ESCUELA DE ELECTRÓNICA Y TELECOMUNICACIONES

DISEÑO E IMPLEMENTACIÓN DE UN SERVIDOR WEB EMBEBIDO BASADO EN UNA ARQUITECTURA RECONFIGURABLE CON FPGAS PARA EL CONTROL Y MONITOREO DE PERIFÉRICOS.

AUTOR: Christian Emanuel Lojan Herrera

Proyecto de Tesis previa obtención del Título de Ingeniero en Electrónica y Telecomunicaciones

DIRECTOR: Ing. Manuel Quiñones Cuenca.

Loja - Ecuador

Enero 2012

I. CERTIFICACIÓN

Ing. Manuel Quiñones Cuenca.
Docente Investigador de la Escuela de Ingeniería en Electrónica y Telecomunicaciones de la Universidad Técnica Particular de Loja.

Certifica:

Que una vez concluido el trabajo de investigación con el tema "**DISEÑO E IMPLEMENTACIÓN DE UN SERVIDOR WEB EMBEBIDO BASADO EN UNA ARQUITECTURA RECONFIGURABLE CON FPGAS PARA EL CONTROL Y MONITOREO DE PERIFÉRICOS**", previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones, realizado por el señor Christian Emanuel Loján Herrera, egresado de la Escuela de Ingeniería en Electrónica y Telecomunicaciones; haber dirigido, supervisado y asesorado en forma detenida cada uno de los aspectos de la tesis de pregrado.

Además, en mi calidad de DIRECTOR DE TESIS y al encontrar que se han cumplido con todos los requisitos investigativos, autorizo su presentación y sustentación ante el tribunal que se designe para el efecto.

Atentamente

Ing. Manuel Quiñones Cuenca.
DIRECTOR DE TESIS

II. CESIÓN DE DERECHOS.

Yo, **Christian Emanuel Loján Herrera**, declaro ser autor del presente trabajo y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales.

Adicionalmente declaro conocer y aceptar la disposición del Art.67 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que su parte pertinente textualmente dice: **"Forman parte del patrimonio de la Universidad la propiedad intelectual de investigadores, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad"**

Christian Emanuel Loján Herrera.

III. AUTORÍA

Las ideas, opiniones, conclusiones y contenidos expuestos en el presente informe de investigación son de exclusiva responsabilidad de su autor.

Christian Emanuel Loján Herrera.

IV. DEDICATORIA

A Dios, a mis; padres, hermanos, maestros y amigos.

V. AGRADECIMIENTOS

A todas las personas que de una u otra manera me han ayudado a culminar este proyecto de tesis. Es oportuno reconocer el esfuerzo, la paciencia y el apoyo de mi director de tesis Ing. Manuel Quiñones y de los docentes de la Universidad Técnica Particular de Loja.

VI. GLOSARIO DE TÉRMINOS

ACK: Acknowledgement (*Acuse de recibo*).

ADC: Analog to digital convert (*Conversor de señales analógicas a digitales*).

ALU: Arithmetic logic unit (*Unidad aritmética lógica*).

API: Application programming interface (*Interfaz de programación de aplicaciones*).

ASIC: Application specific integrate circuit (*Circuito integrado para aplicaciones específicas*).

CPLD: Complex programmable logic device (*Dispositivo lógico complejo programable*).

CSMA/CD: Carrier sense multiple access with collision detection (*Acceso múltiple por detección de portadora con detección de colisiones*).

CSS: Cascading style sheets (*Hojas de estilo en cascada*).

DCE: Data communication equipment (*Equipo de comunicación de datos*).

DDR: Double data rate (*Doble tasa de transferencia de datos*).

DNS: Domain name system / service (*Sistema/Servicio de nombre de dominio*).

DTE: Data terminal equipment (*Equipo terminal de datos*).

E/S: Acrónimo de dispositivos de Entrada/Salida.

EDK: Embedded development kit (*Kit de desarrollo embebido*).

FPGA: Field programmble gate array (*Arreglo de compuertas lógicas programables*).

GET: Método de acceso a recursos HTML.

HDL: Hadware description language (*Lenguaje de descripción de hardware*).

HTML: Hyper text markup language (*Lenguaje de marcas de hipertexto*).

HTTP: Hypertext transfer protocol (*Protocolo de transferencia de hipertexto*).

IP: Internet protocol (*Protocolo de Internet*).

IPC: Interprocess communication (*Comunicación interproceso*).

IPCORE: Intellectual property Core (*Bloque de lógica programable utilizado para conformar un sistema en un FPGA*).

ISE: Integrated software environment (*Ambiente integrado de software*).

LTU: LookUp tables (*Tablas de consulta*).

LWIP: Lighweight IP (*Pila IP de bajo peso*).

MAC: Media access controller (*Controlador de acceso al medio*).

MFS: Memory file system (*Sistema de archivos en memoria*).

MPMC: Multiport memory controller (*Controlador multipuerto de memoria*).

PCB: Protocol control block (*Bloque de control de protocolo*).

PLB: Processor local bus (*Bus local del procesador*).

POST: Método de acceso a recursos HTML.

RFC: Request for comments (*Solicitud de comentarios*).

RISC: Reduced instruction set computer (*Computador de conjunto de instrucciones reducidas*).

SDK: Software development kit (*Conjunto de software de desarrollo*)

SDRAM: Synchronous dynamic random access memory (*Memoria de acceso aleatorio dinámico síncrono*).

SoC: System on chip (*Sistema sobre un chip*).

SWE: Servidor web embebido.

TCP: Transmission control protocol (*Protocolo de control de transmisión*).

TFTP: Trivial file transfer protocol (*Protocolo de transferencia de archivos triviales*).

TTL: Time to live (*Tiempo de vida*).

UCF: User constraints file (*Archivo de restricciones de usuario*).

URL: Uniform resource locator (*Localizador uniforme de recursos*).

XMD: Xilinx microprocessor debugger (*Depurador de microprocesador de Xilinx*).

XML: Extensive markup language (*Lenguaje de marcado extensible*).

VII. CONTENIDO

VIII. LISTA DE FIGURAS

IX. ÍNDICE DE TABLAS

X. RESUMEN

El presente trabajo trata del diseño e implementación de un servidor web embebido en una tarjeta Spartan 3E Starter Kit de Digilent con el propósito de monitoreo y control de periféricos. El sistema fue diseñado con el método SoC que permite incluir módulos de propiedad intelectual (IPCORES), a manera de coprocesadores, con funcionalidades como: procesadores, periféricos, memorias y protocolos.

La plataforma de hardware y software fue diseñada y desarrollada en ISE 11.1 de Xilinx, y está gestionada por el procesador Firm Core Microblaze para proveer: un servicio de transporte basado en la pila TCP/IP de bajo peso LWIP, transporte HTTP y funcionalidades de testeo de red. Además posee un módulo ADC externo basado en el microcontrolador ATMEGA32.

Se ha desarrollado una interfaz gráfica basada en: HTML, CSS y JS, con características dinámicas, la misma que mediante métodos y objetos HMTL, DOM y JS, permite modificar y acceder a recursos específicos del sistema. Por último se ha realizado pruebas de estabilidad, conectividad, procesos de carga, ancho de banda y eficiencia energética del sistema.

XI. INTRODUCCIÓN.

Por ahora los sistemas embebidos en diversos países han inundado los campos de: la industrial, la medicina, militar, aeroespacial, transporte, comunicaciones, etc. Debido principalmente a las funcionalidades que presentan estos sistemas, su cantidad ha ido aumentando con el paso del tiempo y lo seguirá haciendo en los próximos años. Debido a esta tendencia, ha surgido la preocupación e interés por estos sistemas modernos, tanto en países desarrollados como emergentes, esto se traduce en destinar mayores recursos económicos para: el estudio de estas tecnologías y capacitación de personal que manejará las mismas.

Los FPGAs, son uno de los dispositivos más empleados para la elaboración de sistemas embebidos flexibles y de bajo costo. La compañía Xilinx es una de las más fuertes en el campo de los Chips para FPGAs y ASICs, además posee un soporte muy bueno para desarrolladores.

Tomando en consideración esto, surge la motivación de este proyecto de fin de carrera, cuyo objetivo principal es el desarrollo de un sistema embebido basado en un FPGA de Xilinx, que proveerá una interfaz gráfica HTML capaz de controlar o supervisar periféricos del sistema embebido. El servidor brindará un servicio de envío de datos de hipertexto (HTTP), accesible a los clientes que soliciten acceso a un recurso físico (periféricos) o lógico (ficheros) del sistema embebido.

Un servidor web embebido es un sistema electrónico hecho a medida para alojar una página HTML, con propósitos de brindar un servicio WEB, posee una dirección IP asignada a la interfaz Ethernet, mediante la cual, se puede acceder al sistema con un navegador WEB tradicional. Un servidor web embebido está en la capacidad de realizar transacciones gracias a la implementación de una pila de protocolos de red Ethernet (TCP/IP), esta pila garantiza una comunicación confiable de datos necesarios para el funcionamiento de las aplicaciones.

El método de diseño de sistemas embebidos suele hacerse modular y con codiseño entre software y hardware, para lograr una completa integración. Esto con el objetivo de: agregar flexibilidad al sistema y de acortar el tiempo de diseño e implementación. Ambas partes en conjunto conformarán lo que se llama Firmware.

Todo sistema embebido cuenta supervisión de errores, el servidor web embebido lo hace mediante una interfaz RS232 DCE. Además, posee un módulo de adquisición de datos analógicos externo mediante el protocolo RS232. El sistema embebido fue implementado sobre una tarjeta Spartan 3E Starter Kit de Digilent, completamente compatible con el entorno de desarrollo de Xilinx.

Todas las características de: flexibilidad, bajo consumo y conectividad, hacen de este sistema embebido una opción muy viable para su adaptación en el hogar o en la industria. Este proyecto es un referente muy válido para la adaptación de sistemas embebidos con capacidades de conectividad a la WEB.

OBJETIVO GENERAL.

Diseñar e implementar un servidor web embebido en una tarjeta electrónica Spartan 3E Starter Kit, basado en el método modular System On Chip (SoC) utilizando módulos de la propiedad intelectual (IPCORES) de Xilinx, aplicando una metodología de codiseño entre hardware y software. Con propósitos de control y monitoreo de variables analógicas y digitales.

OBJETIVOS ESPECÍFICOS.

Los objetivos específicos que se plantean en el desarrollo del presente trabajo de tesis son:

- Diseñar e implementar una plataforma de hardware con un alto nivel de síntesis lógica basada en el método de codiseño SoC con IPCORES Xilinx sobre una tarjeta electrónica comercial basada en FPGAs, que proporcione un uso óptimo de recursos: físicos, lógicos y energéticos.
- Diseñar y construir un módulo externo de hardware que permita un monitoreo de señales analógicas.
- Diseñar una plataforma de software capaz de gestionar una plataforma de hardware específica, con el propósito de dotar al sistema de protocolos de comunicación para servicios de internet configurables para aplicaciones de monitoreo de señales analógicas y digitales.
- Implementar una plataforma hardware/software que proporcione soporte para funcionalidades HTTP, de tal manera que, el sistema sea capaz de proporcionar un servicio WEB y funcionalidades de diagnóstico de red.
- Diseñar una interfaz gráfica basada en HTML de fácil uso, capaz de controlar periféricos del sistema y de visualizar el estado de los mismos.

1. CAPITULO I: GENERALIDADES.

1.1. Estado del arte de los servidores web embebidos (SWEs).

Los servidores web embebidos (SWEs), son dispositivos construidos a medida para proveer un servicio de transporte de hipertexto (HTTP) permitiendo a un cliente visualizar páginas HTML mediante un navegador WEB [1]. El dispositivo emplea un modelo clásico de cliente-servidor para el control y monitoreo de dispositivos, poniendo a disposición del cliente una poderosa herramienta de control y supervisión de procesos en la industria o en el hogar.

El mercado de los dispositivos electrónicos está creciendo vertiginosamente, principalmente, debido; a su bajo costo y sus múltiples campos de aplicación. Los sistemas electrónicos se encuentran en casi todos los aspectos de nuestras vidas, desde un teléfono celular, hasta el más moderno sistema de cómputo. Muchos de los sistemas embebidos suelen construirse con el método de diseño denominado System on Chip (SoC), que consiste en incluir núcleos de propiedad intelectual (IPCORES) dentro del propio chip, la inclusión de los IPCORES acorta el tiempo de diseño de los sistemas y los vuelve flexibles gracias al método modular de inclusión de funcionalidades y protocolos. El tipo de IPCORES que se suelen incluir en un Chip son: procesadores, periféricos, memorias, protocolos de comunicación, algoritmos de codificación, entre otros [2].

Los sistemas embebidos son desarrollados sobre dispositivos electrónicos conocidos como Chips, tales como; microprocesadores, microcontroladores, ASICs, FPGAs o DSPs, el uso de cada uno de estos dispositivos depende en gran medida de los requerimientos de la aplicación a la que se destinará el sistema embebido:

- El microprocesador es comúnmente utilizado en sistemas de cómputo de propósito general, contiene; registros, unidad de control, unidad aritmética y lógica (ALU), unidad de interfaz de bus. El microprocesador en un sistema embebido que obligatoriamente debe ir acompañado de: unidades de almacenamiento, buses y/o periféricos, ya que, es únicamente una unidad

1 W. Nicholas. *Designing* an Embedded Web Server (2000). U.S.A: Applied Computing Technologies. Recuperado el 2011 de http://pdf.cloud.opensystemsmedia.com/embedded-computing.com/USSoftware.Win00.pdf

2 A. Miguel. y J. Alvaro. *Servidor Web Embebido en una FPGA con Codiseño como Metodología de Diseño*. Eighth LACCEI Latin American and Caribbean Conference for Engineering and Technology (2010). "*Innovation and Development for the Americas*", Junio 1-4, 2010, Arequipa, Perú.

de procesamiento dedicada y generalmente resulta ser más rápido que un microcontrolador [3].

- Los microcontroladores, se han popularizado en aplicaciones de sistemas embebidos de propósito general, debido a que integran en un solo chip; unidades de memoria, unidades de procesamiento y periféricos [4]. En la actualidad los dispositivos de este tipo se han popularizado en sistemas embebidos de bajo costo.

- El mercado de los DSPs se concentra en el procesamiento digital de señales en tiempo real, tales como: filtros digitales, osciloscopios, analizadores de espectro, etc. [5]. Los dispositivos DSPs suelen ser apreciados por el paralelismo que aportan en sistemas embebidos destinados al cálculo matemático complejo.

- Los ASICs son circuitos integrados personalizados para una aplicación en particular, como un SWE de altas prestaciones. Los ASICs utilizan una descripción de hardware específica para su desarrollo (VHDL) [6]. Al ser hechos a medida son mucho más rápidos que los FPGAs.

- Los Field Programmable Gate Array (FPGAs), son circuitos integrados que contienen componentes de lógica programables (bloques lógicos), así como interconexiones programables que permiten realizar la función de los DSPs, además permiten desarrollar prototipos para ASICs, la principal característica es el elevado nivel de síntesis y de paralelismo [6].

El mercado de Chips para sistemas embebidos, está mayormente dominado por las empresas: Freescale, Altera, Atmel y Xilinx [7].

- Freescale, es una división de Motorola fundada en 2004, se encuentra en el ranking 16 de los mayores proveedores de dispositivos con ganancias de 3.4 billones de dólares en el 2010 [8]. Es fabricante de microcontroladores,

3 Master Magazine. *Definición de microprocesador* (2011). Recuperado el 2011 de http://www.mastermagazine.info/termino/5881.php

4 O. Juan y A. Vicente . *Arquitectura y programación de microcontroladores (2010)*. Valencia: Universidad de Valencia.

5 C. Rorabaugh. *DSP Primer* (1999). USA: McGraw Hill.

6 C. Maxfield. *The design warrior's guide to FPGAs.* (2004). USA: Mentor graphics Corporation y Xilinx Inc.

7 Nextinning.*Technology & Semiconductor investment analysis*. (2010). Recuperado el 2011 de http://www.nextinning.com.

8 Motorola Mobility Holdings. Inc.*Motorola Mobility Announces Fourth-Quarter and Full-Year 2010 Financial Results. (2010)*. Recuperado el 2011 de http://mediacenter.motorola.com/Press-Releases/Motorola-Mobility-Announces-Fourth-Quarter-and-Full-Year-2010-Financial-Results-359a.aspx

ASICs, DSPs, memorias, sensores, entre otros. Con aplicaciones orientadas al sector; automotriz, electrónica de consumo, industria, redes, conectividad, control de motores y comunicaciones inalámbricas. Los dispositivos de Freescale han sido integrados en productos como: BMW X5 (FlexRay MFR4200), Lavadoras y secadoras Whirlpool (microcontroladores MC908AX) [9]. La herramienta para el desarrollo de sistemas embebidos de Freescale es "*CodeWarrior Development Studio for Microcontrollers V6.0*", que incluye el soporte para las familias: RS08, S08 y Coldfire V1, posee herramientas optimizadas de compilación y vinculación 'linker', para cada una de las subfamilias [10].

- ALTERA, es uno de los primeros fabricantes de; FPGAs, ASICs y CPLDs, construye productos como: FPGAs Cyclone, Arria GX y Stratix, MAX CPLDs, ASICs HardCopy. Sus dispositivos se utilizan en aplicaciones; médicas, militares, inalámbricas, automotrices, entre otras. Es proveedor de procesadores embebidos; NIOS II, CooldFire y ARM, además de IPCORES. Para el 2010 reportó ganancias por 1.95 billones de dólares [11], sus dispositivos se encuentran en: televisores Sanyo (NIOS II y FPGA Stratix), cámaras de video Panasonic (FPGA Cyclone) y plataformas de aprendizaje Leapfrog (CPLD MAX II) [12]. El software de desarrollo para sistemas embebidos de Altera es "*Embedded Initiative*", que proporciona a los diseñadores un único flujo para el diseño de FPGAs, basado en Quartus® II: incluye herramientas de integración al nivel del sistema Qsys, una biblioteca común de propiedad intelectual (IP) y soporte para procesadores embebidos ARM®, Cortex™-A9, MPCore™ y MIPS® Technologies MIPS32 [12]. El software de diseño, permite que los diseñadores de sistemas embebidos trabajen con soluciones basadas en; Nios® II, ARM, MIPS e Intel® Atom™. La herramienta de integración a nivel del sistema Qsys aprovecha la primera tecnología de red en un chip optimizada para FPGAs, para ofrecer soporte a

9 Freescale. *Sales and Support. (2011*). Recuperado el 2011 de http://www.freescale.com/webapp/sps/site/homepage.jsp?code=SUPPORTHOME&tid=FSH

10 C. Luis. *Nueva familia de microcontroladores de 8 y 32 Bits FLEXIS de Freescale*. (2007). Recuperado el 2011 de http://www.bairesrobotics.com.ar/data/flexis.pdf .

11 P. John. Letter to Shareholders. (2010). Recuperado el 2011 de http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9NDE0NDExfENoaWxkSUQ9NDI2MDQ0fFR5cGU9MQ==&t=1.

12 Altera. *CPLD MAX* (2011). Recuperado el 2011 de http://www.altera.com/devices/cpld/max-about/max-about.html

una amplia variedad de protocolos IP estándar, QoS y numerosas funciones de productividad [13].

- ATMEL es fabricante de: microcontroladores, ASICs, FPGAs y memorias, con aplicación en el sector: de comunicación, militar, aeroespacial, industrial, entre otras. Provee dispositivos microcontroladores para aplicaciones comerciales como: reproductores MP3 (AT8xC51SND1C), manejo de "ring tones" MP3 para teléfonos celulares (AT8xC51SND2C), lectura de tarjetas MMC-SD, reproducción MP3 e interfaz Bus universal en serie (USB). En diciembre del 2009, reportó una ganancia de 1.19 billones de dólares [14]. La herramienta de desarrollo para dispositivos de ATMEL es el *"AVR Studio 5.0"*, soporta microcontroladores AVR de 8 y 32 bits, posee un ambiente de desarrollo y depuración basado en los lenguajes: C, C++ y ensamblador [15].

- Xilinx es la mayor empresa dedicada al desarrollo y fabricación de dispositivos FPGAs, desarrolla; FPGAs y CPLDs, que son usados en numerosas aplicaciones de: telecomunicaciones, automoción, productos de consumo, industria militar y otros campos. Las familias de dispositivos de Xilinx son: glue logic (CoolRunner y CoolRunner II), bajo coste (Spartan) y alto rendimiento (Virtex) [16]. Xilinx también crea núcleos IP (IPCORES) en lenguajes HDL para permitir a los diseñadores reducir los tiempos de desarrollo. Estos núcleos van desde funciones simples como: controladores de dispositivos a sistemas complejos como microprocesadores (Microblaze). En abril del 2011 Xilinx reporto ventas por 2.37 billones de dólares [17]. El ambiente para desarrolladores Xilinx se denomina ISE, que se describirá con mayor detalle en los capítulos siguientes.

---

13 P. Mark. Altera Launches Embedded Initiative with New System Level Integration Tool for Embedded Systems Configurability. 2010. En http://www.prnewswire.com/news-releases/altera-launches-embedded-initiative-with-new-system-level-integration-tool-for-embedded-systems-configurability-104763669.html

14 Aarkstore Enterprise. *Altera Corporation - SWOT Analysis - Market Research Report On Aarkstore Enterprise.* (2011). Recuperado el 2011 de http://www.aarkstore.com/reports/Altera-Corporation-SWOT-Analysis-26748.html.

15 Atmel corporation. *Atmel AVR Studio 5.0* (2011). Recuperado el 2011 de http://www.atmel.com/dyn/products/tools_card.asp?source=cms&tool_id=17212

16 Xilinx Inc. Products. (2011). Recuperado el 2011 de http://www.xilinx.com/products/index.htm

17 McG. Dylan. *Xilinx beats estimates on record annual sales.* (2011). Recuperado el 2011 de http://www.eetimes.com/electronics-news/4215542/Xilinx-beats-estimates-on-record-annual-sales

Un SWE se ejecuta en el nivel de aplicación de un sistema embebido, existen varias opciones de servidores web en el mercado, las que poseen requerimientos mínimos de: hardware y software, para poder ejecutarse en un sistema embebido, en la Tabla 1. 1 se mencionan algunos SW para sistemas embebidos y sus particularidades.

Tabla 1. 1. Servidores web embebidos.

Servidor	Creador	S.O.	Lenguaje de desarrollo	Licencia	Versión	F. liberación.
Boa	Paul Phillips	Unix	C, Perl	GPL 1	0.94.14rc21	23/02/05
Cherokee	Álvaro López Ortega	Windows, Mac OS X, GNU/Linux, Solaris, BSD	C	GPL	1.0.18	19/01/11
darkhttpd	Emil Mikulic	Unix	C	BSD	1.6.8	03/05/11
Hiawatha	Hugo Leisink	Linux, BSD, Mac OS X, Windows, Haiku os.	C	GPL 2	7.8.2	18/11/11
KLone	Koan Logic Srl	Mayoría de plataformas.	C	GPL	2.4.0	02/05/11
lighttpd	Jan Kneschke	Unix, Linux, Windows.	C	BSD	1.4.29	03/07/11
NanoHTTPD	Jarno Elonen	Java enabled, incluido mobile.	Java	BSD modificado	1.24	04/08/11
nginx	Igor Sysoev	Unix-like, Windows.	C	BSD	1.0.6	05/09/11
Perlbal	Brad Fitzpatrick	Todo SO con Perl	Perl	GPL / Artístico	1.73	13/10/09
thttpd	Acme Labs	Unix	C	BSD	2.25b	29/12/03
TJWS	Dmitriy Rogatkin	Java enabled, incluido mobile	Java	BSD	1.93	10/10/11
Tntnet	Tommi Mäkitalo	Unix	C++	LGPL	2.0	10/08/11
UIP	Adam Dunkels	Embedded 8bit	C	BSD	1.0	12/06/06

Existen soluciones comerciales de SWEs integrados, por ahora existen varias opciones basadas en diferentes sistemas operativos, en la Tabla 1. 2 se menciona algunas de ellas con sus principales características.

Tabla 1. 2. Servidores web embebidos comerciales.

Característica	XPort Embedded Device Server [18]	AVR460 [19].	Microchip PIC SBC65EC [20].
Precio	$60-$200	$900	$72-$100
Subida de datos.	TFTP, Serial	FTP, Serial	ISCP

18 Xport AR. *XPort AR Embedded ProcessorModule*. (2011). Recuperado el 2011 de http://www.lantronix.com/pdf/XPort-AR_PB.pdf.

19 Atmel. *AVR460: Embedded Web Server*. (2011). Recuperado el 2011 de http://www.atmel.com/dyn/resources/prod_documents/doc2396.pdf.

20 McPros. *Microchip PIC Ethernet Board w/ RS232 & Web-Based Configuration*.(2011). Recuperado el 2011 de http://microcontrollershop.com/product_info.php?cPath=98&products_id=893.

10/100 Mbps Ethernet	10 Mbps.	10 Mbps	10 Mbps
Telnet	Si	Si	No
Memoria.	256 SRAM, 512 Kb Flash.	32 Kb SRAM, Data Flash 2 Mb, 128K Programmable flash.	98KBytes FLASH, 3840 Bytes SRAM y expansión de EEPROM
Velocidad de Transmisión.	300 bps – 921 kbps.	4.608 Mbps.	500 bps – 700 kbps.
Web extras	Configuración basada en Windows	Soporte AVR.	Ejemplos Online.
Otras características	Auto IP, Encriptación AES de 256 BITS, actualizaciones de firmware.	DHCP, EMAIL.	DHCP, Compresión HTTP, NetBios, DNS, MXFS.

1.2. Importancia y aplicaciones de los SWEs.

En la actualidad el desarrollo de redes de internet permite implementar aplicaciones de control y monitoreo remoto, la posibilidad de tener un dispositivo de bajo costo con capacidades de: controlar o monitorear remotamente dispositivos electrónicos, y que además sea capaz de adaptarse a una red global, es uno de los factores que han aportado para el desarrollo de los SWEs; algunos de los proveedores más importantes de dispositivos de gestión de red como: Cisco [21] y D-link [22], incluyen interfaces web amigables con propósitos de: configuración o gestión de dispositivos de red, de manera local o remota.

La capacidad de supervisión y control a distancia que brindan los SWEs, ha dado lugar a muchas aplicaciones, tales como:

- Control de procesos productivos.
- Puntos de información electrónica.
- Sistemas de posicionamiento global.
- Sistemas de video vigilancia.
- Laboratorios remotos virtuales.
- Domótica.

21 Cisco Systems. *Cisco Router Web SetUp Tool*. (2011). Recuperado el 2011 de http://www.cisco.com/en/US/products/sw/netmgtsw/ps2076/index.html

22 Dlink. *D-link solutions*. (2011). Recuperado el 2011 de www.dlink.com.

Existen en el mercado soluciones integradas para el control y monitoreo de procesos, que de una u otra manera, utilizan la filosofía de un SWE, entre estas soluciones podemos mencionar:

- Cámara de red AXIS 211(Sistema de video vigilancia remoto) [23].
- Dd-7122 Cámara IP Motorizada Wifi Zoom Óptico X10 Dual Codec / Dual Stream (Video vigilancia remota) [24].
- Dd-6310 Ip Motor Kit Control Por Internet (Control físico de dispositivos) [25].
- Siteplayer microcontrolador con servidor Telnet s310268 (Control de dispositivos con interfaz serial vía web) [26].

1.3. Sistemas embebidos System on Chip (SoC).

Se denominan sistemas embebidos SoC a los sistemas embebidos que son construidos mediante el método de diseño modular, ésta técnica permite incluir módulos de propiedad intelectual (IPCORES) dentro de un chip FPGA, con el objetivo de reducir tiempo de diseño e implementación y costos. Los IPCORES que se añaden a un Chip, ocupan recursos del mismo y pueden ser: protocolos de comunicación, procesadores, memorias o periféricos. Un IPCORE añade una alta capacidad de reconfiguración a cualquier sistema embebido.

Los IPCORES utilizados como procesadores suelen ser: Hard Cores, Firm Cores o *Soft Cores*. Los Firm Cores están definidos como una mezcla de código fuente y netlist dependientes de la tecnología utilizada. En este tipo de procesadores, el código fuente es visible para el diseñador y algunas partes podrán ser modificadas por él. El netlist está determinado por la tecnología, y por tanto, el usuario se verá obligado a utilizar estos procesadores sobre los Chips del mismo fabricante [27]. El tipo de Firm Core utilizado en el SWE es Microblaze de Xilinx.

En la Tabla 1. 3, se puede apreciar los tipos Cores y sus particularidades**.**

23 Axis Communications. *Camaras de red Axis 211*. (2011). Recuperado el 2011 de http://www.axis.com/es/products/cam_211/accessories.htm

24 Domo Desk. *Dd-7122 Camara Ip Motorizada Wifi Zoom Optico X10 Dual Codec / Dual Stream*.(2011). Recuperado el 2011 de http://www.domodesk.com/product/450/14/17/1/CAMARA_IP_MOTORIZADA_WiFi_ZOOM_OPTICO_x10_DUAL_CODEC_DUAL_STREAM.htm

25 Domo Desk. *Dd-6310 Ip Motor Kit Control Por Internet*. (2011). Recuperado el 2011 de http://www.domodesk.com/product/20/14/37/1/KIT_CONTROL_IP_MOTOR_por_INTERNET.htm

26 Superrobotica. *Siteplayer Microcontrolador con Servidor Telnet S310268*. (2011). Recuperado el 2011 de http://www.superrobotica.com/S310268.htm

27 C. Jerry, G. Nupur, M. Jayant y R. David. *Design Methodologies for Core- Based FPGA Designs*. (2011). Recuperado el 2011 de www.xilinx.com/.

