Sistema de monitoreo de una red de comunicación basado en un FPGA y Linux

Alberto Rodríguez Sánchez, Oscar Alvarado-Nava y Francisco Javier Zaragoza Martínez*
Departamento de Electrónica,*Departamento de Sistemas
Universidad Autónoma Metropolitana, Unidad Azcapotzalco, México D.F., México
al204303680@alumnos.azc.uam.mx, {oan,franz}@correo.azc.uam.mx

***Abstract*—Los paradigmas de computación distribuida, cómputo móvil y los actuales sistemas colaborativos, han generado la creciente interconexión de sistemas de cómputo en redes de comunicación. El permitir el acceso a múltiples dispositivos móviles a una red de comunicación, ha creado la necesidad de diseñar aplicaciones autónomas y de bajo consumo de energía capaces de monitoriar y generar estadísticas sobre el uso del ancho de banda y la detección de actividades sospechosas, con el fin de mantener una comunicación confiable entre dispositivos. El presente trabajo describe creación de un sistema de monitoreo de red en un FPGA ejecutando linux, el cual utiliza el protocolo SNMPv2 para gestionar los dispositivos que están conectados a la red. Los agentes SNMP son ejecutados como aplicaciones de linux y almacenan la información en una base de datos relacional utilizando el manejador SQLite. El sistema muestra que es posible crear una aplicación embebida con soporte de bases de datos relacionales, para almacenar la información del uso de la red de comunicación y así poder realizar distintos análisis de tráfico y seguridad.**

***Index Terms*—FPGA, Linux, Network, SNMP, SQLite, Security.**

I. Introducción

El desarrollo de aplicaciones de cómputo sobre dispositivos embebidos tiene gran interés para la industria de dispositivos de bajo consumo de energía, ya que permiten crear sistemas de cómputo orientados a una tarea específica sin la necesidad de tener un sistema de cómputo completo.

Un disposivo autónomo capaz de manipular y realizar estadísticas del tráfico de red, puede ser de gran utilidad para la administración y seguridad en redes de computadoras, ya que permitiría detectar actividades mal intensionadas en el uso del ancho de banda al mantener un histórico y así modelar el comportamiento del uso de la red con respecto al tiempo.

El presente trabajo se centra en la descripción del desarrollo de un sistema embebido basado en un FPGA, en el cual se ejecutan aplicaciones y servicios a través del sistema operativo Linux. El sistema es capaz de monitoriar las estaciones conectadas en una red almancenado información, estadísticas y gráficas del uso del ancho de banda de la red. Los resultados se pueden consultar al ingresar al sistema embebido para gestionar directamente la base de datos o puden ser mostrados a través de imágenes de gráficas insertadas en una página WEB generada de manera automática. En la Figura 1 se muestra un esquema en capas del sistema embebido a desarrollado.

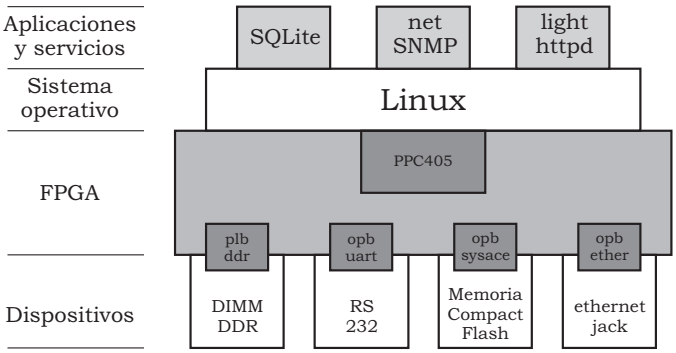

Fig. 1. Sistema embebido en capas.

En la actualidad el desarrollo de este tipo de sistemas cómputo simplificado, permiten explicar de manera práctica y a la vez concisa, los distintos niveles de abstracción para el diseño y desarrollo de sistemas embebidos, considerando los temas de diseño digital, arquitectura de computadoras, sistemas operativos y redes de computadoras en favor de una formación integral de un ingeniero en electrónica o ingeniero en computación.

