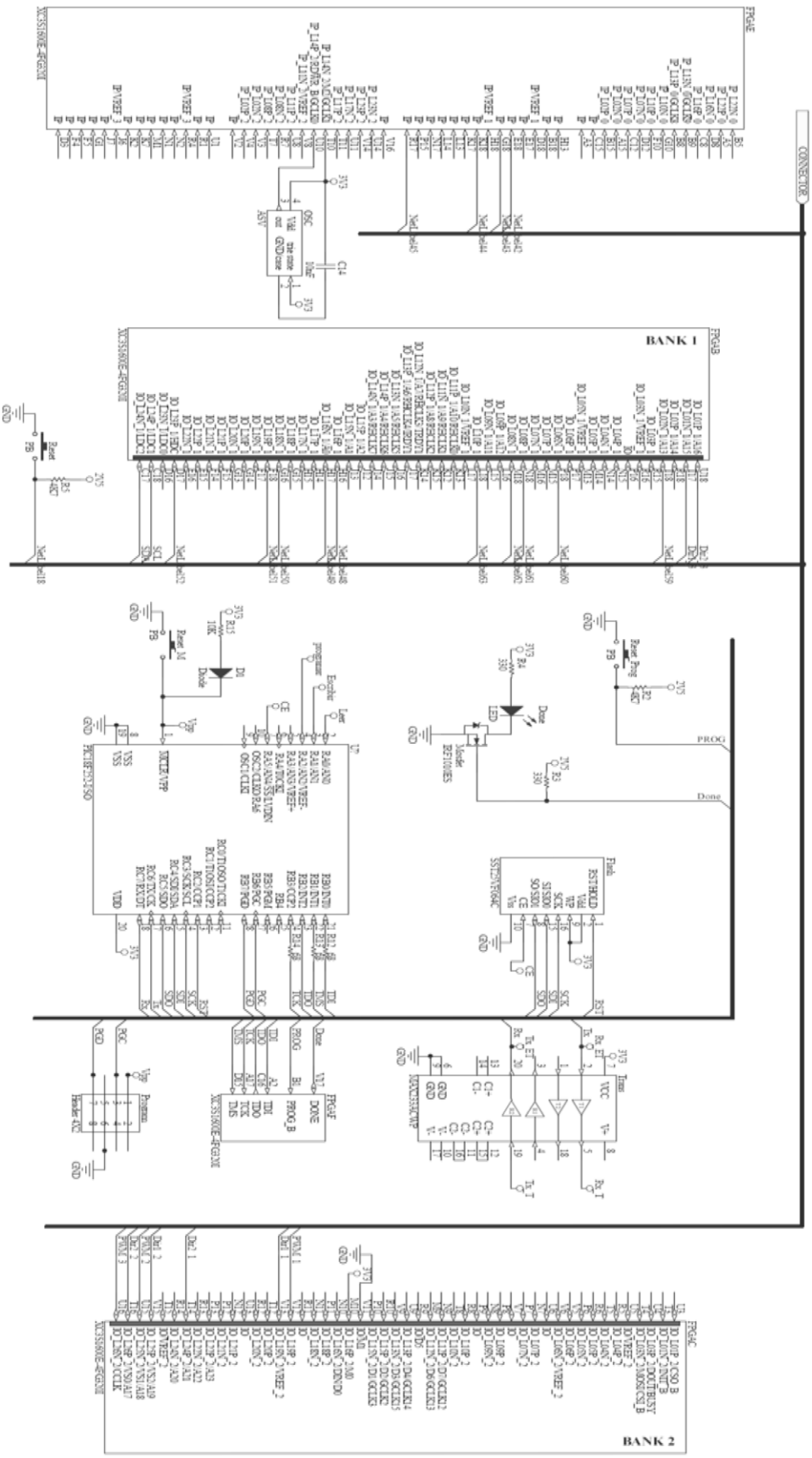


Figura 8.6. Sección de reconfiguración, comunicaciones, adquisición y generación de comandos del FPGA, del subsistema de control de orientación.

En las figuras 8.7a y 8.7b se muestra el diseño del circuito impreso correspondiente al diagrama esquemático mostrado anteriormente. En la figura 8.7a se muestra la cara superior de la tarjeta, y en la figura 8.7b se muestra la cara inferior

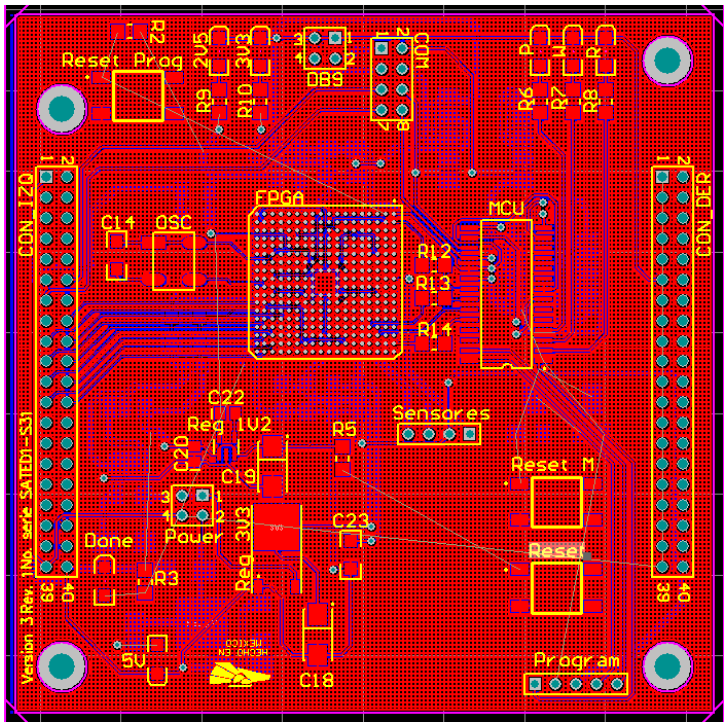


Figura 8.7a. PCB de la tarjeta del subsistema de control de orientación.

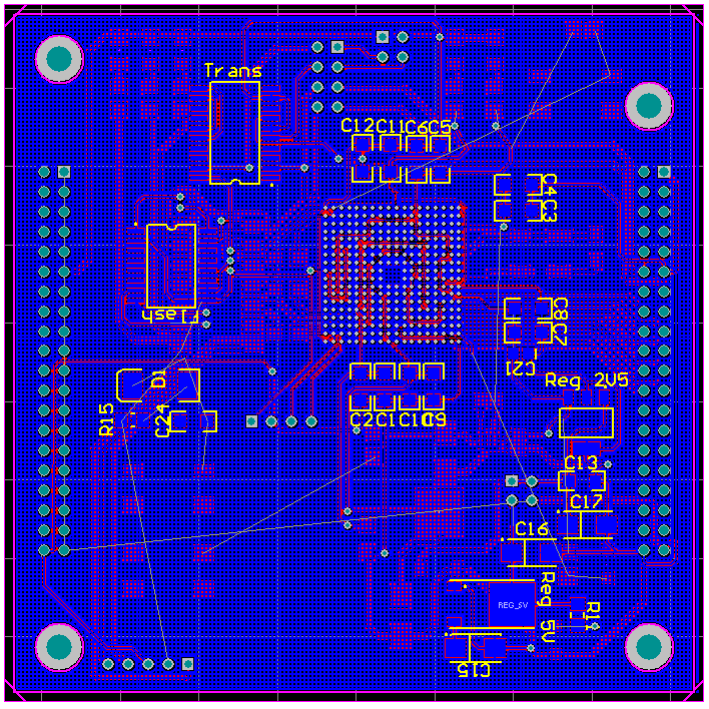


Figura 8.7b. PCB de la tarjeta del subsistema de control de orientación.

En la figura 8.8 se muestra la imagen en 3D de la tarjeta. En la parte izquierda de la figura se muestra la cara superior de la tarjeta y de lado derecho se muestra la cara inferior.

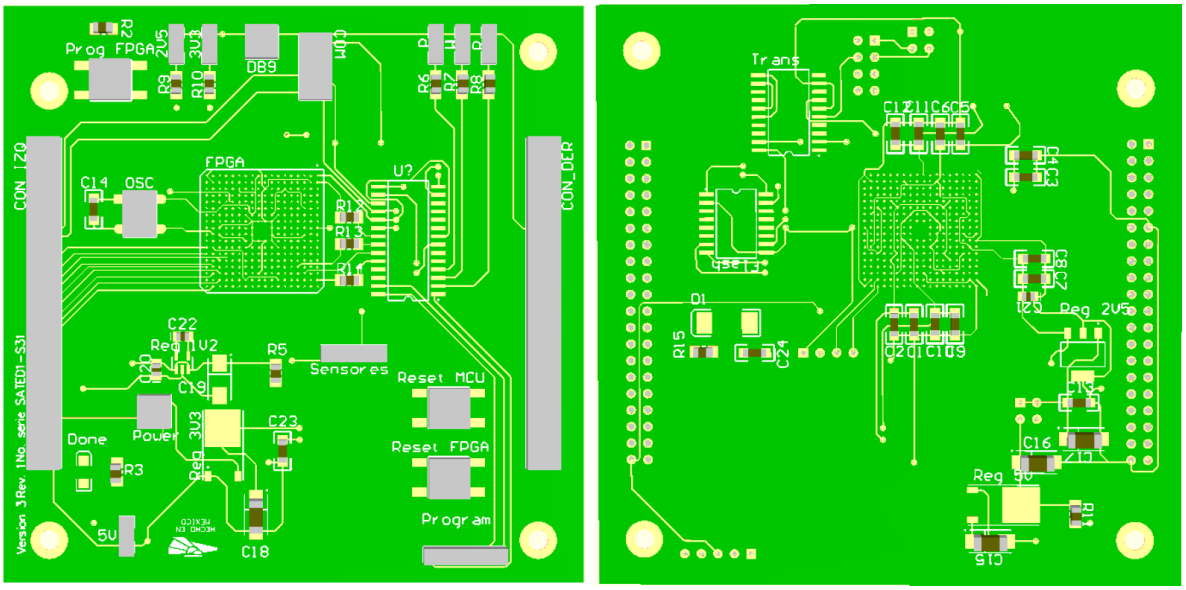


Figura 8.8. Esquema en 3D de la tarjeta en proceso de desarrollo.

