# Linux embebido como herramienta para realizar reconfiguración parcial

Oscar David Sánchez, Carlos Iván Camargo
Universidad Nacional de Colombia
{odsanchezg, cicamargoba}@unal.edu.co

***Abstract*— Las características de las *FPGAs*, tales como la reconfiguración parcial y dinámica, permiten el diseño de sistemas digitales, y en especial de sistemas embebidos, para aplicaciones con fuertes restriccones temporales. En este trabajo se presenta el desarrollo de una plataforma embebida que permite la reconfiguración de dispositivos externos (existe una unidad de control que configura, de acuerdo a sus necesidades, una *FPGA*). Se presentan las herramientas desarrolladas para la configuración de *FPGAs*, así como las ventajas que trae la utilización de la configuración parcial y dinámica junto con un sistema operativo basado en *Unix*, en este caso *uClinux*.**

***Palabras Clave*— Sistemas Embebidos, Reconfiguración parcial, SoC, Linux Embebido.**

## I. Introducción

Los dispositivos basados en lógica reconfigurable como las *FPGAs* fueron utilizados inicialmente para la implementación de tareas que exigían demasiado al procesador (i.e requerían muchos ciclos de máquina). Sin embargo, la creciente capacidad de integración ha permitido el desarrollo de circuitos más complejos en un solo chip. En particular, esto ha permitido que se puedan implementar procesadores y memorias en un dispositivo lógico programable. Por otro lado, estos dispositivos han sido dotados con características que permiten la reconfiguración parcial y dinámica soportando, por ejemplo, actualizaciones de Hardware desde sitios remotos, algoritmos adaptativos de Hardware y manejo más eficiente de espacio y potencia [1].

Actualmente los sistemas embebidos presentan una demanda creciente [2], y se exige de ellos la prestación de servicios tales como: interconexión en red, manejo de sistemas de archivos, interfaz adecuada con el usuario, entre otros [3]. El diseño de sistemas digitales, y en especial de sistemas reconfigurables, debe ir acompañado de un conjunto de herramientas de alto nivel apropiadas para reducir tiempo de desarrollo, manteniendo un desempeño aceptable. Es por esto que se han desarrollado herramientas de software que intentan hacer transparente el codiseño software-hardware manteniendo las prestaciones de la tecnología.

Los diseñadores tienen entonces una gran gama de herramientas para el diseño de sistemas, por lo que se espera que estos últimos sean más eficientes en términos de espacio y capacidad de cómputo. Sin embargo es importante el desarrollo de una plataforma genérica que permita reducir costos en el desarrollo de prototipos e implementación de sistemas reconfigurables [4].

Este trabajo está organizado de la siguiente forma: En la sección II se hace una breve descripción del hardware reconfigurable. En III se muestran las posibilidades que existen para implementar *SoCs* en *FPGAs*. En la sección IV se describe la importancia de contar con un sistema operativo, en este caso *uClinux*, para coordinar las tareas que debe realizar un sistema que use computación reconfigurable. Finalmente en V se muestra el diseño de la plataforma.

## II. Hardware Reconfigurable

Las *FPGAs* se pueden reconfigurar, a través de un archivo llamado *bitstream*, para que implementen diferentes circuitos lógicos. Este archivo se almacena en una memoria *RAM* interna y describe la forma en la que se deben configurar todos los bloques del dispositvo. En los últimos años los

fabricantes de *FPGAs* han incorporado en éstas características que permiten realizar la reconfiguración parcial [1], i.e, en el dispositivo se carga un *bitstream* (*bitstream parcial*) de menor tamaño que el que se utilizó para configurar la *FPGA* inicialmente, que solo contiene información de la configuración de un sector del dispositivo, manteniendo las funciones implementadas en los otros sectores. El proceso de reconfiguración parcial se puede realizar de forma dinámica[1]: el dispositivo no es enviado al estado de *reset*, manteniendo así el funcionamiento de los sectores no reconfigurados mientras ocurre el proceso de reconfiguración. La función de un módulo implementado en un sector no reconfigurado puede ser por ejemplo encargarse de las tareas de reconfiguración.

