UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE DISPOSITIVO DE CONTROL Y SUPERVISIÓN ADMINISTRABLE EN FORMA REMOTA POR ETHERNET

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA

FELIPE EDUARDO CONCHA AVELLO

PROFESOR GUÍA:
MAURICIO BAHAMONDE BARROS

MIEMBROS DE LA COMISIÓN:
HELMUTH EDGAR WILKE THIEMER WILCKENS
VICTOR GRIMBLATT HINSZPETER

SANTIAGO DE CHILE
ABRIL 2008

RESÚMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: FELIPE CONCHA AVELLO
FECHA: ABRIL 2008
PROF. GUÍA: SR. MAURICIO BAHAMONDES

DESARROLLO DE DISPOSITIVO DE CONTROL Y SUPERVISIÓN ADMINISTRABLE EN FORMA REMOTA POR ETHERNET

El presente trabajo de memoria tuvo el objetivo de realizar el desarrollo de una interfaz capaz de realizar el control de un bus I2C mediante el acceso a este desde una red Ethernet. El estudio de las posibles soluciones se centró en los principales requerimientos del sistema, los cuales fueron, el costo de la solución y la escalabilidad de este sistema, siendo capaz de llegar con la misma arquitectura a una velocidad de enlace de 1Gbps.

Dicho estudio se realizó para una empresa de radiodifusión llamada Continental Lensa, la cual busca desarrollar una plataforma de conexión que logre ser incluida en sus equipos de radiodifusión como producto de valor agregado, con la finalidad de que la empresa obtenga una plataforma estándar para realizar el manejo de equipos remotamente a través de esta interfaz.

Para la realización de este desarrollo, se escogió entre toda una gama de soluciones posibles, el uso de dispositivos semiconductores FPGA, en los cuales es posible producir la escalabilidad necesaria ya que existen núcleos capaces de operar a 1Gbps.

El resultado de tal estudio, demostró que la mejor solución es aquella que consta de la utilización de un procesador embebido implementado en la lógica del dispositivo FPGA, el cual no produce costos extras por cada dispositivo nuevo realizado. En este dispositivo, además se integró un sistema operativo Open Source basado en Linux llamado uClinux sobre el cual se realizará cada uno de los programas.

El controlador del bus I2C necesario se realiza mediante un núcleo de libre distribución, obtenido desde Internet, el cual es controlado desde el sistema operativo, con lo cual se muestra que es posible añadir nuevos periféricos a este sistema sin aumentar los costos de la solución y además se logra el control deseado.

En cuanto a la interfaz WEB, fué desarrollada mediante Applets, de modo de poder tener un elemento, el cual pueda ser configurado gráficamente en la empresa y que pueda ser visto desde cualquier computador, previa instalación de la maquina virtual de Java, en el PC que permita la visualización de la aplicación.

Posteriormente se analizó las tasas de transferencias que se pueden alcanzar con el sistema implementado y además se presenta una explicación acerca de todas las herramientas necesarias para la realización de este sistema.

Como conclusión del desarrollo realizado, se obtuvo un sistema, capaz de producir una reducción de costos, teniendo una gama de posibilidades para su extensión, modificando la lógica implementada de acuerdo a las necesidades y requerimientos de la empresa, en la cual se trabajó.

Índice

Índice de ilustraciones

1 Introducción

Continental Lensa S.A. es una empresa dedicada a desarrollar transmisores para radiodifusión, tanto analógicos como digitales. En el área de la radiodifusión digital, se están desarrollando nuevos transmisores en Chile, a los cuales se les busca dar el mayor valor agregado, de manera de ofrecer equipos cada vez más completos y con más funcionalidades.

Uno de los puntos que se esta abordando en la empresa, es la supervisión y el control de estos dispositivos, los cuales, hasta el momento son realizados directamente sobre el equipo mediante un panel de control y remotamente a través de controles que se comunican de manera paralela con una botonera, sin tener demasiada retroalimentación del sistema en el lado remoto.

Lo que se buscó desarrollar en esta memoria, fue un sistema que permita obtener información de los distintos componentes de un equipo de radiodifusión que se producirá por la empresa, y poder controlar este sistema, proveyendo una interfaz gráfica mas avanzada del sistema y con posibilidades de tener incluso acceso remotamente desde Internet, siendo el control posible de realizar desde otros países, sin necesidad de un computador dedicado a ello, sino que integrado en el mismo sistema.

El control del sistema es realizado actualmente mediante varios microprocesadores y circuitos integrados, los cuales interactúan realizando las distintas funciones del proceso de transmisión. El sistema que se realizó en esta memoria puede ser accedido remotamente mediante una red LAN, pudiendo controlar los distintos componentes del proceso de transmisión, los cuales al no estar aún construidos en la fecha de desarrollo de esta memoria, se supondrán como una interconexión de sistemas mediante un bus I2C, donde todo el control será realizado modificando las distintas memorias del sistema que contienen las variables de cada proceso o accediendo a los distintos dispositivos del sistema directamente.

Sobre el equipo que se desarrolló en este trabajo, la empresa planea en futuras etapas, basar distintos desarrollos tomando como base la conexión a red Ethernet que dispondrá esta plataforma, así como también, el procesador que esta disponga, de manera de producir una serie de equipos con la capacidad de ser accedidos mediante una pagina Web, o un servidor remoto, cada vez con mayor procesamiento en el sistema embebido que controla el sistema y con la posibilidad de agregar distintos servicios tales como transmisión de audio mediante UDP.

Uno de los requerimientos del sistema que se desarrolló, debido a las distintas aplicaciones que la empresa pretende realizar próximamente, es que este debe poder ser escalable, en particular permitiendo una conexión de LAN, la cual soporte el envío y recepción de datos con una red operando con el estándar de la IEEE 802.3ab (Gigabit Ethetnet). De este modo, poder obtener una conexión con otros dispositivos con los requerimientos de este enlace, pero sin necesariamente poder ocupar este ancho de banda completo. Lo anterior se busca para poder conectarlo con distintos dispositivos de Radio Digital que están integrándose en la empresa, como también, lograr transmitir audio por el enlace Ethernet.

En este desarrollo, se planteó realizar inmediatamente la conexión Gigabit con el sistema, pero debido a las condiciones de la placa de desarrollo utilizada, se trabajó finalmente con una interfaz de 100 Mbps. No obstante, siempre teniendo en cuenta que el sistema desarrollado cuenta con las facultades para poder manejar una capa física de mayor velocidad.

Según expuesto lo anterior, el tema de memoria se finalizó cuando se implementó una plataforma capaz de desplegar una pagina Web con información leída desde los distintos componentes del sistema, tanto de la placa misma, como también desde una memoria I2C. Además logró escribir información sobre este sistema, y mostrar información que este cambiando dinámicamente mientras se observa el sistema mediante la interfaz Web. Es importante como ya se mencionó, que esta pagina Web pueda producir cambios sobre el sistema presionando botones o ingresando datos en esta misma.

La solución que se plantea en esta memoria, es la utilización de un dispositivo FPGA (del inglés *Field Programmable Gate Array*), basado en memoria RAM de la marca Altera, de la familia Cyclone II. Este dispositivo, es programable en un idioma de descripción de hardware (HDL), el cual puede ser por ejemplo Verilog o VHDL, como también gráficamente mediante esquemáticos utilizando las herramientas adecuadas.

En este dispositivo se montó un sistema completo, incluyendo interfaces para el manejo de memoria, manejo de periféricos entre los que cuentan; una interfaz Ethernet de 100Mbps y un procesador embebido. Todas estas herramientas serán de la misma empresa que desarrolla la FPGA utilizada, tanto en sus herramientas de desarrollo de código para la FPGA como en el procesador embebido (llamado Nios II) que se utilizó.

Posteriormente, para el manejo de todo el stack TCP/IP, así como de los distintos procesos que necesitará el sistema, se montó un sistema operativo de Open Source sobre este procesador embebido, para lo cual se eligió utilizar uClinux, el cual es una versión derivada del kernel Linux 2.0, dirigido para microcontroladores sin Unidad de Manejo de Memoria (MMU según sus siglas en ingles). La versión utilizada posee soporte para el procesador embebido Nios II, entre muchos otros procesadores más, en este sistema se montara la pagina Web, se manejaran el periférico de I2C y se desarrollarán los programas para el manejo de esta pagina.

En esta memoria no se implementó la conexión Gigabit Ethernet, sino que se realizó una conexión de 100Mbps, con la cual se tiene una velocidad de acceso al medio mayor que otros dispositivos mas simples como un PIC y se deja la posibilidad de que, mediante el cambio del periférico implementado en la FPGA, un chip externo, y con los respectivos controladores de Linux, poder acceder a este requerimiento sin tener que cambiar la arquitectura que se esta utilizando, sólo cambiar el bloque de conexión Ethernet.

1.1 Antecedentes generales.

Se busca desarrollar una plataforma de control mediante acceso Ethernet, que permita controlar un sistema mediante una interfaz I2C, lo cual se realizará cambiando y leyendo registros de distintos componentes de un sistema que se esta implementando en la empresa. De esta manera, se podrá tener un equipo con la posibilidad de controlarlo sin necesidad de un computador externo, teniendo un sistema completo desarrollado esta empresa, siendo a su vez muy flexible.

Existen memorias anteriormente realizadas en el departamento, realizando el control de plataformas utilizando, tanto acceso telefónico, como acceso Ethernet de 10Mbps. En esta memoria, se realizará una acceso a tasas de transmisión mayores al medio siendo estas 100Mbps con la posibilidad de con el mismo procesador utilizado, llegar a una conexión de 1Gbps, cambiando algunas partes del hardware, pero continuando con la misma estructura del sistema.

Todo este desarrollo está buscado con el fin de poder dar una plataforma que logre ser ocupada para otros proyectos de la empresa, modificando la cantidad de periféricos empleados y la configuración de esta, sin necesariamente tener que cambiar de tecnología.

1.2 Antecedentes específicos

El estado de avance de la plataforma que se estuvo desarrollando, se conocían los esquemáticos realizados de la nueva plataforma encargada de realizar las transmisiones, el bus I2C que se utilizará está desarrollado en una plataforma independiente aún del diseño final. Para el desarrollo de esta memoria, se trabajará con un tipo de que se usa actualmente en la empresa con bus I2C, con la finalidad de lograr el control del sistema mediante escrituras y lecturas a esta.

Se estudiaron distintas soluciones de hardware para implementar la conexión a red y en base a esto, se tomó una decisión sobre la plataforma y la forma en que se desarrollará en el sistema, puesto que no estaban fijos en los requerimientos del diseño.

Se realizó un estudio entre las distintas soluciones de hardware que permitan los requerimientos anteriores. Seleccionado finalmente la tecnología de FPGA para el diseño. En este mismo hardware, una segunda etapa del desarrollo de la empresa, es el diseño de una interconexión PCI a utilizar con el sistema operativo la cual se utilizó, con el objetivo de controlar tarjetas con este bus, desde el mismo sistema desarrollado.

Entre las tecnologías trabajadas por la empresa, se encuentran desarrollos realizados en sistemas basados en PIC de la empresa Microchip, los cuales están siendo cambiados por nuevos desarrollos en DSP de la empresa Texas Instruments.

Finalmente se están desarrollando algunos equipos que utilizan FPGA, tanto de la marca Xilinx como Altera.

1.3 Objetivos generales

El objetivo general de esta memoria, consistió en la realización del diseño y del desarrollo de una solución con el fin de realizar una plataforma de interconexión entre la red Ethernet y un bus I2C, realizando una interfaz gráfica que permitiera poder observar y modificar remotamente un sistema que este controlado por este sistema.

Todo el trabajo fué realizado sobre una plataforma de desarrollo y en la empresa interesada en la utilización de este sistema, como parte del área de ingeniería de esta.

1.4 Objetivos específicos

1. Investigación de las distintas tecnologías existentes para realizar la conexión al medio.

2. Definición de una interfaz de conexión a la red Ethernet para el dispositivo, que cumpla con los requerimientos de precio y escalabilidad.

3. Diseño del sistema en la arquitectura seleccionada y su desarrollo en una plataforma de desarrollo.

4. Integración al sistema de una solución en software de manera de tener protocolos a lo menos, de TCP y UDP .

5. Montar un servidor WEB en el sistema y desarrollar una interfaz gráfica de control de este sistema.

6. Controlar una memoria I2C remotamente mediante la interfaz gráfica. Además poder ver y cambiar el estado de la plataforma de desarrollo.

2 ANTECEDENTES

En este capitulo se presentan los distintos temas, los cuales es necesario su manejo para el resto del trabajo. Estos temas, principalmente se relacionan con la arquitectura que se seleccionó para el diseño de esta solución. En ninguna instancia se piensa en esta como una descripción detallada de los conceptos aquí mencionados, lo cual escaparía del alcance de esta memoria, sino mas bien como una breve descripción de cada tecnología sólo para ubicar al lector con los temas que se tratan más adelante en esta memoria.

2.1 FPGA

Siglas que vienen del ingles *Field Programmable Gate Array*, este es un dispositivo semiconductor que contiene bloques cuya lógica de interconexión y funcionalidad puede programarse. La lógica programable puede reproducir funciones que van desde puertas logicas, sistemas combinacionales hasta complejos sistemas integrados en un chip (SOPC System On a Programable Chip).

Las FPGAs son en principio similares a los dispositivos ASICs (Circuito Integrado para Aplicaciones Especificas) los cuales son circuitos integrados hechos a la medida para un uso particular, con las diferencias de que una FPGA son más lentas, tiene un mayor consumo de potencia, pero tiene las ventajas de ser re-programables y con costos de desarrollo y adquisición menores en pequeñas cantidades de dispositivos, siendo especialmente útiles para el desarrollo de nuevas tecnologías, pero con la posibilidad de poder ser hasta utilizadas en productos finales por su precio que cada vez es menor.

Estos dispositivos surgen como una evolución de los dispositivos CPLDs (*Complex Programmable Logic Device*). Los dispositivos CPLD se forman en base a múltiples bloques lógicos, cada uno de ellos compuestos por una mezcla de compuertas AND y OR. Estos bloques se comunican entre sí utilizando una matriz programable de interconexiones, que es la que permite unir los pines de entrada/salida del bloque lógico a las entradas de otro bloque lógico o al mismo bloque en caso de ser necesario.

La matriz de interconexiones de los dispositivos CPLDs está diseñada generalmente usando dos opciones: mediante bloques (utilizando una matriz de filas y columnas la cual en cada intersección posee una celda programable) o mediante multiplexores conectados en cada entrada a los bloques lógicos. Con esta tecnología pueden llegar hasta decenas de miles de compuertas NAND equivalentes.

En la actualidad la mayoría de los CPLD son programables y borrables eléctricamente, además de ser no volátiles dado que no son lo suficientemente grandes como para justificar el tener que programar celdas SRAM en cada encendido del dispositivo, siendo esta la principal diferencia entre un CPLD y una FPGA, junto con la cantidad de compuertas equivalentes que cada uno dispone.

Las FPGA están basadas en el uso de un gran numero de pequeños bloques programables, compuestos típicamente por una tabla de búsqueda de 4 entradas y un flip-flop en cada uno de los elementos lógicos. En la figura a continuación se puede observar que el bloque lógico dispone en total de 5 entradas , 4 para la tabla de búsqueda mas un reloj y una única salida que puede o no pasar por el registro (flip flop final).

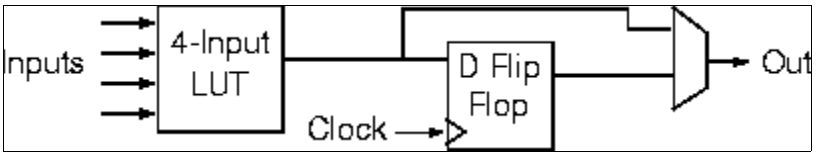


Ilustración 1: Bloque lógico de una FPGA

Por otro lado, los canales de interconexión están compuestos por un gran número de segmentos de cable en paralelo, permitiendo que cada entrada o salida, pueda conectarse a cualquier segmento del canal de interconexión mediante una caja de conmutación. Estas cajas de conmutación que están ubicadas a la salida de cada elemento lógico permite realizar todas las conexiones posibles entre los distintos elementos lógicos del sistema.

En la ilustración 2 se muestra el diagrama de conexión de los bloques en que se agrupan los elementos lógicos (LAB, del ingles Logic Array Block), las conexiones como se observa se realizan entre elementos lógicos adyacentes directamente mediante el bus local de interconexión, entre bloques lógicos adyacentes, o mediante interconexiones de columnas y filas, siendo el compilador del sistema (software) el encargado de realizar estas conexiones.

La tecnología que se utiliza para el diseño de estos dispositivos, utiliza elementos volátiles, siendo necesario en cada encendido configurar la FPGA con el diseño que requiere ejecutar, esta complejidad extra va con el echo de que el rango de compuertas NAND equivalentes que es posible alcanzar con estos dispositivos va desde cientos de miles hasta millones de estas, siendo bastante superior a lo alcanzado en un CPLD con tecnología no volátil.

La programación de una FPGA, se realiza con ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en una FPGA. Este diseño puede ser ingresado ya sea como un esquemático o haciendo uso de un lenguaje de programación, conocido como HDL del ingles *Hardware Description Language* (lenguaje de descripción de hardware) de donde los mas utilizados actualmente son VHDL y Verilog.

