
FIE' 08
Conferencia Internacional
5ta edición
Santiago de Cuba, Cuba
14 al 16 de Julio del 2008
UNIVERSIDAD DE ORIENTE
CIENCIA Y CONCIENCIA
GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN
CSIC
cigré
ISBN: 978-84-00-08680-0
NIPO: - - - -

FIE ' 08
international conference

Fifth edition
Santiago de Cuba, Cuba
From 14 to 16 July 2008

Santiago de Cuba, Cuba July 14-16 2008

FIE ' 08
International Conference
fifth edition

Home

HOME

THEMATICS

AUTOMATIC CONTROL

TELECOMMUNICATIONS

ELECTRIC

COMPUTER SCIENCE ENGINEERING

PATTERN RECOGNITION
AND BIOMEDICAL ENGINEERING

ENGINEERING LEARNING

topicname

Control Electrónico (Electronic Control)
Ingeniería Telemática. (Telematic Engineering)
Microelectrónica (Microelectronics)
Procesamiento Digital de Señales (Digital Signal Processing)
Sensores y Circuitos. (Sensors and Circuits)
Sistemas de Radiocomunicaciones (Radiocommunications Systems)
Sistemas Electrónicos y de Control de Potencia (Power and Electronic Systems)
Tecnologías e Infraestructura de Redes de Telecomunicaciones (Technologies and infrastructures for telecommunications networks)
Tecnologías para Aplicaciones y Servicios (Technologies for applications and services)
Tecnologías para el Desarrollo de Sistemas Distribuidos (Technologies for the Development of Distributed Systems)

HOME

THEMATICS

AUTOMATIC CONTROL

TELECOMMUNICATIONS

ELECTRIC

COMPUTER SCIENCE ENGINEERING

PATTERN RECOGNITION AND BIOMEDICAL ENGINEERING

ENGINEERING LEARNING

Title	Paper
AMPLIACIÓN DE PERIFÉRICOS PARA APLICACIONES EMBEBIDAS BASADAS EN HARDWARE Y SOFTWARE LIBRE	PDF
Aplicación CTI para consultas a bases de datos Oracle.	PDF
Aplicación de PicoBlaze al diseño de sistemas de control industrial	PDF
Detección de Fibras en la Calibración de un Mazo Incoherente para la Transmisión de Imágenes	PDF
Electro síntesis	PDF
Estrategia para la Migración hacia el Software Libre en la carrera de Ingeniería en Telecomunicaciones y Electrónica	PDF
Metodología de Diseño SoC con OpenRisc sobre FPGA	PDF
Procedimientos para el cálculo de la capacidad de transmisión en GPRS	PDF
SIGACIT: Sistema para la gestión de actividades de ciencia e innovación tecnológica	PDF
SIPNO: SISTEMA PROVEEDOR DE NOTICIAS ONLINE	PDF
Sistemas de localización en interiores: Tecnologías empleadas.	PDF

AMPLIACIÓN DE PERIFÉRICOS PARA APLICACIONES EMBEBIDAS BASADAS EN HARDWARE Y SOFTWARE LIBRE

A.Muñoz, E.Ostúa, M.J.Bellido, P.Ruiz-de-Clavijo, J.I.Villar,J.Quirós
Grupo de Investigación y Desarrollo Digital (ID2)
Departamento de Tecnología Electrónica - Universidad de Sevilla
Av. Reina Mercedes, s/n (E. T. S. Ingeniería Informática) - 41012 Sevilla (Spain)
Tel.: +34 954556160 - Fax: +34 954552764
email: { amrivera; ostua; bellido; paulino; jose; quiros } @ dte.us.es

Abstract *— This paper presents a method for developing embedded systems based on the LEON soft core microprocessor for which has been implemented a Linux kernel. The target implementation of the system is a FPGA from Xilinx. The article presents the main features of the base system, which consists of the LEON microprocessor and a Linux operating system distribution (Debian) running on it. Moreover, it shows a complete example of how to add new peripherals to the system. The peripheral that has been added is the UART 16550 compatible peripheral available in OpenCores. Given that the design has been prepared for the WishBone bus, it was necessary to adapt it to the APB bus within the LEON core. Furthermore, it has been adapted to work with the Linux driver for that UART to get a full coupling of peripheral with the system. The experimental results confirm the work done.*

Index Terms — Open Embedded Systems, Peripherals Interfacing, Linux and LEON Microprocessor, SoC applications.

I. INTRODUCCIÓN

Los últimos avances tecnológicos están permitiendo el desarrollo de chips de mayores densidades de integración, posibilitando la inclusión de todo un sistema en un único chip. Es lo que conocemos como metodología SoC. Disponemos de dos variantes tecnológicas para el diseño SoC: FPGAs y ASICs.

Usando la tecnología ASIC como objetivo de la implementación obtendremos unos rendimientos de consumo y velocidad superiores. Sin embargo, desde un punto de vista industrial, los costes de desarrollo y producción son muy elevados, siendo solo rentable si se pretende fabricar en grandes cantidades.

