Diseño e implementación de un controlador domótico reconfigurable basado en hardware y software libre

J. Ruiz[1] , J.I. Villar.[1], M.J. Bellido[1], D. Guerrero[1], J. Viejo[1], P. Ruiz-de-Clavijo[1], J. Juan[1]
[1] Grupo ID2, Dpto. de Tecnología Electrónica, E.T.S. Ing. Informática, Universidad de Sevilla, España,
{jonathan, jose, bellido, guerre, julian, paulino, jjchico}@dte.us.es
http://www.dte.us.es/id2/

Abstract:

En este trabajo se presenta el diseño e implementación de un sistema empotrado que se emplea como nodo de control y comunicación en una red domótica heterogénea basada en buses cableados e inalámbricos. Para diseñar el sistema se ha empleado una plataforma basada en sistemas abiertos tanto a nivel de hardware como del sistema operativo implantada sobre dicho hardware. La implementación del sistema se ha llevado a cabo sobre una FPGA de bajo coste, aunque tiene la característica de que el diseño en si, es independiente de la tecnología de implementación. Para llevar a cabo el diseño, la plataforma base ha sido modificada tanto a nivel de hardware como del sistema operativo, para la funcionalidad que se requería. Posteriormente se ha completado el sistema con el desarrollo de la aplicación software de control del nodo de la red domótica.

Con este ejemplo de aplicación real queremos mostrar tanto que la tecnología de SoC basada en FPGA así como los diseños abiertos en hardware y software son adecuados para la implementación de sistemas reales.

1 Introducción

Los sistemas empotrados son aquellos sistemas que implementan una función específica para la cual se tienen que cumplir una serie de restricciones. Las restricciones principales son el bajo coste y consumo, así como un tamaño y peso lo mas reducido posible. Además, muy recientemente, se ha incrementado significativamente la demanda de sistemas empotrados con diferentes funcionalidades y con mayor complejidad en las mismas. Por último, debido a la gran competencia que hay en este campo, se ha convertido en una restricción fundamental el ser capaz de reducir significativamente los tiempos de desarrollo de estos sistemas [1].

Todos estos problemas han dado lugar a una evolución en la metodología de diseño que se aplica. Así, el principal cambio metodológico ha sido transformar en aspecto principal del diseño de sistemas empotrados no tanto el diseño del hardware específico sino, más bien, el desarrollo del software que implementa las diferentes aplicaciones. Esto ha sido posible gracias al avance microelectrónico que hoy en día permite el diseño completo de un sistema en un chip (i.e., System On Chip – SoC) lo que ha permitido partir en el diseño de plataformas hardware base bien desarrolladas. Efectivamente, la arquitectura hardware de un sistema empotrado en tecnología SoC consiste en uno o varios microprocesadores del sistema sobre los que se organizan diferentes periféricos necesarios para llevar a cabo la función deseada. Para completar el sistema empotrado, la aplicación concreta se implementa a través del software que se ejecuta en el microprocesador [2,3].

Para lograr las restricciones de reducción de costes y tiempo de desarrollo, las metodologías de diseño de sistemas empotrados parten de una plataforma hardware base sobre la que se pueden construir un amplio número de aplicaciones. Esta variedad en las aplicaciones se logra gracias a que las plataformas disponen de múltiples puertos que son capaces de manejar múltiples señales de diferentes tipos. Las plataformas mas comunes hoy en día siguen siendo los microcontroladores (MCUs) y los procesadores digital de señal (DSPs). Con plataformas hardware basadas en estos tipos de componentes, el diseño de la aplicación consiste, primordialmente, en el desarrollo del software. Para reducir los tiempos de desarrollo así como para poder emplear esos componentes en funciones de alta complejidad, los fabricantes de los mismos han echo un gran esfuerzo en los entornos de desarrollo software [4].

Así, hoy en día, las nuevas series de MCUs se caracterizan porque soportan sistemas operativos que facilitan significativamente el desarrollo de las varias aplicaciones que va a soportar el sistema final.

El mayor inconveniente que tiene los MCUs es que son plataformas hardware cerradas.

Como alternativa a las plataformas de hardware cerrado esta la posibilidad de emplear FPGAs [2,3,5,6,7]. La alta capacidad de integración, el bajo coste para tiradas bajas y medias, y el alto rendimiento, en términos de frecuencia de operación y consumo de potencia, hacen posible implementar un SoC en este tipo de dispositivo programable. Los SoC que se pueden implementar sobre las FPGAs tienen una serie de ventajas sobre las plataformas cerradas, como la adaptación a cualquier necesidad especificva o la reconfiguración dinámica.