Tabla 1. 3. Tipos de Processor Cores. Referencia: [27].

	Hard Cores	Firm Cores	Soft Cores
Dureza	*Layout* predefinido.	Mezcla de código fuente y tecnología dependiente de la *netlist.*	Dependiente del comportamiento del código.
Modelado	Modelado como librería de elementos.	Mezcla de bloques fijos y sintetizables que pueden ser compartidos por otros *Cores.*	Sintetizable con otra lógica.
Flexibilidad	No puede ser modificado por el diseñador. La utilización de varios hard Core en un chip puede resultar ineficiente.	Tecnología dependiente.	El diseño puede variarse.
Predictibilidad	Garantiza los *timing.*	Camino crítico es fijo.	El *timing* no está garantizado.
Coste	Bajo.	Medio.	Alto.
Descripción	Ficheros *layout* y *timing information.*	Código sintetizable HDL y ficheros *layout* y *timing information.*	Código sintetizable HDL.

1.4. Codiseño de sistemas embebidos SoC.

El término codiseño hardware/software es usado para denotar la interacción entre los flujos de diseño entre: hardware y software, en un sistema embebido. El principal objetivo de usar técnicas de codiseño hardware/software, es dar la posibilidad real al diseñador de decidir cuáles funciones se ejecutan en hardware y cuáles en software. El flujo tradicional de codiseño hardware/software, se describe en la Fig. 1. 1.

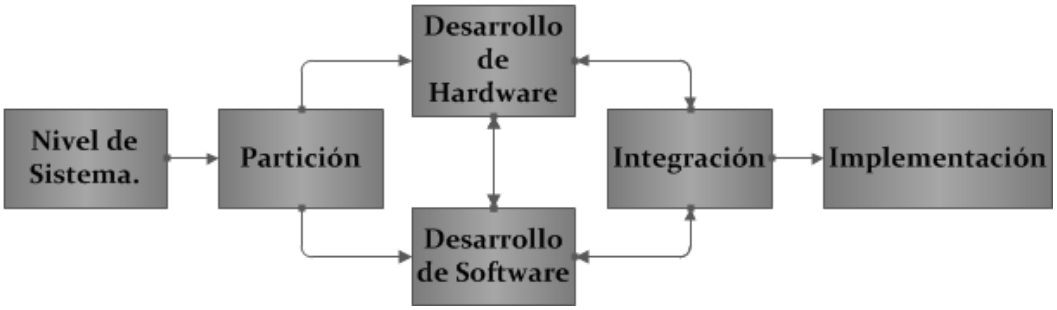


Fig. 1. 1. Flujo de codiseño. Referencia: [2].

Cada una de las estas etapas esquematizadas anteriormente son: paralelas y complementarias, a continuación se detallan tal como se describen la referencia [2]:

- En el *nivel del sistema* se definen las especificaciones funcionales del sistema y los parámetros de desempeño.

- *En la partición* de diseño se deben definir: qué parte se implementara en hardware y qué parte en software.

- El *desarrollo del software y desarrollo de hardware*, son actividades paralelas y complementarias referentes al diseño global del sistema.

- La *integración,* es una actividad con un alto nivel de reconfiguración del software y del hardware, que garantiza una completa integración.

- La *implementación*, es una fase de pruebas sobre un sistema físico en condiciones reales de operación.

1.5. Servidor web embebido (SWE).

La administración y gestión de dispositivos electrónicos por ahora es una tarea fácil gracias a la implementación de interfaces graficas WEB sobre un SWE, el mismo provee una interfaz gráfica enriquecida en HTML con variedad de: imágenes, fuentes de texto, colores, etc., que ponen a disposición del usuario una herramienta flexible para la administración de un dispositivo [28]. El usuario está en la capacidad de enviar instrucciones o monitorear el estado de algún dispositivo, desde una estación de trabajo: local o remota. Un SWE agrega a cualquier sistema ciertas particularidades como: bajo costo de desarrollo, un alto nivel de reconfiguración y una interfaz amigable. Un escenario típico de aplicación de un SWE puede observarse en la Fig. 1. 2.

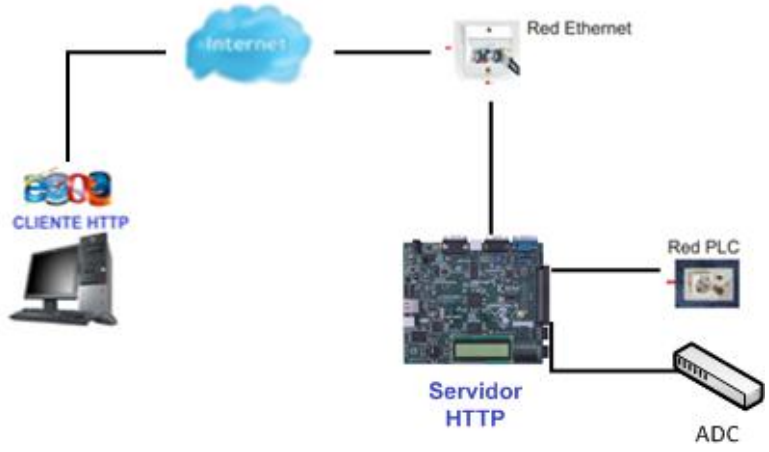


Fig. 1. 2. Aplicación típica de un SWE.

28 H. Ju, M. Choi y J. Hong. *EWS-Based Management Application Interface and Integration Mechanisms for Web-Based Element Management.* (2001). Journal of Network and Systems Management, Vol. 9, No. 1, pp. 31-50.

Para la implementación de cualquier SWE es necesario: una interfaz Ethernet y un conjunto de protocolos que garanticen una comunicación confiable. En un SWE, la plataforma de comunicaciones suele estar basada en la pila de protocolos TCP/IP, en versiones de bajo peso (Light-Weight) como LWIP [1].

1.6. Protocolos de comunicación de Internet.

En Internet la información es transmitida en paquetes de códigos binarios. Los códigos son agrupados en Octetos (Bytes) y los bytes son agrupados dentro de paquetes de datos. Algunos de los protocolos necesitan que la información enviada sea interpretada y validada por el receptor, es decir algunos protocolos son orientados a la conexión. Los protocolos son agrupados en capas que en su conjunto forman una pila de protocolos denominada pila TCP/IP, ésta garantiza la comunicación a través de una red Ethernet [29]. En la Tabla 1. 4, puede apreciarse el resumen de las capas de comunicación con sus características y la función que cumplen en la implementación de un SWE.

Tabla 1. 4. Modelo pila TCP/IP del SWE.

Capa:	**Protocolos característicos:**	**Función dentro del sistema:**
Capa de aplicación.	HTTP, telnet, TFTP, email.	Plataforma para las aplicaciones: servidor web HTTP V1.1 [30] y servidor IPERF
Capa de transporte.	Pueden ser orientados a la conexión TCP (RFC 793) [29] o no orientados UDP (RFC 768).	Provee un servicio confiable de transporte TCP a la capa de aplicación. Permite configurar: conexiones y puertos, para el transporte seguro de datos. Corrección de errores a nivel de paquete.
Capa de Red.	IP e ICMP.	Proporciona los servicios a la capa de transporte. Se puede establecer la dirección IP y máscaras de red. Control de errores a nivel de tramas.
Capa de enlace.	Acceso al medio mediante algoritmos de negociación CSMA/CD.	Permite establecer y manejar una dirección MAC a la interfaz física, y evita colisiones de datos en el medio.

1.7. Aplicaciones WEB HTML.

Se denominan aplicaciones WEB a aquellas cuya interfaz se construye a partir de páginas WEB. Las páginas WEB no son más que ficheros de texto en un formato estándar denominado HyperText Markup Language (HTML). Estos ficheros se

29 T. Andrew. *Redes de computadoras. 4ta Ed*. (2003). U.S.A: Prentice Hall.

30 Network Working Group. *Hypertext Transfer Protocol -- HTTP/1.1.*(1999). Recuperado el 2011 de http://www.w3.org/Protocols/rfc2616/rfc2616.html.

almacenan en un servidor WEB, al cual se accede utilizando el protocolo HTTP. Para utilizar una aplicación WEB desde una máquina concreta, basta con tener instalado un navegador WEB en esa máquina, ya sea; Internet Explorer, Google Chrome, Opera, Mozilla o cualquier otro navegador. Desde la máquina cliente, donde se ejecuta el navegador, se accede a través de la red al servidor WEB, donde está alojada la aplicación y, de esa forma, se puede utilizar la aplicación sin que el usuario tenga que instalarla previamente en su máquina [31]. El lenguaje HTML es un estándar común y constituye los bloques básicos para la construcción de una interfaz gráfica, sus normas son definidas por el World Wide Web Consortium (W3C). W3C recientemente lanzó el primer borrador de HTML5 [32], retrasado en gran medida por el diseño del estándar XHTML.

Las páginas WEB HTML estáticas pueden ser la solución más adecuada cuando una página web se limite a ofrecer siempre la misma información, pero, la naturaleza dinámica de la web y las expectativas que ha creado en la actualidad hacen necesaria la implementación de aplicaciones WEB que generen dinámicamente el contenido, que finalmente se les ofrece a los usuarios. De tal manera que se pueda; seleccionar, filtrar, ordenar y presentar, la información de la forma más adecuada en función de las necesidades de cada momento.

1.7.1. HTML dinámico y JS (JavaScript).

Las limitaciones del formato HTML para construir interfaces de usuario (algo para lo que nunca fue diseñado), han propiciado la aparición de numerosas tecnologías que permiten ejecutar código en la máquina del cliente, generalmente dentro del propio navegador WEB [31].

El HTML dinámico (DHTML), crea un modelo basado en objetos de un documento HTML, de forma que, se pueda acceder fácilmente a los distintos elementos que lo componen como se observa en la Fig. 1. 3. La modificación dinámica de la página HTML, se realiza a través de: macros o Scripts, que suelen incluirse en el mismo fichero de la página.

---

31 B. Fernando, C. Francisco y C. Juan (2005). *Desarrollo Profesional de Aplicaciones Web con ASP.NET*. Recuperado el 2011 de http://csharp.ikor.org/

32 Networking Group. *HTML5 A vocabulary and associated APIs for HTML and XHTML*. (2011). Recuperado el 2011 de http://www.w3.org/TR/html5/.

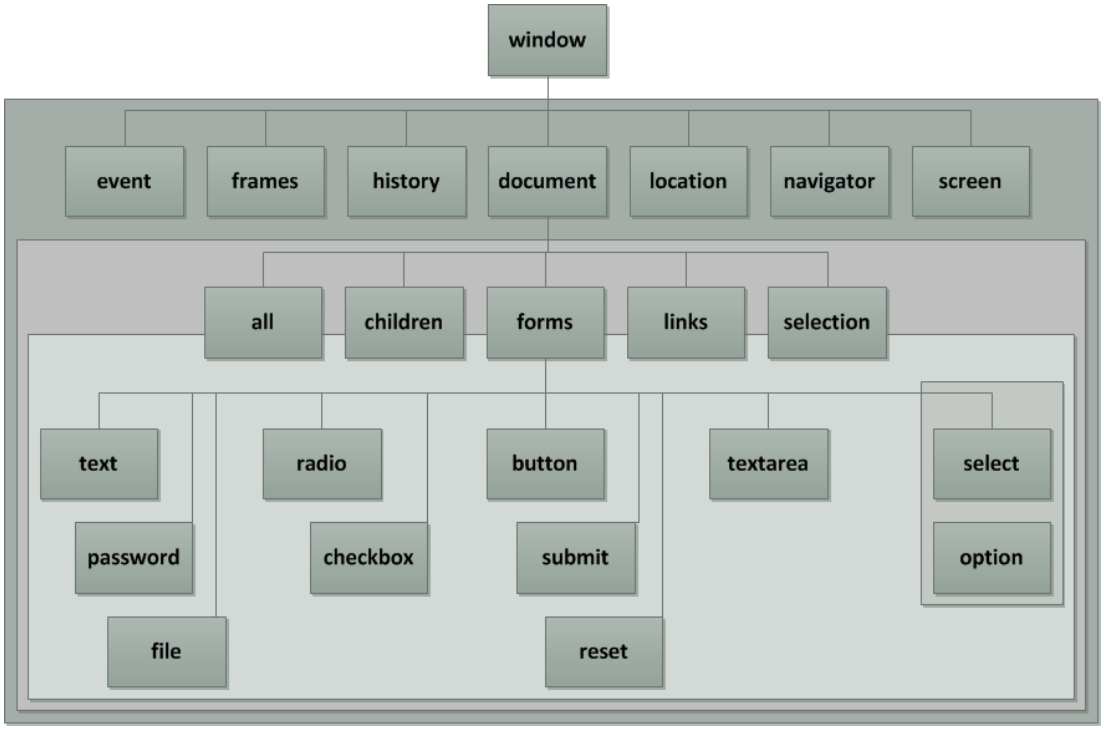


Fig. 1. 3. DHTML DOM. Referencia: [31].

En HTML dinámico, cada etiqueta HTML se convierte en un objeto con sus propiedades y eventos asociados. Los Scripts han de proporcionarle al navegador el código correspondiente a la respuesta prevista por el programador para los distintos eventos que se pueden producir, usualmente, los Scripts se escriben utilizando JavaScript por cuestiones de portabilidad.

JavaScript es un lenguaje interpretado originalmente llamado LiveScript que Netscape desarrolló para sus productos relacionados con la WEB. De hecho, JavaScript, funciona tanto en navegadores WEB como en el servidor HTTP de Netscape, al más puro estilo de las páginas ASP de Microsoft [33].

1.7.2. HTML y las hojas de estilo.

El principal inconveniente que tiene el formato HTML a la hora de crear páginas web es que debemos indicar el formato de los objetos (colores, tamaños, fuentes, etc.) junto con el código HTML del contenido del documento, lo cual aumenta el tamaño de la página considerablemente. Las hojas de estilo en cascada, Cascading Style Sheets (CSS), permiten detallar un formato uniforme para un conjunto de objetos en forma global, facilitando el mantenimiento de una o varias páginas HTML

33 W3Schools. *JavaScript Introduction*. (2010). Recuperado el 2011 de http://www.w3schools.com/js/js_intro.asp.

[34]. Todo el código del formato del estilo de los objetos se encuentra en un archivo de extensión *.css*, con lo cual, organizamos de una manera más eficiente el código HTML y CSS.

Es oportuno destacar que el código CSS, tiene su propia sintaxis diferente a HTML, la hoja de estilos a ser aplicada para un archivo HTML debe ser llamada desde el encabezado del documento.

1.8. Nivel de sistema y partición del SWE.

El objetivo de este proyecto es desarrollar un SWE con capacidades de comunicación TCP/IP, capaz de controlar o supervisar un sistema físico de manera remota. El sistema propuesto está compuesto de: una plataforma hardware y una plataforma de software, que permite la interacción, con diferentes; sensores y actuadores conectados al sistema, desde cualquier navegador tradicional conectado a una red Ethernet.

El sistema físicamente es una tarjeta de desarrollo Spartan 3E Starter Kit de Digilent y un módulo de adquisición y conversión de señales analógicas a digitales (ADC). Los recursos utilizados serán:

- Dispositivo externo ADC.
- FPGA.
- Memoria DDR SDRAM.
- Dispositivo Ethernet MAC (Interfaz RJ-45).
- Dispositivos para comunicación RS-232 DTE y DCE (Interfaz DB-9).
- Dispositivos discretos internos de salida (LEDS).
- Dispositivos discretos internos de entrada (SWITCHS).
- Dispositivos Reloj y fuentes de alimentación.

El esquema general de la propuesta para el SWE, puede apreciarse en la Fig. 1. 4

34 Networking Group. *CSS 2.1.* (2009) . Recuperado el 2011 de http://www.w3.org/TR/CSS21/.

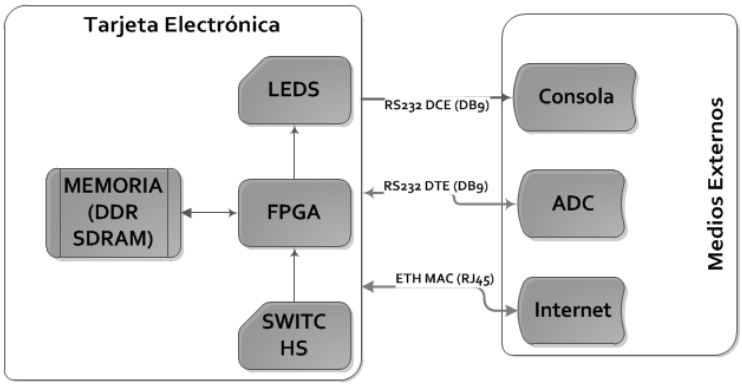


Fig. 1. 4. Esquema requerimientos de hardware para el SWE.

Sobre un plataforma de software se encuentra el SWE, una aplicación típica de un SWE contempla un servidor más una aplicación adjunta. El SWE provee una interfaz entre el navegador y la aplicación, con esto el navegador está en la capacidad de acceder a un nivel bajo de hardware con una interfaz gráfica sencilla de manejar. La propuesta de aplicación de Software implementada puede observarse en la Fig. 1. 5.

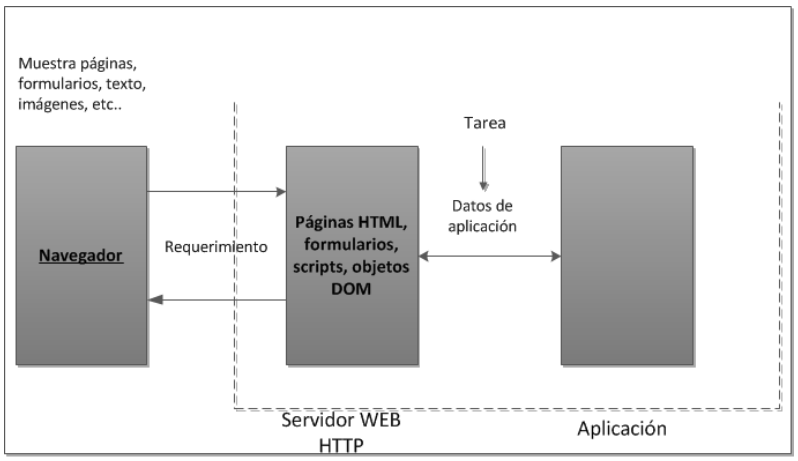


Fig. 1. 5. Esquema requerimientos de software del SWE.

2. CAPITULO II: DESARROLLO DEL HARDWARE.

En este capítulo se detalla el desarrollo del hardware del sistema según los requerimientos del SWE. Inicialmente se hace una breve descripción de la herramienta de integración de Xilinx para la construcción de sistemas embebidos, luego se presentan las características de la tarjeta de desarrollo y del módulo ADC externo. De igual forma se presentan las características y configuraciones de los IPCORES que manejan las interfaces físicas. Por último se presenta en forma detallada la arquitectura completa del sistema generado por EDK 11.1.

2.1. Ambiente de software integrado (ISE 11.1) de Xilinx

ISE 11.1 de Xilinx cuenta con un conjunto de herramientas para el desarrollo de sistemas embebidos (EDK), según se detalla en [35]:

- Xilinx Platform Studio (XPS): este entorno permite el desarrollo de hardware e incluye al EDK 11.1.
- Software Development Kit (SDK): Es un entorno de diseño de aplicaciones de software basado en la plataforma de soporte opensource Eclipse.
- IPCORES, que incluyen tanto periféricos, memorias y procesadores.

EDK 11.1 es parte de un ambiente integrado de software (ISE), permitiendo de una manera ordenada realizar el diseño e implementación de uno o varios sistemas embebidos sobre un dispositivo Xilinx, como se puede observar en la Fig. 2. 1.

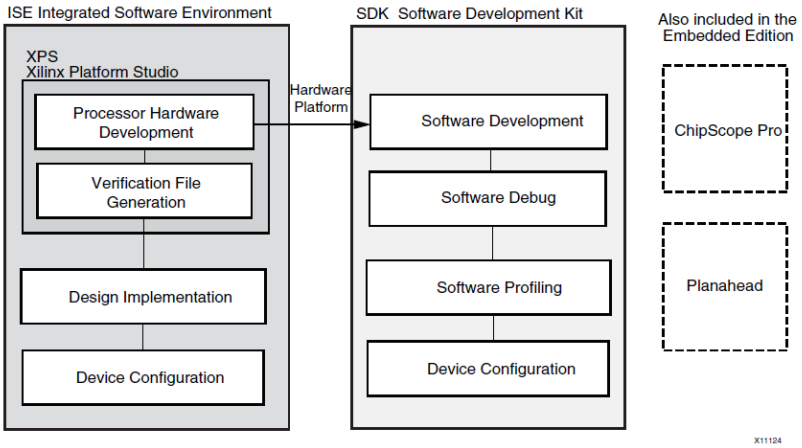


Fig. 2. 1. Herramientas para el diseño de un sistema embebido de Xilinx. Referencia: [35].

35 Xilinx Inc. *Xilinx Embedded System Tools Reference Guide 11.3.1.* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf

Como se pudo observar en la Fig. 2. 1, ISE incluye las herramientas para el desarrollo de la plataforma lógica de hardware: IPCORES, Firm Core o Hard Cores. El ambiente XPS es muy fácil e intuitivo de utilizar, soporta FPGAs Xilinx de las familias; Virtex, Spartan, Cool Runner y XC9500, en sus versiones básicas y profesionales. XPS cuenta con herramientas de diseño, verificación y depuración de: hardware (modos de simulación behavioral, structural, o timing-accurate), software (debugging tool, Instruction Set Simulator o execution profiling) y configuración de dispositivos (Impact) [35].

Una plataforma de hardware embebido normalmente consiste en uno o más; procesadores, periféricos y bloques de memoria, conectados a través de buses de datos. Cada uno de los núcleos de propiedad intelectual (IPCORES) tiene una serie de parámetros que pueden ajustarse para personalizar su comportamiento. Incluso se puede personalizar el mapa de direcciones de los periféricos y memorias. XPS permite seleccionar entre varias características opcionales y, en consecuencia, el FPGA sólo necesita implementar el subconjunto de funcionalidades que requiera la aplicación. La Fig. 2. 2, se muestra la interacción entre las diferentes herramientas de EDK 11.1 para la construcción de un sistema embebido.

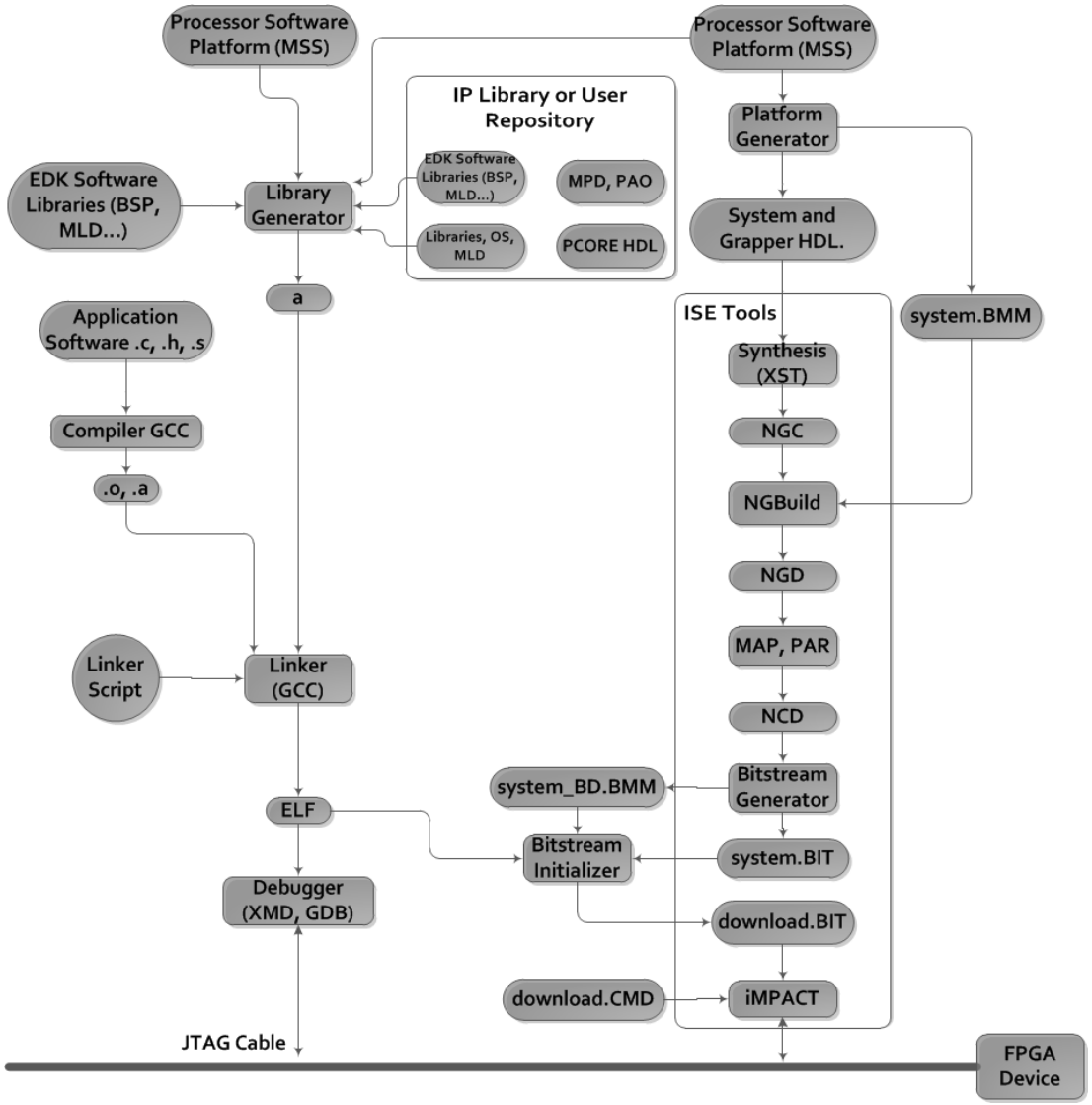


Fig. 2. 2. Desarrollo de sistemas embebidos según Xilinx EDK. Referencia: [35].

2.2. Hardware del sistema.

En la Fig. 2. 3, se puede observar una tarjeta Spartan 3E Starter Kit fabricada por Digilent, compatible con ISE de Xilinx, todos los dispositivos electrónicos se encuentran montados sobre una placa de 15,2x17,5 cm, el consumo de potencia depende de varios factores, en condiciones de operación normales la potencia disipada por el modulo con un procesador Microblaze es de 0,345 W. [36].

36 Xilinx Inc. *XPower Estimator User Guide*. (2011). Recuperado el 2011 de http://www.xilinx.com/support/documentation/user_guides/ug440.pdf

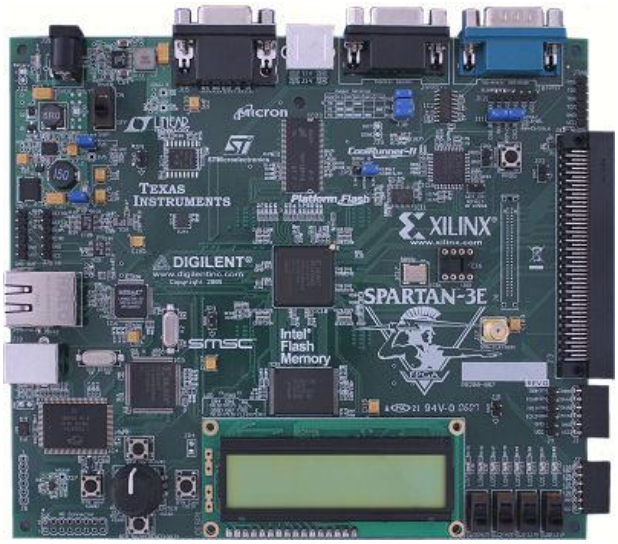


Fig. 2. 3. Spartan 3E Starter Kit. Referencia: [37].

Las interfaces físicas de los periféricos permitirán comunicar el sistema con el mundo exterior, se ha utilizado en el SWE las interfaces que se detallan en la Tabla 2. 1 de acuerdo con la referencia [37]:

Tabla 2. 1. Interfaces físicas de E/S del SWE.

Interfaz:	**Pines:**	**Características:**
DB9	9	Modo DCE, para reporte de operaciones mediante RS232.
DB9	9	Modo DTE, para conexión con módulo externo ADC, mediante RS232.
RJ-45	8	Para la comunicación con estándar IEEE 802.3 Ethernet.
Leds	8	Salida de variables digitales.
Switchs	4	Entrada de variables digitales.

Los dispositivos utilizados como; memorias, circuitos integrados y fuentes de poder son listados en la Tabla 2. 2, según se detallan en [37]:

Tabla 2. 2. Dispositivos utilizados por el SWE.