Xilinx Corporation ha desarrollado dos flujos de diseño para implementar la configuración parcial o parcial dinámica en sus *FPGAs* [1]: Flujo *diferencial* y *modular*. El primero consiste en realizar cambios a un diseño implementado previamente en la *FPGA* (hechos con *FPGA_Editor*[2] o modificando el código HDL) y después generar un *bitstream* que contiene las diferencias entre el diseño inicial y el modificado. Este flujo de diseño permite una rápida descarga del archivo de configuración porque solo se alteran sectores puntuales de la *FPGA*. Sin embargo, es un proceso que consume tiempo para generar el *bitstream* diferencial ya que es necesario sintetizar nuevamente todo el diseño así los cambios realizados sean pequeños.

El flujo de diseño *modular* busca disminuir el tiempo necesario para generar el *bitstream* parcial. Se divide el área de la *FPGA* en diferentes secciones de acuerdo a ciertas restricciones impuestas por el fabricante [1], [5]. El *bitstream* parcial describe la configuración de uno de estos sectores que ha sido instanciado como un módulo del sistema. La síntesis de este módulo, debido a que es una parte del sistema total, tarda menos tiempo que el gastado en la recofiguración diferencial.

[1]Como en las familias Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE y Spartan3 de Xilinx Corporation.

[2]*FPGA_Editor* es una herramienta que permite visualizar y editar la forma en la que se establecen las conexiones entre las *LUTs*, y las tablas de verdad que éstas implementan.

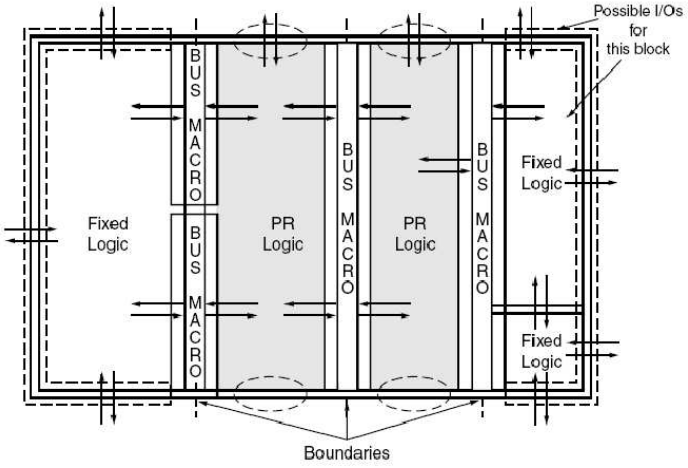


Fig. 1. Disposición del área fija y reconfigurable en una *FPGA*. Tomado de [1].

En la figura (1) se muestra un ejemplo de la forma en la que se dividiría el área de una *FPGA* para implementar la configuración modular. Existen diferentes sectores que implementan circuitos fijos (fixed logic) y reconfigurables (PR logic), cada uno de los cuales está confinado a un área fija. Siempre que se intercambie señales entre un módulo fijo y uno programable debe existir en la frontera un bloque denominado *bus macro*. Este bus macro es usado para garantizar que cada vez que se presente una reconfiguración, las señales externas del bloque modificado permanezcan en el mismo lugar evitando cortos y posibles daños al dispositivo.

La figura (2) muestra el diagrama observado mediante *FPGA_Editor* de un proyecto sencillo implementado con el flujo modular. La figura consiste en dos módulos: un oscilador y un registro, que componen el módulo fijo, y un subsistema que puede configurarse como sumador o restador. El oscilador en el bloque fijo tiene el propósito de evaluar la capacidad de reconfiguración dinámica del dispositivo (continúa funcionando en el proceso de reconfiguración del sumador/restador). El registro captura los datos de entrada y los pasa a través del *bus macro* para que el bloque reconfigurable los opere de acuerdo a la función, suma o resta, que esté implementando.

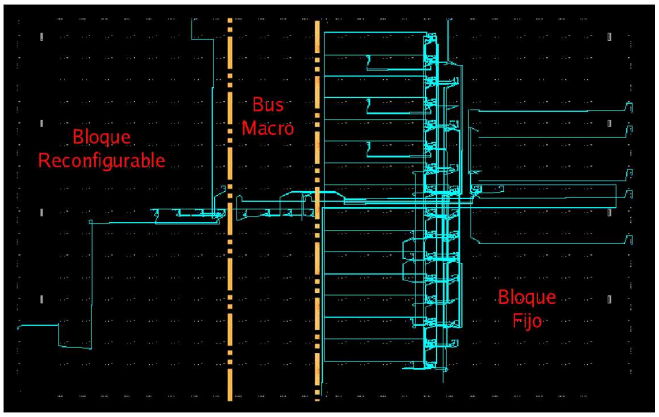


Fig. 2. Layout del diseño visto con *FPGA_Editor*.