Sin embargo, el uso de FPGAs implica necesariamente un esfuerzo para diseñar la arquitectura hardware que, evidentemente, incrementa el tiempo de desarrollo total del sistema.

Para hacer realmente útil el diseño de SoC sobre FPGAs en los sistemas empotrados es necesario facilitar la construcción de la arquitectura hardware a través de buenas metodologías de diseño, así como de un conjunto de herramientas CAD adecuado. De hecho, hoy en día, el principal esfuerzo que están haciendo los fabricantes va en la línea de mejorar las herramientas de diseño de SoC sobre FPGAs. Además, existe un alto nivel de competitividad por encontrar la mejor solución que combine la facilidad de diseño de la arquitectura hardware junto con el mejor rendimiento del sistema final. Ejemplos de estas herramientas son Xilinx EDK[8], Altera ESD [9], etc..

No solo es necesario facilitar el diseño hardware sino también el desarrollo software. Este es un punto donde, sin duda, hay una gran diferencia de madurez entre los MCUs y el diseño de SoC sobre FPGAs. Como mencionamos anteriormente, los MCUs soportan

sistemas operativos, lo que dota a estas plataformas de una alta versatilidad en cuanto a la funcionalidad y su complejidad, que puede alcanzar un sistema.
En contraste, los SoC sobre FPGAs, solo muy recientemente, empiezan a tener soporte para sistemas operativos [10] tales como petalinux MicroBlaze [11], o proyectos de portar Linux a Nios [12] o a Lattice Mico32 [13].
Un alternativa muy interesante de explorar en el diseño de SoC sobre FPGAs es emplear sistemas abiertos tanto a nivel de hardware como de software. Los sistemas abiertos se caracterizan porque son tecnológicamente independientes y pueden ser implementables tanto a nivel de FPGAs de bajo coste como, incluso, a nivel de ASIC. El principal problema que siempre se aduce para no emplear este tipo de sistemas es la dificultad que implican tanto a nivel de diseño hardware como de desarrollo software. Sin embrago, hoy en día empiezan a aparecer diversas plataformas hardware base abiertas en forma de diseño de SoC que han llegado a ser utilizadas en el diseño de sistemas empotrados reales [14,15].
En este trabajo vamos a presentar un nuevo ejemplo de sistema empotrado real que esta basado en una plataforma base hardware/software abierta como es el diseño ORPSOC [16]. El sistema que se ha desarrollado es una plataforma para la implementación de nodos de control y comunicación de una red domótica heterogénea basada en buses cableados e inalámbricos.
El resto de trabajo se organiza como sigue: el siguiente apartado esta dedicado a presentar el sistema domótico que se ha implementado; en el apartado 3 presentaremos el diseño del controlador de red domotica implementado, tanto a nivel de hardware como de software (firmware y aplicación) y, por último, presentaremos las principales conclusiones del trabajo.