Por el gran tiempo requerido para realizar este tema de tesis fue necesario congelar el escrito de ésta, en un 75% del desarrollo de la nueva tarjeta del subsistema de control de orientación que será parte de SATEDU y con ello acelerar el proceso de titulación.

CAPÍTULO 9

Conclusiones y recomendaciones

En este capítulo se presentan las conclusiones obtenidas en el desarrollo de esta tesis. En la última parte de este capítulo se mencionan algunas recomendaciones que pueden ser tomadas en cuenta, con objeto de mejorar el desempeño del sistema desarrollado en esta tesis.

9.1 Conclusiones

Del trabajo realizado en esta tesis se concluye lo siguiente:

Se logró desarrollar e integrar un sistema de evaluación basado en una plataforma FPGA de diseño propio, totalmente ensamblado y validado en el II-UNAM, el cual sentó las bases para la integración del nuevo sistema de determinación y control triaxial de apuntamiento del satélite educativo SATEDU y que permitirá la validación experimental de algoritmos numéricos de control de apuntamiento implantados como sistemas embebidos. Todo el *hardware* será montado sobre un simulador de vuelo, con que se cuenta en el II-UNAM y está basado en un cojinete de aire, que recrea con gran aproximación uno de los factores presentes en el ambiente espacial: la fricción cero.

Una de las principales innovaciones del sistema desarrollado, consistió en la integración de un módulo de reconfiguración remota, que incluye un microcontrolador de propósito general que, por un lado, gestiona el almacenamiento en una memoria a bordo de un archivo de reconfiguración transmitido desde una PC que hace las veces de estación terrena y, por otro lado, controla el proceso de reconfiguración total del FPGA. Lo que permitirá que una vez que el sistema sea montado sobre el simulador de vuelo satelital, las arquitecturas computacionales que describen a los diferentes algoritmos de determinación y control de actitud del SATEDU, puedan ser actualizadas de forma inalámbrica, tantas veces como sea necesario.

El módulo de reconfiguración remota, además de dar un alto valor agregado al sistema en términos del modo en que permite la descarga del archivo de reconfiguración al FPGA, permitirá utilizar al sistema como una plataforma de validación de algoritmos de control descritos por medio de arquitecturas de cómputo a la medida y que satisface los requerimientos de la aplicación.

El sistema desarrollado, cuenta con los recursos suficientes para funcionar tanto dentro como fuera de la plataforma de SATEDU, lo cual le confiere un alto grado de versatilidad, permitiendo no sólo fungir como herramienta para el entendimiento y base para la experimentación de diversos algoritmos de control de apuntamiento satelital, sino también como herramienta didáctica fuera del satélite para el aprendizaje de dispositivos lógicos programables como los FPGAs.

Los resultados exitosos que se obtuvieron en esta tesis han reforzado la idea original de emplear tecnología FPGA como base computacional del subsistema de determinación y control de orientación para los proyectos SATEDU y HumSat-México. Este último, parte de una constelación en la que participan instituciones y centros de investigación de todo el mundo y que actualmente se encuentra en fase de diseño en el II-UNAM.

El desarrollo de este sistema permitirá implantar y evaluar en *hardware* reconfigurable por primera vez en México, el desempeño de algoritmos de estimación, filtrado y control de orientación en tres ejes, de satélites pequeños, en un esquema lo más cercano a la realidad, con las limitaciones inherentes de hacerlo en un laboratorio en tierra.

Respecto a su integración a SATEDU, el sistema le conferirá una gran versatilidad en términos de funcionar como una plataforma altamente flexible para la integración de

esquemas de control y validarse experimentalmente en tierra. Esta característica ubica a SATEDU como una de las únicas plataformas a nivel mundial que ofrece esta opción.

En este aspecto es muy importante subrayar que el desarrollo de este tipo de sistemas de características tan novedosas en la UNAM y en muchas otras instituciones nacionales, permitirá a través de estos sistemas la automatización de funciones tan complejas como la estabilización triaxial de satélites con capacidad de respuesta en tiempo real.

Luego de concluida esta tesis, queda la enorme satisfacción personal de observar un trabajo terminado, que si bien está sujeto aún a una serie de constantes mejoras y evoluciones, contribuye por un lado a robustecer líneas de trabajo e investigación tecnológica y teórica del laboratorio de desarrollo de sistemas aeroespaciales del II-UNAM y, por otro lado, permite la expresión proactiva y técnica de la juventud mexicana.

9.2 Recomendaciones

Como fue mencionado anteriormente, un desarrollo tecnológico muchas de las ocasiones está sujeto a mejoras y a evoluciones, y el sistema que se describió en esta tesis no es la excepción. El objetivo principal de este trabajo fue cumplido parcialmente, debido a limitantes técnicas en cuanto al dispositivo seleccionado y cuyo criterio de selección se basó inicialmente en la capacidad para el montaje del circuito integrado.

Sin embargo, una vez que ha sido terminada la primera versión de la tarjeta SMIN y con la experiencia teórica y práctica ganada, es posible mencionar que como trabajo futuro ya se ha considerado el cambio del FPGA hacia uno con mayores recursos lógicos, en el cual se pueda montar completamente el esquema de control de orientación sin riesgo de agotar los recursos internos del dispositivo (como se describió en el capítulo anterior).

Se ha recomendado también el uso de una memoria de mayor capacidad, la cual permite el almacenamiento de un archivo de reconfiguración de mayor tamaño. Además esta memoria almacenará de forma permanente un archivo de reconfiguración básico, el cual será cargado en el FPGA en caso de que se presente algún error o eventualidad al configurar al FPGA con un archivo de reconfiguración actualizado.

Para lograr una disminución significativa del tiempo de transmisión y almacenamiento del *bitstream* de reconfiguración para dispositivos FPGAs de una gama más alta, se recomienda trabajar a frecuencias de transmisión mayores a 57600 baudios, o cambiar de medio de comunicación.

Se recomienda también la inclusión de un mayor número de terminales disponibles de entrada/salida lógicas del FPGA, lo cual permitirá que puedan interconectarse un mayor número de dispositivos externos a la tarjeta, además de permitir físicamente realizar futuras actualizaciones en *hardware*.

Bibliografía

Libros:

1. Yale Patt, Jim Smith, Mateo Valero. Fine and coarse grain reconfigurable computing. #Edicion. Lugar de edición: Springer, 2007. ISBN: 978-1-4020-6504-0
2. Pong P. Chu. FPGA prototyping by VHDL examples. #Edición. New Jersey: Wiley-Interscience, 2008. ISBN 978-0-470-18531-5
3. Scott Hauck, André DeHon. Reconfigurable computing: the theory and practice of FPGA- based computation. # edición. Lugar: Elseiver, 2008, ISBN 978-0-12-370522-8
4. Ron Sass, Andrew G. Schmidt. Embedded Systems Design with Platform FPGAs: principles and practices. Elsevier, 2010. ISBN 978-0-12-374333-6
5. Romero Troncoso, René de Jesús. Electrónica digital y lógica programable. Facultad de Ingeniería Mecánica Eléctrica y Electrónica, Universidad de Guanajuato. Guanajuato, México. Primera edición, 2007.
6. Gockhale, Maya. Reconfigurable Computing Acceleration Computation with Field-Programmable Gate Arrays. Springer, 2005. ISBN-13 978-0-387-26106-5.
7. Sass, Ron. Embedded System with Platforms FPGAs: Principles and Practices. Morgan Kaufmann, 2010. ISBN 978-0-12-374333-6.
8. Noergaard, Tammy. Embedded System Architecture A comprehensive Guide for Engineers and Programmers. Elsevier. Embedded Technology Series. Elsevier, 2005. ISBN 0-7506-7792-9
9. Mesquita, Daniel, et.al. Remote and Partial Reconfiguration of FPGAs: tools and trends. Paper, IEEE, 2003. ISBN: 0-7695-1926-1/03
10. González, Juan, et.al. Tarjeta entrenadora para FPGA, basada en hardware abierto. Paper. Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2009.
11. Pacheco, E., Mendieta, F.J., Vicente-V. E. Satellite and Space Communications Research in Mexico: Contributions to a National Program. AIAA SPACE 2009 Conference and Exposition, 2009. Pasadena, California, E.U.

Manuales, Notas de aplicación y hojas de especificaciones:

12. Kuramoto, Randal. Xilinx In-System Programming Using an Embedded Microcontroller. March 6, 2009. XAPP058.
13. Goldbatt Kim. The low-Cost, Efficient Serial Configuration Spartan FPGAs. November 13, 1998. XAPP098.
14. Ng Mark, Peattie Mike. Using a Microprocessor to configure Xilinx FPGAs via Slave Serial or SelectMAP Mode. November 13, 2002. XAPP502.
15. Bridgford, B, Cammon,J. SVF and XSVF File formats for Xilinx Devices. XAPP503. August 17, 2009. XAPP503.
16. Khu, A. et. Al. Using Xilinx XCF02S/XCF04S JTAG Proms for Data Storage Applications. January 11, 2008. XAPP544
17. DS312: Spartan-3E FPGA Family: Data Sheet. August 26, 2009. Xilinx Inc.
18. DS332: Spartan 3 Generation Configuration User Guide. October 26, 2009. Xilinx Inc.
19. JTAG Programmer Guide. 1991-2000. Xilinx Inc.
20. Data sheet PIC18F2420/2520/4420/4520.
21. Data Sheet LM2937
22. Data Sheet LP2983
23. Datasheet 24AA1025/LC1025/FC1025

Paginas de Internet:

24. http://www.genso.org/.
25. http://www.humsat.org/.
26. http://www.iafastro.com/docs/2010/iac/nanosat/13_Aguado.pdf.
27. http://park.itc.u-tokyo.ac.jp/nsat/NS1/files/10th.AM/Presentation_Esau-Vicente.pdf.