Por el contrario, implementar sistemas sobre una FPGAs, aunque presenten un rendimiento de inferior en velocidad, tienen un coste muy similar en desarrollo y en producción. Además esta tecnología posee la ventaja de ser reconfigurable en tiempo real, permitiéndonos reutilizar el mismo dispositivo para varias tareas distintas al mismo tiempo, siempre y cuando ninguna de ellas requiera un atención constante.

Por lo tanto, teniendo en cuenta que la mayoría de las aplicaciones industriales no requieren un gran número de unidades, la implementación FPGA se convierte en una solución apropiada en estos entornos.

Por otra parte, la mayoría de los SoC se construyen en torno a un microprocesador que actúa como el núcleo central del todo sistema. La elección de este microprocesador determina en gran medida el rendimiento obtenido. En la mayoría de las aplicaciones industriales no es necesario alcanzar grandes cotas en la capacidad de procesado, pero sin embargo, suele ser determinante obtener una buena fiabilidad del sistema. En consecuencia los microprocesadores utilizados en estos sistemas tienden a ser poco complejos.

En el diseño de soluciones SoC, es habitual el uso núcleos comerciales, a saber, ARM [1], PowerPC [2], NIOS3 [3], MicroBlaze [4] que requieren la compra de una licencia para su uso, o de otros, de código abierto, como los núcleos LEON [5] u OpenRisc [6].

Los microprocesadores comerciales, en general, están bien probados y optimizados, y disponen de sistemas que facilitan el desarrollo de la aplicación. Sin embargo, para determinadas empresas o industrias de los altos costes de adquirir la totalidad del paquete desarrollo comercial puede convertir el sistema a desarrollar en inviable. A pesar de que los sistemas abiertos tienen la clara ventaja de la disminución de costes, es generalizada la idea de que es muy complejo y poco fiable trabajar con estos sistemas.

Nuestro grupo de investigación lleva varios años trabajando en sistemas abiertos tanto desde la perspectiva del hardware como del software. El trabajo presentado en este artículo es parte de los últimos desarrollos del grupo en el diseño de SoC para aplicaciones embebidas.

En concreto, se ha desarrollado una plataforma basada en un SoC implementado sobre una FPGA. Este SoC se compone de diversos componentes de hardware abierto. El principal componente es el microprocesador LEON. El resto de los componentes del sistema se organizan en torno a un bus de especificación AMBA, que subdividimos en bus rápido y algo complejo denominado AHB, y un bus de periféricos, bastante simple, denominado APB.

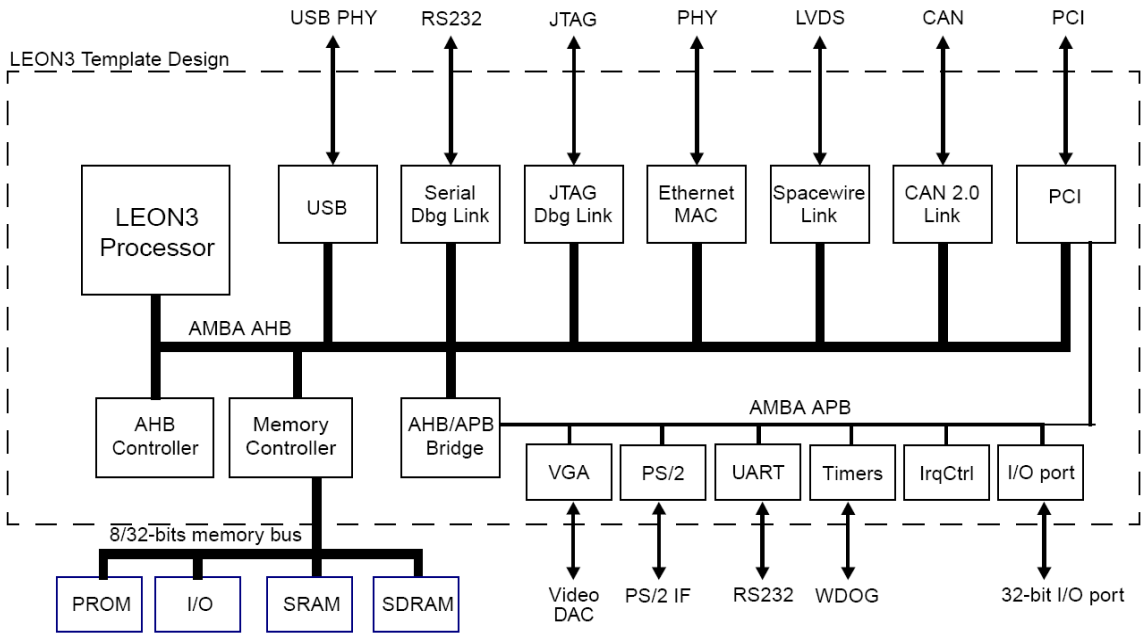

Figura 1 Arquitectura del sistema LEON (tomado de [5]).

Esta plataforma basada en hardware abierto se completa con una solución software también de código abierto como es Linux. Concretamente, se ha puesto en marcha un núcleo de Linux 2.6 sobre el que se ha implantado una distribución Debian [8][9]. Es importante mencionar que disponer de una distribución Linux como Debian funcionando en el SoC garantiza una enorme versatilidad en las aplicaciones que se pueden realizar. Entre otros motivos, la posibilidad de utilizar todo el software ya compilado y probado disponible en los repositorios de software de la distribución, tanto aplicaciones como bibliotecas, mejoran sustancialmente la complejidad y el tiempo de desarrollo de una solución basada en este SoC.