Dispositivo	**Descripción**	**Uso en el Sistema.**
FPGA	Xilinx XC3S500E-4FGG320C cuenta con 10476 celdas lógicas que equivale a 500000 compuertas, con 9,312 flip-flops. Con 4 rangos de velocidad con una RAM de 368640 bits, además cuenta con 250 de entradas y salidas de un número total de 320 pines [38].	Procesamiento de información, permite embeber IPCORES.
Plataforma Flash	Xilinx XCF04. Memoria PROM, en modo maestro Serial.	Permite almacenar y descargar aplicaciones para el SWE en conjunto con ISE 11.1.
Memoria SDRAM	DDR SDRAM 32MB Micron.	Almacena sistemas de ficheros,

37 Digilent Inc. *Spartan-3E Starter Kit Board User Guide*. (2006). Recuperado el 2011 de http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

38 Xilinx Inc. *Spartan-3E FPGA Family*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

	Memoria 512 Mbit (32M x 16) con tecnología Micron DDR SDRAM (MT46V32M16) con una interfaz de datos de 16 bits.	programas para el SWE.
Fuente de Poder	Linear technologies LT3412, 2.5 v de salida	Brinda la alimentación regulada de potencia para la memoria DDR-SDRAM.
Fuente de Poder	Texas Instruments TPS75003, triple regulador de voltaje. 1.2V al pin VCCINT del FPGA, 2.5V LA PIN VCCAUX del FPGA, y 3.3V a otros componentes VCCO del FPGA.	Proporciona la alimentación regulada de potencia al FPGA.
Ethernet PHY	SMSC LAN83C185, Reloj 25 MHz. 10 Mbps.	Realiza el procesamiento Ethernet, independiente, para el servicio de la capa física con negociación CSMA/CD.

2.2.1. Módulo ADC de 8 canales.

La tarjeta Spartan 3E Starter Kit posee 2 canales analógicos para la captura de datos analógicos, que consta de un preamplificador y un ADC [37]. Debido al reducido número de canales que posee la tarjeta Spartan 3E, se ha optado por construir un módulo ADC externo que permita el monitoreo de un conjunto mayor de variables. El módulo es capaz de monitorear ocho canales analógicos multiplexados y está basado en el microcontrolador ATMEGA32. El módulo se encuentra sobre una placa de circuitos que consta de:

- Interfaz DB9 normalizada con MAX 232.
- Microcontrolador ATMEGA32.
- Led indicador on/off.
- Switch on/off.
- Conector de alimentación de 5V.
- Salida de 5V.
- 8 pines de entrada de canales analógicos (0 V - 5 V).
- 1 pin común para los canales analógicos (GND).
- 2 pines para fuente de alimentación de salida (VCC).
- Pulsador de reset.

La comunicación desde y hacia el módulo ADC se la realiza mediante el protocolo RS232. El microcontrolador ATMEGA32 posee un consumo de 1.1 mA, lo que lo hace eficiente para este tipo de sistemas. Los ocho canales multiplexados del ADC

poseen una resolución de 10 bits en modo diferencial con una referencia seleccionable: interna (2.16V) o externa (0V a 5V) [39].

Según las especificaciones el tiempo de conversión es configurable 13 µs a 260 µs dependiendo del reloj configurado en nuestro caso 500 kHz, cada conversión toma 13 ciclos de reloj, además el rango de entrada es de 0 V a 5 V. El microcontrolador puede ser alimentado en un rango de 4.5 V a 5.5 V. El módulo se encuentra especificado en la Fig. 2. 4.

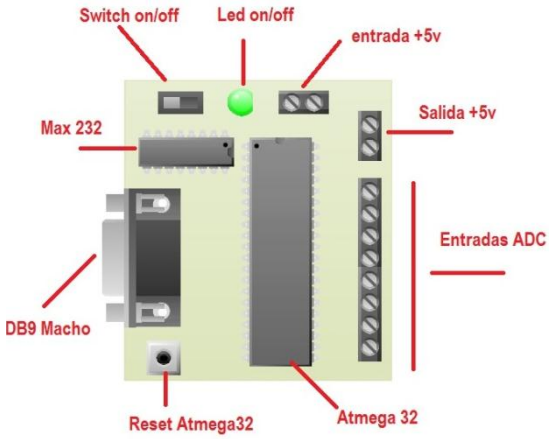


Fig. 2. 4. Esquema módulo ADC.

2.3. Microblaze.

Microblaze es el procesador del SWE, es un *Firm Core* Xilinx de arquitectura RISC de 32 bits, totalmente provisto para una compatibilidad y optimización para FPGAs: Spartan II, Spartan III y Virtex de Xilinx.

Microblaze se construye mediante la combinación de bloques de código denominados núcleos dentro de un FPGA Xilinx. El enfoque de núcleos plantea una optimización completa, ya que solo se ha incluido los bloques que son necesarios según los requerimientos del SWE.

Como se puede observar en la Fig. 2. 5, la arquitectura de Microblaze es altamente reconfigurable, posee 32 registros de propósito general de 32 bits visibles al usuario. También posee registros especiales como; contador de programa, estado de máquina, excepción de dirección, estado de excepción y estado de punto flotante, que son visibles al usuario pero no pueden ser modificados.

39 Atmel Corporation. *ATmega32*. (2011). Recuperado el 2011 de www.atmel.com.

Microblaze trabaja con una palabra de instrucción de 32 bits con 3 operadores y 2 modos de direccionamiento, un bus de direcciones de 32 bits, además de un pipeline de 3 etapas paralelo con control de saltos de programa [40].

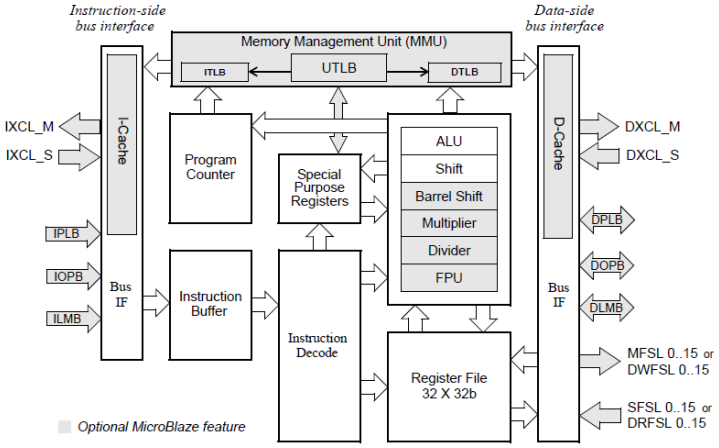


Fig. 2. 5. Arquitectura Microblaze v 7.1. Referencia: [40].

Microblaze usa el formato de Big-Endian (bit-reverso) para representar los datos. Los tipos de datos soportados por Microblaze son: palabra (32 bits), media palabra (16 bits), Byte (8 bits) y bit [40].

En el caso de Microblaze, el número total de instrucciones que soporta son 87, si se consideran diferentes las instrucciones que operan con valores inmediatos de aquellas que realizan la misma operación con registros. Adicionalmente, cada instrucción se ha elegido para que el tamaño de la ALU también sea reducido. El tamaño de instrucción en Microblaze es de 32 bits, definidas como: Tipo A y Tipo B.

- La Tipo A, tienen un máximo de dos operadores registro fuente y un operando registro destino.
- La Tipo B, tienen un registro fuente y un operando inmediato de 16 bits. Además de un operando registro destino.

Las instrucciones son provistas en las siguientes categorías funcionales: aritméticas, lógicas, bifurcación, cargar/guardar y especiales.

2.3.1. Diseño de Microblaze mediante EDK 11.1.

Todo diseño de un sistema embebido en EDK 11.1 empieza con la plataforma de hardware. EDK cuenta con un asistente (*Wizard*) para facilitar la creación de la

40 Xilinx Inc. *MicroBlaze Processor Reference Guide for EDK 10.1.* (2008). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf.

plataforma de Hardware de todo sistema embebido basado en Microblaze. En el Anexo 1, se detalla la creación del sistema de hardware basado en Microblaze del SWE desarrollado en el presente proyecto.

Existen tres paneles de configuración en EDK 11.1 de especial interés al momento de desarrollar la plataforma de hardware: *Bus Interfaces, Ports y Addresses*.

- *Bus Interfaces*, muestra las conexiones existentes entre los buses: PLB y LMB, y los IPCORES del sistema embebido, tal como se muestra en la Fig. 2. 6.

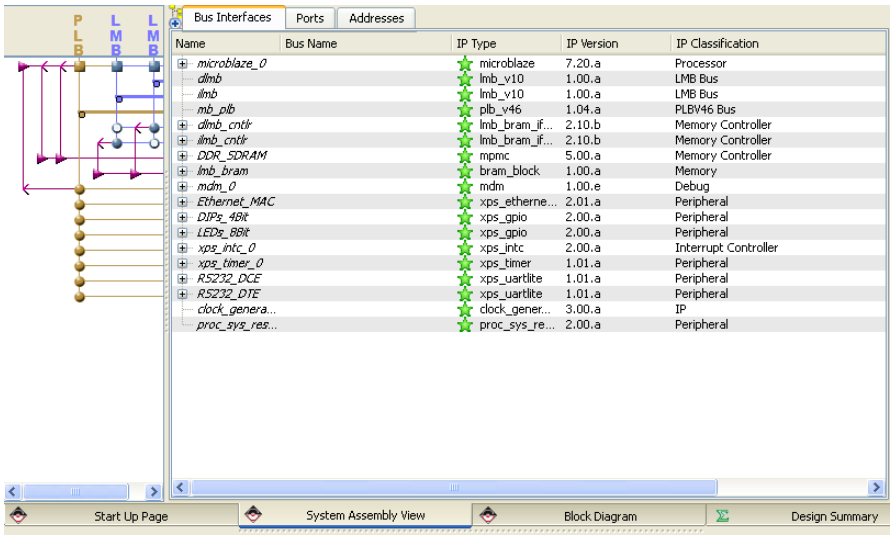


Fig. 2. 6. Buses e IPCORES. Fuente: EDK. 11.1.

- *Ports*, permite; crear, eliminar, visualizar y configurar los puertos y redes activos del sistema. Esta pestaña es de especial interés ya que en conjunto con el archivo system.ucf, permiten vincular un puerto lógico a una red física del FPGA. Los puertos del sistema pueden observarse en la Fig. 2. 7.

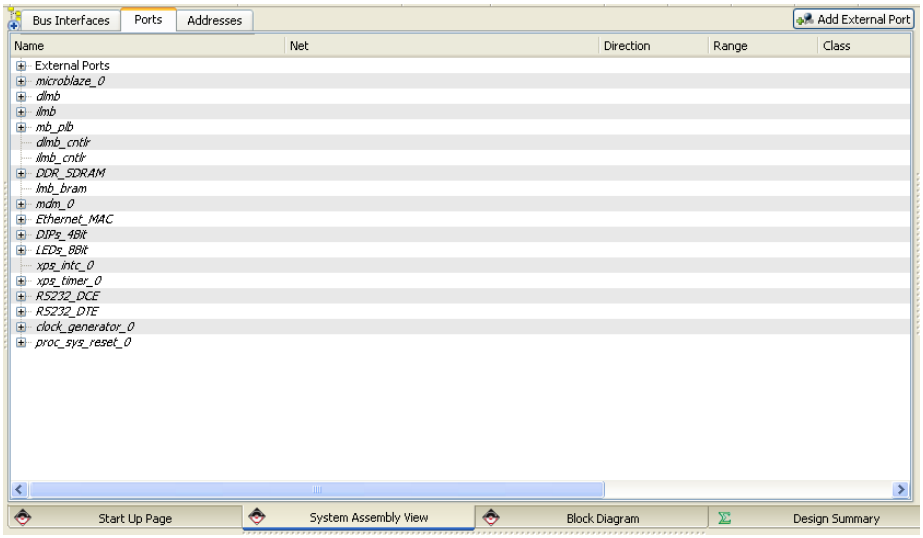


Fig. 2. 7. Puertos del sistema. Fuente: EDK 11.1.

- *Addresses*, esta sección permite dimensionar los espacios de memoria donde se ubicarán los IPCORES. Cada vez que se agregue un IPCORE será necesario generar una nueva distribución de memoria (*Generate Addesses*). La distribución de memoria puede apreciarse en la Fig. 2. 8.

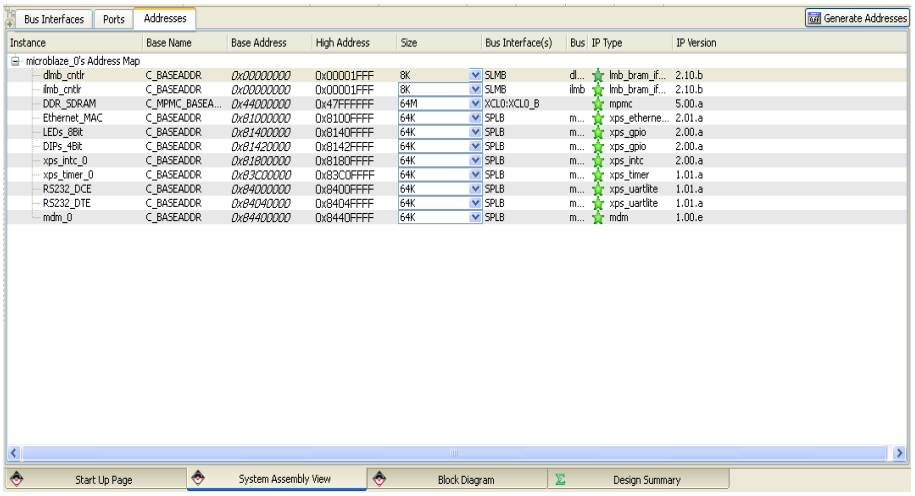


Fig. 2. 8. Dimensionamiento de memoria para IPCORES. Fuente: EDK 11.1.

2.4. Buses de comunicación.

2.4.1. Bus local de memoria (LMB).

Para conectar a Microblaze con los dispositivos y la memoria de alta velocidad, se ha utilizado el LMB V1.0a, este soporta una velocidad de hasta 125 MHz y un ancho de banda máximo de 500 MB/s. Solo se permite un bus maestro por sistema

embebido sin ningún método de arbitración. LMB trata separadamente; datos e instrucciones, consumiendo 1 LUTs por cada bus [41].

Se crean 2 instancias de este bus; una para instrucciones y otra para datos, que se conectarán a los puertos internos de Microblaze, estas dos instancias son parte del Bus de memoria local (LMB). Son conocidos en el sistema como: Bus de memoria local de instrucciones (ILMB) y Bus de memoria local de datos (DLMB), estos son generados automáticamente por EDK 11.1.

2.4.2. Bus local del procesador (PLB).

Es un bus síncrono de alta velocidad, utilizado para conectar periféricos y bloques de memoria interna del FPGA, posee una capacidad de 128 bits, proporciona una infraestructura donde se puede conectar un número opcional de buses PLB maestros y esclavos. Está constituido por una unidad de control de bus, watchdog y separación de direcciones de lectura y escritura de una unidad en particular. Con un número máximo de 16 maestros o esclavos, puede alcanzar velocidades de hasta 630 MHz, dependiendo de: las características físicas del sistema, la topología y el número de buses (esclavos y maestros) [42]. El ancho de banda depende del número de IPCORES conectados al bus y del tamaño de palabra que utilizan los mismos.

El PLB es un bus indispensable y al crear cualquier sistema embebido en EDK 11.1, se configura automáticamente según las necesidades de los diferentes periféricos, memorias y del procesador Microblaze.

2.4.3. Bus local de memoria del bloque de RAM (LMB BRAM).

El controlador de interfaz LMB BRAM (lmb_bram_if_cntrl) se conecta al bus lmb_v10 para proveer una interfaz de manejo de memoria. Posee una interfaz con el bus LMB v1.0 con soporte para Byte, se utiliza generalmente con un bram_block para proveer una solución de alta velocidad BRAM para puertos ILMB y DLMB de

41 Xilinx Inc. *Local Memory Bus (LMB) v1.0 (v1.00a).* (2005). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/lmb.pdf

42 Xilinx Inc. *Processor Local Bus (PLB) v4.6 (v1.04a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/ds531.pdf

Microblaze, soporta modos de transferencia: Byte, media palabra y una palabra [43].

2.5. IPCORES del SWE.

Los IPCORES son el concepto fundamental del diseño de sistemas SoC con arquitecturas reconfigurables. Los IPCORES proporcionan la lógica de conexión interna del FPGA que permiten el funcionamiento de la plataforma de hardware.

2.5.1. Manejo de memoria.

2.5.1.1. Controlador de memoria multipuerto (MPMC).

MPMC, es un controlador de memoria completamente configurable, proporciona acceso a memoria a través de uno de los ocho puertos disponibles, donde cada puerto puede ser escogido de un grupo de interfaces de módulos configurables (PIMs) esto permite la conectividad dentro del procesador Microblaze, utilizando un PLBv4.6 y una estructura MPMC Native Port Interface (NPI).

MPMC soporta el controlador Soft Direct Memory Access (SDMA) que provee interfaces: fullduplex, gran ancho de banda, interfaces LocalLink dentro de memoria. Adicionalmente, MPMC soporta un código corrector de errores (ECC), Performance Monitoring (PM), y registros de depuración, puede alcanzar velocidades entre 125 MHz– 133 MHz [44].

El MPMC se agrega en la etapa *Peripheral* del asistente EDK 11.1, mediante la inclusión de DDR SDRAM. Para optimizar recurso se utilizó un bus Dual XCL sobre un XCL PIM, la configuración de esta característica se detalla en el Anexo 1. Con esto se garantizó el acceso por métodos de arbitración ROUND-ROBIN interna a los buses de instrucciones y datos de Microblaze. Cabe recalcar que la prioridad lo tiene el canal B (datos).

2.5.1.2. Bloque de memoria de acceso aleatorio (BRAM).

El bloque BRAM es un módulo de memoria configurable que se puede ajustar a una gran variedad de controladores de interfaces BRAM. La estructura HDL del bloque

43 Xilinx Inc. *LMB BRAM Interface Controller (v2.10b).* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/lmb_bram_if_cntlr.pdf

44 Xilinx Inc. *Multi-Port Memory Controller (MPMC) (v5.00.a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf

BRAM es generado por el EDK *Design tools*, basándose en la configuración del controlador de interface BRAM. Todos los parámetros del bloque BRAM son automáticamente calculados y asignados por las herramientas: *Platgen* y *Simgen* de EDK 11.1 [37].

El bloque BRAM es una interfaz entre la memoria DDR SDRAM y los puertos de salida de Microblaze (DXCL e IXCL) que se moldea de acuerdo a los requerimientos de Microblaze en la etapa de diseño [45].

2.5.2. Receptor/Transmisor asíncrono universal (UART Lite).

La interfaz UART Lite, puede ser conectada al PLB y provee un método de control para la transmisión de datos seriales asíncronos. Soporta: 8 bits, 2 canales uno para recepción y otro para transmisión full-duplex con FIFO de 16 caracteres en ambos canales. También cuenta con la facilidad de configurar el tamaño de bits de carácter entre: 5 y 8, al igual que la paridad y la velocidad de transmisión según el estándar RS232c [46].

El SWE utiliza el IPCORE RS232 DCE para la supervisión mediante consola del sistema embebido desde un PC con un software gestor como Hyperterminal y una interfaz DB9. El sistema también posee otra instancia del IPCORE para el manejo del dispositivo RS232 DTE usado para la comunicación con el módulo de adquisición de datos analógicos (ADC). En ambos casos se deben adoptar los parámetros de comunicación mencionados en la Tabla 2. 3.

Tabla 2. 3. Parámetros de configuración para RS232.

Propiedad:	Valor:
Bits por segundo:	9600
Bits de datos:	8
Paridad:	Ninguno
Bits de Parada:	1
Control de Flujo:	Ninguno

2.5.3. Control de acceso al medio Ethernet Lite (MAC).

El IPCORE Ethernet Lite MAC, permite controlar la capa de acceso al medio según el modelo TCP/IP, está diseñado para incorporar las características especificadas en el estándar IEEE Std. 802.3 para la capa física (PHY). Se conecta a la FPGA mediante el estándar Media Independent Interface (MII), se vincula al sistema

45 Xilinx Inc. *Block RAM (BRAM) Block (v1.00a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf

46 Xilinx Inc. *XPS UART Lite (v1.01a).* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/xps_uartlite.pdf

mediante una interfaz PLB, el diseño HDL del IPCORE provee velocidades entre 10 Mbps y 100 Mbps, el SWE trabaja únicamente con el 10 Mbps por la limitante de hardware [47].

Algunas características se exponen a continuación tal como se detalla en la referencia [47]:

- La interfaz PLB del IPCORE es compatible con la especificación PLBV4.6.
- Mapeo de memoria directo en las interfaces de Entrada/Salida y un puerto doble de memoria para transmisión y recepción.
- MII para la conexión externa a tranceivers PHY en velocidades de 10 ó 100 Mbps.
- Puerto interno doble de memoria de 4 kB, repartidos simétricamente para recepción y transmisión, totalmente independiente para el manejo de datos de un paquete.
- Doble buffer de memoria opcional, 4 kB ping-pong para transmisión y recepción.
- Soporte para interrupciones de recepción y transmisión.

Ethernet MAC provee el acceso al medio de transmisión, en este caso el acceso al dispositivo PHY con interfaz RJ45. Posee 2 interfaces de confrontación: hacia fuera del sistema embebido a través del dispositivo y hacia adentro del sistema con una interfaz PLB esclava.

El encapsulado en tramas que realiza el IPCORE se muestra en la Fig. 2. 9. Los campos en trama y fuera de ella se transmiten de izquierda a derecha (LSB).

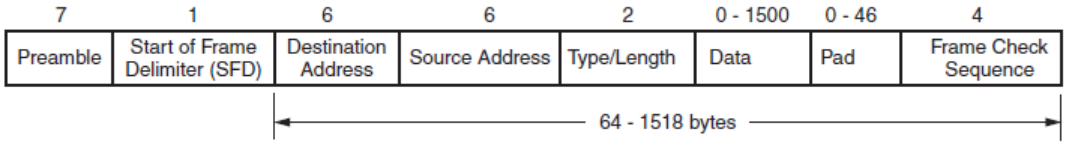


Fig. 2. 9. Entramado Ethernet MAC. Referencia: [47]

Proporciona dos modos de transmisión; Half-duplex y Full-duplex, el modo que se utilizará para el SWE es el half-duplex, por ende es necesario el protocolo de CSMA/CD debido a que la comunicación es en doble sentido.

47 Xilinx Inx. *XPS Ethernet Lite Media Access Controller (v2.01a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v2_01_a/ds759_axi_ethernet.pdf.

2.5.4. Entrada/Salida de propósito general (GPIO).

Las interfaces de salidas digitales de 8 bits y entradas digitales de 4 bits, son gestionadas por el IPCORE xps_gpio 2.00a, este IPCORE es de propósito general para entrada y salida de datos por los puertos discretos del FPGA. El IPCORE puede adaptarse a cualquier bus PLB de: 32, 64 o 128 bits [48].

Posee 2 canales con tamaños de 1 a 32 bits, que pueden ser de entrada o salida. Las instancias de los IPCORES están pre construidos en EDK 11.1, para el manejo de entradas y salidas digitales (LEDS y SWITCHS). Por ende pueden ser agregados o personalizados a conveniencia del diseño. El SWE, posee dos instancias del IPCORE GPIO; una para el control de los leds y otra para la supervisión de los switchs.

2.5.5. Interrupciones.

Todo sistema de cómputo posee mecanismos de interrupciones para sus módulos (E/S, memoria o procesador), mediante los cuales pueden interrumpir el procesamiento normal de la CPU, las clases más comunes son [49]:

- Programa, por una ejecución de una instrucción que produzca desbordamiento, división por cero o una instrucción invalida.
- Temporización, un periodo de tiempo planificado por el procesador para realizar tareas adicionales.
- E/S, generadas por un controlador de un dispositivo de E/S.
- Fallo de hardware, generadas por un fallo de hardware como: falta de potencia o un error de paridad de memoria.

Las interrupciones optimizan el tiempo del procesador de un sistema embebido, ya que éste puede dedicarse a otra tarea mientras espera a que un dispositivo entregue la información requerida. Cuando el dispositivo está listo para recibir más instrucciones envía una señal de interrupción que suspende cualquier programa que se esté ejecutando en ese momento y el procesador salta a un programa gestor de interrupciones.

48 Xilinx Inc. *XPS General Purpose Input/Output (GPIO) (v2.00a*). (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/xps_gpio.pdf

49 S. William. *Organización y Arquitectura de computadores 7ma Ed*. (2006). U.S.A: Prentice Hall.

Microblaze soporta una sola fuente de interrupción externa. El procesador solo atiende a la interrupción sí el bit en el registro de estado de máquina Interruption Enable (IE) está activo (1). En una interrupción, la instrucción en etapa de ejecución se completa, mientras la ejecución en etapa de decodificación es remplazada por una bifurcación al vector de interrupción (dirección 0x10). El retorno de la dirección cuando se produjo la interrupción es automáticamente guardado en un registro de propósito general R14. Además, el procesador también desactiva interrupciones futuras borrando el bit IE en el MSR [40].

El controlador de interrupciones (XPS INTC) se utiliza para concentrar múltiples entradas de interrupciones para dispositivos periféricos a una sola salida hacia el procesador del sistema. Posee una interfaz PLB para la conexión a un bus de datos [50].

El xps_timer de Xilinx, trabaja en conjunto con el Interrupt controller (INTC) y permite hacer la planificación para las tareas de red, este dispositivo se encuentra más vinculado al IPCORE Ethernet MAC, posee una interfaz PLB y puede conectarse con el INTC mediante el puerto de interrupciones.

El esquema de conexión de puertos de interrupciones de los IPCORES vinculados se muestra en la Fig. 2. 10.

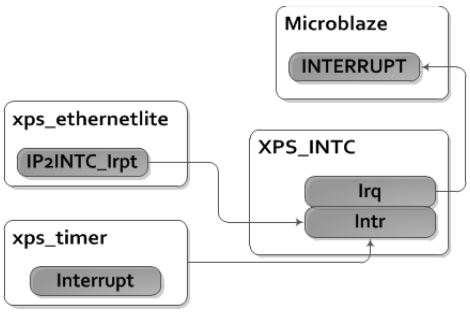


Fig. 2. 10. Esquema de conexión de los puertos de interrupción.

El Anexo 1, muestra las líneas de código necesarias a ser agregadas en el fichero de descripción de hardware (system.msh) para la inclusión de puertos y redes para obtener un correcto funcionamiento de las interrupciones en el sistema.

50 Xilinx Inc. *XPS Interrupt Controller (v2.00a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/dcr_intc.pdf

2.6. Arquitectura del sistema.

El resultado de agregar y configurar de todos los IPCORES en los apartados anteriores y en el Anexo 1, es un diagrama de la arquitectura del sistema embebido para el SWE, donde se esquematizan las instancias de; periféricos, memorias y microprocesador, relacionados entre sí mediante buses de datos.

Los buses se los ha categorizado como:

- P2P, son bus dedicado a dispositivos de depuración.
- LMB, son los buses locales mencionados en el apartado 2.4.1.
- PLB, el bus de periféricos analizado en el apartado 2.4.2.

En la Fig. 2. 11, se puede apreciar la arquitectura completa del SWE.