II. Plataforma de desarrollo

En el diseño e implementación de aplicaciones embebidas, es de gran importancia conocer tanto el proceso como las herramientas de desarrollo. El proceso de diseño considera el desarrollo de software y el desarrollo de hardware, los cuales pueden darse de manera simultanea o de manenera independiente, integrandose finalmente en el proceso de implementación o programación del dispositivo. Cabe destacar que una caraterística de las herramientas de desarrollo actuales, es desarrollar una aplicación completa presindiendo del dispositivo objetivo, incluso en muchos casos, sólo será necesario cambiar algunos parámetros para la implementación del proyecto en un dispositivo con una tecnología diferente. Lo anterior es debido a que actualmente en las herramientas de desarrollo se cuenta con varios tipos de simuladores y aplicaciones para la depuración de errores, tanto para software como para hardware. En general, las compañias fabricantes de dispositivos, ofrecen IDE's (*Integrated Development Environment*) de desarrollo que incluyen el software necesario para la planificación, edición, compilación-síntesis e implementación de aplicaciones, creando así toda una plataforma de desarrollo

En la siguientes secciones describiremos la plataforma de desarrollo Linux-ISE-EDK y la tarjeta de desarrollo de proyectos universitario XUPV2P, ambos de la compañía Xilinx [1].

A. Linux-ISE-EDK

La plataforma de desarrollo conformada por los entornos de desarrollo Xilinx ISE 10.1 y Xilinx EDK 10.1, se instaló sobre una computadora de escritorio HP ProLiant DL365G5; la cual cuenta con dos procesadores AMD Opteron cada uno con dos núcleos, 4GBytes de SRAM y el sistema operativo Debian GNU/Linux 5.0, kernel 2.6.26-1-686. El Xilinx ISE (*Integrated Software Environment*) es utilizado para el análisis y síntesis de proyectos desarrollados con lenguajes de descripción de hardware (HDL), el cual incluye simuladores funcionales y analizadores de desempeño en el tiempo. A través del Xilinx EDK (*Embedded Development Kit*) se desarrolla tanto el software del proyecto como la configuración del sistema cómputo embebido, seleccionado por ejemplo ya sea un *hard processor core* y/o un *soft processor core*; la cantidad de bloques de SRAM y los IP (*Intellectual Property*) *cores* que funcionarán como periféricos del sistema cómputo embebido.

Instalar la plataforma de desarrollo sobre un sistema estable, libre y fácilmente replicable como Debian GNU/Linux, es de gran ayuda para obtener las herramientas de software básicas y su documentación sin que esto represente un gasto económico extra. Dichas herramientas de software van desde un manejador de versiones distribuido, hasta el uso aplicaciones y herramientas más complejas como un compilador cruzado, depuradores, bases de datos relacionales, etc.

III. Hardware del sistema

A. Tarjeta XUPV2P

La tarjeta desarrollo XUPV2P (*Xilinx University Program, Virtex II Pro*) [2], es parte del proyecto de colaboración entre la compañía Xilinx y las universidades, el cual tiene como propósito acercar a los estudiantes y académicos a la tecnología de los circuitos programables, como los FPGAs y sus herramientas de desarrollo. La tarjeta cuenta con un FPGA Virtex-II Pro (XC2VP30) y dispositivos para el desarrollo de aplicaciones, destacando 10/100Mbps Ethernet PHY, una ranura para una tarjeta Compact Flash, dos ranuras para DIMMs de memoria DDR-SRAM, y un puerto RS-232, entre muchos otros. El FPGA XC2VP30 además de contar con una gran densidad de CLBs (*Configurable Logic Block*) para la implementación de circuitos digitales, tiene incrustados dos *hard processors core* PowerPC-405 y varios bloques de SRAM que en conjunto suman 2 MBytes.

Mediante los CPUs y los bloques de SRAM, es posible cargar y ejecutar programas desarrollados en un lenguaje de alto nivel, como C. Dichos programas deben ser compilados, ensamblados y enlazados previamente por las herramientas de desarrollo, para después ser cargados en los bloques SRAM en el momento de integrar el hardware y software. El programa será ejecutado inmediatamente después de la programación del FPGA.