Con el fin de utilizar esta plataforma de desarrollo en diferentes aplicaciones, es necesario incorporar nuevos periféricos al sistema, y darles soporte en desde el nivel del sistema operativo. Como objetivo principal de este articulo, se pretende mostrar una metodología para alcanzar tal fin de forma satisfactoria.

En concreto, ha sido necesario para incorporar un controlador estándar serie RS-232, ya que aunque los desarrolladores de LEON nos facilitan uno, este es demasiado simple para determinadas situaciones, como por ejemplo para el control de dispositivos físicos de comunicaciones donde pueden ser muy necesarias las líneas de control de MODEM de las que carece.

A tal fin, se ha optado por un diseño abierto de un controlador de comunicaciones serie RS-232, concretamente el diseño de la UART 16550 disponible en OPENCORES.ORG [10].

Para incluir este diseño dentro de la plataforma base de que disponemos ha sido necesario realizar tareas de adaptación tanto a nivel hardware como software.

El resto del trabajo se organiza de la siguiente manera: en la siguiente sección se presentará la arquitectura hardware/software de un SoC basado en el microprocesador LEON y sistema operativo Linux-Debian; La dos siguientes secciones mostrarán la metodología a seguir en pos de adaptar la UART 16550 de Opencores a la plataforma de desarrollo.

En la quinta sección se muestran los resultados experimentales de una implementación concreta sobre tecnología FPGA. Por último, se presenta un resumen de las principales conclusiones del trabajo realizado.

II. DESARROLLO DE SISTEMAS BASADOS EN MICROPROCESADOR LEON

LEON es un microprocesador de 32 bits que incorpora un núcleo de arquitectura RISC que se ajusta a la especificación SPARC v8 [11]. Es un núcleo sintetizable escrito en VHDL y se puede ser implementado tanto en FPGAs como en ASICs. Se distribuye bajo los términos de la licencia GNU GPL, por lo que es un sistema de hardware abierto [12] y ha sido diseñado específicamente para aplicaciones embebidas.

Fue originalmente desarrollado por la Agencia Espacial Europea y hoy en día es mantenido por la empresa Gaisler Research. Debido a la naturaleza de la licencia GNU GPL, la empresa Gaisler Research también dispone de una versión comercial, que no obliga a distribuir el código fuente de la aplicación donde se haga uso.

LEON3 es un procesador de 32 bits de formato big-endian implementa un completo núcleo SPARC v8. Posee un juego de 72 instrucciones en 3 formatos de instrucción, así como 3 modos de direccionamiento (inmediato, desplazamiento e indexado). Implementa multiplicación y división con signo, la operación MAC (multiplicación y acumulación), así como una arquitectura segmentada (pipeline) de instrucciones en 7 etapas.

Con respecto a la caché, puede optarse por una implementación conjunta de instrucciones y datos, o bien por caches separadas de datos e instrucciones, lo que se conoce como Harvard Architecture [13].

La Figura 1 muestra el diagrama de bloques de una configuración típica de una aplicación SoC. Gran parte de estos bloques son opcionales, y pueden ser fácilmente removidos para adaptar el sistema a la aplicación concreta.

El modelo VHDL es totalmente sintetizable, soportando el uso de la mayoría de herramientas comunes de síntesis. Su alto grado de reconfigurabilidad y el uso del bus de especificación AMBA-2.0 AHB/APB como mecanismo de interconexión, permiten extender el sistema con cierta facilidad. En consecuencia, estas bondades convierten a LEON3 en un microprocesador ideal para el uso en entornos de aplicación SoC.

El uso del bus AMBA nos permite clasificar los dispositivos incluidos en el sistema según su complejidad y velocidad de operación. Aquellos que necesiten altas tasas de transferencia o mecanismos de comunicación más elaborados con el resto del sistema, se conectarán y harán uso de las características proporcionadas por el bus AMBA AHB. En este grupo encontramos la caché del sistema, el controlador de memoria RAM, el controlador Ethernet, etc..

Para el resto de dispositivos que no requieran altas tasas de transferencia ni mecanismos complejos de comunicación, la especificación AMBA 2.0 proporciona el bus APB, optimizado para operaciones simples y el bajo consumo. Se conecta al bus AHB mediante el puente AHB/APB que actúa como maestro del bus APB. En este bus es donde pretendemos conectar el nuevo periférico que añadiremos al sistema, la UART 16550.

La memoria del sistema puede ser controlada por distintos núcleos, pero normalmente se opta por un controlador programable de memoria que unifica el acceso a PROM, SRAM y SDRAM con capacidad para el direccionamiento de 2 GBytes. También está disponible un controlador para memoria DDR si fuera necesario.

Sobre esta configuración hardware del SoC se ha implementado un sistema operativo Linux, concretamente una distribución Debian, que proporciona los drivers necesarios para el funcionamiento de cada periférico perteneciente al sistema. Por último tendremos la capa de aplicación software, que interactuará con el sistema mediante las llamadas de sistema al núcleo Linux y a sus librerías.

