



Artículos

Arquitecturas de Procesadores



Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos

Micolini, Orlando
 Laboratorio de Arquitectura de Computadoras
 FCEFyN - UNC
 Córdoba, Argentina
 omicolini@compuar.com

Pereyra, Martín
 tinchommp@gmail.com

Gallia, Nicolás A.
 ngallia@gmail.com

Alasia, Melisa A.
 alasiameli@gmail.com

Resumen— En este informe se describe el desarrollo de un mecanismo de hardware para mejorar la sincronización en una arquitectura multi-core, haciendo uso de los formalismos de Petri. Se confecciona un módulo que se interconecta con dos procesadores Microblaze. Luego, es implementado en una FPGA; constituyéndose así, un sistema multi-core heterogéneo con capacidad de sincronización por hardware.

Palabras Claves— FPGA, IPCore, Microblaze, Multi-Core, Petri Net, Sincronización, SoftCore.

I. INTRODUCCIÓN

Tanto la exclusión mutua como la sincronización tienen un importante rol en la programación paralela de sistemas multi-core; como solución a esto, se han desarrollado diversos métodos para el acceso a recursos compartidos y su sincronización, implementados por software; con el consiguiente aumento de tiempo en la ejecución.

El presente trabajo muestra una solución descripta en hardware, sobre una arquitectura multi-core utilizando FPGA[1]. ; sin la penalización de tiempo añadida mediante las soluciones por software. Para llevarlo a cabo, se parte de investigaciones previas [6]. [7].

Incluir esta solución en un sistema multi-core homogéneo permite desacoplar, de los procesadores, las responsabilidades de sincronización y exclusión mutua; delegándolas al nuevo módulo programable, el cual funciona bajo los formalismos de Petri.

A. Objetivo General

Diseñar e implementar un procesador que utiliza el formalismo de Petri para la sincronización en una arquitectura multi-core sobre una FPGA.

La Figura 1, describe esta arquitectura.

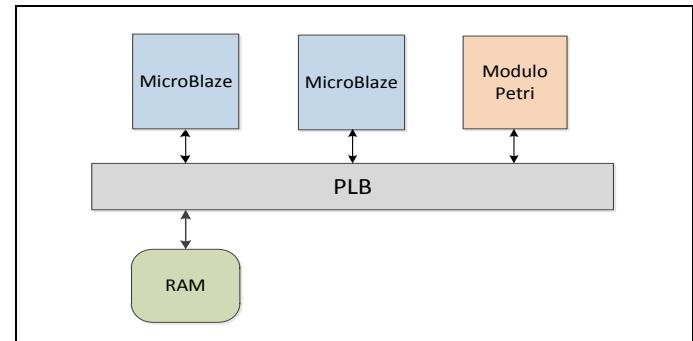


Figura 1: Arquitectura a desarrollar

B. Objetivos Específicos

- El módulo de Petri debe ser programable, inicializando los componentes que definen a la red de Petri (o Petri Net, PN) a utilizar desde RAM (ver Figura 1).
- Los microprocesadores tienen que ser capaces de cargar los programas a ejecutar desde la memoria principal.
- Los microprocesadores deben enviar eventos para la sincronización, al procesador de Petri (PP), si un evento produce un disparo posible se modifica, según su prioridad, el estado.
- El PP tiene que resolver y responder a los eventos tan rápido como sea posible.

II. MARCO TEÓRICO

A. Field Programmable Gate Arrays (FPGAs)

Son circuitos integrados formados por bloques lógicos programables e interconectables entre sí. Permiten implementar sistemas de alta complejidad y flexibilidad, pudiendo combinarse con periféricos y microprocesadores; para formar los llamados «Sistemas programables en chip».

Otra alternativa, es hacer uso de núcleos de procesadores implementados haciendo uso de la lógica de la FPGA (Softcores), teniendo en cuenta que los bloques programados pueden ser ejecutados concurrentemente.



B. Intellectual Property Core (IP Core)

Es un bloque o modulo reutilizable, de lógica o esquemático, para ser usado en diseños implementados en FPGAs [3].

Las características que lo identifican son:

- Funcionalidad pre-definida.
- Tamaño pequeño.
- Creado con el uso de estándares.

Pueden clasificarse de la siguiente manera:

- Hard Cores.
- Firm Cores.
- Soft Cores.

En la Tabla 1 se hace una descripción de los distintos tipos de cores.

C. Soft Core o Soft CPU

Software Microprocessor, es implementado completamente utilizando un semiconductor programable. Particularmente destacados por su alto nivel de configuración, pudiendo agregar y quitar funcionalidades en base a las necesidades específicas [12].

El objetivo de embeber una CPU es integrarlo con la lógica necesaria sobre la FPGA. Esto implica, que la CPU se encargue de la ejecución de programas secuenciales y cálculos; mientras que la lógica de la FPGA se encarga de las interfaces, además de facilitar el procesamiento paralelo. En cuanto a performance, varía respecto a cada implementación, dependiendo de las características inherentes a la CPU, como tipos y tamaños de cache, buses, controladores, etc.

Tabla 1: Resumen de los distintos tipos de cores

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

D. Redes de Petri

Es un modelo gráfico, formal y abstracto para describir y analizar el flujo de información. Conforma una herramienta matemática que puede aplicarse especialmente a los sistemas paralelos que requieran simulación y modelado de la concurrencia en los recursos compartidos. [4].

Están asociadas con la teoría de grafos y se pueden considerar como autómatas formales y generadores de lenguajes formales.

Los siguientes componentes conforman una PN:

- Lugares o Plazas: Permiten representar estados del sistema.
- Tokens: Representan el valor específico de un estado.
- Transiciones: Representan el conjunto de sucesos que provocan la modificación de los estados del sistema.
- Arcos dirigidos: Indican las conexiones entre lugares y transiciones.
- Peso: Número entero asociado a cada arco.

El disparo de una transición retira tantos tokens de cada uno de sus lugares de entrada como lo indican los pesos de los arcos conectores y añade los tokens a sus lugares de salida como lo indican los pesos de los arcos de salida.

En la Figura 2 se observan los componentes de una PN.

1) Definición Matemática

Una PN se compone de los siguientes elementos:

1. $L = \{l_1, l_2, l_3 \dots l_m\}$, conjunto de m lugares con m finito y distinto de cero.
2. $T = \{t_1, t_2, t_3 \dots t_n\}$, conjunto de n transiciones con n finito y distinto de cero. (Siendo T y L conjuntos disjuntos).
3. Marcado, es la distribución de los tokens en los lugares. Se corresponde con estados específicos de la red. Es un vector de dimensiones $m \times 1$ siendo m la cantidad de elementos del conjunto L .
4. I^- , matriz de incidencia negativa, de dimensiones $m \times n$, representa los pesos de los arcos que ingresan desde los lugares L a las transiciones T .

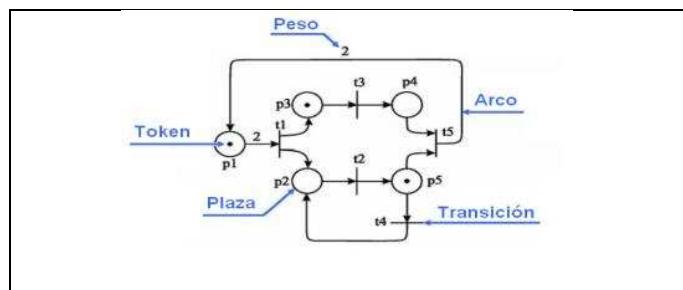


Figura 2: Componentes de un PN

5. I^+ , matriz de incidencia positiva, de dimensiones $m \times n$, representa los pesos de los arcos que salen desde las transiciones de T hacia los lugares de L .

A partir de estas dos últimas matrices, se obtiene la Matriz de Incidencia:

$$I = I^+ - I^- \quad (1)$$

2) Transiciones sensibilizadas

Se dice que t_i está sensibilizada si y sólo si todos sus lugares de entrada tienen al menos la cantidad de tokens igual al peso de los arcos que los unen a t_i .

$$\forall l_i \in I^-(t_i) \rightarrow m(l_i) \geq W_{ij}, \text{ con } W_{ij} \text{ peso del arco que une } l_i \text{ con } t_j$$

3) Disparo de una transición

Ocasiona el cambio de estado de una red. La función de disparo ∂ para una transición sensibilizada t_j establece:

$$\partial(m_k, t_j) \quad (2)$$

De la Ecuación (2) se tiene:

$$\begin{aligned} m_{k+1}(l_i) &= m_k(l_i) - W_{ij} \\ \forall l_i \in \text{Conj. de lugares de entrada a } t_j \\ m_{k+1}(l_i) &= m_k(l_i) + W_{ij} \\ \forall l_i \in \text{Conj. de lugares de salida a } t_j \\ m_{k+1}(l_i) &= m_k(l_i) \text{ para otros casos} \end{aligned}$$

4) Ejecución de una PN

Es la secuencia de pasos que resultan de disparar la red n veces partiendo desde el marcado inicial m_0 .

5) Ecuación de Estado

La ecuación de estado de una red de Petri queda definida en función de la matriz de incidencia I , y de un vector de disparo σ_i de dimensión $1 \times n$, cuyas componentes son todas nulas, excepto la que corresponde a la transición disparada en el instante i , que vale 1.

$$m_{i+1} = m_i + I \times \sigma_i \quad (3)$$

6) Plazas acotadas

Un lugar se dice que es acotado para un marcado inicial, si existe un número entero k tal que, para todo el conjunto de marcados alcanzables desde el marcado inicial, el número de marcas en ese lugar, no es más grande que k .

Una PN es acotada para un marcado inicial si todos los lugares son acotados para el estado inicial (la PN es k -acotada si todos los lugares son k -acotados).

7) Arcos inhibidores

Las redes de Petri tal cual se han definido no poseen la capacidad para representar la posibilidad de disparo de una transición ante la ausencia de tokens en un lugar. Para lograr esto, se debe agregar un arco inhibidor, como se muestra en la Figura 3.

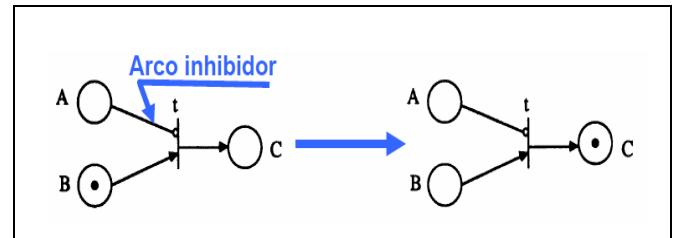


Figura 3: Ejemplo de disparo con arco inhibidor

Se dice que una transición t_j está sensibilizada si y sólo si se cumplen las siguientes condiciones:

- Todos sus lugares de entrada, unidos mediante arcos dirigidos tienen al menos la cantidad de tokens igual al peso de los arcos que los une a t_j .

$$\forall l_i \in I^-(t_j) \rightarrow m(l_i) \geq W_{ij} \text{ con } W_{ij} \text{ peso del arco que une } l_i \text{ con } t_j$$

- Todos sus lugares de entrada, unidos mediante arcos inhibidores a t_j no contienen tokens.

$$\forall l_i \text{ unido con arcos inhibidores a } t_j \rightarrow m(l_i) = \emptyset$$

La ecuación de estado de la red debe incluir una matriz que represente aquellos lugares con arcos inhibidores. Las demás características definidas para redes de Petri simples no cambian.

a) Ecuación Ejecutable de Arcos Inhibidores [11].

$$M_{i+1} = M_0 + I * (d \text{ and } \overline{f(M_0, d)}) \quad (3)$$

De la Ecuación (3) se tiene:

$$f(M_0, d) = \text{MayorØ}(M_0) \text{ and } (H * d).$$

I (Matriz de Incidencia), rango $m \times n$.

d (Vector de Disparos), rango $n \times 1$.

M_0 (Vector de Marcado Inicial), rango $m \times 1$.

MayorØ es una función de $R^m \rightarrow R^m$

$$\text{MayorØ}_i \text{ es } \begin{cases} 1 & \text{si } m_{0i} > 0 \\ 0 & \text{si } m_{0i} = 0 \end{cases}$$

III. SOLUCIÓN GENERAL

A. Evaluacion de trabajos ya desarrollados

Existen diferentes maneras de implementar las redes de Petri: En software y en hardware. Las primeras pueden ser realizadas a través de markup language o desarrollos específicos como los referenciados en la Tabla 2. Las redes de Petri implementadas por hardware son realizadas con módulos que representan las plazas y las transiciones, para construir con estos patrones la red en cuestión.

En este trabajo se presenta la implementación de redes de Petri con peso de arco igual a 1 o mayor, con arcos inhibidores y plazas acotadas, de una forma directa, a partir de las matrices ejecutando el algoritmo de Petri. Lo que permite usar directamente la matriz sin ninguna compilación, pudiendo programarse prioridades.



Tabla 2: Referencia de otros trabajos realizados.

Implementa el algoritmo de disparo de la red de Petri por medio de un bloque en FPGA [13].

Necesita demasiados ciclos para dispararse y lo implementa con bloques, no usa prioridades ni plazas acotadas [14].

Lo modela pero no lo ejecuta, lo codifica solamente [15].

Lo modela pero no lo ejecuta [16].

Lo modela lo ejecuta, parcialmente, debe hacer dos matrices y no maneja las prioridades [17].

Lo programa en software [18].

La usa para modelar sistemas en VHDL, lo construye con bloques [19].

Lo programa hace el algoritmo en software [20].

Lo programa en software [21].

A continuación, se describe la creación de la creación de un IP Core (Procesador de Petri, PP) sobre una arquitectura del procesador Microblaze, que comparta el bus principal PLB (Peripheral Local Bus), como solución a los objetivos planteados. La arquitectura del mismo, como puede observarse en la Figura 4, debe estar formada por los siguientes componentes, conectados al bus:

- Colas de espera: Tanto de entrada como salida.
- Registros: Se describen a continuación los cuatro utilizados.
- Pendientes: para acumular los disparos que fueron recibidos y no estén repetidos; quedando éstos últimos, almacenados en la cola de entrada.
- Matriz de Incidencia y Matriz Inhibidora: para almacenar los valores de éstas, que son propias de los formalismos de Petri; cargadas desde el bus PLB enviadas desde el programa principal.

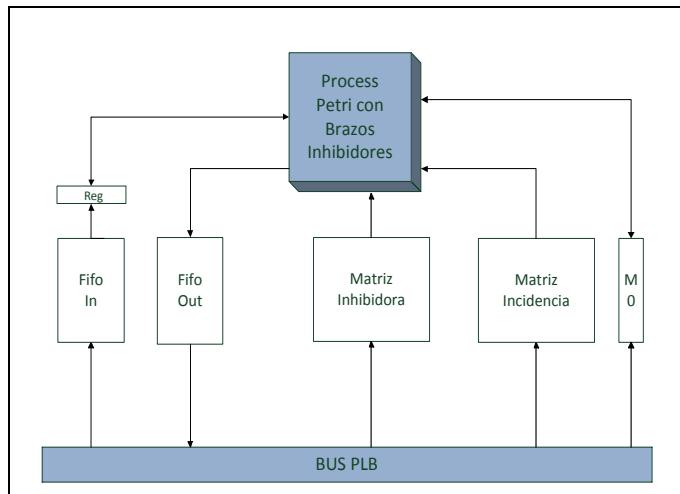


Figura 4: Interconexión de los componentes

- Plaza acotada: se almacena la máxima cantidad de tokens que admite una plaza (es integrado con el registro de estado).
- Estado: donde se almacena el estado inicial de la PN.

IV. ARQUITECTURA Y FUNCIONAMIENTO DEL PP

A. Operación de carga

La carga de datos enviados desde el exterior, se realiza de manera secuencial. El PP cuenta con direcciones asignadas a cada elemento interno para identificarlos, que se pueden acceder exteriormente.

Los elementos relacionados con la carga de datos son: la cola de entrada, el vector de estado inicial y las matrices de incidencia e inhibidora. El formato de los datos es específico a cada elemento que sea direccionado.

- **Disparo:** Es recibido por el módulo desde el PLB y almacenado en la cola de entrada. El formato es de 32 bits de longitud, con sólo un bit puesto a 1, para que se diferencie del resto.

Tiene como función realizar la ejecución de la red de Petri (Programa de Petri). Para esto, los procesos realizan disparos cada vez que quieren hacer uso de un recurso:

Si fue exitoso, decisión tomada por la Lógica de Petri, el programa continúa su curso, caso contrario, espera hasta que se pueda efectivizar el disparo.

- **Cola de Entrada (FIFO In):** Se trata de una FIFO de salida paralela, es decir, todos los disparos únicos, salen en paralelo; quedando almacenados los repetidos, esto justifica la elección de un bit en la palabra de disparos por cada disparo. Cuando este elemento se direcciona, almacena los disparos enviados desde el exterior de manera secuencial. Solamente se mantienen los repetidos, ya que cada uno que no lo sea, es cargado como un bit en el registro Pendientes.

- **Pendientes:** Es un registro de 32 bits, lo cual implica que se almacenan hasta 32 disparos distintos, provenientes de la FIFO In. Es responsabilidad de la Lógica de Petri analizar cuáles son los disparos válidos, como así también, cuál se llevará a cabo; reseteando el bit que representa a éste. A continuación, se comprueba en la FIFO In la presencia de disparos. Esta comprobación sucede también, si ningún disparo fue válido.