Si se agregan módulos DDR-SRAM en las ranuras de la tarjeta XUP, el sistema será capaz de ejecutar programas más

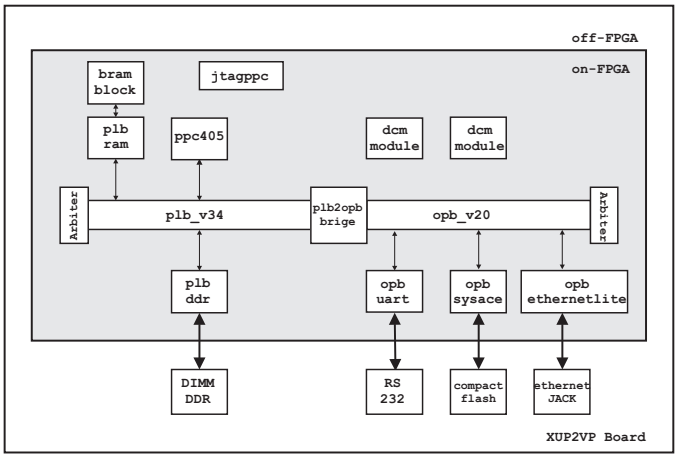

Fig. 2. Hardware del sistema embebdido basado en un FPGA.

grandes y complejos, como un sistema operativo moderno. El sistema operativo deberá ser compilado previamente para la arquitectura objetivo, ya sea para el PowerPC o para el *soft processor core* MicroBlaze [3], y así mismo deberá contener los *drivers* para el árbol de periféricos seleccionado.

La elección del tipo procesador (*hard core* o *soft core*) es relevante tanto para el desempeño y como para la utilización de recursos del FPGA. Por ejemplo, al utilizar un *hard core* permite la inclusión de más componente de hardware o periféricos, logrando una mayor funcionalidad al proyecto. Además de la elección del procesador, es necesario generar el árbol de dispositivos dentro del proyecto de hardware, para esto se puede utilizar las herramientas disponibles en los repositorios de Xilinx.

B. Sistema en un Chip

En el FPGA XC2VP30 se creó el sistema de cómputo embebido mostrado en la Figura 2, el cual está compuesto del procesador incrustrado (*hard core*) PowerPC 405 a 400Mhz (`ppc405`), un bloque de memoria SRAM *On-Chip* de 128KBytes (`bram_block`) con su respectivo módulo de acceso al bus (`plb_ram`), un sistema de buses jerárquico compuesto del bus local del procesador (`plb`), del bus de perifricos en chip (`opb`) junto con sus respectivos árbitros y un puente entre buses (`plb2opb`). Para el almacenamiento de la información fue necesario agregar el mdulo (`opb_sysace`) para el acceso a un sistema de archivos ext2 sobre una *Compact Flash Memory* de 1GBytes. Así mismo, para poder ejecutar el sistema operativo Linux se agregró al espacio de direcciones del procesador un módulo de memoria externo DIMM DDR SDRAM de 256 MBytes (`plb_ddr`). Se incluyó un transmisor-receptor serial para la entrada y salida estándar del sistema (`opb_uart`).

IV. Software del sistema

A. Cross toolchain

Una toolchain es un conjunto herramientas de software para crear un producto, el producto puede ser desde un programa ejecutable hasta una aplicación completa. Se le llama toolchain

por que las herramientas que la conforman son utilizadas una detrás de otra hasta lograr el producto final. Así, se puede configurar una toolchain para que contenga tanto los elementos necesarios como la secuencia de ejecución de los mismos para crear un sistema operativo, su *bootloader* y las aplicaciones de usuario final que podrán ser utilizadas. Una toolchain típica basada en GNU incluye las herramientas mostradas en la Tabla I. En la Tabla II se muestran las variables de ambiente necesarias por la toolchain asi como los binarios generados.

TABLA I
TOOLCHAIN BASADA EN GNU.

Herramienta	Descripción
binutils	Binarios básicos:ensamblador y ligador
gcc	GNU Compiler Collection: C/C++
glibc,uclibc,eglibc	Bibliotecas de C del sistema
linux-headers	Encabezados del kernel para bibliotecas de C
gbd	GNU debugger

TABLA II
VARIABLES DE AMBIENTE POR ÁMBITO DE CONSTRUCCIÓN.

Variable de ambiente	Toolchain	Binario
`CBUILD`	x686-pc-linux-gnu	x686-pc-linux-gnu
`CHOST`	x686-pc-linux-gnu	powerpc-405-linux-gnu
`CTARGET`	powerpc-405-linux-gnu	-
`ROOT`	Default	Custum

Al proceso de compilar un programa para una plataforma distinta a la que se lleva a cabo la compilacón, se le denomina compilación cruzada. Debido a que tradicionalmente los sistemas embebidos cuentan con recursos limitados de cómputo, la compilación cruzada es una práctica común en el desarrollo de sus aplicaciones y se lleva a cabo en sistemas de cómputo con mayores recursos de cómputo beneficiando el tiempo de desarrollo.