La Figura 2 muestra una visión general de la arquitectura del sistema, partiendo del hardware físico que contendrá al SoC en la base, hasta la capa de aplicación de usuario en la parte superior.

III. Añadiendo un dispositivo UART 16550 en el sistema Leon

Después de analizar las posibilidades de la UART que acompaña al sistema LEON3 con motivo de la adaptación del sistema para una aplicación de comunicaciones concreta, observamos ciertas limitaciones importantes. La UART sintetizable incluida para integrar en el sistema LEON3 solamente dispone de las líneas de transmisión y de control de flujo, además de fuertes limitaciones de configuración. Con motivo de dotar al sistema de una UART capaz de proporcionar una funcionalidad completa, incluyendo, por ejemplo, las líneas de control de modem , decidimos optar por un modelo sintetizable, proveniente de OpenCores y de libre distribución, de una UART 16550, frecuentemente utilizada en los ordenadores de personales.

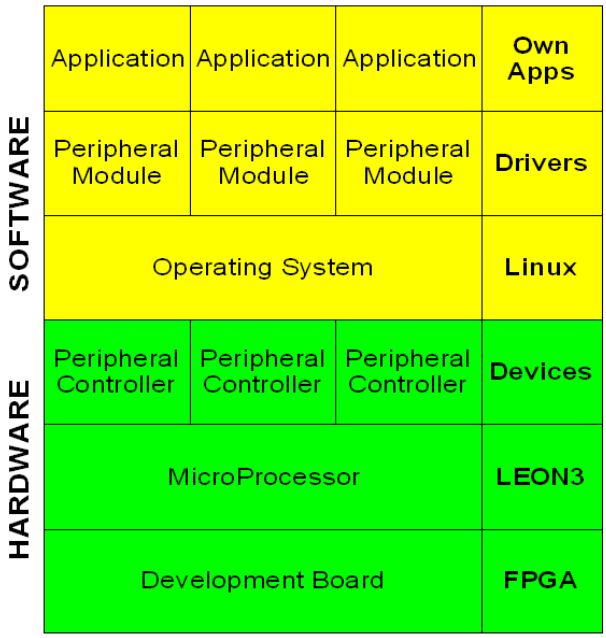

Figura 2 Arquitectura Global Hardware / Software

En primer lugar, será necesario adaptar el dispositivo al bus interno de nuestro SoC. La mayoría de los periféricos disponibles en OpenCores implementan la arquitectura de interconexión WishBone [14], un bus de especificaciones abiertas creado por la comunidad OpenCores y de uso frecuente en aplicaciones con sistemas embebidos. Sin embargo, como ha sido comentado con anterioridad, nuestro sistema dispone un bus AMBA 2.0 AHB y APB. En particular, utilizaremos el bus APB para conectar la nueva UART 16550, ya que se trata de un bus específico y preparado para este tipo de periféricos relativamente simples y lentos. La arquitectura y el comportamiento de ambos

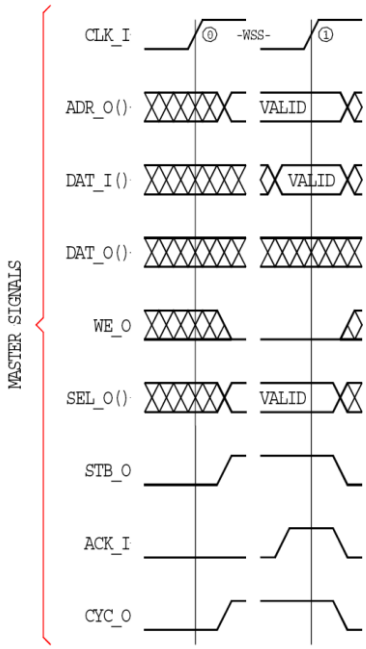

Figura 3 Lectura simple en bus WishBone (tomado de [14])

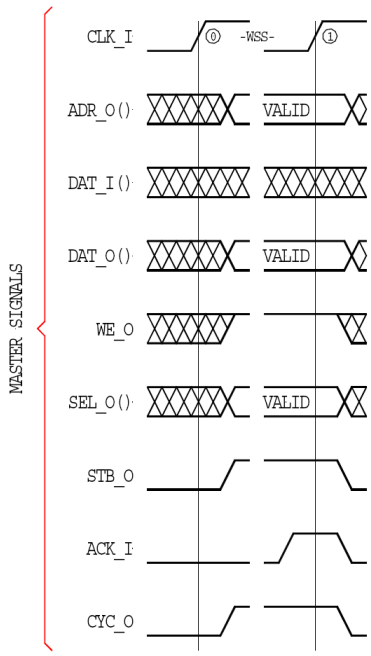

Figura 5 Escritura simple en bus WishBone (tomado de [14])