Apéndice A

Software en Visual C++

Software Intel_Hex

```
// * * * * * * * * * * * * * * * * ****************************************
//          Este codigo es el encargado de generar un archivo en formato
//              Intel HEX, de algún archivo existente en la PC
// * * * * * * * * * * * * * * * * ****************************************

#include <stdio.h>
#include <conio.h>
#include <string.h>

#define RECORD_SIZE 0x10/* Size of a record. */
#define BUFFER_SIZE 128

char *line, buffer[BUFFER_SIZE];
FILE *infile;
FILE *escribe;

extern char hex( int c );
extern void puthex( int val, int digits );

int main( void )
{
        int     c=1, address=0;
        int     sum, i;
        char    punta[20];
        char    M;
        i=0;

        escribe = fopen("NUEVO.hex","w");       /** First argument - Binary input file **/
        printf("Archivo: ");
        scanf("%s", &punta);
        infile = fopen(punta,"rb");

        if (!infile)
        {
                printf ( "Error en abrir el archivo %s \n",punta);
        }
        else
        {
                while (c != EOF)                /** Read the file character by character **/
                {
                        if(address % 0x10000 == 0)
                        {
                                int offset;
                                fprintf(escribe,":02000004");
```

```
                                offset = address >> 8;
                                offset &= 0x0000FFFF;
                                puthex(offset,4);
                                sum = offset >> 8;
                                sum += offset & 0xf0;
                                sum += 6;
                                puthex(0-sum,2);
                                fprintf(escribe,"\n");
                        }
                        sum = 0;
                        line = buffer;
                        for (i=0; i<RECORD_SIZE && (c=getc(infile)) != EOF; i++)
                        {
                                *line++ = hex(c>>4);
                                *line++ = hex(c);
                                sum += c;                       /* Checksum each character. */
                        }
                        if (i)
                        {
                                sum += address >> 8;                    /* Checksum high address byte.*/
                                sum += address & 0x0f;  /* Checksum low address byte.*/
                                sum += i;                               /* Checksum record byte count.*/
                                line = buffer;                          /* Now output the line! */
                                putchar(':');
                                fprintf(escribe,":");
                                puthex(i,2);                            /* Byte count. */
                                puthex(address,3);                      /* Do address and increment */
                                address += i;                           /* by bytes in record. */
                                puthex(0,2);                            /* Record type. */
                                for(i*=2;i;i--) /* Then the actual data. */
                                {
                                        fprintf(escribe,"%c",putchar(*line++));
                                }
                                puthex(0-sum,2); /* Checksum is 1 byte 2's comp.*/
                                printf("\n");
                                fprintf(escribe,"\n");
                        }
                }
                fprintf(escribe, ":00000001FF\n");/* End record. */
        }
        fclose(infile);
        fclose(escribe);

        printf("\n\nLISTO!!");
        getche();
        return 0;
}

/* Return ASCII hex character for binary value. */
char hex( int c )
{
        if((c &= 0x000f)<10)
                c += '0';
        else
                c += 'A'-10;
        return((char) c);
```

```
}

/* Put specified number of digits in ASCII hex. */
void puthex( int val, int digits )
{
        if (--digits)
                puthex(val>>8,digits);
                putchar(hex(val & 0x0f));
                fprintf(escribe,"%c",hex(val & 0x0f));
}
```