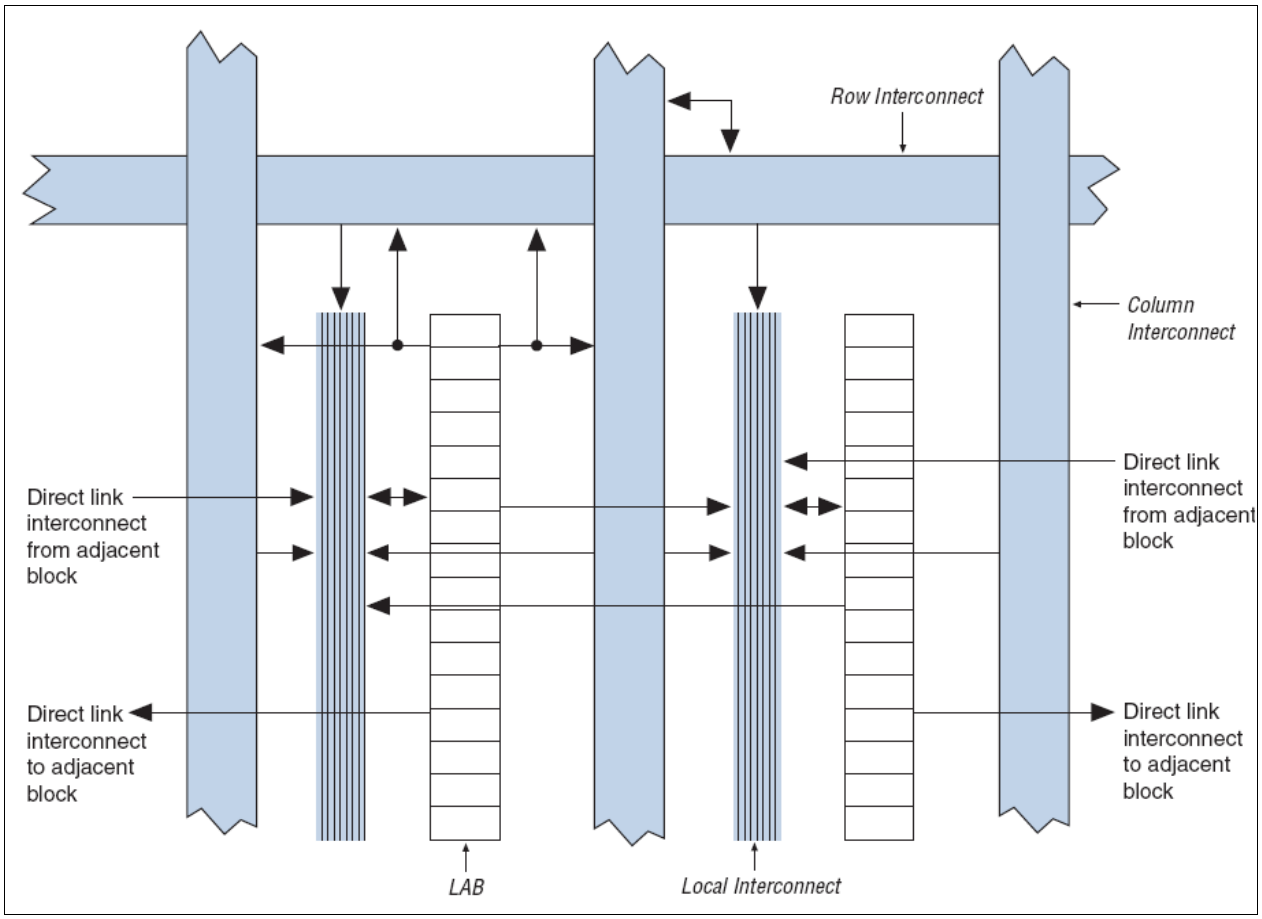


Ilustración 2: Estructura de Bloques lógicos en FPGA Cyclone II e interconexiones

El desarrollo de en esta memoria se basó principalmente en un dispositivo FPGA, entre los distintos proveedores de estos dispositivos, la selección fue por la empresa Altera y la familia de FPGA de bajo costo Cyclone II.

Con la gran cantidad de elementos lógicos programables que disponen estos circuitos integrados, es posible realizar casi cualquier función digital, dado que este sistema es volátil, permite que pueda ser configurado todas las veces que sea necesario con distintas funcionalidades, las cuales deben ser diseñadas, compiladas y cargadas desde un computador o cargadas en una memoria no volátil del sistema diseñada para configurar el dispositivo por la empresa fabricante de los mismos, esta última solución se adoptó para el diseño.

Para el desarrollo en esta tecnología, las herramientas provistas por Altera, disponen de la versión Web gratuita para Windows y de versiones pagadas para Windows y Linux. Con el kit de desarrollo, se incluía una licencia para el uso de este programa tanto en Windows como en Linux.

En el caso de esta memoria las herramientas y todo el desarrollo se realizó en Linux, debido a que el sistema operativo que se montara esta basado en este.

2.2 Procesador Embebido Nios II y SOPC (Sistema en un chip programable)

Para el desarrollo realizado en esta memoria, se optó por utilizar un procesador para la operación de las distintas rutinas que el sistema necesitará realizar, en particular la de ser servidor WEB.

En esta sección se describirán algunas de los características del procesador embebido utilizado:

Nios II, es la segunda generación de procesadores embebidos soft-cores (soft cores refiriéndose a que puede ser complementamente implementados usando síntesis lógica y por ende en una FPGA), procesador que es propietario de la empresa Altera.

Este procesador es de propósito general con arquitectura RISC y provee las siguiente funcionalidades:

- Set de instrucciones, bus de datos y espacio de direcciones de 32 bit
- 32 registros de propósito general
- 32 fuentes externas de interrupciones
- Instrucciones de punto flotante de precisión simple
- Acceso a periféricos integrados en el chip, e interfaces a memoria así como periféricos fuera del chip
- Entorno de desarrollo de software basado en las herramientas GNU C/C++ y el entorno de desarrollo Eclipse
- Arquitectura de set de instrucciones (ISA) compatible con todos los sistemas procesador Nios II
- Desempeño de hasta 250 DMIPS

En la ilustración 3 se muestra un diagrama de bloques del núcleo de este procesador, donde se ven las unidades funcionales que están disponibles para el usuario. No es la idea de esta memoria explicar todos los componentes de un procesador, información que se espera que sea conocida para el lector, pero si se busca destacar algunos bloques que usualmente no están disponibles en otras arquitecturas:

El bloque “Custom Instrucion Logic” permite al diseñador añadir lógica a la ALU (Unidad Aritmetica y Logica) implementada en la misma FPGA.

Los bloques de “Tightly Coupled Data Memory” permiten el acceso del procesador directo a memorias sin tener que utilizar los buses de instrucciones ni de datos sino utilizando buses dedicados para estas memorias, permitiendo accesos de baja latencia a estas memorias, siendo estas accedidas directamente en el espacio de direcciones como cualquier otra memoria. El bloque “JTAG Debug Module” permite realizar la depuración de código mediante una interfaz JTAG conectada a la FPGA mediante pines dedicados de esta.

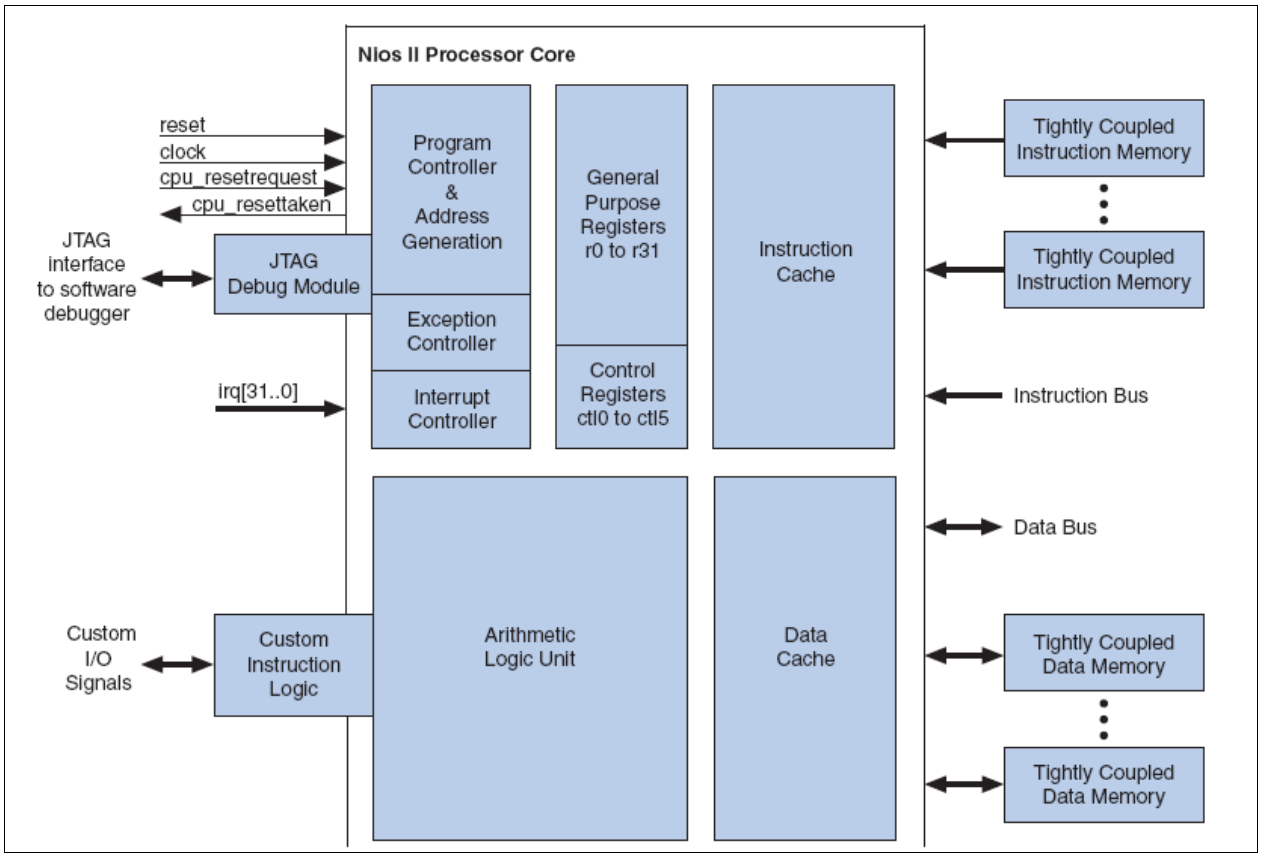


Ilustración 3: Diagrama de bloques del núcleo del procesador Nios II

Una de las ventajas de utilizar un procesador embebido, es la completa flexibilidad para seleccionar cualquier combinación de periféricos y controladores. Es posible crear incluso nuevos periféricos que se pueden conectar directamente al bus del procesador, lo que no es posible lograr con microprocesadores convencionales, donde se tiene el número de periféricos fijos para cada sistema.

La naturaleza del procesador, permite que puedan ser altamente configurable, permitiendo variar la cantidad de lógica de la FPGA que se quiera utilizar, cambiando así la velocidad del procesador, como también la utilización de recursos que este posee. Estos parámetros configurables, incluyen la utilización de memorias cache, uso o no uso de módulo de depuración JTAG, uso de multiplicaciones y divisiones por software o hardware. Incluso es posible añadir instrucciones extras directamente en la ALU del sistema, permitiendo implementar en hardware operaciones que pueden ser accedidas como instrucciones nativas del sistema.

Dado todas estas características configurables del procesador, existe una herramienta de desarrollo disponible del fabricante Altera para poder crear sistemas que incluyan procesadores, periféricos y memorias de forma que este proceso pueda ser más productivo. Esta herramienta es llamada SOPC Builder que permite definir y generar un sistema en un chip programable (SOPC) completo, de manera mas rápida que utilizando métodos de integración manual. Esta herramienta viene incluida en el software de Altera para la programación de sus FPGA, llamado Quartus II.

Esta herramienta automatiza la tarea de integrar componentes de hardware en un sistema más grande, esto se realiza mediante una interfaz gráfica desde donde el programa SOPC Builder genera la lógica de interconexión automáticamente, sin ser necesario tener que escribir un HDL de "nivel superior" que una todos estos componentes. Esta aplicación entrega los archivos HDL que definen todos los componentes del sistema, y un HDL de nivel superior que conecta todos los componentes. La salida de este programa pueden ser archivos de diseño tanto en Verilog como en VHDL, pudiendo seleccionarse el idioma de preferencia al diseñar el nuevo sistema.

Además de ser una herramienta capaz de generar los sistemas, este programa proporciona características para facilitar la escritura del software, como por ejemplo entregar archivos de información del sistema, con todos los datos necesarios para conocer rangos de memoria e interrupciones de los componentes creados, datos fundamentales para el uso de un procesador con estas características de configuración.

Todos los dispositivos realizados con esta interfaz, y en particular el procesador Nios II ocupan un bus desarrollado por Altera cono estándar abierto, llamado bus Avalon el cual entre sus principales características tiene:

- Lineas de direcciones, datos y control separadas.
- Ancho de bus de datos arbitrario hasta 1024 bits, este ancho no tiene que se necesariamente potencia de 2.
- Operación sincrónica con el reloj del bus.
- Tamaño de bus dinámico, este bus permite transferir datos entre periféricos con diferentes ancho de datos.

Un sistema típico basado en este bus Avalon multimaestro combina múltiples módulos funcionales, llamados "*Avalon-MM peripherals"*. La lógica en el chip que conecta los periféricos juntos es llamada "*system interconect fabric*" En la ilustración 4 se muestra un sistema de ejemplo utilizando múltiples periféricos utilizando este bus.

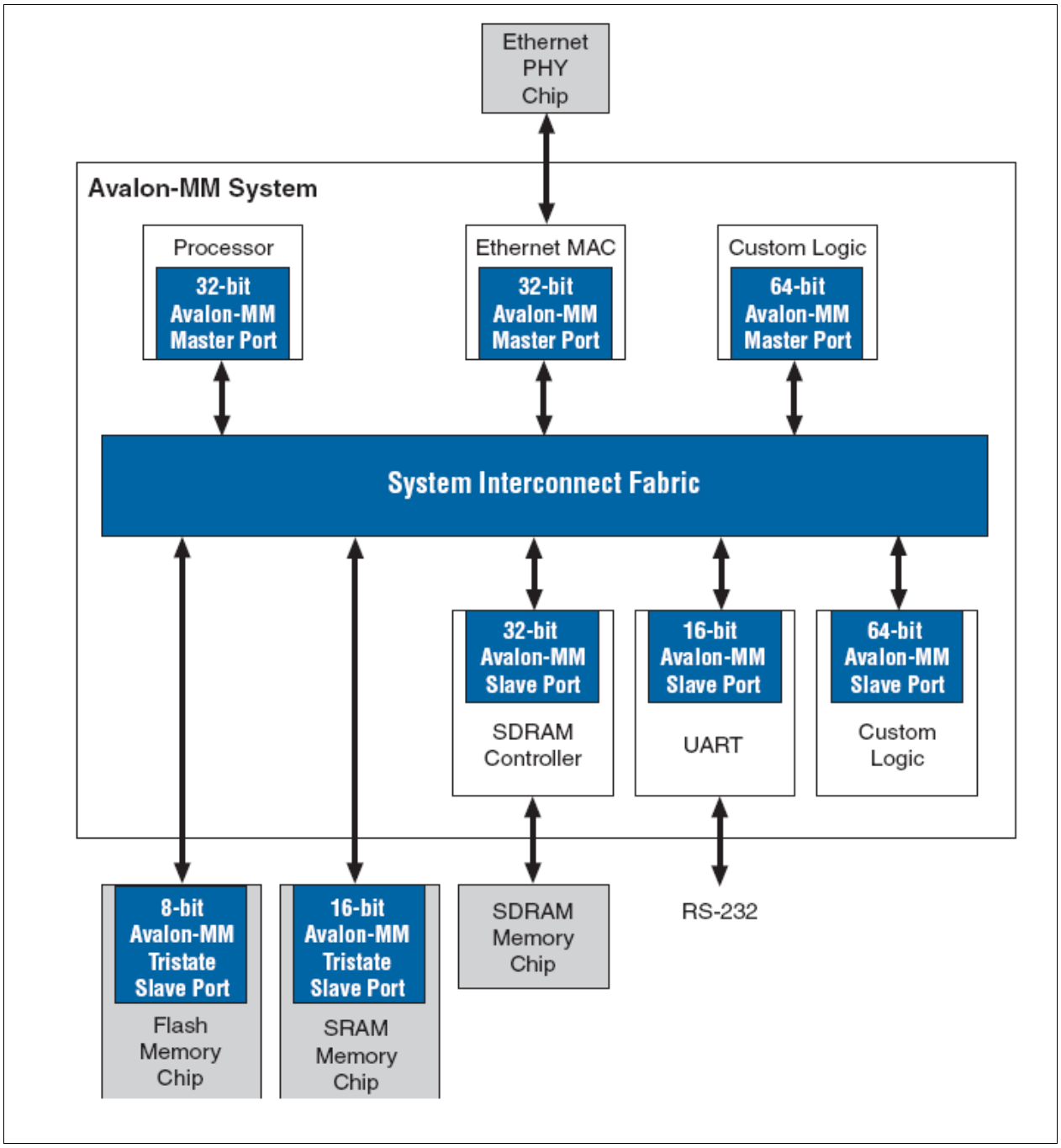


Ilustración 4: Ejemplo de interconexión utilizando el bus Avalon

Lo anterior, son los elementos básicos para poder diseñar un sistema utilizando las herramientas de Altera,.