- **Lógica de Petri:** Evalúa el estado inicial y las matrices de incidencia e inhibidora, según el algoritmo de Petri. Procesa paralelamente, todos los disparos almacenados en pendientes, y decide por orden de prioridad entre los válidos, cuál es el próximo a ejecutarse. Posteriormente, éste es guardado en la cola de salida; caso contrario; se mantendrá en pendientes.

- Estado Inicial (M_0): Es enviado al PP solamente la primera vez. Es un vector dimensionado por la cantidad de plazas de la PN, donde cada componente tiene un valor de 8 bits. En la inicialización, se usan 7 bits, puesto que sólo se admiten valores positivos y el octavo, es usado como signo en la ejecución. Tiene como propósito establecer el estado inicial del sistema, cuando algún disparo recibido es exitoso, M_0 es remplazado por un nuevo estado calculado.
- Matrices de Incidencia e Inhibidora: Se envían al modulo la primera vez, en ráfagas de 4 valores de 8 bits cada uno; con lo cual, se ahorra tiempo de inicialización. El envío de datos se hace por columnas. Son las responsables de describir, en forma matemática, a la PN planteada; dicho de otro modo representan el programa a ejecutar.

B. Algoritmo de Petri con Brazos Inhibidores y Plazas Acotadas

Para poder ejecutar el programa cargado en el PP, la Lógica de Petri lleva a cabo el siguiente algoritmo:

- Espera de un disparo en FIFO In.
- Toma el disparo de la FIFO In y, si no está ocupada su posición en pendientes, lo marca en pendientes y lo borra de la FIFO In.
- Evaluá en paralelo de pendientes, todos los disparos, según las restricciones planteadas (matriz incidencias, inhibidora y vector de limitación de plaza) y calcula los nuevos estados para cada posible disparo (i), que son almacenados en el vector auxiliar (S^i).
 - Con el vector auxiliar (S^i), se decide cual disparo es posible, o sea, disparos que produzcan nuevos estados sin elementos negativos.
 - Se obtiene un vector auxiliar (F^i) a partir de la matriz inhibidora, el vector de plazas acotadas y el vector de estado, que contiene los disparos posibles, o sea disparos que no estén inhibidos ni acotados y sean posibles.
- Selecciona aquel disparo que sea simultáneamente un disparo pendiente (según punto b) y un disparo posible (según punto c). De haber más de uno, selecciona el de mayor prioridad.
- Si el disparo es resuelto, lo envía a la FIFO Out y lo remueve de pendientes.
- Remplaza el estado M_0 por el estado nuevo:
 - Cada disparo está asociado a una fila dada de la matriz resultado M_R , por lo tanto cuando el disparo (i) se resuelve, se buscará la fila S^i de la matriz asociada a ese disparo y dicha fila S^i será el nuevo estado del sistema.

En la Figura 5 se describe el funcionamiento del procesador para resolver redes de Petri Ordinarias y de Brazos Inhibidoras; se ha obviado, por razones de espacio, el vector

de plazas acatadas puesto que es similar a de brazos inhibidores.

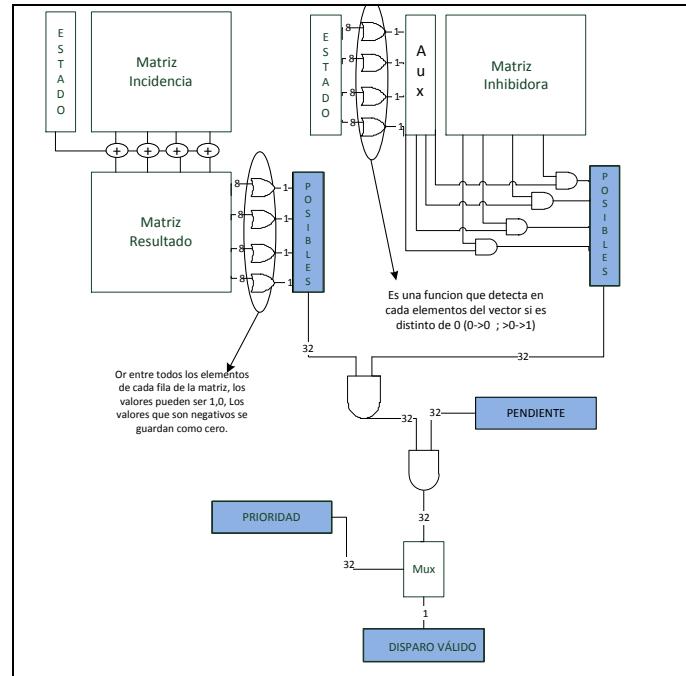


Figura 5: Implementación final detallada

C. Salida de Datos

- Cola de Salida (FIFO Out): Almacena los disparos exitosos, conforme se reciben desde la Lógica de Petri. Cumple la función de buffer, con el fin de amortiguar las esperas necesarias para la comunicación. Los disparos mantienen el orden en la FIFO Out, según han sido resueltos por la Lógica de Petri.

V. PRUEBAS

Se trabajó con la FPGA Spartan 3E-1600, utilizando el framework Xilinx Design Suite versión 12.2, específicamente su herramienta EDK, sobre el Sistema Operativo Windows 7 Ultimate 2009 Microsoft Corporation (versión de 32 bits). Esta herramienta brinda tanto el procesador como el Sistema Operativo utilizados.

El microprocesador con el que se trabajó es el Microblaze 7.3b, con el Sistema Operativo Xilkernel versión 5.00.a.[2]. [5].

A continuación se observan las comparaciones realizadas de las pruebas corridas para el problema escritor – escritor, el que consiste en: los escritores son procesos que compiten por escribir en el mismo recurso (variable compartida), por lo que deben ser sincronizados; esta sincronización ha sido realizada con el PP y con semáforos, respectivamente. Asimismo se han evaluado configuraciones con dos y cuatro hilos, como se muestra en la Tabla 3. En el cuadro anterior se observa que a mayor cantidad de hilos la mejora es aproximadamente del 6%, mostrando de esta manera que a mayor cantidad de hilos el PP tiene mejor performance que la solución con semáforos, lo que muestra la Figura 6.



Tabla 3: Tiempos usando 2 y 4 Hilos

Pruebas	2 hilos	4 hilos
	(ns)	(ns)
Petri	77,19	125,16
Semáforo	143,99	303,54
Mejora	53,60%	58,87%

- Decide si el disparo puede ejecutarse o no en 2 ciclos de reloj.
- Es más simple la programación, puesto que los procesos están desacoplados de la sincronización y exclusión mutua.

VII. BIBLIOGRAFÍA

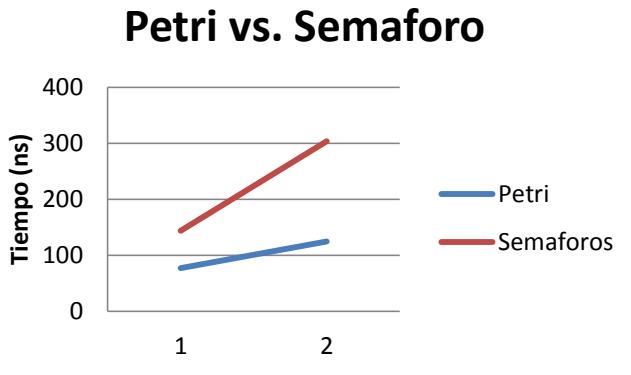


Figura 6: Tiempos utilizando 2 y 4 Hilos

VI. APORTES

En el presente informe, se desarrolla el Procesador de Petri paralelo de funcionalidades ampliadas, que permite desacoplar la sincronización y la exclusión mutua de los procesos ejecutados en los distintos procesadores:

- La inclusión del PP a un sistema multi-core homogéneo, lo convierte en un sistema heterogéneo. El PP se comunica con todos los microprocesadores en forma simétrica, ejecutando un set de instrucciones (assembler) distinto de los demás.
- En los problemas planteados, el PP permite la sincronización de hilos, con mejoras de al menos un 40%.
- Existe una relación directa entre el grafo y el programa de Petri que ejecuta el PP, puesto que éste es programado por la matriz de incidencia, la de brazos inhibidores, el vector de plazas acotadas y el vector de estado.
- Se admiten múltiples disparos simultáneos en la misma transición.
- El PP compara en paralelo, cuáles de todos los disparos son posibles de resolver.
- Permite la programación de prioridades; seleccionando, de los posibles disparos a resolver (en caso de existir más de uno), cuál será efectivamente realizado.

- [1]. P. P. Ch., FPGA Prototyping by VHDL examples, John Wiley 2008.
- [2]. MicroBlaze, <<http://www.xilinx.com/tools/microblaze.htm>>
- [3]. N. G. J. M. a. D. R. Jerry Case, «Design Methodologies for Core-Based FPGA Designs,» www.xilinx.com, 1997.
- [4]. M. Granda, «Redes de Petri,» Departamento de Electrónica y Computadores, Universidad de Cantabria, 2010.
- [5]. F. V. J. S. R. Jesman, «MicroBlaze Tutorial,» Illinois Institute of Technology, 2006.
- [6]. O. Micolini, M. Tejeda, J. Pailler, «Costo Sincronización en un procesador multi-core» U.N.C., FCEFyN, Prunie 2011, Cba, Argentina.
- [7]. F. Baldoni, M. Romano, O. Micolini, «Simulación de procesadores Superescalares para la sincronización de múltiples procesos,» U.N.C., FCEFyN, Prunie 2011, Cordoba Argentina..
- [8]. M. Pereyra, N. Gallia, O. Micolini, «Sistemas embebidos en FPGA,» U.N.C., FCEFyN, Prunie 2011, Cordoba Argentina..
- [9]. P. Pellitero, «Sistemas de multiprocesamiento simétrico sobre FPGA».
- [10]. F. S. P. D. C. - R. C. a. B. Harding, «www.eetimes.com,» 2005.
- [11]. O. Micolini, «Redes de Petri,» LAC-UNC, Cordoba, 2011.
- [12]. XILINX, Inc., Embedded Processing Peripheral IP Cores, http://www.xilinx.com/ise/embedded/edk_ip.htm, 2011.
- [13]. Gary A. Bundell, "An FPGA Implementation of the Petri net Firing Algorithm," Department of Electrical and Electronic Engineering Information Systems Engineering Research Group (citeseerx), The University of Western Australia, 1997.
- [14]. Hideki MURAKOSHI , Miki SUGIYAMA, and Guojun DING, "A HIGH SPEED PROGRAMMABLE CONTROLLER BASED ON PETRI NET," Faculty of Engineering Yokohama National University , Japan , Tokiwadai Hodogaya-ku Yokohama, 1966.
- [15]. FANG Xianwen , XU Zhicai, and YIN Zhixiang, "A Study about the Mapping of Process- Processor based on Petri Nets," Anhui University of Science and Technology, 2006.
- [16]. SERGIO C. BROFFERIO, "A Petri Net Control Unit for High-speed Modular Signal Processors," IEEE TRANSACTIONS ON COMMUNICATIONS, JUNE 1987.
- [17]. Hideki MURAKOSHI MURAKOSHI et al., "MEMORY REDUCTIOON FIRENIT FOR PETRI NET CONTROLLED MULTIPROCESSOR," IEEE, pp. 1961-1966, 1991.
- [18]. Ramón Piedrafita Moreno and José Luis Villarreal Salcedo, "Adaptive Petri Nets Implementation. The Execution Time Controller," in Workshop on Discrete Event Systems Göteborg, Sweden, 2008
- [19]. Wegrzyn Marek, Wolanski Paweł , Adamski Marian, and Joao L. Monteiro, "Coloured Petri Net Model of Application Specific Logic Controller Programs," in ISIE '97, Guimarees, Portugal, 1997
- [20]. Ayba Aydin and Iftar Altug, "Deadlock Avoidance Controller Design for Timed Petri Nets Using Stretching," IEEE SYSTEMS JOURNAL, vol. VOL. 2, no. 2, pp. 178-189, JUNE 2008.
- [21]. MURAKO SHI Hideki et al., "Hardware Architecture for Hierarchical Control of Large Petri Net," IEEE, pp. 115-126, 1993

Simulación de procesadores multicore para sincronizar múltiples procesos utilizando redes de Petri

Micolini, Orlando

Laboratorio de Arquitectura de Computadoras
FCEFyN - UNC
Córdoba, Argentina
omicolini@compuar.com

Romano, Matías

Laboratorio de Arquitectura de Computadoras
FCEFyN - UNC
Córdoba, Argentina
matir87@gmail.com

Baldoni, Federico

Laboratorio de Arquitectura de Computadoras
FCEFyN - UNC
Córdoba, Argentina
febaldoni@gmail.com

Abstract—Este trabajo posee la particularidad de innovar en el uso de las redes de Petri no solo como modelo de sistemas, sino como lenguaje de ejecución de algoritmos concurrentes. Se propone un mecanismo para mejorar la sincronización entre procesos (hilos) a nivel de hardware en procesadores multicore, haciendo uso del formalismo de las Redes de Petri (procesador de Petri). Mediante el empleo de SESC, un simulador de multiprocesadores, se implementa el mecanismo planteado logrando así desarrollar una propuesta de una nueva arquitectura de procesadores multicore.

El módulo propuesto permite obtener mejoras tanto en tiempo de ejecución como en cantidad de instrucciones ejecutadas para algoritmos concurrentes que requieren ser sincronizados, cuanto mayor es el uso de primitivas de sincronización en un algoritmo mayor es el beneficio de esta propuesta. Las mejoras en tiempos de ejecución e instrucciones ejecutadas por el procesador simulado que incorpora el módulo de Petri tienen una disminución de entre un 25% y 50% respecto del procesador original (openSparcT1).

Index Terms— Procesador multicore, sincronización, Petri nets.

I. INTRODUCCIÓN

Los problemas de exclusión mutua y sincronización, en procesos que comparten recursos, producen una sobrecarga considerable en los tiempos de ejecución [1]. Existen diversos métodos para implementar la sincronización al acceso de recursos compartidos. Los más conocidos y desarrollados son pipes, semáforos y monitores [2]. Estas técnicas son implementadas por software, mediante la intervención del sistema operativo para hacer la gestión de recursos compartidos, siendo estos componentes los responsables del incremento de los tiempos de ejecución [2].

El modelo utilizado en este trabajo para la resolución de los problemas de concurrencia es el formalismo de Redes de Petri [3], siendo este el utilizado para implementar las primitivas de sincronización [2].

Este trabajo muestra el modelo de una arquitectura de multiprocesadores, haciendo uso del procesador de Petri, que es verificada con un simulador de procesadores multicore

(SESC [4]) en procesos que utilizan primitivas de sincronización y exclusión mutua. Esta nueva arquitectura permite, en este tipo de problemas, disminuir los tiempos de ejecución y la cantidad de instrucciones ejecutadas.

Obteniendo con el módulo planteado una arquitectura de procesadores con mejor performance en programas que requieren de sincronización y exclusión mutua.

Este trabajo está dividido en cuatro secciones. En la primera se exponen brevemente los mecanismos de sincronización, conceptos básicos del modelo de Redes de Petri (PN) y se concluye con la aplicación de PN a problemas de concurrencia. En la segunda sección se selecciona y describe la arquitectura propuesta en hardware del módulo de sincronización, y la implementación sobre el simulador. La tercera sección contiene las mediciones realizadas sobre SESC para mostrar los alcances de la arquitectura desarrollada. En la última sección se realiza la conclusión del trabajo y propuestas de trabajos futuros.

II. CONCEPTOS

A. Teoría de Sincronización

En esta parte se definen los conceptos de dos tipos de sincronizaciones entre procesos concurrentes [5], exclusión mutua y condición de sincronización.

1) Exclusión mutua

Cuando los procesos desean utilizar un recurso no compartible, la sincronización necesaria entre ellos se denomina exclusión mutua. Un proceso que está accediendo a un recurso no compartible se dice que se encuentra en su sección crítica. Solo un proceso puede estar en su sección crítica, el resto de los procesos que quieren ingresar a esta en ese momento deben permanecer en espera hasta que se libere.

2) Condición de sincronización

La condición de sincronización hace referencia a la situación en la cual es necesario coordinar la ejecución de procesos concurrentes, esta se lleva a cabo cuando un objeto de datos compartido está en un estado inapropiado para ejecu-

tar una operación determinada. Cualquier proceso que intente realizar esta operación sobre el objeto, debe ser retrasado hasta que el estado del objeto cambie por las operaciones de otros procesos. Por lo tanto se define la condición de sincronización como la necesidad para que un proceso no realice una acción hasta que otro proceso haya finalizado otra acción determinada. [6]

B. Mecanismos de sincronización a sustituir con PN

A continuación se listan los distintos y más importantes mecanismos [2] existentes para resolver e implementar la sincronización [7] necesaria entre procesos concurrentes, estos son:

- Inhibición de las interrupciones (solo monoprocesadores)
- Espera activa
- Semáforos
- Regiones críticas
- Monitores
- Operaciones de paso de mensaje síncronos

Se hace foco solo en los mecanismos de espera activa, semáforos, regiones críticas y monitores los cuales pertenecen al grupo de soluciones basadas en variables compartidas, el resto de los mecanismos pertenecen al grupo de soluciones basadas en paso de mensajes. Este último grupo no es de nuestro interés ya que posee su principal aplicación en sistemas distribuidos que exceden este trabajo.