Apéndice B

Firmware del microcontrolador PIC

Firmware de recepción, transmisión y almacenamiento en memoria

```
// ********************************************************************************
//          Este firmware es el encargado de llevar a cabo las tareas de recepción, transmisión
//                          y almacenamiento en memoria, dentro del microcontrolador.
// ********************************************************************************

//***************************//
//       Programa principal          //

#include <18F4520.h>
#include "string.h"
#fuses H4, NOWDT, NOMCLR,DEBUG,NOLVP
#use delay(clock=40000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#include "memory.h"

#define ACK    0x06
#define NL     0x0A
#define CR           0x0D
#define XON    0x11
#define XOFF   0x13
#define NAK    0x15

#define SendAck()      printf("ACK\n");
#define SendNack()     printf("NACK\n");
#define StopSender()   printf("XOFF\n")
#define StartSender()  printf("XON\n")

#define OnLedProg()    output_high(LED_PROG);
#define OffLedProg()   output_low(LED_PROG);

#define OnLedBusy()    output_high(LED_BUSY);
#define OffLedBusy()   output_low(LED_BUSY);

/*   Server Message Pump   */

#define CHECK_COMUNICATION '='
#define IDLE_MODE         '&'
#define PROGRAM_MEMORY     '!'
#define READ_MEMORY        ''''
#define ERASE_MEMORY       '$'
#define VERIFY_MEMORY      '%'
#define CANCEL             '*'
#INT_RDA
```

```
void isr_rad(void)
{
   disable_interrupts(INT_RDA);
}

void main()
{
        int8    message;
        init_ext_eeprom();          /**Initialize EEPROM memory**/

        do
        {
                message = getc();/** Input Message**/


        /*  Task Menu  */
        switch(message)
        {
                case CHECK_COMUNICATION:
                        SendAck();
                        break;
                case PROGRAM_MEMORY:
                        ProgramMemory();
                        break;
                case READ_MEMORY:
                        ReadMemory();
                        break;
                case ERASE_MEMORY:
                        EraseMemory();
                        break;
                case IDLE_MODE:
                       enable_interrupts(int_rda);
                       SendAck();
                       sleep();
                       break;
                       default:
                        SendNack();
     }
        }while (TRUE);
}



//*************************************************************************//
//             "memory.h"   Librería que ejeccuta todas last areas de almacenamiento,          //
//                              lectua, ecritura  y borrado de memoria                          //

#define ERR_COM_0 0x10
#define ERR_COM_1 0x11
#define ERR_COM_2 0x12
#define ERR_COM_3 0x13
#define ERR_COM_4 0x14
#define ERR_COM_5 0x15
#define ERR_COM_6 0x16

#include "24lc512.h"
```

```
#include "hex32.h"

void ProgramMemory(void);
void ReadMemory(void);
void EraseMemory(void);

void err(int8 error)
{
	switch (error)
	{
		case ERR_FORMAT;
		   break;
	}
}

void ProgramMemory(void)
{
	Hex32 package;
	int1  _continue;
	int8  error, option;
	int32 Line=0;
	int32 offset=0;

	_continue = true;
	error = ERR_NONE;

	while(_continue == true)
	{
		if( !kbhit() )
		{
			/**    Recive package    **/
			error = RecivePackage(&package);

			//**     An error occurred in the reception    **/
			if( error != ERR_NONE )
			{
				printf("Ocurrio un error en la recepcion\n");
				SendNack();
				_continue = true;
			}
			else if( package.lineType == 4 )
			{
				Line = make16(atoi(&package.stringHex32[9]),
				atoi(&package.stringHex32[11]));
				offset = Line << 16;
				offset &= 0xFFF10000;

			}
			/**     Finished Programming              **/
			else if( package.lineType == 1 )
			{
				StartSender();
				SendAck();
				_continue = false;
		}
			/**     He received a package worth    **/
```

```
                        else if( package.lineType == 0 )
                        {
                                package.address |= offset;
                                error = WritePackage(&package);
                        }
                }
                else
                {
                /**     Cancel     **/
                        option = getc();
                        if(option == CANCEL)
                        {
                                SendNack();
                                _continue = false;
                }
                }
        }
}

void ReadMemory(void)
{
        Hex32 package;
        int1  _continue;
        int8  error, option;
        int32 nextAddress;

        _continue = true;
        error = ERR_NONE;
        nextAddress = 0x00;

        while( _continue )
        {
                if( !kbhit() )
                {
                        error = ReadPackage(&package, nextAddress);
                        /**     generate package    **/
                        if(nextAddress % 0x10000 == 0)
                        {
                                SendPackage4(nextAddress);
                        }
                        /**    If an error occurred    **/
                        if( error != ERR_NONE )
                        {
                                sendNack();
                                _continue = false;
                        }
                        else if( package.lineType == 1 )
                        {
                                error = SendPackage(&package);
                                sendAck();
                                _continue = false;
                        }
                        else
                        {
                                 error = SendPackage(&package);
                                 nextAddress = package.address + RECORD_SIZE;
```

```
                        }
                }

                /**     Cancel     **/
                else
                {
                        option = getc();
                        if(option == CANCEL)
                        {
                                SendNack();
                                _continue = false;
        }
                }
         }
}

void EraseMemory(void)
{
        int32 address;
        int8  _continue;
        int8  option;
        address = 0x00;
        _continue = true;

        while(_continue)
        {
                if(!kbhit())
                {
                write_ext_eeprom(address,0xFF);
                address++;
                        if(address % 0x20 == 0x00 || address == 0x00)
                        {
                                StopSender();
                        }

                        if(address > EEPROM_SIZE)
                        {
                                StartSender();
                                SendAck();
                                _continue = false;
                        }
                }
                /**      Cancel    **/
                else
                {
                        option = getc();
                        if(option == CANCEL)
                        {
                                StartSender();
                                SendNack();
                                _continue = false;
                        }
                }
        }
}
```

```
/*void VerifyMemory(void)
{
	Hex32 RecPack;
	Hex32 SavedPack;
	int1  _continue;
	int8  error;
	int8 index;

	_continue = true;
	error = ERR_NONE;

	while(_continue == true)
	{
		error = RecivePackage(&RecPack);
		if( error != ERR_NONE )
		{
			SendNack();
			_continue = true;
		}
		/**     Finiched programming     **/
		else if( RecPack.lineType == 1 )
		{
			StartSender();
			SendAck();
			_continue = false;
		 }
		/**      Paquage NULL     **/
		else
		{
			error = ReadPackage(&SavedPack, RecPack.address);
			if( error != ERR_NONE )
			{
			sendNack();
			_continue = false;
			}
			else
			{
				index = 0;

			do{
			if(RecPack.stringHex32[index] == SavedPack.stringHex32[index])
			{
				_continue = true;
			}
			else
			{
				_continue = false;
			SendNack();
			}
				while(RecPack.stringHex32[index++] != null && _continue == true);
			}
		}
	}
}
```

```
//*************************************************************************//
//   "hex32.h"  Librería que convierte los paquetes ASCII Hexadecimal a enteros y viceversa   //
//              al igual que obtiene la dirección y el dato a almacenar                       //

#define RECORD_SIZE  0x10
#define BUFFER_SIZE  0x41
#define LINE_DATA    0x00
#define LINE_LAST    0x01
#define LINE_EXTEN   0x04
#define ERR_NONE     0x00
#define ERR_INVALID  0x00
#define ERR_FORMAT   0x02
#define ERR_CHECKSUM 0x03

typedef struct Hex32
{
      int8  stringHex32[BUFFER_SIZE];
      int8  data[RECORD_SIZE];
      int8  checksum;
      int8  lineType;
      int8  count;
    int32 address;
    int8  error;
}Hex32;

void InitPackage(Hex32 *package);
int8 Hex2Package(Hex32 *package);
int8 Bin2Package(Hex32 *package,
                 int8  lineType=1,
                 int8 count=0,
                 int32 address=0);
int8 RecivePackage(Hex32 *package);
int8 SendPackage(Hex32 *package);
void SendPackage4(int32 address);
int8 atoi(char *s);

void InitPackage(Hex32 *package)
{
      package->stringHex32[0]  = ':';
      package->stringHex32[1]  = '0';
      package->stringHex32[2]  = '0';
      package->stringHex32[3]  = '0';
      package->stringHex32[4]  = '0';
      package->stringHex32[5]  = '0';
      package->stringHex32[6]  = '0';
      package->stringHex32[7]  = '0';
      package->stringHex32[8]  = '1';
      package->stringHex32[9]  = 'F';
      package->stringHex32[10] = 'F';
      package->stringHex32[11] = null;

      package->data[0]  = 0xFF;
      package->data[1]  = 0xFF;
      package->data[2]  = 0xFF;
      package->data[3]  = 0xFF;
      package->data[4]  = 0xFF;
      package->data[5]  = 0xFF;
```

```
      package->data[6]  = 0xFF;
      package->data[7]  = 0xFF;
      package->data[8]  = 0xFF;
      package->data[9]  = 0xFF;
      package->data[10] = 0xFF;
      package->data[11] = 0xFF;
      package->data[12] = 0xFF;
      package->data[13] = 0xFF;
      package->data[14] = 0xFF;
      package->data[15] = 0xFF;

      package->checksum = 0xFF;
      package->lineType = 0x01;
      package->count    = 0x00;
      package->address  = 0x0000;
      package->error    = ERR_NONE;
}

int8 RecivePackage(Hex32 *package)
{
    int8 error;
    int8 index;
    int1 _continue;

    error = ERR_NONE;
    _continue = true;
    index = 0x00;
    StartSender();
    do{
        package->stringHex32[index] = getc();
        if(package->stringHex32[index] == NL)
        {
              error = ERR_NONE;
              package->stringHex32[index] = null;
              _continue = false;
        else if(index >= BUFFER_SIZE)
        {
              error = ERR_COM_1;
              initPackage(package);
              _continue = false;
        }
        index++;
    } while (_continue);
    StopSender();

    if(error == ERR_NONE)
        error = Hex2Package(package);
    return error;
}

int8 SendPackage(Hex32 *package)
{
    int8 index;
    if(package->error == ERR_NONE)
    {
        index = 0;
        while(package->stringHex32[index] != null)
```

```
        {
            putc(package->stringHex32[index++]);
        }
        putc(NL);
    }

    return package->error;
}

int8 ReadPackage(Hex32 *package, int32 address)
{
    int8 error;
    int8 count;
    int1 last;
    last = true;
    error = ERR_NONE;

    if( address >= EEPROM_SIZE )
    {
        error = Bin2Package(package);
    }
    else
    {
        for(count=0; count < RECORD_SIZE && address < EEPROM_SIZE;
count++, address++)
        {
            package->data[count] = read_ext_eeprom(address);
            if(last == true && package->data[count] != 0xFF)
                last = false;
        }
        if(last == true)
             error = Bin2Package(package);
        else
            error = Bin2Package(package,0, count, address-count);
    }
    return error;
}

int8 WritePackage(Hex32 *package)
{
    int8 index;
    int32 address;

    address = package->address;
    if(package->error == ERR_NONE)
        for(index=0; index<package->count; index++)
        {
           write_ext_eeprom(address, package->data[index]);
           address++;
        }
    return package->error;
}

int8 Bin2Package(Hex32 *package,
                 int8 lineType = LINE_LAST,
                 int8 count    = 0x00,
                 int32 address = 0x0000)
```

```
{
    int8  buffer[5];
    int8  checksum;
    int8  error;
    int16 addressH;
    int16 addressL;
    int8  index;
    int8 _count;

    checksum = 0x00;
    index    = 0x00;
    error    = ERR_NONE;

    package->count    = count;
    package->address  = address;
    package->lineType = lineType;

    if(lineType == 1)
    {
        sprintf(package->stringHex32,":01000001FF\0");
        package->checksum = 0xFF;
    }
    else if(lineType == 0)
    {
        package->stringHex32[index++] = ':';
         sprintf(buffer,"%2X",count);
         package->stringHex32[index++] = buffer[0];
         package->stringHex32[index++] = buffer[1];

        sprintf(buffer,"%4LX",address);
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];
        package->stringHex32[index++] = buffer[2];
        package->stringHex32[index++] = buffer[3];

        sprintf(buffer,"%2X",lineType);
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];

        for(_count = 0; _count < count ; _count++)
      {
            sprintf(buffer,"%2X",package->data[_count]);
            package->stringHex32[index++] = buffer[0];
            package->stringHex32[index++] = buffer[1];

      }
        addressL = address & 0x00FF;
        addressH = address >> 6;
        addressH = addressH & 0x00FF;
        checksum = 0;
        checksum = count + lineType + addressH + address;
        for(_count = 0; _count < count; _count++)
        {
            checksum += package->data[_count];
        }
        checksum = 0xFF - checksum + 1;
        sprintf(buffer,"%2X",checksum);
```

```
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];
        package->stringHex32[index++] = null;
    }
    else
    {
        package->error = true;
    }

    return error;
}

int8 Hex2Package(Hex32 *package)
{
    int8 error;
    int8 index;
    int8 indexData;
    error = ERR_NON;
    if(package->stringHex32[0] == ':')
    {
        package->error = ERR_NONE;
        package->count = atoi(&package->stringHex32[1]);
        package->address = make16(atoi(&package
->stringHex32[3]),atoi(&package->stringHex32[5]));
        package->lineType = atoi(&package->stringHex32[7]);
        if (package->lineType == 0x01)
        {
            for(index = 0; index < RECORD_SIZE ; index++)
            {
                  package->data[index] = 0xFF;
            }
            package->checksum = 0xFF;
        }
        else
        {
            if(package->lineType == 0x00)
            {
                package->checksum = 0x00;
                indexData = package->count*2+8;
                for (index = 1; index< indexData ; index += 2)
                {
                    package->checksum += atoi(&package
                ->stringHex32[index]);
                }
                package->checksum = ~package->checksum + 1;

                if(package->checksum != atoi(&package
           ->stringHex32[index]))
                {
                    package->error = ERR_CHECKSUM;
                }
                else
                {
                    for (indexData=0, index=9; indexData
                  < package->count; index += 2, indexData++)
                    {
```

```
                        package->data[indexData] = atoi(&package
                    ->stringHex32[index]);
                   }
                }
            }
        }
    }
    else
    {
        package->error = ERR_FORMAT;
    }
    return package->error;
}

int8 atoi(char *s) {
   int8 result = 0;
   int8 i;

   for (i=0; i<2; i++,s++)  {
      if (*s >= 'A')
         result = 16*result + (*s) - 'A' + 10;
      else
         result = 16*result + (*s) - '0';
   }
   return(result);
}

void SendPackage4(int32 address)
{
      char buffer[15];
      int32 offset;
      int8 sum;
      offset = address >> 16;
      offset &= 0x0000FFFF;
      sum = offset>>8;
      sum += offset & 0xFF;
      sum += 6;
      sum = 0 - sum;
      sprintf(buffer,":02000004%4X%2X",offset,sum);
      printf("%s\n",buffer);
}
```