Con estas herramientas se puede crear un sistema utilizando un procesador Nios II, configurarlo, instanciar sus periféricos, y construir finalmente un bloque para realizar las conexiones con los pines externos de la FPGA.

2.3 Bus I2C

I2C es un bus serial de dos cables bidireccional, que provee un método de intercambio de datos entre dispositivos. Esta diseñado para aplicaciones que requieren comunicación ocasional sobre distancias cortas entre varios dispositivos. El estándar I^2C es un bus multimaestro que incluye detección de colisiones, arbitraje que previene corrupción de los datos si dos o más maestros intentan acceder al bus simultáneamente.

El sistema I2C usa una linea de datos serial (SDA) y una linea de reloj serial (SCL) para las transferencias de datos. Todos los dispositivos conectados a estas dos señales deben tener salidas open-drain o open-collector. Ambas lineas necesitan la utilización de resistencias de pull-up externas para producir la función AND lógica de múltiples dispositivos, que es la que en definitiva permite que este bus sea multimaestro como se explicará a continuación.

Los datos son transferidos entre un maestro y un esclavo, por la linea SDA síncronamente a SCL, usando una base de transferencia byte a byte. Cada byte de datos es de 8 bits de largo. Existe un solo pulso de reloj en SCL por cada bit de datos con el bit mas significativo (MSB) transmitido al principio. Un bit de acknoledge sigue a cada byte transferido. Cada bit es muestreado durante el periodo alto de la señal SCL, por ende la linea SDA puede cambiarse solo en el periodo bajo de SCL y debe ser mantenido estable durante el periodo alto de SCL. Una transición en la linea SDA mientras la señal SCL esta en estado alto se interpreta como un comando.

Normalmente una comunicación estándar en el protocolo I2C consiste de cuatro partes:

1. Se genera una señal START
2. Se transmite la dirección del esclavo
3. Se transmite los datos
4. Se genera una señal de STOP

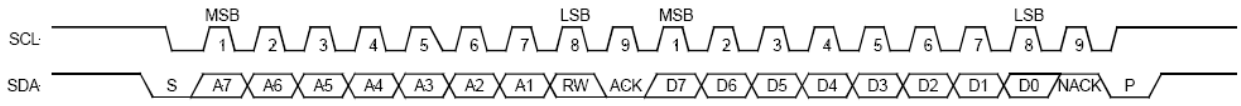


Ilustración 5: Ejemplo de una transición típica por un bus I^2C

El bus esta libre, es decir ningún maestro está utilizando el bus, cuando las dos lineas SCL y SDA están en estado alto, en este momento un maestro puede inicial una transferencia enviando la señal START.

La señal START, es usualmente referenciado por el bit S y es definida como una transición de alto a bajo de la linea SDA cuando SCL esta en alto. La señal START denota el inicio de una nueva transferencia de datos. Una señal de repetida de START, es una señal START sin generar previamente una señal de STOP, este método puede ser usado por un maestro para comunicarse con otro esclavo o con el mismo esclavo en otra dirección de transferencia (por ejemplo de escribir al dispositivo a leer desde este) sin tener que liberar el bus previamente.

El primer byte de datos transferido por el maestro inmediatamente después de la generación de una señal START es la dirección del esclavo. Esta es una dirección de 7 bits seguida de un bit de escritura/lectura. El bit de escritura/lectura indica al esclavo la dirección de transferencia. Dos esclavo en el sistema no pueden tener la misma dirección. Sólo el esclavo con la dirección que es igual a la trasmitida por el maestro responde retornando un bit de acknowlegde (ACK) bajando la linea SDA al noveno ciclo de reloj de SCL.

Una vez que se ha logrado direccionar correctamente al esclavo, puede procederse a realizar la transferencia de datos en la dirección especificada por el bit de escritura/lectura. Cada byte transferido es seguido de un bit ACK al noveno ciclo de reloj de SCL. Si el esclavo envía una señal de No Acknowledge (NACK, es decir el esclavo no baja la linea SDA al noveno ciclo de reloj) el maestro puede generar una señal de STOP para cancelar la transferencia o bien generar una señal repetida de START para comenzar un nuevo ciclo de transferencia.

Si el maestro,mientras esta recibiendo, no genera una señal de ACK al esclavo, el esclavo libera la linea de SDA para que le maestro pueda generar una señal de STOP o de START repetida.

El maestro puede terminar la comunicación generando un señal de STOP. Esta señal usualmente referida como el bit P, esta definida como un transición de bajo a alto mientra SCL este en estado alto. Posteriormente a esto, cualquier otro maestro puede tomar el control del bus, pues ambas lineas están en el estado alto.

2.4 uClinux (Sistema Operativo)

Para el desarrollo de este sistema, se utilizó el sistema operativo uClinux en su versión uClinux-dist, la cual puede ser encontrada en www.uClinux.org.

A continuación se detallan los principales componentes de este sistema:

En el corazón de la distribución de uClinux, esta el **kernel de uClinux**, modificado especialmente para procesadores sin MMU. El kernel está específicamente construido para el procesador NIOS II de Altera, y está además disponible para varias otras arquitecturas. El kernel es esencialmente el sistema operativo sin ningún sistema de archivos. Sus tareas incluyen realizar el soporte para los drivers del hardware utilizado para comunicarse con los distintos periféricos, como también permitir que múltiples tares puedan ser ejecutadas concurrentemente. El kernel puede ser reconfigurado para incluir o remover drivers de dispositivos y funcionalidades de sistema operativo. La imagen incluida esta disponible en código fuente, siendo necesario la configuración y la generación de archivos binarios de estas fuentes.

El siguiente componente del sistema operativo es el **sistema de archivos**. Una vez que el kernel parte, debe montar un sistema de archivos raíz (rootfs). Además, es posible montar distintos sistemas de archivos luego, como tarjetas Flash removibles o sistemas de archivos en red remotos (NFS del ingles Network File System). Esto es dependiente del hardware utilizado y de la existencia de los drivers apropiados. Incluido con uClinux existe un sistema de archivos capaz de ser montado en la memoria RAM usada por el sistema operativo, y que puede ser guardado en un solo archivo que es posible guardar en una memoria no volátil para cargarlo al iniciarse el sistema.

Busybox (http://www.busybox.net) es un paquete que combina todos los comandos comunes de Linux en un solo ejecutable. Esto es particularmente útil para sistemas embebidos pues simplifica el sistema de archivos final. Comandos como ping, cp, rm y reboot son ejemplos de comandos que están incluidos en Busybox.

Finalmente es necesaria una **"toolchain"**, la cual es usada para compilar el kernel, busybox y aplicaciones de usuario en un PC con alguna distribución de Linux. Dado que uClinux requiere ejecutables en un distinto formato que un Linux PC (pues su arquitectura es distinta) es necesario realizar compilación cruzada (cross compilation), donde es que la toolchain se utiliza. Basado en GCC (compilador GNU de Linux), cuando se activa el flag de compilación cruzada, los binarios de la toolchain son usados en vez de los binarios nativos de Linux para compilar los componentes especificados. Además cualquier aplicación creada por un usuario que sea realizada después de que el sistema este finalizado puede ser compilada con la toolchain.

Buildroot (http://buildroot.uclibc.org/) es el programa encargado de la creación de esta herramienta, este programa esta compuesto por un set de Makefiles, y parches que facilitan la creación de un toolchain para compilación cruzada, siendo esta la herramienta en definitiva utilizada para compilar cualquier aplicación para el sistema.

3 Desarrollo de la memoria

En este capitulo se explicarán todo el desarrollo realizado en esta memoria para poder obtener finalmente el sistema funcional.

3.1 Selección de la tecnología y sistema a utilizar

La plataforma que se buscaba desarrollar no poseía ninguna preferencia sobre el sistema en el cual se implementaría el sistema, por ende se analizaron los dispositivos con los cuales se estaba trabajando en la empresa para ver cual sería el más acertado para este tipo de tarea.

Los sistemas que poseen conexión a LAN integrada, son los únicos que eran capaces de producir conexión a 1Gbps.El desarrollo de esta memoria, se utilizó los nuevos DSP que recientemente salieron al mercado para principalmente realizar el procesamiento de Video.

Pero estos dispositivos incluían demasiados periféricos que no se utilizarían para el sistema, por estar diseñados para video, con lo cual el precio de los dispositivos no fue rentable para poder producir una interfaz Ethernet que se pudiera utilizar en los equipos de la empresa. Con lo cual, se deja de lado esta solución.

Para poder disponer de un periférico de relativo bajo costo que permitiera lograr los requerimientos de velocidad, se analizaron desarrollos de soluciones basadas en FPGA, en donde existen algunos núcleos que podrán producir la interfaz necesaria para ingresar a un chip con interfaz GMII, de esta manera se ve que con esta tecnología es posible conectarse con periféricos que permitirían lograr producir conexiones de Gbps, a los cuales se accede a sus registros a velocidades de 125 Mhz con 8 bits, siendo estos dispositivos luego los que luego producen la señal de alta velocidad.

Se procedió entonces en ahondar sobe los dispositivos FPGA, en donde se encontró la tecnología de microprocesadores embebidos, los cuales permiten desarrollar un sistema en uno de estos dispositivos, conformado por uno o mas procesadores, y de una cantidad de periféricos totalmente flexible y configurable al momento de crear los archivos de configuración de estos dispositivos (archivos SOF). Algunos de estos periféricos, también llamados módulos, son distribuidos por la misma empresa que realiza los dispositivos lógicos, u otras empresas externas a las cuales se les puede comprar estos módulos, mientras que también existen módulos de libre distribución, así como también la posibilidad de escribir nuevos periféricos en cualquier idioma de descripción de hardware como Verilog o VHDL, directamente en la empresa.

De esta manera, es posible en una primera instancia realizar un sistema en un solo chip, que incluya un microprocesador, y distintas interfaces, en particular la interfaz i2c y la de conexión con un controlador Ethernet, que es lo que se busca para el desarrollo de la memoria. Se procedió a realizar la compra de una placa de desarrollo de estos dispositivos y se acoto el trabajo de memoria a desarrollar un sistema que fuera capaz de conectarse a 100 Mbps, quedando garantizada la posibilidad de expansión a 1Gbit con bajo costo.

Entre la selección de los dispositivos a utilizar, se acordó que el sistema se desarrollaría en una FPGA de bajo costo, las cuales no disponen de periféricos especiales de alta velocidad, y producen sistemas con limitaciones mayores de velocidad que las FPGA de alta velocidad, pero que permiten tener un total control sobre las compuertas del sistema. La empresa seleccionada fue Altera, la cual dispone de un microprocesador embebido llamado Nios II, y la FPGA en que desarrollara el sistema es una FPGA de la familia Cyclone II.

La empresa Altera, además dispone de un nuevo dispositivo de esta familia de FPGA de bajo costo, llamada Cyclone III, la cual tiene menor consumo de potencia, una mayor cantidad de memoria RAM integrada y un precio levemente menor al de su versión mas antigua. Pero al ser este dispositivo mas reciente, aún no se disponían de Kits de desarrollo para el procesador embebido y de este dispositivo, razón por la cual se trabajó finalmente sobre el dispositivo Cyclone II .

El procesador embebido de la empresa Altera, llamado Nios II, requiere una licencia vino incluida con el kit de desarrollo que se compro en la empresa, con la posibilidad de ser programado directamente en C, pero también, es posible programa sobre un Sistemas Operativos ejecutándose en el procesador. Dado esto para disponer de protocolos TCP/UDP y aprovechar las posibilidades de multiprocesos que un Sistema operativo entrega, se decidió utilizar un Sistema Operativo de libre distribución en particular uClinux, el cual está basado en el núcleo de Linux, pero con énfasis para trabajar en sistemas embebidos, sin MMU, lo cual es una restricción de la mayorías de microprocesadores, en particular del seleccionado para esta memoria.

Tomadas estas decisiones sobre la tecnología y la arquitectura en la que se trabajó en la memoria, se procedió a desarrollar la implementación del procesador mismo, en el cual se utilizaron los siguientes dispositivos externos a la FPGA:

- Interfaz Ethernet de 10/100 Mbps, la serialización fue realizada por el chip LAN91C111, un chip no PCI que dispone de capa física (PHY) y capa MAC, alivianando el trabajo del procesador en la comunicación por Ethernet.

- Memoria del sistema, una memoria DDR, conectada a la FPGA. En esta memoria estará corriendo cualquier programa de la FPGA, así como también el sistema de archivos del sistema operativo. No se trabaja directamente sobre una memoria no volátil por asuntos de velocidad. La memoria que dispone la placa de desarrollo utilizada es de 32MB, la que se controla mediante 16 lineas de datos y 13 lineas de direcciones.

- La memoria no volátil del sistema, es necesaria pues todo el sistema esta montado en una FPGA la cual tiene cada uno de sus compuertas realizadas en RAM, se utilizará la ofrecida por la misma empresa Altera, la que dispone de la posibilidad de cargar automáticamente el procesador y sus periféricos en la FPGA en el encendido del sistema. Además, todo el núcleo de uClinux será guardado en esta misma memoria, de manera que el procesador luego de ser cargado en la FPGA, procede a ejecutarse desde esta misma memoria, cargando el sistema en la memoria RAM externa finalmente para que el sistema inicie su funcionamiento normal.

- La conexión I2C, será realizada mediante un periférico HDL multi-maestro Open Source, diseñado en Verilog. Dado que este Core utilizado no soporta modo esclavo, se impone la condición de que todos los datos que se quieran ser accedidos desde el sistema sean desde dispositivos esclavos del bus I2C, de esta manera este dispositivo al esta supervisando y controlando otros equipos, debe accederlos, pudiéndose eso si, tener mas de un maestro en el bus, siempre y cuando estos otros dispositivos tengan la posibilidad de ser multi-maestro.

3.2 Desarrollo del SOPC montado sobre la FPGA (Procesador y periféricos)

Para el desarrollo de esta memoria, se decidió trabajar directamente sobre una plataforma de desarrollo de la empresa Altera, de manera de poder centrar el trabajo en el desarrollo del sistema desarrollado en la FPGA, ya teniendo el hardware sobre el cual trabajar. Además, el uso de una plataforma de desarrollo, también trae la ventaja de tener la licencia del procesador que se utilizará para el desarrollo, sin la cual esta debería comprarse aparte.

La plataforma de desarrollo que se utilizó fue la siguiente:

"**Nios II Development Kit, Cyclone II Edition (2C35)**"

Esta plataforma de desarrollo, además de incluir todos los periféricos necesarios para hacer funcionar a la FPGA, incluye varios chips que no se ocuparán en el desarrollo final. En la ilustración 6 se muestra el diagrama de bloque de la placa utilizada, con todos los chips externos que esta incluye:

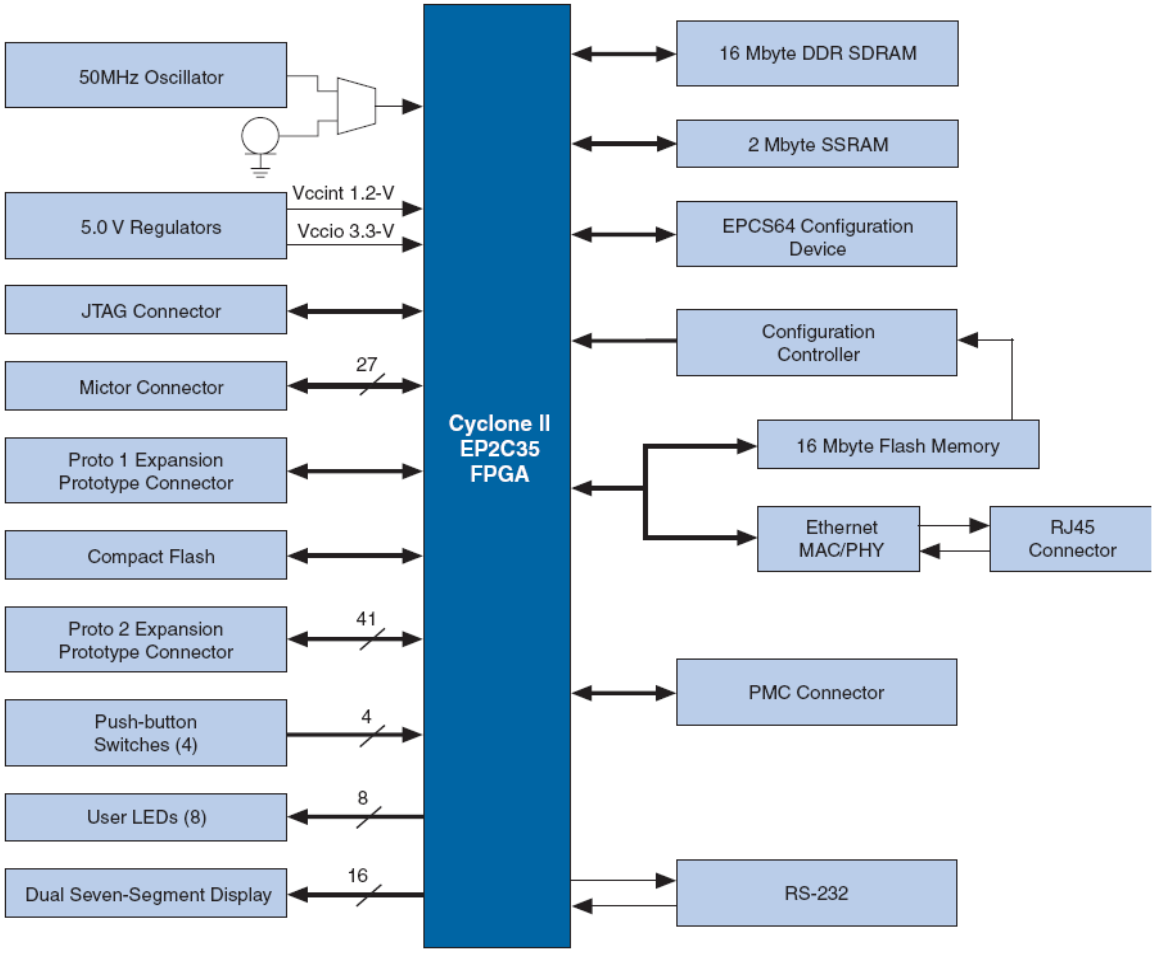


Ilustración 6: Diagrama de bloques de plataforma de desarrollo utilizada

La FPGA que viene en esta plataforma incluye 33.262 elementos lógicos que pueden ser programados de cualquier manera. Los dispositivos externos de esta plataforma que realmente fueron utilizados se verifican en la configuración de la FPGA, lo cual es explicado un poco mas adelante en este mismo punto, junto con la descripción de cada uno de los componentes utilizados realmente.