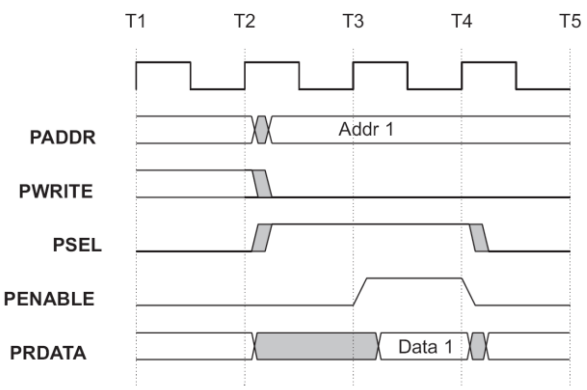

Figura 4 Lectura en bus AMBA APB (tomado de [7])

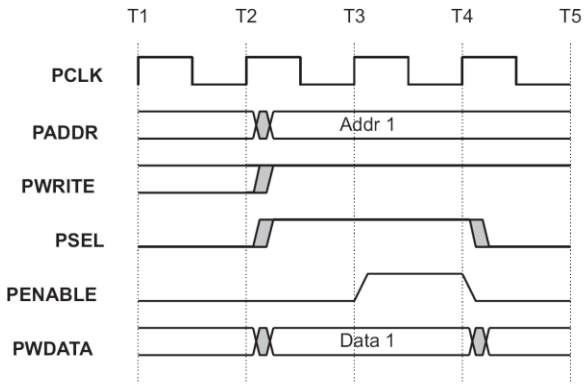

Figura 6 Escritura en en bus AMBA APB (tomado de [7])

buses (WishBone y AMBA APB) son algo diferentes en general, siendo la especificación WishBone más completa y funcional. Sin embargo, el mecanismo de la escritura y la lectura de los registros internos de un dispositivo es muy similar entre ellos. Ambos buses poseen 32 líneas de dirección así como dos buses independientes de datos de 32 bits cada uno, uno para las operaciones de lectura y otro para las de escritura.

A fin de crear una interfaz de adaptación entre ambos buses, es necesario estudiar las operaciones simples de lectura y escritura en los periféricos desde ambos buses.

Así, por ejemplo, una operación simple de lectura se muestra en las Figuras 3 y 4, mientras que una operación simple de escritura se observa en las Figuras 5 y 6, para el bus Wishbone y el bus AMBA-APB, respectivamente.

Después de un análisis preliminar, es evidente que la principal diferencia entre ambos buses en este tipo de operaciones simples reside en la señal *ack_o* del bus WishBone. Esta señal permite al dispositivo esclavo del bus insertar ciclos de espera en la operación, por ejemplo, permite mantener al dispositivo maestro a la espera de que el dispositivo esclavo complete una operación de transferencia.

Por contra, el bus AMBA-APB no tiene esta señal de reconocimiento, de modo que cada operación tiene siempre la misma duración. Esto implica que el dispositivo maestro nunca podrá esperar a que el dispositivo esclavo termine la

operación, es decir, las operaciones de lectura y escritura en el bus APB deben tener siempre la misma duración.
Esta diferencia es fundamental a la hora de prepara una interfaz de conexión entre ambos buses.
Así, la UART 16550 de OpenCores se comporta como un periférico esclavo en sus operaciones, con la particularidad de que si hace uso de la señal de reconocimiento *ack_o* a la hora de aceptar las transferencias de lectura de sus registros.
Por tanto actúa insertando estados de espera en el maestro, comportamiento que, a priori, impide realizar un simple adaptador de conexionado entre ambos buses apoyado simplemente en la lógica combinacional.
En conclusión, a fin de cumplir la especificación AMBA-APB es necesario hacer algunos cambios importantes en la implementación de la interfaz WishBone de UART 16550, de manera que eliminemos la inserción de los estados de espera en las operaciones de transferencia y respetemos el diagrama de estados del bus AMBA APB, como se muestra en la Figura 7.

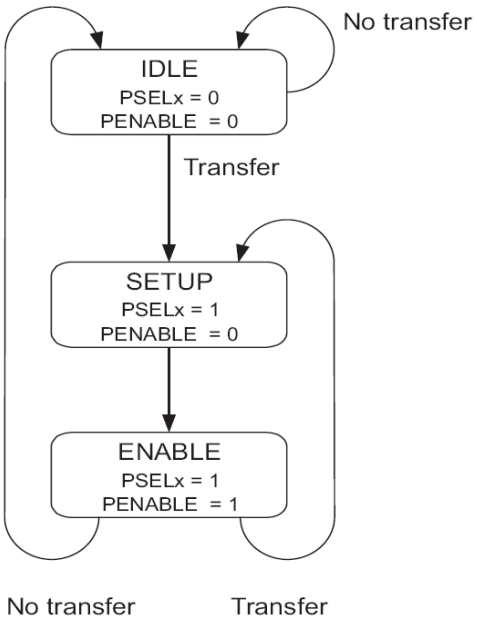

Figura 7 Diagrama de estados de ciclo de bus AMBA APB (tomado de [7])