## III. Sistemas embebidos en *FPGA*

Embeber un microcontrolador en una *FPGA* tiene múltiples ventajas: permite la reutilización y optimización de diseños anteriores evitando la obsolescencia de los mismos, al proporcionar una plataforma flexible [6]. Por otro lado, se logra disminuir el tiempo de desarrollo además de permitir, como se dijo anteriormente, actualizaciones de Hardware y Software de forma local y desde lugares remotos. Sin embargo, debido a las limitaciones de espacio y velocidad, los microcontroladores embebidos en *FPGA* resultan ser por lo general más lentos que los implementados en un chip creado para tal fin.

*MicroBlaze* [7] es un procesador (*soft-core*) *RISC* de 32 bits, arquitectura Harvard diseñado por Xilinx Corporation para sus *FPGAs* Virtex y Spartan-II/3. Es distribuido a través del *Embedded Development Kit* (*EDK*). La figura (3) muestra un diagrama de bloques del procesador. Se observa la distribución de los componentes de la CPU, los periféricos embebidos en la *FPGA* y dos memorias externas.

*MicroBlaze* se comunica con los periféricos y bloques de memoria a través de 2 buses: LMB (*Local Memory Bus*) y OPB (*On-chip peripheral Bus*). El primero es un bus asíncrono de alta velocidad. Admite solo un maestro y garantíza el acceso en un ciclo de reloj a la memoria *RAM* interna. El segundo, desarrollado por *IBM*, es usado para conectar periféricos y memorias externas. Soporta varios maestros.

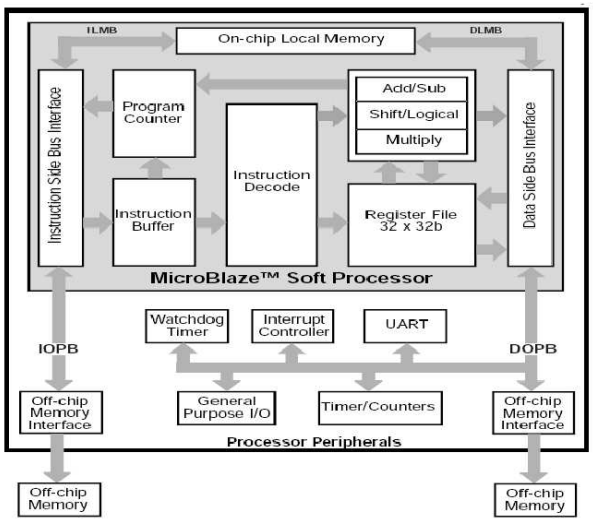


Fig. 3. Diagrama de bloques del núcleo del procesador *MicroBlaze*. Adaptado de [8].

## IV. Sistemas operativos y computación reconfigurable

Dadas las ventajas que ofrece el diseño de sistemas basados en Hardware reconfigurable, es necesario crear una arquitectura apropiada que permita un manejo del proceso de reconfiguración. En la actualidad este proceso se realiza con la intervención de herramientas especializadas diseñadas por el fabricante. En particular, Xilinx a través de su entorno de desarrollo EDK, proporciona herramientas para la creación de *drivers*, realización de simulación y depuración por software (*XMD*). Xilinx utiliza herramientas GNU (*gcc*) para la compilación del código de cada aplicación. Estas herramientas impiden la realización del proceso de reconfiguración en un ambiente embebido ya que están diseñadas para correr en computadores personales. Es por esto que es necesario diseñar herramientas software y hardware que permitan realizar el proceso de reconfiguración parcial de forma autónoma. En [9] se han discutido las características de un sistema operativo para ser usado en computación reconfigurable, y se ha argumentado que un sistema operativo embebido representa una mejor alternativa sobre sistemas basados en microkernel.