Para la programación de la FPGA en la cual se incluye este KIT, así como también, varios otros dispositivo programables del mismo fabricante, se requiere utilizar herramientas de software que el mismo fabricante pone a la disposición de sus consumidores. Dado esto, los programas utilizados para desarrollar, tanto el archivo de configuración de la FPGA (HDL), así como también, para programar el sistema, con el microprocesador Nios II incluido.

Se enumeran a continuación estos programas en el orden requerido de instalación:

- **Quartus® II Software:** Es el software encargado del diseño, síntesis, simulación y programación de las FPGA de Altera, esta disponible para Windows y Linux, en su versión de suscripción y disponible en Windows en una versión Web sin costo.

- **MegaCore IP Library:** Es la librería con distintos dispositivos de Propiedad Intelectual (IP), entregada por Altera para su evaluación. Esta librería, incluye el procesador Nios II, el cual cuenta con licencia en la empresa.

- **Nios II Embedded Design Suite:** Es el software para programar directamente sobre el procesador Nios II, incluye además ejemplos de sistemas con este procesador, para su utilización en plataformas de desarrollo, así como también programas en linea de comando para poder programar este sistema.

Todos estos programas están disponibles para Windows, así como también para Linux, en sus versiones de Red Hat Enterprise Linux 3-4 y SUSE Entrerprise Linux 9. Dado que se tuvo una licencia del programa Quartus II y a que se trabajó con un sistema operativo basado en Linux en el sistema embebido, se optó por utilizar la versión de Linux de los programas, la cual al ser compatible con Red Hat Enterprise Linux 4, se utilizará una distribución de Linux gratuita compatible con esta distribución, particularmente Centos 4.

El primer paso para el desarrollo del sistema, fue la instalación de los programas antes mencionados. En el kit de desarrollo vienen las versiones 7.0 de estos mismos, pero se optó por trabajar con la versión mas reciente a la fecha de estos software, la versión 7.2, la cual entrega una compilación mas rápida del sistema a montar en la FPGA, así como también, una mejor solución a la compilación del Procesador (es decir una mayor frecuencia de operación posible).

En este software se desarrollo el sistema integrado en la FPGA, este sistema fue desarrollado con la herramienta “SOPC Builder”, en la cual, se añaden componentes al sistema y este programa se encarga integrar todos los componentes con el bus propietario de Altera Avalon®, asignado los relojes necesarios para cada componente, los rangos de memoria en los cuales estarán mapeados estos, así como también, las interrupciones y su prioridad para el sistema.

En la figura 7 se muestra el sistema implementado en el SOPC Builder, para la plataforma de desarrollo en el cual se trabajó en la memoria. Todos los componentes que posee dicho sistema están disponibles con la instalación ya realizada de la herramientas de Altera, con excepción, del bloque I2C, para lo cual se utilizará un periférico de código abierto, el cual puede ser descargado desde **www.opencores.org**. Este periférico viene especificado con un bus Wishbone, el cual es de código abierto, pero se integro con el bus Avalon realizando las conexiones entre ambos buses en el diseño del sistema en HDL, dado que las funcionalidades implementadas en este núcleo, así lo permiten.

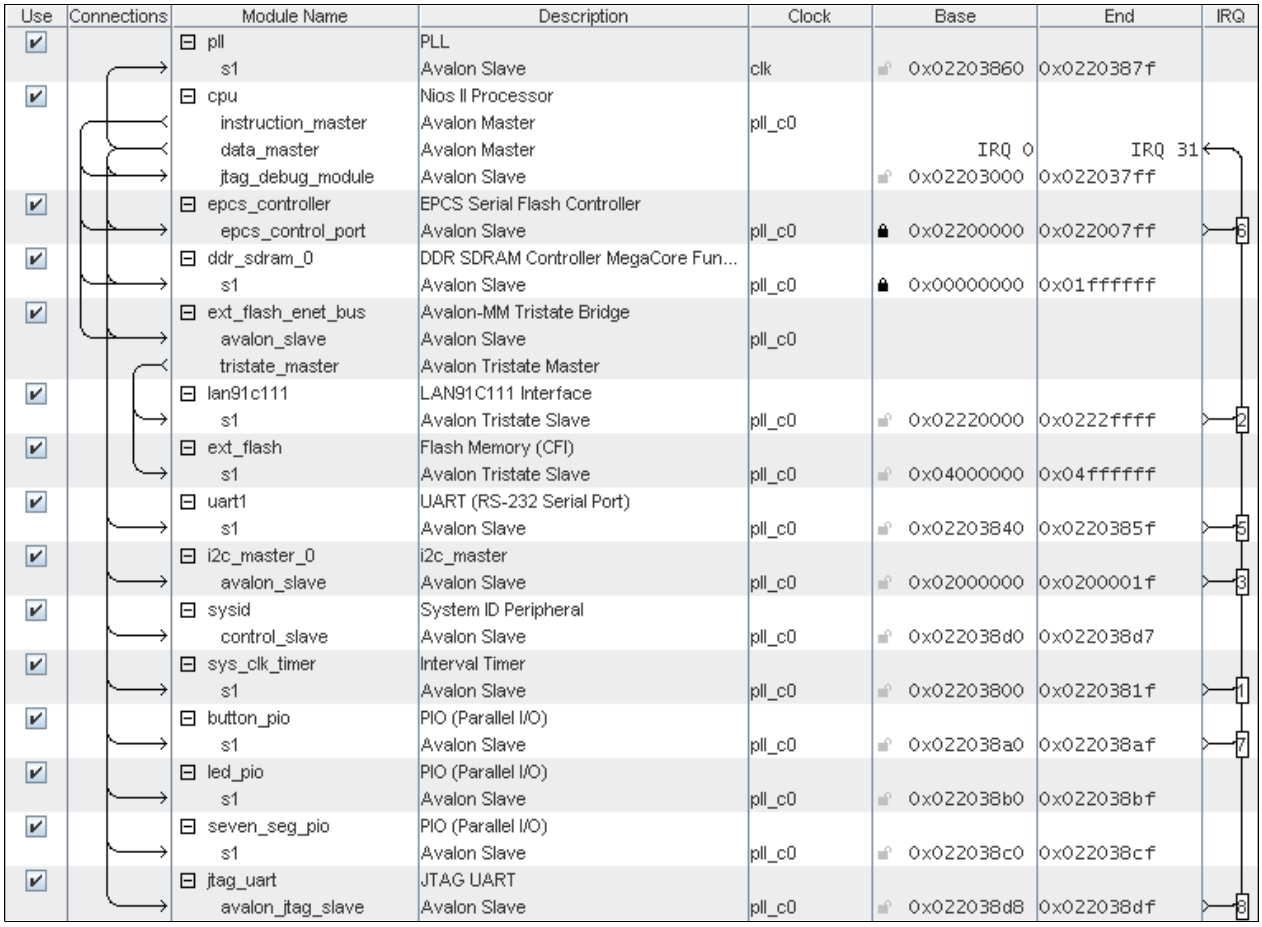


Ilustración 7: Sistema en SOPC Builder

Los módulos utilizados, y su funcionalidades se detallan a continuación:

- PLL: Se encarga de cambiar el reloj de entrada. En este caso, se tenía un reloj de entrada de 50 Mhz, desde donde se obtiene dos relojes de 85 Mhz. Un Primer reloj sin desface que sirve para todos los componentes del sistema internamente. El segundo reloj se utiliza de entrada para la memoria sdram del sistema y se crea con un desface entre esta y el reloj principal del sistema, con lo cual se puede tener control sobre cuando llega el flanco de subida del reloj a esta memoria, buscando que esto sea una vez que todas las señales de entrada estén en un estado estable.

- CPU: Es el procesador del sistema, como ya se mencionó, es un procesador propietario de la empresa Altera llamado Nios II, este procesador, permite llegar (a una frecuencia de 85 Mhz) hasta 86 DMIPS.
El procesador, posee ademas en esta configuración 4 Kbytes de memoria cache de instrucciones y 2 Kbytes de memoria cache de datos.

- EPCS_CONTROLLER: Es la memoria no volátil que se usará en el sistema final. Es una memoria que permite configurar directamente la FPGA, sin necesidad de tener otro dispositivo dedicado a realizar la programación de esta una vez que el sistema es alimentado. Esta memoria además se usará para guardar tanto la configuración de la FPGA, como también para guardar el sistema operativo, junto con todas los programas desarrollados y la pagina Web.

- DDR_SDRAM_0: Esta es la memoria sobre la cual se ejecutará el sistema operativo. Es una memoria DDR síncrona, pero es posible cambiar el tipo de memoria RAM, cambiando este periférico por otro controlador. Este dispositivo no está limitado a un solo chip de memoria DDR SDRAM, sino que permite cambiar sus parámetros para funcionar con cualquier memoria de este tipo.

- EXT_FLASH_ENET_BUS: Es un puente de tres estados, el cual conecta dispositivos fuera del chip, al sistema de interconexión en el chip. Este dispositivo crea las señales de entrada y salida, que se deben conectar a los pines de la FPGA en el proyecto de Quartus II. Este puente crea pines de direcciones y de datos que pueden se compartidos entre varios dispositivos externos, en el caso de la placa de desarrollo, están compartidos la memoria FLASH, y el dispositivo LAN91C111.

- LAN91C111: Asigna direcciones de memoria, interrupciones, y especifica como está conectado el chip LAN91C111, el cual, en este diseño es el encargado de realizar la capa MAC y PHY de conexión a la red LAN. Una de las ventajas del uso de este chip, es que ya están disponibles los controladores para uClinux, de este dispositivo, pudiendo habilitarse sin tener que realizar nuevos drivers.

- EXT_FLASH: Esta memoria flash, va a ser removida del diseño final, pero se mantiene en este proyecto, debido a que el programa de compilación del sistema uClinux, requiere una entrada para este tipo de memoria, aunque no se utilice. Mencionado esto, en esta memoria no se utilizó en el desarrollo, sino que se utilizó como memoria no volátil, la memoria EPCS descrita anteriormente.

- UART1: Interfaz serial de conexión al sistema. El terminal de uClinux está dirigido a este periférico para poder acceder a la plataforma directamente sin necesidad de la interfaz Ethernet, lo cual es útil. Por ejemplo: Para configurar la IP que utilizará el equipo, como también, para ver los mensajes de inicialización del sistema. Este dispositivo está configurado para funcionar a una velocidad de 115200 bps.

- I2C_MASTER_0: Módulo encargado del manejo del bus I2C, es de libre distribución y permite trabajar en modo multi-maestro con velocidades de transmisión de hasta 400 kHz. Sus características principales son:

    - Permite operación multi maestro
    - Frecuencia de reloj configurable mediante registros
    - Envío de bit de "acknowledge"
    - Generación de señales de Inicio, detección, inicio repetido, y su detección.
    - Detección de perdida de arbitraje del bus, terminando automáticamente la transferencia
    - Detección de bus I2C ocupado

Este módulo será el encargado de comunicar el dispositivo con el resto del sistema a controlar, por esto se utilizó un periférico que permitiera tener el manejo del bus integrado, en vez de optar por una interfaz menos compleja en base a pines de propósito general. El manejo de este módulo, se realizó directamente sobre los registros del periférico sin utilizarse drivers de Linux, debido a que la complejidad que esto agrega al diseño no compensa los beneficios que puede traer esta solución, pudiendo tenerse todas las funcionalidades solicitadas sin recurrir a drivers de Linux.

- SYSID: Dispositivo de sólo lectura, que entrega un identificador único del SOPC Builder. Su uso es principalmente para que un programa diseñado para un procesador, no sea utilizado con otra versión del sistema, para la cual no fue diseñada. Este modulo se mantiene para utilizarlo mientras se compila directamente sobre el procesador Nios II, pero uClinux no lo utiliza.

- SYS_CLK_TIMER: Módulo para controlar intervalos, puede generar interrupciones periódicas o contar el tiempo desde el inicio del sistema.

- BUTTON_PIO – LED_PIO – SEVEN_SEG_PIO: Módulos de entrada y salida paralela, permiten escribir y leer, dependiendo del sentido del periférico con que se ha compilado, los estados de pines genéricos. En la placa de desarrollo se utilizaron para poder verificar fácilmente el modelo de programación utilizado, accediendo a estos periféricos simples

- JTAG_UART: Interfaz serial realizada mediante el puerto JTAG, tiene la desventaja de que al dirigirse la salida de uClinux a este puerto, el procesador se bloquea hasta que se active una consola en este puerto mediante un cable de programación.

Este sistema, es el que finalmente contiene todos los periféricos necesarios para el sistema que se implementará definitivamente en la empresa. En esta plataforma al momento de la redacción de esta memoria, se empezó a realizar como parte de otro proyecto de la empresa, donde los bloques necesarios son los mismos, y solo se diferencia en que se añade un bloque PCI al sistema (conectores unidos a la FPGA).

Una vez creado el sistema (SOPC), se realizó una compilación del sistema, lo que finalmente se traduce en la creación de un archivo .ptf con información de los módulos creados, con su rango de memoria e interrupciones, el cual, es el archivo de entrada para la compilación del sistema operativo entregándole al uClinux toda la información de rangos de memorias y periféricos disponibles en el sistema donde este se ejecutará.

Finalizado el proceso de creación del SOPC, se utilizó el bloque creado por este programa, para agregarlo a un proyecto. En este caso, el bloque superior del sistema se realizó en modo esquemático, donde posteriormente se agregaron los nombres de los pines que realmente van a salir de la FPGA, también es posible en este nivel añadir más lógica al sistema, sin necesariamente sacar las señales de la FPGA y pre-procesarlas en el interior de esta.

Ilustración 8: Esquemático del Sistema

El siguiente paso, una vez asignado nombres a las señales que saldrán de la FPGA como en la Ilustración anterior, se realiza la asignación de pines en el mismo sistema, y la compilación del sistema. Finalmente se obtiene la siguiente utilización de recursos del procesador y los periféricos creados:

Elementos lógicos usados:	5,067
Pines totales (incluido FLASH):	164
Bits de memoria utilizados:	78,336
PLL utilizados:	1

Estos datos son los que finalmente delimitan cual es el limite mínimo de dispositivos lógicos que debe poseer la FPGA que se debe utilizar en el diseño definitivo de la empresa. Dado que se trabajó en una placa de desarrollo con una FPGA de 35,000 compuertas, la utilización del dispositivo es de aproximadamente un 15%.

La compilación del sistema generó un archivo el cual puede cargarse directamente en la RAM de la FPGA mediante un cable de programación, o en la memoria no volátil del sistema mediante la herramientas provistas para este fin. Finalizado este proceso, se comenzó a trabajar directamente sobre la plataforma de desarrollo.

3.3 Integración del Sistema Operativo uClinux en el sistema

Una vez obtenido un procesador embebido que se instaló sobre la FPGA del sistema. Se procedió a montar en este mismo procesador el sistema operativo uClinux. Para la realización de este paso, se utilizaron herramientas disponibles desde Internet, desde donde la pagina http://nioswiki,jot.com la cual fué de gran utilidad.

El sistema operativo utilizado fue la versión de uClinux-dist con versión 20070103. Esta versión además fué parchada para el procesador nios2. Estos parches, pueden ser encontrados en la pagina http://nioswiki.jot.com. Lo anterior resulta ser necesario, debido a que las herramientas utilizadas están a un en fase de desarrollo, teniendo el sistema de generación del sistema operativo aún varias aplicaciones que no funcionan directamente sobre el sistema, siendo necesario modificar los códigos fuentes de las aplicaciones que se incluirán.

Con esta versión de uClinux, se puede compilar un kernel, y todo un sistema de archivos para utilizar en la FPGA. En el caso de esta memoria, se optó por usar un sistema de archivos en la memoria RAM disponible del sistema, pues era la opción más rápida. Por ende, el sistema puede ser cargado directamente en la memoria RAM mediante un cable de programación, perdiéndose el sistema de archivos al des-energizarse el sistema, y en un diseño definitivo cargarse en la memoria no volátil serial, que mencionó en el diseño del dispositivo.