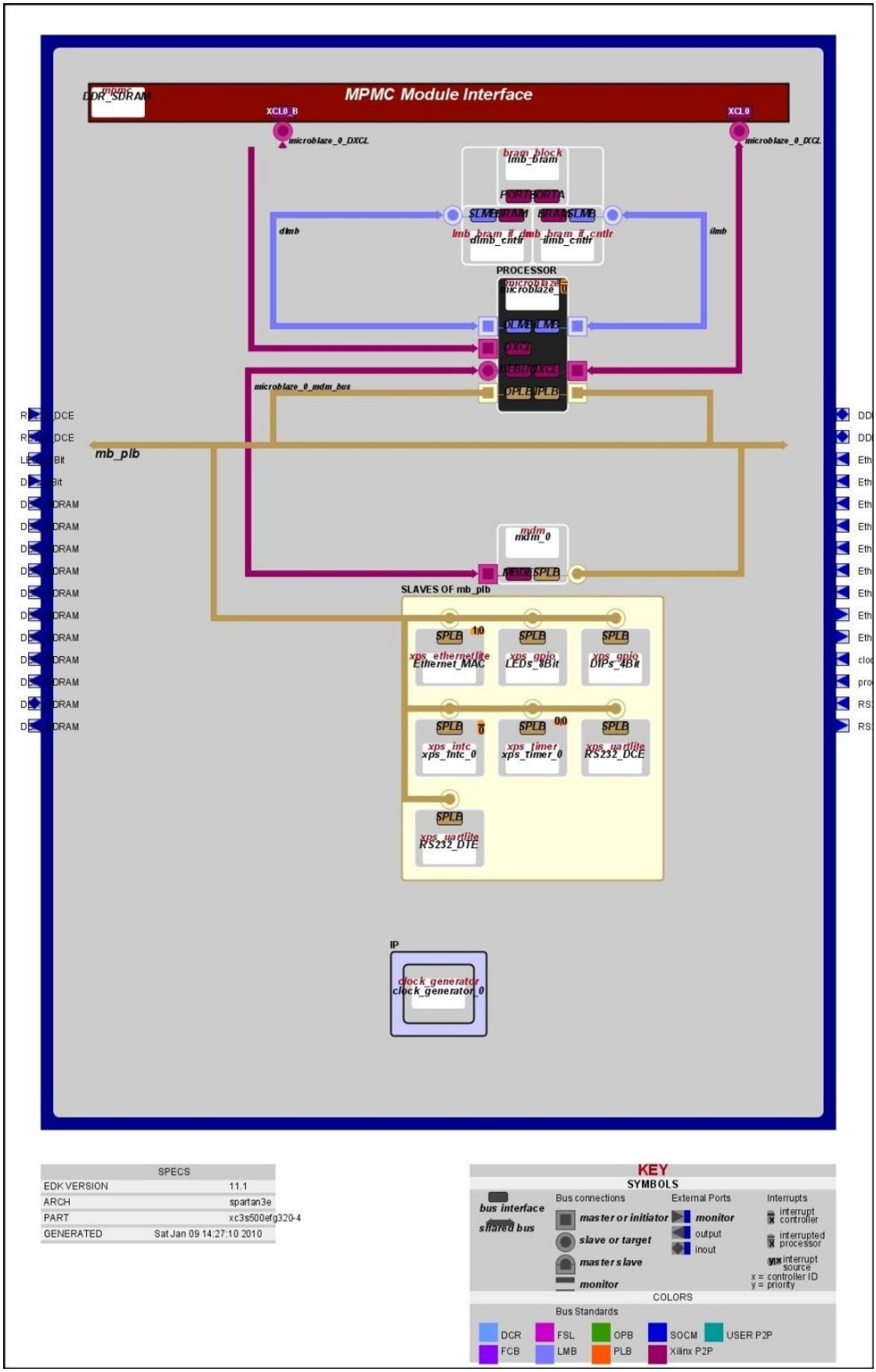


Fig. 2. 11. Arquitectura del SWE. Fuente: EDK11.1.

El diseño del hardware del sistema embebido en EDK 11.1, se resume en tres archivos de especial importancia:

- El archivo system.mhs, contiene la descripción de hardware; puertos, señales de entrada y salida, opciones de configuración del Microblaze, buses, recursos de utilización del hardware y opciones de solución de errores (debug).
- El archivo system.mss, contiene las opciones de compilación del software del sistema, tales como: modo de compilación de código, asignación de librerías a periféricos, método de solución de errores.
- El archivo system.ucf, que contiene la relación de los puertos con los pines físicos de la tarjeta de acuerdo a los requerimientos del usuario.

3. CAPITULO III: DESARROLLO DEL SOFTWARE.

En el presente capítulo se detalla: la creación y configuración de la plataforma de software (tipo, bibliotecas y controladores), programación de módulos para el servicio del SWE, programación de módulos para el servicio IPERF. También se presenta el diseño de la página Web en HTML.

3.1. Diseño de plataforma de software.

El diseño de la plataforma de software involucra las siguientes actividades:

- La importación de la plataforma hardware desde EDK 11.1 hacia SDK 11.1.
- Creación de una nueva plataforma tipo STANDALONE.
- Asignación y compilación de drivers de los IPCORES.
- Inclusión y configuración de bibliotecas (LWIP130 y Xilmfs).

La importación de la plataforma de hardware se la puede realizar desde el menú *project* del EDK 11.1, para realizar la importación de la plataforma es importante asegurarse que se haya realizado el mapeo de direcciones de memoria de los IPCORES y se haya generado el archivo system.bit.

El tipo de plataforma utilizado es STANDALONE, cuenta con todas las funcionalidades del lenguaje C, además de: semáforos, interrupciones, rutinas, entre otras utilidades optimizadas para Microblaze.

Al momento de generar la plataforma de software se crean directorios que guardan las bibliotecas propias de la plataforma como; lib.a, libm.a y libxil.a, éstas agregan capacidades al sistema, como se detalla en la referencia [51]:

- lib.a, es la biblioteca que permite la utilización de las funciones estándar del lenguaje C pre-compiladas para el procesador Microblaze.
- libm.a, ésta biblioteca posee todas las funcionalidades matemáticas como: seno, coseno, exponencial, etc.
- libxil.a, contiene las funcionalidades pre compiladas para el manejo de excepciones, interrupciones, cache, etc., propias de Microblaze.

51 Xilinx Inc. *OS and Libraries Document Collection.* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf

En el árbol de Microblaze se encuentran bibliotecas que definen cada uno de los parámetros importados desde el archivo system.mss. En el directorio libsrc se encuentran los códigos fuente de los controladores para los periféricos incluidos en el archivo system.mhs, tal como se muestra en la Fig. 3. 1.

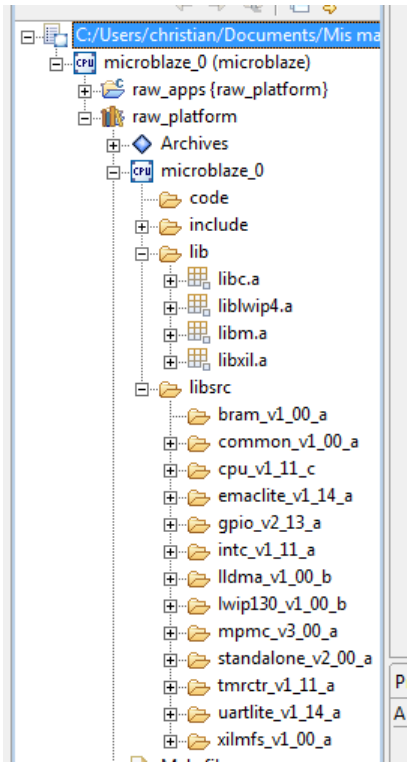


Fig. 3. 1. Árbol de directorios. Fuente: EDK. 11.1.

El esquema general de la plataforma de software tipo STANDALONE de Xilinx puede apreciarse en la Fig. 3. 2.

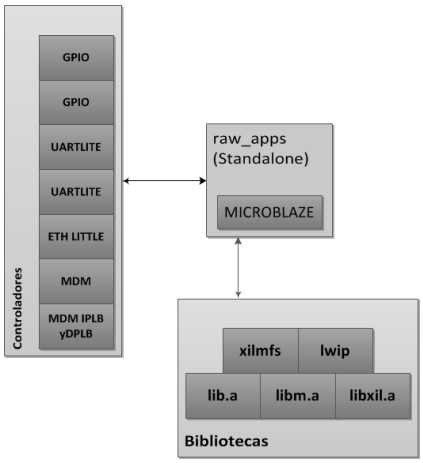


Fig. 3. 2. Esquema de la plataforma de software.

3.1.1. Biblioteca Xilinx memory file system (Xilmfs).

La biblioteca de funcionalidades Xilmfs, agrega la capacidad de ordenar la memoria del sistema en espacios contiguos como si se tratara de un sistema de ficheros, se puede acceder a ellos mediante llamado a funciones de C de alto nivel [51].

Las funcionalidades de la biblioteca Xilmfs van más allá del almacenamiento de ficheros, se puede; crear, eliminar, renombrar, buscar, leer o escribir archivos y directorios. La biblioteca Xilmfs permite cargar una imagen MFS donde se guardarán los archivos y recursos que permitirán brindar el servicio WEB.

3.1.1.1. Configuración.

La inclusión de esta biblioteca se la realizó desde las propiedades de la plataforma (*Software Platform Settings*), activando el visto correspondiente a la biblioteca Xilmfs, como se puede observar en la Fig. 3. 3.

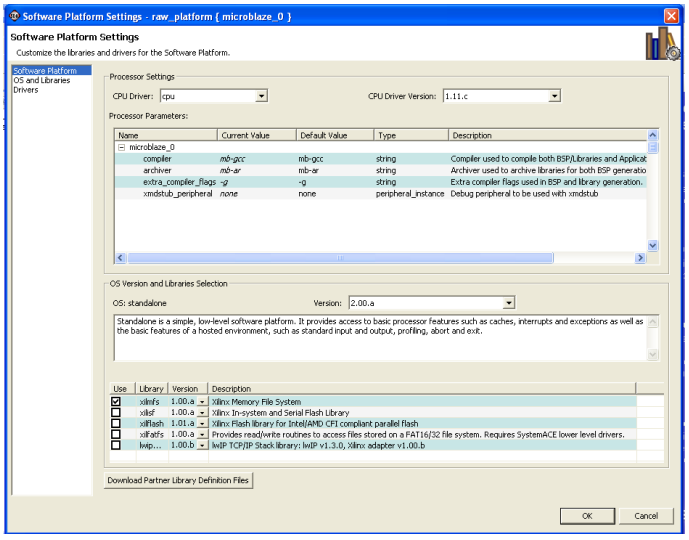


Fig. 3. 3. Inclusión de la biblioteca Xilmfs. Fuente: SDK 11.1.

Para configurar los parámetros de la biblioteca, se recurrió al menú *OS and libraries*, se debe establecer los parámetros: tamaño del sistema de ficheros en bytes (*numbytes*) y dirección de inicio (*baseaddress*) que debe ser coherente con el inicio de la memoria DDR SDRAM, en nuestro caso la dirección de inicio del IPCORE SDRAM es 0x47000000 (Hexadecimal). Esta dirección se la puede encontrar en el parámetro *BaseAddress* de la instancia mdm del archivo system.mhs.

El parámetro *numbytes*, deberá asignarse una vez que se haya desarrollado el sistema de ficheros para las aplicaciones WEB y se haya generado la imagen MFS, el mismo que deberá ser coherente con el tamaño de imagen obtenido.

También se debe configurar los parámetros: init_type y need_utils La configuración de la biblioteca puede observarse en la Fig. 3. 4.

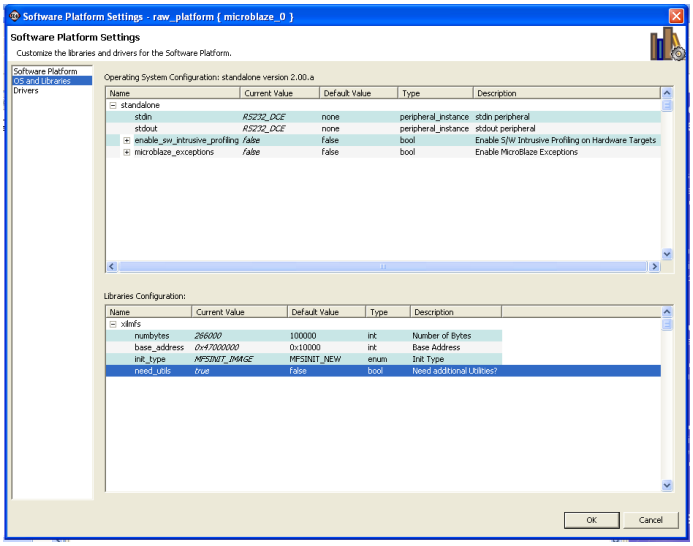


Fig. 3. 4. Configuración de la biblioteca Xilmfs. Fuente: SDK 11.1.

3.1.2. Biblioteca LWIP 1.3. (v1.00.a).

La biblioteca de funcionalidades de red LWIP es una implementación de la pila de protocolos TCP/IP de 40 kB [52]. El concepto central de la implementación de la pila LWIP es reducir el uso de memoria y el tamaño del código, permitiendo su uso en servidores con recursos muy limitados.

LWIP usa un API hecho a la medida, el cual no requiere ninguna copia de datos. Esta es una pila de protocolos de tipo STANDALONE que no depende de cierto sistema operativo, pero tiene la capacidad de emular alguno para su funcionamiento [51].

La versión de la biblioteca LWIP en EDK 11.1 se conoce como LWIP130 v1.00.a, es compatible con los IPCORES: Ethernet MAC xps_ethernetlitle y xps_ll_temac de Xilinx, esta implementación de Xilinx está basada en la versión de código abierto LWIP1.3.0.

52 D. Adam. Raw TCP/IP interface for lwIP. (2001). Suecia: Swedish Institute of Computer Science. Recuperado el 2011 de http://lwip-avr.googlecode.com/svn-history/r2/trunk/lwip/doc/rawapi.txt

La implementación de la interfaz de programación de aplicaciones (API), se divide nativamente en dos partes como se muestra en Fig. 3. 5, una parte de la API se implementa como una biblioteca vinculada al programa de aplicación, y la otra parte se implementa en el proceso TCP/IP de la plataforma. Las dos partes se comunican mediante los mecanismos de comunicación interproceso (IPC) provistos por la capa de emulación del sistema operativo [52].

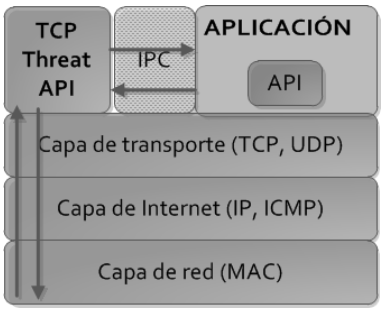


Fig. 3. 5. Pila TCP/IP basada en LWIP.

El IPC, es un mecanismo de la capa de emulación del sistema operativo (SO) que permite la comunicación entre el API del proceso LWIP y el API del programa de aplicación, esta capa de emulación brinda una interfaz uniforme para el manejo de; servicios de temporización, procesos de sincronización (semáforos) y mecanismos de paso de mensajes (mailbox). Se utiliza tres mecanismos de IPC:

- Memoria compartida,
- Paso de mensajes, y
- Semáforos.

LWIP está basada en un modelo en el cual todos los protocolos se encuentran en un solo proceso independiente del Kernel del SO, donde todos los programas de aplicación pueden o no residir en el proceso LWIP. La comunicación entre los programas de aplicación y la pila TCP/IP se la hace mediante el llamado a funciones, compartiendo el hilo de proceso con LWIP.

3.1.2.1. Requerimientos.

Según se describe en la hoja de especificaciones de la biblioteca LWIP [52], los requerimientos mínimos para su funcionamiento son:

- Procesador Microblaze o Power PC (PPC).
- IPCORE EMAC (Ethernet Lite).

- Timer (Control de tiempos de interrupciones para TCP/IP).

El esquema de interconexión de los IPCORES según los requerimientos se puede observar en la Fig. 3. 6, el cual coincide plenamente con la arquitectura de hardware construida en el apartado 2.

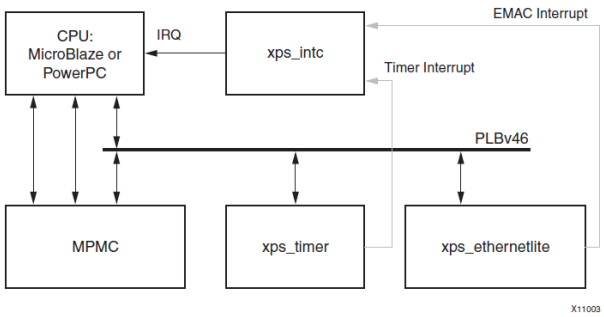


Fig. 3. 6. Conexión de los componentes necesarios para la biblioteca LWIP. Referencia: [51].

3.1.2.2. Manejo de memoria.

La pila de protocolos segmenta la memoria en buffers con capacidades variadas de almacenamiento, los buffers soportan diferentes tamaños de paquetes entrantes en una comunicación TCP/IP. Los buffers residen en un lugar específico de memoria destinado a las aplicaciones que no es manejado por el subsistema de red [52]. La representación de un paquete dentro de LWIP se la hace mediante un buffer llamado PBUF, que soporta asignación de memoria estática o dinámica. En la Fig. 3. 7, se puede visualizar la estructuración de memoria de un PBUF.

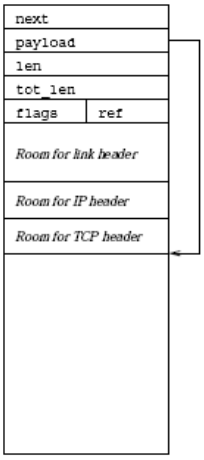


Fig. 3. 7. Buffer PBUF, manejado por un subsistema PBUF. Referencia: [52].

3.1.2.3. Configuración de parámetros LWIP TCP para HTTP.

Con el propósito de optimizar los recursos y garantizar el transporte seguro vía HTTP, es necesario configurar los parámetros TCP de la librería LWIP. Se debe tener en cuenta que los módulos de TCP deciden cuándo: bloquear o enviar datos [53]. Existen varios parámetros TCP configurables para la biblioteca LWIP130:

- Ventana TCP, el tamaño de ventana configurado es de; 4096 Bytes, e indica que no se pueden recibir datos de mayor tamaño que la ventana, para evitar una congestión en la red.
- Tamaño de buffer, éste almacena la información a enviar y recibir, se lo limitó a 2048 Bytes por transferencia, esto es debido a que la mayoría de transacciones HTTP son cortas.
- Tamaño máximo de un segmento TCP debe ser 2 kB.

Una correcta configuración de los parámetros LWIP130 TCP en SDK 11.1, se puede apreciar en la Fig. 3. 8.

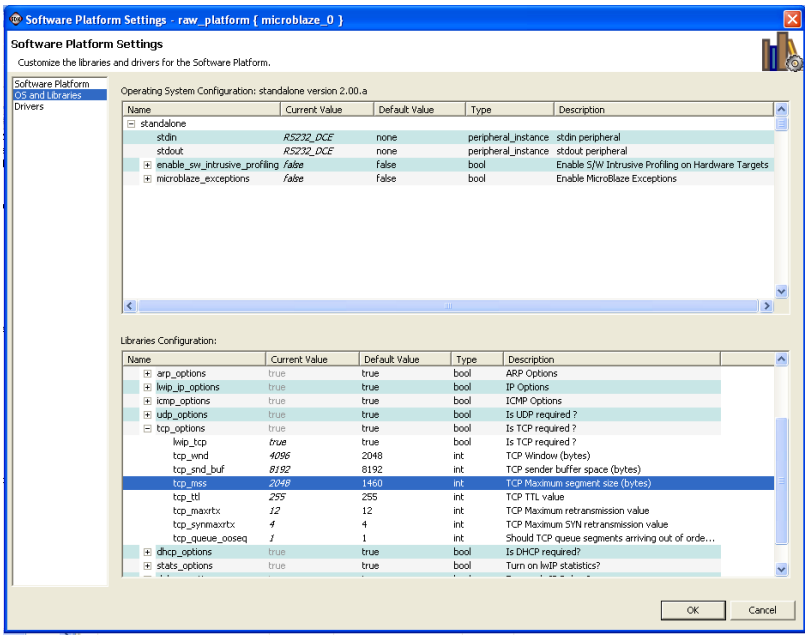


Fig. 3. 8. Configuración LWIP130 TCP. Fuente: EDK 11.1.

3.2. Estructura modular de componentes del sistema.

El diseño de aplicaciones se lo ha realizado de forma modular de tal manera que facilite la depuración del programa. Como se mencionaba en el apartado 2.1, SDK 11.1, es la herramienta principal en el diseño de aplicaciones de software. Con una

53 Networking Group. *Protocolo De Control De Transmisión RFC0793*. (1981). Recuperado el 2011 de http://www.rfc-es.org/rfc/rfc0793-es.txt.

correcta estructuración modular basada en los métodos de programación estructurada y la implementación de funcionalidades de las bibliotecas: LWIP, Xilmfs y Microblaze, se logró obtener un programa: compacto, flexible y funcional.

Se optó por el lenguaje C debido a su repertorio de funcionalidades compatibles con Microblaze, que permiten un nivel más bajo de programación, necesarias al momento de trabajar con arquitecturas reconfigurables y acceso a registros de periféricos del sistema.

La estructura de toda la plataforma de aplicación se la menciona a continuación y se realiza una breve descripción de las funcionalidades que presentan.

- main.c, realiza el llamado para: inicialización de Microblaze (interrupciones, buffers, contadores), inicialización de la plataforma de hardware, inicialización de la pila TCP/IP, inicialización de servicios del sistema y escucha conexiones entrantes.
- dispatch.c, direccionamiento para la inicialización de los servicios e impresión de cabeceras en consola para todos los servicios. Además hace la redirección para los llamados a función para el procesamiento de paquetes de cada una de las aplicaciones.
- platform.c, contiene todas funciones para la inicialización de la plataforma como: cache de instrucciones y datos, timers, registros, interrupciones necesarios para la operación de LWIP130.
- platform_fs.c, destinado a la inicialización del sistema de ficheros MFS, contiene todas las funcionalidades para verificar la existencia e integridad de una imagen MFS y que en su directorio raíz se encuentre el archivo index.html.
- platform_gpio.c, el módulo contiene las funciones de bajo nivel que permiten la inicialización y la operación con los dispositivos de entrada y salida GPIO.
- prot_malloc.c, este módulo permite reservar espacio en memoria DDR SDRAM para las estructuras de datos destinadas al almacenamiento de los paquetes, además, tiene la funcionalidad de proteger de accesos no autorizados a la memoria que contenga datos.
- webserver.c, este módulo está destinado a ofrecer el servicio WEB. Contiene funciones de: inicialización del servicio WEB, impresión de

cabeceras por consola, función de llamado para aceptar una conexión, función para la recepción y transmisión de datos.

- http_response.c, permite generar una respuesta coherente con el formato HTTPv1.1. Puede reconocer y generar respuestas: POST y GET, además es capaz de generar mensajes de recursos no encontrados en caso de no existir los recursos en la imagen MFS.
- web_utils.c, permite decodificar el tipo de recurso solicitado en el método POST o GET, genera la cabecera para la respuesta del servidor en formato HTTPv1.1. Además es capaz de reservar y liberar espacio en memoria DDR para el procesamiento HTTP.
- rxperf.c, contiene todas las funcionalidades del servicio IPERF. Funciones específicas para la aceptación y establecimiento de una conexión TCP para medición del ancho de banda desde un cliente IPERF.

En la Fig. 3. 9, se puede observar estructuración de módulos y sus relaciones con funciones de: C, LWIP130 y Microblaze, que se verán con más detalle en las secciones posteriores.

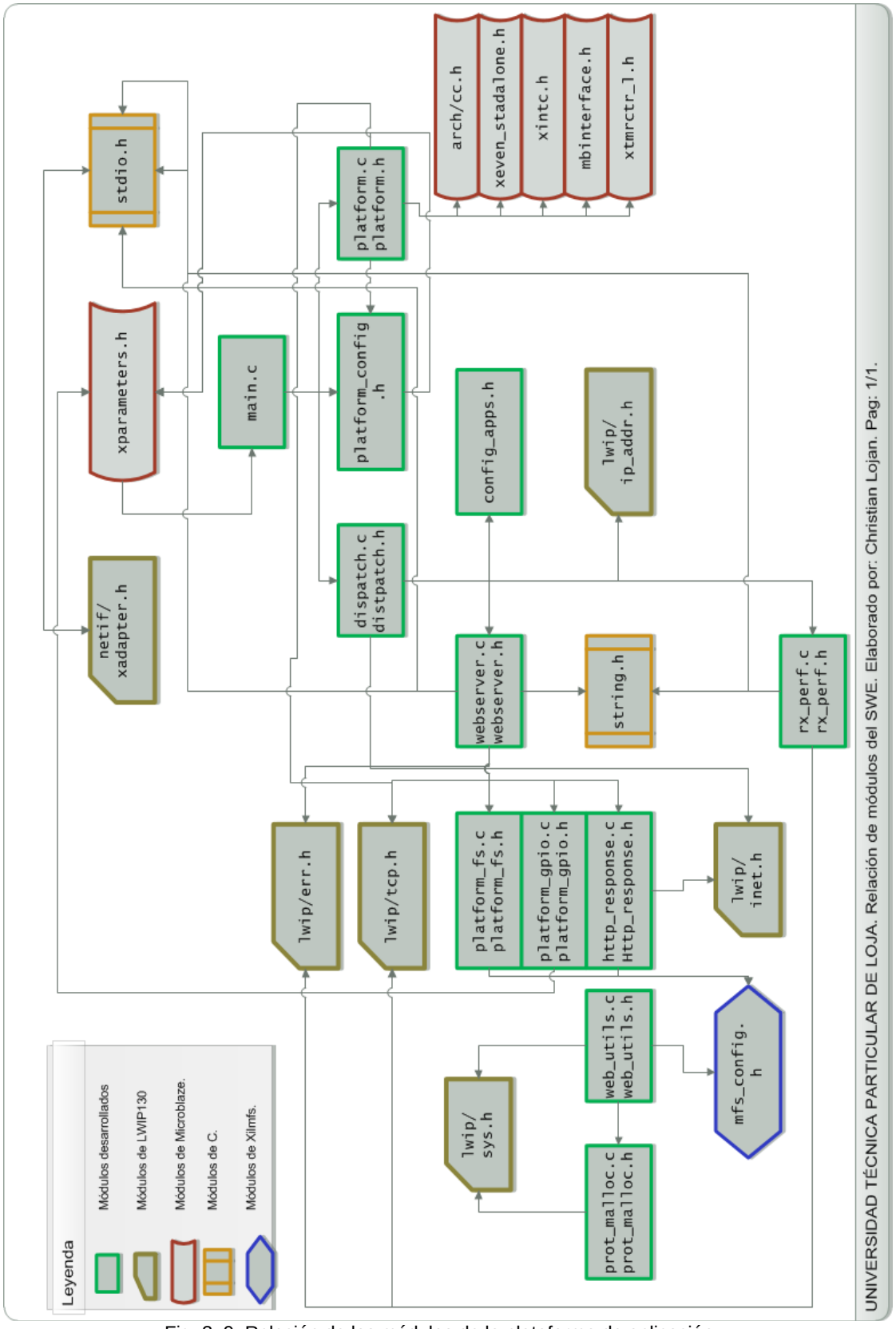


Fig. 3. 9. Relación de los módulos de la plataforma de aplicación.

Los módulos de todo el sistema visualizados desde el explorador del SDK 11.1, pueden apreciarse en la Fig. 3. 10.

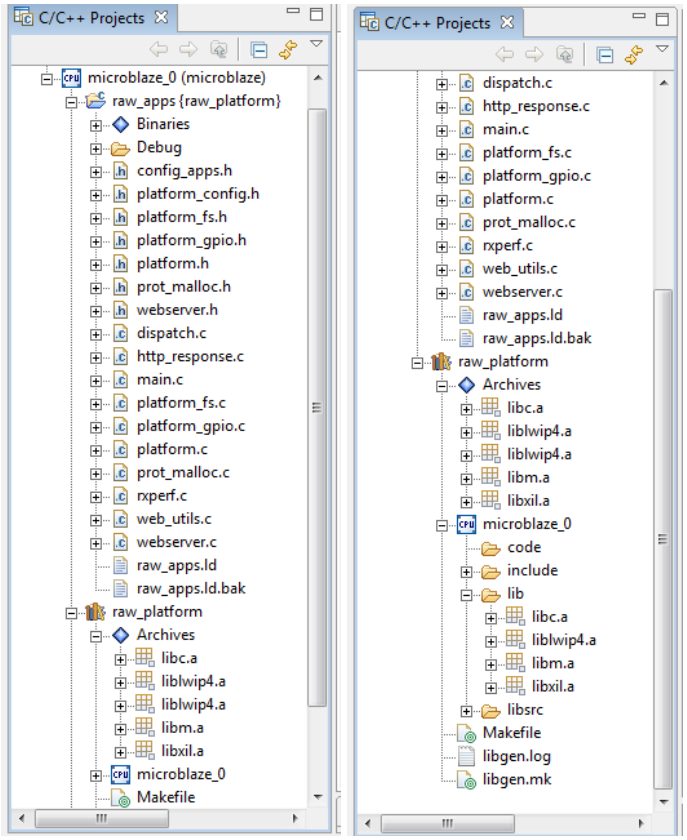


Fig. 3. 10. Módulos del sistema. Fuente: SDK 11.1.

3.3. Programa principal.

El programa principal es el corazón de la plataforma, es capaz de llamar a los módulos de aplicación directa o indirectamente. El diagrama de flujo en la Fig. 3. 11, muestra un esquema operacional sencillo, con el objetivo de una mejor comprensión de las funcionalidades del módulo principal.

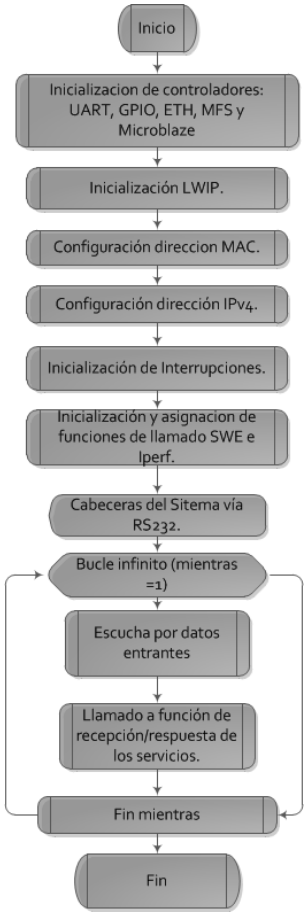


Fig. 3. 11. Diagrama de flujo main.c.