Trabajar con *Linux* trae importantes ventajas. En general, los sistemas operativos basados en *Unix*

proveen un conjunto de herramientas sencillas, pero que en conjunto (mediante el uso de por ejemplo segmentación y redirecionamientos), permiten la ejecución de tareas complejas. En particular, si tenemos un *bitstream* generado por las herramientas convencionales de síntesis para ser usado en la reconfiguración de cierta *FPGA*, se podría ejecutar el comando:

*$ cat bitstream.bit > /dev/port*

En donde */dev/port* representa el archivo mediante el cual se accede al puerto de configuración. Es importante notar que el *bitstream* puede estar almacenado en cualquier parte (memoria interna, externa o en un sistema de archivos remoto), sin alterar el comando utilizado.

El proceso de configuración también se puede llevar a cabo desde servidores remotos [10]:

*$ wget -O /dev/port*

*ftp://ftp.bitstreams.com/bitstream.bit*

Se observa entonces la forma en la que se simplifica el proceso de reconfiguración.

La figura (4) muestra la arquitectura del *kernel* de *Linux*. Está compuesta por 6 bloques: por un lado está el programador de tareas (Scheduler), el manejador de memoria, la comunicación entre procesos (IPC), que administran los procesos que se realizan, y por otra parte está el sistema de archivos, la interfase de red y los drivers, encargados de conectar el sistema con el exterior.

*uClinux* [12] es un sistema operativo basado en *Linux* diseñado para microprocesadores que no posee unidad de Unidad de Manejo de Memoria (*MMU*). Ha sido portado a múltiples microprocesadores, como *MicroBlaze* [13]. La arquitectura del *kernel uClinux* es similar a la mostrada en la figura (4), salvo que no existe el bloque manejador de memoria. La falta de *MMU* implica cambios importantes: no existe protección de memoria ni memoria virtual, la pila no se redimensiona automáticamente y algunas llamadas al sistema se ven afectadas haciendo que la multitarea sea compleja.

El desarrollo de *uClinux* se ha llevado a cabo mediante el uso de las herramientas libres del proyecto GNU tales como gdb (GNU debugger), gcc (GNU Compiler Collection) y binutils (colección de herramientas como *ld* (*linker*) y *as* (*assembler*)). Actualmente las distribuciones de *uClinux* se basan en el *kernel* 2.4 pero se está trabajando en las versiones de *kernel* 2.5 y 2.6.

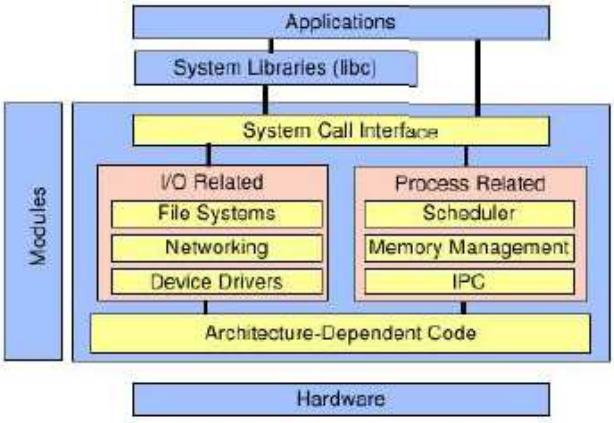


Fig. 4. Representación del *kernel* de *linux*. Tomado de [11].

## V. Plataforma de Desarrollo

### A. Descripción Hardware

En la figura (5) se muestra la plataforma utilizada. Fue diseñada por *Symbiosis* [14]. Esta plataforma presenta enormes ventajas para implementar un sistema embebido. El núcleo es una *FPGA* (XC3S400PQ208) fabricada por Xilinx Corporation, la cual se utiliza principalmente para implementar un *MicroBlaze* junto con algunos periféricos. La plataforma cuenta con una memoria *SDRAM* de 8MBIT, dos memorias flash: memoria NOR de 16-MBIT y memoria NAND de 1GBIT, una memoria PROM (XC04S), que hace parte de la cadena *jtag* para configurar la *FPGA* al ser alimentada, un controlador *Fast Ethernet* y un reloj en tiempo real.