Apéndice C

Firmware del microcontrolador PIC

Firmware de reconfiguración

```
// * * * * * * * * * * * * * * * * ***********************************************
//          Este firmware es el encargado de llevar a cabo la reconfiguración del FPGA
// * * * * * * * * * * * * * * * * ***********************************************

//*************************//
//       Programa principal          //

typedef struct tagSXsvfInfo
{
   /* XSVF status information */
   unsigned char   ucComplete;
   unsigned char   ucCommand;
   long          lCommandCount;
   int           iErrorCode;
   /* TAP state/sequencing information */
   unsigned char   ucTapState;
   unsigned char   ucEndIR;
   unsigned char   ucEndDR;

   /* RUNTEST information */
   unsigned char   ucMaxRepeat;
   long          lRunTestTime;

   /* Shift Data Info and Buffers */
   long          lShiftLengthBits;
   short         sShiftLengthBytes;
   lenVal         lvTdi;
   lenVal         lvTdoExpected;
   lenVal         lvTdoCaptured;
   lenVal         lvTdoMask;

#ifdef  XSVF_SUPPORT_COMPRESSION
   /* XSDRINC Data Buffers */
   lenVal         lvAddressMask;
   lenVal         lvDataMask;
   lenVal         lvNextData;
#endif
} SXsvfInfo;
typedef int (*TXsvfDoCmdFuncPtr)( SXsvfInfo* );

/* encodings of xsvf instructions */
#define XCOMPLETE       0
#define XTDOMASK        1
```

```
#define XSIR            2
#define XSDR            3
#define XRUNTEST        4
/* Reserved             5 */
/* Reserved             6 */
#define XREPEAT         7
#define XSDRSIZE        8
#define XSDRTDO         9
#define XSETSDRMASKS    10
#define XSDRINC         11
#define XSDRB           12
#define XSDRC           13
#define XSDRE           14
#define XSDRTDOB        15
#define XSDRTDOC        16
#define XSDRTDOE        17
#define XSTATE          18
#define XENDIR          19
#define XENDDR          20
#define XSIR2           21
#define XCOMMENT        22
#define XWAIT           23
#define XLASTCMD        24

/** XSVF Command Parameter Values  **/

#define XSTATE_RESET    0
#define XSTATE_RUNTEST  1

#define XENDXR_RUNTEST  0
#define XENDXR_PAUSE    1

/* TAP states */
#define XTAPSTATE_RESET    0x00
#define XTAPSTATE_RUNTEST  0x01
#define XTAPSTATE_SELECTDR 0x02
#define XTAPSTATE_CAPTUREDR 0x03
#define XTAPSTATE_SHIFTDR  0x04
#define XTAPSTATE_EXIT1DR  0x05
#define XTAPSTATE_PAUSEDR  0x06
#define XTAPSTATE_EXIT2DR  0x07
#define XTAPSTATE_UPDATEDR 0x08
#define XTAPSTATE_IRSTATES 0x09
#define XTAPSTATE_SELECTIR 0x09
#define XTAPSTATE_CAPTUREIR 0x0A
#define XTAPSTATE_SHIFTIR  0x0B
#define XTAPSTATE_EXIT1IR  0x0C
#define XTAPSTATE_PAUSEIR  0x0D
#define XTAPSTATE_EXIT2IR  0x0E
#define XTAPSTATE_UPDATEIR 0x0F

/** XSVF Function Prototypes   **/

int xsvfDoIllegalCmd( SXsvfInfo* pXsvfInfo );
int xsvfDoXCOMPLETE( SXsvfInfo* pXsvfInfo );
int xsvfDoXTDOMASK( SXsvfInfo* pXsvfInfo );
```

```
int xsvfDoXSIR( SXsvfInfo* pXsvfInfo );
int xsvfDoXSIR2( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDR( SXsvfInfo* pXsvfInfo );
int xsvfDoXRUNTEST( SXsvfInfo* pXsvfInfo );
int xsvfDoXREPEAT( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDRSIZE( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDRTDO( SXsvfInfo* pXsvfInfo );
int xsvfDoXSETSDRMASKS( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDRINC( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDRBCE( SXsvfInfo* pXsvfInfo );
int xsvfDoXSDRTDOBCE( SXsvfInfo* pXsvfInfo );
int xsvfDoXSTATE( SXsvfInfo* pXsvfInfo );
int xsvfDoXENDXR( SXsvfInfo* pXsvfInfo );
int xsvfDoXCOMMENT( SXsvfInfo* pXsvfInfo );
int xsvfDoXWAIT( SXsvfInfo* pXsvfInfo );

#ifdef  DEBUG_MODE
   char* xsvf_pzCommandName[]  =
   {
      "XCOMPLETE",
      "XTDOMASK",
      "XSIR",
      "XSDR",
      "XRUNTEST",
      "Reserved5",
      "Reserved6",
      "XREPEAT",
      "XSDRSIZE",
      "XSDRTDO",
      "XSETSDRMASKS",
      "XSDRINC",
      "XSDRB",
      "XSDRC",
      "XSDRE",
      "XSDRTDOB",
      "XSDRTDOC",
      "XSDRTDOE",
      "XSTATE",
      "XENDIR",
      "XENDDR",
      "XSIR2",
      "XCOMMENT",
      "XWAIT"
   };

   char*   xsvf_pzErrorName[]  =
   {
      "No error",
      "ERROR:  Unknown",
      "ERROR:  TDO mismatch",
      "ERROR:  TDO mismatch and exceeded max retries",
      "ERROR:  Unsupported XSVF command",
      "ERROR:  Illegal state specification",
      "ERROR:  Data overflows allocated MAX_LEN buffer size"
   };
```

```
   char*   xsvf_pzTapState[] =
   {
      "RESET",       /* 0x00 */
      "RUNTEST/IDLE", /* 0x01 */
      "DRSELECT",     /* 0x02 */
      "DRCAPTURE",    /* 0x03 */
      "DRSHIFT",      /* 0x04 */
      "DREXIT1",      /* 0x05 */
      "DRPAUSE",      /* 0x06 */
      "DREXIT2",      /* 0x07 */
      "DRUPDATE",     /* 0x08 */
      "IRSELECT",     /* 0x09 */
      "IRCAPTURE",    /* 0x0A */
      "IRSHIFT",      /* 0x0B */
      "IREXIT1",      /* 0x0C */
      "IRPAUSE",      /* 0x0D */
      "IREXIT2",      /* 0x0E */
      "IRUPDATE"      /* 0x0F */
   };
#endif  /* DEBUG_MODE */

#ifdef DEBUG_MODE
   FILE* in;
   int xsvf_iDebugLevel;
#endif /* DEBUG_MODE */


/** Description:  Initialize the xsvfInfo data  **/
int xsvfInfoInit( SXsvfInfo* pXsvfInfo )
{
   pXsvfInfo->ucComplete       = 0;
   pXsvfInfo->ucCommand        = XCOMPLETE;
   pXsvfInfo->lCommandCount    = 0;
   pXsvfInfo->iErrorCode       = XSVF_ERROR_NONE;
   pXsvfInfo->ucMaxRepeat      = 0;
   pXsvfInfo->ucTapState       = XTAPSTATE_RESET;
   pXsvfInfo->ucEndIR          = XTAPSTATE_RUNTEST;
   pXsvfInfo->ucEndDR          = XTAPSTATE_RUNTEST;
   pXsvfInfo->lShiftLengthBits = 0L;
   pXsvfInfo->sShiftLengthBytes= 0;
   pXsvfInfo->lRunTestTime     = 0L;

   return( 0 );
}

/**  Description:  Cleanup the xsvfInfo data  **/
void xsvfInfoCleanup( SXsvfInfo* pXsvfInfo )
{
}

/** Calculate the number of bytes the given number of bits **/
short xsvfGetAsNumBytes( long lNumBits )
{
   return( (short)( ( lNumBits + 7L ) / 8L ) );
}
```

```
/**Apply TMS and transition TAP controller by applying one TCK **/
void xsvfTmsTransition( short sTms )
{
   setPort( TMS, sTms );
   setPort( TCK, 0 );
   setPort( TCK, 1 );
}

/** From the current TAP state, go to the named TAP  **/
int xsvfGotoTapState( unsigned char*   pucTapState,
                      unsigned char    ucTargetState )
{
   int i;
   int iErrorCode;

   iErrorCode  = XSVF_ERROR_NONE;
   if ( ucTargetState == XTAPSTATE_RESET )
   {
      xsvfTmsTransition( 1 );
      for ( i = 0; i < 5; ++i )
      {
         setPort( TCK, 0 );
         setPort( TCK, 1 );
      }
      *pucTapState    = XTAPSTATE_RESET;
   }
   else
   {
      if ( ucTargetState == *pucTapState )
      {
         if ( ucTargetState == XTAPSTATE_PAUSEDR )
         {
            xsvfTmsTransition( 1 );
            *pucTapState    = XTAPSTATE_EXIT2DR;
         }
         else if ( ucTargetState == XTAPSTATE_PAUSEIR )
         {
            xsvfTmsTransition( 1 );
            *pucTapState    = XTAPSTATE_EXIT2IR;
         }
      }
      while ( ucTargetState != *pucTapState )
      {
         switch ( *pucTapState )
         {
         case XTAPSTATE_RESET:
            xsvfTmsTransition( 0 );
            *pucTapState    = XTAPSTATE_RUNTEST;
            break;
         case XTAPSTATE_RUNTEST:
            xsvfTmsTransition( 1 );
            *pucTapState    = XTAPSTATE_SELECTDR;
            break;
         case XTAPSTATE_SELECTDR:
            if ( ucTargetState >= XTAPSTATE_IRSTATES )
            {
```

```
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_SELECTIR;
    }
    else
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_CAPTUREDR;
    }
    break;
  case XTAPSTATE_CAPTUREDR:
    if ( ucTargetState == XTAPSTATE_SHIFTDR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_SHIFTDR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_EXIT1DR;
    }
    break;
  case XTAPSTATE_SHIFTDR:
    xsvfTmsTransition( 1 );
    *pucTapState    = XTAPSTATE_EXIT1DR;
    break;
  case XTAPSTATE_EXIT1DR:
    if ( ucTargetState == XTAPSTATE_PAUSEDR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_PAUSEDR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_UPDATEDR;
    }
    break;
  case XTAPSTATE_PAUSEDR:
    xsvfTmsTransition( 1 );
    *pucTapState    = XTAPSTATE_EXIT2DR;
    break;
  case XTAPSTATE_EXIT2DR:
    if ( ucTargetState == XTAPSTATE_SHIFTDR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_SHIFTDR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_UPDATEDR;
    }
    break;
  case XTAPSTATE_UPDATEDR:
    if ( ucTargetState == XTAPSTATE_RUNTEST )
    {
```

```
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_RUNTEST;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_SELECTDR;
    }
    break;
  case XTAPSTATE_SELECTIR:
    xsvfTmsTransition( 0 );
    *pucTapState    = XTAPSTATE_CAPTUREIR;
    break;
  case XTAPSTATE_CAPTUREIR:
    if ( ucTargetState == XTAPSTATE_SHIFTIR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_SHIFTIR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_EXIT1IR;
    }
    break;
  case XTAPSTATE_SHIFTIR:
    xsvfTmsTransition( 1 );
    *pucTapState    = XTAPSTATE_EXIT1IR;
    break;
  case XTAPSTATE_EXIT1IR:
    if ( ucTargetState == XTAPSTATE_PAUSEIR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_PAUSEIR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_UPDATEIR;
    }
    break;
  case XTAPSTATE_PAUSEIR:
    xsvfTmsTransition( 1 );
    *pucTapState    = XTAPSTATE_EXIT2IR;
    break;
  case XTAPSTATE_EXIT2IR:
    if ( ucTargetState == XTAPSTATE_SHIFTIR )
    {
      xsvfTmsTransition( 0 );
      *pucTapState    = XTAPSTATE_SHIFTIR;
    }
    else
    {
      xsvfTmsTransition( 1 );
      *pucTapState    = XTAPSTATE_UPDATEIR;
    }
```

```
            break;
         case XTAPSTATE_UPDATEIR:
            if ( ucTargetState == XTAPSTATE_RUNTEST )
            {
               xsvfTmsTransition( 0 );
               *pucTapState    = XTAPSTATE_RUNTEST;
            }
            else
            {
               xsvfTmsTransition( 1 );
               *pucTapState    = XTAPSTATE_SELECTDR;
            }
            break;
         default:
            iErrorCode      = XSVF_ERROR_ILLEGALSTATE;
            *pucTapState    = ucTargetState;    /* Exit while loop */
            break;
         }
      }
   }

   return( iErrorCode );
}

/**  Assumes that starting TAP state is SHIFT-DR or SHIFT-IR  **/
void xsvfShiftOnly( long    lNumBits,
                    lenVal* plvTdi,
                    lenVal* plvTdoCaptured,
                    int     iExitShift )
{
   unsigned char*  pucTdi;
   unsigned char*  pucTdo;
   unsigned char   ucTdiByte;
   unsigned char   ucTdoByte;
   unsigned char   ucTdoBit;
   int             i;

   pucTdo  = 0;
   if ( plvTdoCaptured )
   {
      plvTdoCaptured->len = plvTdi->len;
      pucTdo              = plvTdoCaptured->val + plvTdi->len;
   }

   pucTdi  = plvTdi->val + plvTdi->len;
   while ( lNumBits )
   {
      /* Process on a byte-basis */
      ucTdiByte   = (*(--pucTdi));
      ucTdoByte   = 0;
      for ( i = 0; ( lNumBits && ( i < 8 ) ); ++i )
      {
         --lNumBits;
         if ( iExitShift && !lNumBits )
         {
            /* Exit Shift-DR state */
```

```
            setPort( TMS, 1 );
         }

         /* Set the new TDI value */
         setPort( TDI, (short)(ucTdiByte & 1) );
         ucTdiByte   >>= 1;

         /* Set TCK low */
         setPort( TCK, 0 );

         if ( pucTdo )
         {
            /* Save the TDO value */
            ucTdoBit    = readTDOBit();
            ucTdoByte   |= ( ucTdoBit << i );
         }

         /* Set TCK high */
         setPort( TCK, 1 );
      }

      /* Save the TDO byte value */
      if ( pucTdo )
      {
         (*(--pucTdo))   = ucTdoByte;
      }
   }
}

/**  Goes to the given starting TAP state  **/
int xsvfShift( unsigned char*   pucTapState,
               unsigned char    ucStartState,
               long             lNumBits,
               lenVal*          plvTdi,
               lenVal*          plvTdoCaptured,
               lenVal*          plvTdoExpected,
               lenVal*          plvTdoMask,
               unsigned char    ucEndState,
               long             lRunTestTime,
               unsigned char    ucMaxRepeat )
{
   int            iErrorCode;
   int            iMismatch;
   unsigned char  ucRepeat;
   int            iExitShift;

   iErrorCode  = XSVF_ERROR_NONE;
   iMismatch   = 0;
   ucRepeat    = 0;
   iExitShift  = ( ucStartState != ucEndState );

   if ( !lNumBits )
   {
      if ( lRunTestTime )
      {
         /* Wait for prespecified XRUNTEST time */
```

```
      xsvfGotoTapState( pucTapState, XTAPSTATE_RUNTEST );
      waitTime( lRunTestTime );
   }
}
else
{
   do
   {
      /* Goto Shift-DR or Shift-IR */
      xsvfGotoTapState( pucTapState, ucStartState );

      /* Shift TDI and capture TDO */
      xsvfShiftOnly( lNumBits, plvTdi, plvTdoCaptured, iExitShift );

      if ( plvTdoExpected )
      {
         /* Compare TDO data to expected TDO data */
         iMismatch   = !EqualLenVal( plvTdoExpected,
                                     plvTdoCaptured,
                                     plvTdoMask );
      }

      if ( iExitShift )
      {
         /* Update TAP state:  Shift->Exit */
         ++(*pucTapState);

         if ( iMismatch && lRunTestTime && ( ucRepeat < ucMaxRepeat ) )
         {
            /* Do exception handling retry - ShiftDR only */
            xsvfGotoTapState( pucTapState, XTAPSTATE_PAUSEDR );
            /* Shift 1 extra bit */
            xsvfGotoTapState( pucTapState, TAPSTATE_SHIFTDR );
            /* Increment RUNTEST time by an additional 25% */
            lRunTestTime    += ( lRunTestTime >> 2 );
         }
         else
         {
            /* Do normal exit from Shift-XR */
            xsvfGotoTapState( pucTapState, ucEndState );
         }

         if ( lRunTestTime )
         {
            /* Wait for prespecified XRUNTEST time */
            xsvfGotoTapState( pucTapState, XTAPSTATE_RUNTEST );
            waitTime( lRunTestTime );
         }
      }
   } while ( iMismatch && ( ucRepeat++ < ucMaxRepeat ) );
}

if ( iMismatch )
{
   if ( ucMaxRepeat && ( ucRepeat > ucMaxRepeat ) )
   {
```

```
            iErrorCode  = XSVF_ERROR_MAXRETRIES;
        }
        else
        {
            iErrorCode  = XSVF_ERROR_TDOMISMATCH;
        }
    }

    return( iErrorCode );
}

/** Get the XSDRTDO parameters and execute the XSDRTDO command **/
int xsvfBasicXSDRTDO( unsigned char*    pucTapState,
                      long              lShiftLengthBits,
                      short             sShiftLengthBytes,
                      lenVal*           plvTdi,
                      lenVal*           plvTdoCaptured,
                      lenVal*           plvTdoExpected,
                      lenVal*           plvTdoMask,
                      unsigned char     ucEndState,
                      long              lRunTestTime,
                      unsigned char     ucMaxRepeat )
{
    readVal( plvTdi, sShiftLengthBytes );
    if ( plvTdoExpected )
    {
        readVal( plvTdoExpected, sShiftLengthBytes );
    }
    return( xsvfShift( pucTapState, XTAPSTATE_SHIFTDR, lShiftLengthBits,
                       plvTdi, plvTdoCaptured, plvTdoExpected, plvTdoMask,
                       ucEndState, lRunTestTime, ucMaxRepeat ) );
}

/** pdate the data value with the next XSDRINC data and address**/
#ifdef  XSVF_SUPPORT_COMPRESSION
void xsvfDoSDRMasking( lenVal*  plvTdi,
                       lenVal*  plvNextData,
                       lenVal*  plvAddressMask,
                       lenVal*  plvDataMask )
{
    int             i;
    unsigned char   ucTdi;
    unsigned char   ucTdiMask;
    unsigned char   ucDataMask;
    unsigned char   ucNextData;
    unsigned char   ucNextMask;
    short           sNextData;

    addVal( plvTdi, plvTdi, plvAddressMask );

    ucNextData  = 0;
    ucNextMask  = 0;
    sNextData   = plvNextData->len;
    for ( i = plvDataMask->len - 1; i >= 0; --i )
    {
        ucDataMask  = plvDataMask->val[ i ];
```

```
      if ( ucDataMask )
      {
         /* Retrieve the corresponding TDI byte value */
         ucTdi       = plvTdi->val[ i ];
         ucTdiMask   = 1;
         while ( ucDataMask )
         {
            if ( ucDataMask & 1 )
            {
               if ( !ucNextMask )
               {
                  ucNextData  = plvNextData->val[ --sNextData ];
                  ucNextMask  = 1;
               }

               if ( ucNextData & ucNextMask )
               {