2 Sistema Domótico

Actualmente, la automatización de nuestro entorno es un hecho innegable que lleva décadas desarrollándose, sobre todo y de un modo intensivo, en el ámbito industrial. Es en el entorno doméstico donde esta automatización ha tenido un desarrollo más lento, debido a multitud de factores, entre los cuales los más importantes han sido el alto coste económico y la baja rentabilidad, para el usuario final, de este tipo de inversión. Sin embargo, en los últimos años, hemos asistido a una explosión en cuanto al interés de los fabricantes en este campo, que se ha concretado en el nacimiento de multitud de alternativas para llevar un cierto grado de automatización no solo a grandes espacios públicos, sino también al hogar, ofreciendo precios razonablemente competitivos.
Así, paulatinamente se van introduciendo avances perceptibles en el día a día, haciendo a los espacios no solo más inteligentes y confortables, sino que además, la incorporación de automatismos, redunda en que estos espacios sean más seguros y accesibles.
Estas soluciones tecnológicas, que cada fabricante ha desarrollado de un modo independiente son de lo más heterogéneas en cuanto a su arquitectura y filosofía de funcionamiento. Así, existen multitud de aproximaciones en relación en relación a sus aspectos fundamentales de diseño, como el sistema de comunicación entre los distintos elementos, la filosofía de funcionamiento y organización lógica de la red de dispositivos, etc... Podríamos decir que casi tantas como actores participan en el mercado. Debido al problema que supone la imposibilidad de interoperar entre las distintas tecnologías de cada fabricante, en los últimos años, algunos de éstos se han asociado, formando consorcios estandarizadores en torno a tecnologías concretas. Así, hoy encontramos algunos estándares de facto, debido principalmente a su antigüedad o alta presencia. Un ejemplo de esto es el X10 [17], debido a su gran popularidad y alto nivel de implantación desde hace décadas en los Estados Unidos, por otro lado como ejemplo de estándar fruto de la agrupación de fabricantes uno de los más significativos es KNX[18], que nació fruto de la convergencia y estandarización de EHS [19], Batibus e Instabus [20].
Los estándares propietarios, sin embrago, tienen una serie de desventajas, pues no siempre facilitan la integración con dispositivos de terceros. Algunos utilizan chips propietarios para la capa de acceso al medio o incluso pueden llegar a cobrar carísimos precios en concepto de royalties, licencias, uso de patentes o membresías al consorcio estandarizador, todo esto para poder desarrollar y comercializar nuevos dispositivos. Como contrapunto a esto, basándose de algún modo en el auge de los sistemas abiertos, como los que rigen en redes heterogéneas como la misma Internet, surgen alternativas a la automatización con buses propietarios de lo más diversas, siendo éstas especialmente interesantes no solo para los usuarios finales, sino también para el mundo académico, pues resultan ser la plataforma perfecta sobre la que poder experimientar y llevar a la realidad cualquier tipo de idea en este ámbito. Es una red de este tipo la que hemos desarrollado y sobre la que se implantará el controlador de comunicaciones que se presenta en este artículo.
Una de las características de identidad de esta red de control, es su heterogeneidad con respecto al medio utilizado para la comunicación de dispositivos. Podemos clasificar estos medios en dos grupos: los utilizados entre los distintos nodos de control de la red y los utilizados entre los nodos de control y los dispositivos finales conectados en cada uno de los segmentos de la red.
Así, la topología de esta red a nivel lógico se estructura en dos capas. En la primera encontraríamos a todos los controladores de la red conectados entre sí, y en la segunda, dependiendo de cada controlador de la red, un número arbitrario de segmentos de control a los que se conectan los dispositivos finales. Un aspecto importante es que la arquitectura de esta red es independiente de los buses utilizados. En el estado actual del desarrollo, los segmentos de control están implementados utilizando los buses CAN para los segmentos cableados y ZigBee para los segmentos inalámbricos, CAN fue elegido debido no solo a su facil manejo y disponibilidad en casi cualquier familia de microcontroladores sino porque también tenía la ventaja de ser un bus multimaestro con prioridad, en el que cada uno de los nodos puede iniciar una transacción sin necesidad de solicitud por un controlador maestro de la red y fundamentalmente, por su altísima inmunidad al ruido electromagnético. Por

otro lado, ZigBee permite integrar nuevos dispositivos sin necesidad de establecer un enlace cableado con el controlador, lo que en este ámbito de aplicación, para algunos dispositivos y para facilitar la instalación es esencial. Así cada uno de los segmentos CAN, además del interfaz ZigBee están conectados a un controlador de red, que realiza una triple función, la de monitorización, la de comunicación y la de control. Estos controladores se comunican entre sí utilizando el protocolo IP. En el caso concreto de la implementación del controlador que presentamos, sobre Ethernet.

Con respecto al modo de operación del controlador, éste se modela como un procesador de eventos y reglas a partir de las cuales se generan otros eventos en respuesta a los recibidos desde los dispositivos finales. Sus principales funciones son las siguientes:

- Monitorización de la red: control del enlace tanto con los dispositivos conectados a sus distintos segmentos como con el resto de controladores. También la gestión de la respuesta ante mensajes de alerta de dispositivos de seguridad y de aspectos relacionados con la energía y parámetros del suministro eléctrico.
- Comunicación: gestión del paso de mensajes entre los distintos segmentos de la red, de modo que si un mensaje proveniente de un segmento fuese dirigido a un dispositivo de otro segmento o bien de otro controlador, se encargaría de realizar el rutado.
- Control: en la operación normal de la red, los distintos dispositivos emiten eventos y reciben mensajes o comandos que modifican su estado. Esta relación entre recepción de eventos y generación de comandos la lleva a cabo el procesador de reglas del controlador.

Otro aspecto fundamental de esta topología es la tolerancia a fallos, pues en el ámbito en que se aplican estas tecnologías es de vital importancia no solo asegurar un comportamiento correcto en todo momento, sino gestionar adecuadamente los posibles problemas que puedan surgir por causas externas o ante la caída de ciertas partes del sistema. Así, la presencia del controlador es detectada por el resto de miembros del segmento y por el resto de controladores en base a un tipo especial de mensaje denominado pulso o *heartbeat* que cada controlador emite con una determinada cadencia para señalizar al resto de elementos que está presente. De este modo, en cada segmento pueden participar más de un controlador, estando tan solo uno de ellos en activo en cada momento, y en reserva el resto. Ante el evento de un fallo en el controlador, si deja de detectarse el heartbeat del controlador activo, la red se reestructura, pasando alguno de los controladores pasivos a hacerse cargo del segmento que ha quedado huérfano.