Al tratarse de un modelo de libre distribución y de código abierto, facilitado por la comunidad de OpenCores, no tenemos más que analizar la interfaz de comunicación usada por la UART, escrita en lenguaje Verilog, para detectar la principal causa de la inserción de ciclos de espera.
Estos estados indeseados de espera son debidos al muestreo que se realiza de todas las señales de entrada en el dispositivo. Así, este muestreo y su subsecuente almacenamiento en registros intermedios, provoca al menos un ciclo de retraso en cada operación. Por tanto, necesitamos eliminar este comportamiento de muestreo, obligando a que los datos sean transferidos desde y hacia los registros internos de la UART de forma inmediata tras cada solicitud del bus.
También es necesario hacer una segunda modificación, debido a la disparidad entre el tamaño de los buses de datos (32 bits) y el tamaño de los registros de control internos de la UART (8 bits). Así, en nuestro diseño la dirección de byte en cada palabra está alineada con la parte menos significativa del bus de datos. Cada transferencia de datos en el bus se realiza en palabras de 32 bits, por lo que en cada acceso a registros de control consecutivos de la UART tendremos un desplazamiento de 4 posiciones de direccionamiento. En consecuencia, el nuevo periférico deberá ignorar los dos bits menos significativos del bus de direcciones provenientes del APB.
Desde el punto de vista de los lenguajes de descripción de hardware, es necesario modificar el núcleo periférico para incluir un nivel envoltura mayor que aporte esta funcionalidad. Esta entidad se encargará de la correcta alineación entre las señales de ambos buses y de la generación de la información para el sistema *Plug & Play* del que LEON dispone. La mayor parte de las conexiones se realizan manera directa, a excepción de algunas señales de control como la señal de reset que debe se invertida, o las señales *stb_i* y *cyc_i* del bus WishBone, que controlan el ciclo de transferencia y el ciclo de bus respectivamente, y serán generadas mediante la señal *apb_sel* del bus APB que controla la selección de dispositivo esclavo en el bus.
Además, proporcionar la información Plug & Play que incorpora LEON, permitirá la identificación del nuevo dispositivo y asignación de un mapa correcto de direcciones de forma automática.
Finalmente no debemos olvidar la conexión de la señal de interrupción de la UART con una línea libre del controlador de interrupciones que incorporamos al sistema.

IV. SOPORTE PARA EL DISPOSITIVO UART 16550 EN EL NÚCLEO LINUX

Los núcleos de Linux estándar proveen de soporte para numerosos modelos de dispositivos UART. En concreto, el modelo 16550 esta soportado por el driver denominado 8250.c. Sin embargo, este controlador está diseñado para dar soporte a diversas arquitecturas como la x86, pero no así para la arquitectura SPARC que utiliza nuestro sistema LEON. Por lo tanto, es necesario modificar y adaptar el driver para su funcionamiento en esta arquitectura.
En primer lugar, deberemos modificar los scripts de compilación del núcleo Linux, para permitir la inclusión de este driver aunque estemos compilando para la arquitectura SPARC. Una vez completado, comenzaremos la adaptación de los ficheros fuente del driver, comenzando por el fichero de definiciones de registros, denominado *serial_reg.h*. Como fue comentado con anterioridad, deberemos modificar las direcciones de cada registro en un desplazamiento de dos

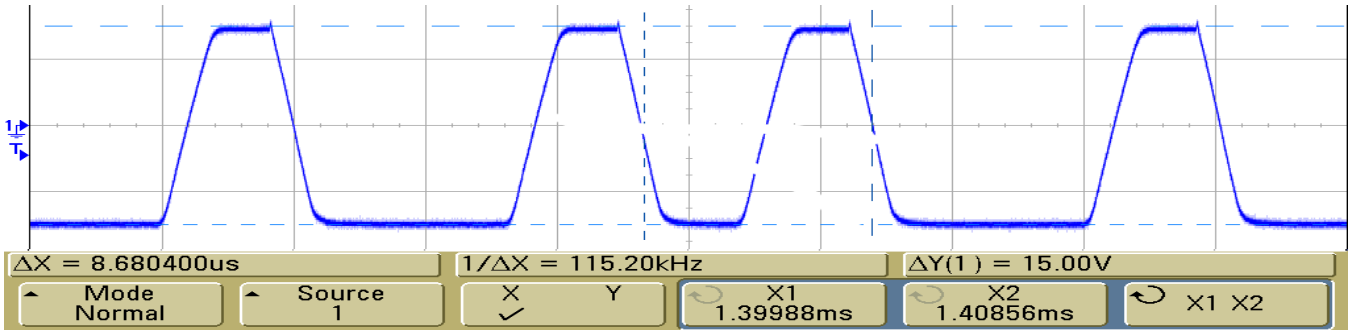

Figura 9 Señal Tx de transmisión de un carácter a través de la UART 16550

bits debido a la adaptación entre los buses APB y WishBone (en la práctica bastara con multiplicar cada registro por cuatro).

El mecanismo de detección que utiliza el driver para las distintas UART que se puedan encontrar en el sistema consiste en el sondeo de las direcciones de entrada salida donde se suelen encontrar típicamente estos dispositivos en cada arquitectura. Por ejemplo, para la arquitectura x86 se realiza el sondeo de cuatro puertos, desde COM1 a COM4 (Denominados /dev/ttyS0 a S3 en Linux), en sus correspondientes direcciones de entrada salida que van desde 0x3F8 hasta 0x2E8.