Para el proyecto que se describe, se creó una toolchain basada en la crosstool-ng [4], la cual además de ser constantemente actualizada es escalable, permitiendo configurar características avanzadas como la inclusión de locales (idiomas e internacionalización) y funciones de red como IPv6. El resultado de generación de una toolchain debe ser evaluado usando herramientas que permitan hacer *test regretion*, para asegurar su correcto funcionamiento.

B. Linux kernel

Una característica importante de los sistemas operativos es que permiten a los programadores aumentar el nivel abstracción para la interacción con el hardware, generando un ambiente más sencillo en el desarrollo de aplicaciones. Además de lo anterior, el kernel linux proporciona una interfaz estándar creando la portabilidad de aplicaciones entre diferentes plataformas de hardware. Para el presente proyecto, se compiló la versión 3.0_Xilinx [5] del kernel de Linux para dar soporte al hardware mostrado en la Figura 2.

El resultado de la compilación de las fuentes del kernel de Linux, es una imagen almacenada en el archivo **vmlinux**. Dicha imagen puede ser cargada directamente en la memoria, sin embargo, es preferible crear una imagen comprimida y autoextraibles (**zImage**) o en un formato especial para algún *bootloader*, como por ejemplo U-boot (**uImage**), con el fin de maximizar el espacio de almacenamiento. La imagen está compuesta de los subsistemas principales, como el *Scheduler*, el *Virtual File System*, el *Memory Manager* y el *Inter Process Comunication*, así como los *drivers* necesarios para el acceso a los dispositivos. Ya que los *drivers* pueden ocupar una gran parte de la imagen, es importante configurar adecuadamente los parámestros de compilación para soportar solo hardware seleccionado y las funcionalidades del kernel necesarias en el proyecto.

C. Root file system

Un sistema de archivos jerárquico o *Root File System* (RFS), es la clasficación de los archivos de las aplicaciones por su tipo, función y acceso de acuerdo a un estándar. El RFS es una parte muy importante en el desarrollo de un sistema embebido basado en el kernel de Linux, ya que a diferencia de otros sistemas operativos embebidos, Linux requiere de un sistema de archivos para funcionar.

Ya que el espacio de almacenamiento también es limitado en un sistema embebido, el crear un sistema de archivos pequeño pero suficiente para almacenar los archivos necesarios ha sido una gran preocupación para los desarrolladores. El *Filesystem Hierarchy Standard* (FHS) [6] establece el sistema de archivos mínimo con el que debe contar un sistema Linux para poder funcionar, el cual es mostrado en la Tabla III.

TABLA III
SISTEMA DE ARCHIVOS MÍNIMO.

Directorio	Descripción
`/bin`	Comandos comunes, compartidos por el administrador y los usuarios del sistema
`/dev`	Referencias a todos los periféricos de hardware que son representados como archivos especiales
`/etc`	Principales archivos de configuración tanto para aplicaciones del sistema como de servicios
`/lib`	Archivos bibliotecas de aplicaciones de sistema
`/sbin`	Aplicaciones para el sistema y el administrador
`/tmp`	Espacio para archivos temporales
`/var`	Archivos de sistema y de usuarios de tamaño variable como bitácoras, spoolers, etc.

El kernel podría ser compilado para contener una versión comprimida del RFS y de esta forma no necesitar de un dispositivo externo para almacenar los archivos, de manera similar a como funcionan los sistemas operativos *live*. A pesar de que con un RFS cargado en memoria se podría obtener una mejora en el rendimiento y en el almacenamiento, no siempre es recomendable ya que si no se hace un análisis minucioso del archivos requeridos, la imagen podría alcanzar un tamaño considerable y consecuentemente no podría cargarse en la limitada RAM del sistema embebido.

La solución comunmente utilizada para poder contar con un sistema de archivos completo, es crear inicialmente un sistema de archivos pequeño en un dispositivo de arranque

(como una memoria flash) y posteriormente montar un sistema de archivo más grande almacenado en otro dispositivo de mayor capacidad, o incluso montarlo desde un servidor NFS a través de una red de datos. El RFS se puede montar desde el proceso de arranque del kernel al indicarle por medio del Kernel Command Line (KCL) el dispositivo y en el cual está el RFS.