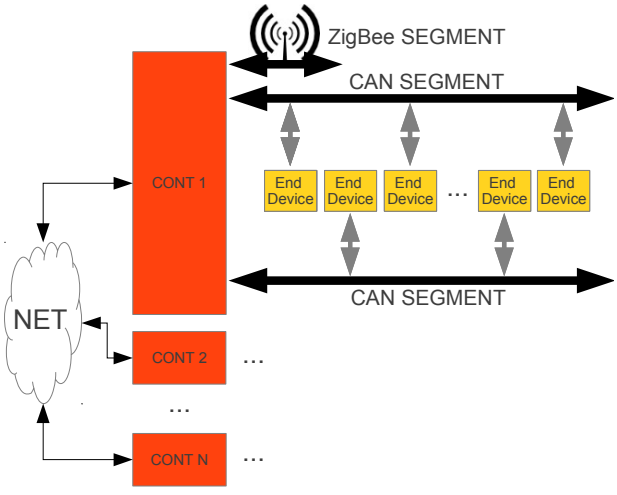

Figura 1. Arquitectura de la red de control.

Incluso en caso de que no hubiera ningún controlador de reserva, los nodos finales implementan un sistema de procesado básico de reglas de emergencia, de modo que se posibilita la operación autónoma intrasegmento en ausencia de controlador.

Con la arquitectura descrita, el uso de una plataforma basada en SoC sobre FPGA para desarrollar el controlador, aporta una versatilidad difícil de obtener en otras alternativas basadas en microcontrolador, pues las sucesivas mejoras y extensiones, no estan limitadas a nivel de software, sino que modificando el hardware se posibilita la adición de nuevos interfaces de comunicación de un modo modular y el soporte de nuevos buses de segmento. Actualmente, el SoC se comunica con los segmentos CAN y ZigBee mediante interfaces RS232. Asimismo, se posibilita que en futuras versiones, el rutado de paquetes sea llevado a cabo mediante un IP-Core, que descargue al microprocesador de las funciones de gestor de las comunicaciones. De este modo el router hardware estaría conectado directamente a los distintos segmentos externos y al microprocesador mediante un interfaz on-chip.

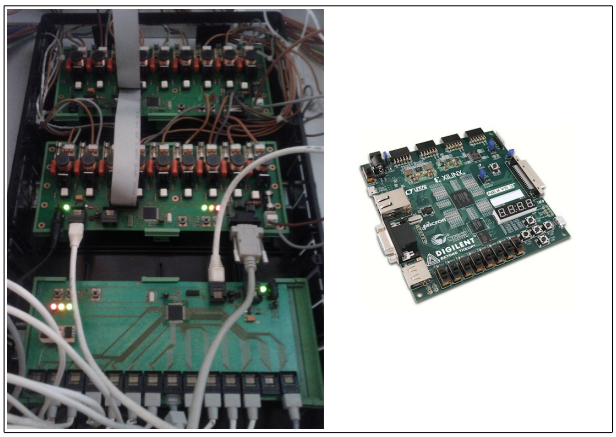

Figura 2: Módulo básico de iluminación (izquierda), placa de desarrollo del controlador (derecha)

Actualmente, hemos desarrollado dispositivos finales para la automatización básica de distintos aspectos, entre los que se encuentran:

- Controladores de iluminación (dimmer y conmutado).
- Sensores de estado en pulsadores e interruptores.
- Conmutación de carga mediante relays (motores, persianas, electroválvulas, enchufes).
- Monitorización del consumo eléctrico mediante sensado de la corriente.

- Sensores ambientales, de presencia, climáticos, gases, inundación, etc...

Este esquema de red, asimismo tiene un valor añadido pues aporta la posibilidad de introducir modificaciones al controlador para dar soporte e integrar tanto a nuevos sistemas domóticos, ya sean comerciales o libres, como a otros sistemas, que tecnológicamente han sido superados, pero que debido a su alto nivel de implantación, como en el caso del X10, aun siguen siendo un actor importante en este mercado.

3 Diseño del Controlador de Red Domótica

A. Plataforma Hardware

Como ya dijimos, el diseño hardware de nuestro sistema empotrado esta basado en la plataforma SoC abierta denominada ORPSoC (*OpenRISC Reference Platform System-on-Chip*) [16].