C. Redes de Petri

Una Red de Petri o Petri Net (PN) es un modelo gráfico, formal y abstracto para describir, modelar y analizar el flujo de información. Conforma una herramienta matemática que puede aplicarse especialmente a los sistemas paralelos que requieren simulación y modelado de la concurrencia en los recursos compartidos. Las PN están asociadas con la teoría de grafos y se pueden considerar como autómatas formales y generadores de lenguajes formales. [3]

De los conceptos vertidos en Petri Nets: Fundamental Models, Verification and Applications, las PN y sus extensiones son de un interés fundamental innegable. Teniendo en cuenta que las redes de Petri además de proveer de un mecanismo formal para modelar procesos concurrentes son en sí mismas un lenguaje formal [3], se las utiliza para modelar y probar sistemas.

En este trabajo se utilizarán las PN para modelar y ejecutar los problemas de exclusión mutua y sincronización [8], debido a que incluyen y extienden a las primitivas de sincronización mencionadas anteriormente (ej. Semáforos).

D. Sincronización y Redes de Petri

Las redes de Petri permiten describir de forma fácil y precisa sistemas en los que se producen dependencias de datos, exclusión mutua y sincronización entre procesos. Como así también el estado de procesos paralelos.

1) Sincronización de tareas

El problema productor/consumidor [6] es uno de los ejemplos clásicos de acceso a recursos compartidos que debe arbitrarse mediante algún mecanismo de concurrencia que implemente la exclusión mutua.

Este problema involucra elementos de datos compartidos por ambos procesos y, en consecuencia, es necesario plantear mecanismos de sincronización entre ambos. En la Figura

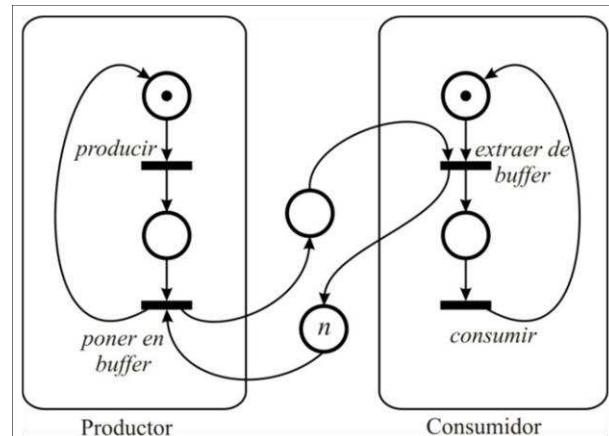


Figura 1. PN de Productor/Consumidor con buffer limitado a n unidades.

ra 1, se muestra la red de Petri de un productor/consumidor con buffer limitado a n unidades.

2) Exclusión mutua entre secciones críticas

Cuando varios procesos se ejecutan en paralelo y comparten información, es necesario garantizar que los accesos a la información común no se lleven a cabo simultáneamente, para evitar problemas de interferencia que conduzcan a resultados erróneos.

Una solución a este problema es hacer mutuamente excluyentes las secciones críticas de los diferentes procesos encargados de acceder a la información. En la Figura 2, se muestra una PN que resuelve este problema, garantizando que los procesos acceden de a uno por vez a la sección crítica. [3]

Las redes de Petri pueden expresar también monitores como se puede ver en Implementación de un Monitor con una Red de Petri Ejecutable. [8]

III. DESARROLLO

El objetivo propuesto para gestionar los recursos y esfuerzos que den cumplimiento al desarrollo del trabajo es simular un módulo de sincronización por hardware que permita mejorar los tiempos de sincronización entre procesos que sean ejecutados en procesadores multicore, múltiples hilos, empleando primitivas de sincronización por PN y además debe ser integrable al simulador SESC, cumpliendo con las restricciones al presente trabajo, estas son:

- No cambiar la Arquitectura del Set de Instrucciones (ISA) del procesador.

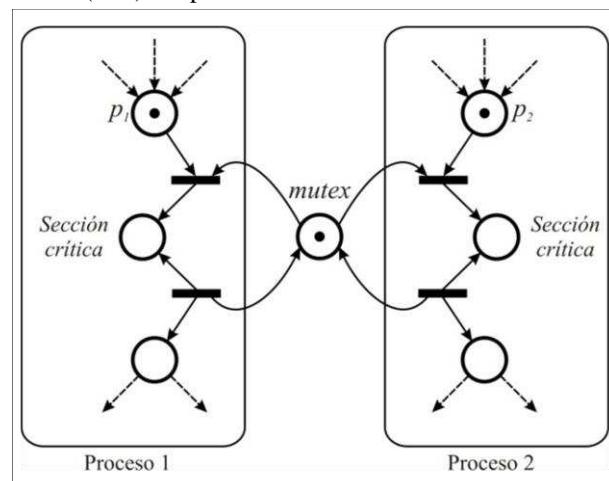


Figura 2. PN de una exclusión mutua.

- No requerir operaciones específicas del Sistema Operativo (SO).
- Los cambios a nivel de programa deben ser mínimos.
- Debe permitir reemplazar fácilmente las primitivas de sincronización del SO.
- Los cambios a nivel de hardware, en lo posible, deben ser implementados con módulos existentes (ya probados, a los cuales solo se le debe modificar la lógica interna) para permitir así su instalación en distintos procesadores.

Las posibilidades que fueron contempladas para llevar a cabo la implementación del módulo de hardware capaz de realizar la sincronización de procesos concurrentes son:

- Módulo incorporado al procesador controlado por nuevas instrucciones.
- Módulo independiente actuando como co-procesador para gestionar un sistema de colas. [9]
- Módulo en IPCore conectado al local bus como memoria compartida.
- Módulo a nivel de RAM comunicado a través de posiciones de memoria compartida.
- Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida.

Para la evaluación de estas posibilidades se realizó la Tabla 1 con el objeto de seleccionar el mecanismo que sea compatible con las restricciones de funcionamiento contempladas anteriormente.

Consideramos que el mecanismo más adecuado para la implementación es el módulo a nivel de cache (L1) comunicado a través de posiciones de memoria compartida, puesto que mantiene la comunicación entre los procesadores multicore a la mayor velocidad posible, utiliza bloques funcionales existentes manteniendo la arquitectura y no hace uso del sistema operativo, ni implementa nuevas instrucciones. Esto podrá ser realizado siempre y cuando el procesador cuente con un nivel de memoria cache compartida. De no ser así, el mecanismo adecuado sería un IPCore conectado al local bus, mapeado como direcciones de memoria compartida.

Luego de seleccionar el mecanismo encargado de llevar a cabo la sincronización, se está en condiciones de detallar el algoritmo a implementar en el módulo.

A. Algoritmo de Petri

Este algoritmo permite la ejecución de la Red de Petri, es decir la evolución de los estados a través de los disparos de las transiciones.

Restricción: Los disparos se ejecutan de a uno por vez, o sea el vector de disparo contiene solamente un elemento igual a uno y el resto son ceros. Y una transición puede tener solo un disparo pendiente de ejecución. Estas restricciones no implican una disminución en el poder de expresividad de las redes. [3]

A partir de la expresión:

$$M_{\varphi+1} = M_{\varphi} + I * \sigma \quad (1)$$

donde I es la Matriz de Incidencia, M_{φ} el vector de marcado después del disparo φ , σ el vector disparo. Siendo la matriz I formada por las columnas I^1, I^2, \dots, I^v con las cuales se obtienen los vectores T^i efectuando las operaciones que indica la siguiente ecuación:

$$T^i = M_0 + I^i \quad (2)$$

A partir de cada T^i definimos el escalar E_i , según la siguiente condición: Si T^i contiene algún elemento menor a cero es $E_i = 0$ de lo contrario es $E_i = 1$.

Los cálculos de todas las variables E_i se obtienen de forma paralela según la cantidad de columnas I^1, I^2, \dots, I^v

Siendo el vector de disparo σ_i , el cual tiene el elemento i -ésimo igual a uno y el resto cero.

Los pasos del algoritmo al recibir un disparo son:

1. Se evalúa que el disparo cumpla con las restricciones planteadas. Si es correcto se pasa al punto 2. De lo contrario el disparo es eliminado.
2. Se evalúa una transición en la red caracterizada por el vector de disparo σ_i , cuyo i -ésimo elemento se utiliza para seleccionar E_i , con el fin de ejecutar o no la transición dependiendo de si E_i vale uno o cero respectivamente.
 - a. Si la transición no puede realizarse debido a $E_i = 0$ el disparo es puesto en una cola de disparos pendientes y se pasa al punto 4.
 - b. Si la transición puede ejecutarse correctamente se remplaza M_{φ} por T^i , se calculan todos los T^i .
3. Luego se controla toda la cola de disparos pendientes.

	Módulo incorporado al procesador controlado por nuevas instrucciones	Módulo independiente actuando como co-procesador para gestionar un sistema de colas	Módulo en IPCore conectado al local bus como memoria compartida	Módulo a nivel de RAM comunicado a través de posiciones de memoria compartida	Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida
No modificar ISA	No cumple	No cumple	Si cumple	Si cumple	Si cumple
No requerir operaciones del SO	No cumple	Si cumple	Si cumple	Si cumple	Si cumple
Cambios a nivel de programa mínimos	No cumple	Si cumple	Si cumple	Si cumple	Si cumple
Cambios en hardware con módulos testeados	No cumple	No cumple	Si cumple	Si cumple	Si cumple
Viabilidad en SESC	Si cumple	No cumple	No cumple	Si cumple	Si cumple
Velocidad de ejecución	Muy alta	Alta	Media	Muy baja	Alta

Tabla 1. Comparación de las alternativas de implementación

tes para verificar si es posible el disparo de alguno de estos, ejecutándose en caso de ser posible. Si se produce un disparo exitoso se repite el paso 3 hasta que no exista ningún disparo posible.

4. Se queda a la espera de un nuevo disparo.

El objetivo que persigue este algoritmo es el de tener todos los escalares (E_i) calculados a la hora de evaluar el disparo, para obtener el menor tiempo de respuesta, ya que está pensado para ser implementado en hardware.

B. Implementación y ejecución del algoritmo de Petri en SESC

Se crea un archivo el cual contiene la matriz de incidencia I , el vector de marcado inicial M_0 , la cantidad de vectores disparo y cada uno de los disparos (σ). Este archivo se utiliza como inicialización del programa que se desea ejecutar (Escritor/Escritor, Productor/Consumidor, etc.). Cada programa a ejecutar posee su propio archivo de inicialización. Cuando el programa necesita acceder a alguna variable en exclusión mutua (se debe sincronizar su uso), se efectúa el disparo sobre una variable (posición de memoria asociada al disparo), este “disparo” (petición de escritura de un dato) indica a que variable sincronizada se está queriendo acceder. “Dicho disparo es reconocido por el algoritmo de Petri que ejecuta la cache, actuando como controlador de cache” y se encarga de efectuar las operaciones pertinentes para obtener el nuevo estado de la red (nueva marcación). Según el resultado de estas operaciones matemáticas, se determina si es posible acceder o no a la variable deseada (sincronización). Debe aclararse que el mecanismo de la memoria cache de Petri, solo almacena las variables utilizadas para llevar a cabo la sincronización, el dato que se desea modificar (variable en exclusión mutua) se encuentra en L1 y/o L2 y/o en RAM. En el caso que no se pueda acceder a la variable a sincronizar, dicho disparo es almacenado en una cola de disparos pendientes, la cual es controlada cada vez que se efectúa un disparo de forma exitosa.

Si el disparo es exitoso la cache devuelve un Hit al procesador y la ejecución del proceso continua. En cambio, si el disparo no es posible se retorna un Miss, deteniendo la ejecución del proceso hasta que el disparo sea posible, lo cual es comunicado con un Hit.

El mecanismo aquí expuesto permite realizar la sincronización en los tiempos requeridos para una escritura en cache y una comparación en un registro de cache, lo que implica el menor tiempo posible. Y además, cuando una sincronización pendiente es resuelta el tiempo de comunicación es el del envío del Hit.

En la Figura 3 se muestra la arquitectura propuesta, donde el módulo está implementado como una cache compartida, con una política de Petri.

IV. MEDICIONES

Los algoritmos de medición simulados se han seleccionado por su elevada sincronización entre procesos, y simulada su ejecución para medir los tiempos consumidos.

A. Escritor / Escritor

Este es el caso en que dos procesos tratan de escribir en el mismo buffer, accediendo de forma alternada.

Un par de hilos realizan las escrituras sobre un array compartido de datos, la cantidad de escrituras a realizar por cada hilo en la sección crítica no van a ser fijas, pero si será

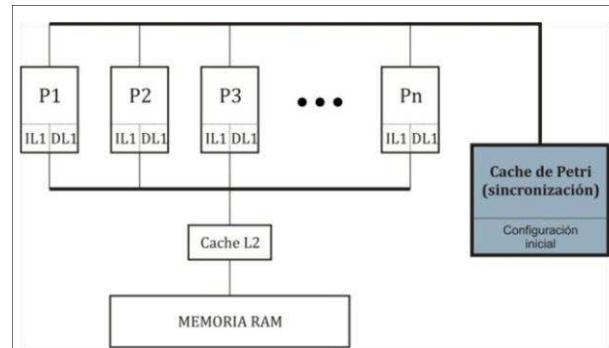


Figura 3. Componente implementado.

fijo el número total de escrituras de cada hilos sobre el array. La Figura 4, representa el modelo del Escritor/ Escritor por una PN.

Las gráficas, Figura 5 y Figura 6, muestran los resultados obtenidos en las mediciones del algoritmo comparando el simulador con la arquitectura estándar respecto del que posee el módulo de sincronización de Petri.

La Figura 5 muestra las mediciones de los tiempos empleados para ejecutar el código respecto a la cantidad de escrituras realizadas por sincronización. Claramente puede verse que el tiempo de ejecución disminuye notablemente a medida que disminuye la cantidad de sincronizaciones realizadas. Destacándose el hecho que existe una disminución de hasta un 25% de los tiempos en la ejecución del código sobre el simulador modificado respecto del original.

La Figura 6 muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simulador modificado respecto del original. Se aprecia con claridad la disminución en las instrucciones ejecutadas en la arquitectura propuesta respecto de la original, dicha mejora se encuentra más pronunciada para una cantidad de escrituras por sincronización igual a 1 y 2, esto es debido al alto impacto producido por la ejecución de una primitiva de sincronización en cada ciclo o par de ciclos de escritura.

En ambas figuras es claro como tienden a igualarse los valores en los tiempos de ejecución e instrucciones ejecutadas por ambos procesadores para cantidades de escrituras por sincronización superiores a 64. Este fenómeno se debe a que los tiempos o instrucciones empleados para ejecutar la primitiva de sincronización no representan más de un 1% del tiempo o instrucciones totales.

B. Productor / Consumidor

El problema productor/consumidor [6] es uno de los ejemplos clásicos de acceso a recursos compartidos que debe arbitrarse mediante algún mecanismo de concurrencia que

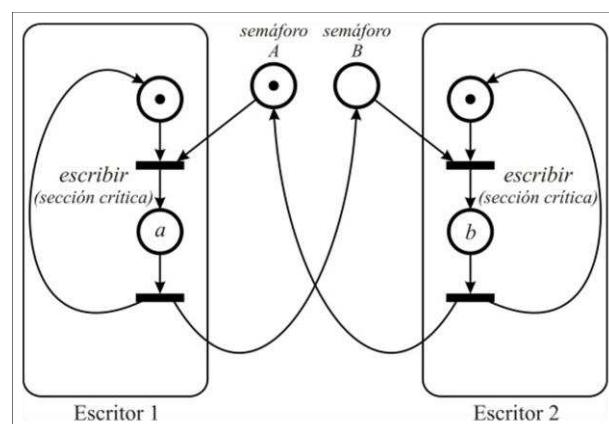


Figura 4. PN de escritura de variable compartida.

Gráficas de los resultados arrojados en la simulación del algoritmo Escritor/ Escritor.

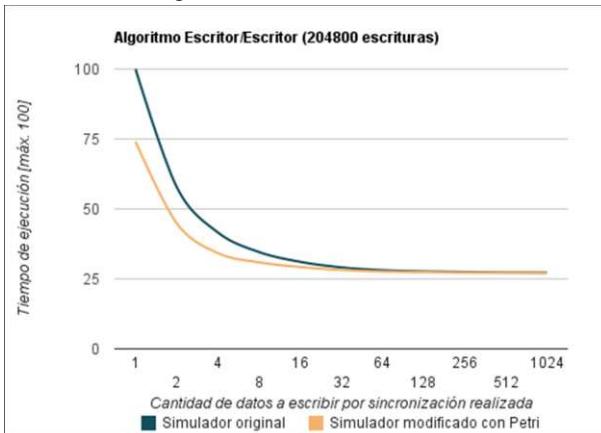


Figura 5. Comparativa entre simulador original y modificado respecto del tiempo de ejecución

Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.

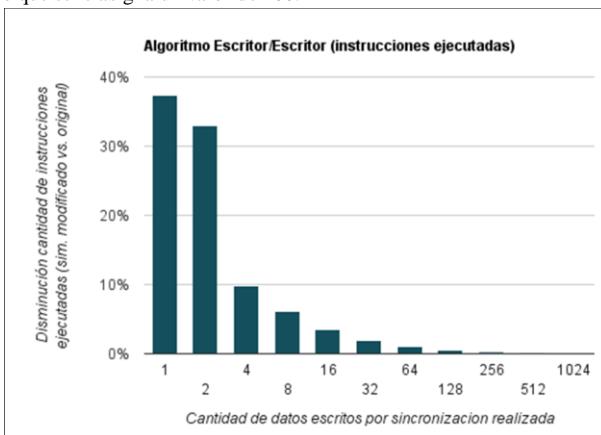


Figura 6. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas

implemente la exclusión mutua. En este caso el proceso productor genera información que es colocada en un buffer de tamaño limitado y es extraída de este por un proceso consumidor. Si el buffer se encuentra lleno, el productor no puede colocar el dato, hasta que el consumidor retire elementos. Si el buffer se encuentra vacío, el consumidor no podrá retirar elementos hasta que el productor deposite en él.