Para crear el sistema de archivos por medio de una toolchain externa, se utilizó buildroot [7] la cual es una herramienta para generar sistemas de archivos, kernels e incluso *bootloaders* con una configuración relativamente sencilla.Así mismo, buildroot hace uso de busybox [8] que es un conjunto de herramientas populares y acertadamente creadas para el desarrollo de sistemas embebidos. Busybox es un binario independiente que proporciona soporte para muchas utilidades comunes de línea de comandos de Linux.

En el presente proyecto se creó un sistema de archivos mínimo almacenado en una partición ext2 de 750MB sobre una Compact Flash de 1 GByte. Las principales aplicaciones utilizadas en el proyecto se listan en la Tabla IV.

TABLA IV
APLICACIONES UTILIZADA EN EL PROYECTO.

Aplicación	Descripción
`tcpdump`	Muestra una descripción del contenido de un paquete almacenado en una interfaz de red
`net-snmp`	Conjunto de herramientas que facilitan el el uso del protocolo SNMP
`iptables`	Programa de usuario y módulo del kernel para la administración de firewalls
`OpenSSH`	Conjunto de herramientas de para la comunicacin segura en redes
`lighttpd`	Servidor web ligero
`tcl`	Lenguaje de script de sintaxis sencilla
`sqlite3`	Sistema ligero de gestin de para bases de datos relacionales
`dhclient`	Cliente DHCP

V. CONFIGURACIÓN

Una vez programado el FPGA y lanzado el sistema operativo, se realizan configuraciones propias del sistema, las cuales van desde la seguridad del mismo hasta el inicio ordenado de los servicios. Las configuraciones se realizan por medio de scripts de *shell* y se encargan de probar e iniciar dispositivos, acceder a archivos de configuración, lanzar las aplicaciones e iniciar bitácoras.

A. Fortificación

Las primeras configuraciones consisten en la fortificación o disminución de la vulnerabilidad del propio sistema (*Hardening*), el cual consiste por ejemplo, en deshabilitar o remover servicios inecesarios e iniciar en una secuencia adecuada los servicios que se requieren para que el sistema realice su función. Una vez inciado el servcio de red, se activa un *firewall* a través iptables. Las reglas en iptables tienen la siguiente finalidad: filtrar o desechar todo el tráfico que no este asociado a algún servicio activo en el sistema, el reenvió (*forwarding*) de paquetes queda deshabilitado y todo el tráfico saliente del sistema es permitido. Adicionalmente se activa en el kernel del sistema el *Reverse Path Filtering* (rp_filter) para evitar ataques de suplantación (*spoofing*) dentro de la red a monitorear. Así, solo se permite el tráfico de *loopback* y de los servicios SSH, SNMP, DHCP, ICMP y HTTP. El servicio SSH permite tanto la administración del sistema de manera remota como el envío y recepción segura de archivos. El monitoreo del estado de distintos dispositivos en la red se lleva a cabo por medio de SNMP. El protocolo DHCP es utilizado para configurar de manera dinámica la interfaz de red del sistema. El servicio HTTP se utilizará para mostrar resultados del análisis a través de la generación de páginas WEB.

B. SNMP

El servicio snmpd se configura con opciones de lectura pública dentro de la red local y escritura protegida localmente, permitiendo así que agentes externos a la red puedan solicitar el estado del dispositivo con el propósito de verificar su funcionalidad. Se decidió utilizar la versión SNMPv2 ya que la versión SNMPv3 cifra los mensajes entre agentes SNMP con el algoritmo AES y verifica la integridad de los mismos con el algoritmo SHA1, generando una carga de procesamiento innecesaria para el proyecto.

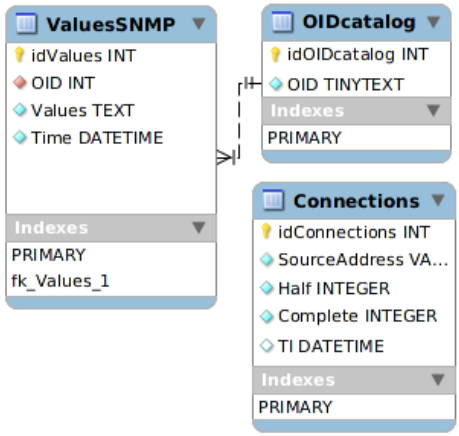

Fig. 3. Base de datos manejada por SQLite.