ORPSoC esta basado en componentes abiertos . Los principales elementos son el microprocesador OpenRisc 1200 [21] y el bus Wishbone [22].

En la Figura 3 se muestra la arquitectura básica del procesador OpenRisc1200. Es una procesador RISC con arquitectura Harvard de 32-bit. Esta diseñado en código Verilog [23] ocupando 32k lineas aprox. Es un código de fuentes abiertas bajo licencia LGPL [24]. Como características principales es importante destacar que es un diseño bastante configurable. Así por ejemplo, se pueden incluir o no tanto memorias cache como la unidad MMU. Por todos estos aspectos, OpenRisc 1200 es un procesador con una alta capacidad de adaptación y personalización para las diferentes necesidades y tecnologías de implementación que encontramos en el mercado como, por ejemplo, FPGAs, tecnologías semi-custom y tecnologías full-custom.

El bus wishbone está libre de patentes, royalties y copyright, lo que hace que su uso sea gratuito. Wishbone es un bus que se caracteriza principalmente por su simplicidad y flexibilidad. La simplicidad ha sido entendida como la capacidad de conseguir grandes tasas de datos con una complejidad de hardware mínima. A diferencia de otros estándares que definen uma jerarquía de varios protocolos, en Wishbone sólo existe uno, que es de alta velocidad con capacidad de adaptarse a dispositivos lentos. De este modo, en caso de necesitar conectar dispositivos de alta y baja velocidad es recomendable utilizar dos interfaces Wishbone, una para dispositivos de alta velocidad y otra para dispositivos de baja velocidad en lugar de complicar la gestión del bus. Desde el punto de vista de la flexibilidad, el estándar nos permite utilizar diversas topologías de bus (bus compartido, crossbar switch, punto a punto) y deja margen para configurar otros parametros, como los anchos de los buses, el significado de las etiquetas de datos y direcciones, endianness, niveles eléctricos, etc...

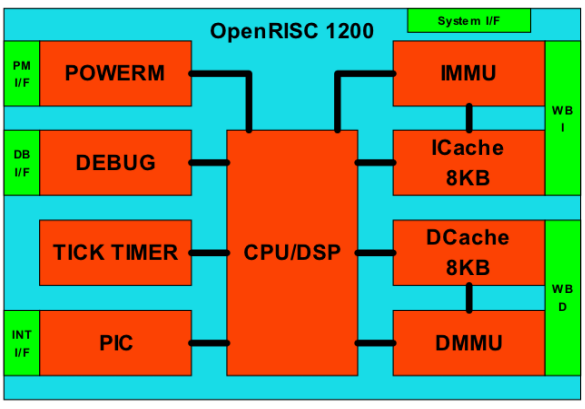

Figura 3. Arquitectura básica Openrisc1200

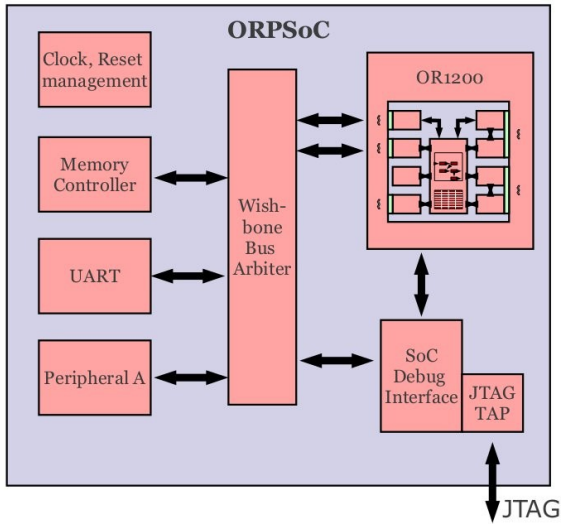

Figura 4. Estructura básica de ORPSOC[23]

Estos son los dos componentes principales entorno a los cuales se pretende articular una plataforma hardware/software completamente abierta desde el proyecto de OPENCORES [25]. Asi, por ejemplo, la mayoría de los IP Cores disponibles en OPENCORES tiene interfaz del bus Whisbone para poder emplearlos de una manera abierta y sin restricciones. Centrándonos en ORPSoC, este consta de un diseño de referencia, que es una versión con los componentes mínimos necesarios para el SoC, como son el procesador, la memoria, una unidad de debug y el bus de interconexión. (Figura 4). Además del diseño de referencia existe versiones de ORPSoC para diversas placas de desarrollo que, además, de los componentes mínimos, incluyen otro tipos de periféricos.