A modo de aclaración de la PN expuesta en la Figura 7, definimos a T_{00} , T_{01} , T_{10} , T_{11} como las transiciones que

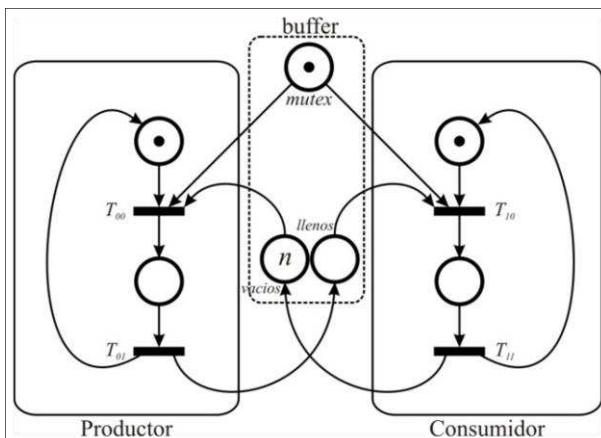


Figura 7. Red de Petri del modelo Productor/Consumidor

forman el modelo del productor/consumidor. El buffer esta inicializado con n elementos, la plaza denominada vacíos indica los lugares vacíos disponibles al proceso productor y la plaza llenos le indica al consumidor la cantidad de elementos que le restan por consumir.

En la Figura 8 se grafica las mediciones de los tiempos empleados para llevar a cabo la ejecución del código, con respecto a la cantidad de producciones y consumiciones realizadas por sincronización. Claramente puede verse que los tiempos de sincronización disminuyen a medida que disminuyen la cantidad de sincronizaciones realizadas. Es importante mencionar que **existe una disminución de más del 50% de los tiempos empleados en ejecutar el código sobre el simulador modificado y un 25% menos de instrucciones ejecutadas**.

En ambas curvas de la Figura 8 se tienen valores similares para una cantidad de producciones/consumiciones por sincronización igual a 15000 y 10000, esto se debe a que los tiempos empleados para esas cantidades tienden a asemejarse a los valores de tiempo obtenidos sin emplear sincronización, debido a que el tiempo empleado en sincronizar pasa a ser mínimo por que reducen en 50 veces la cantidad de sincronizaciones realizadas respecto a la máxima sincronización.

En la Figura 9 se muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simu-

Gráficas de los resultados arrojados en la simulación del algoritmo Productor/Consumidor.

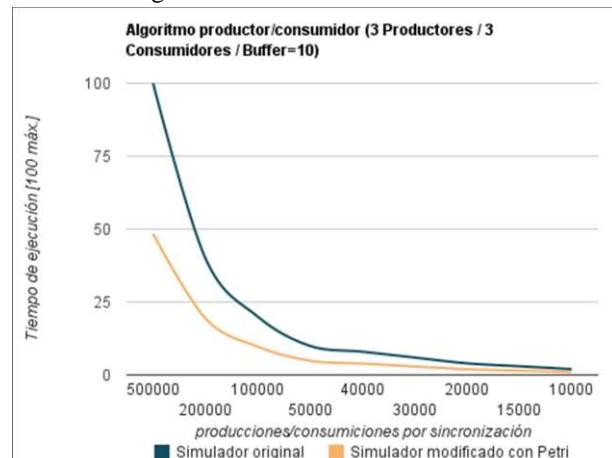


Figura 8. Comparativa entre simulador original y modificado respecto del tiempo de ejecución

Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.

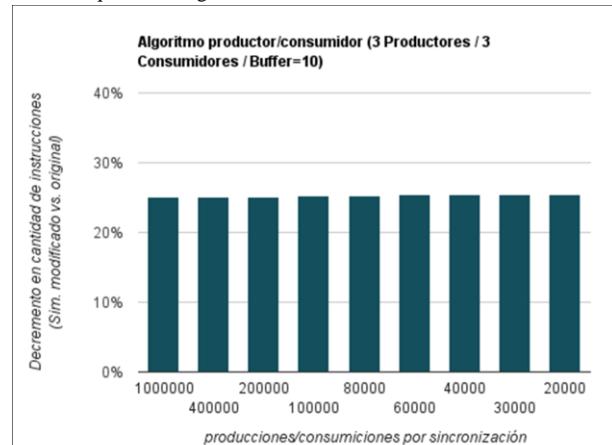


Figura 9. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas



lador modificado respecto del original. En esta figura se aprecia la disminución en la cantidad de instrucciones empleadas en el simulador modificado, siendo este decreto de un 25%.

Se pueden consultar las tablas con los valores obtenidos, como también otros algoritmos medidos en el trabajo final Simulación de procesadores multicore para la sincronización de múltiples procesos [10].

V. CONCLUSIÓN

En este trabajo se utilizó a las Redes de Petri no solo como un modelo para describir sistemas concurrentes, sino además se las implementó dentro de un componente para ser ejecutadas por hardware, con el fin de mejorar el desempeño de los procesadores multicore actuales (openSparcT1).

La solución encontrada fue implementada en el simulador y su empleo permitió obtener una disminución por encima del 25% (alcanzando valores del 75%) en los tiempos de ejecución de procesos fuertemente sincronizados, reduciendo también el número de instrucciones ejecutadas en más de un 20% respecto de los procesadores estándares simulados. Es importante destacar que las mejoras más significativas se dan en programas que empleen fuerte sincronización entre procesos, remarcando el hecho que el nuevo procesador es más rápido ya que tiene mejor respuesta en tiempo real a la hora de sincronizar.

Por otro lado, es importante mencionar que la realización de los programas para que puedan ser ejecutados en el nuevo procesador son de fácil programación, destacando el hecho que una vez realizado el grafo de la red de Petri se pueden obtener, de manera muy sencilla, las matrices mediante la utilización de cualquier simulador de PN la cual se carga como el programa que controla al algoritmo de Petri contenido en la cache.

Hay que destacar que si se tratara de múltiples escritores o múltiples productores/consumidores la sobrecarga impacta en los recursos de memoria y procesador pero no sobre el algoritmo de Petri puesto que este solo atiende a los requerimientos de sincronización y exclusión mutua, es decir que la sobrecarga sobre el módulo está dada por la cantidad de disparos por unidad de tiempo.

Trabajos futuros

Desarrollo de un IPCore que implemente el algoritmo de Petri. En la actualidad se está llevando a cabo en el LAC de la FCEFyN-UNC.

Realizar la extensión del algoritmo a Redes de Petri con brazos inhibidores y de reset.

La extensión del algoritmo citado a Redes de Petri temporales y coloreadas.

La simulación, implementación y experimentación de estas propuestas para comprobar las mejoras en el desempeño de la ejecución de algoritmos fuertemente sincronizados en sistemas multicore.

VI. BIBLIOGRAFÍA

- [1] Maurice Herlihy and Nir Shavit, The Art of Multiprocessor Programming.: Elsevier, 2008.
- [2] Mauerer Wolfgang, Professional Linux Kernel Architecture. Indiana, USA: Wiley Publishing, Inc.,

2008.

- [3] Michel Diaz, Petri Nets: Fundamental Models, Verification and Applications.: Wiley, 2009.
- [4] Pablo Montesinos Ortego and Paul Sack. SESC: SuperEScalar Simulator. [Online]. <http://iacoma.cs.uiuc.edu/~paulsack/secdoc/>
- [5] Clay Breshears, The Art of Concurrency. USA: O'Reilly Media, Inc., 2009.
- [6] José Tomás Palma Méndez and María del Carmen Garrido Carrera, Programación Concurrente. Madrid, España: Thomson, 2003.
- [7] Hesham El-Rewini and Mostafa Abd-El-Barr, ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING. New Jersey, USA: Wiley, 2005.
- [8] Orlando Micolini, Miguel Tejeda, and Hugo Pailer, "Implementación de un Monitor con una Red de Petri," 2011.
- [9] Julio Pailer and Miguel Tejeda, "Implementación en hardware de un procesador de Petri para resolver problemas de sincronización y exclusión mutua en sistemas multicore," 2011.
- [10] Matias Romano and Federico Baldoni, Simulación de procesadores multicore para la sincronización de múltiples procesos. Cordoba, Argentina: FCEFyN - UNC, 2012.
- [11] Julio Pailer, "Medición del costo de sincronización, con carga de variables de cache y tiempos de ejecución de procesos concurrentes en un procesador multicore," Córdoba, 2010.
- [12] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, Introduction to Parallel Computing, Second Edition ed., Addison Wesley, Ed. England: Pearson Education, 2003.



Implementación de un procesador multi-núcleo basado en el procesador Plasma

Federico Giordano Zacchigna, Ariel Lutenberg
Laboratorio de Sistemas Embebidos
Facultad de Ingeniería, Universidad de Buenos Aires
Paseo Colón 850,
Ciudad Autónoma de Buenos Aires
federico.zacchigna@gmail.com, lse@fi.uba.ar

Fabian Vargas
Departamento de Ingeniería Eléctrica
Universidad Católica - PUCRS
Av. Ipiranga, 6681.
90619-900 Porto Alegre – Brazil
Email: vargas@pucrs.br

Abstract—En este trabajo se presenta una implementación de un procesador multi-núcleo. El mismo esta basado en el procesador Plasma. La implementación es de forma tal que se puedan instanciar un número genérico de núcleos durante el proceso de síntesis. Uno de los objetivos principales de esta implementación, es lograr que la misma sea del menor tamaño posible. En este trabajo se detallan los principales factores que influyen sobre el diseño de la arquitectura. También se muestran las principales complicaciones que aparecen y las soluciones a las mismas. Se realizan pruebas del mismo en un kit Nexys2 de Digilent, basado en una FPGA Xilinx Spartan3E-1200. Finalmente se presentan los resultados obtenidos.

I. INTRODUCCIÓN

A. Procesador Plasma y motivación del presente trabajo

El procesador Plasma es un procesador de código libre [1], es simple y ha sido utilizado para la realización de varios proyectos y pruebas. Este procesador esta basado en la arquitectura MIPS [2]. Ha sido objeto de estudio en varios trabajos, entre los que se cuentan trabajos realizados por integrantes de nuestro grupo en dónde se estudia su funcionamiento bajo efectos de radiación e interferencia electromagnética [3] [4]. En la actualidad se desea continuar con esta línea de investigación sobre procesadores multi-núcleos. De aquí surge la necesidad de implementar este procesador. Las características antes nombradas sobre el procesador Plasma, son ideales para la realización de este trabajo, y es por eso que se elige el Plasma.

B. Procesadores multi-núcleo

Los procesadores multi-núcleo surgen de la necesidad de aumentar la capacidad de procesamiento, las limitaciones del paralelismo a nivel de instrucción de los programas y el gran aumento en la parallelización a nivel de tarea que se encuentra hoy en dia en todo sistema computacional.

El paralelismo a nivel de tareas es fácilmente implementable en procesadores mono-núcleo, pero esto implica un desaprovechamiento del potencial de procesamiento paralelo. Al querer implementar esto mismo en procesadores SMP (Symmetric Multiprocessing) aparecen nuevos aspectos a tener en cuenta, para lograr que la ejecución se realice en forma correcta.

La forma de implementar un multiprocesador no es única. Existen variros tipos de multiprocesadores y a la vez varias formas de implemetarlos. Cada una de ellas tiene sus ventajas y desventajas. En el trabajo se detallan cuales son estas opciones y en particular se presenta la implementación realizada en el Plasma Multicore.

II. DETALLES DEL DISEÑO, PROBLEMATICAS Y POSIBLES SOLUCIONES

A. Clasificación de los procesadores

A lo largo de la historia han surgido varias clasificaciones para las computadoras. Muchas de ellas quedaron rápidamente obsoletas con el paso del tiempo. Sin embargo, la clasificación propuesta por Flynn en el año 1966 [5] sigue siendo válida. La misma se basa en un modelo de dos dimensiones: Datos e Instrucciones. Cada una de estas dimensiones tiene dos posibles valores: Simple o Múltiple. Se obtienen así cuatro combinaciones posibles:

- SISD: En inglés *Single Instruction Single Data*. Es el caso de un procesador simple, como el plasma.
- SIMD: En inglés *Single Instruction Multiple Data*. Es el caso de las llamadas computadoras vectoriales.
- MISD: En inglés *Multiple Instruction Single Data*. Que es de uso poco frecuente.
- MIMD: En inglés *Multiple Instruction Multiple Data*. Es el caso de los multiprocesadores que se tratan en este trabajo.

Para aprovechar las ventajas de las computadoras MIMD es necesario que existan al menos la misma cantidad de tareas que de procesadores. De esto se desprende que el surgimiento de los mismos se basa en el aumento de la parelización de los programas en distintas tareas. En secciones posteriores de este trabajo se tratan consecuencias y problemáticas de esta paralelización.

B. La comunicación entre los procesadores

Al tener en cuenta la comunicación entre los procesadores, surgen dos alternativas principales [6] [7]:

- Procesadores comunicados a través de un bus de datos.

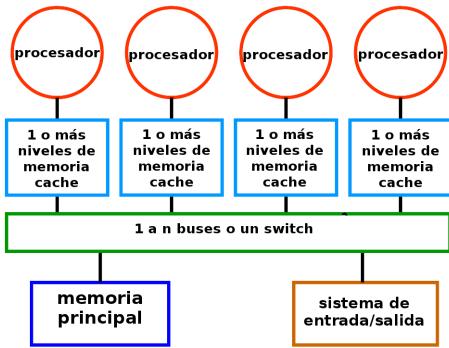


Fig. 1. **Arquitectura Shared Memory** - Los procesadores tienen todos el mismo tiempo de acceso a cualquier posición de memoria, UMA (*Uniform Memory Access*). La comunicación puede ser a través de uno o varios buses, o también a través de un switch.

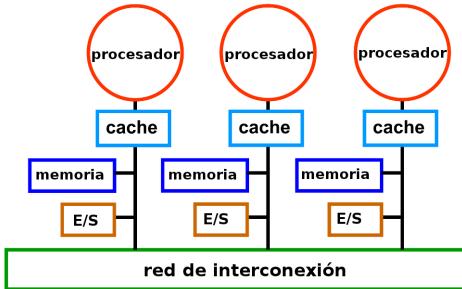


Fig. 2. **Arquitectura Distributed Shared Memory** - Los procesadores tienen distintos tiempos de acceso a distintas zonas de memoria, dependiendo si deben acceder a través del bus o no, NUMA (*Non Uniform Memory Access*). La comunicación es normalmente a través de una red de comunicación.

- Procesadores comunicados a través de una red de comunicación.

Para el plasma multi-núcleo se elige la comunicación a través de un bus por ser mas simple y rápida que a través de una red. El bus puede a su vez estar formado por uno o varios buses de comunicación o bien por un switch [6] [7]. Para este trabajo nuevamente se elige la opción más simple que es un solo bus. Luego se verán algunas ventajas que tiene esta decisión, aparte de su simplicidad, por sobre las otras, sobre todo al hablar sobre las memorias cache y los algoritmos de coherencia utilizados.

Para la memoria compartida existen dos posibilidades:

- UMA: En inglés *Uniform Memory Access*, y que se relaciona con la memoria compartida (en inglés *Shared Memory*), ver figura 1.
- NUMA: En inglés *Non Uniform Memory Access*, y que se relaciona con DSM (en inglés *Distributed Shares Memory*), ver figura 2.

El sistema operativo debe de tener en cuenta el tipo de distribución que se tiene de la memoria, si se desea tener un buen rendimiento. El PlasmaOS [1] (sistema operativo diseñado para el procesador Plasma), no cuenta con estas características, así que se decide por implementar una arquitectura de memoria tipo UMA. Lo cual a su vez es otra ventaja, ya que este tipo de implementación es más simple.

C. La Arquitectura y Jerarquía de la Memoria

Como se dijo en la sección anterior, se utiliza una arquitectura UMA, es necesaria una memoria principal, que sea accesible por todos los procesadores a través del bus de interconexión.

Se implementa también una pequeña memoria interna en cada uno de los procesadores. Esta memoria interna brinda una forma fácil para que cada uno de los procesadores ejecute inicialmente programas distintos. Esto es de gran utilidad en varios casos. Por ejemplo puede ser utilizada como modo de booteo, antes de comenzar a correr un sistema operativo, entre otras cosas. En nuestro caso se utiliza para correr un programa en un solo núcleo que se encarga de cargar el sistema operativo en memoria principal, y también es la sección de memoria que se utiliza como *stack* temporal, antes de que el sistema operativo comience a correr las distintas tareas y procesos. Puede ser quitada fácilmente si se desea.

También se implementa una memoria cache. La memoria cache brinda una forma de mejorar la performance del procesador al aprovechar el principio de localidad [6] [7]. Vale destacar que tanto en el plasma original como en el plasma multi-núcleo, sólo se hace uso del principio de localidad temporal, y no del de localidad espacial, ya que el tamaño de bloque de la memoria cache es de una palabra. En el plasma original, esta memoria cache es opcional. En el plasma multi-núcleo deja de ser opcional, ya que carece de sentido tener más de un núcleo queriendo acceder a memoria principal sin tener memoria cache. Sin memoria cache habría colisiones todo el tiempo. La cache implementada se comparte para instrucciones y datos.