La figura (6) muestra el diagrama de bloques del sistema finalmente implementado. En la *FPGA* se sintetizó un *MicroBlaze*. Este procesador cuenta con una memoria para almacenar datos e instrucciones (RAM_1) la cual se accede por medio de

dos buses (ilmb)[3] y (dlmb)[4]. La memoria Ram_2 provee, a través de la interfaz OPB_bram_if_ctl, memoria caché al procesador (4kbytes para datos y 4kbytes instrucciones). Existen 7 periféricos, que se comunican con la *CPU* mediante el *OPB* (On chip Peripheral Bus):

- Debug: Interfaz entre el sistema y un computador para realizar acciones de depuración.
- EMC (Externa Memory Controller): Interfaz entre el *microblaze* y la memoria *Flash*.
- Timer: Timer de 32 bits utilizado para manejar la multitarea por medio del sistema operativo (*uClinux*).
- Interrupt_ctl: controlador de interrupciones. Recibe las señales de interrupción del timer y la UART.
- Inter_SDRAM: Interfaz entre la memoria SDRAM externa y el OPB.
- GPIO: Puerto de 4 bits por medio del cual, mediante el protocolo JTAG, se reconfigura el dispositivo externo.
- UART: Comunicación serial por medio de la cual se pueden ver los mensajes del sistema operativo.

Fig. 5. Tarjeta utilizada.

[3]Instruction Local Memory Bus.
[4]Data Local Memory Bus.

### B. Descripción Software

Discutidas las ventajas que trae la utilización de un sistema operativo, es casi que imperativo implementar uno para la plataforma. Se ha logrado portar con éxito el *kernel* de *uClinux*.

Para compilar el *kernel* es necesario obtener la fuentes [13], compuestas de el *kernel* en si (uClinux-2.4.x) y las librerías y programas de usuario (uClinux-dist). Las fuentes se compilan, para una arquitectura específica, de acuerdo al archivo que describe la arquitectura del procesador (auto-config.in), que es generado por las herramientas de Xilinx Corporation. Una vez generado este archivo, se debe ingresar a la carpeta uClinux-dist y ejecutar el comando *make menuconfig*. En este momento se despliega un menú en el cual se escoge el fabricante. A continuación se despliegan diferentes menús en los que se puede configurar a la media la imagen del kernel. Una vez seleccionadas las opciones del *kernel* se ejecutan los comandos *make dep* y *make*, de la forma usual, para iniciar la compilación del *kernel*, después de la cual se obtiene este en binario.

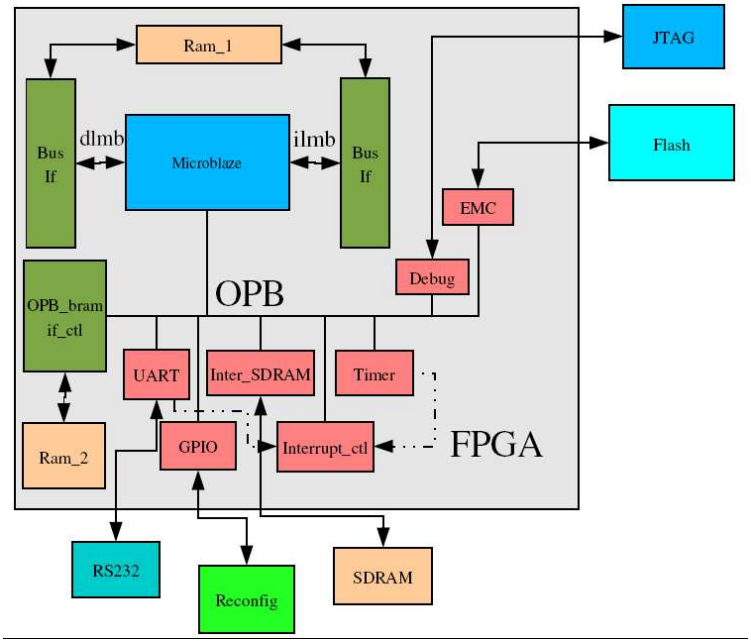


Fig. 6. Diagrama de bloques de la plataforma desarrollada.