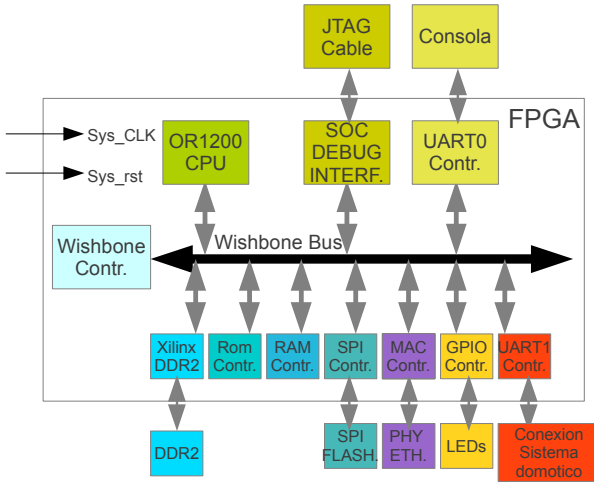

Figura 5. Sistema implementado en la placa ATLYS

En nuestro caso concreto hemos empleado la placa de desarrollo ATLYS para la implementación del sistema [26]. El SoC final que ha sido implementado es el que se muestra en la Figura 5. Destacamos en el SoC que se

implementa el controlador de DDR2 propio de Xilinx (esta automatizado en el proceso de síntesis). Al diseño base de ORPSoC para la placa ATLYS se le ha añadido una nueva UART (UART1) que es por donde se va a conectar al sistema domótico que se pretende controlar.
En la Tabla 1 se incluye un resumen de los recursos que el ocupa el SoC sobre la FPGA.

Device Utilization Summary. FPGA: Spartan-6 LX45	
Number of Slice Reg.:	6,991 out of 54,576 12%
Number of Slice LUTs:	12,868 out of 27,288 47%
Number used as Mem.:	472 out of 6,408 7%

Tabla 1. Resumen de ocupación del SoC en la FPGA

Analizando la Tabla I se observa como el SoC implementado no alcanza ni siquiera el 50% de los recursos disponibles en la FPGA lo que significa que el sistema es perfectamente ampliable a nuevos periféricos si fuera necesario.

B. Plataforma Software

Una vez descrita el hardware de la plataforma base vamos a describir el firmware o software básico sobre el que, posteriormente desarrollaremos el software de aplicación.
El microprocesador OpenRISC dispone de dos toolchains para el desarrollo de software, uno para aplicaciones en modo standalone y otro para aplicaciones bajo sistema operativo Linux. Hasta la versión del Kernel de Linux 3.1 se ha estado trabajando con un port realizado a partir de los fuentes de Linux de manera no oficial. Sin embargo desde el pasado Octubre la arquitectura de OPENRISC de 32-bits ha pasado ha estar soportado oficialmente [27], lo que, indudablemente mejorara los procesos de desarrollo de sistemas basados en esta arquitectura.
En cualquier caso, antes de poder desarrollar la aplicación es necesario desarrollar el firmware básico que incluya el kernel de Linux. En la Figura 6 se puede ver como se organiza en memoria los ficheros de programación y arranque del sistema que conforman el firmware del sistema. Como se observa, inicialmente se carga el hardware con una aplicación software de modo stand-alone. Esta aplicación no es mas que una pequeña rutina que busca en una dirección dada (en nuestro caso "0x1c0000") el tamaño de lo que tiene que copiar en RAM y posteriormente lo copia en la RAM y salta a la posición de RAM ejecutando ese código (en este caso, el bootloader u-boot [28]). U-boot es uno de los cargadores mas utilizados en los sistemas empotrados con Linux. Es un software que permite carga en tiempo real imágenes del kernel a través de diferentes interfaces, lo que es fundamental para el proceso de desarrollo de la imagen de linux en los sistemas empotrados. Este cargador permite, además, poder actualizar la imagen del sistema operativo de la plataforma en memoria flash en cualquier momento, lo que le confiere una gran versatilidad y capacidad de adaptación del sistema a nuevos requerimientos. Por defecto, u-boot esta preparado para cargar la imagen del kernel en memoria RAM y arrancar el sistema operativo.

Con respecto al kernel de Linux, a partir de los fuentes es necesario incorporar los módulos de los diferentes periféricos para poder acceder a ellos desde la aplicaciones. Además, para desarrollar la aplicación final se ha incorporado al kernel la aplicación gdbserver [29] para poder llevar a cabo la depuración de la aplicación final sobre el propio sistema empotrado.