El programa principal, se encarga de las siguientes tareas:

- Inicializa la plataforma de hardware y software.
- Inicializa la biblioteca LWIP.
- Inicializa la dirección MAC.
- Inicializa la configuración de las capas de red y establece la dirección IP.
- Inicializa las interrupciones necesarias para LWIP y Microblaze.
- Llama al módulo de inicialización de las aplicaciones (SWE e IPERF).
- Llama al módulo de impresión de cabeceras visibles mediante consola.
- Escuchar constantemente por paquetes entrantes.

Para cumplir con las tareas antes mencionadas, el sistema hace uso de las funciones que se mencionan en la Tabla 3. 1.

Tabla 3. 1. Funciones del programa principal.

Función:	**Módulo huésped:**	**Módulos asociados:**	**Descripción:**
init_platform()	platform.h, platform.c	platform_config.h, xenv_standalone.h, xparameters.h, xintc.h, mb_interface.h, xtmrctr_l.h, lwip/tcp.h.	Inicialización de la memoria cache, contadores, controladores RS232 (DCE y DTE), sistema de ficheros, interrupciones y comprobación del archivo index.html.
lwip_init()	liblwip4.a	Todos los módulos LWIP130.	Inicializa las funcionalidades TCP/IP.
xemac_add()	liblwip4.a	Netif/xadapter.h, lwip/tcp.h, ip_addr.o, ip.o, sys.o, xadapter.o, xemacliteif.o.	Configura las direcciones: MAC e IPv4 de la plataforma.
platform_enable_interrupts()	platform.c	mb_interface, xtmrctr_l.h, xintc.h, xparameters.h, xenv_standalone.h.	Inicializa las interrupciones del sistema.
netif_set_up(netif)	liblwip4.a	netif/xadapter.h, lwip/tcp.h, ip_addr.o, ip.o, sys.o, xadapter.o, xemacliteif.o.	Configura e inicializa la interfaz de red.
start_applications()	distpatch.c	lwip/inet.h, lwip/ip_addr.h, config_apps.h	Inicializa el SWE e IPERF.
xemacif_input()	liblwip4.o	lwip/inet.h, netif/xadapter.h, lwip/tcp.h.	Escucha las conexiones entrantes.
transfer_data()	distpatch.c	lwip/inet.h, netif/xadapter.h, lwip/tcp.h.	Envía respuestas a solicitudes de conexiones entrantes.

En el Anexo 2, se puede apreciar en detalle el código en C correspondiente al programa principal de la plataforma de aplicación.

3.4. Programa SWE.

3.4.1. Procesamiento TCP.

TCP es más complejo que otros protocolos en la pila TCP/IP, su código constituye el 50% de todo el peso del código de LWIP [52]. El protocolo TCP, es de vital importancia en el SWE, ya que utiliza un método de transporte confiable mediante el protocolo HTTP.

Se implementó un Socket donde se utiliza el concepto de primitivas. Las primitivas son genéricas. Las primitivas requeridas deben ser inicializadas, lo que se realiza

por medio de funciones específicas proporcionadas por la biblioteca LWIP. En nuestro caso se emplea LWIP_TCP.h.

En la Tabla 3. 2, se puede apreciar el resumen de las primitivas para el procesamiento de paquetes TCP.

Tabla 3. 2. Primitivas TCP utilizadas. Referencia: [52].

Primitiva:	**Función:**
tcp_new()	Devuelve una estructura llamada PCB que inicializa un socket sin ser configurado.
tcp_bind()	Realiza un enlace de la estructura PCB, con un puerto y una dirección IP determinada. El puerto seleccionado es el 80, y será aceptada cualquier dirección IP a ese puerto.
tcp_arg()	Paso de argumentos a funciones de llamado.
tcp_listen()	Configura la estructura PCB para trabajar como servidor. Eso quiere decir que aceptará peticiones de conexión provenientes de los clientes. Según la configuración previa de la primitiva Bind, los clientes podrán poseer cualquier dirección IP, los servicios aceptados será por el puerto 80 (HTTP) y 5001 (IPERF)
tcp_accept()	Especifica cual función o evento será ejecutado inmediatamente después que se acepte una solicitud de conexión desde un cliente cualquiera (IPERF o HTTP).

El proceso de envío recepción mediante TCP se puede visualizar en la Fig. 3. 12.

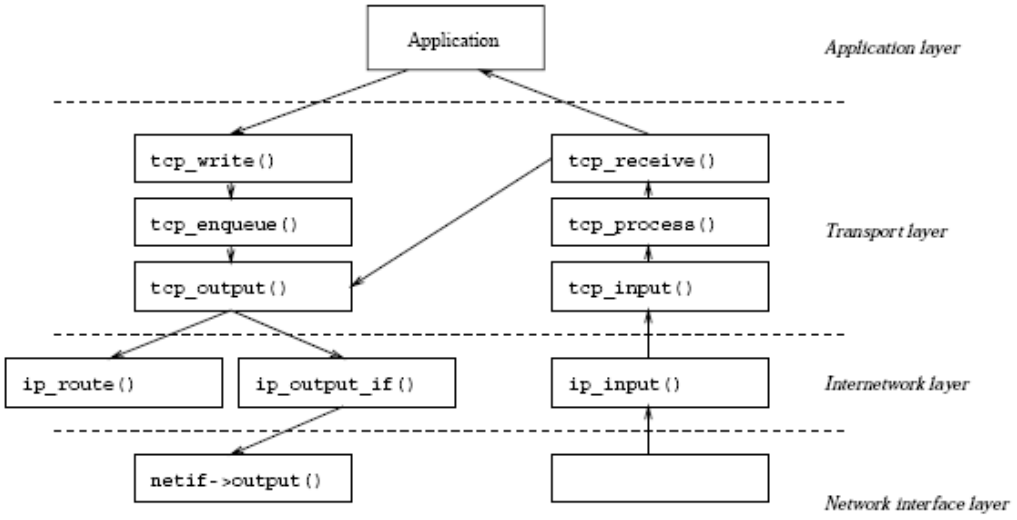


Fig. 3. 12. Procesamiento TCP. Referencia: [54].

Para cada conexión TCP se necesita una estructura PCB TCP, la estructura PCB contiene; variables, subestructuras y punteros, necesarias para el manejo del protocolo, en la Fig. 3. 13, se puede apreciar la estructura PCB TCP.

```
struct tcp_pcb {
struct tcp_pcb *next;
enum tcp_state state; /* TCP state */
void (* accept)(void *arg, struct tcp_pcb *newpcb);
void *accept_arg;
struct ip_addr local_ip;
u16_t local_port;
struct ip_addr dest_ip;
u16_t dest_port;
u32_t rcv_nxt, rcv_wnd; /* receiver variables */
u16_t tmr;
u32_t mss; /* maximum segment size */
u8_t flags;
u16_t rttest; /* rtt estimation */
u32_t rtseq; /* sequence no for rtt estimation */
s32_t sa, sv; /* rtt average and variance */
u32_t rto; /* retransmission time-out */
u32_t lastack; /* last ACK received */
u8_t dupacks; /* number of duplicate ACKs */
u32_t cwnd, u32_t ssthresh; /* congestion control var/
u32_t snd_ack, snd_nxt, /* sender variables */
snd_wnd, snd_wl1, snd_wl2, snd_lbb;
void (* recv)(void *arg, struct tcp_pcb *pcb, struct pbuf *p);
void *recv_arg;
struct tcp_seg *unsent, *unacked, /* queues */
*ooseq;
};
```

Fig. 3. 13. Estructura PCB TCP. Referencia [54].

3.4.2. Generalidades.

El servidor WEB es un servicio basado en HTTPV1.1 sobre TCP, posee únicamente un conjunto reducido de métodos basado en el procesamiento de mensajes almacenados en variables tipo carácter (char), receptados y almacenados en espacios de memoria PBUF, específicamente en el campo *payload*. Según la RFC2616 [30], HTTP es un extenso estándar con una amplia lista de funcionalidades, se ha optado por implementar los métodos: GET y POST, éstos brindan las funcionalidades básicas.

El diagrama de flujo de inicialización del SWE se puede apreciar en la Fig. 3. 14.

54 D. Adam. Design and Implementation of the lwIP TCP/IP Stack. (2001). Suecia: Swedish Institute of Computer Science. Recuperado el 2011 de http://www.sics.se/~adam/lwip/doc/lwip.pdf.

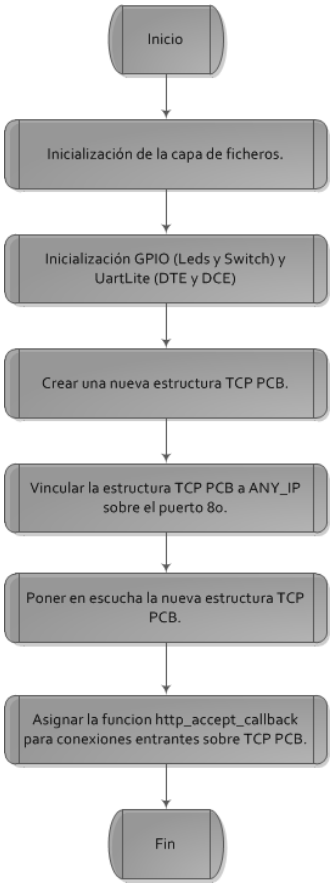


Fig. 3. 14. Diagrama de flujo inicialización del servicio WEB.

El SWE, es una aplicación vinculada al programa principal, está conformado por las funciones que se mencionan en la Tabla 3. 3.

Tabla 3. 3. Funciones del SWE.

Función.	Reside en:	Descripción:
decode_http_request()	http_response.c	Accede a la carga útil para decodificar el método HTTPv1.1.
do_404()	http_response.c	Genera y responde la cabecera y el cuerpo del mensaje HTTPv1.1 ante un recurso no encontrado.
do_http_get()	http_response.c	Accede al sistema de ficheros para extraer en porciones un recurso solicitado.
do_http_post()	http_response.c	Accede a los periféricos para generar una respuesta en JavaScript Object Notation (JSON) ante una solicitud HTTP POST de acceso a los periféricos.
extract_file_name()	web_utils.c	Extrae el nombre del recurso solicitado de una solicitud HTTP GET.
generate_http_header()	web_utils.c	Genera una cabecera de respuesta en formato HTTPV1.1.
generate_response()	http_response.c	Direcciona a funciones para el tratamiento de la carga útil del paquete.
get_adc_state()	platform_gpio.c	Inicia una conexión serial para recuperar los valores del módulo ADC.
get_file_extension()	web_utils.c	Extrae de la carga útil la extensión del fichero solicitado mediante el método HTTP GET.
get_switch_state()	platform_gpio.c	Recupera el valor del registro del GPIO switchs.

http_accept_callback()	webserver.c	Guarda un registro del número de conexiones y designa a las funciones: http_recv_callback() para la recepción de datos y http_send_callback() para el envío de datos.
http_recv_callback()	webserver.c	Recibe los datos enviados si se ha establecido una conexión TCP o existe una activa. Genera un ACK y llama a funciones para procesar la información recibida. Finalmente libera la estructura PBUF.
http_sent_callback()	webserver.c	Extrae datos de MFS y los envía en porciones mediante TCP, solo si se ha establecido una conexión.
is_cmd_adc()	web_utils.c	Devuelve un valor de verdadero si la solicitud HTTP contiene el comando de solicitud de lectura de uno de los canales del módulo ADC.
is_cmd_led()	web_utils.c	Devuelve un valor de verdadero si la solicitud HTTP contiene el comando de solicitud de escritura de GPIO leds.
is_cmd_switch()	web_utils.c	Devuelve un valor de verdadero si la solicitud HTTP contiene el comando de solicitud de lectura de GPIO switchs.
platform_init_gpios()	platform_gpio.c	Establece valores iniciales de operación del sistema para los periféricos.
start_web_application()	webserver.c	Inicializa el sistema de ficheros y comprueba los periféricos. Crea una nueva estructura PCB TCP y la vincula al puerto 80, donde permanece en escucha. Por último designa a la función http_accept_callback() para atender trafico entrante a la estructura PCB TCP.
toggle_leds()	platform_gpio.c	Escribe en el registro del periférico GPIO leds un valor recibido en la carga útil.

El código del módulo webserver.c y su cabecera webserver.h, se encuentran especificados en el Anexo 3.

3.4.3. Decodificación del método HTTP.

Una vez que el paquete ingresa a la interfaz de red, el paquete TCP es alojado en una estructura PBUF, el manejo de este paquete lo realiza internamente LWIP. Según HTTPV1.1 el encabezado que envía el usuario contiene básicamente una cadena de caracteres en formato JSON que inicia con el método HTTP [30].

Mediante la función http_rec_callback() se asegura que la conexión se encuentre en estado ESTABLISHED [54], luego se procede a llamar a la función generate_response(), al ser invocada necesita ser vincula a una estructura PCB TCP, lo primero que se hace es acceder al texto de requisición y decodificar el método mediante la función decode_http_request(), la misma invocará a una de las funciones que se mencionan a continuación, según sea el caso:

- do_http_get(), si la función decode_http_request() ha devuelto HTTP_GET.
- do_http_post(), si la función decode_http_request() ha devuelto HTTP_POST.
- do_404(), si no se ha reconocido el método dentro de la requisición del usuario.

En la Fig. 3. 15, se puede observar el diagrama de flujo para decodificar el método HTTP ubicado en la carga útil del paquete de entrada.

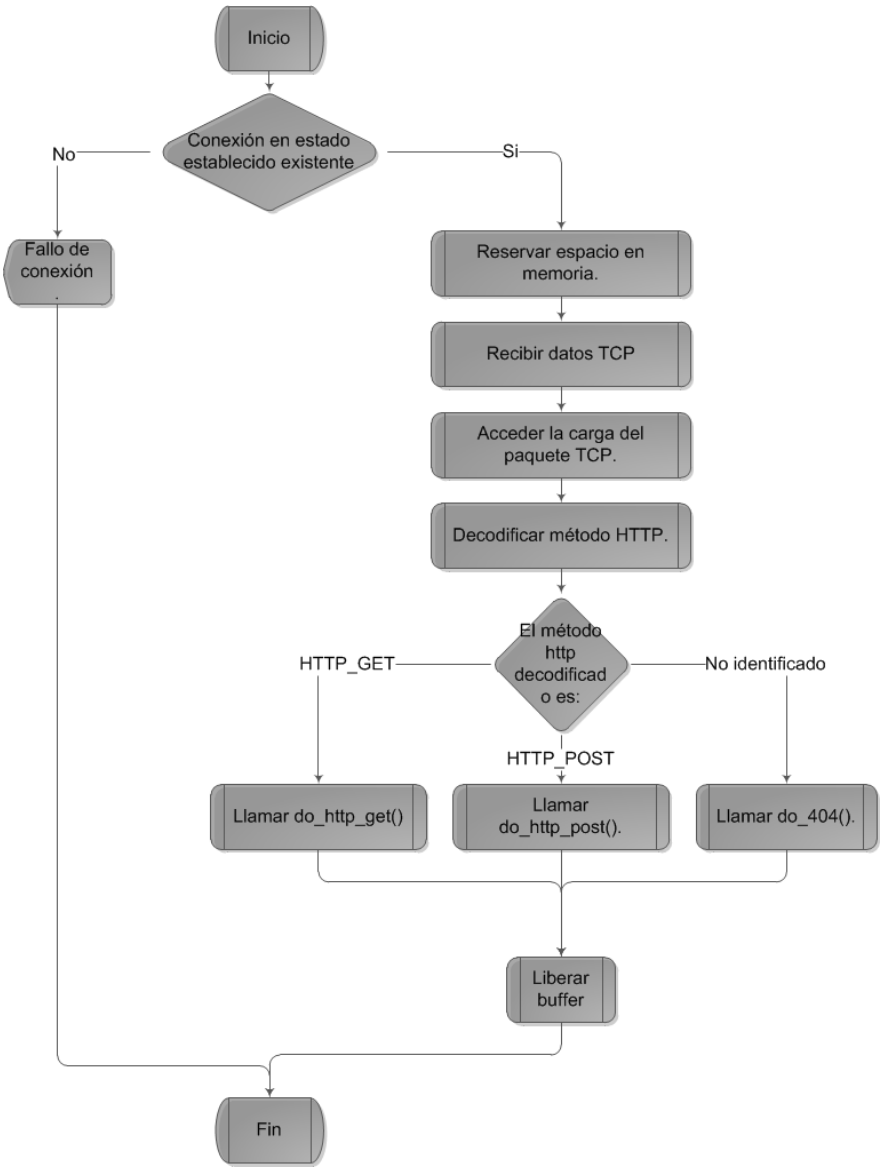


Fig. 3. 15. Decodificación de método HTTP.

3.4.4. Programación del método HTTP GET.

El método GET es enviado por el usuario dentro de un mensaje HTTP y permite obtener un recurso HTTP, cualquier archivo contenido en la imagen MFS puede ser accedido mediante este método.

El método HTTP GET está vinculado a la función do_http_get(). Las funciones relacionadas con la operación y acceso al sistema de ficheros se encuentran especificadas en el módulo platform_fs.c y la cabecera platform_fs.h. Una vez que se ha decodificado el método se sigue el diagrama de flujo expuesto en la Fig. 3. 16, para enviar al cliente el recurso solicitado.

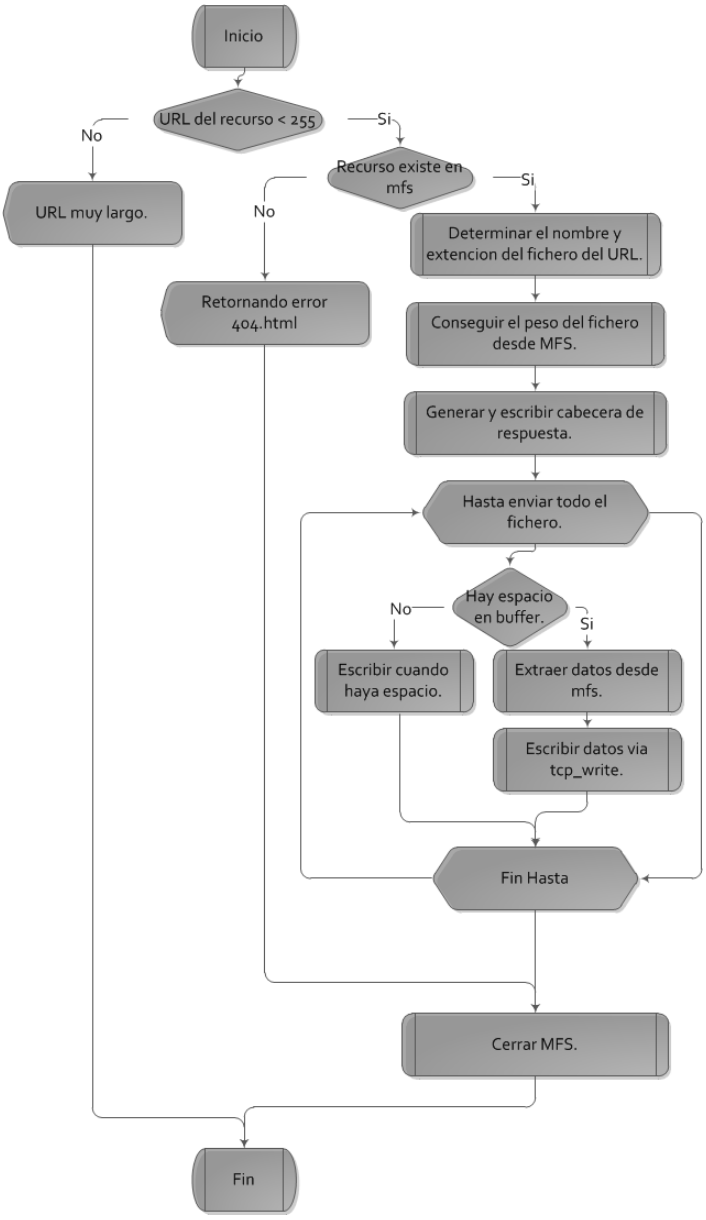


Fig. 3. 16. Flujograma método HTTP GET.

3.4.5. Programación del método HTTP POST.

El servicio HTTP POST se utiliza en conjunto con las funcionalidades JavaScript y Yui2 de Yahoo, para generar una conexión asíncrona para acceder a los dispositivos.

Una vez que se ha decodificado el método contenido en la carga útil, se hace uso de la función do_http_post() para atender a la solicitud de este método. Al iniciar la función se define un buffer de 1024 caracteres para la transmisión de las cadenas JSON. El llamado a las funciones is_cmd_led(), is_cmd_switch() o is_cmd_led(), retornan un valor lógico en caso de que el mensaje HTTP contenga una solicitud de obtener el valor de los switchs (cmd/switchxhr), escribir los leds (cmd/ledxhr) o leer el valor del ADC(cmd/adc). En la Fig. 3. 17, se muestra el algoritmo para el procesamiento de método de solicitud POST.

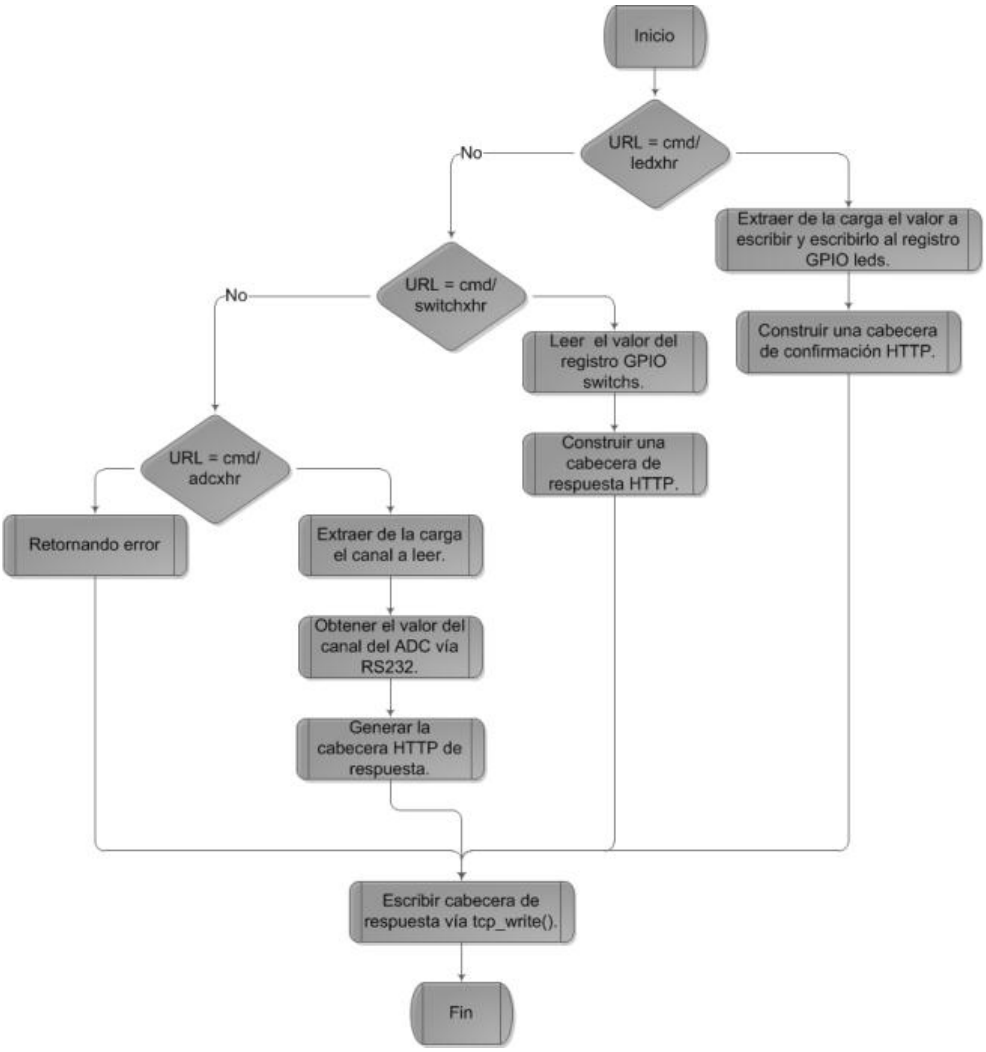


Fig. 3. 17. Flujograma método HTTP POST.

En la Fig. 3. 18, puede observar la respuesta del servidor ante la solicitud de interactuar con un dispositivo.

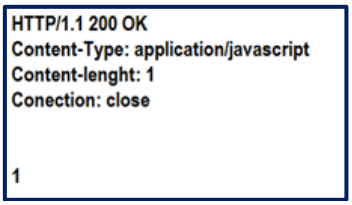


Fig. 3. 18. Mensaje de respuesta ante una solicitud POST SWITCH.

3.4.6. Sistema de ficheros en memoria (MFS).

La imagen del sistema de archivos, es una colección de los archivos que son utilizados por el SWE y que son accesibles al cliente. Cada recurso posee una ubicación exacta en el sistema que puede ser accedido mediante su URL.

Toda petición al servidor genera una repuesta HTTP, que consta del campo protocolo, seguido del código de estado y el tipo de contenido (/text/html, imagen/gpg, imagen/ico, aplication/json, aplication/javascript, aplication/pdf, text/css o text/plain), la longitud y el contenido del recurso. Por ejemplo si se solicita la página inicial de la aplicación el dialogo entre cliente y servidor se lo puede apreciar en la Tabla 3. 4.

Tabla 3. 4. Dialogo entre cliente y servidor mediante el método GET.

Dirección:	**Mensaje:**
Cliente - Servidor	GET http://192.168.1.10/index.htm HTTP/1.1 Host: mozillaorg
Servidor - Cliente	HTTP/1.1 200 OK Content-Type: text/html Content-Length: 105 <HTML> <BODY> Web embebido! </BODY> . . . </HTML>

Los recursos URL que se encuentran en una petición y que no se encuentren en el MFS de la memoria DDR SDRAM generan un código de error 404 (Recurso no encontrado), el servidor responde con la versión del protocolo seguido del código de error (HTTP/1.1 404), según se define en la RFC2616 [57]. El servidor responderá con una página HTML que indica el error 404.

Los recursos accesibles vía HTTP GET pueden ser: imágenes JPG, gifs, texto plano, texto html, scripts de javascript. Todos los objetos poseen una URL y están

vinculados a una página HTML accesibles mediante objetos DOM desde los controles y formularios del documento HTML.

La estructuración de contenidos URL se la puede observar en la Fig. 3. 19.

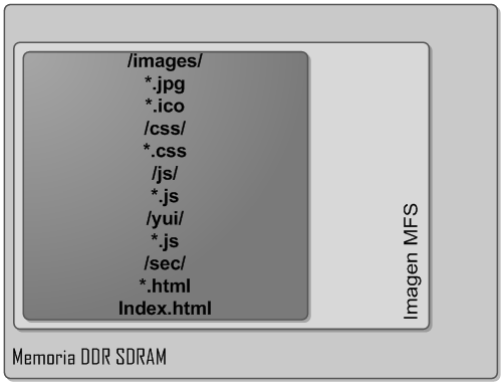


Fig. 3. 19. Estructuración recursos URL en MFS.

Los directorios se han distribuido de la siguiente manera:

- /images, está destinado al almacenamiento de imágenes de la página HTML.
- /css, directorio para el almacenamiento de las hojas de estilo.
- /js, aquí se encuentran los Scripts elaborados para el manejo de la página.
- /yui, se almacenan los Scripts propietarios de Yui2.
- /sec, aquí se guardan las secciones en formato HTML, necesarias para la operación de la página principal.

3.5. Programa servidor IPERF.

El servicio IPERF es una aplicación, que permite la medición del ancho de banda entre el cliente y el servidor. IPERF es una aplicación de punto a punto que envía una serie de paquetes desde el cliente al servidor, permitiendo conocer el rendimiento de la red, los tiempos de ida y vuelta ó Time to Life (TTL), paquetes perdidos, ancho de banda, entre otros [55].

La implementación comprende la interacción entre la interfaz física de red y la capa TCP. Debido a que IPERF hace uso del protocolo TCP, se hace necesario establecer una conexión TCP entre el cliente y el servidor mediante el puerto 5001. El servidor IPERF, necesita crear una nueva estructura PCB TCP, que se encuentra en escucha por cualquier conexión entrante en el puerto 5001, el servidor actúa

55 A. Tirumala, M. Gates, F. Qin, J. Dugan & J. Ferguson. *Iperf - The TCP/UDP band-width measurement tool*. (2009). Recuperado el 2011 de http://dast.nlanr.net/Projects/Iperf.

como un espejo que devuelve los paquetes inmediatamente después de su arribo, el tiempo de ida y vuelta el cliente los interpreta para generar las estadísticas de envío y recepción.

3.5.1. Inicialización.

La inicialización del servicio se lo realiza mediante el llamado a función de start_rxperf_application(). La función crea una nueva estructura PCB TCP, la estructura es ligada a un puerto (5001) y a una dirección IP mediante la función de la librería LWIP tcp_bind(). Una vez vinculada la estructura procedemos a poner a la nueva estructura en estado de escucha mediante la función tcp_listen(), por ahora la conexión TCP está en estado de escucha, por ende es necesario designar una función que maneje los datos cuando se soliciten una conexión por el puerto 5001, utilizamos la función tcp_accept(pcb, rxperf_accept_callback) para designar a la función rxperf_accept_callback() para el manejo de datos cada vez que ingresen datos a la nueva estructura PCB.