Se utilizó la versión del núcleo de Linux 2.6.19-uc1, con drivers para el manejo del puerto serial, dispositivo Ethernet, manejo de botones desde el sistema de archivos. En las aplicaciones se destacan la selección del servidor web BOA, así como también la integración de la mayoría de aplicaciones de manejo de archivos y directorios implementadas directamente sobre el programa BusyBox, el cual integra las herramientas estándar típicas de Linux, en un solo pequeño ejecutable.

Este sistema, en su totalidad, produce finalmente un sólo archivo, que incluye todas las aplicaciones ya seleccionadas el cual tiene un tamaño aproximado de 1.3MB (1357902 bytes) comprimido, el cual al pasar a memoria RAM, en un primer paso es descomprimido ocupando un total aproximado de 2MB en memoria RAM.

En el inicio del sistema, parten automáticamente, servidores telnet, y ftp, servidor web boa, además del servidor que se encargará finalmente de enviar la información del sistema y recibir respuestas desde el usuario final.

A continuación se muestran los mensajes de inicio del sistema a través del puerto serial:

```
   Uncompressing Linux... Ok, booting the kernel.
   Linux version 2.6.19-uc1 (fconcha@centos) (gcc version 3.4.6) #47 PREEMPT
Thu Oct 25 16:56:59 CLST 2007


   uClinux/Nios II
   Altera Nios II support (C) 2004 Microtronix Datacom Ltd.

   setup_arch: No persistant network settings signature at 04FF0000
   Built 1 zonelists.  Total pages: 8128
   Kernel command line:
   PID hash table entries: 128 (order: 7, 512 bytes)
   Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
   Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
   Memory available: 30208k/32768k RAM, 0k/0k ROM (1562k kernel code, 701k
data)
   Mount-cache hash table entries: 512
   NET: Registered protocol family 16
   NET: Registered protocol family 2
   IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
   TCP established hash table entries: 1024 (order: 0, 4096 bytes)
   TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
   TCP: Hash tables configured (established 1024 bind 1024)
   TCP reno registered
   io scheduler noop registered
   io scheduler deadline registered (default)
   NIOS serial driver version 0.0
   ttyS0 (irq = 5) is a builtin NIOS UART
   smc91x.c: v1.1, sep 22 2004 by Nicolas Pitre <nico@cam.org>
   eth0: SMC91C11xFD (rev 2) at 82220300 IRQ 2 [nowait]
   eth0: Ethernet addr: 00:07:ed:00:00:00
   TCP cubic registered
   NET: Registered protocol family 1
   NET: Registered protocol family 17
   Freeing unused kernel memory: 580k freed (0x194000 - 0x224000)
   Shell invoked to run file: /etc/rc
```

```
Command: hostname uClinux
Command: mount -t proc proc /proc
Command: mount -t sysfs sysfs /sys
Command: mount -t usbfs none /proc/bus/usb
mount: Mounting none on /proc/bus/usb failed: No such file or directory
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo
Command: ifconfig eth0 192.168.0.200  # staic ip
eth0: link down
Command: route add default gw 192.168.0.1     # gateway
BusyBox v1.00 (2007.10.11-18:35+0000) multi-call binary

Usage: route [{add|del|delete}]

Command: cat /etc/motd
Welcome to
          ____ _  _
         /  __| ||_|
     _   _| |  | | _ ____  _   _  _  _
    | | | | |  | || |  _ \| | | |\ \/ /
    | |_| | |__| || | | | | |_| |/    \
    |  ___\____|_||_|_| |_|\____|\_/\_/
    | |
    |_|

For further information check:
http://www.uclinux.org/

Command: inetd &
[20]
Command: boa &
[21]
eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
Command:
Execution Finished, Exiting

Sash command shell (version 1.1.1)
/>
```

Una vez teniendo el sistema operativo ejecutándose, se puede acceder a este, ingresando comandos desde la interfaz serial, o desde el programa telnet, teniéndose este servicio habilitado para desarrollo pero sin ser recomendable de usar para el producto final. Pues, este tipo de acceso no ofrece ningún tipo de seguridad en el dispositivo, permitiendo a cualquier persona en la red a cambiar las configuraciones del sistema trabajando directamente sobre el sistema operativo uClinux, lo cual no es lo que se desea en el producto final, punto en el cual se restringirá el acceso a través de la interfaz LAN al dispositivo.

En este sistema es posible realizar programas en código C, compilarlos para la arquitectura utilizada, cargarlos en el sistema ya sea por ftp o mediante el nfs (network file system) y poder ejecutarlo en la memoria RAM del dispositivo.

A modo de ejemplo se muestra un programa simple, que enciende y apaga luces de los led de la placa de desarrollo, y leyendo los contenidos del modulo de botones. El programa modifica directamente los registros del periférico paralelo de entrada y salida del dispositivo led_pio, y lee los datos del periférico button_pio. Este programa lee el número que se desea mostrar desde la entrada estándar, la cual puede ser una consola de telnet o mediante la interfaz serial, sin tener ningún cambio en el código, pues para Linux esto es transparente.

```
#include <stdio.h>
#include <stdlib.h>
#include "/home/fconcha/uClinux-dist/linux-2.6.x/include/nios2_system.h"

#define readl(addr)     \
({                                      \
      unsigned int __res;\
      __asm__ __volatile__( \
            "ldwio %0, 0(%1)" \
            : "=r"(__res)     \
            : "r" (addr));    \
      __res;                            \
})

#define writel(b,addr)  \
({                                      \
      __asm__ __volatile__( \
            "stwio %0, 0(%1)" \
            : : "r"(b), "r" (addr));      \
})

#define SYSTEM_BUS_WIDTH 32
#define __IO_CALC_ADDRESS_NATIVE(BASE, REGNUM) \
  ((void *)(((unsigned char*)BASE) + ((REGNUM) * (SYSTEM_BUS_WIDTH/8))))
#define __builtin_ldwio(addr) readl(addr)
#define __builtin_stwio(addr,data) writel(data,addr)
#define IORD(BASE, REGNUM) \
  __builtin_ldwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)))
#define IOWR(BASE, REGNUM, DATA) \
  __builtin_stwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)), (DATA))

int main(void)
{
size_t n_datos=10;
int dato;
int i;
char *string;
int datos;
long numeros;
printf("¡Hola mundo!\n");
string=malloc(n_datos);

printf("Datos iniciales del periferico button_pio %x\n",(unsigned
```

```
int)na_button_pio );
printf("%s=%x ","data",IORD(na_button_pio ,0));
printf("%s=%x\n","direction",IORD(na_button_pio ,1));
printf("%s=%x ","interruptmask",IORD(na_button_pio ,2));
printf("%s=%x\n","edgecapture",IORD(na_button_pio ,3));

while(1){
   printf("Ingrese un numero:");
   datos=getline(&string,&n_datos,stdin);//lee una linea
   if(datos<=0)break;
   numeros=strtol(string,NULL,10);
   printf("\nComo entero es: %d\n",(int)numeros);
   printf("El contenido del boton es: %x\n",IORD(na_button_pio ,0));
   IOWR(na_led_pio,0,numeros);
}

free(string);
printf("Termine el programa sin ningun problema\n") ;
return(0);
}
```

(Los acentos fueron omitidos para evitar problemas de representación en el código diseñado.)

Este programa simple, modifica los led accediendo directamente al periférico realizado en el HDL como se había mencionado anteriormente. Para realizar las pruebas dicho programa se ejecutó mediante un acceso telnet al dispositivo, estando el archivo ubicado en un computador externo, realizando el acceso a este el Sistema Operativo uClinux mediante NFS. Con esto, se muestra las ventajas de multi-proceso y funcionalidades que permite tener este sistema operativo en el sistema, permitiendo tener múltiples sistemas de archivos e interfaces de acceso (telnet, serial) sin tener que realizar cambio en el código del programa ejecutable. El resultado de la prueba anterior se muestra a continuación:

```
/mnt/nfs> ./hello
¡Hola mundo!
Datos iniciales del periferico button_pio 822038a0
data=f direction=0
interruptmask=0 edgecapture=0
Ingrese un numero:1

Como entero es: 1
El contenido del boton es: f
Ingrese un numero:2

Como entero es: 2
El contenido del boton es: 7
Ingrese un numero:255

Como entero es: 255
El contenido del boton es: 3
Ingrese un numero:30

Como entero es: 30
El contenido del boton es: 8
Ingrese un numero:Termine el programa sin ningun problema
```

El programa mostrado anteriormente, tiene un tamaño compilado de 31Kbytes, cumpliendo con el acceso a los periféricos con lo cual ya se tiene un manejo de todo el sistema creado en el programa SOPC_Builder, con la única limitación de no poder utilizar interrupciones con este sistema, para lo cual es necesario trabajar con los controladores de Linux. Pese a esto esto ya es suficiente para desarrollar todas las aplicaciones deseadas.

3.4 Implementación de los programas de acceso al periférico I2C en uClinux

Una vez de que se dispuso de una forma de acceso a los módulos agregados en el programa SOPC Builder, se procedió a realizar programas específicos para el manejo del bus I2C.

Este bus, como fue explicado en el capitulo dos, dispone de 2 lineas, y tiene las posibilidades de arbitraje, y distintas velocidades de transición, todo lo cual es realizado por el periférico utilizado, el cual esta disponible en Open Cores (www.opencores.org). Para realizar las prueba de manejo del periférico, se utilizó una memoria I2C, en particular una modelo 24LC04B, bajo el supuesto que una vez que se tenga diseñado el bus con el equipo que se controlara finalmente, se tenga ya unos ejemplos de como realizar el acceso al bus y respuestas a este con una memoria que se utiliza realmente en la empresa.

Las funciones básicas implementadas son las siguientes:

- Rutina de inicialización del bus, configurando la velocidad a la cual este operara, y liberando cualquier interrupción que quede pendiente en este bus.

- Escritura de un byte:
  (`int32_t` **`i2c_write_byte`**`(uint8_t ctrl,uint8_t data,uint8_t *i2c_status)`)
  - Se asigna el registro del dato que se requiere enviar
  - Se envía el comando de escritura con cualquier otro comando que se requiera
  - Se verifica el estado del módulo, hasta que se active el bit de interrupción del dispositivo u ocurra mucho tiempo sin tener respuesta del bus.
  - Se verifica si existió respuesta del dispositivo que se esta comunicando.
  - Se limpia la interrupciones.
    En caso de cualquier error, la rutina entrega el ultimo estado leído desde el núcleo i2c, y un código de error.

- Lectura de un byte:.
  (`int32_t` **`i2c_read_byte`**`(uint8_t ctrl,uint8_t *data,uint8_t *i2c_status)`)
  - Se asigna en el registro del módulo i2c el comando de lectura
  - Se verifica el estado del módulo hasta que se active el registro de interrupción, u ocurra mucho tiempo sin recibir una respuesta del núcleo i2c.
  - Se retorna el dato obtenido por el núcleo i2c.
    Es caso de error, se retorna el ultimo estado leído desde el núcleo i2c, así como un código de error

Con estas dos rutinas se realizaron todas las funciones de escritura y lectura, tanto de bytes como de una pagina de varios bytes, específicas para el dispositivo del que se busca controlar. En este caso las rutinas implementadas fueron realizadas para el manejo de la memoria eeprom i2c antes mencionada, la cual se usa regularmente en los diseños de la empresa.

El esquema de conexión de la memoria utilizada con la FPGA se muestra en la ilustración 9, acá ademas se muestra ademas que dentro de la FPGA, se activaron las resistencias de pull-up en ambas entradas del bus I2C, resistencias que son necesarias para el funcionamiento del bus, y que están activas en la lógica del dispositivo antes de entrar al diseño del HDL programado en la FPGA.

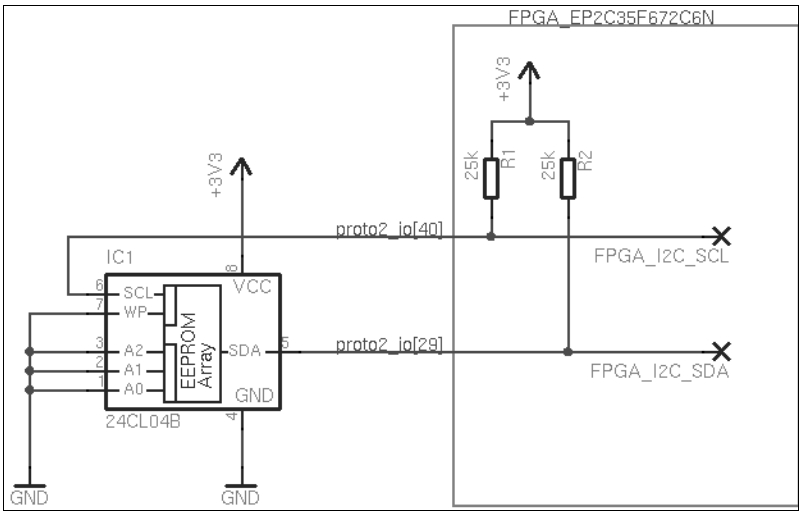


Ilustración 9: Esquemático de conexión de memoria I2C

En la ilustración 10 se muestran los diagramas de flujo de las funciones utilizadas para escribir y leer paginas de la memoria EEPROM, estas fueron las funciones que se utilizaron como base para realizar el control desde la pagina web que se desarrolló

Finalmente para realizar pruebas de las rutinas se realizaron lecturas y escrituras de la memoria EEPROM ya mencionada, enviando un registro de las actividades realizadas por el programa, a un archivo de registro. La velocidad que se logró alcanzar es dependiente de las resistencia de pull-up utilizadas en el bus, las cuales en el caso de las pruebas fueron simplemente las integradas dentro de la FPGA, las cuales tienen un valor típico de 25kΩ, logrando una velocidad máxima de 100kHz.

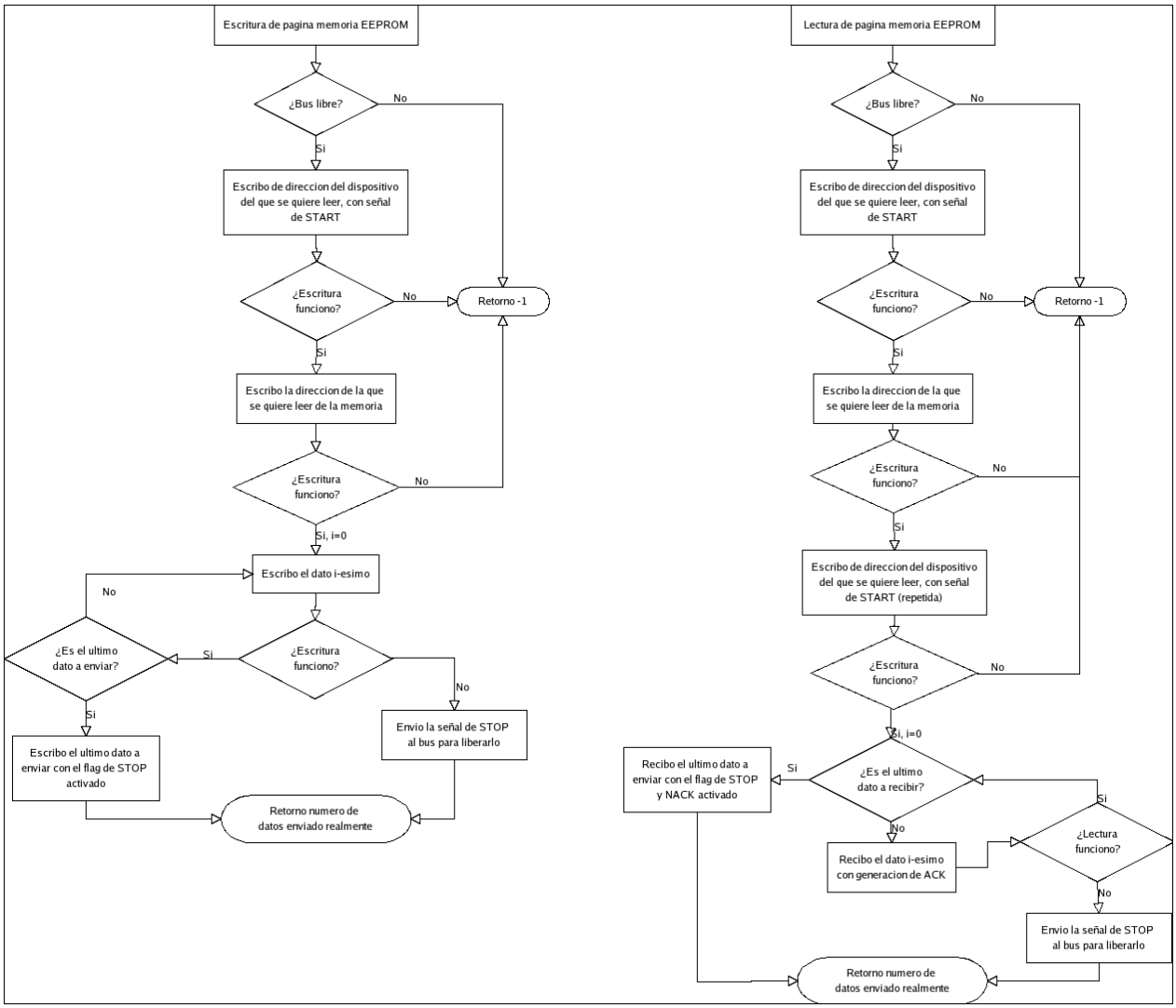


Ilustración 10: Diagramas de Lectura y Escritura de pagina con módulo I2C

3.5 Manejo de la interfaz Web para el control del sistema

El sistema operativo uClinux, compilado para ser utilizado con el procesador embebido Nios II, es por defecto, con un servidor web BOA activado. Dado esto, se aprovechó de utilizar este mismo servidor para el desarrollo de la aplicación web de control del sistema.