Las políticas para actualizar datos en memoria principal pueden ser:

- Write-Back: Las instrucciones de escrituras en memoria escriben en cache.
- Write-Through: Cada instrucción de escritura en memoria escribe directamente en memoria principal.

Las políticas para actualizar datos en memoria cache pueden ser:

- Write-Allocate, en un miss de escritura, se escribe la cache.
- Write-No-Allocate, en un miss solo se escribe en memoria principal.

En el plasma multi-núcleo se implementa una cache *Write-Through Write-Allocate*, que son políticas poco convencionales en procesadores, pero por el tamaño del proyecto y su complejidad se vuelven convenientes. Una cache *Write-Through* es más simple de implementar que una *Write-Back*, esa es la principal razon de su elección.

D. Árbitro del bus

En el plasma multi-núcleo se producen colisiones al querer acceder a memoria. Esto sucede cada vez que dos o mas núcleos acceden a memoria simultáneamente. Es por esta razón que se debe implementar una entidad encargada de arbitrar el bus, cediendo acceso al bus a un núcleo y bloqueando

al resto, hasta que el que tiene acceso haya culminado con la operación.

Otro requisito buscado en esta entidad, es que arbitre el bus en forma pareja, o sea que en caso en que haya muchas colisiones por muchos accesos a memoria, el acceso sea parejo entre los distintos núcleos. De esta manera ningún núcleo tiene prioridad sobre los demás y por ende los procesos que se estén ejecutando tampoco.

E. Algoritmo de coherencia de cache

En un sistema SMP (Simultaneous Multi-processing), como el Plasma, existen tareas pertenecientes a un mismo proceso que comparten el contexto de memoria. Es necesario entonces implementar algún algoritmo de coherencia de cache.

Existen basicamente dos protocolos para mantener la coherencia en las memorias cache [6] [7]:

- Protocolos basados en un directorio: Existe un directorio que almacena los estados de los bloques de memorias compartidos. Las caches se comunican con el directorio.
- Protocolos *Snooping*: No es centralizado. Cada cache guarda el estado de los bloques que posee. Cada cache escucha a las señales de snoop de otras caches. El snooping requiere un mecanismo de broadcast, que puede ser implementado con un bus simple, no en cambio con un bus switcheado. Protocolos tipo: MSI, MOSI, MOESI [6].

En nuestro caso, al contar con un bus simple que permite *broadcast* y una política *Write-Through*, se vuelve simple y eficiente implementar un protocolo de *Snooping*. En la sección de la implementación se ve este mecanismo de coherencia con mayor detalle.

F. Manejo de operaciones atómicas

Una operación atómica hace referencia a la lectura-modificación-escritura de una posición de memoria. En ningún caso es posible resolverlo en una única instrucción, por lo tanto se implementa mediante un conjunto de instrucciones. Este conjunto de instrucciones debe de ser ejecutado en forma secuencial y sin ser interrumpidas. El efecto buscado con estas instrucciones puede ser afectado si esto no se cumple.

Para asegurarse el correcto funcionamiento se hace uso de la exclusión mutua: se brinda acceso a las tareas de a una a la vez, a las secciones de memoria compartidas. En procesadores de un sólo núcleo la implementación es simple deshabilitando y habilitando las interrupciones. Para procesadores multi-núcleo esto se vuelve mucho más complejo y se hace uso de secciones críticas y *locks* [8].

Ni en el Plasma original ni en el multi-núcleo están implementadas las instrucciones ‘LL-Load Link’ y ‘SC-Store Conditional’, que son instrucciones utilizadas para implementar un *lock*. En cambio se hace uso de otro recurso: *spin locks* [8]. La eficiencia del procesador se ve afectada debido a esto.

G. Manejo de interrupciones

Otro factor a tener en cuenta en sistemas SMP es como manejar las interrupciones. En un procesador de un sólo

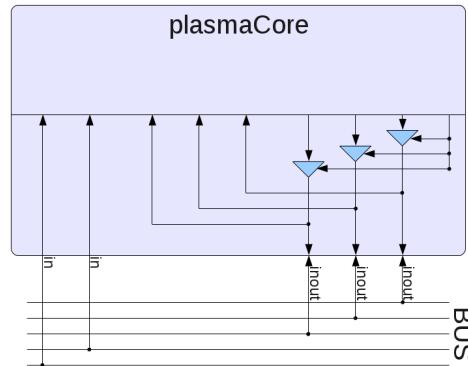


Fig. 3. Interfaz entre los núcleos y el Bus

núcleo, es simple, si la máscara y el estado de interrupciones coinciden en algún bit, el procesador es interrumpido. En un procesador multi-núcleo las posibilidades aumentan ya que existen varios núcleos que pueden ser interrumpidos. Cada uno de los núcleos a su vez puede habilitar y deshabilitar las interrupciones por separado. Algunos sistemas implementan también las interrupciones inter-procesadores.

Para el plasma multi-núcleo se quiere un manejador de interrupciones que las reparta en forma pareja entre todos los procesadores, de modo tal que no sea siempre el mismo el encargado de manejárlas, y tratar de que todos los núcleos se comporten igual. No tiene en cuenta si el procesador tiene bloqueadas o no las interrupciones, esto podría implementarse en alguna versión futura del plasma de modo de mejorar la latencia que tiene una interrupción. La latencia de la interrupción es el tiempo que tarda desde que da una excepción hasta que se ejecuta la tarea indicada.

III. IMPLEMENTACIÓN

A. Memoria Principal y Memoria Interna

El kit Nexys2 tiene incorporada una memoria de 16MB¹, lo cual equivale a 4.194.304 posiciones de memoria de 32 bits, suficiente para cargar el sistema operativo y programas de usuarios de tamaños considerables. Se diseñó un controlador de memoria para la misma. La memoria disponible en la placa tiene un ancho de palabra de 16 bits, por lo que para cada lectura/escritura del bus son necesarios 2 ciclos de lectura/escritura en la memoria principal o la memoria externa del kit.

La memoria interna implementada tiene un tamaño de 8kB, equivalente a 2.048 posiciones de memoria de 32 bits.²

B. Bus

El bus implementado posee las siguientes señales:

- 32 bits de lectura de datos/instrucciones.
- 32 bits de escritura de datos.
- 30 bits de direcciones.

¹128Mbits = 4MInstrucciones * 32bits/Instruccion.

²8kbyte = 64kbits = 2kInstrucciones * 32bits/Instruccion.

- 4 bits para la habilitación de escritura, uno por cada byte en las palabras de 32 bits.
- Memoria ocupada.

Cada núcleo tiene un buffer tri-state para conectarse al bus en las líneas de escritura de datos, direcciones y bits de habilitación de escritura. Las líneas de Lectura de datos/instrucciones y memoria ocupada, son entradas en cada núcleo. A la vez tiene como entradas a las líneas de escritura de datos, direcciones y las de habilitación como escritura, para leer estos valores cuando otro núcleo está utilizando el bus. Éstas líneas tendrán datos válidos sólo cuando alguno de los núcleos tenga acceso al bus, de lo contrario estarán en un estado de alta impedancia. La lectura de estas señales se utiliza para la coherencia de cache, que se explica en la sección III-C.

Otras señales de control que posee la interfaz de cada núcleo son:

- Bus snoop, para la implementación del algoritmo de coherencia, ver sección III-C.
- Pedido del Bus, utilizado para arbitrar el bus, ver sección III-D.
- Bus Ocupado, utilizado para arbitrar el bus, ver sección III-D.
- Habilitación de acceso al Bus, ver sección III-D.

C. Memoria cache y Algoritmo de coherencia de cache

Como se dijo previamente, la coherencia de cache se implementa mediante un protocolo de *Snooping*. Las características del bus y de la memoria cache son favorables para la utilización de este protocolo:

- Un bus simple: permite realizar *broadcast*, o sea que los diferentes núcleos pueden estar leyendo los datos del bus en todo momento. Esto no es posible en un bus switcheado.
- La política de escritura en memoria principal Write-Through: Implica que cada vez que se modifica un dato de memoria, esta información debe viajar por el bus hasta la memoria principal, lo cual normalmente es una desventaja. Pero en nuestro caso nos permite informarle a cada núcleo cada una de estas operaciones.

Por estas razones sólo es necesario una única señal extra para la implementación de la coherencia de cache. A esta señal la llamaremos *busSnoop*, y será una entrada a cada uno de los núcleos. La función de esta señal será informarle a cada uno de ellos cuando existe un dato nuevo en el bus. Si un mismo dato debe permanecer varios ciclos de clock en el bus, sólo debemos leerlo una vez. Decimos que es un dato nuevo cuando está por primera vez en el bus, luego de un período de clock, dejamos de enviar la señal de *snooping*. Con esta señal se indica que cada núcleo debe leer estos datos para luego comparar la dirección que viaja por el bus, con los identificadores que tiene en memoria cache. En caso de coincidir, implica que ese núcleo tiene almacenada esa misma posición de memoria en cache, y debe actualizarla con el dato que se encuentra en el bus que también puede ser leído mediante el mecanismo de broadcast.

La cache se implementa mediante una memoria ram dual-port sincrónica. El puerto A se utiliza para Lectura y Escritura:

- Lectura de indicadores, y en caso de estar almacenado en cache, lectura del dato en cuestión.
- Escritura de datos e indicadores en cache cuando se realiza alguna escritura de datos en memoria principal. El dato también se escribe en cache debido a la política de *Write-Allocate*³.
- Escritura de Datos e identificadores en cache cuando se produce un *read-miss*.

El puerto B se utiliza sólo para lectura: el puerto B está disponible únicamente para la lectura de identificadores cada vez que hay una señal de *snooping*. En esos casos lee el identificador de cache y lo compara con el de los datos que se encuentran en el bus.

Los datos en cache son actualizados con un ciclo de clock después de que algún núcleo tiene acceso al bus para alguna escritura.

D. Árbitro de bus

El árbitro del bus habilita o bloquea a los núcleos el acceso al bus. Cada núcleo envía una señal de pedido de bus a esta entidad cuando lo requiere. Esta señal se envía con un ciclo de anticipación. El árbitro elige uno solo de los núcleos para darle acceso, y al resto los bloquea, lo cual no quiere decir que sean pausados, sólo sucede si el núcleo bloqueado solicita acceso al bus en ese momento.

Cuando sólo un núcleo pide acceso al bus, el árbitro le brinda acceso directamente. En los casos que hay más de un núcleo pidiendo acceso al bus o que algún núcleo tiene acceso al bus mientras otro núcleo hace un pedido del mismo, el árbitro almacena en un cola, un buffer FIFO, los pedidos en el orden que se fueron sucediendo, de forma tal que se respete el ‘orden de llegada a la cola’ y de esta manera se tiene un reparto en la utilización del bus equitativo entre los diferentes núcleos. A medida que se va liberando el bus, el árbitro va quitando leyendo los pedidos de la cola. La cola tiene tantas posiciones como número de núcleos hay en el procesador, de forma tal que nunca puede quedarse sin espacio, ni siquiera en el peor caso que es cuando todos los núcleos hagan pedido del bus simultáneamente.

E. Manejo de interrupciones

La implementación de la entidad de manejo de interrupciones es simple. Cada vez que se da una interrupción se interrumpe un procesador distinto. Se va avanzando secuencialmente a través de todos los procesadores, de forma tal de no cargar sólo a un procesador con la tarea de manejo de interrupciones.

Esta entidad no tiene en cuenta si el procesador que se interrumpe tiene bloqueadas o no las interrupciones. Si se interrumpe un procesador cuando tiene bloqueadas las interrupciones, se debe de esperar a que las vuelva a habilitar para

³No se habla de write-miss, ya que la política es Write-Through y siempre se debe escribir en memoria principal, este o no almacenado la posición de memoria en cache.

TABLA I
TAMAÑO DEL PROCESADOR PLASMA

Número de núcleos	Slices Utilizadas
1	2635
2	4624
4	8030

hacer efectiva la interrupción y ejecutar la tarea pertinente. En los casos que se da esta situación, se tiene como consecuencia un aumento en el tiempo que se tarda en atender la interrupción. Es decir un aumento en la latencia de la respuesta ante interrupciones. En algunos sistemas operativos esta característica es fundamental, como los que se utilizan en algunos sistemas embebidos, donde las restricciones de tiempo son fundamentales. En algunas circunstancias esta latencia puede llegar a ser demasiado extensa, y deja de cumplir con las tolerancias permitidas.

F. Otras entidades necesarias

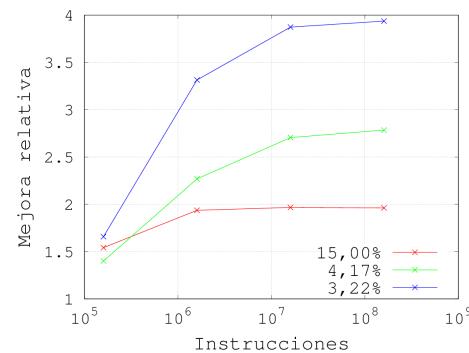
Se implementan también otras entidades necesarias en el procesador:

- Se implementa un registro fijo, al leerlo se obtiene como resultado el número de núcleo que realiza la lectura. Se mapea en memoria interna al final de la memoria ram efectiva. Se utiliza en el PlasmaOS para implementar la función *OS_CpuIndex()*.
- Se implementa un contador. Éste puede ser leído para obtener el valor de mismo y a su vez se utiliza para realizar interrupciones periódicas cada 10ms aproximadamente. Se utiliza en el PlasmaOS para hacer un reschedule por tick de las tareas que se están ejecutando.
- Se implementa registro que guarda la máscara que se aplica al estado de las interrupciones.
- Se implementa una UART, para la comunicación con otros dispositivos, como por ejemplo una PC, a través de la cual se le cargan los programas en memoria principal. Así como el contador, la UART puede interrumpir a los procesadores cuando hay datos disponibles o cuando se encuentra lista para enviar datos.

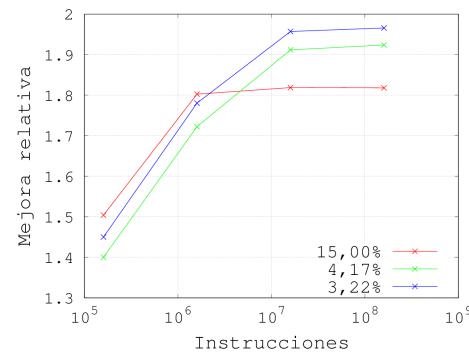
IV. CARACTERIZACIÓN DEL PROCESADOR

En esta sección se evalúan características de tamaño y desempeño. En la tabla I se puede ver el tamaño en *slices* que ocupa el plasma multi-núcleo. En estas condiciones se pueden implementar hasta cuatro núcleos en una FPGA Spartan3E-1200. Las evaluaciones de rendimiento que se muestran en este trabajo se realizan en procesadores de uno a cuatro núcleos. No se realizan pruebas con mayor cantidad de núcleos. En los resultados expuestos en las figuras 4 a 6 se ve que el diseño actual muestra mejoras en el rendimiento al ejecutar los programas de prueba.

La figura 4 muestra como mejora el rendimiento en el procesamiento a medida que aumenta la cantidad de trabajo que realizan las tareas de prueba, el trabajo que realizan las tareas se mide en número de instrucciones que ejecutan. Los resultados mostrados son para tres tareas distintas, donde la



(a) Al pasar de uno a cuatro núcleos.



(b) Al pasar de uno a dos núcleos.

Fig. 4. Mejora en el tiempo de ejecución al pasar de uno a cuatro núcleos. Muestra la relación en los tiempos de ejecución en función de la cantidad de procesamiento de las tareas que se corren. La cantidad de procesamiento se mide instrucciones que ejecutan las tareas.

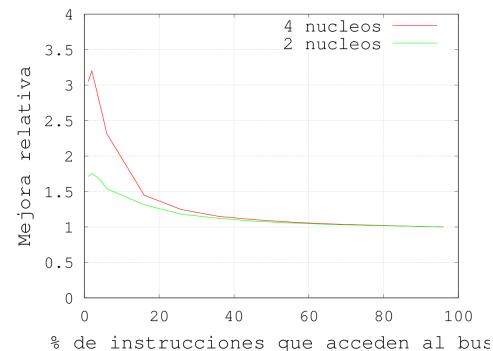


Fig. 5. Mejora en el tiempo de ejecución para cuatro y dos núcleos, frente a un núcleo, en función del porcentaje de instrucciones que acceden al bus en las tareas.

fracción de instrucciones que acceden a memoria cambia. En estas gráficas en particular se ven tareas con un 15,00%, 4,17% y 3,22% de tareas que acceden al bus del procesador. Las figuras 4(a) y 4(b) muestran los resultados al comparar el procesador de cuatro núcleos frente al de uno y el de dos núcleos frente al de uno respectivamente. Se ve que en todos los casos el rendimiento aumenta a medida que aumenta el trabajo que realizan las tareas en sí. Esto se debe a que el porcentaje de tiempo que le toma al sistema operativo

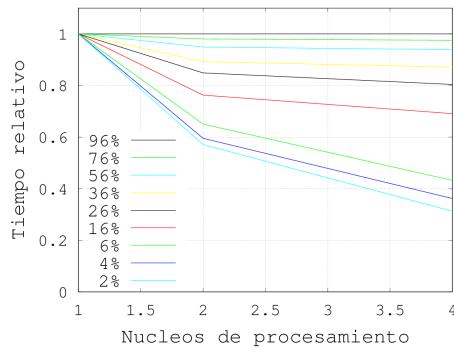


Fig. 6. Tiempo de ejecución para distintas tareas en función del número de núcleos de procesamiento. Las tareas difieren en el porcentaje de instrucciones que acceden al bus.

reescalar las tareas es cada vez menor. A partir de cierto valor, este tiempo se vuelve despreciable frente al tiempo de procesamiento efectivo. También se aprecia que siempre se llega a un límite en la mejora, el cual depende del porcentaje de instrucciones que acceden al bus. En el mejor caso de la figura 4(a) este valor se acerca a 4, mientras que en la 4(b) a 2, que son los respectivos valores teórico máximos que se pueden alcanzar en casos ideales.