                  ucTdi   |= ucTdiMask;
               }
               else
               {
                  ucTdi   &= ( ~ucTdiMask );
               }

               /* Update the next data */
               ucNextMask  <<= 1;
            }
            ucTdiMask   <<= 1;
            ucDataMask  >>= 1;
         }

         /* Update the TDI value */
         plvTdi->val[ i ]    = ucTdi;
      }
   }
}
#endif  /* XSVF_SUPPORT_COMPRESSION */

/** Function place holder for illegal/unsupported commands  **/
int xsvfDoIllegalCmd( SXsvfInfo* pXsvfInfo )
{
   pXsvfInfo->iErrorCode   = XSVF_ERROR_ILLEGALCMD;
   return( pXsvfInfo->iErrorCode );
}

/** XCOMPLETE (no parameters) **/
int xsvfDoXCOMPLETE( SXsvfInfo* pXsvfInfo )
{
   pXsvfInfo->ucComplete   = 1;
   return( XSVF_ERROR_NONE );
}

/**  XTDOMASK <lenVal.TdoMask[XSDRSIZE]>  **/
int xsvfDoXTDOMASK( SXsvfInfo* pXsvfInfo )
{
   readVal( &(pXsvfInfo->lvTdoMask), pXsvfInfo->sShiftLengthBytes );
```

```
   return( XSVF_ERROR_NONE );
}

/**  XSIR <(byte)shiftlen> <lenVal.TDI[shiftlen]>  **/
int xsvfDoXSIR( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucShiftIrBits;
   short           sShiftIrBytes;
   int             iErrorCode;

   /* Get the shift length and store */
   readByte( &ucShiftIrBits );
   sShiftIrBytes   = xsvfGetAsNumBytes( ucShiftIrBits );

   if ( sShiftIrBytes > MAX_LEN )
   {
      iErrorCode  = XSVF_ERROR_DATAOVERFLOW;
   }
   else
   {
      /* Get and store instruction to shift in */
      readVal( &(pXsvfInfo->lvTdi), xsvfGetAsNumBytes( ucShiftIrBits ) );

      /* Shift the data */
      iErrorCode  = xsvfShift( &(pXsvfInfo->ucTapState),
                        XTAPSTATE_SHIFTIR,
                           ucShiftIrBits, &(pXsvfInfo                                    -
>lvTdi),
                           /*plvTdoCaptured*/0
            , /*plvTdoExpected*/0,
                           /*plvTdoMask*/0, pXsvfInfo                                    -
>ucEndIR,
                           pXsvfInfo->lRunTestTime
            , /*ucMaxRepeat*/0 );
   }

   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** XSIR <(2-byte)shiftlen> <lenVal.TDI[shiftlen]>  **/
int xsvfDoXSIR2( SXsvfInfo* pXsvfInfo )
{
   long            lShiftIrBits;
   short           sShiftIrBytes;
   int             iErrorCode;

   /* Get the shift length and store */
   readVal( &(pXsvfInfo->lvTdi), 2 );
   lShiftIrBits    = value( &(pXsvfInfo->lvTdi) );
   sShiftIrBytes   = xsvfGetAsNumBytes( lShiftIrBits );

   if ( sShiftIrBytes > MAX_LEN )
```

```
   {
      iErrorCode  = XSVF_ERROR_DATAOVERFLOW;
   }
   else
   {
      /* Get and store instruction to shift in */
      readVal( &(pXsvfInfo->lvTdi), xsvfGetAsNumBytes                                    ( lShiftIrBits
) );

            iErrorCode  = xsvfShift( &(pXsvfInfo->ucTapState)                                    ,
           XTAPSTATE_SHIFTIR,
            lShiftIrBits, &(pXsvfInfo->lvTdi),
            /*plvTdoCaptured*/0, /*plvTdoExpected*/0,
            /*plvTdoMask*/0, pXsvfInfo->ucEndIR,
            pXsvfInfo->lRunTestTime, /*ucMaxRepeat*/0 );
   }

   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** Shift the given TDI data into the JTAG scan chain  **/
int xsvfDoXSDR( SXsvfInfo* pXsvfInfo )
{
   int iErrorCode;
   readVal( &(pXsvfInfo->lvTdi), pXsvfInfo->sShiftLengthBytes );
   iErrorCode  = xsvfShift( &(pXsvfInfo->ucTapState)                                    ,
XTAPSTATE_SHIFTDR,
            pXsvfInfo->lShiftLengthBits, &(pXsvfInfo->lvTdi),
              &(pXsvfInfo->lvTdoCaptured),
              &(pXsvfInfo->lvTdoExpected),
              (pXsvfInfo->lvTdoMask), pXsvfInfo->ucEndDR,
             pXsvfInfo->lRunTestTime, pXsvfInfo->ucMaxRepeat );
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** Prespecify the XRUNTEST wait time for shift operations **/
int xsvfDoXRUNTEST( SXsvfInfo* pXsvfInfo )
{
   readVal( &(pXsvfInfo->lvTdi), 4 );
   pXsvfInfo->lRunTestTime = value( &(pXsvfInfo->lvTdi) );
   return( XSVF_ERROR_NONE );
}

/**  Prespecify the maximum number of XC9500/XL retries **/
int xsvfDoXREPEAT( SXsvfInfo* pXsvfInfo )
{
   readByte( &(pXsvfInfo->ucMaxRepeat) );
   return( XSVF_ERROR_NONE );
```

```
}

/** Prespecify the XRUNTEST wait time for shift operations **/
int xsvfDoXSDRSIZE( SXsvfInfo* pXsvfInfo )
{
   int iErrorCode;
   iErrorCode  = XSVF_ERROR_NONE;
   readVal( &(pXsvfInfo->lvTdi), 4 );
   pXsvfInfo->lShiftLengthBits = value( &(pXsvfInfo->lvTdi) );
   pXsvfInfo->sShiftLengthBytes= xsvfGetAsNumBytes( pXsvfInfo                ->lShiftLengthBits );
   if ( pXsvfInfo->sShiftLengthBytes > MAX_LEN )
   {
      iErrorCode  = XSVF_ERROR_DATAOVERFLOW;
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** Get the TDI and expected TDO values.  Then, shift  **/
int xsvfDoXSDRTDO( SXsvfInfo* pXsvfInfo )
{
   int iErrorCode;
   iErrorCode  = xsvfBasicXSDRTDO( &(pXsvfInfo->ucTapState),
                           pXsvfInfo->lShiftLengthBits,
                           pXsvfInfo->sShiftLengthBytes,
                           &(pXsvfInfo->lvTdi),
                           &(pXsvfInfo->lvTdoCaptured),
                           &(pXsvfInfo->lvTdoExpected),
                           &(pXsvfInfo->lvTdoMask),
                           pXsvfInfo->ucEndDR,
                           pXsvfInfo->lRunTestTime,
                           pXsvfInfo->ucMaxRepeat );
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** Get the prespecified address and data mask for the XSDRINC **/
#ifdef  XSVF_SUPPORT_COMPRESSION
int xsvfDoXSETSDRMASKS( SXsvfInfo* pXsvfInfo )
{
   /* read the addressMask */
   readVal( &(pXsvfInfo->lvAddressMask), pXsvfInfo                              ->sShiftLengthBytes
);
   /* read the dataMask    */
   readVal( &(pXsvfInfo->lvDataMask), pXsvfInfo                                 ->sShiftLengthBytes
);

   XSVFDBG_PRINTF( 4, "    Address Mask = " );
   XSVFDBG_PRINTLENVAL( 4, &(pXsvfInfo->lvAddressMask) );
   XSVFDBG_PRINTF( 4, "\n" );
   XSVFDBG_PRINTF( 4, "    Data Mask    = " );
   XSVFDBG_PRINTLENVAL( 4, &(pXsvfInfo->lvDataMask) );
   XSVFDBG_PRINTF( 4, "\n" );
```

```
   return( XSVF_ERROR_NONE );
}
#endif  /* XSVF_SUPPORT_COMPRESSION */

/** Get the XSDRINC parameters and execute the XSDRINC command **/
#ifdef  XSVF_SUPPORT_COMPRESSION
int xsvfDoXSDRINC( SXsvfInfo* pXsvfInfo )
{
   int            iErrorCode;
   int            iDataMaskLen;
   unsigned char  ucDataMask;
   unsigned char  ucNumTimes;
   unsigned char  i;

   readVal( &(pXsvfInfo->lvTdi), pXsvfInfo->sShiftLengthBytes );
   iErrorCode  = xsvfShift( &(pXsvfInfo->ucTapState)                                                    ,
XTAPSTATE_SHIFTDR,
             pXsvfInfo->lShiftLengthBits,
             &(pXsvfInfo->lvTdi), &(pXsvfInfo->lvTdoCaptured),
             &(pXsvfInfo->lvTdoExpected),
             &(pXsvfInfo->lvTdoMask), pXsvfInfo->ucEndDR,
            pXsvfInfo->lRunTestTime, pXsvfInfo->ucMaxRepeat );
   if ( !iErrorCode )
   {
      /* Calculate number of data mask bits */
      iDataMaskLen    = 0;
      for ( i = 0; i < pXsvfInfo->lvDataMask.len; ++i )
      {
         ucDataMask  = pXsvfInfo->lvDataMask.val[ i ];
         while ( ucDataMask )
         {
            iDataMaskLen    += ( ucDataMask & 1 );
            ucDataMask      >>= 1;
         }
      }

      readByte( &ucNumTimes );

      /* For numTimes, get data, fix TDI, and shift */
      for ( i = 0; !iErrorCode && ( i < ucNumTimes ); ++i )
      {
         readVal( &(pXsvfInfo->lvNextData),
                xsvfGetAsNumBytes( iDataMaskLen ) );
         xsvfDoSDRMasking( &(pXsvfInfo->lvTdi),
                     &(pXsvfInfo->lvNextData),
                     &(pXsvfInfo->lvAddressMask),
                     &(pXsvfInfo->lvDataMask) );
         iErrorCode  = xsvfShift( &(pXsvfInfo->ucTapState),
                           XTAPSTATE_SHIFTDR,
                           pXsvfInfo->lShiftLengthBits,
                           &(pXsvfInfo->lvTdi),
                           &(pXsvfInfo->lvTdoCaptured),
                           &(pXsvfInfo->lvTdoExpected),
                           &(pXsvfInfo->lvTdoMask),
                           pXsvfInfo->ucEndDR,
```

```
                            pXsvfInfo->lRunTestTime,
                            pXsvfInfo->ucMaxRepeat );