Esta información está contenida en la fuentes del driver, concretamente en el archivo *serial.h*, dentro del directorio *asm* de cada arquitectura específica.

En este punto, podremos optar por utilizar las funciones de búsqueda de dispositivos facilitadas por los mecanismos *Plug&Play* o bien, por asignar un mapa de direcciones fijo para el nuevo periférico UART 16550 que hemos añadido al sistema. Por simplicidad se implementó la primera opción tal y como lo hace el driver original para el resto de arquitecturas. Sin embargo, la inclusión de los mecanismos de búsqueda automática es una tarea relativamente simple en este punto, y aportarían la ventaja de no tener que recompilar el núcleo Linux si añadiésemos nuevos dispositivos extras o modificáramos sus posiciones relativas en el bus.

El último archivo fuente a modificar es precisamente el núcleo del driver, el fichero *8250.c*, donde, en primer lugar, reemplazaremos las funciones de escritura y lectura en el bus del sistema. Concretamente, debemos sustituir las funciones en lenguaje C , *inb* y *out*, usadas en la lectura y escritura de bytes sobre direcciones de entrada salida. Debemos adaptarlas al procedimiento de escritura en los buses de periféricos AMBA APB que implementa el sistema LEON. Con tal objetivo, utilizaremos las funciones Leon_bypass_load_pa y Leon_bypass_store_pa definidas en el archivo fuente *Leon.h* proporcionado por Gaisler Research en su distribución de Linux. Su tarea principal es llevar a cabo lecturas y escrituras de 32 bits no cacheadas directamente en el bus AMBA APB.

Un aspecto notable a modificar en el núcleo del driver es el parámetro que indica la frecuencia de reloj que rige la UART 16550. Por lo general, este tipo de UART vienen equipadas con un reloj de 1,8432 MHz, lo que limita su velocidad máxima a 115200 baudios. Es por ello, que el driver asume por defecto ese valor de frecuencia de reloj para realizar los cálculos de los divisores internos. En nuestro diseño debemos corregirlo al valor adecuado de 40 MHz, que es la velocidad de reloj utilizada por el SOC.

En último lugar, es necesario modificar las rutinas de detección del tipo de UART que implementa el sistema. Esta rutina se sirve de un registro que incorporan los dispositivos UART comerciales denominado registro cero o registro *scratch*, y cuya única finalidad es permitir al driver detectar el modelo exacto de UART que está gobernando.

Debido a que la UART 16550 de OpenCores utilizada no incluye este registro, debemos por tanto modificar la rutina de detección y obligarla a que configure la nueva UART como el modelo NS16550A que es compatible a nivel de registros con la nuestra.

V. RESULTADOS EXPERIMENTALES

Como dispositivo programable que contendrá el sistema, se ha utilizado una Spartan3 XC3S-1500-4 FPGA de Xilinx, que cuenta con una capacidad de un millón y medio de puertas lógicas. La ocupación del dispositivo dependerá en gran medida de las características que asignemos al microprocesador. Así dependerá por ejemplo del tamaño de las memorias caché y de los buffers de dispositivos como Ethernet, o también de las latencias elegidas para ciertas operaciones como la multiplicación. La dependencia del espacio ocupado por el sistema en relación a las latencias en el nivel de transferencia entre registros viene dada por la necesidad de las herramientas de síntesis de disponer de espacio suficiente para realizar el rutado de las señales críticas de sincronización como el reloj de sistema. Nuestro sistema funcionará a una velocidad de reloj de 40 Mhz, y en tales condiciones hemos ajustado las características del procesador para el mejor rendimiento posible, obteniendo una ocupación superior al 90% del dispositivo cumpliendo con las restricciones temporales, tal y como se puede observar en la figura 8. Para la obtención de estos resultados se ha utilizado el paquete de software Xilinx Ise Fundation.

```
Device Utilization Summary:

   Number of BUFGMUXs                  4 out of 8      50%
   Number of DCMs                      2 out of 4      50%
      Number of LOCed DCMs             2 out of 2     100%

   Number of External IOBs           221 out of 333    66%
      Number of LOCed IOBs           192 out of 221    86%

   Number of MULT18X18s                1 out of 32      3%
   Number of RAMB16s                  14 out of 32     43%
   Number of Slices                12809 out of 13312  96%
      Number of SLICEMs              391 out of 6656    5%


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 21241138 paths, 0 nets, and 106771
connections

Design statistics:
   Minimum period:  24.822ns (Maximum frequency:
40.287MHz)
   Minimum input required time before clock:   6.591ns
   Minimum output required time after clock:  12.193ns
```

Figura 8 Ocupación y temporización de la síntesis

Tras una compilación e instalación exitosa, se observa en la Figura 9, como el núcleo Linux ha detectado y configurado automáticamente el nuevo dispositivo. Esta figura presenta la información de inicio volcada en la consola de sistema en el proceso de arranque del sistema operativo.

Los resultados son bastante satisfactorios, la nueva UART implementa todas las características de los computadores de escritorio tales como el control de flujo o las líneas de control de MODEM.