El caso ideal es cuando no existen colisiones en el bus, y la única manera de asegurar que sean nulas es que ningún núcleo acceda al bus en ningún momento, lo cual es imposible. El aumento en las colisiones provoca que el rendimiento disminuya. Las colisiones en el bus generan un límite en el rendimiento de procesadores multi-núcleo. Esto se puede ver claramente en la figura 5, donde se muestra que la mejora en el tiempo de procesamiento de las tareas disminuye junto con el aumento del porcentaje de instrucciones que acceden al bus. También se ve un resultado indeseado, que es que para porcentajes muy bajos, la mejora vuelve a disminuir. Esto se debe a que para estos valores tan pequeños, el tiempo de procesamiento efectivo de las tareas disminuye y comienza a influir el tiempo necesario para reescalar las tareas, es decir deja de ser despreciable. Para poder despreciarlo se deben elegir tareas con mayor cantidad de procesamiento, como se mostró en los resultados expuestos en las figuras 4(a) y 4(b).

La figura 6 muestra resultados del tiempo de procesamiento para procesadores de uno a cuatro núcleos para tareas con distinta utilización del bus. Nuevamente se ve como el porcentaje de instrucciones que acceden al bus es crítico. Dependiendo de la frecuencia de utilización del bus la mejora es más o menos marcada al aumentar el número de núcleos. A medida que disminuye la utilización los tiempos mejoran en mayor medida al aumentar los núcleos de procesamiento. En algunos casos solo se ve que existe una mejora significativa al pasar de uno a dos núcleos y no de dos a cuatro. Esto se debe al límite impuesto por las colisiones en el bus, al aumentar la cantidad de núcleos, las colisiones aumentan. En esta gráfica también tenemos un resultado indeseado para los casos de 2% y 4%, también debido al hecho de que el tiempo efectivo

de procesamiento disminuye y empieza a influir el tiempo de reescalonamiento de las tareas. Este efecto se notaría menos en caso de ejecutar tareas que realicen mayor trabajo.

Por último se corren programas de prueba en implementaciones sin la memoria cache resultando en la nula mejora en el rendimiento, independientemente del número de núcleos y del tipo de tarea, ya sea que tenga o no gran porcentaje de instrucciones de acceso al bus. Este resultado es esperado, ya que todos los núcleos acceden al bus por el solo hecho de tener que leer la instrucción que ejecutan. El arbitro del bus permite el acceso al mismo de a un núcleo a la vez pausando al resto. Tener más de un núcleo carece de sentido, ya que el procesamiento efectivo sin memoria cache es igual o inclusive peor.

V. CONCLUSIONES

Se diseñó e implementó un sistema multi-núcleo sobre el cual se realizaron testeos de eficiencia. Los resultados presentados muestran la dependencia del rendimiento con el número de núcleos, el tipo de tarea que se tenga y la memoria cache. Los factores más determinantes para la obtención o no de una mayor eficiencia al procesar distintas tareas son la utilización del bus y la carga de procesamiento que éstas tengan. Se mostró también que existe un límite en la posible mejora al rendimiento y que puede estar muy alejado del caso teórico ideal.

Existen otros factores a evaluar en un futuro, entre ellos como se ve modificado el límite de una posible mejora al implementar otro tipo de memorias cache con otras políticas y que no sea compartida entre instrucciones y datos. Otros trabajos futuros pueden ser la implementación del plasma multi-núcleo, que soporte una cantidad de núcleos que no se necesariamente potencia de dos. También la implementación del monitor del sistema operativo, para controlar la correcta distribución de las tareas que se ejecutan, en ambientes de gran exposición electromagnética, que fue uno de los factores que impulsó este trabajo.

REFERENCES

- [1] Procesador Plasma y PlasmaOS: <http://opencores.org/project/plasma>
- [2] Arquitectura MIPS: <http://www.mips.com/>
- [3] J. Tarrillo, L. Bolzani, F. Vargas, *A Hardware-Scheduler for Fault Detection in RTOS-Based Embedded Systems* 2009.
- [4] D. Silva, K. Stangerlin, L. Bolzani, F. Vargas, *A Hardware-Based Approach to Improve the Reliability of RTOS-Based Embedded Systems* 2011.
- [5] Michael J. Flynn, *Computer Architecture*. Jones & Bartlett Learning, 1995.
- [6] John L. Hennessy, David A. Patterson *Computer Architecture - A Quantitative Approach*, 4th Edition. Morgan Kaufman Publishers, 2007.
- [7] David A. Patterson, John L. Hennessy *Computer Organization and Design - The Hardware/Software Interface*, 3rd Edition. Morgan Kaufman Publishers, 2005.
- [8] Andrew S. Tanenbaum, *Modern Operating Systems*, 2nd edition. Prentice Hall, 2002.



Foro Tecnológico

Pósters

ASICs





Implementación iterativa en hardware de un algoritmo de búsqueda de distancia más corta utilizando árboles k-D

Oscar Mario Saborío Berrocal
osaboriob@gmail.com

Sergio Saborío Taylor
ssergio92@gmail.com

Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica, Costa Rica.

Como parte de un proyecto de protección ambiental por medio de reconocimiento de patrones acústicos sobre redes inalámbricas de sensores, se propone una arquitectura de hardware a ser implementada como parte de un ASIC cuya tarea es encontrar cuál de N diferentes patrones conocidos (denominados centroides), se encuentra más cerca de un patrón de entrada dado, en un espacio de k dimensiones y utilizando un número de comparaciones menor que N. Se programará utilizando el lenguaje de programación C la estructura de datos árbol k-D propuesta por Friedman et ál. [1], para organizar los centroides de manera tal que cada nodo elige una dimensión de comparación, sobre la cual los centroides se clasifican en los que tienen el valor de esa dimensión mayor o menor a un valor específico, que se selecciona de modo tal que los centroides se separen en dos grupos de igual cardinalidad. Posteriormente se implementará el algoritmo de búsqueda del vecino más cercano sobre el árbol, de forma iterativa, basándose en la implementación recursiva de la biblioteca kd-tree, con el objetivo final de contar con una base para la implementación de la estructura y algoritmos previamente mencionados, en el lenguaje de descripción de hardware Verilog, procurando minimizar el consumo de potencia (orden de los microwatts o menor). Asimismo se desarrollará un módulo para interesar éste con el resto del sistema de reconocimiento de patrones acústicos.

Referencias:

- [1] Friedman, J., Bentley, J. & Ari, R. An algorithm for finding best matches in logarithmic expected time. Mayo 1975



“Simulations softwares: HSpice versus LTSpice”

Muswieck B.; Severo L.

Unipampa University

Eletroeste Tecnlogia & Automação

Unipampa University

Simulation software reduce engineers time of development, where SPICE simulation programs are used in electronic circuits. There are many SPICE software available on the market, on this paper we made a comparison between HSpice and LTSpice.

HSpice is expensive and considerable a high performance simulation software, because of it, if we have simulation results from it and LTSpice, at first, HSpice results are trusted with no doubt on it. For sure, a freeware against a paid and well know simulation tool, the freeware will start losing, but is that truth? LTSpice are comparable with HSpice? Will both perform the same result? To answer this questions two circuit were analyzed, transistor and an inverter, where modification in size of transistors, temperature and specifications were made. Each program produce an output file and both files are plotted using Matlab to be compared. The transistor model used was: T49S SPICE BSIM3 VERSION 3.1.

This paper put LTSpice against HSpice and demonstrate that both software produce same results on the simulations. We intend to simulate more complex circuits on both software to analysis if them still give same results. Some warnings about perform the same results on both software are: if using schematic on LTSpice be sure to connect all the nets and watch your temperature simulation's, because default temperature on LTSpice is 27°C and HSpice is 25°C.



Sistema para testeo en dosis total de memorias micro-SD

José Lipovetzky¹, Lucas Chiesa¹, Ariel Burman¹, Gerardo Richarte², Adrián Faigon¹

jlipove@fi.uba.ar, lucas.chiesa@gmail.com, arielburman@gmail.com, gera@satellogic.com, afaigon@fi.uba.ar

¹Facultad de Ingeniería-Universidad de Buenos Aires, Av. Paseo Colón 850, C1063ACV, Ciudad de Buenos Aires, Argentina; ² Satellogic, San Carlos de Bariloche, Argentina

En algunos sistemas embebidos expuestos a radiación como micro-satélites es conveniente, por razones de costo, utilizar memorias microSD como sistema de almacenamiento masivo. En el presente trabajo se muestra el diseño de un sistema para caracterizar frente a efectos de dosis total de radiación ionizante memorias microSD, y los primeros resultados obtenidos.

El sistema diseñando y construido permite alimentar las memorias con diferentes tensiones, escribir datos, y medir el consumo de corriente en diferentes modos de operación, de forma de excitar a los dispositivos durante la irradiación y además verificar su funcionamiento bajo diferentes condiciones.

Como primer experimento, se irradiaron ocho memorias de dos fabricantes diferentes, bajo las condiciones del método 1019.H de la norma MIL-STD 883 para testeo ante dosis total, usando una fuente de ^{60}Co hasta una dosis de 150 krads y una tasa de dosis de 75 rads/s. La mitad de las memorias fueron escritas durante toda la irradiación, mientras que en las otras sólo se evaluó la capacidad de retención de datos. Se detectaron errores en los datos guardados, aunque las memorias continuaron siendo funcionales. Se presenta un primer análisis cuantitativo de los resultados.





Artículos

DSPs





Analysis and implementation of a noise reduction algorithm for a low-cost hearing aid device

Alejandro J. Uriz, Jorge Castiñeira Moreira Pablo Agüero, Juan C. Tulli, Esteban González,Francisco Denk
CONICET - Communications Lab
National University of Mar del Plata
Mar del Plata, Argentina
Email: ajuriz@conicet.gov.ar, casti@fimdp.edu.ar

Communications Lab
National University of Mar del Plata
Mar del Plata, Argentina
Email: (pdaguero,jctulli)@fimdp.edu.ar

Abstract—Nowadays there are many people suffering from hearing impairments. The high cost of assistive listening devices leaves out of reach for many people a technological solution. The main goal of this work is to carry out an analysis of noise reduction methods. This study is focused on selecting and implementing a denoising algorithm in a low-cost hearing aid device. Also, digital signal processing techniques are applied to obtain basic features of a high-end assistive listening device. Objective experiments are performed to analyze the performance of the system.

Index Terms—Assistive listening device, digital signal processing, SNR improvement, dsPIC.

I. INTRODUCTION

HEARING impairments are conditions that affect an important percentage of the Society. Several techniques have been developed to assist people with hearing impairments. Such techniques have been used to design several types of assistive devices, for instance, headsets. Some of these devices are developed by Widex [1]. This company offers several types of products, from analog headsets (in some cases only a fixed-gain amplifier) to devices based on digital signal processors, which are capable of obtaining better results than the analog devices. Actual devices perform tasks like voice compression, noise reduction and dynamic equalization, among others. The most important contribution of this kind of devices is the potential solution for each particular hearing impairment condition by customizing it to a particular user.

A vital feature to improve the performance of an assistive listening device is the signal to noise ratio (SNR). This parameter measures the relation between the power of a desired signal to the power of background noise.

There are several algorithms to improve the SNR of a signal. One of the most widely used is the singular value decomposition (SVD) method [2], [3]. This technique decomposes a signal to obtain its singular values. Then, it considers that the higher singular values are associated to the desired signal, while the lower singular values are related to noise. Thus, the signal is reconstructed using the highest singular values. Consequently, the reconstructed signal has less noise than the original. The main limitation of this technique is the difficulty to obtain the singular values using a matrix representation of the signal, which is generally done using a Toeplitz matrix.

Thus, a considerable number of instructions and a significant amount of memory is required to apply this method. On the other hand, noise reduction techniques using Wavelets[4], [5], [6] require a domain transformation which involves a large number of instructions. But, if it is desired to perform an implementation of these techniques in a low cost DSP device, limitations appear due to processing time and memory usage. Thus, a reasonable choice to improve the SNR is the cross-correlation algorithm [7], which also operates in time domain. A similar approach was used in [8] to determine the direction of arrival of a signal on a microphone-array using a low-cost hardware based on dsPIC devices.

The goal of the present paper is to implement a technique to improve the output signal to noise ratio of a digital assistive listening device implemented in a low cost digital signal processor. Algorithms are implemented in a DSP device from Microchip. The chosen device is the dsPIC33FJ128GP802 [9].

The paper is organized as follows. In Section II the most important features of three classical denoising algorithms are presented, focusing on hardware requirements. Section III presents the developed hardware and depicts the implementation of the algorithm used to improve the SNR of the system. Section IV describes the carried out objective experiments. Finally, Section V summarizes the conclusions of the work and provides future research lines.

II. NOISE REDUCTION ALGORITHMS

Digital assistive listening devices [1] are the best solution for people with hearing impairments. These solutions are based on digital signal processors, which process the sounds to be perceived by a person with a specific hearing impairment. The main features of a hearing aid of that family are:

- **Soft sound amplifier.** It allows the user to perceive soft sounds associated to weak or distant speech.
- **Audibility extender.** This is the most important functionality. It allows the user to hear sounds spectrally placed into the band where he/she has an impairment.

Although there are several features that improve the usability of a hearing aid device. It is vital to maximize the comfort of the user, thus it is necessary to maximize the signal to noise ratio (SNR) of the output signal. In the next subsection three of the most widely used algorithms are studied.



A. Singular value decomposition algorithm

Noise reduction using the singular value decomposition algorithm is a method based on a matrix representation decomposition of a signal. This technique models the signal as a matrix A , which is decomposed to obtain an approximation in three matrices U , Σ and V . Thus, the p highest singular values are associated to the desired signal, while the lower singular values are considered related to noise. Suppose we want to apply noise reduction to a signal $s(n)$, which has a length of $N=256$ samples. The first step of this method is to obtain a matrix model of the signal $s(n)$. It is done by obtaining A , the Toeplitz matrix of $s(n)$, which is a $N \cdot N$ square matrix. Each row of this matrix is obtained by left-shifting the first N elements of $s(n)$. A more detailed description of this matrix representation is presented in [3]. Once A is obtained, it is necessary to obtain U , Σ and V matrices, so that:

$$A = U \cdot \Sigma \cdot V^T \quad (1)$$

Where U and V are unitary matrices and Σ is a diagonal matrix, which is composed of the singular values of A . The definition of Eq. (1) corresponds to the classic definition of a full range matrix. If A had been a matrix of $M \cdot N$ dimensions, the maximum rank would be $\text{rank}(A)=R < M$. Then, Eq. (1) can be expressed as:

$$A = \sum_{k=1}^r \sigma_k u_k v_k^T \quad (2)$$

where u_k is the k^{th} column of the U matrix; v_k , k^{th} column of the V matrix, and σ_k are the first r singular values sorted in a decreasing order. Then, due to $\text{rank}(A)=r$, and $\sigma_{r+1};\sigma_{r+2};\dots;\sigma_m$ are zero. If we consider that the $r - p$ last singular values are very small, we could truncate Eq. (2) assuming $k = p$, then:

$$p < r : \quad \hat{A} = \sum_{k=1}^p \sigma_k u_k v_k^T \quad (3)$$

This new matrix \hat{A} has rank p , and is called low rank approximation. Also, it is the best approximation to A with a p rank matrix in the sense of mean squared error. In noise reduction, the highest singular values are associated to the desired signal, while the lowest singular values associated to the noise components. Then, truncating the number of singular values is equivalent to filter components of a signal. There are several criteria to determine the new rank p . They are based on an estimation of the noise contained by the signal. Consequently, the signal is filtered by truncating the corresponding p singular values. It should be noted that the truncation of an excessive amount of singular values generates a degradation in the desired signal. Therefore, it is necessary to control truncation in order to avoid degrading the desired signal.

The main limitation of this technique is the amount of required memory to implement the U , Σ and V matrices.

TABLE I
MEMORY REQUIREMENTS TO IMPLEMENT THE SVD DENOISING ALGORITHM, FOR A FULL SIZE SVD IMPLEMENTATION AND A REDUCED SIZE SVD IMPLEMENTATION.

	Full SVD	Truncated SVD
Variable	Size	Size
input signal $s(n)$	$2 \cdot N$	$2 \cdot N$
A	$N \cdot N$	$p \cdot N$
U	$N \cdot N$	$N \cdot p$
Σ	$N \cdot N$	$p \cdot p$
V	$N \cdot N$	$N \cdot p$
denoised signal $\hat{s}(n)$	N	N

Also, the computational load of the procedure for obtaining the singular values of the matrix is very high. For example, given a signal $s(n)$ of N samples, $2 \cdot N$ samples are needed to generate the Toeplitz matrix. Once the Toeplitz matrix has been generated, singular values of A must be obtained. Then, depending on the level of required truncation, U , Σ and V matrices must be obtained. The last step consists of applying the operation of Eq. (3), and to synthesize the denoised signal $\hat{s}(n)$. The minimum memory requirements for this implementation are detailed in the second column of Table I.