3.5.2. Procesamiento de datos.

La función rxperf_accept_callback() es la encargada de crear una nueva estructura PCB TCP, ante la solicitud de conexión de un cliente hacia el nuevo servicio, cuando se solicite una nueva conexión la estructura crea una nueva estructura PCB TCP denominada newpcb y designa a la función rxperf_recv_callback() para generar una respuesta hacia el cliente mediante la función lwip tcp_recv(newpcb, rxperf_recv_callback).

Lo primero que hace la función rxperf_recv_callback() es asegurarse que los datos sean válidos constatando que el campo buffer de la estructura newpcb no sea un campo nulo, en tal caso la conexión se cierra. En caso de contener datos simplemente envía el acuse de entregar, debido principalmente a que estos datos son simplemente generados por el cliente para medir la disponibilidad de la red. Finalmente se debe liberar el buffer destinado a la carga mediante pbuf_free() para que esté disponible para otros datos. La transmisión culmina cuando el cliente envía una carga útil del tipo NULL con lo cual se cerrará la conexión.

3.6. Generación del Link Script para aplicaciones.

En el desarrollo de software embebido, es también habitual que el desarrollador se tenga que preocupar del script de enlace, que determina cómo se mapean las aplicaciones en la memoria del sistema. Tras la compilación, el código embebido almacenado en un archivo con extensión *.elf*, ha de ser descargado en la plataforma objetivo. El código del sistema está diseñado para ser almacenado en diferentes zonas del mapa de memoria, es necesario escribir un Linker Script conteniendo las instrucciones necesarias para que se compile cada parte del código en el rango de direcciones correspondiente. En la Fig. 3. 20, puede apreciarse las diferentes dependencias de código a integrarse durante la generación del Link Script.

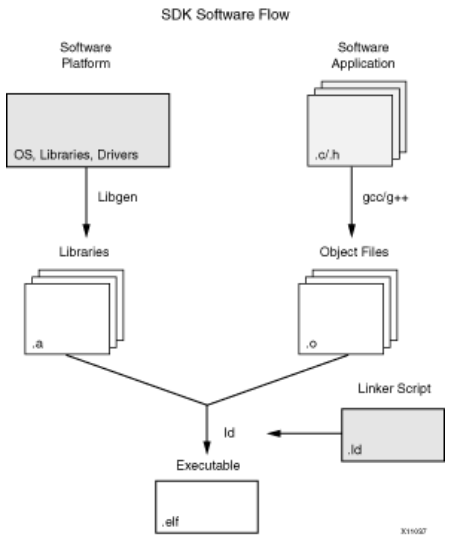


Fig. 3. 20. Flujo de proceso para generar el Link Script. Referencia: [56].

Para una correcta configuración del Link Script se debe tomar en cuenta los espacios reservados para el código y para los drivers, ya que estos toman valores fijos. Para agregar el Link Script se debe recurrir al menú *tools* de SDK 11.1, se debe configurar según se muestra en la Fig. 3. 21.

56 Xilinx Inc. *Building applications in SDK*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/concepts/sdk_c_build.htm.

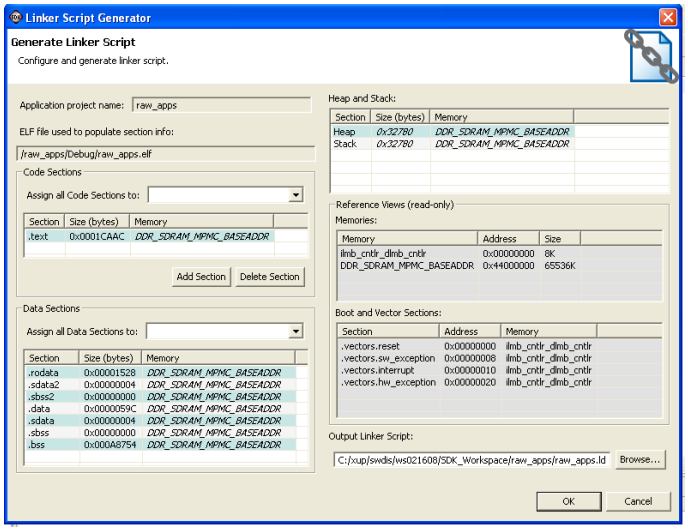


Fig. 3. 21. Configuración de Link Script en SDK 11.1. Fuente: SDK 11.1.

Existen varios parámetros de memoria a asignarse, se mencionan a continuación:

- Code section: representada por el archivo .text fue asignado a la memoria DDR SDRAM y contiene todo el código fuente de la aplicación.
- Data Sections: Contiene las secciones de datos del archivo ELF a generarse. Reside en la memoria DDR SDRAM
- Heap and stack: contiene las secciones de memoria para aplicaciones: estática (Heap) y dinámica (stack). Reside en la memoria DDR SDRAM.

Una vez que se han asignado los valores, se debe aceptar para que se genere el Link Script, la mayoría de errores en esta etapa se debe al desconocimiento del tamaño total del código, de no tener el espacio suficiente para las variables dinámicas o estáticas (heap o stack) o de haber asignado memoria reservada para los buses locales de instrucciones o datos.

3.7. Programación de la página principal del SWE.

El diseño de la página WEB es una tarea que involucra técnicas de programación en: HTML, CSS y JavaScript. Se lo ha realizado de esta forma, principalmente debido al recurso de memoria limitado para la imagen MFS, las hojas de estilos CSS permiten que se reduzca el código HTML. Se ha optado por este conjunto de soluciones ya que la página presentará características dinámicas.

El SWE utiliza Scripts de JAVA preconstruidos por Yahoo que se encuentran pre construidos en la biblioteca YUI2, estos permite realizar conexiones asíncronas y traspaso de datos en formato de cadenas de caracteres en formato JSON.

3.7.1. Diseño en HTML.

El código HTML es basado en reversión 4.01 [57] de este lenguaje y ha sido desarrollado en la herramienta Webuilder 2010 10.1 [58], éste ambiente posee utilidades que facilitan la programación en: html, css y javascript.

La página está segmentada en; clases, divisiones y formularios, para facilitar la agrupación de los objetos en categorías, las mismas permitirán asignar o extraer atributos de uno o varios objetos. Esta particularidad resulta muy importante cuando se trabaja con Scripts de Java, cada objeto o grupo de objetos se les ha asignado un id o clase que permite reconocer al objeto, acceder a él y establecer los atributos CSS mediante la filosofía DOM. En la Fig. 3. 22, puede apreciarse la página web visualizada en el navegador Firefox 5.0.

57 Networking Group. *HTML 4.01.* Recuperado el 2011 de http://www.w3.org/TR/html401/).

58 Bluementals. *Webuilder2010*. (2010). Recuperado el 2010 de http://www.blumentals.net/webuilder/

Fig. 3. 22. Página principal index.html. Fuente: Firefox 5.1.

3.7.2. DOM y Scripts de Yui2.

Una página WEB dinámica incorpora; efectos gráficos, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario [59].

YUI2, es un conjunto estandarizado de; conceptos, prácticas y criterios, JavaScript y CSS de código abierto que permite construir aplicaciones WEB con un rico contenido interactivo [60]. La mayor ventaja al trabajar con la biblioteca YUI2, es que se puede agregar Scripts según las necesidades de la aplicación, permitiendo reducir el tamaño que ocupa en la memoria. YUI2, ha sido ubicado en el directorio

59 E. Xavier. *Intoduccion a JavaScript*. (2009). Recuperada el 2011 de WWW.librosweb.es.

60 Yahoo. *YUI FAQ*. (2009). Recuperado el 2011 de http://yuilibrary.com/yui/docs/tutorials/faq/#what-is-yui.

/yui/ y se ha incluido únicamente los scripts básicos para su funcionalidad y la animación de los objetos DOM de la página WEB:

- yahoo.js: Configura y crea las primitivas de los nombres para el objeto YAHOO, para que pueda ser utilizado por la biblioteca YUI.
- Anim.js: Provee la autorización para agregar efectos a los objetos HTML.
- conn.js: Provee una interfaz sencilla a un objeto XMLHttpRequest, provee las funcionalidades de conexión y negociación con un servidor.
- dom.js: Provee funcionalidades de ayuda para el manejo de objetos DOM de una página HTTP.
- event.js: Posee funcionalidades parecidas al módulo dom.js, además de esto posee herramientas para la construcción de eventos del sistema.

Se ha desarrollado dos Scripts por separado que determinan el funcionamiento de los controles de la página, a continuación se los detalla:

- /js/main.js: Este script contiene todas las primitivas para establecer una conexión asíncrona con el servidor web para leer o escribir datos del mismo. Además permite animar algunos elementos HTML relacionados con la visualización de la página, también aloja el código del reloj digital que aparece en la parte superior derecha de la página.
- /js/ref.js: Gestiona el conjunto de elementos de enlace del menú con los contenidos que serán visualizados en el iframe de la página, además gestiona funcionalidades para la operación del envío de datos a los LEDS.

3.7.3. Generación y descarga de la imagen MFS.

La imagen es la representación física del sistema de archivos de toda la página WEB, que se guardará en la memoria del dispositivo y es accesible al usuario a través de un intérprete HTML. Todo el sistema de archivos debe ser agregado en la imagen para que esté disponible cuando el usuario así lo solicite. Para la creación de la imagen se hizo uso de la consola XMD del entorno XPS 11.1. Todo el sistema de ficheros debe ser ubicado en un solo directorio, ubicando index.html en su raíz, es recomendable hacerlo en el directorio c:/imagen.

Desde la consola XMD se recurre al comando mfsgen para generar una imagen valida con la sintaxis:

```
mfsgen –cvbfs ../imagen.mfs 3000 index.html js yui images css
```

Donde:

- mfsgen: es el llamado al programa generador de imagen.
- -cvbfs: c especifica que el contenido de la imagen se especificará en la línea de comandos, **v** indica que se especificara los contenidos en palabra completa, **b** especifica que se detallará el número de bloques de la imagen, **f** indica que se especificará el nombre de la imagen y **s** indica que se trabajará con formato de representación Big-Endian para los datos.
- ../imagen.fs: indica el nombre de la imagen y la ruta donde se guardará.
- 2000: es el número de bloques de la imagen y debe ser superior al peso del sistema de ficheros a incluir en la imagen.

El proceso de carga de una imagen a la tarjeta destino es muy sencilla, se debe recordar que la imagen debe ser descargada en una dirección de memoria válida, pudiendo estar en una dirección de memoria DDR o flash. En nuestro caso en concreto, se descarga en la memoria DDR SDRAM en su dirección de inicio (0x47000000).

Una vez descargado el archivo system.bit del sistema embebido se conectará con ISE 11.1 mediante el cable JTAG, ejecutando en consola XMD el comando:

connect mb mdm

Una vez conectado procedemos a descargar la imagen, ubicándose en el directorio donde se encuentra la imagen (imagen.mfs) navegando con la consola XMD al estilo clásico del símbolo del sistema y el comando:

dow -d imagen.mfs 0x47000000.

4. CAPÍTULO VI: RESULTADOS.

En el presente capítulo se detallan las pruebas aplicadas al SWE, además se hace un análisis de rendimiento completo del sistema.

4.1. Pruebas SWE y Servidor IPERF.

Es necesario efectuar la evaluación del sistema mediante pruebas de comunicación para medir factores como: comportamiento del prototipo, su estabilidad, su capacidad de carga máxima ante solicitudes múltiples y consumo energético.

A continuación se presentan las pruebas consideradas de mayor importancia, ya que involucran los aspectos de comunicación antes mencionados. Se pueden realizar otras pruebas al sistema, como la integración en alguna planta industrial para la supervisión de variables, sin embargo tal actividad de integración está fuera del alcance de este proyecto.

Se ha utilizado los siguientes recursos para realizar las pruebas de red:

- Mozilla FireFox 6.0 (Intérprete de datos HTML).
- 3 PCs Windows 7, que actúan como clientes.
- Tarjeta Spartan 3E con Firmware embebido (SWE).
- Switch Ethernet Cisco Catalyst 2050 10/100 Mbps.
- Cables de red Ethernet Cat. 5e directos ponchados con Cat. A.

Los dispositivos han sido ubicados como se detalla en la topología esquematizada en la Fig. 4. 1, los tres clientes y el SWE han sido conectados al switch Cisco para simular una topología de red típica.

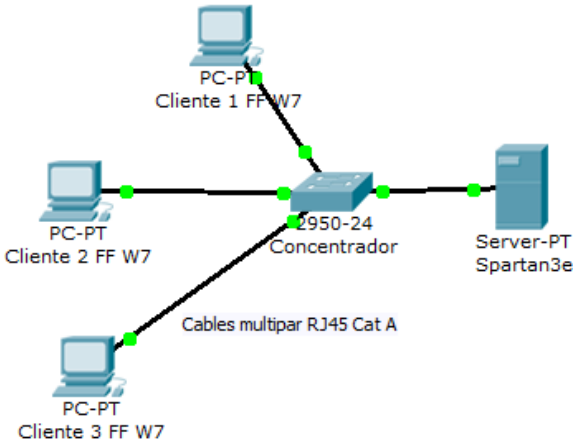


Fig. 4. 1. Esquema de pruebas para el SWE.

En la Tabla 4. 1, se listan las configuraciones de los parámetros TCP/IP realizadas a cada uno de los elementos de la topología de pruebas.

Tabla 4. 1. Configuración de dispositivos.

Dispositivo.	PHY (Mbps)	IP	Gateway	Mascara de red	Descripción.
PC-Cliente 1	100	192.168.1.15	192.168.1.200	255.255.255.0	Interprete Mozilla Firefox 5. SO W7.
PC-Cliente 2	100	192.168.1.16	192.168.1.200	255.255.255.0	Interprete Mozilla Firefox 5. SO W7.
PC-Cliente 3	100	192.168.1.17	192.168.1.200	255.255.255.0	Interprete Mozilla Firefox 5. SO W7.
Spartan 3E (SWE)	10	192.168.1.10	192.168.1.200	255.255.255.0	Servidor HTTP. Servidor IPERF.
Cisco Catalyst 2950	100	192.168.1.200	No aplica	255.255.255.0	Concentrador

4.1.1. Proceso de carga de la página HTML.

Mediante el uso del complemento FireBug 1.9.0 de Mozilla Firefox 6.0, se ha logrado comprobar que el tiempo de la carga de la página en condiciones normales de operación es de 7.46 s., con una carga total de 113.2 kB correspondientes a 23 solicitudes GET y POST. En la Fig. 4. 2, se puede mostrar los resultados en un diagrama de tiempo de la carga inicial de la página HTML.

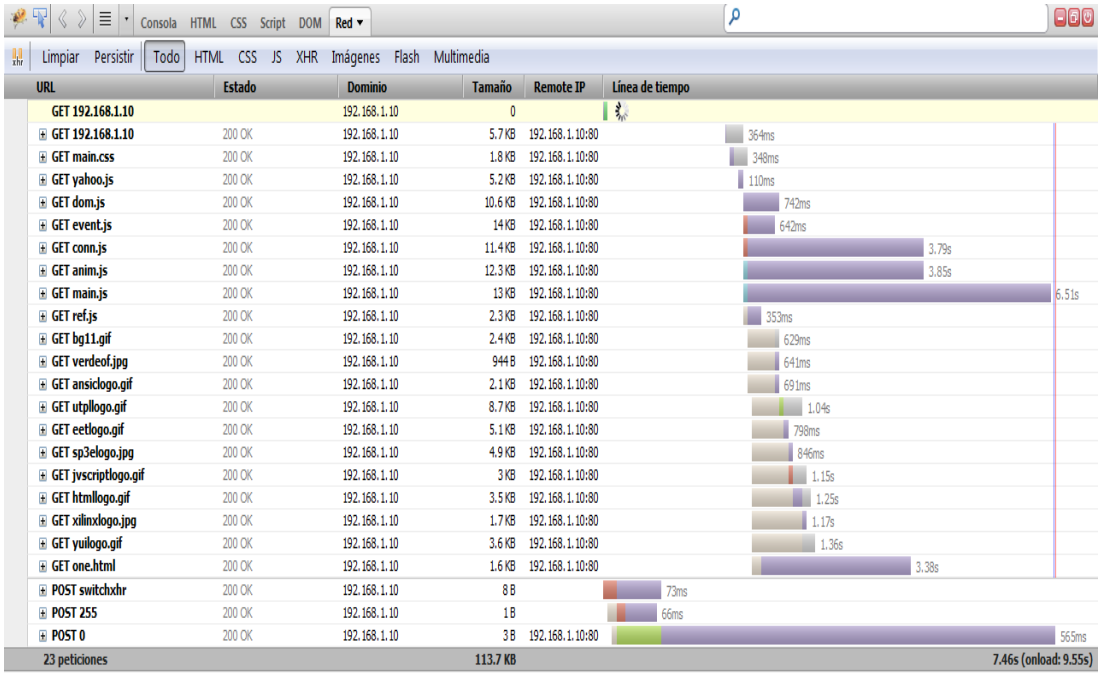


Fig. 4. 2. Línea de tiempo de carga de la página HTML principal. Fuente: Firebug v1.9.

4.1.2. Múltiples solicitudes.

Ante múltiples solicitudes, el servidor ha sido capaz de atender a 25 solicitudes simultáneas, degradando sus tiempos respuesta a medida que la carga de solicitudes se va incrementando, se ha observado una media de la degradación de los tiempos de respuesta de 400 ms por cada cliente conectado.

También se ha enviado 30 solicitudes de PING simultáneas, obteniendo una respuesta para el 80% de los paquetes enviados.

4.1.3. Ancho de banda mediante IPERF.

El servidor de rendimiento se lo ha ejecutado desde el cliente 1 mediante IPERF. El servidor se ha comportado de una manera estable y óptima para este tipo de sistemas. En la Fig. 4. 3, se puede apreciar los resultados de la prueba mediante IPERF. Se ha realizado 10 pruebas, de éstas se ha determinado que la media es de 500 kbps, las fluctuaciones se generan debido a servicios propios del sistema operativo del cliente que intentan acceder simultáneamente al servidor web.

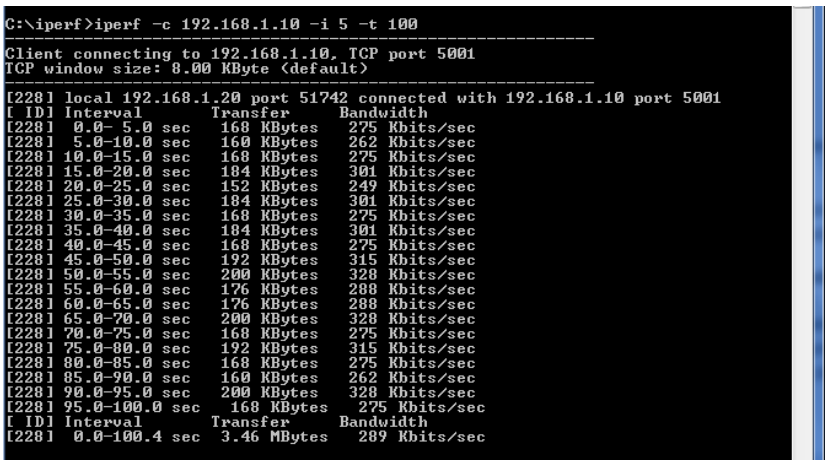


Fig. 4. 3. Medición de ancho de banda mediante IPERF.

Los resultados han sido satisfactorios para todas las pruebas realizadas. Esto se sustenta ya que en la prueba de conectividad se logró establecer una comunicación fiable y un comportamiento estable de envío y recepción de datos entre los tres clientes involucrados y el SWE. Los tiempos de respuesta del sistema en este caso son del orden de los milisegundos de 4 ms a 20 ms.

La prueba de la capacidad de carga máxima del servicio para enviar información a los clientes y evalúa su estabilidad en los procesos de alta demanda, se evidenció

un comportamiento estable dentro 20 peticiones. El tráfico a este nivel corresponde a 80 kbps, lo que establece la carga máxima del sistema.

4.2. Rendimiento de la plataforma de hardware.

Los resultados de rendimiento de consumo de potencia y de utilización de recursos fueron satisfactorios al hacer una comparativa con sistemas comerciales. En la Tabla 4. 2, se detalla el uso de los recursos del sistema basado en un FPGA con arquitectura reconfigurable.

Tabla 4. 2. Recursos del FPGA utilizados. Fuente: EDK 11.1

Utilización Lógica:	**Usada:**	**Disponible:**	**Utilización (%):**
Número de slice flip flops:	3395	9312	36%
Número de LUT'S de 4 entradas:	4255	9312	45%
Número de Slices ocupados:	3206	4656	68%

Mediante la utilización *Xpower Estimator (XPE) 11.1* [36] para la tarjeta Spartan 3E Starter Kit, se pudo obtener un estimado del consumo de potencia de la tarjeta de 0.384 W, en la Fig. 4. 4 se puede mostrar en más detalle los resultados.

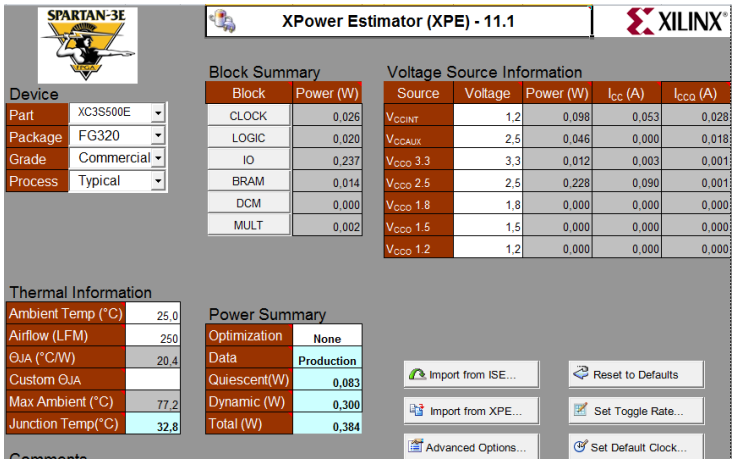


Fig. 4. 4. Estimación del consumo de potencia del sistema. Fuente: [36].

CONCLUSIONES.

- En el presente proyecto de fin de carrera se ha desarrollado un sistema embebido sobre la tarjeta Spartan 3E Starter Kit, accesible vía web con capacidades HTTP sobre TCP/IP, capaz de monitorear y controlar variables analógicas o digitales sobre un FPGA. El SWE puede adaptarse a cualquier sistema de control o monitoreo con una estabilidad operacional aceptable.
- El sistema cuenta con periféricos internos y externos accesibles a través de buses y protocolos de comunicación, con lo cual, se ha obtenido una integración completa con el Firm Core Microblaze para la captura y supervisión de variables del entorno, garantizando una arquitectura reconfigurable accesible desde la plataforma de aplicación.
- El prototipo posee capacidades HTTP para el transporte de datos además de una interfaz gráfica HTML que facilita la operación del mismo. El usuario puede acceder al sistema remotamente mediante cualquier intérprete HTML con capacidades JavaScript y CSS, y está en la capacidad de modificar y monitorear parámetros de bajo nivel de los periféricos del sistema mediante el uso de objetos DOM HTML como: botones, checkboxes, etc.
- Uno de los aportes de este proyecto está en la experiencia adquirida en el uso de FPGAs de Xilinx, la cual deberá ser considerada para realizar desarrollo de sistemas basados en comunicaciones Ethernet, ya que se comprendió su tendencia aplicativa en campos emergentes como: control y supervisión de procesos productivos, domótica, transporte, tele-medicina, posicionamiento global, servicios bancarios, información turística virtual, entre otros. Además, el aporte es un dispositivo embebido que podrá ser integrado a sistemas electrónicos con una gama amplia de requerimientos de control y/o monitoreo, debido principalmente a la naturaleza reconfigurable de las FPGAs y a la universalidad del lenguaje C. Un uso útil es adaptar la tarjeta como unidad de control y supervisión para estaciones meteorológicas.
- El Firmware desarrollado programado sobre el FPGA XC3S500E, ocupa el 47% de los recursos disponibles, por lo cual, posee el 53% de recursos para almacenar programación, se puede pensar en una futura incorporación de multiprocesamiento mediante la inclusión de otra instancia del Firm Core Microblaze.

- Al comprobar la alta capacidad de tecnologías embebidas basadas en los FPGAs, se concluye que, se puede implementar sistemas electrónicos complejos, al quedar demostrado cómo es factible implementar una arquitectura de un microcomputador mediante la incorporación de: un procesador Microblaze, memorias, contadores, protocolos de comunicación, etc. En comparación con otros dispositivos como los microcontroladores que tienen su capacidad de lógica limitada por las características físicas, que generan dificultades al momento de querer realizar una mejora o cambio en los procesos del dispositivo. En los FPGAs la lógica interna se puede adaptar según se requiera, reprogramando el microchip.
- Mediante las pruebas realizadas al SWE, se pudo comprobar la estabilidad de conexión ante 25 conexiones entrantes, una de carga máxima de 80 kbps ante solicitudes simultáneas al servicio. El servidor se comporta estable y presenta un buen funcionamiento cuando se generan peticiones de datos HTML con una cantidad igual o menor a ocho clientes y un factor de degradación promedio de 400 ms a medida que se incrementan los clientes.
- El cliente IPERF ha demostrado que el tráfico de datos en el servidor puede ser menor o igual a 500 kbps y que la cantidad de tráfico se ve degradada a medida que se conecten más clientes al sistema.

TRABAJOS FUTUROS.

El proyecto de tesis denominado "*Diseño e implementación de un servidor web embebido basado en una arquitectura reconfigurable con FPGAs para el control y monitoreo de periféricos*", abre un campo de aplicaciones para investigadores y estudiantes que están en la capacidad de implementar sistemas de telecomunicación basado en redes LAN o WAN para el control y supervisión de sistemas de la más variada gama:

- La facilidad de construcción de plataformas de hardware mediante la utilización de herramientas de diseño como el ISE de Xilinx, incentivarán el diseño de sistemas digitales complejos de alta capacidad de procesamiento basados en el Firm Core Microblaze.
- El alto nivel de síntesis lógica, bajo consumo de recursos y corto ciclo de implementación de los IPCORES, permitirán construir plataformas de hardware adaptables a sistemas industriales o académicos.
- El método de diseño seguido en el presente proyecto siembra las guías para el diseño de sistemas con arquitecturas reconfigurables a las necesidades del investigador. La separación de diseño entre hardware y software adiestran al lector para lograr un corto tiempo de diseño e implementación.
- La posibilidad de multiprocesamiento en el FPGA, debe ser uno de los siguientes pasos a seguir para lograr un sistema más rápido. El diseño de dos entidades Microblaze que trabaje en conjunto podría duplicar la capacidad de procesamiento del sistema.
- El bajo consumo de potencia del sistema permite visualizar un amplio campo de aplicaciones donde se necesita una alta capacidad de procesamiento con un bajo consumo energético, el lector estará en la capacidad de construir un sistema de telecomunicaciones basado en Ethernet con un bajo margen de consumo energético.
- La posibilidad de reconfiguración del SWE permitirá que se adapte a plataformas o arquitecturas con: iguales o mayores prestaciones de procesamiento y optimización de recursos físicos y/o lógicos.
- El alto nivel de portabilidad inter plataformas del lenguaje C, permitirán que el código pueda ser adaptado a cualquier arquitectura de hardware con pequeños cambios en sus primitivas y servicios. La modificación de la plataforma de códigos es completamente abierta con lo que se garantiza la

colaboración de futuros programadores que deseen reutilizar o mejorar el código.

- El uso del protocolo HTTP, en conjunto con HTML, CSS y JavaScript, proporcionan la flexibilidad al momento de crear las interfaces graficas HTML que se ejecutarán en el SWE. El lector podrá crear interfaces de acuerdo a sus necesidades sin afectar la capacidad de procesamiento de la capa de hardware.
- El paralelismo de los FPGAs, hace pensar en trabajos futuros como la coexistencia de un SWE y un DSP en un mismo dispositivo, para aplicaciones como: lectura de signos vitales remotamente, video vigilancia remota, control de consumo energético o control de acceso mediante voz.

REFERENCIAS BIBLIOGRÁFICAS

[1] W. Nicholas. *Designing an Embedded Web Server* (2000). U.S.A: Applied Computing Technologies. Recuperado el 2011 de http://pdf.cloud.opensystemsmedia.com/embedded-computing.com/USSoftware.Win00.pdf

[2] A. Miguel. Y J., Alvaro. *Servidor Web Embebido en una FPGA con Codiseño como Metodología de Diseño*. Eighth LACCEI Latin American and Caribbean Conference for Engineering and Technology (2010). "*Innovation and Development for the Americas*", Junio 1-4, 2010, Arequipa, Perú.

[3] Master Magazine. *Definición de microprocesador* (2011). Recuperado el 2011 de http://www.mastermagazine.info/termino/5881.php

[4] O. Juan y A. Vicente . *Arquitectura y programación de microcontroladores (2010)*. Valencia: Universidad de Valencia.

[5] C. Rorabaugh. *DSP Primer* (1999). USA: McGraw Hill.

[6] C. Maxfield. *The design warrior's guide to FPGAs.* (2004). USA: Mentor graphics Corporation y Xilinx Inc.

[7] Nextinning. *Technology & Semiconductor investment analysis*. (2010). Recuperado el 2011 de http://www.nextinning.com.