El envío y la recepción de datos se realizan de forma adecuada en el rango completo de tasas de baudios estándares. En la Figura 8 se muestra una captura en el osciloscopio de la señal TX mientras se envía un byte desde la UART, con una configuración de 115200 baudios, 8 bits de datos, sin paridad y 1 bit de parada (es decir 115200@8N1).

Esta velocidad de transmisión puede ser fácilmente aumentada escribiendo valores menores en el registro divisor de reloj, en el interior de la UART, ya que no dependemos la limitación de 1,8432 MHz como se discutió con anterioridad.

VI. CONCLUSIONES

En este artículo se ha presentado el desarrollo de una plataforma hardware y software para aplicaciones SoC implementadas en tecnología FPGA. Caracterizándose por el uso de diseños basados en sistemas abiertos tanto hardware como software.

Concretamente, el núcleo hardware del sistema consiste en un diseño basado en el microprocesador LEON3 entorno a una arquitectura de bus de especificación AMBA.

```
Leon3Debian:~# dmesg | grep "NS16550A" C 5

grlib apbUART: system frequency: 40000 khz, baud rates: 38400 38400
ttyS0 at MMIO 0x80000100 (irq = 2) is a Leon
ttyS1 at MMIO 0x80000900 (irq = 3) is a Leon
Serial driver 16550 Opencore/APB, by D.T.E/U.S.
Looking for UART 2 as ttyS2: I/O address 0x80000c00
serial8250: ttyS2 at I/O 0x80000c00 (irq = 11) is a NS16550A
Looking for UART 3 as ttyS3: I/O address: 0x80000d00
serial8250: ttyS3 at I/O 0x80000d00 (irq = 13) is a NS16550A
loop: loaded (max 8 devices)
Probing GRETH Ethernet Core at 0x80000b00
10/100 GRETH Ethermac at [0x80000b00] irq 12. Running 100 Mbps full
duplex
PHY: Vendor 4de Device e Revision 2
Uniform MultiPlatform EIDE driver Revision: 7.00alpha2

Leon3Debian:~#
```

Figura 9 Mensajes de Sistema en Arranque del S.O.

En el nivel del software, se ha implementado un núcleo Linux que da soporte a la instalación de una distribución Linux completa como Debian. Esto añade una gran versatilidad a la plataforma, permitiendo la instalación de infinidad de aplicaciones y librerías de forma directa y sencilla, haciendo uso, por ejemplo, de los repositorios de software oficiales de la distribución Linux.

Además, se ha presentado una metodología a seguir para ampliar el sistema con nuevos periféricos de manera satisfactoria. En concreto se ha mostrado el proceso de completar un sistema LEON con un diseño basado en hardware libre de un dispositivo UART 16550, al que también se le ha dado soporte en el sistema operativo que gobierna el SoC.

En conclusión, el uso de sistemas basados en recursos de hardware y software abiertos se hace factible en aplicaciones industriales, donde son múltiples las ventajas desde el punto de vista del coste de desarrollo y de la versatilidad funcional, sin perjuicio del rendimiento obtenido.

VII. RECONOCIMIENTOS

Este trabajo ha sido parcialmente soportador por el Ministerio de Educación y Ciencia del Gobierno de España a través del proyecto TEC2007-61802/MIC (HIPER) y el Gobierno Regional Andaluz (Junta de Andalucía) a través del proyecto EXC-2005-TIC-1023.

VIII. REFERENCIAS

[1] Steve Furber: "ARM system-on-chip architecture, 2nd edition", Ed.Addison-Wesley 2000.

[2] "IBM PowerPC Quick Reference Guide", IBM Corp. 2005.

[3] "NIOS 3.0 CPU Data Sheet", Altera Corporation, 2004.

[4] “Microblaze Processor Reference Guide”, Xilinx Inc. 2005,
http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf

[5] Jiri Gaisler, Sandi Habinc, Edvin Catovic: “GRLIB IP Library User's Manual”, Gaisler Research, 2006,
http://www.gaisler.com/products/grlib/grlib.pdf

[6] Damjan Lampret: “OpenRISC 1200 IP Core Specification”, 2001,
http://www.OpenCores.org/cvsget.cgi/or1k/or1200/doc/or1200_spec.pdf

[7] “AMBATM Specification (Rev 2.0)”. ARM Ltd corporation, 1999.
http://www.arm.com/products/solutions/AMBA_Spec.html

[8] Debian GNU/Linux distribution, http://www.debian.org/intro/about

[9] A. Muñoz, E. Ostua, et al.: “Un ejemplo de implantación de una distribución Linux en un SoC basado en hardware libre”, III JCRA Workshop on Reconfigurable Computing and Applications, pp. 85-92, Zaragoza (Spain), 2007.

[10] OpenCores, http://www.OpenCores.org/

[11] “The SPARC Architecture Manual, Version 8”, SPARC International Inc., 1992.

[12] Graham Seaman: “Free Hardware: Past, Present & Future”, Erste Oekonux Konferenz, 2002.

[13] John L. Hennessy, David A. Patterson: “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann Publishers, Inc., 1990

[14] “WISHBONE SoC Architecture Specification, Revision B.3”, OpenCores Organization, http://www.OpenCores.org/