In the second column of Table I U , Σ and V matrices were estimated of dimension $N \cdot N$, allowing to denoise low levels of noise without degrading the signal. Also, the first row of Table I shows that $2 \cdot N$ samples are required for a N samples signal to generate the Toeplitz matrix. The amount of data required to generate the matrices and vectors is about $4 \cdot N^2 + 3 \cdot N$. If the implementation is done using *16 bits* data, in *long* format, and $N = 256$, $(4 \cdot N^2 + 3 \cdot N) \cdot 2 = 525824$ bytes of data are required, which is an excessive amount of memory for a DSP.

In order to reduce the memory requirements, it is possible to determine a maximum value of p . This limitation produces an important reduction in the memory space required. Then, for $N = 256$ and $p = 50$, the variables have sizes as presented in the third column of Table I. Under the conditions established in the third column of Table I, the memory requirements of the SVD algorithm are around 57736 bytes. Although, in this conditions the memory usage is considerably reduced, there are not low-cost DSP with 58KB of data memory. Also, the processing time associated with the required operations is very high for real-time work, which in addition discards the possibility of working with an external memory. Also, the computational load of this implementation is very high, thus is not possible to implement it for working in real-time.

B. Wavelets filtering

Wavelet analysis is a relatively mature filtering method, it has good local behavior in time-frequency. Also, it has an inherent advantage dealing with non-stationary signals. Wavelet noise filtering algorithms [5], [6] can be divided into two steps: filtering and signal reconstruction. The first stage decomposes the noisy signal $s(n)$ using a suitable wavelet

TABLE II
MEMORY REQUIREMENTS TO IMPLEMENT A DB4 WAVELET DENOISING.

Variable	Standard	Optimized
Variable	Size	Size
input signal $s(n)$	N	N
W	$N \cdot N$	$4 \cdot N$
analysis and synthesis coefficients	$8 \cdot N$	$8 \cdot N$
denoised signal $\hat{s}(n)$	N	N

base. Also, this stage processes the high frequency coefficient of each layer using the corresponding threshold. The second stage, recovers the processed signal $\hat{s}(n)$. The data memory requirements of a noise reduction algorithm using wavelets are shown in Table II.

Second column of Table II shows that the memory requirements are $N \cdot N + 10 \cdot N$, which for $N = 256$, using *16 bits, long* format data, requires 137KB of data memory. Also, the third column of Table II presents the memory requirements for an optimized implementation of a wavelet noise reduction algorithm. In this case, the coefficient matrix was reduced to four vectors of length N . Thus, the memory requirements are $4 \cdot N + 10 \cdot N = 7KB$. These amount of memory can be more than the RAM available for a low-cost DSP. Then, it is necessary to implement a noise reduction method which utilizes less amount of data memory.

C. Crosscorrelation algorithm

Methods presented in the Subsections II-A and II-B are algorithms to improve the SNR of a signal by means of monoaural techniques, but data memory limitations and processing time exist. Hence, due to these restrictions one of the best methods to improve the SNR is the cross-correlation algorithm [7], [8], which allows us to improve the SNR by using two monoaural systems as an array of microphones. This algorithm is used in array systems to determine the arrival direction of a wave. It can also be used to improve the reception SNR of the array. In this case, the cross-correlation algorithm is used to improve the SNR of the synthesized audio. The system is based on the calculation of the cross-correlation between the signals acquired by each receptor, in this case each microphone. Figure 1 shows an scheme of the proposed array of microphones.

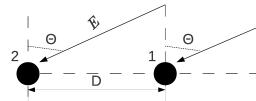


Fig. 1. Scheme of the array of microphones.

This method is based on the fact that the signal from the direction θ_s arrives at the microphone 1, then travels a distance ξ and, is received by the microphone 2. Since: $\xi = D \cdot \sin\theta_s$, the direction of arrival (DOA) can be determined using the Eq. (4).

TABLE III
MEMORY REQUIREMENTS TO IMPLEMENT A CROSSCORRELATION DENOISING.

Variable	Size
left and right input signals ($s_L(n)$ $s_R(n)$)	$2 \cdot N$
crosscorrelation result vector	$2 \cdot N$
averaging result vector (denoised signal $\hat{s}(n)$)	N

$$\theta_s = \sin^{-1}\left(\frac{v \cdot \tau}{D}\right) \quad (4)$$

where v is the speed of sound, and τ the time that the signal requires to travel the distance v . Then, once a value τ is obtained in the range $-T < \tau < T$, DOA can be estimated. In this paper, τ is used to determine the displacement between signals. Then, one of the signals is shifted the number d of samples $d = \frac{\tau}{T_s}$, where T_s is the sampling period. In this work, τ is determined by maximizing the cross-correlation $\Phi(\tau)$ (see Eq. (5)) between $X_1(t)$ and $X_2(t)$, which are the signals acquired by each microphone.

$$\Phi(\tau) = \frac{1}{N} \sum_{t=0}^{N-1} X_1(t)X_2(t + \tau) \quad (5)$$

Where $\Phi(\tau)$ is a vector of $2 \cdot N - 1$ elements.

Then, assuming that the signal that arrives to both microphones is the same, if this maximum value of $\Phi(\tau)$ coincides with the center of the obtained vector, both signals were simultaneously received by the microphones. But, if the maximum value of the result does not match the center of the cross-correlation vector, it is possible to conclude that the signal has arrived with a time difference τ to each microphone. The minimum delay ($d = 1$) between microphones is established by the sampling frequency of AD converters, which determines the sampling period T_s .

Once the resulting cross-correlation vector is obtained, the distance $d = \frac{\tau}{T_s}$ between the position of the maximum and the center of the cross-correlation vector determines the delay between received signals. Then, this method averages the signal received by each microphone to improve the performance of the total received signal. To perform this operation, it is necessary to shift one of the received vectors in order to compensate the difference in the arrival time τ . Once the signal has been shifted, both signals can be averaged, and with additive white gaussian noise (AWGN), the resulting signal has an improvement in the signal to noise ratio. For an array of two microphones the improvement in the SNR is 3dB for levels of input noise higher than 0.01Vpp. For lower values of input noise, the improvement is less evident, because the levels of noise are negligible with respect to the signal level. In Table III the memory requirements are presented for an implementation of the crosscorrelation algorithm applied to two input signals $s_L(n)$ and $s_R(n)$ of length N .

According to Table III for $N = 256$, the data memory requirement is 2560 bytes, which is an amount of data memory available in most commercial low-cost DSP. By comparing the



obtained result it can be seen that the crosscorrelation denoising algorithm is the best choice to carry out an implementation in a low-cost dsPIC. The hardware used for the implementation of the algorithm is presented in the next section.

III. IMPLEMENTATION

The performance of studied methods was measured using MATLAB. The algorithms with the best SNR improvement were wavelets denoising and the cross-correlation method. Based on results obtained from the resources analysis of the section II, the cross-correlation noise reduction algorithm was chosen for the implementation. In the next subsection, the implemented hardware and software are presented in detail.

A. Hardware developed

The main goal of the proposed digital assistive listening device is to process sounds that are perceived by a person with a hearing impairment, so they can be heard more naturally. The system is based on a dsPIC33FJ128GP802, a digital signal processor of Microchip. The most relevant features of the implemented system are listed in the next subsections.

1) *Microphone preamplifier*: When the voice signal is extracted from the microphone, it has a very low level. Then, it is necessary to amplify the level of the signal for the next stages.

2) *Antialiasing filter*: In order to limit the bandwidth of the signal for sampling, an antialiasing filter is implemented. The filter parameters are obtained from specifications of the digital signal processor. In this case, the sampling frequency is 16KHz and the AD converter has a resolution of 12 bits. Then, the maximum stop frequency is 8KHz, and the rejection in the stopband must be at least 72dB. These filter specifications can be satisfied synthesizing a 8th order elliptic Sallen-Key filter. In this case, in order to reduce discrepancies between the theoretical and the implemented model, the problem was faced using the MAX7404 [10], an 8th order, lowpass, elliptic, switched-capacitor filter. This integrated filter is configured by means of an oscillator, which defines the cut-off frequency of the filter. There are two ways to configure it:

- External oscillator: In this case, the cut-off frequency f_{cut} is defined by means of an external oscillator f_{osc} , which must be tuned as $f_{osc} = 100 \cdot f_{cut}$.
- Internal oscillator: This configuration uses an external capacitor to define the frequency of an internal oscillator, and consequently sets the cut-off frequency as $f_{cut}[KHz] = \frac{34 \cdot 10^3}{100 \cdot C_{osc}}$, with C_{osc} defined in pF.

Other advantages of this integrated filter is the increased consistency. This is due to the reduction of the number of discrete components, which have a tolerance of up to 20%. Also, the reliability of the system is incremented. It is a direct consequence of the reduction in the number of connections between components.

3) *Automatic gain control (AGC)*: Once the signal is filtered, it is necessary to adjust its level to fit the AD converter input range. In addition, since the distance between the microphone and sound source changes along the time, the level

of signal also changes. The best way to solve this problem is to use an automatic gain control system, which amplifies the level of the signal, maximizing the use of the AD converter input range. Consequently, the weakest sounds are amplified by a higher factor, while the loudest sounds are amplified by a lower factor. Then, this stage solves the problem of perceived weak or soft sounds.

The implementation of the AGC system was made using two blocks. The first one, uses a PIC12F683 to acquire the audio signal and applies the Automatic Gain Control algorithm. The output of this stage has pulse-width modulation (PWM), and it is used to control the gain by means of the second block: a voltage controlled amplifier. Then, the audio signal entering the filter stage is amplified using a factor determined by the AGC algorithm.

4) *Digital signal processor*: The digital signal processor used in this project is the dsPIC33FJ128GP802. The relevant features of this low cost DSP are listed below:

- **40KB of program memory**. This makes it suitable for use with cross compilers.
- **16KB of RAM**. 2KB are shared with direct memory access (**DMA**) buffer as dual ported RAM.
- **Up to 40 MIPS operation**.
- **Low cost**. Its price is US\$ 4, much lower than a classic DSP.
- **16-bit wide data path**.
- **12-bit@500ksps integrated analog-to-digital converter (ADC)**.
- **16-bit@100ksps integrated digital-to-analog converter (DAC)**.
- **Serial Peripheral Interface (SPI)**. This allows the communication between the DSP and several peripherals.

It should also be noted that the documentation about the Microchip devices and their libraries is available on the internet without any cost. The dsPIC33FJ128GP802 allows running multiple tasks **simultaneously** which is a great advantage. In this particular case those tasks are the acquisition of the current data segment and the processing of the previous one. This is accomplished using the DMA module of the device [9], which operates independently from the main processor. As a result, the processing time of each segment is reduced to almost half. Considering the fact that the main goal of the project is to develop a device that works in real time, this time saving is of utmost importance. Another aspect to be taken into account is the utilization of DMA techniques, which increase the system performance, because they minimize interruption sources. In other words, peripherals perform the data transfer using the DMA module, and delays are not added to the main program execution. In particular, the reduced times are:

- Processing time of the interruption routine.
- System stack accessing, read and storage time.
- Access time to peripherals.

In this case, an audibility extender[1] is implemented using the SPINC function [11]. This device can be adapted to the needs of each individual user, by simply changing the

compression factor value and the displacement to be made. Because the chosen dsPIC does not have hardware support to perform fast floating-point operations, those operations must be emulated in software, which greatly decreases performance. Hence, the FFT algorithm was adapted to work in 16-bit fixed-point arithmetic using magnitude scaling.

The processing time of the compression algorithm is $T_{SPINC} = 0.29ms$ significantly less than the one for the FFT and IFFT algorithms ($1.85ms$ and $1.83ms$ respectively) in both configurations.

Times were measured using a $f_{sampling} = 16.288KHz$, and in these conditions it is possible to see a difference between the acquisition and processing times ($15.72ms$ and $3.97ms$ respectively). This difference occurs because the tasks run in parallel, so the total processing time of the system is regarded as the greatest of them, which in the case of segments of $N=256$ samples is $t_{total} = 15.72ms$. Another aspect to consider is the amount of necessary RAM and program memory, which remains low. There is a limitation due to the amount of DMA memory available (in this case is 2KB), which is fully utilized. This imposes a limitation on the maximum performance that can be obtained from the system.

5) *Output stage:* The DA converter has differential outputs. Then, it is necessary to obtain the difference of both signals to take advantage of the benefits of this type of output. This is made using a stage composed by two operational amplifiers, which also amplifies the synthesized signal, so it can be properly perceived.

Figure 2 shows a block diagram of the implemented system.

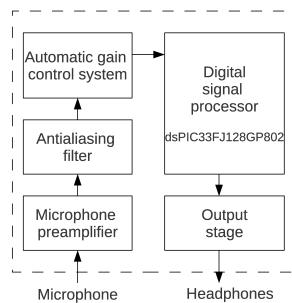


Fig. 2. Block diagram of the implemented system.

B. Description of the algorithm implementation

The cross-correlation algorithm uses the signal received from multiple receivers (in this case two microphones) to improve the SNR. In order to carry out the improvement, the next architecture composed by two systems presented in subsection III-A was defined. The basic system associated to the right ear microphone, transmits data via SPI to the basic system associated to the left ear microphone. Once the data is received, the cross-correlation algorithm is applied to the data acquired by the AD converter of the receptor. In order to take advantage of the DSP, this link is established using the serial peripheral interface (SPI) module [9], [12] of the dsPIC. This is a synchronous serial interface widely used to communicate

several types of peripherals. Another advantage is that it allows to transfer data with a speed up to 10Mbps, which is a critical factor to obtain a real time final system. Thus, the right ear system was configured as SPI master, and the left ear device as SPI slave. The communication was established using the SPI Framed mode [9], [12].

In this configuration, the system frequency of each dsPIC must be exactly the same. Thus, an external cristal was used to generate the clock signal of the master device. The same signal was used also as external clock signal of the slave device. A critical consideration must be done to connect the oscillator to the slave device: the effect of the input capacitance of the gate where the clock signal is injected. This capacitance is in the order of 60pF. In addition, there is a capacitance due to the shielded cable used to connect the device, so that an equivalent capacitance of around 70pF appears. This capacitance affects the system frequency of the master device. This problem was solved connecting a 8.2pF serial capacitor to the cable. Then, the resulting capacitance is reduced and the master clocks normally operates. Caution must be taken about the level of input clock signal amplitude on the slave device. An scheme of the resulting system is presented in Fig. 3.

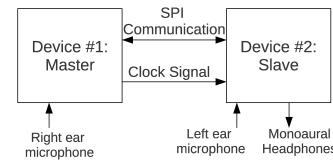


Fig. 3. Implemented architecture. The clock signal of the master device is used to excite the slave device. This is done to synchronize both devices.

Once the systems are synchronized and the communication is established, the cross-correlation method must be applied to improve the SNR of the signal. The proposed algorithm operates using the vector received via SPI and the vector acquired by the AD converter of the slave device.

In the implemented configuration, the master transmits a vector of $2.N$ samples for each frame, with N defined as $N = 256$. The receiver uses this vector, and the vector of the last $3.N$ samples acquired via the AD converter to carry out the improvement of the SNR from each frame.

Once the data was received, two pointers are defined. The first pointer, called p_0 , points to the middle of the acquired vector, while the second pointer ($p_{reception}$) points to the sample in the position $N/4$ of the received vector. The proposed structure is presented in Fig. 4.

The N first elements from each pointer, are cross-correlated using the VectorCorrelate() instruction of the C30 Microchip Library [13]. The result of this operation is stored into a vector of $2N - 1$ elements. If the maximum value of this resulting vector is placed in the center of them, then both signals do not have a delay. But, if a difference of d samples exists between the center of the resulting vector and the position of the maximum value, then both signals have a delay of around $\tau = d.T_s$, where T_s is the sampling period. A representation of the described registers is shown in Fig. 5.

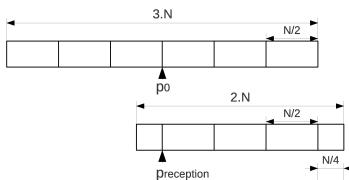


Fig. 4. Representation of the acquired buffer (top), and the received via SPI buffer(bottom). Pointers p_0 and $preception$ are defined in the desired positions.

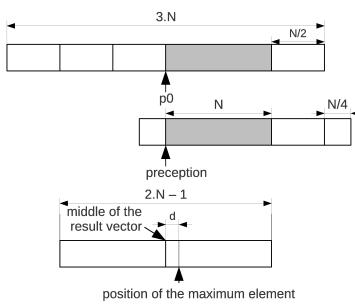


Fig. 5. Scheme of the vectors used into the cross-correlation operation. The N elements that follow the pointer p_0 (top) are correlated with the N elements that follow $preception$ (middle). The resultant vector (bottom), has a length of $2N - 1$ elements, and the distance between the center of the vector and the maximum element d defines the delay between signals.

Once the delay d is established, the pointer $preception$ is incremented (or decremented) d elements. This displacement can be in a positive or negative direction, depending on the result of the cross-correlation.

Then, the first $3.N/2$ elements starting in the pointers p_0 and $preception + d$ are averaged, and stored in a new vector, which is used to synthesize the signal using the TD-OLA algorithm. Two new pointers p_{ODD} and p_{EVEN} are defined in the resulting vector, which points to the start of odd and even synthesis frames respectively.

Finally, in order to save the received information, $2.N$ elements starting from the pointer p_0 are left shifted N elements to save the past information. Then, a new acquisition starts from the pointer p_0 , and the procedure is repeated.