[8] Motorola Mobility Holdings. Inc. *Motorola Mobility Announces Fourth-Quarter and Full-Year 2010 Financial Results. (2010)*. Recuperado el 2011 de http://mediacenter.motorola.com/Press-Releases/Motorola-Mobility-Announces-Fourth-Quarter-and-Full-Year-2010-Financial-Results-359a.aspx.

[9] Freescale. *Sales and Support. (2011*). Recuperado el 2011 de http://www.freescale.com/webapp/sps/site/homepage.jsp?code=SUPPORTHOME&tid=FSH

[10] C. Luis. *Nueva familia de microcontroladores de 8 y 32 Bits FLEXIS de Freescale*. (2007). Recuperado el 2011 de http://www.bairesrobotics.com.ar/data/flexis.pdf

[11] P. John. Letter to Shareholders. (2010). Recuperado el 2011 de http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9NDE0NDExfENoaWxkSUQ9NDI2MDQ0fFR5cGU9MQ==&t=1

[12] Altera*. CPLD MAX* (2011). Recuperado el 2011 de http://www.altera.com/devices/cpld/max-about/max-about.html

[13] P. Mark. *Altera Launches Embedded Initiative with New System Level Integration Tool for Embedded Systems Configurability* (2010). Recuperado el 2011 de http://www.prnewswire.com/news-releases/altera-launches-embedded-initiative-with-new-system-level-integration-tool-for-embedded-systems-configurability-104763669.html

[14] Aarkstore Enterprise. *Altera Corporation - SWOT Analysis - Market Research Report On Aarkstore Enterprise.* (2011). Recuperado el 2011 de http://www.aarkstore.com/reports/Altera-Corporation-SWOT-Analysis-26748.html

[15] Atmel Corporation. *Atmel AVR Studio 5.0*. (2011). Recuperado el 2011 de http://www.atmel.com/dyn/products/tools_card.asp?source=cms&tool_id=17212

[16] Xilinx Inc. *Products*. (2011). Recuperado el 2011 de http://www.xilinx.com/products/index.htm

[17] McG. Dylan. *Xilinx beats estimates on record annual sales.* (2011). Recuperado el 2011 de http://www.eetimes.com/electronics-news/4215542/Xilinx-beats-estimates-on-record-annual-sales

[18] Xport AR. *XPort AR Embedded Processor Module*. (2011). Recuperado el 2011 de http://www.lantronix.com/pdf/XPort-AR_PB.pdf

[19] Atmel Corporation. *AVR460: Embedded Web Server*. (2011). Recuperado el 2011 de http://www.atmel.com/dyn/resources/prod_documents/doc2396.pdf

[20] McPros. *Microchip PIC Ethernet Board w/ RS232 & Web-Based Configuration*.(2011). Recuperado el 2011 de http://microcontrollershop.com/product_info.php?cPath=98&products_id=893

[21] Cisco Systems. *Cisco Router Web SetUp Tool*. (2011). Recuperado el 2011 de http://www.cisco.com/en/US/products/sw/netmgtsw/ps2076/index.html

[22] Dlink. *D-link solutions*. (2011). Recuperado el 2011 de www.dlink.com

[23] Axis Communications. *Camaras de red Axis 211*. (2011). Recuperado el 2011 de http://www.axis.com/es/products/cam_211/accessories.htm

[24] Domo Desk. *Dd-7122 Camara Ip Motorizada Wifi Zoom Optico X10 Dual Codec / Dual Stream.*(2011). Recuperado el 2011 de http://www.domodesk.com/product/450/14/17/1/CAMARA_IP_MOTORIZADA_WiFi_ZOOM_OPTICO_x10_DUAL_CODEC_DUAL_STREAM.htm

[25] Domo Desk. *Dd-6310 Ip Motor Kit Control Por Internet*. (2011). Recuperado el 2011 de http://www.domodesk.com/product/20/14/37/1/KIT_CONTROL_IP_MOTOR_por_INTERNET.htm

[26] Superrobotica. *Siteplayer Microcontrolador Con Servidor Telnet S310268*. (2011). Recuperado el 2011 de http://www.superrobotica.com/S310268.htm

[27] C. Jerry, G. Nupur, M. Jayant y R. David. *Design Methodologies for Core-Based FPGA Designs*. (2011). Recuperado el 2011 de www.xilinx.com/

[28] H. Ju, M. Choi, y J. Hong. *EWS-Based Management Application Interface and Integration Mechanisms for Web-Based Element Management.* (2001). Journal of Network and Systems Management, Vol. 9, No. 1, pp. 31-50.

[29] T. Andrew. *Redes de computadoras. 4ta Ed*. (2003). U.S.A: Prentice Hall.

[30] Network Working Group. *Hypertext Transfer Protocol -- HTTP/1.1*.(1999). Recuperado el 2011 de http://www.w3.org/Protocols/rfc2616/rfc2616.html

[31] B. Fernando, C. Francisco y C. Juan (2005). *Desarrollo Profesional de Aplicaciones Web con ASP.NET*. Recuperado el 2011 de http://csharp.ikor.org/

[32] Networking Group. *HTML5 A vocabulary and associated APIs for HTML and XHTML*. (2011). Recuperado el 2011 de http://www.w3.org/TR/html5/

[33] W3Schools. *JavaScript Introduction*. (2010). Recuperado el 2011 de http://www.w3schools.com/js/js_intro.asp

[34] Networking Group. *CSS 2.1.* (2009) . Recuperado el 2011 de http://www.w3.org/TR/CSS21/

[35] Xilinx Inc. *Xilinx Embedded System Tools Reference Guide 11.3.1.* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf

[36] Xilinx Inc*. XPower Estimator User Guide*. (2011). Recuperado el 2011 de http://www.xilinx.com/support/documentation/user_guides/ug440.pdf

[37] Digilent Inc. *Spartan-3E Starter Kit Board User Guide*. (2006). Recuperado el 2011 de http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

[38] Xilinx Inc. *Spartan-3E FPGA Family*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[39] Atmel Corporation. *ATmega32*. (2011). Recuperado el 2011 de www.atmel.com

[40] Xilinx Inc. *MicroBlaze Processor Reference Guide for EDK 10.1.* (2008). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[41] Xilinx Inc. *Local Memory Bus (LMB) v1.0 (v1.00a).* (2005). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/lmb.pdf

[42] Xilinx Inc. *Processor Local Bus (PLB) v4.6 (v1.04a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/ds531.pdf

[43] Xilinx Inc. *LMB BRAM Interface Controller (v2.10b).* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/lmb_bram_if_cntlr.pdf

[44] Xilinx Inc. *Multi-Port Memory Controller (MPMC) (v5.00.a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf

[45] Xilinx Inc. *Block RAM (BRAM) Block (v1.00a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf

[46] Xilinx Inc. *XPS UART Lite (v1.01a).* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/xps_uartlite.pdf

[47] Xilinx Inc. *XPS Ethernet Lite Media Access Controller (v2.01a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v2_01_a/ds759_axi_ethernet.pdf.

[48] Xilinx Inc. *XPS General Purpose Input/Output (GPIO) (v2.00a*). (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/xps_gpio.pdf

[49] S. William. *Organización y Arquitectura de computadores 7ma Ed*. (2006). U.S.A: Prentice Hall.

[50] Xilinx Inc. *XPS Interrupt Controller (v2.00a)*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/ip_documentation/dcr_intc.pdf

[51] Xilinx Inc. *OS and Libraries Document Collection.* (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf

[52] D. Adam. Raw TCP/IP interface for lwIP. (2001). Suecia: Swedish Institute of Computer Science. Recuperado el 2011 de http://lwip-avr.googlecode.com/svn-history/r2/trunk/lwip/doc/rawapi.txt

[53] Networking Group. *Protocolo De Control De Transmisión RFC0793*. (1981). Recuperado el 2011 de http://www.rfc-es.org/rfc/rfc0793-es.txt.

[54] D. Adam. Design and Implementation of the lwIP TCP/IP Stack. (2001). Suecia: Swedish Institute of Computer Science. Recuperado el 2011 de http://www.sics.se/~adam/lwip/doc/lwip.pdf

[55] A. Tirumala, M. Gates, F. Qin, J. Dugan y J. Ferguson. *Iperf - The TCP/UDP band-width measurement tool*. (2009). Recuperado el 2011 de http://dast.nlanr.net/Projects/Iperf

[56] Xilinx Inc. *Building applications in SDK*. (2009). Recuperado el 2011 de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/SDK_doc/concepts/sdk_c_build.htm

[57] Networking Group. *HTML 4.01. (2011).* Recuperado el 2011 de http://www.w3.org/TR/html401/

[58] Bluementals. *Webuilder2010*. (2010). Recuperada el 2010 de http://www.blumentals.net/webuilder/

[59] E. Xavier. *Intoduccion to JavaScript*. (2009). Recuperada el 2011 de www.librosweb.es

[60] Yahoo. *YUI FAQ*. (2009). Recuperada el 2011 de http://yuilibrary.com/yui/docs/tutorials/faq/#what-is-yui

ANEXOS

ANEXO 1

CREACIÓN Y CONFIGURACIÓN DE LA PLATAFORMA DE HARDWARE DE UN SISTEMA EMBEBIDO BASADO EN MICROBLAZE CON EDK 11.1

AVISO: Las instrucciones y gráficos mostrados en la presente guía han sido creados bajo un ambiente Windows XP, en el Software EDK 11.1 de Xilinx.

1. Ingreso a la plataforma EDK, Empezamos haciendo doble click en el icono EDK 11.1 de Xilinx, al cargarse el programa, se mostrará una ventana que nos dará 3 opciones para empezar a desarrollar nuestro sistema o a su vez continuar con el desarrollo de uno previamente guardado. Como se puede observar en la Fig. 1.

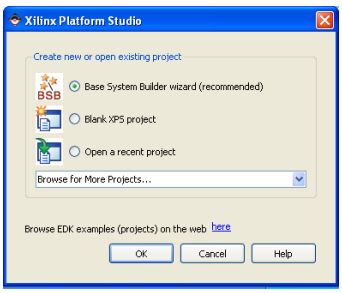


Fig. 1. Ventana de inicio de creación o edición de Sistemas embebidos. Fuente: EDK 11.1.

DESCRIPCION: La primera opción nos da la posibilidad de crear un sistema embebido nuevo con un asistente de diseño (Base System Builder wizard), la segunda opción nos da la posibilidad de crear una plantilla XPS en blanco por lo general se recomienda esta opción para usuarios con un conocimiento medio (Blank XPS project) y la última opción nos permite abrir un sistema reciente previamente guardado. Opcionalmente se puede seleccionar el cuadro de lista del final para buscar proyectos XPS que se encuentran en otras ubicaciones. También podemos hacer doble click desde cualquier carpeta en un proyecto XPS para llamar al EDK.

2. Configuración del directorio de Trabajo, empezamos a trabajar con la primera opción Base System Builder Wizard, aceptamos, la siguiente ventana nos pide 2 parámetros muy importantes la ruta donde se guardarán todos los archivos necesarios para el desarrollo del sistema embebido y también nos permite detallar si existe un repositorio de periféricos (Set Project Peripheral Repositories). El directorio donde se guardará el sistema de archivos del sistema embebido debe ser destinado únicamente para este propósito y no se debe compartir con otros

proyectos, hacemos click en el botón browser, ubicamos el directorio y hacemos click en aceptar del cuadro de búsqueda, una vez determinado el directorio hacemos click en OK, como se aprecia en la Fig. 2.

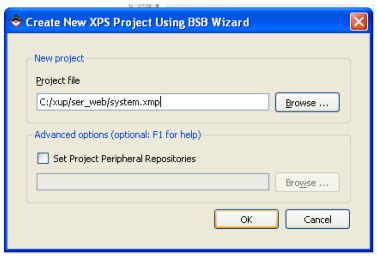


Fig. 2. Configuración de repositorios y ruta de alojamiento del proyecto. Fuente: EDK 11.1.

3. **Inicio de la creación del sistema, u**na vez hemos elegido el directorio de trabajo de todo el sistema, se muestra la ventana del asistente, en la parte superior nos irá mostrando una barra de progreso de la creación de todo el sistema embebido (Welcome, Board, System, Processor, Peripheral, cache, aplication y summary). Todas estas etapas son secuenciales, en su transcurso e irán detallando de manera sencilla los parámetros del sistema embebido.

4. **Primera Etapa (Welcome),** En la primera etapa (Welcome), no pide determinar si crearemos un nuevo diseño (I Would Like to créate a new design) o simplemente trabajaremos en base a una plantilla de ajustes de previamente diseñada (I Would like to load an existing .bsb setting file), en este caso se tendrá que detallar la ruta de ubicación del archivo de configuración .bsb y el asistente de creación configurará el sistema como lo indica el archivo .bsb, simplemente el usuario deberá ir aceptando las configuraciones hasta llegar al final del asistente. En nuestro caso elegiremos crear un nuevo diseño (I Would like to créate a new design), como se muestra en la Fig. 3.

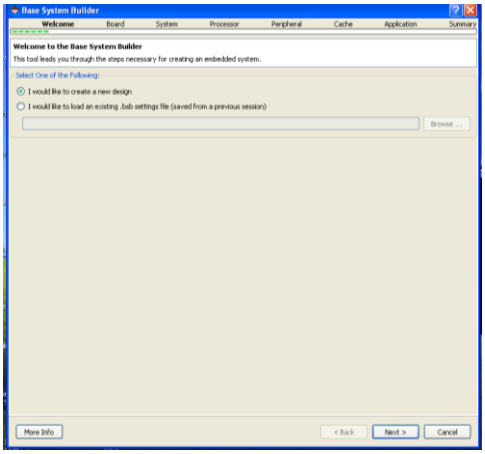


Fig. 3. Creación de un Nuevo diseño. Fuente EDK 11.1.

5. **Segunda etapa (Tipo de tarjeta),** Proseguimos a la siguiente etapa (Board) mediante el botón next, la siguiente etapa nos permite detallar las características del dispositivo a utilizar puede ser un board de xilinx (I would like to créate a system for the following development board) o un board creado a medida con un FPGA Xilinx (I would like to créate a system for a custom board). Se trabajará con la tarjeta Spartan 3E starter kit Rev. D, por lo tanto detallamos los parámetros que se muestran en la Fig. 4.

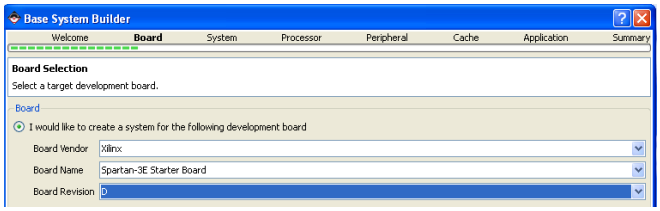


Fig. 4. Configuración del Board Xilinx Spartan 3E Starter Kit Rev.D. Fuente EDK 11.1.

6. **Tercera Etapa**, la siguiente fase (System), nos permite especificar el número de Procesadores en la FPGA: single-processor System (Un procesador Microblaze) y Dual-Processor System (Dos Procesadores Microblaze). Crearemos un sistema con un solo procesador Microblaze y continuamos con la siguiente etapa processor.

7. **Cuarta Etapa,** en la etapa processor se debe detallar características que regirán el comportamiento del mismo así detallaremos la velocidad del reloj (Reference

clock Frequency) que es la velocidad a la que trabajará todo el sistema, también el tipo de procesador Microblaze (Processor Type), detallamos también el reloj del sistema (System Clock Frequency) y el tamaño de memoria local, tal como se muestra en la Fig. 5.

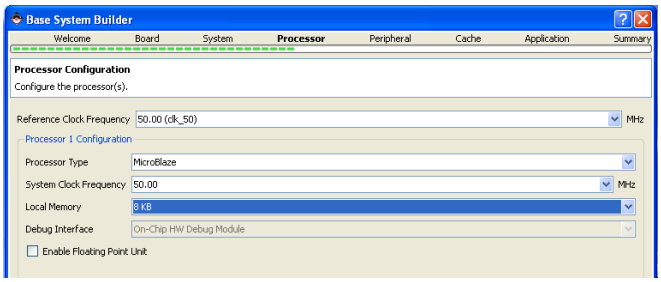


Fig. 5. Configuración de Microblaze. Fuente EDK 11.1

8. Las siguiente etapa son los periféricos, por ahora únicamente dejaremos los siguientes periféricos agregando (Add) o quitando (Remove) los que no sean necesarios, estos son: Buttons 4 bits, DDR SDRAM, Ethernet MAC, Leds 8 bits, RS232 DCE, RS232 DTE, dlmb cntlr, ilmb cntlr y xps_timer_0, continuamos con la etapa de cache, asignamos un cache de 4 KB en DDR SDRAM para instrucciones y datos como se puede observar en la Fig. 6 y Fig. 7.

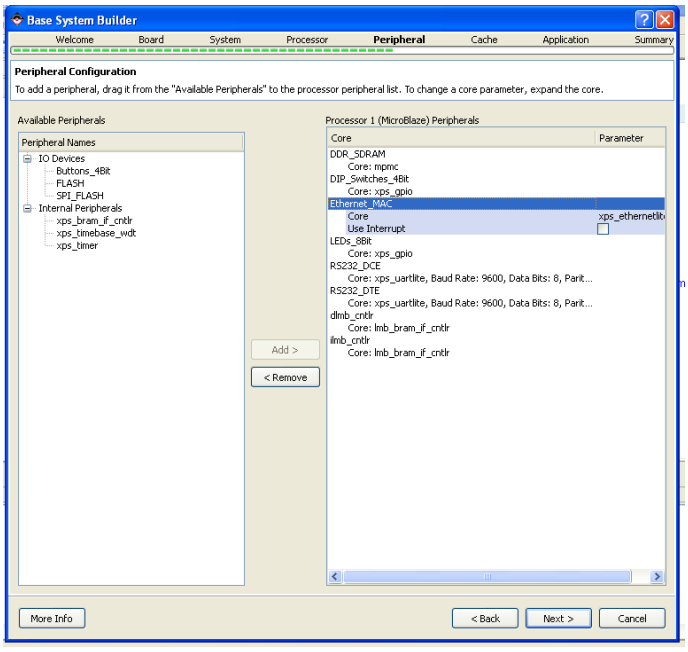


Fig. 6. Configuración de periféricos para Microblaze. Fuente EDK 11.1

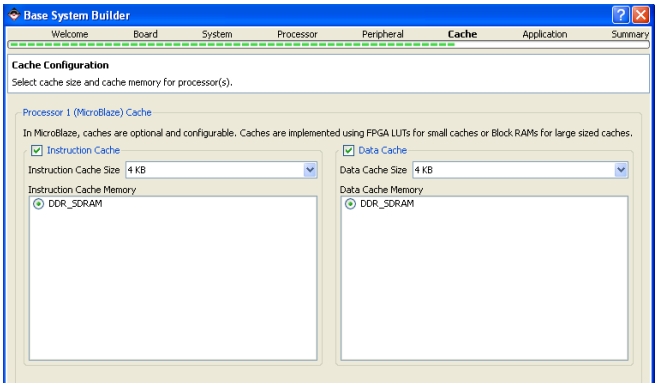


Fig. 7. Configuración de cache de instrucciones y datos para Microblaze. Fuente EDK 11.1

Las 2 etapas de application y summary dejaremos con los valores por defecto y procederemos hasta que el asistente finalice. Una vez terminado el asistente (Wizard), escogeremos la opción start using platform studio para terminar la configuración de Microblaze. En la pestaña de system assembly view entramos a la ventana de configuración del IPCORE Microblaze haciendo click derecho en Microblaze_0 y click derecho en configure IP. Configuramos como se muestra en la Fig. 7:

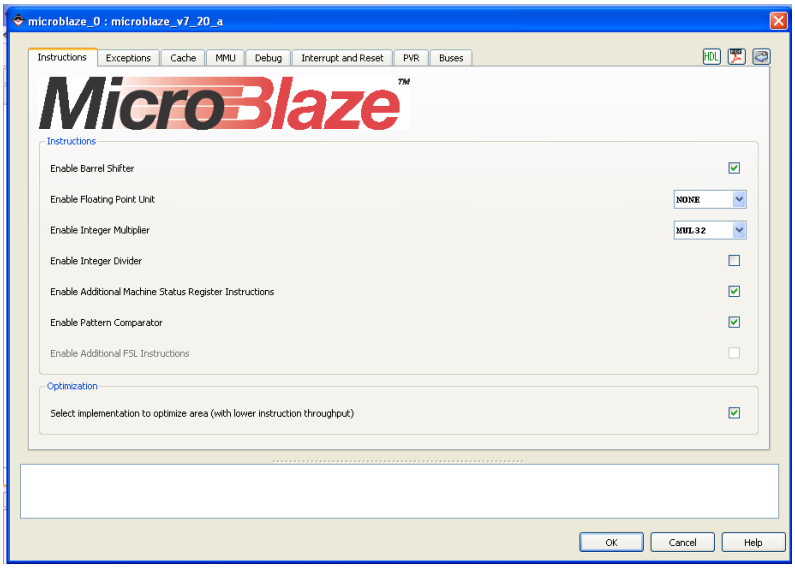


Fig. 8. Configuración IPCORE Microblaze. Fuente EDK 11.1

Hemos modificado únicamente Enable Barrel Shifter, con esto permitimos que se optimicen las aplicaciones que se ejecutaran en el procesador. En esta ventana también podemos cambiar los tamaños de la memoria caché en la pestaña *caché*.

9. Procedemos a configurar del IPCORE MPMC con un bus dual xcl sobre un xcl pim en el archivo .mhs. Para este propósito bastará con incluir o modificar las siguientes líneas en el archivo System.mhs del proyecto.

```
BEGIN mpmc
.
.
PARAMETER C_NUM_PORTS = 1
PARAMETER C_MEM_DM_WIDTH = 2
PARAMETER C_MEM_DQS_WIDTH = 2
PARAMETER C_PIM0_BASETYPE = 1
PARAMETER C_XCL0_B_IN_USE = 1
BUS_INTERFACE XCL0 = microblaze_0_IXCL
BUS_INTERFACE XCL0_B = microblaze_0_DXCL
.
.
End
```

10. Se deberán configurar también los puertos y la interconexión de redes para las interrupciones en el ARCHIVO system.mhs. Como se detalla a continuación:

- **Configuración de la entidad Ethernet Little.**

```
BEGIN xps_ethernetlite
.
.
PORT IP2INTC_Irpt = Ethernet_MAC_IP2INTC_Irpt
END
```

- **Configuración de la entidad XPS Timer:**

```
BEGIN xps_timer
.
.
.
PORT Interrupt = xps_timer_0_Interrupt
END
```

- **Configuración de la entidad XPS_INTC.**

```
BEGIN xps_intc
```

.

.

.

PORT Intr = xps_timer_0_Interrupt & Ethernet_MAC_IP2INTC_Irpt

PORT Irq = microblaze_0_Interrupt

END

Configuración de Microblaze.

BEGIN microblaze

.

.

.

PORT INTERRUPT = microblaze_0_Interrupt

END

ANEXO 2

Programa principal módulo main.c.

```
#include <stdio.h>

#include "xparameters.h"

#include "netif/xadapter.h"

#include "platform.h"
#include "platform_config.h"
//funciones prototipo
void print_headers(); //impresión de cabeceras a consola
int start_applications();//arranque de aplicaciones
int transfer_data();//Atiende peticiones de respuesta vinculado a
una aplicacion

void print_ip(char *msg, struct ip_addr *ip) {//imprime direccion
ip4
    print(msg);
    xil_printf("%d.%d.%d.%d\n\r", ip4_addr1(ip), ip4_addr2(ip),
                ip4_addr3(ip), ip4_addr4(ip));
}

void print_ip_settings(struct ip_addr *ip, struct ip_addr *mask,
struct ip_addr *gw){
    print_ip("Direc. IP:       ", ip);
    print_ip("Mascara :       ", mask);
    print_ip("Puerta de E. :       ", gw);
}

int main()
{
      struct netif *netif, server_netif; //declaracion estructuras
nuevas netif
      struct ip_addr ipaddr, netmask, gw;//declaracion direcciones
IP4

      /* arregl de caracteres MAC 6 elementos */
      unsigned char mac_ethernet_address[] = { 0x00, 0x0a, 0x35,
0x00, 0x01, 0x02 };

      netif = &server_netif;

        if (init_platform() < 0) {
            xil_printf("ERROR: Inicializacion Plataforma .\n\r");
            return -1;
        }

      /* Construyendo Dir IP, P.E Y Net Mask*/
#if 1
      IP4_ADDR(&ipaddr,  192, 168,   1, 10);
      IP4_ADDR(&netmask, 255, 255, 255,  0);
      IP4_ADDR(&gw,      192, 168,   1,  1);
#else
      IP4_ADDR(&ipaddr,  172, 16,   0, 10);
      IP4_ADDR(&netmask, 255, 255, 255,  0);
      IP4_ADDR(&gw,      172, 16,   0,  1);
#endif
```

```
        xil_printf("\n\r\n\r");
        xil_printf("-----!..Servicios para Microblaze..! ------
\n\r");
        print_ip_settings(&ipaddr, &netmask, &gw);

      lwip_init(); //inicio de la biblioteca LWIP130

      /* Asignando la direccion IP, MAC, Mascara y gateway a la
interfaz */
      if (!xemac_add(netif, &ipaddr, &netmask, &gw,
mac_ethernet_address, PLATFORM_EMAC_BASEADDR)) {
            xil_printf("Error adding N/W interface\n\r");
            return -1;
      }
      //por defecto la estructura IP antes creada
      netif_set_default(netif);

      /* Activando interrupciones */
      platform_enable_interrupts();

      /* Inicia la interfaz de red */

      netif_set_up(netif);

      /* Arranque de las aplicaciones HTTP, RXiperf*/
      start_applications();
      //imprimiendo las cabeceras que seran vistas mediante RS232.
      //BR: 9800, sin paridad, sin control de hardware
      print_headers();

      /* Bucle infinito para la recepcion y envio de paquetes */
      while (1) {
            // la interfaz de red recive los paquetes, los traslada
a el
            //stack tcp/ip LWIP, LWIP segun el tipo de paquete llama
a la funcion
            //que tomara el paquete.
            xemacif_input(netif);
            //En caso de que se necesite enviar un dato, llama a una
funcion especifica
            //para procesar el paquete.
            transfer_data();
      }

        /* en ccaso de error se limpian los buffers del sistema */
        cleanup_platform();

      return 0;
}
```

ANEXO 3

Código fuente Servidor WEB en C.

webserver.c

```
/*
 servidor web
 *
 */

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

#include "lwip/err.h"
#include "lwip/tcp.h"

#include "webserver.h"

/* variables estaticas que controlan la depuracion en este archivo
*/
static int g_webserver_debug = 0;

static unsigned http_port = 80;
static unsigned http_server_running = 0;

int transfer_web_data() {
      return 0;
}

err_t
http_sent_callback(void *arg, struct tcp_pcb *tpcb, u16_t len)
{
      int BUFSIZE = 1400, sndbuf, n;
      char buf[BUFSIZE];
      http_arg *a = (http_arg*)arg;

      if (g_webserver_debug)
            xil_printf("%d (%d): S%d..\n\r", a?a->count:0, tpcb-
>state, len);

      if (tpcb->state > ESTABLISHED) {
            if (a) {
                  pfree_arg(a);
                  a = NULL;
            }
            tcp_close(tpcb);
                return ERR_OK;
      }

      if (a->fd == -1 || a->fsize <= 0) /* no hay mas datos para
enviar */
            return ERR_OK;

      /* lee mas datos fuera del archivo y los envia */
        while (1) {
            sndbuf = tcp_sndbuf(tpcb);
            if (sndbuf < BUFSIZE)
                return ERR_OK;
```

```
            xil_printf("tratando leer %d bytes, left = %d
bytes\n\r", BUFSIZE, a->fsize);
            n = mfs_file_read(a->fd, buf, BUFSIZE);
            tcp_write(tpcb, buf, n, 1);
            a->fsize -= n;

            if (a->fsize <= 0) {
                mfs_file_close(a->fd);
                a->fd = 0;
                break;
            }
        }

      return ERR_OK;
}

err_t
http_recv_callback(void *arg, struct tcp_pcb *tpcb, struct pbuf *p,
err_t err)
{
      http_arg *a = (http_arg*)arg;

      if (g_webserver_debug)
            xil_printf("%d (%d): R%d %d..\n\r", a?a->count:0, tpcb-
>state, p->len, p->tot_len);

      /* no se leen paquetes si la conexion no esta en estado
ESTABLISHED */
      if (tpcb->state >= 5 && tpcb->state <= 8) {
            if (a) {
                  pfree_arg(a);
                  a = NULL;
            }
            tcp_close(tpcb);
            return;
      } else if (tpcb->state > 8) {
            return;
      }

      /* acknowledge que se ha leido la carga */
      tcp_recved(tpcb, p->len);

      /* se lee y se decifra la peticion
       * esta funcion se encarga de generar una requicicion y la
envia,
       * y cierra la conexion si todos los datos ha sido enviados.
Si
       * no se han enviados, establece los parametros adecuados
       * para enviar al manejador de llamadas apropiado
       */
      generate_response(tpcb, p->payload, p->len);

      /* libre para recivir un paquete */
      pbuf_free(p);

      return ERR_OK;
}

static err_t
```

```
http_accept_callback(void *arg, struct tcp_pcb *newpcb, err_t err)
{
	/* guarda el numero de conexiones */
	tcp_arg(newpcb, (void*)palloc_arg());

	tcp_recv(newpcb, http_recv_callback);
	tcp_sent(newpcb, http_sent_callback);

	return ERR_OK;
}

int
start_web_application()
{
	struct tcp_pcb *pcb;
	err_t err;

	/* inicializa la capa de sistema de archivos */
	platform_init_fs();

	/* inicializa los dispositivos */
	platform_init_gpios();

	/* crea una nueva estructura TCP PCB */
	pcb = tcp_new();
	if (!pcb) {
		xil_printf("Error creando PCB. Fuera de memoria\n\r");
		return -1;
	}

	/* se vincula al puerto 80 http */
	err = tcp_bind(pcb, IP_ADDR_ANY, http_port);
	if (err != ERR_OK) {
		xil_printf("Incapaz de vincularce al puerto 80: err =
%d\n\r", err);
		return -2;
	}

	/* no necesitamos mas argumentos para la primera llamada */
	tcp_arg(pcb, NULL);

	/* escuchando por conexiones */
	pcb = tcp_listen(pcb);
	if (!pcb) {
		xil_printf("Fuera de memoria mientras tcp_listen\n\r");
		return -3;
	}

	/* especifica la llamada para conexiones entrantes */
	tcp_accept(pcb, http_accept_callback);

	  http_server_running = 1;

	return 0;
}

void print_web_app_header()
{
    xil_printf("%20s %6d %s\n\r", "Servidor http",
```

```
                        http_port,
                        "Escriba en su explorador to
http://192.168.1.10");
}
```

webserver.h

```
/*
 Funciones prototipo y variables para el servidor HTTP
 */

#ifndef __WEBSERVER_H__
#define __WEBSERVER_H__

#include "lwip/tcp.h"

#define MAX_FILENAME 256

/* Inicializacion del sistema de ficheros */
int platform_init_fs();

/* inicializacion de la capa de dispositivos */
int http_init_devices();
//funciones prototipo
//peticion de archivo a travez de http
void extract_file_name(char *filename, char *req, int rlen, int
maxlen);
char *get_file_extension(char *buf);
//actualizar resultados de los switchs
int is_cmd_switch(char *buf);
//encendera apagar leds
int is_cmd_led(char *buf);
//si es la solicitud adc
int is_cmd_adc(char *buf);
//genera la cabecera http
int generate_response(struct tcp_pcb *pcb, char *http_req, int
http_req_len);

typedef struct {
      int count;
      int fd;
      int fsize;
} http_arg;

http_arg *palloc_arg();
void pfree_arg(http_arg *);

#endif
```