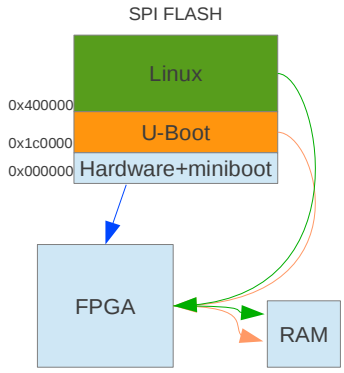

Figura 6. Plataforma software: Estructura del firmware en memoria Flash

C. Aplicación Software

Valiéndonos de la flexibilidad que aporta una plataforma basada en un sistema operativo de propósito general, la aplicación que realiza la operación del controlador ha sido implementada como una aplicación Linux que se ejecuta en espacio de usuario y ha sido desarrollada en lenguaje C.
De entre las tres funciones principales del controlador , las de monitorización y control han sido modeladas de un modo homogéneo mediante la implementación de un procesador de reglas capaz de detectar patrones de entrada, y en base a una configuración preestablecida, emitir mensajes de control a los dispositivos que corresponda.
Con respecto a la función de gestión de las comunicaciones, ya sean entre segmentos o de un mismo controlador o entre las distintas unidades finales directamente conectadas, al mismo, un módulo de rutado analiza las cabeceras para establecer el origen y el destino de cada mensaje. En función de éstos, el mensaje llegará al procesador de reglas local o bien será redirigido al controlador remoto que corresponda, al cual se le delegará la entrega.
Además de estas funciones, los controladores realizan una serie de tareas auxiliares de gestión tanto de la red, como de los nodos terminales. En lo relativo a la red, los controladores colaboran para reestructurar la topología en caso de fallo en alguno de los controladores, asumiendo alguno de los secundarios la labor de gestión del segmento huérfano. Con respecto a los nodos terminales, estos implementan un protocolo de gestión que permite que el controlador acometa acciones de mantenimiento, lo que permite que los controladores actualicen el firmware de los dispositivos si hay alguna nueva versión, comprobar el buen funcionamiento y reiniciar a los dispositivos si fuera necesario.
Con respecto a las funciones relativas a la gestión del enlace entre controladores, éstos exponen al exterior un interfaz basado en un protocolo estándar, en el que, de un modo similar al utilizado por UPNP [30], se permite

el autodescubrimiento de los recursos conectados a cada controlador y la ejecución remota de los métodos que implementa cada uno de estos recursos. El uso de un interfaz homogéneo y estándar como una red IP en esta capa, facilmente accesible desde cualquier nodo de la red, abre la puerta a una serie de interesantes posibilidades. Por un lado, permite que la configuración del funcionamiento y reglas asociadas a los controladores pueda ser llevado a cabo por una aplicación software desde un alto nivel de abstracción, facilitando enormemente la labor del usuario y no requiriendo hardware específico para ello. Por otro lado, la incorporación masiva de terminales con una alta capacidad de procesamiento como teléfonos móviles o tablets, hacen posible la implementación de controladores virtuales que interoperen con el resto de controladores de la red de un modo transparente.

4 Conclusiones

El mundo del diseño de sistemas domóticos esta, fundamentalmente controlado por tecnologías de fabricación que emplean estándares de interconexión y comunicación que son propietarias. Esto es un inconveniente a la hora de implementar sistemas en los que se pretendan utilizar dispositivos de diversos fabricantes ya que, generalmente, no son interconectables entre sí. Una alternativa a estos problemas puede venir del uso de sistemas abiertos que puedan adaptarse con mayor facilidad a la interconexión con cualquier tipo de dispositivo. Efectivamente, en este trabajo hemos demostrado que es posible emplear los sistemas abiertos en los sistema domóticos. En concreto se ha diseñado e implementado un controlador de comunicaciones de una red domótica.

El diseño del controlador se ha realizado a partir de una plataforma base hardware/software que emplea el microprocesador OpenRisc y sistema operativo Linux. Esta plataforma ha sido adaptada a nivel hardware y software para poder interactuar con la red domótica compuesta por buses CAN y ZigBee en sus diversos segmentos. La implementación final del controlador se ha realizado sobre una FPGA de bajo coste y opera correctamente cuando se interconecta al sistema domótico.

Con este trabajo podemos concluir que, efectivamente, los sistemas abiertos pueden ser una alternativa muy interesante de prospecionar a la hora de implementar los sistemas domóticos por la reducción de costes que supone frente a las alternativas comerciales existentes hoy en día.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación del Gobierno de España a través del proyecto TEC2011-27936 (HIPERSYS) y por el Fondo Europeo de Desarrollo Regional (FEDER).