C. SQLite

Por medio de scripts del lenguaje interpretado TCL [9], se invoca a un sistema manejador de bases de datos relacionales (RDBMS), el cual almacena la información en una base de datos. La selección de un RDBMS para un sistema embebido tienen que ver tanto con la simplicidad y como el tamaño del mismo. SQLite [10] es un RDBMS libre y compatible con SQL que al ser desarrollado totalmente en lenguaje C, cumple con los requerimientos de tamaño y desempeño de un sistema embebido. La integración entre SQLite y TCL es transparente y solo se creará un archivo que contendrá toda la base de datos. Facilitando su recuperación y posterior análisis.

La base de datos creada se muestra en la Figura 3 y consiste en tres tablas. La tabla **Connections** observa las

peticiones completas e incompletas y las asocia a una marca de tiempo, básicamente cada 5 minutos, esto cuenta el total de conexiones en el intervalo. La tabla **ValuesSNMP** almacena una instantánea de las variables disponibles por snmp. El OID (*Object Identifiers*) de cada variable se guarda en un catalogo solo referenciado su índice con la tabla **OIDcatalog**. El Valor del OID consiste en una cadena de números separados por puntos, por ejemplo 1.3.6.1.4.1.2682.1. Dada que su longitud varia y es dependiente del MIB (*Management Information Base*), se usa texto pequeo para guardarlo, de igual forma, el campo *Values* de la tabla **ValuesSNMP** puede guardar campos que sean descripciones o notas y tienen una longitud variable, por tanto en campo será de texto. Todos estos tipos de datos están soportados en SQLite y se ha probado su afinidad.

D. Solicitud de información y almacenamiento

Por medio de scripts de TCL se invocan a las funciones snmpwalk, snmpget y tcpdump para obtener y dar un formato estándar a la información obtenida de los agentes SNMP residentes en un dispositivo administrado. La ejecución del script se puede llevar de manera periódica calendarizado por el demonio crond.

El comando tcpdump se ejecuta todo el tiempo pero solo registra las conexiones completas (SYN/ACK) y los intentos de conexión (SYN). Observar el comportamiento de solicitudes y estableciemientos de conexiones, pude ayudarnos a descubir el mal funcionamieto de un aplicación o un dispositivo e incluso operaciones mal intensionadas. Por ejemplo, las conexiones incompletas pueden dar un indicio de un posible inundamiento SYN generando un ataque DOS (*Denial-Of-Service*). Los datos sobre la conexiones se almacenan en la tabla **Connections** y su análisis se podría llevar por medio de un script TCL.

VI. Resultados

El sistema de monitoreo embebido se integró en una LAN clase C con otras cuatro estaciones. Dos estaciones generan peticiones HTTP cada tres minutos, una tercera estación funciona como servidor DHCP y la cuarta envía peticiones de conexión con la herramienta hping. Las solicitudes de conexión se intentan cada 4 minutos con 100 paquetes.

Se realizó el experimento durante 48 minutos. La base de datos adquirió un total de 239 registros nuevos adicionales al los 22 originales del catalogo y su tamaño aumento a 14.3KB. En la Figura 4 se muestra una gráfica generada con MRTG [11] la cual muestra el uso del ancho de banda por una de las estaciones monitoreadas. La imagen puede ser incrustada en una página WEB y accedida desde un navegador desde una estación.

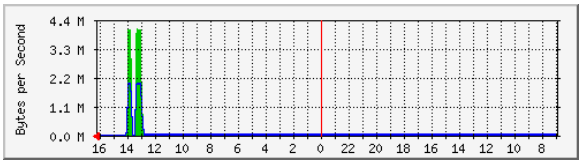

Fig. 4. Gráfica de uso de ancho de banda.

VII. Conclusiones

El presente proyecto demuestra que es posible crear todo un sistema de cómputo de propósito general o de propósito específico en un solo chip. Al estar embebido en un chip el sistema cómputo adquiere dos grandes ventajas: un bajo consumo de energía y un nivel de vulnerabilidad mínimo. Además, al contar con un sistema operativo el sistema embebido logra una gran flexibilidad en su configuración y modificación, permitiendo mejorar su funcionamiento en un tiempo de desarrollo corto.

El sistema descrito fue desarrollado en un FPGA ejecutando Linux, sin embargo con los IDEs de desarrollo actuales es posible migrar facilmente el proyecto a otras arquitecturas o plataformas más poderosas y populares como MIPS o ARM y ganar con ello valor agregado en la industria.