Para realizar el proceso de reconfiguración de una *FPGA* conectada externamente a la plataforma, se

creó un módulo de *kernel* que maneja el puerto Reconfig (figura 6) mediante el *GPIO*. Un módulo es un trozo de código que puede ser añadido y eliminado del *kernel* sin necesidad de reiniciar el sistema. De esta forma se evita compilar un *kernel* cada vez que se quiera de este nuevas características. En particular, los drivers son módulos de *kernel* que le permiten a este acceso al hardware externo.

Cada dispositivo es visto por el *kernel* como un archivo, usualmente localizado en el directorio */dev*. En la plataforma diseñada, el puerto Reconfig se accede mediante el archivo */dev/jtag*. Al leer el archivo mediante, por ejemplo, el comando *cat /dev/jtag*, el driver detecta si existe algún dispositivo conectado. Escribir algún *bitstream* en este archivo origina la programación de la *FPGA* externa.

Inicialmente se encuentra almacenada, en la memoria *flash* (figura (6)), la imagen del *kernel* de *uClinux*. Cuando se alimenta la plataforma, el *MicroBlaze* empieza a ejecutar un programa almacenado en la memoria local Ram_1, que copia el contenido de la *flash* en la *SDRAM*. Después de esto empieza la ejecución del *kernel* desde la *SDRAM*.

## VI. Conclusiones

El diseño de sistemas digitales debe ir acompañado de herramientas apropiadas para aprovechar el rápido crecimiento de la capacidad de integración de los dispositivos actuales. La utilización de un sistema operativo, para administrar y controlar los recursos de un sistema, se está convirtiendo en un requisito básico, ya que permite reducir considerablemente tiempo de diseño sin afectar significativamente el desempeño.

El Hardware Reconfigurable brinda nuevas posibilidades de diseño. Sin embargo, es importante desarrolla herramientas que permitan aprovechar el potencial que ofrece. *uClinux* es un sistema operativo apropiado para tal fin, ya que siguiendo la filosofía de los sistemas basados en *Unix*, permite la ejecución de tareas poderosas mediante comandos sencillos.

La utilización de un sistema operativo libre trae ventajas importantes ya que disminuye los costos en la fabricación de dispositivos comerciales. Existen diversas herramientas que facilitan la labor de los diseñadores. Además, al tener acceso al código fuente, se pueden crear sistemas ajustados a necesidades particulares, imposibles de conseguir de otra forma.

## Referencias


[1] Xilinx Inc. *Two flows for partial reconfiguration: Module Based and difference based*, 2004. Application Note 290.

[2] E. Sánchez. El futuro de la infomática y la desaparción del computador. Medellin, 2005.

[3] N.W. Bergmann J.A.Williams and X. Xie. Fifo communication models in operating systems for reconfigurable computing. *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

[4] John Williams Neil Bergmann and Peter Waldeck. *Egret: A Flexible Platform for Real-Time Reconfigurable System on Chip*. School of ITEE, The University of Queensland. Brisbane, Australia.

[5] Gregory Mermoud. *A Module-Based Dynamic Partial Reconfiguration tutorial*. Logic Systems Laboratory, École Polytechnique Fédérale de Lausanne, Noviembre 2004.

[6] Carlos Augusto Juménes Millán y William Serrato Gutíerrez. *Diseño e implementación de un sistema embebido para manejo de periféricos sobre una FPGA*. Universidad Nacional de Colombia, 2005.

[7] Xilinx Inc. *Microblaze Processor Reference Guide*, 2003.

[8] Xilinx Inc. *MicroBlaze RISC 32-Bit Soft Processor*, August 2002. LogiCORE description.

[9] John Williams and Neil Bergmann. *Reconfigurable Linux for Spaceflight Applications*. School of ITEE, The University of Queensland. Brisbane, Australia.

[10] John Williams and Neil Bergmann. *Embedded Linux as a platform for dynamically self-reconfiguring systems-on-chip*. School of ITEE, University of Queensland Brisbane, Australia.

[11] An introduction to the linux architecture. http://wiki.cs.uiuc.edu/cs427/An+Introduction+to+the+Linux+Architecture.

[12] uclinux. Embedded Linux Microcontroller Project. http://www.uclinux.org/.

[13] uclinux. Microblaze uClinux Project. http://www.itee.uq.edu.au/ jwilliams/mblaze-uclinux/.

[14] Symbiosis. http://www.symbiosis.com.co/.