Entre las posibles soluciones para desplegar información dinámica del sistema en un navegador web externo, se optó por utilizar para tal fin, un Applet de java, el cual puede correr en un navegador web utilizando la Maquina Virtual de Java (JVM de Java Virtual Machine) y que dispone de varias clases que facilitan el diseño de interfaces usuario maquina de forma interactiva.

El esquema de comunicación utilizado para este desarrollo, se basa en que el cliente ubicando normalmente en un computador conectado a la misma red de la plataforma. Una vez que haya accedido a la pagina web del sistema operativo uClinux, descarga y ejecuta el Applet obtenido desde este servidor, el cual, a su vez se encarga de desplegar toda la interfaz gráfica frente al usuario. Además de realizar una conexión mediante sockets con un programa ejecutándose en uClinux, escrito en el idioma de programación C, desde donde recibe datos de la placa, y a su vez puede enviar datos a esta.

Dicho esto el flujo de la comunicación entre el servidor y su interfaz web son los siguiente:

1. El servidor se inicia, ejecutando el servidor web BOA con las paginas web necesarias y además se carga un programa encargado de comunicarse con el Applet del cliente.

2. Un cliente accede a la pagina principal del sistema mediante su dirección IP, la cual en este caso, es la 192.168.0.200, un vez ahí se le muestra la pagina inicial de donde debe seleccionar el enlace a la página que incluye el Applet

3. Ingresando a la pagina del Applet, este es transmitido por el servido web hacia el navegador web, el cual una vez que lo recibe, inicia el Applet, mostrando de esta forma la interfaz en el servidor.

4. El Applet una vez iniciado, realiza una conexión mediante un socket TCP con el servidor desde el cual fue descargado, en un puerto fijo (1818), hacia donde envía una solicitud de datos para conocer el valor actual de los datos que el Applet (y por ende el usuario) puede modificar. Además este Applet inicia un thread que se ejecuta en paralelo a este, encargado de solamente recibir los datos desde el socket recién abierto, y modificar la interfaz gráfica acorde a estos cambios.
5. El programa servidor, que está enviando y recibiendo datos de los clientes que tienen socket activos con el, o en caso de no tener clientes conectados, esperando nuevas conexiones indefinidamente. Incluye al nuevo cliente en su lista de clientes activos, además en caso de que hubiera estado indefinidamente configura un tiempo de espera para iniciar el envío periódico de datos a el o los clientes que estén conectados.

6. El servidor envía al cliente una respuesta sobe cualquier dato que este solicite, y además periódicamente envía una actualización de las variables que se están monitoreando desde el sistema.

7. En el Applet, cada vez que el usuario solicita un cambio en el sistema usando un botón de acción de este mismo, envía directamente desde la rutina que detecto la acción, el comando mediante el socket hacia el servidor, el cual, a su vez lo decodifica y realiza la acción solicitada respondiendo finalmente al cliente (o a todos los clientes en caso de ser un cambio que todos estos deban observar) el resultado de la acción.

8. Cuando un Applet se desconecta, el servidor verifica si este es el ultimo cliente que tiene conectado, de no ser así, continua enviando datos periódicamente al resto de los clientes. En caso de ser el ultimo cliente, el servidor se bloquea indefinidamente hasta que un nuevo cliente se conecte, iniciando el ciclo nuevamente.

Como se mostró anteriormente, para poder acceder al sistema se diseñaron 2 paginas web, la pagina inicial con un logotipo de la empresa y un enlace a su pagina principal. Un mensaje de bienvenida y además dos de enlace; uno a un programa escrito en cgi de ejemplo y el otro que accede a la pagina donde se muestra el Applet. La pagina de inicio se muestra en la ilustración 11.

La pagina que contiene el Applet, tiene una referencia al Applet que se ejecutará en el cliente con unas etiquetas en HTML que explican al navegador web que el archivo que está ahí es un Applet. Para ahorrar espacio, se comprimen las clases necesarias en un archivo (.jar) siendo este archivo junto con las paginas web antes mencionadas los que se guardan en el sistema de archivos de la plataforma. El tamaño del Applet diseñado, sin comprimir es de 59.4 KB y comprimido es de 32.2 KB.

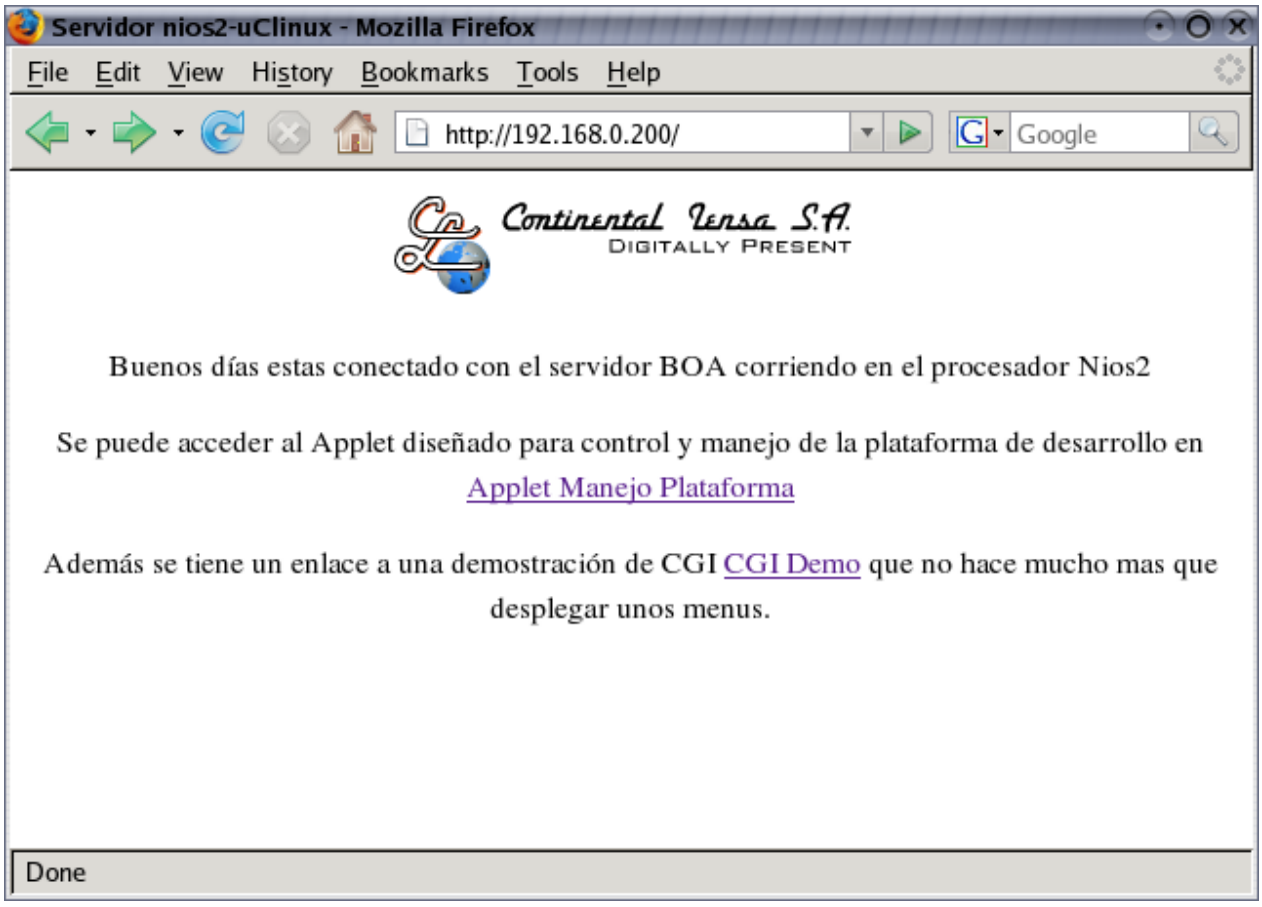


Ilustración 11: Pagina principal de la aplicación desarrollada

Para el desarrollo del Applet se optó por trabajar con la herramienta de programación de libre distribución Netbeans, la cual proporciona de una interfaz gráfica para el diseño de Applet, facilitando todo el trabajo del diseño de la interfaz y permitiendo a la empresa poder cambiar fácilmente la distribución, color y formato de los componentes ingresados en la interfaz gráfica.

La comunicación entre el servidor, ejecutándose en uClinux y el Applet, ejecutándose en la JVM, es realizada en base a arreglos de bytes (8 bits) transmitidos entre ambos programas. Siendo el tamaño mínimo de la transmisión de 4 bytes, enviándose primero un código que identifica la instrucción que se quiere ejecutar, seguido del numero de bytes extras que sigue luego de la instrucción, pudiendo ser estos 0. Después de esta cabecera obligatoria de 4 bytes, siguen los datos extras de la instrucción.

Dado que en java y en C, no existen los mismos tipos básicos de almacenamiento se tuvo que tomar en cuenta los siguientes puntos al comunicar estos dos lenguajes de programación:

- En C el tamaño de un char es de 8 bits, en cambio en java este tipo es de normalmente 16 bits, por esta razón en el programa realizado en java (Applet) se utiliza el tipo de almacenamiento byte que es de 8 bits.

- El procesador Nios II, es de 32 bits, razón por la cual es necesario definir una forma de enviar enteros de 32 bits entre servidor y cliente. Esto se realiza siguiendo la forma de almacenamiento en uClinux, donde el primer byte es el menos significativo, y el cuarto byte es el más significativo. En java el almacenamiento de los enteros no sigue necesariamente esta forma, pues depende de la arquitectura, por ende se debe tomar precaución para las conversiones de enteros a arreglos de bytes, y viceversa.

- En java no existen los tipos de datos sin signo (unsigned), luego la forma que se utiliza para transformar un byte con signo, a un numero sin signo, es almacenarlo como un tipo short (16 bits) con una mascara adecuada, esto es necesario para, por ejemplo convertir un arreglo de bytes en un entero mediante sumas.

El esquema utilizado en el servidor escrito en C, fue que el servidor trabaja dentro de un ciclo infinito, utilizando la función *select()* para saber si tiene mensajes nuevos, los que pueden ser desde el socket que se abre para recibir conexiones nuevas, como de los socket de los clientes que están siendo atendidos. Para poder actualizar a los clientes con información dinámica del sistema, se optó por enviar mensajes periódicamente mientras existan clientes conectados, trabajando en el mismo ciclo utilizando el temporizador del que dispone la función *select*(). Este tiempo entre los envíos periódicos del servidor es configurable desde la interfaz web con el usuario, siendo en un principio de 1 segundo y por el diseño, igual para todos los clientes conectados, pudiendo cualquiera de estos cambiarla.

En esta memoria se optó por realizar el envió del estado de los botones en la placa de desarrollo, para simular un dato dinámico, pudiendo ser reemplazado por otro tipo de datos una vez que se tenga el sistema definitivo en la empresa, pues se aprovecha el que las instrucciones que se envían por el enlace tienen un campo de largo que se puede cambiar, lo que permite tener datos de largo variable, eso sí solo hasta 255 bytes con la implementación realizada.

El protocolo de comunicación que se implementó, se basa en el hecho que está realizado sobre un socket TCP, por ende los datos que se leen de este socket están ordenados y sin pérdidas en la transición. Los comandos que se implementaron, así como una breve descripción del funcionamiento del envió de datos entre el servidor y el arreglo que identifica el comando en C se muestran a continuación:

- Consulta y cambio de la tasa de refresco del servidor: La tasa de refresco es necesaria para enviar el estado de los botones (y/o otros datos) periódicamente a los clientes.

```
static char cmd_cambia_refresh[]={1,8,'U','S'};
```

Al recibirse este comando, el servidor cambia la tasa de actualización actual, utilizando los 8 bytes que vienen a continuación, decodificando estos como 2 enteros. El primer entero son los segundos, y el segundo son los micro segundos, que recibe la instrucción select() como argumentos. Una vez cambiado el tiempo máximo de espera del comando select() se les enviá este mismo comando a todos los clientes para que actualicen el valor que muestran de la tasa de refresco.

```
static char cmd_pregunta_refresh[]={1,0,'?','?'};
```

Esta instrucción hace que el servidor envíe un comando de cambiar la tasa de refresco al cliente que la solicito. Esto es necesario para cuando un cliente se conecta y tenga un valor actualizado de la tasa de refresco con la que está trabajando el servidor.

- Consulta y cambio de los led encendidos en el servidor: Para poder observar un cambio en el dispositivo directamente con un cambio en el Applet de java.

```
char cmd_cambia_led[]={2,0,'L',0};
```

El valor entregado en el ultimo byte de esta cadena, es el numero que se decodificará en el servidor y se desplegará en los led del sistema. Una vez realizado el cambio se les informa a todos los clientes el nuevo valor de los led en el sistema, para que todos estos queden actualizados con el ultimo estado.

```
static char cmd_pregunta_led[]=  {2,0,'?','?'};
```

Enviá al cliente que lo solicite el estado del led actual, nuevamente es necesario para cuando los clientes se conectan al sistema, de manera que no queden con información que realmente no está codificada en el dispositivo.

- Consulta del estado del periférico i2c: Para conocer y desplegar en el Applet de forma decodificada el estado del núcleo i2c, en particular poder ver la velocidad a la que esta configurada el periférico, el registro de estado, y el registro de comandos.

```
static char cmd_pregunta_status_i2c[]=  {3,0,'I','?'};
static char cmd_envia_status_i2c[]=  {3,4,3+4+5,'N'};
```

El servidor envía 4 bytes leídos desde el módulo I2C, 2 bytes del pre-escalador, un byte del registro de control , y finalmente un byte del estado. El primer comando es la consulta realizada desde el Applet cliente, y el segundo cliente es la respuesta a esta petición con los 4 bytes a continuación del comando.

- Escritura de datos en la memoria I2C:

```
char cmd_escribe_en_i2c[]=  {4,'N',0,'W'};//[0] Escribe a la eeprom,[1]
```

El cliente enviá este comando seguido de un arreglo de 'N' bytes para ser escrito en la memoria eeprom. Dada las características de la memoria utilizada, es posible escribir en bloques de máximo de 16 datos de 8 bits. Una vez que la función encargada de la escritura en la memoria eeprom termina, se enviá un comando de estado del periférico i2c, incluyendo como 4º dato ('N') el numero de datos escritos.

- Lectura de datos desde la memoria I2C:

```
char cmd_lee_desde_i2c[]=  {5,0,0,'N'};
```

  El cliente envía esta petición, en donde el 3er byte es la dirección desde donde se desea leer, y el 4to byte ('N') es la cantidad de datos que se desean leer. En este caso, como se trabaja con una sola memoria, no es necesario el envío de más datos. La respuesta del servidor utiliza el mismo comando, cambiando el 2do byte por el número de datos que se adjuntan a este mensaje, de manera que el cliente pueda decodificarlo sin problemas. En caso de error, se envía ademas el estado del periférico para que el cliente lo decodifique.

- Instrucción periódica de envío de datos:

```
static char cmd_interrupcion_periodica[]= {100,0,'B',0};
```

  Este comando se envía cada vez que el tiempo del comando select() llega a 0 en el servidor, se envía solamente un byte con el estado de los botones del sistema, el cual viaja en el 4to byte de esta instrucción. Esta instrucción luego es leída por el Applet.

En el Applet, el funcionamiento esta realizado en base a eventos para todo lo relacionado con el envío de datos hacia el servidor, detectándose cualquier cambio en los botones asignados para esto y enviando el comando hacia el servidor inmediatamente.

En el caso de la lectura de datos desde el servidor, este se realiza en un thread aparte, que se encarga de leer desde el socket hasta que este se cierre. La lectura del socket bloquea a este thread hasta que hay un nuevo dato, en cuyo momento se decodifican los primeros 4 bytes recibidos y se reciben los datos extras en caso de estar informados en el encabezado del comando, para luego realizarse las acciones explicadas anteriormente dependiendo del código inicial del comando recibido.

Finalmente, con todas estas funciones realizadas, el Applet final, se dividió en dos categorías, la primera encargada de enviar información dinámica hacia y desde la placa de desarrollo que se muestra en las ilustraciones 12, 13 y 14, de manera que se puedan observar cambios en esta plataforma producida desde el Applet en el servidor así como también observar cambios producidos en la plataforma de manera gráfica en el Applet.