Es importante señalar que generalmente el precio a pagar por una gran flexibilidad, es un impacto negativo sobre el rendimiento del sistema. Por ejemplo, el uso de un lenguaje de programación interpretado ayuda en la integración de aplicaciones mejorando el tiempo de desarrollo, pero debido a las numerosas capas de sofware necesarias, aumenta el tiempo de ejecución de la aplicación.

Debido al amplio rango de conocimientos requerido, el proyecto descrito tiene el suficiente nivel de dificultad para que un estudiante de Ingeniería Electrónica o Ingeniería en Computación pueda aplicar los conocimientos adquiridos durante su formación. En este tipos proyectos es recomendable desarrollarlo en fases dentro de un taller complementario a un curso avanzado de sistemas embebidos. El beneficio para el alumno tiene que ver con elevar su percepción de la complejidad de los sistemas de computo actuales y motivarlo a realizar mejoras graduales al sistemas.

VIII. Trabajo futuro

Para que este proyecto pueda funcionar en un medio no controlado, será necesario llevar el almacenamiento a un medio masivo, por ejemplo, un disco duro SATA o un incluso a un NAS (*Network-attached storage*) tomando en cuenta que este generará tráfico que deberá ser obviado. Aunado al almacenamiento, se deberá mejorar el diseño de la base de datos para guardar valores que cambian poco o no cambian en una tabla aparte y solo actualizar cuando exista algún cambio.

El uso de la tarjeta de desarrollo XUPV2P nos confina a una sola interfaz de red, trabajos futuros incluyen llevar este proyecto a las tarjetas de desarrollo más modernas como la XUPV5 y NET-FPGA. Fuera del mundo de los FPGAs, es viable llevar el proyecto a las arquitecturas ARM y MIPS. Estas últimas son populares en dispositivos embebidos y ampliamente usadas en el mundo de las redes.

Variantes de esta proyecto ofrecen posibilidades de explotar temas específicos de temas relacionados con arquitectura de computadoras, sistemas operativos, sistemas distribuidos y redes de computadoras.

Referencias

[1] Xilinx. (2012, Jan.) Fpga, cpld, and epp solutions from xilinx, inc. [Online]. Available: http://www.xilinx.com/

[2] (2012, Jan.) Xupv2p documentation. [Online]. Available: http://www.xilinx.com/univ/xupv2p.html
[3] Xilinx. (2012, Jan.) Microblaze soft processor. [Online]. Available: http://www.xilinx.com/tools/microblaze.htm
[4] (2012, Jan.) Crosstool ng. [Online]. Available: http://www.crosstool-ng.org/
[5] K. Sievers. (2012, Jan.) git.xilinx.com git. [Online]. Available: http://git.xilinx.com
[6] D. Quinlan. (2004, Jan.) Filesystem hierarchy standard. [Online]. Available: http://www.pathname.com/fhs/
[7] E. Andersen. (2005, Jan.) Buildroot. [Online]. Available: http://buildroot.uclibc.org/
[8] ——. (2008, Jan.) Busybox. [Online]. Available: http://busybox.net/
[9] (2010, Jan.) Tcl developer site. [Online]. Available: http://www.tcl.tk/
[10] (2010, Jan.) Sqlite home page. [Online]. Available: http://www.sqlite.org/
[11] T. Oetiker. (2012, Jan.) Mrtg - tobi oetiker's mrtg - the multi router traffic grapher. [Online]. Available: http://oss.oetiker.ch/mrtg/

Alberto Rodríguez Sánchez Actualmente es estudiante de la carrera en Ingeniería en Computación en la Universidad Autónoma Metropolitana Unidad Azcapotzalco. Su interés se ha centrado en el desarrollo de aplicaciones con orientación educativa.

Oscar Alvarado-Nava He graduated as Electronic Engineer from the Universidad Autnoma Metropolitana, Unidad Azcapotzalco, and earned mater's degree in Computer Science at CINVESTAV, I.P.N, Mexico City. Currently is Full Professor in the Departamento de Electrónica of UAM Azcapotzalco.

Francisco Javier Zaragoza Martínez was born in Mexico City. He obtained his bachelor degree in electronic engineering from UAM Azcapotzalco, his master's degree in Computer Science from CINVESTAV and his PhD from University of Waterloo. He is Full Professor at Departamento de Sistemas, UAM Azcapotzalco.