      }
   }
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}
#endif  /* XSVF_SUPPORT_COMPRESSION */

/** If not already in SHIFTDR, goto SHIFTDR  **/
int xsvfDoXSDRBCE( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucEndDR;
   int             iErrorCode;
   ucEndDR = (unsigned char)(( pXsvfInfo->ucCommand == XSDRE ) ?
                        pXsvfInfo->ucEndDR : XTAPSTATE_SHIFTDR);
   iErrorCode  = xsvfBasicXSDRTDO( &(pXsvfInfo->ucTapState),
                            pXsvfInfo->lShiftLengthBits,
                            pXsvfInfo->sShiftLengthBytes,
                            &(pXsvfInfo->lvTdi),
                            /*plvTdoCaptured*/0
,/*plvTdoExpected*/0,
                            /*plvTdoMask*/0, ucEndDR,
                            /*lRunTestTime*/0
            ,/*ucMaxRepeat*/0 );
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** If not already in SHIFTDR, goto SHIFTDR  **/
int xsvfDoXSDRTDOBCE( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucEndDR;
   int             iErrorCode;
   ucEndDR = (unsigned char)(( pXsvfInfo->ucCommand == XSDRTDOE ) ?
                        pXsvfInfo->ucEndDR : XTAPSTATE_SHIFTDR);
   iErrorCode  = xsvfBasicXSDRTDO( &(pXsvfInfo->ucTapState),
                            pXsvfInfo->lShiftLengthBits,
                            pXsvfInfo->sShiftLengthBytes,
                            &(pXsvfInfo->lvTdi),
                            &(pXsvfInfo->lvTdoCaptured),
                            &(pXsvfInfo->lvTdoExpected),
                            /*plvTdoMask*/0, ucEndDR,
                            /*lRunTestTime*/0, /*ucMaxRepeat*/0 );
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}
```

```
/** XSTATE: Get the state parameter and transition   **/
/**           the TAP to that state.              **/
int xsvfDoXSTATE( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucNextState;
   int             iErrorCode;
   readByte( &ucNextState );
   iErrorCode  = xsvfGotoTapState( &(pXsvfInfo->ucTapState), ucNextState );
   if ( iErrorCode != XSVF_ERROR_NONE )
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** XENDIR/XENDDR: Get the prespecified XENDIR or XENDDR  **/
int xsvfDoXENDXR( SXsvfInfo* pXsvfInfo )
{
   int             iErrorCode;
   unsigned char   ucEndState;

   iErrorCode  = XSVF_ERROR_NONE;
   readByte( &ucEndState );
   if ( ( ucEndState != XENDXR_RUNTEST ) && ( ucEndState != XENDXR_PAUSE ) )
   {
      iErrorCode  = XSVF_ERROR_ILLEGALSTATE;
   }
   else
   {

   if ( pXsvfInfo->ucCommand == XENDIR )
   {
         if ( ucEndState == XENDXR_RUNTEST )
         {
            pXsvfInfo->ucEndIR  = XTAPSTATE_RUNTEST;
         }
         else
         {
            pXsvfInfo->ucEndIR  = XTAPSTATE_PAUSEIR;
         }
      }
   else    /* XENDDR */
   {
         if ( ucEndState == XENDXR_RUNTEST )
         {
            pXsvfInfo->ucEndDR  = XTAPSTATE_RUNTEST;
         }
   else
   {
            pXsvfInfo->ucEndDR  = XTAPSTATE_PAUSEDR;
         }
      }
   }

   if ( iErrorCode != XSVF_ERROR_NONE )
```

```
   {
      pXsvfInfo->iErrorCode   = iErrorCode;
   }
   return( iErrorCode );
}

/** XCOMMENT: text string ending in \0> == text comment **/
int xsvfDoXCOMMENT( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucText;

   do
   {
      readByte( &ucText );
   } while ( ucText );

   pXsvfInfo->iErrorCode   = XSVF_ERROR_NONE;

   return( pXsvfInfo->iErrorCode );
}

/**    XWAIT: If not already in <wait_state>, then go      **/
/**  to <wait_state>. Wait in <wait_state> for <wait_time> **/
/**  microseconds.Finally, if not already in <end_state>   **/
/**             then goto <end_state>                                          **/
int xsvfDoXWAIT( SXsvfInfo* pXsvfInfo )
{
   unsigned char   ucWaitState;
   unsigned char   ucEndState;
   long            lWaitTime;

   /* <wait_state> */
   readVal( &(pXsvfInfo->lvTdi), 1 );
   ucWaitState = pXsvfInfo->lvTdi.val[0];

   /* <end_state> */
   readVal( &(pXsvfInfo->lvTdi), 1 );
   ucEndState = pXsvfInfo->lvTdi.val[0];

   /* <wait_time> */
   readVal( &(pXsvfInfo->lvTdi), 4 );
   lWaitTime = value( &(pXsvfInfo->lvTdi) );

   /* If not already in <wait_state>, go to <wait_state> */
   if ( pXsvfInfo->ucTapState != ucWaitState )
   {
      xsvfGotoTapState( &(pXsvfInfo->ucTapState), ucWaitState );
   }

   /* Wait for <wait_time> microseconds */
   waitTime( lWaitTime );

   if ( pXsvfInfo->ucTapState != ucEndState )
   {
      xsvfGotoTapState( &(pXsvfInfo->ucTapState), ucEndState );
   }
```

```
   return( XSVF_ERROR_NONE );
}


/** Initialize the xsvf player  **/
int xsvfInitialize( SXsvfInfo* pXsvfInfo )
{
   /* Initialize values */
   pXsvfInfo->iErrorCode   = xsvfInfoInit( pXsvfInfo );

   if ( !pXsvfInfo->iErrorCode )
   {
      /* Initialize the TAPs */
      pXsvfInfo->iErrorCode   = xsvfGotoTapState( &(pXsvfInfo->ucTapState), XTAPSTATE_RESET );
   }

   return( pXsvfInfo->iErrorCode );
}

/** Run the xsvf player for a single command and return **/
int xsvfRun( SXsvfInfo* pXsvfInfo )
{
   /* Process the XSVF commands */
   if ( (!pXsvfInfo->iErrorCode) && (!pXsvfInfo->ucComplete) )
   {
      /* read 1 byte for the instruction */
      readByte( &(pXsvfInfo->ucCommand) );
      ++(pXsvfInfo->lCommandCount);

      if ( pXsvfInfo->ucCommand < XLASTCMD )
      {
         /* Execute the command.  Func sets error code. */
        switch(pXsvfInfo->ucCommand )
         {
            case 0:
               xsvfDoXCOMPLETE( pXsvfInfo );
               break;
            case 1:
               xsvfDoXTDOMASK( pXsvfInfo );
               break;
            case 2:
               xsvfDoXSIR( pXsvfInfo );
               break;
            case 3:
               xsvfDoXSDR( pXsvfInfo );
               break;
            case 4:
               xsvfDoXRUNTEST( pXsvfInfo );
               break;
            case 5:
               xsvfDoIllegalCmd( pXsvfInfo );
               break;
            case 6:
               xsvfDoIllegalCmd( pXsvfInfo );
               break;
```

```
    case 7:
      xsvfDoXREPEAT( pXsvfInfo );
      break;
    case 8:
      xsvfDoXSDRSIZE( pXsvfInfo );
      break;
    case 9:
      xsvfDoXSDRTDO( pXsvfInfo );
      break;
#ifdef  XSVF_SUPPORT_COMPRESSION
    case 10:
      xsvfDoXSETSDRMASKS( pXsvfInfo );
      break;
    case 11:
      xsvfDoXSDRINC( pXsvfInfo );
      break;
#else
    case 10:
      xsvfDoIllegalCmd( pXsvfInfo );
      break;
    case 11:
      xsvfDoIllegalCmd( pXsvfInfo );
      break;
#endif  /* XSVF_SUPPORT_COMPRESSION */
    case 12:
      xsvfDoXSDRBCE( pXsvfInfo );
      break;
    case 13:
      xsvfDoXSDRBCE( pXsvfInfo );
      break;
    case 14:
      xsvfDoXSDRBCE( pXsvfInfo );
      break;
    case 15:
      xsvfDoXSDRTDOBCE( pXsvfInfo );
      break;
    case 16:
      xsvfDoXSDRTDOBCE( pXsvfInfo );
      break;
    case 17:
      xsvfDoXSDRTDOBCE( pXsvfInfo );
      break;
    case 18:
      xsvfDoXSTATE( pXsvfInfo );
      break;
    case 19:
      xsvfDoXENDXR( pXsvfInfo );
      break;
    case 20:
      xsvfDoXENDXR( pXsvfInfo );
      break;
    case 21:
      xsvfDoXSIR2( pXsvfInfo );
      break;
    case 22:
      xsvfDoXCOMMENT( pXsvfInfo );
```

```
                break;
                case 23:
                xsvfDoXWAIT( pXsvfInfo );
                break;
            }
        }
        else
        {
            xsvfDoIllegalCmd( pXsvfInfo );
        }
    }

    return( pXsvfInfo->iErrorCode );
}

/** cleanup remnants of the xsvf player **/
void xsvfCleanup( SXsvfInfo* pXsvfInfo )
{
    xsvfInfoCleanup( pXsvfInfo );
}

/** Execute: Process, interpret, and apply the XSVF command **/
int xsvfExecute()
{
     SXsvfInfo   xsvfInfo;

    xsvfInitialize( &xsvfInfo );

    while ( !xsvfInfo.iErrorCode && (!xsvfInfo.ucComplete) )
    {
        xsvfRun( &xsvfInfo );
    }

    xsvfCleanup( &xsvfInfo );
    return xsvfInfo.iErrorCode;

}

/** ProgramFPGA: Embedded users should call xsvfExecute() **/
int ProgramFPGA( void)
{
    int iErrorCode;
    iErrorCode = XSVF_ERROR_NONE ;

    setPort( TMS, 1 );

    /* Execute the XSVF in the file */
    iErrorCode  = xsvfExecute();

    return iErrorCode ;
}
```

Se omite código de las librerías “porth”, “LenVal.h” y “micro.h”, además los códigos fuente de “port.c” y “LenVal.c”.