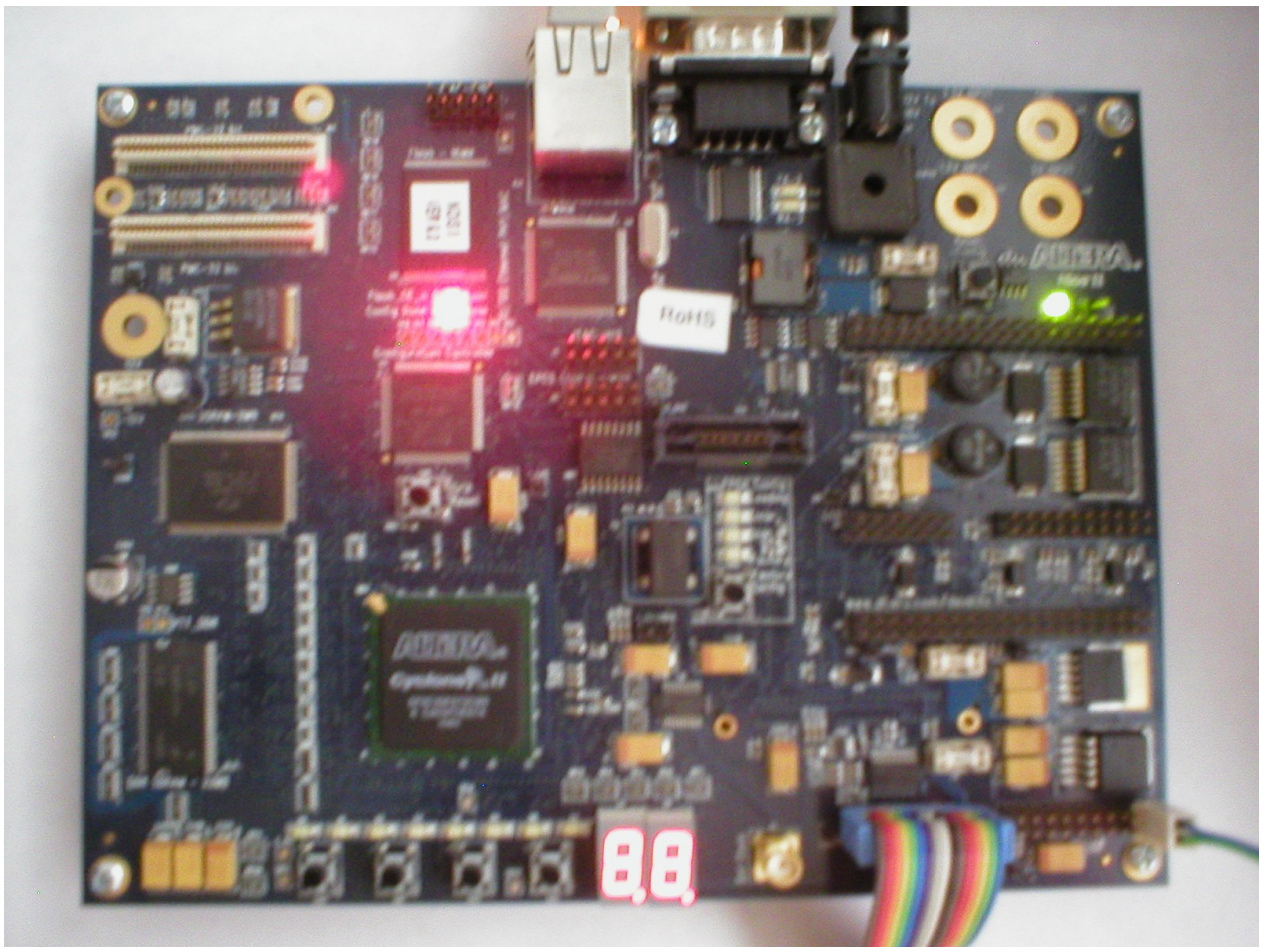
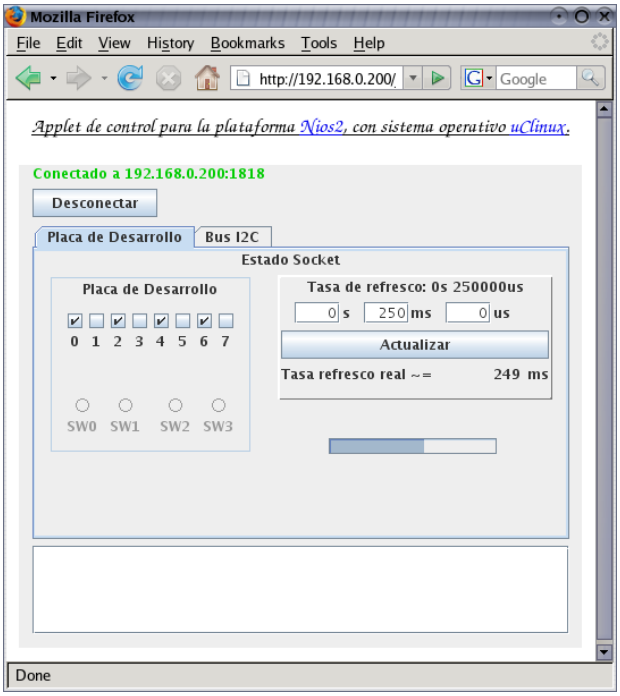


Ilustración 12: Primer ejemplo de manejo de la interfaz web



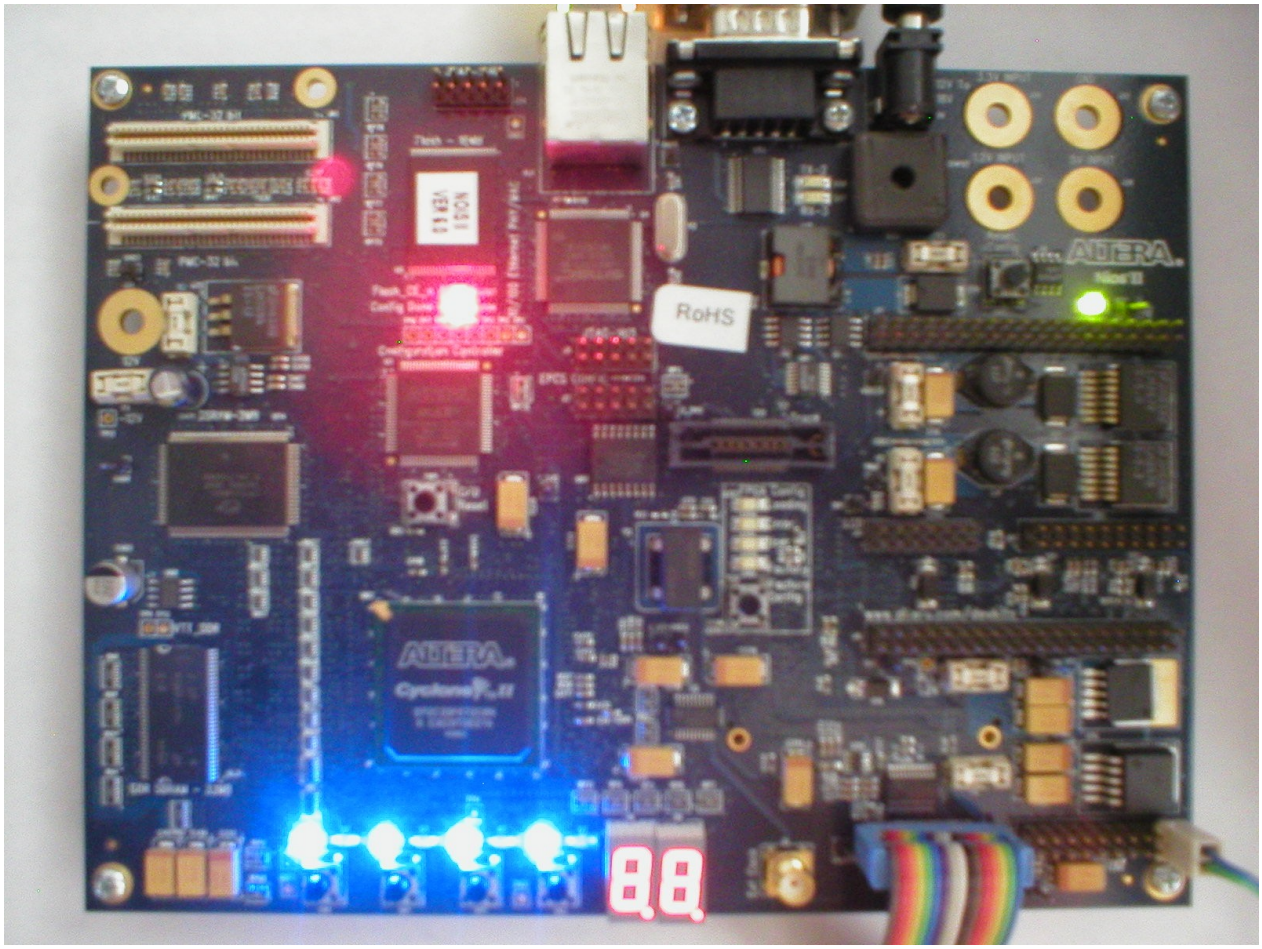
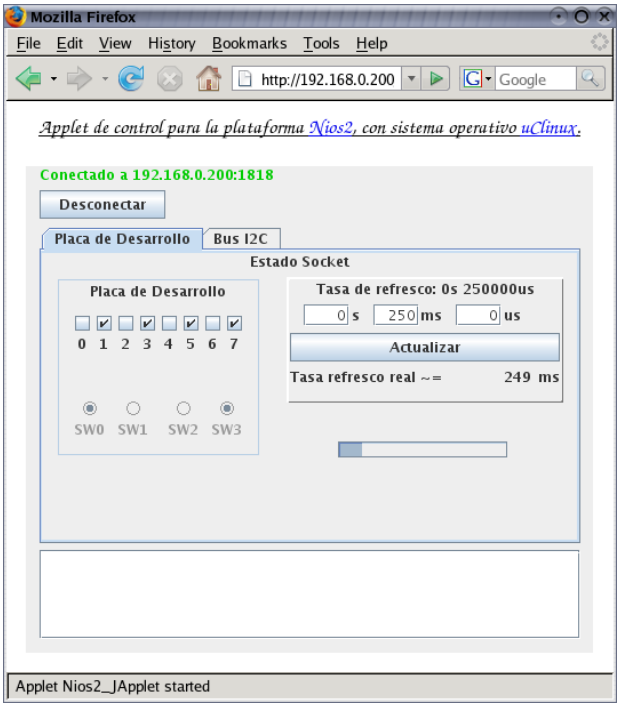


Ilustración 13: Segundo ejemplo de la interfaz web, con leds pares encendidos



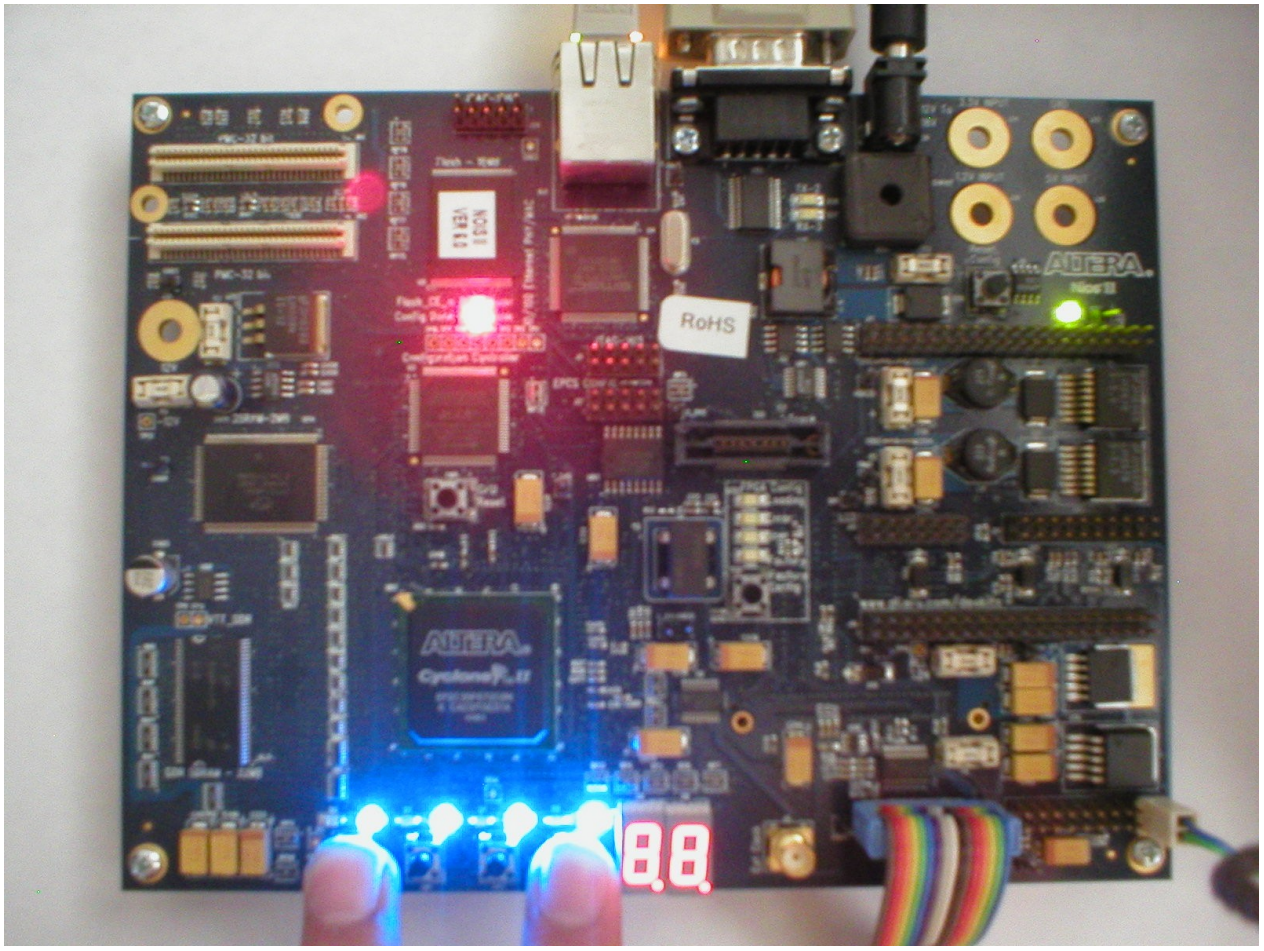


Ilustración 14: Tercer ejemplo, leds impares encendidos, y botones activados

La segunda categoría esta encargada de escribir y leer en la memoria I2C en la ilustración 15, donde se puede ver el estado de este modulo de la FPGA, así como también escribir y obtener datos desde esta memoria, aplicándose las limitaciones de rangos de direcciones y número de datos en las solicitudes efectuadas desde el Applet. Para simplificar la forma de mostrar la información leída y escrita en la memoria, se opta por enviar y desplegar la información en campos de texto.

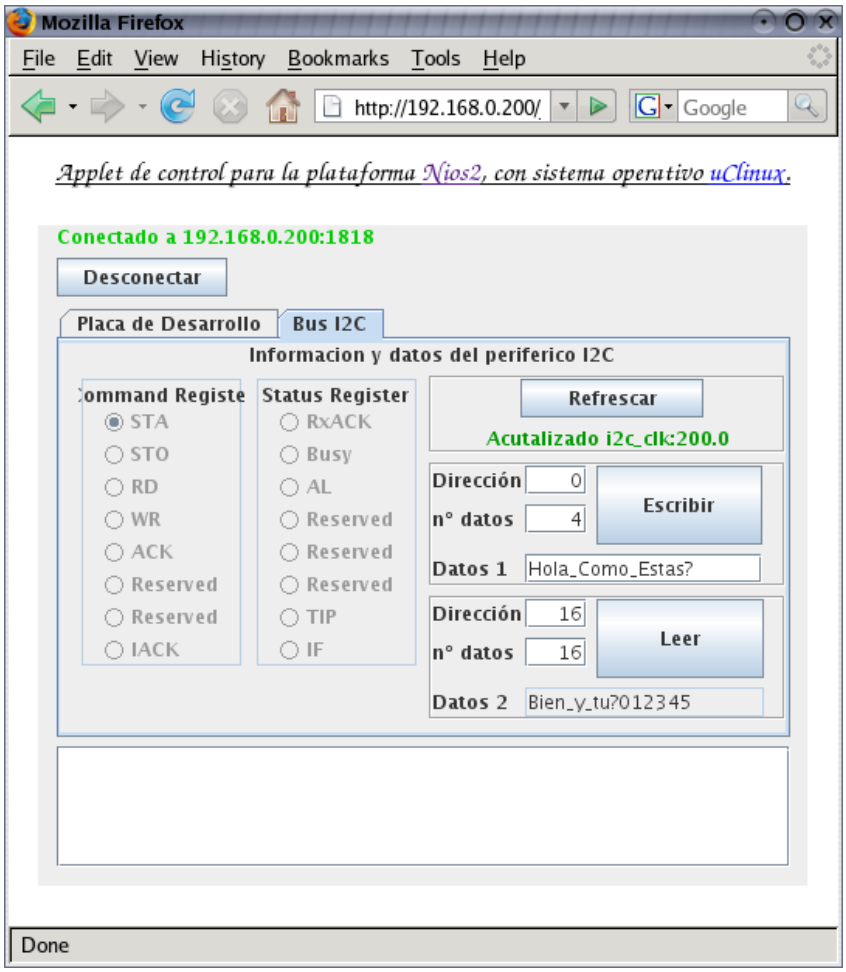


Ilustración 15: Escritura y lectura de memoria I2C

Con estas funciones y su respectiva interfaz gráfica, se logra finalmente obtener una base que cumple con los requerimientos iniciales de la memoria, de poder obtener una interfaz gráfica que se pueda desplegar en una pagina Web, integrada en el sistema, capaz de mostrar información dinámica del sistema que se esta supervisando, el cual en este caso quedo reducido a botones, leds y una memoria I2C, teniéndose libertad en el código de luego realizar el control especifico al el o los equipos nuevos que se integren a la empresa.