Referencias

[1] Sangiovanni-Vincentelli, A. and Martin, G. 2001. Platform-Based Design and Software Design Methodology for Embedded Systems. *IEEE Des. Test* 18, 6 (Nov. 2001), 23-33. DOI= http://dx.doi.org/10.1109/54.970421

[2] Beeckler, J.S. , Gross, W.J., 2007. A Methodology for Prototyping Flexible Embedded Systems. Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on, 1679-1682

[3] A. Ben Atitallah · P. Kadionik · N. Masmoudi · H. Levi. FPGA implementation of a HW/SW platform for multimedia embedded systems . Des Autom Embed Syst (2008) 12: 293–311

[4] Salewski, F., Wilking, D., and Kowalewski, S. 2005. Diverse hardware platforms in embedded systems lab courses: a way to teach the differences. *SIGBED Rev.* 2, 4 (Oct. 2005), 70-74.

[5] J. Williams and N. Bergmann, "Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip," in *Proc. Eng. Reconfig. Syst. Algorithms (ERSA)*, Jun. 2004, pp. 171-176.

[6] Salewski, F. and Kowalewski, S. 2007. Hardware platform design decisions in embedded systems: a systematic teaching approach. *SIGBED Rev.* 4, 1 (Jan. 2007), 27-35.

[7] Nancy Eastman. 2005. Moving Embedded Systems onto FPGAs. 00 Embedded Magazine. w**ww.xilinx.com/publications .**

[8] (2009) EDK Concepts, Tools, and Techniques . **http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf**. Accedido 15 Mayo 2012

[9] (2009) **Nios II Processor Reference Handbook http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf**. Accedido 15 Mayo 2012

[10] Joachim Henkel, Mark Tins (2004) Munich/MIT Survey: Development of Embedded Linux . **http://ifipwg213.org/system/files/henkeltins.pdf&sa=X&scisig=AAGBfm3hRB26bXYYwHrUKhwjU55CstX78g&oi=scholarr** . Accedido 15 Mayo 2012

[11] (2010) Petalinux User Guide. **http://www.petalogix.com/resources/documentation/petalinux/userguide**. Accedido 15 Mayo 2012

[12] (2010) Nios Forum: Operative Systems. **http://www.alteraforum.com/forum/forumdisplay.php?f=38** . Accedido 15 Mayo 2012

[13] (2010) uCLinux for LatticeMico32. **http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/mico32uclinux.cfm** . Accedido 15 Mayo 2012

[14] Sebastien Bourdeauduck **http://milkymist.org/mmsoc.html** Accedido 15 Mayo 2012

[15] E. Ostua, A. Muñoz, P. Ruiz-De-Clavijo, M. J. Bellido, D. Guerrero and A. Millan : "Open Development Platform for Embedded Systems", book Grid Computing - Technology and Applications, Widespread Coverage and New Horizons, ISBN 978-953-51-0604-3. Ed. INTECH

[16] **http://opencores.org/openrisc,orpsocv2** Accedido 15 Mayo 2012

[17] Jon Burroughs , X-10 Home Automation Using the PIC16F877A , Microchip Technology Inc.., 2002.

[18] Knx Association, 2009. KNX System Specifications 3 1. Integration The Vlsi Journal, p.1-26.

[19] European Home Systems specification 1.3, European Home Systems Association, Brussels, 1997

[20] The EIB Handbook Issue 2.21. EIBA 1996.

[21] Openrisc 1200: **http://opencores.org/svnget,or1k?file=/trunk/or1200/doc/openrisc1200_spec.pdf** Accedido 15 Mayo 2012

[22] Bus Wishbone: **http://cdn.opencores.org/downloads/wbspec_b4.pdf** Accedido 15 Mayo 2012

[23] HDL VERILOG: **http://www.verilog.com/** Accedido 15 Mayo 2012

[24] LGPL, GNU Lesser General Public License: **http://www.gnu.org/licenses/lgpl.html** Accedido 15 Mayo 2012

[25] OPENCORES: **http://opencores.org** Accedido 15 Mayo 2012

[26] Placa ATLYS: **http://www.digilentinc.com/Products/Detail.cfm?Prod=ATLYS** Accedido 15 Mayo 2012

[27] Linux in Openrisc: **http://kernelnewbies.org/Linux_3.1#head-37c60fa1253db74ce7d224718a71f5836bd5be09** Accedido 15 Mayo 2012

[28] U-boot bootloader: **http://www.denx.de/wiki/U-Boot** Accedido 15 Mayo 2012

[29] GDBSERVER: **http://linux.die.net/man/1/gdbserver** Accedido 15 Mayo 2012

[30] UPnP Device Architecture 1.1 , UPnP-Forum., 2008.