ANEXO 4

Programa principal Módulo ADC en C.

```
#include <mega32.h>
#include <stdio.h>
#include <delay.h>
#include <string.h>


#define ADC_VREF_TYPE 0x00

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
    ADMUX=adc_input;
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion
    ADCSRA|=0x40;
    // Wait for the AD conversion to complete
    while ((ADCSRA & 0x10)==0);
    ADCSRA|=0x10;
    return ADCW;
}

void main(void)
{
    unsigned char sp3_in, adc_data;

    // ADC initialization
    // ADC Clock frequency: 1000,000 kHz
    // ADC Voltage Reference: AREF pin
    ADMUX=ADC_VREF_TYPE & 0xff;
    ADCSRA=0x83;


   // USART initialization
    // Communication Parameters: 8 Data, 1 Stop, No Parity
    // USART Receiver: On
    // USART Transmitter: On
    // USART Mode: Asynchronous
    // USART Baud Rate: 9600
    UCSRA=0x00;
    UCSRB=0x18;
    UCSRC=0x86;
    UBRRH=0x00;
    UBRRL=0x33;

   while (1)
   {
    sp3_in=getchar();
    if (sp3_in=='x'){
        putchar('y');
        sp3_in=getchar();
        switch (sp3_in)
        {
        case '0':   adc_data=read_adc(0);
                printf("/%d:\n\r",adc_data);
                break;
        case '1':   adc_data=read_adc(1);
                printf("/%d:\n\r",adc_data);
                break;
        case '2':   adc_data=read_adc(2);
                printf("/%d:\n\r",adc_data);
                break;
```

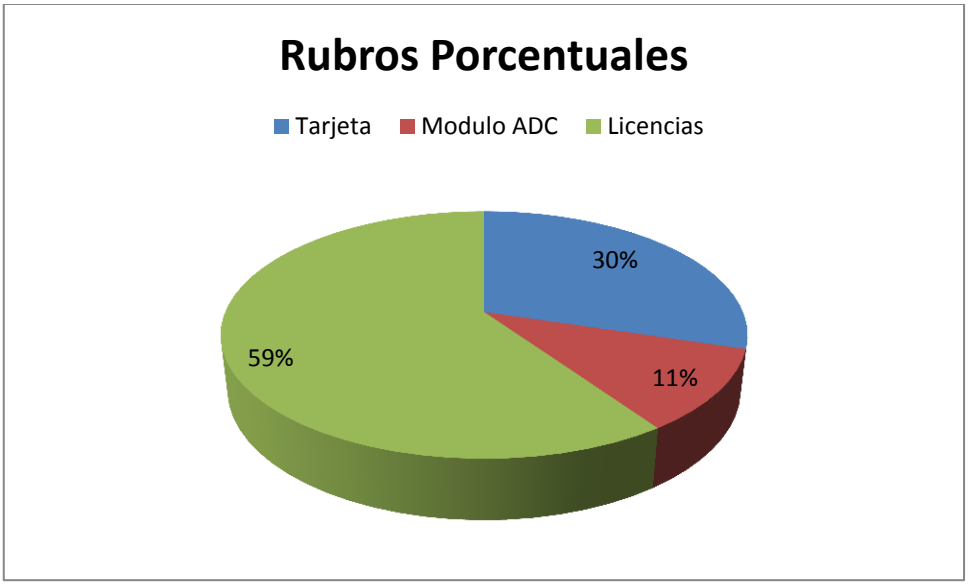
```
        case '3':   adc_data=read_adc(3);
                    printf("/%d:\n\r",adc_data);
                    break;
        case '4':   adc_data=read_adc(4);
                    printf("/%d:%d\n\r",adc_data);
                    break;
        case '5':   adc_data=read_adc(5);
                    printf("/%d:\n\r",adc_data);
                    break;
        case '6':   adc_data=read_adc(6);
                    printf("/%d:\n\r",adc_data);
                    break;
        case '7':   adc_data=read_adc(7);
                    printf("/%d:\n\r",adc_data);
                    break;


        }
    }
}

}
```

ANEXO 5

PRESUPUESTO

PRESUPUESTO GENERAL DE INVERSION PARA SWE

Código	Descripción	Unidad	Cantidad	P. Unitario	P. Total
001	Tarjeta Spartan 3E Starter Kit Digilent	U	1	124,68	124,68
002	**Modulo ADC**	U	1	44,62	44,62
002-1	Microcontrolador ATMEGA 32	U	1	8,47	8,47
002-2	Integrado MAX232	U	1	1,5	1,5
002-3	Conectores DB9-H	U	2	0,45	0,9
002-4	Conectores DB9-M	U	2	0,45	0,9
002-5	Resistores, Capacitores, leds y Switchs	U	7	0,15	1,05
002-6	Cable UTP-32	m	1	0,8	0,8
002-7	Placa Base Baquelita	cm2	50	0,02	1
002-8	Diseño y construcción	u	1	30	30
003	Licencias XILINX ISE 11.1 Educacional	U	1	250	250
				Subtotal:	419,3
				IVA	50,32
				Total:	469,62
	Son: CUATROCIENTOS SESENTA Y NUEVE CON 62/100 USD.				



ANEXO 6

Diseño e implementación de un servidor web embebido basado en una arquitectura reconfigurable con FPGAs para el control y monitoreo de periféricos.

Christian Lojan Herrera, *Profesional en formación, UTPL*, e Ing. Manuel Quiñones Cuenca, *Docente, UTPL.*

***Resumen*--El presente trabajo se centra en el diseño e implementación de un servidor web embebido (SWE) sobre una tarjeta electrónica Spartan 3E Starter Kit, destinado al control y monitoreo de periféricos mediante una interfaz gráfica basada en: HTML, CSS y JavaScript. El sistema se ha diseñado mediante el método SoC de inclusión modular de IPCORES de Xilinx, y está basado en el Firm Core Microblaze que utiliza un modelo de pila de protocolos TCP/IP de bajo peso llamado LWIP para: proveer un servicio WEB mediante HTTP y un servicio de medición de ancho de banda. Finalmente se realizan las pruebas de estabilidad, conexiones simultáneas, ancho de banda y consumo energético.**

***Palabras clave*--** SWE, HTML, IPCORES, LWIP, Microblaze, SoC.

I. INTRODUCCIÓN

Los servidores web embebidos (SWEs), son dispositivos construidos a medida para proveer un servicio de transporte de hipertexto (HTTP) permitiendo a un cliente visualizar páginas HTML mediante un navegador WEB [1].

Los SWEs son una solución muy atractiva para sistemas de comunicación Ethernet de bajos requerimientos de procesamiento. Los sistemas embebidos generan un bajo consumo energético y un bajo coste de implementación y desarrollo, por lo que, son una opción muy viable para sistemas de control y supervisión remotos.

Un servidor embebido con capacidades HTTP presenta una interfaz gráfica amigable, accesible a través de cualquier cliente HTML con capacidades de comunicación TCP/IP.

La técnica de desarrollo SoC de un sistema embebido divide al diseño en etapas mutuamente complementarias. La primera etapa consiste en: determinar los requerimos del sistema y la partición del sistema en software y hardware. Las dos etapas subsiguientes contemplan el diseño del hardware y diseño de software. Por último se realiza la integración y la verificación del sistema. Como se muestra en la Figura 1.

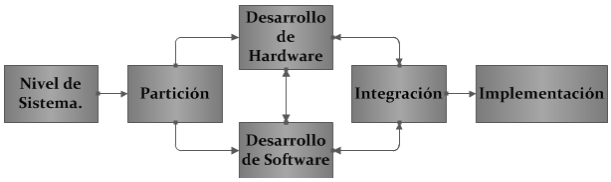


Figura 1. Codiseño de un SoC. Fuente [2].

II. DISEÑO DE HARDWARE

El SWE es capaz de: proveer una comunicación TCP/IP confiable a través de una interfaz física RJ-45, un reporte de acciones a consola a través de una interfaz DB9 sobre un protocolo RS232, una comunicación RS232 con un módulo externo ADC vía una interfaz DB9. El esquema de implementación del SWE puede apreciarse en la Figura 2.

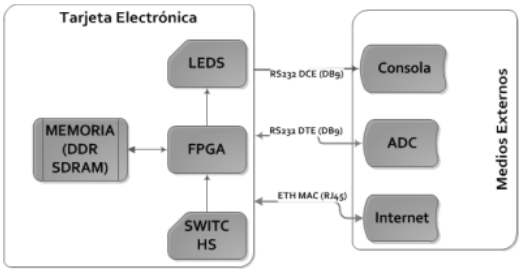


Figura 2. Esquema de requerimientos del SWE. Elaborado por el Autor.

Sistema fue construido sobre una tarjeta electrónica Spartan 3E Starter Kit de Digilent, las características de los dispositivos requeridos según la referencia [3] para la plataforma de Hardware se pueden apreciar en la Tabla 1.

Tabla 1. Requerimientos del SWE. Elaborado por el Autor.

Dispositivo	**Uso en el Sistema.**
FPGA XC3S500E-4FGG320C	Procesamiento de información, permite embeber IPCORES.
Plataforma Flash	Permite almacenar y descargar

Xilinx XCF04	aplicaciones para el SWE en conjunto con ISE 11.1.
Memoria SDRAM DDR SDRAM 32MB	Almacena sistemas de ficheros, programas para el SWE.
Fuente de Poder LT3412	Brinda la alimentación regulada de potencia para la memoria DDR-SDRAM.
Fuente de Poder TPS75003	Proporciona la alimentación regulada de potencia a la FPGA.
Ethernet PHY SMSC LAN83C185	Realiza el procesamiento Ethernet, independiente, para el servicio de la capa física con negociación CSMA/CD.

III. IPCORES Y BUSES.

La plataforma de hardware fue enteramente diseñada en EDK 11.1, que es parte del ISE 11.1 de Xilinx Inc. Los dispositivos como: memorias, buses o procesadores se encuentra integrados en la FPGA, como módulos denominados IPCORES, todos los IPCORES se encuentran vinculados a una contraparte física mediante un mapeo de puertos del FPGA.

El corazón de toda la plataforma es el Firm core RISC de 32 bits Microblaze que proporciona todas las funcionalidades de un microprocesador [5]. Microblaze interactúa con el IPCORE MPMC para hacer uso de la memoria DDR SDRAM, mediante un bus dual XCL por una interfaz PIM.

Los periféricos están representados por dos instancias de un IPCORE de propósito general de E/S denominado GPIO, para los switchs y leds, dos instancias de UART Lite para el control de la interfaz DB9 DTE y DCE para el ADC y la Consola de estado, Ethernet Lite Mac para el control de interfaz de red RJ-45.

El sistema conecta todos los componentes mediante buses de comunicación: los periféricos hacen uso del bus PLB (Peripheral Local Bus), el bus P2P para interactuar con el entorno de programación ISE. Finalmente para interconecta la cache interna y los bloques de DDRAM hace uso de buses LMB (Local Memory Bus).

IV. DISEÑO DE SOFTWARE.

El diseño de la plataforma de software se la realizó en SDK 11.1 que es parte del ISE 11.1 de Xilinx, mientras que las aplicaciones e interfaces de usuario se las realizó en WEBUILDER 2010. Se importó el diseño de Hardware desde EDK 11.1, ya que SDK 11.1 provee un ambiente más propicio para la programación en C.

A. Plataforma.

La plataforma de software es donde se asentarán las funcionalidades del sistema. La plataforma es de tipo Standalone y posee algunas de las funcionalidades del lenguaje C, todas las funcionalidades han sido reducidas para una integración completa con el procesador Microblaze.

LWIP130 es una biblioteca de funcionalidades que necesariamente deben ser incluidas a la plataforma standalone, ya que brindan la funcionalidad de una verdadera pila de protocolos TCP/IP en apenas 40 kB de código [6].

La biblioteca Xilmfs al ser incluida en la plataforma de software provee un servicio de sistema de ficheros, así se puede acceder a ficheros organizados en espacios contiguos de memoria [7].

La configuración de biblioteca LWIP130 debe hacerse tomando en cuenta que sobre el protocolo TCP se ha de transportar hipertexto (HTTP) destinado al servicio WEB. La biblioteca Xilmfs debe ser coherente con el inicio del módulo DDR SDRAM, además debe considerarse que se ha de carga una imagen pre construida tipo mfs que contiene los ficheros destinados a la visualización y operación del servicio web.

B. Programas de servicio.

El programa principal arranca todos los servicios del sistema (contadores, controladores, servicio web e servicio IPERF) y las librerías asociadas a los mismos. Además de establecer una identificación única de la tarjeta en la Red. En la Figura 3, puede apreciarse el algoritmo del programa principal.

Figura 3. Algoritmo del programa principal. Elaborado por el Autor.

El servidor WEB, está basado en la versión HTTPV1.1 y únicamente implementa los métodos de petición GET y POST. El método GET está destinado a la obtención de ficheros como: imágenes, gifs, pdf, html, txt. El método POST únicamente es para la obtención dinámica del estado de los periféricos como: ADC, leds o switchs.

El servidor cuenta con funciones específicas que permiten extraer el método contenido en la carga útil del paquete entrante y proceder según sea el requerimiento. Las funciones solo son llamadas si se ha logrado una conexión TCP exitosa.

El servidor analiza las peticiones tipo MIME y las responde en el mismo formato según se especifica en la referencia [8]. Las peticiones mediante GET están más relacionadas con el sistema de ficheros y extraen los mismos de la imagen mfs en porciones de 1400 bytes para optimizar memoria y el transporte de datos.

El método POST, se utiliza para el acceso a bajo nivel de los registros de los dispositivos. Los módulos para el procesamiento POST interactúan directamente con los dispositivos y tienen acceso a sus registros en bajo nivel para monitorear o controlar los periféricos, el método POST utiliza la notación JSON en conjunto con JavaScript (JS) para realizar peticiones mediante comandos específicos al sistema.

C. Interfaz HTML.

La interfaz de usuario fue diseñada bajo estándar HTML v4.1, se complementa con JS, CSS y Yui2, lo que le agrega a la página una mejor presentación, además de un tamaño pequeño y un tiempo de desarrollo corto.

CSS optimiza líneas de código al establecer un formato de estilo a toda la página. JavaScript brinda todas las facilidades para la programación orientada a objetos que permite manipular interactivamente los objetos de la página y a través de ellos a los periféricos. Yui2, es una biblioteca de Scripts pre construidos que además de brindar funcionalidades de conexión y apariencia optima, permite reducir drásticamente el tiempo de desarrollo. La página puede apreciarse en la Figura 4.

Figura 4. Página SWE. Elaborado por el Autor.

D. Interfaz HTML.

El servicio IPERF es un servicio suplementario destinado únicamente a pruebas de rendimiento de red del sistema, actúa como un espejo ante paquetes entrantes. Puede ser accedido con un cliente iperf. Al igual que el SWE utiliza las primitivas de conexión mediante TCP.

V. INTEGRACIÓN

La integración de hardware y software es una actividad de retroalimentación y muchas de las veces de ensayo error. La plataforma de hardware debe unirse estrechamente con la plataforma de software para garantizar un ambiente óptimo para la ejecución de los servicios ofertados por el sistema.

La página Web debe ser empaqueta en un sistema de ficheros de tipo mfs para poder ser descargada a la memoria DDR SDRAM de la tarjeta, esto se lo realiza mediante la consola xmd y el comando mfsgen.

Los programas de servicio al ser compilados generan un archivo ejecutable elf, es de vital importancia antes de su compilación realizar el Link Script y asignar las porciones destinadas a heap y stack a la memoria DDR SDRAM para evitar posibles errores.

El primer paso en la integración es la descarga de la descripción del hardware a la tarjeta (system.bit), luego se procede con la descarga de la imagen del sistema de ficheros en la memoria DDR SDRAM (*.mfs). Por último se debe descargar el ejecutable de los programas de servicio (*.elf).

VI. PRUEBAS

Se realiza las pruebas de estabilidad, ancho de banda, solicitudes múltiples, tiempos de carga, optimización de LTUS y consumo energético, según el esquema que se detalla en la Figura 5.

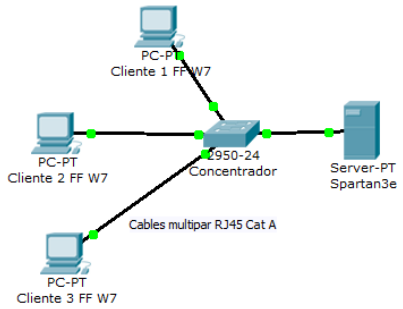


Figura 5. Esquema de pruebas. Elaborado por el Autor.

A. Múltiples solicitudes.

Ante múltiples solicitudes el servidor ha sido capaz de atender a 25 solicitudes simultáneas, degradando sus tiempos respuesta a medida que las solicitudes se van incrementando una media de la degradación de los tiempos de respuesta es 400 ms. También se ha enviado 30 solicitudes de ping simultáneas obteniendo el 80% de los paquetes confirmados.

B. Ancho de banda IPERF.

El servidor de rendimiento se lo ha ejecutado desde el cliente 1 mediante IPERF, en 10 pruebas se ha obteniendo un ancho de banda promedio de 500 kbps. En la Figura 6 se muestra una captura de una de las pruebas.

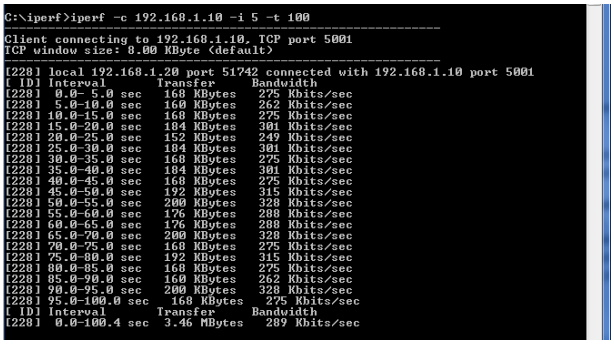


Figura 6. Prueba de rendimiento Iperf. Elaborado por el Autor.

C. Tiempo de carga.

Mediante el uso del complemento FireBug 1.9.0 para Mozilla Firefox 6.0, se ha logrado comprobar que el tiempo de la carga de la página en condiciones normales de operación es de 7.46 s. con una carga total de 113.2 kB de 23 solicitudes GET y POST.

D. Uso FPGA de plataforma de hardware.

En la Tabla 2, se detalla el uso de los recursos del sistema basado en FPGA con arquitectura reconfigurable.

Tabla 2. Uso FPGA. Fuente EDK 11.1.

Utilización Lógica	Usada	Disponible	Utilización (%)
Número de slice flip flops	3395	9312	36%
Número de LUT'S de 4 entradas	4255	9312	45%
Número de Slices ocupados	3206	4656	68%

E. Consumo energético.

Mediante la utilización XPE (Xpower Estimator) 11.1 [9] para la tarjeta Spartan 3E Starter Kit, se pudo estimar el consumo de potencia de la tarjeta en 0.384 W, en la Figura 7, se puede visualizar en más detalle los resultados.

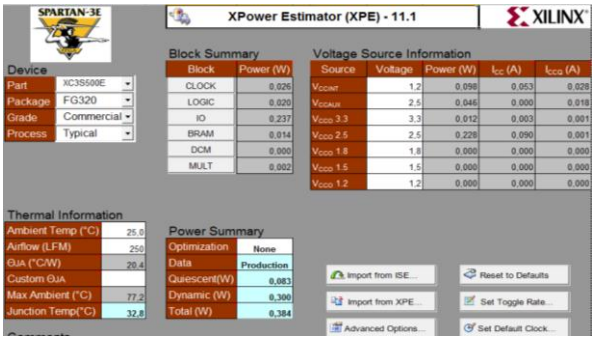


Figura 7. Estimación de consumo de potencia. Referencia [9].

VII. CONCLUSIONES

• Se ha desarrollado un sistema embebido sobre la tarjeta Spartan 3E Starter Kit, accesible vía web con capacidades HTTP sobre TCP/IP capaz de monitorear y controlar variables analógicas o digitales sobre un FPGA. El SWE puede adaptarse a cualquier sistema de control o monitoreo con una estabilidad operacional aceptable

• El prototipo posee capacidades HTTP para el transporte de datos además de una interfaz gráfica HTML que facilita la operación del mismo. El usuario puede acceder al sistema remotamente mediante cualquier intérprete HTML con capacidades JavaScript y CSS, y está en la capacidad de modificar y monitorear parámetros de bajo nivel de los periféricos del sistema mediante el uso de objetos DOM HTML como: botones, checkboxes, etc.

• Uno de los aportes de este proyecto está en la experiencia adquirida en el uso de FPGAs de Xilinx, la cual deberá ser considerada para realizar desarrollo de sistemas basados en comunicaciones Ethernet, ya que se comprendió su tendencia aplicativa en campos emergentes como: control y supervisión de procesos productivos, domótica, transporte, tele-medicina, posicionamiento global, servicios bancarios, información turística, entre otros.

• El Firmware desarrollado programado sobre el FPGA XC3S500E, ocupa el 47% de los recursos disponibles, por lo cual posee el 53% de recursos para almacenar programación, se puede pensar en una futura incorporación de multiprocesamiento mediante la inclusión de otra instancia del Firm core Microblaze.

• Al comprobar la alta capacidad de tecnologías embebidas basadas en FPGAs, se concluye que se puede implementar sistemas electrónicos complejos, al quedar demostrado cómo es factible implementar una arquitectura de un microcomputador mediante la incorporación de: un procesador Microblaze, memorias, contadores, protocolos de comunicación, etc.

• Mediante las pruebas realizadas al servidor WEB, se pudo comprobar la estabilidad de conexión ante 25 conexiones entrantes, una de carga máxima de 80 kbps ante solicitudes simultaneas al servicio. El servidor se comporta estable y presenta un buen funcionamiento cuando se generan peticiones de datos HTML con una cantidad igual o menor a ocho clientes y un factor de degradación promedio de 400 ms a medida que se incrementan los clientes.

• El cliente IPERF ha demostrado que el tráfico de datos en el servidor puede ser menor o igual a 500 kbps y que la cantidad de tráfico se ve degradada a medida que se conecten más clientes al sistema.

VIII. REFERENCIAS

[1] W. Nicholas. Designing an Embedded Web Server (2000). U.S.A: Applied Computing Technologies. Recuperado el 2011 de http://pdf.cloud.opensystemsmedia.com/embedded-computing.com/USSoftware.Win00.pdf

[2] A, M., & J, A. (2010). *"Servidor Web Embebido en una FPGA con Codiseño como Metodología de Diseño"*. Innovation and Development for the Americas. Arequipa: LACCEI.

[3] Digilent. (2006)."*Spartan-3E Starter Kit Board User Guide*". Recuperado el 2011, de http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf

[4] Atmel Corporation. (2011). *"Datasheet Atmega32"*. Recuperado el 2011, de www.atmel.com

[5] Xilinx Inc. (2008). MicroBlaze Processor Reference Guide for EDK 10.1. Recuperado el 2011, de http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[6] Dunkels, A. (2001). *Design and Implementation of the lwIP TCP/IP Stack*. Suecia: Swedish Institute of Computer Science. Recuperado el 2011 de Recuperado el 2011 de http://www.sics.se/~adam/lwip/doc/lwip.pdf

[7] Xilinx Inc. (2009). *OS and Libraries Document Collection*. Recuperado el 2011, de http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf

[8] Network Working Group. (2009). Hypertext Transfer Protocol -- HTTP/1.1. Recuperado el 2011, de http://www.w3.org/Protocols/rfc2616/rfc2616.html

[9] Xilinx Inc. (2011). *XPower Estimator User Guide*. Recuperado el 2011 de http://www.xilinx.com/support/documentation/user_guides/ug440.pdf

ANEXO 7

Utilización del FPGA detallada por IPCORES.

Instancia IPCORE	Parametros	Usado	Disponible	% Utilización
Microblaze	Slices	1062	4656	22
	Slices Flip Flops	1192	9312	12
	4 inputs LTUs	2009	9312	21
	BRAMs	6	20	30
	MULTI18X18SIOs	3	20	15
Microblaze Debug Module	Slices	88	4656	1
	Slices Flip Flops	119	9312	1
	4 inputs LTUs	142	9312	1
	GCLKs	2	24	8
XPS Interrupt Controller	Slices	90	4656	1
	Slices Flip Flops	128	9312	1
	4 inputs LTUs	93	9312	1
LMB DLMB	Slices	1	4656	0
	Slices Flip Flops	1	9312	0
	4 inputs LTUs	1	9312	0
LMB ILMB	Slices	1	4656	0
	Slices Flip Flops	1	9312	0
	4 inputs LTUs	1	9312	0
PLB (mb_plb)	Slices	287	4656	6
	Slices Flip Flops	158	9312	1
	4 inputs LTUs	492	9312	5
BRAM (lmb_bram)	Slices	0	4656	0
	IOS	206	NA	NA
	BRAMs	4	20	20
DDR SDRAM	Slices	768	4656	16
	Slices Flip Flops	1182	9312	12
	4 inputs LTUs	925	9312	9
	BRAMs	3	20	15
LMB BRAM (dlmb_cntrl)	Slices	3	4656	0
	Slices Flip Flops	2	9312	0
	4 inputs LTUs	6	9312	0
LMB RAM (ilmb_cntrl)	Slices	3	4656	0
	Slices Flip Flops	2	9312	0
	4 inputs LTUs	6	9312	0
GPIO (Dips_4bit)	Slices	63	4656	1
	Slices Flip Flops	97	9312	1
	4 inputs LTUs	58	9312	0
ETHERNET MAC LITE	Slices	435	4656	9

	Slices Flip Flops	493	9312	5
	4 inputs LTUs	686	9312	7
	BRAMs	4	20	20
GPIO (LEDS_8bits)	Slices	79	4656	1
	Slices Flip Flops	125	9312	1
	4 inputs LTUs	70	9312	0
UART Lite (RS232 DCE)	Slices	110	4656	2
	Slices Flip Flops	146	9312	1
	4 inputs LTUs	134	9312	1
UART Lite (RS232 DTE)	Slices	110	4656	2
	Slices Flip Flops	146	9312	1
	4 inputs LTUs	134	9312	1
Processor System Reset	Slices	41	4656	0
	Slices Flip Flops	67	9312	0
	4 inputs LTUs	51	9312	0
xps_timer	Slices	326	4656	7
	Slices Flip Flops	359	9312	3
	4 inputs LTUs	377	9312	4
Clock generator	Slices	6	4656	0
	Slices Flip Flops	10	9312	0
	4 inputs LTUs	6	9312	0
	GCLKs	4	24	16
	DCMs	2	4	50