4 Resultados y Análisis

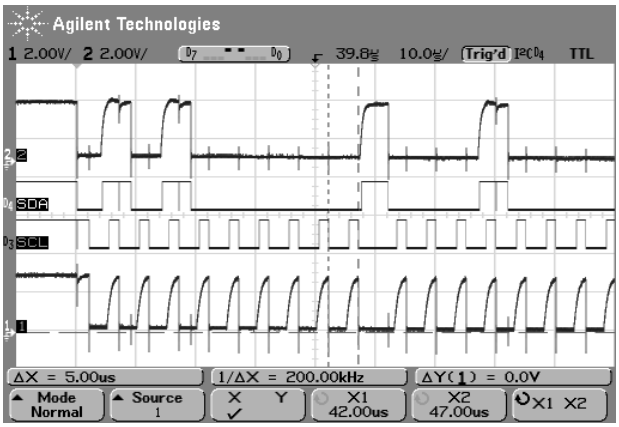
Del sistema realizado, se realizaron dos medidas principales para ver la utilidad de este sistema para las aplicaciones que se propusieron en esta memoria, estas son las velocidades de operación del bus I2C, y la velocidad de transferencia por Ethernet.

4.1 Operación del bus I2C

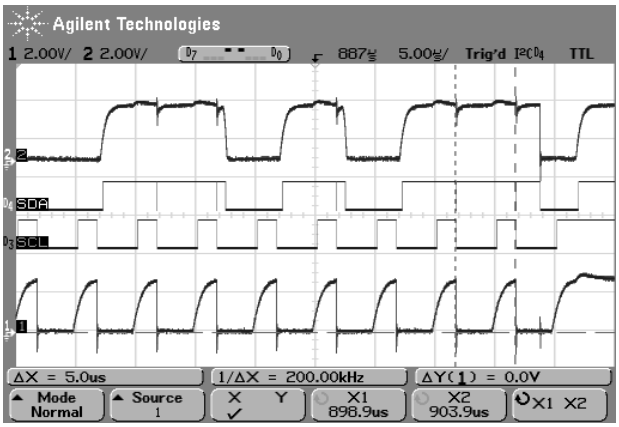
Una vez que se realizaron todos los pasos anteriores, se obtuvo finalmente un procesador operando a una velocidad de reloj de 85 MHz, con un sistema operativo uClinux ejecutando un servidor web, así como también un programa para comunicar datos a un Applet que muestra datos del sistema de forma remota, se da por finalizado todos los desarrollos propuestos para esta memoria.

A continuación se muestran algunas mediciones obtenidas de los puntos principales que son necesarios para esta aplicación, como son la tasa de transferencia que es capaz de alcanzarse con los programas diseñados en el bus I2C, así como también la velocidad máxima de transferencia que el sistema puede producir en la red y su tasa de recepción.

Para poder realizar las medidas del bus I2C, se ocupó el sistema realizado anteriormente, es decir en el procesador Nios II operando a 85 MHz estaban ejecutándose el sistema operativo uClinux, el servidor web BOA, y la aplicación encargada de enviar datos desde y hacia el Applet del usuario. En las ilustraciones 16 y 17 se muestran datos obtenidos desde un osciloscopio de la solicitud de lectura realizada en la ilustración 15.



Inicio de trama



Fin de trama

Ilustración 16: Inicio y fin de transmisión de datos por bus I2C con reloj 200 kHz.

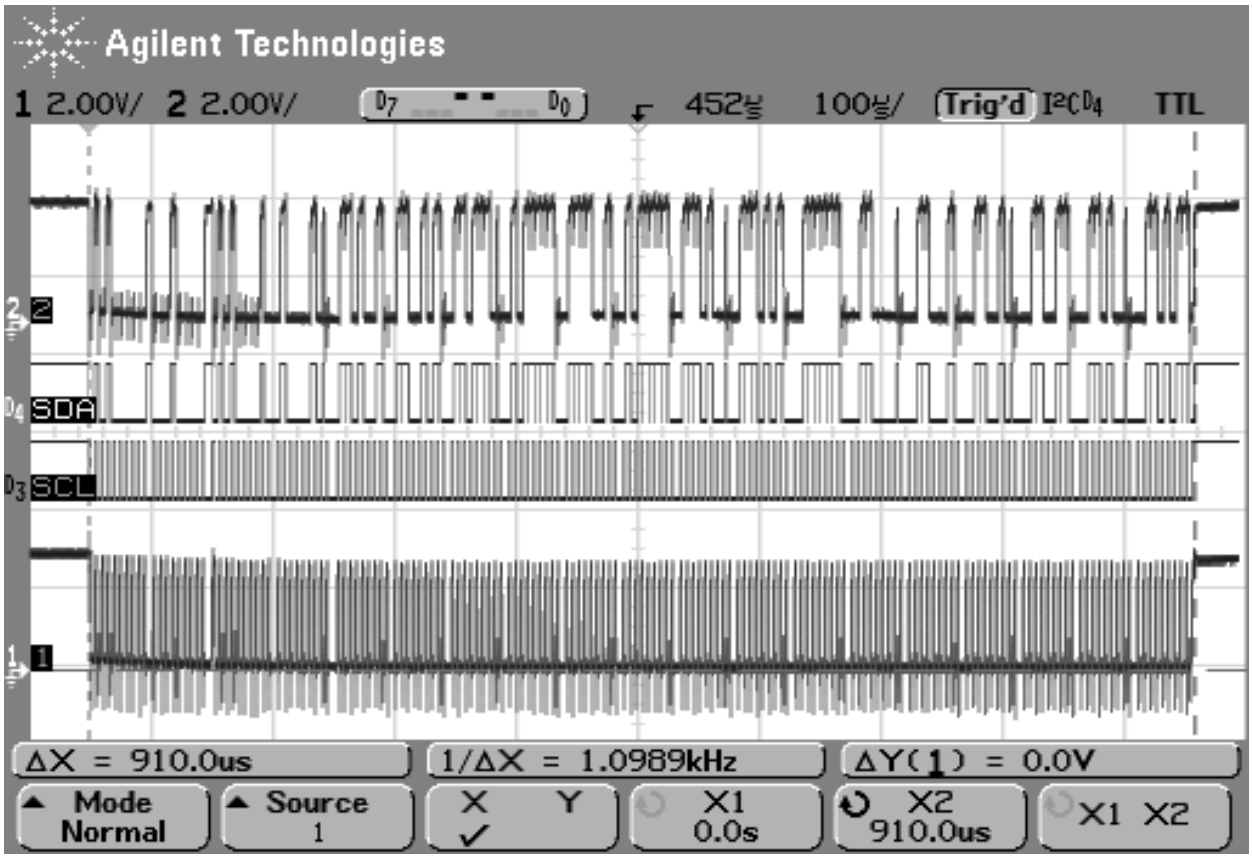


Ilustración 17: Trama de envío de 16 datos a la memoria por el bus I2C

Se observa que el reloj, el cual mediante los registros fue configurado a 200kHz, funciona a esta velocidad sin problemas. Además en la ilustración 17 se ve la lectura de 16 datos desde la memoria I2C. Como se mostró en la ilustración 10, para la lectura de página desde la memoria I2C, es necesario enviar 3 bytes extras, dos identificando el dispositivo, uno para escritura y otro para lectura, así como un dato de la dirección de la cual se quiere leer. En este caso, el envío de los 19 bytes totales enviados por el periférico I2C, toma aproximadamente 910 us, con lo cual se tiene una tasa de transferencia de aproximadamente 20879 bytes por segundo (20.38KBytes/s).

La tasa de lectura es equivalente a esta, con la diferencia que se necesita un byte menos de datos, pues solo se solicita lectura una vez en la memoria.

El reloj de este bus, puede ser aumentado hasta 400kHz, el cual es el máximo soportado por la memoria utilizada, y en general por este bus. Sin embargo, en el laboratorio no se logró alcanzar esta velocidad del periférico, principalmente debido a que al no esta la memoria y la FPGA en la misma tarjeta, se necesitan cables muy largos para unirlos, lo que sumado a la baja resistencia de pull-up disponible cerca de la FPGA, produjeron problemas en los tiempos necesarios para configurar el reloj.

Sin embargo, según conversaciones con la empresa interesada, el reloj de 100kHz le es de mayor utilidad para evitar el envío de datos erróneos en el ambiente de trabajo donde se utilizará el equipo final.

4.2 Mediciones de velocidad de transferencia por Ethernet

Una segunda medición fué la realización de programas cliente y servidor para Linux y uClinux, capaces de medir la tasa de trasferencia máxima del sistema. Para lograr esto, el cliente envía datos por un socket indefinidamente aumentando los primeros 4 bytes en forma de entero, esto último para verificar si hay pérdida de paquetes. El servidor recibe datos durante un tiempo fijo (5 segundos) contando la cantidad de bytes que son recibidas del socket, verifica si no existe alguna perdida de paquetes y de ser así realiza una división entre los datos recibidos y el tiempo entre datos, teniendo de esta manera un calculo de la tasa de transferencia.

El programa cliente además tiene la opción de enviar paquetes de tamaño variable, de manera de poder observar como varía el ancho de banda dependiendo de la cantidad de datos que se envíen por el socket. Recordemos que en el diseño utilizado, todo el trabajo de la capa MAC es realizado mediante un hardware dedicado, el cual es un chip externo cuyo código es LAN91C111, el cual es manejado directamente por un controlador de uClinux.

Mencionado esto, a continuación se muestran algunas de las tasas de transferencias obtenidas desde la plataforma de desarrollo, utilizando el procesador embebido Nios II en la configuración mencionada en el punto 4.3. El sistema estaba ejecutando todos los procesos relativos a uClinux, y además el programa encargado de conexión con el Applet estaba ejecutándose esperando conexiones, pero sin atender a ningún cliente.

Protocolo de comunicación	Tamaño de cada write	Tasa de transferencia.
TCP	4 bytes	9 kBytes/s
TCP	400 bytes	680 kBytes/s
TCP	1448 bytes (no fragmenta)	1770 kBytes/s
TCP	2896 bytes (2 fragmentos)	1920 kBytes/s
TCP	2847 bytes	1579 kBytes/s
TCP	14480 bytes	2020 kBytes/s (15.7 Mb/s)

Como ejemplo en un PC con Linux normal (es decir no embebido), comunicándose con un servidor en Widows (con los mismo archivos fuentes), se obtiene una tasa de transferencia para 4 bytes de 4900 kBytes/s, y de paquetes de 1448 hacia arriba ya se utilizan del orden 11000 kBytes/s, siendo el procesador Nios II, con uClinux al menos un 10% mas lento en transmitir que un PC de 3.5 GHz, con Linux usando la distribución Centos 4,5, usando una tarjeta de red de 10/100 Mb/s y con la misma topología de red que las pruebas anteriores.

Protocolo de comunicación	Tamaño de cada write	Tasa de transferencia.
UDP	4 bytes (perdidas de paquetes)	10 kBytes/s
UDP	400 bytes	831 kBytes/s
UDP	1472 bytes (no fragmenta)	1770 kBytes/s
UDP	1473 bytes	1472 kBytes
UDP	2896 bytes	1970 kBytes/s
UDP	14480 bytes	2480 kBytes/s (19.4 Mb/s)

Nuevamente las pruebas en la topología de un PC con Linux comunicándose a un servidor ejecutándose en Windows se obtiene una tasa de transferencia de 167.7 kBytes/s, con 1472 byte una tasa de 11600 kBytes utilizando casi toda la red de 100 Mb/s.

Se observa generalmente una mayor velocidad en las conexiones UDP en el sistema, en especial cuando es mayor el numero de datos con el que se escribe en el socket. Además se observa que cuando se envían un poco mas de datos que el tamaño máximo de un fragmento TCP o UDP, la tasa de transferencia baja principalmente en el caso del protocolo TCP , en TCP se logran las máximas transferencias enviando múltiplos del tamaño máximo, y en UDP es menos influyente este paramento.

Independiente de este punto del tamaño de datos que se envían en una escritura por el socket, se observa que el ancho de banda del sistema utiliza como máximo un cuarto del enlace de 100Mbit/s del cual dispone este chip. Se realizaron pruebas aumentando toda la velocidad del sistema en 20/17 (PLL produce un reloj de 100MHz), con lo cual se obtuvo una mejora en la misma proporción de las velocidades antes descritas, pero el desarrollo de mas pruebas para aumentar las tasas de transferencia quedan fuera del alcance de esta memoria.

4.3 Desarrollos futuros

- Un paso importante es la integración de nuevos periféricos al sistema, en particular módulos que sean de libre distribución para poder aprovechar aún mas el hardware que se tiene disponible sin aumentar el valor de la plataforma, para esto es necesario estudiar con detalle la interfaz entre el bus Avalon de Altera, y el bus Wishbone que generalmente es el utilizado en los periféricos libres y que depende del periférico que se quiera conectar.

- Pruebas del sistema implementado en la nueva plataforma que se está desarrollando a la fecha de redacción de esta memoria en la empresa, de manera de poder lograr tener finalmente el dispositivo de bajo costo funcionando en la empresa con todas las ventajas que esta trae.

- Se necesita poder realizar la conexión al sistema de un periférico que permita alcanzar una velocidad de enlace de 1 Gbps, para esto será necesario buscar una implementación que le sirva a los propósitos de la empresa y que logre funcionar en la FPGA de bajo costo utilizada.

- En caso de que se decida que la interfaz 10/100 Mbps es suficiente para la empresa, sería útil el uso de un núcleo que realice el protocolo MAC utilizando recursos de la FPGA, definir la cantidad de estos y verificar si resulta mas económico utilizar un dispositivo que tenga esta lógica interna en la FPGA, o el uso de un dispositivo externo que haga este protocolo como se utilizó en la memoria y se utilizó en el diseño de la nueva placa de la empresa

- Un aumento de la velocidad alcanzable con el sistema en Ethernet, como se detalló en la memoria con el sistema implementado no es posible llegar a ocupar los 100 Mbps del enlace, es necesario entonces buscar donde se produce el cuello de botella del sistema y buscar estrategias para acelerar la tasa de transferencia, lo que seguramente implicará cambiar el controlador de la interfaz Ethernet.

5 Conclusiones

El principal logro de esta memoria, es el de integrar una tecnología nueva en la empresa, entregando todo el conocimiento sobre la instalación ejecución y configuración de las distintas aplicaciones que se utilizaron para este desarrollo, siendo la solución de un relativo bajo costo.

La plataforma que se busco diseñar, disponía de la gran restricción del precio de esta, el cual fue el principal factor al descartar soluciones especialmente diseñadas con la velocidad de acceso que se proponía alcanzar con este trabajo. Al elegir una solución utilizando la tecnología de procesadores embebidos, se obtuvo una gran ganancia en flexibilidad para los desarrollos de la empresa de poder diseñar o utilizar periféricos específicos para las aplicaciones de la empresa .

Se utilizó para todo el desarrollo de este trabajo, aplicaciones que son libres de tener que paga royalties, como el procesador embebido Nios II, y se integró un sistema operativo Open Source basado en Linux, lo cual entrega un sistema operativo, que no es demasiado distinto al utilizado en los procesadores normales y que dispone de varios sitios que ofrecen respaldo para esta plataforma.

Al utilizar Applets como interfaz con el usuario y crear este mismo con una herramienta gráfica como lo es la utilizada. Se logra además, la posibilidad de que en la misma empresa puedan afinarse los detalles de la interfaz con el humano sin tener que tener que codificar esta interfaz sino que ese diseño puede ser realizado sin tener necesariamente conocimientos de programación y con la gran ventaja de ser multiplataforma, pudiendo realizarse el diseño tanto en Linux como en Windows, u otro sistema operativo donde este la plataforma de programación Netbeans.

Se desarrolla un sistema, trabajando en una plataforma de desarrollo, capaz de controlar un bus I2C, de modo multimaestro y enviar datos dinámicos del estado de la placa de desarrollo utilizada para ser mostrados en una interfaz web, utilizando Applets. La velocidad del periférico que realiza la transferencia por el bus .

6 Bibliografía y referencias

Altera Corporation (Febrero 2007), Cyclone II Device Handbook, Volume 1 CII5V1-3.2 http://www.altera.com/

Altera Corporation (Mayo 2007), Nios II Device Handbook NII5V1-7.1 http://www.altera.com/

Altera Corporation (Octubre 2007), Quaruts II Version 7.2 Handbook, Volume 4: SOPC Builder QII5V4-7.2 http://www.altera.com/

Richard Herveille rherveille@opnecores.org (Julio 2003), I2C-Master Core Specification Rev 0.9 http://www.opencores.org

Wikipedia (2007) la enciclopedia libre http://es.wikipedia.org/

Nios Forum, Nios Community Forum http://www.niosforum.com/

JotSpot Wiki (nios wiki) uClinux, http://nioswiki.jot.com/%C2%B5Clinux.

FPS-Tech uClinux 2.6 Distribution Overview http://www.fps-tech.net

Stephen Brown and Jonathan Rose (1996), "Architecture of FPGAs and CPLDs: A Tutorial" Department of Electrical and Computer Engineering , University of Toronto.