IV. EXPERIMENTS

A set of objective experiments were made to test the performance of the system. In the first experiment the processing times of each process were measured.

Table IV shows that the sum of the SPI communication, the cross-correlation enhancement algorithm and other involved times, is lower than the acquisition time. Thus, it is possible to implement the system in real time.

Since the sampling frequency is 16kHz, and the distance between microphones is 25cm, the spacial resolution obtained by the system is 10 degrees. Also, in order to verify the correct implementation of the algorithm, the signal to noise ratio was measured for several input values of a sine wave of 1kHz and sustained vowels. The result was an improvement of around 3dB in the operating range, by applying the developed

TABLE IV
PROCESSING TIME, ACQUISITION TIME AND MEMORY USAGE FOR THE IMPLEMENTATION OF THE CROSS-CORRELATION ALGORITHM FOR THE SLAVE DEVICE.

SPI+Enh+Proc.	Acq.	RAM	DMA	ROM
6.09ms	15.72ms	96%	100%	8%

cross-correlation algorithm. Due to it, the final output SNR of the system is 48dB. Consequently, the improvement is in the order of theoretical estimations. Another important factor is the resource utilization for this implementation. Table IV presents the most relevant aspects.

Table IV shows that data memory is almost fully utilized, while DMA memory is totally used. Furthermore, there is a remnant of available program memory. Thus, if any additional functionality is desired, the algorithms should be redesigned to take advantage of the available program memory.

V. CONCLUSIONS

A digital assistive listening device which uses digital signal processing techniques to obtain the most important features of a high-end commercial assistive device was presented.

In addition, it was demonstrated that it is possible to successfully implement the cross-correlation algorithm to improve the Signal-to-Noise Ratio of the output signal. The obtained improvement is in agreement with theoretical expectations.

As a further work, additional features will be implemented, such as a zen music generator and a feedback control to prevent a coupling between the system and other electronic devices.

REFERENCES

- [1] Widex Inc.<http://www.widex.com/>, Lyng, Denmark.
- [2] H. G. Gauch, Jr., *Noise Reduction By Eigenvector Ordinations*, Ecological Society of America, Vol. 63, No. 6, pp. 1643-1649, 1982.
- [3] E. V. de Payer, *Preprocesado de la señal de voz: el método de la descomposición de subespacios*, Revista Argentina de Bioingeniería, Vol.16, No.1, June 2010.
- [4] V. Balakrishnan, N. Borgesa and L. Parchment, *Wavelet Denoising and Speech Enhancement*, 2006.
- [5] T. Young and W. Qiang, *The realization of Wavelet Threshold noise filtering Algorithm*, Proceedings of 2010 Conference on Measuring Technology and Mechatronics Automation, pp 953-956, 2010.
- [6] Ch. Dolabdjian, J. Fadili and E. Huertas Leyva, *Classical low-pass filter and real-time wavelet-based denoising technique implemented on a DSP: a comparison study*, The European Physical Journal Applied Physics, Vol.20, pp 135-140, 2002.
- [7] A. K. Tellakula, *Acoustic Source Localization Using Time Delay Estimation*, Degree Thesis. Bangalore, India: Supercomputer Education and Research Centre Indian Institute of Science, 2007.
- [8] S. Takahashi, T. Morimoto, *Development of Small-size and Low-priced Speaker Detection Device Using Micro-controller with DSP functions*. Proceedings of the IMECS 2011, Vol.1, 2011.
- [9] Microchip Inc., *dsPIC33FJ128GPX02/X04 Data Sheet, High Performance 16-bit Digital Signal Controllers*. <http://www.microchip.com/>, 2009.
- [10] Maxim Inc., *MAX7400/MAX7403/MAX7404/MAX7407 8th-Order, Low-pass, Elliptic, Switched-Capacitor Filters*, 1999.
- [11] E. Terhardt, *The SPINC Function for scaling of frequency in Auditory Models*. Journal of Acoustic, Vol.77 pp.40-42, 1992.
- [12] Microchip Inc., *Serial Peripheral Interface (SPI) - dsPIC33F/PIC24H FRM*. <http://www.microchip.com/>, 2011.
- [13] Microchip Inc. *16-Bit Language Tools Libraries*. <http://www.microchip.com/>, 2005.



Implementación de un Receptor BPSK de uso espacial utilizando un DSP

Adrián Carlotto, José Juárez, Gerardo Sager, Gustavo Mercado

Resumen—El Sistema Satelital Argentino de Recolección de Datos Ambientales, es un sistema en el que se recolectan mensajes transmitidos desde plataformas fijas o móviles, cercanas a la superficie terrestre o sobre ella. La señal recibida en el satélite es procesada en vuelo a partir de las muestras en frecuencia intermedia, utilizando un DSP. El trabajo presenta en forma somera al sistema, como así también la implementación del receptor haciendo incapié en las etapas de procesamiento digital. Finalmente se enumeran las pruebas en tierra y los primeros datos en vuelo.

Palabras Clave—Recolección Datos Remotos, DSP, DCS, Receptor BPSK.

I. INTRODUCCIÓN AL SISTEMA

EL Sistema Satelital de Recolección de Datos Ambientales, es un sistema en el que se recolectan datos transmitidos por plataformas autónomas denominadas DCP (Data Collection Platforms), fijas o móviles. Éstas pueden encontrarse en la superficie terrestre, sobre boyas en los océanos y ríos, en globos, entre otros. Los datos generalmente se corresponden con variables del medioambiente de la plataforma. Estos mensajes son procesados por el receptor DCS (DAta Collection System) ubicado en el satélite, luego almacenados y finalmente transmitidos a la Estación Terrena Córdoba (ETC), para su posterior procesamiento y distribución [1]. El sistema fue concebido para recibir en promedio, dos mensajes por día de hasta 200 plataformas distribuidas sobre la superficie del territorio nacional, aunque podría utilizarse para recibir a una DCP ubicada en cualquier parte del planeta. A escala global, conviven sistemas similares como ARGOS DCLS (ARGOS Data Collection and Location System) el cual se originó con un acuerdo entre la agencia espacial francesa CNES (Centre National D'Etudes Spatiales), NASA (National Aeronautics and Space Administration), NOAA (National Oceanic and Atmospheric Administration) de EEUU y ahora EUMETSAT (European Organisation for the Exploitation of Meteorological Satellites) [2]. En Latinoamérica, un sistema similar actualmente operativo es el Sistema Nacional de Dados Ambientais (SINDA) de Brasil, desarrollado por el Instituto Nacional de Pesquisas Espaciais (INPE), cuyo inicio se remonta al año 1991 [4]. El sistema Argos, como el argentino, realiza un pre-procesamiento de las señales recibidas en vuelo y

Adrián Carlotto es integrante del GrIDComD, se desempeña como JTP ordinario en la FI-UNLP y es miembro del IEEE. Email: carlotto@ing.unlp.edu.ar

José Juárez es integrante del GrIDComD, se desempeña como JTP ordinario en la FI-UNLP y es miembro del IEEE. Email: jjuarez@ing.unlp.edu.ar

Gerardo Sager es integrante del GrIDComD, se desempeña como Profesor Titular en la FI-UNLP y es miembro del IEEE. Email: ger@ing.unlp.edu.ar

Gustavo Mercado se desempeña en la CONAE. Email: mercado@conae.gov.ar

luego las transmite a una estación terrestre desde donde se realiza la validación y distribución de datos a los usuarios [3]. A partir del lanzamiento en el año 2006 del satélite europeo Metop A (primer satélite meteorológico europeo de órbita polar), se puso en operación el sistema Argos 3, que además de aumentar la capacidad de procesamiento de su antecesor, incorporó la posibilidad de una comunicación básica con la DCP. Actualmente este sistema global consta de aproximadamente 21000 plataformas y 60 estaciones en tierra. Durante este año, se agregarán dos satélites al sistema, el Metop B (ESA/EUMETSAT) y el Saral (ISRO, India). A diferencia del sistema argentino, los satélites del sistema brasileño (actualmente SCD1, SCD2 y CBERS-2B) poseen un transponder, que retransmite la señal recibida desde las plataformas a tierra, para su posterior procesamiento. La desventaja de este tipo de diseño se encuentra en que, para recibir a una determinada plataforma, el satélite además deberá estar a la vista de alguna estación terrena del sistema. Por otra parte, posibilita la adaptación continua de los algoritmos de procesamiento de señal. Este sistema posee en la actualidad alrededor de 1000 plataformas propias.

El sistema nacional está integrado por tres segmentos: el espacial, las plataformas y el segmento terrestre. El segmento espacial lo conforman todos los componentes que están a bordo de los satélites como es el receptor y su correspondiente antena. Actualmente solamente el satélite SAC-D de la CONAE, lleva un receptor del sistema. Las DCPs transmiten un mensaje a intervalos prefijados, sin ningún tipo de interrogación por parte del satélite. Todas las DCP pueden acceder al canal sin ningún tipo de coordinación entre sí. Teniendo en cuenta que la tasa de bit es de 400 bps, que el largo de los datos de ciencia varía entre 32 bits (mínimo) y 256 bits (máximo) y que además se transmite un protocolo de inicialización para sincronización e identificación, la duración del mensaje transmitido varía entre 360 y 920 ms.

Finalmente, el segmento terrestre lo componen las estaciones terrenas que reciben, procesan y distribuyen los datos a los usuarios.

En las siguientes secciones se analiza la implementación del receptor, haciendo incapié en las etapas de procesamiento digital implementadas en un DSP, se enumeran algunas de las pruebas realizadas y se muestran los primeros datos recibidos.

II. EL RECEPTOR DCS

La Fig. 1 nos muestra un diagrama en bloques general del receptor. El receptor DCS del SAC-D, puede recibir en la banda de los 401 a 403 MHz, en uno de 200 canales

separados 10 kHz entre sí. De esta manera, el receptor puede captar las transmisiones de plataformas del sistema Argos I y del SINDA. La frecuencia nominal del sistema argentino es 401.550 MHz. El pre-procesamiento de los mensajes se realiza en vuelo, lo que permite el acceso a plataformas alejadas del territorio nacional, aún cuando el satélite no esté a la vista de la estación terrena. La etapa de radiofrecuencia recibe la señal captada por la antena del sistema, que apunta a la superficie terrestre, la filtra, la amplifica, y la traslada a frecuencia intermedia, para finalmente obtener sus componentes en fase y cuadratura. Ambas señales, entonces, son muestreadas para ser procesadas en el DSP. A partir de las muestras, la Unidad

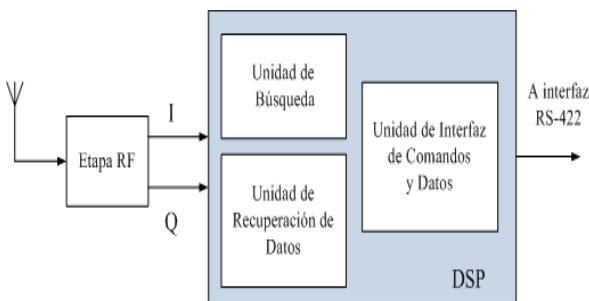


Fig. 1. Diagrama en bloques de las unidades funcionales que componen al receptor .

de Búsqueda (SU: Search Unit) es la encargada de detectar la transmisión de alguna DCP y estimar la frecuencia y amplitud de la portadora recibida. Debido a que todas las DCP transmiten en la misma frecuencia a intervalos regulares, es posible que se produzcan colisiones entre mensajes. La Unidad de Recuperación de Datos (DRU: Data Recovery Unit) es la encargada de adquirir la portadora, recuperar el reloj de bit y encontrar el sincronismo de trama, de manera de obtener el mensaje transmitido. La estructura del mensaje enviado por las DCP puede observarse en la Fig. 2.

La modulación utilizada por el sistema, es PSK binario donde la fase de la portadora se modula en $\pm 1,1$ rad. Los datos se transmiten en la componente en cuadratura, mientras que la componente en fase se corresponde con la portadora (portadora residual). La forma del pulso es Manchester (biphase-L) [1]. Finalmente, la Unidad de Interfaz de Comandos y Datos, permite controlar el estado del receptor por medio de comandos enviados desde tierra y descargar los mensajes recuperados y almacenados por el receptor.

Para realizar las tareas de procesamiento digital de la señal seleccionamos un DSP de Analog Devices. El ADSP21060 SHARC (Super Harvard Architecture Computer) es un procesador de 32 bits de alta performance y pertenece a la familia del 21020 que posee versión endurecida a la radiación. El mismo tiene un ciclo de instrucción de 25 ns y opera a 40 MIPS. El procesador puede ejecutar cada instrucción en un solo ciclo permitiendo realizar una FFT real de 1024 muestras en aproximadamente 0,46 ms. El procesador central consiste en tres unidades computacionales, un secuenciador de programa, dos generadores de direcciones, un temporizador, el cache de instrucciones y un conjunto de registros de propósito general. Las tres unidades computacionales son: una ALU

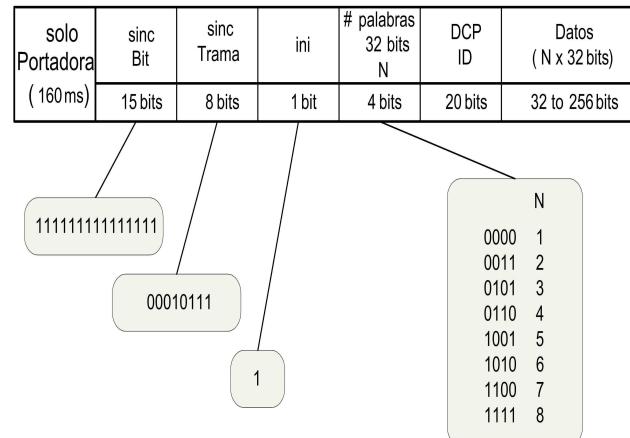


Fig. 2. Estructura del mensaje transmitido por las plataformas.

(Arithmetic Logic Unit), un multiplicador con acumulador de punto fijo y un registro de desplazamiento. Estas unidades son independientes entre sí y pueden procesar datos en tres diferentes formatos: punto fijo de 32 bits y punto flotante de 32 (precisión simple) o 40 bits (precisión extendida). El temporizador programable permite generar una interrupción en forma periódica, la que se utiliza en el muestreo de las señales. El ADSP21060 posee 4 Mb de memoria interna SRAM organizada en dos bancos de 2 Mb cada uno. Mediante la interfaz de puerto externo, el ADSP21060 es capaz de direccionar hasta 4 Giga palabras de memoria en el espacio unificado de memoria, que incluye tanto a los dispositivos externos, en nuestro caso, conversores analógico-digital, conversor digital-analógico, UART y EEPROM. El programa a ejecutarse reside en una memoria EEPROM externa y es descargado a la memoria interna del ADSP durante el encendido. La Fig. 3 muestra la interconexión entre el DSP y los periféricos utilizados.

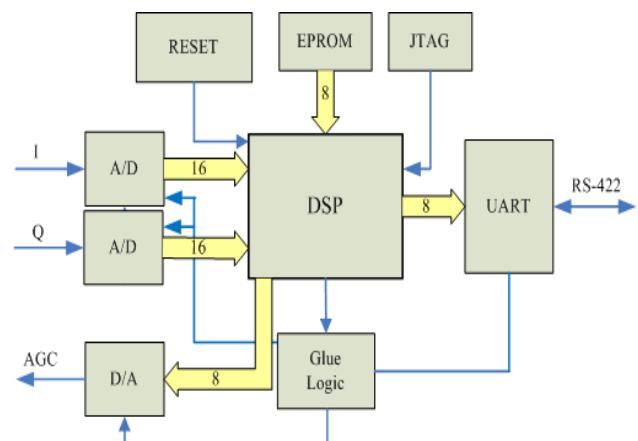


Fig. 3. Conexionado entre el DSP y los periféricos.

La mayoría de los componentes del sistema, fueron adquiridos en su versión resistente a la radiación y al vacío.



III. UNIDADES IMPLEMENTADAS EN EL DSP

A. Unidad de Búsqueda

Dado que al comienzo de cada mensaje se transmite solo portadora (durante los primeros 160 ms), el problema que resuelve la SU consiste en detectar una señal sinusoidal (modelo de señal determinístico) inmersa en ruido, que se modela como aditivo, blanco y gaussiano (RABG). Los parámetros de la señal (amplitud, frecuencia, fase inicial y tiempo de arribo) son desconocidos, lo que resulta en una degradación respecto al filtro adaptado (detector óptimo) [5]. Entre las dos principales líneas de acción para la resolución de este problema se encuentra el test generalizado de máxima verosimilitud (GLRT: Generalized likelihood Ratio Test) y el de Bayes (Bayesian). En nuestro caso se optó por el primero, en donde los parámetros desconocidos se modelizan como determinísticos. Para obtener el detector, subóptimo en este caso, deberemos realizar el siguiente test de hipótesis:

$$\mathcal{H}_0 : \quad x[n] = \omega[n] \quad \text{si } n = 0, \dots, N - 1$$

$$\mathcal{H}_1 : \quad x[n] = \begin{cases} \omega[n] & \text{si } n = 0, \dots, n_0 - 1 \\ A \cos(2\pi f_0 n + \phi) + \omega[n] & \text{si } n = n_0, \dots, N - 1 \end{cases}$$

donde $n \in \mathbb{Z}$, $\omega[n]$ es ruido blanco y gaussiano de media nula y varianza σ^2 , n_0 el número de muestra a partir del cual la señal está presente y N el número total de muestras analizadas. Tanto la amplitud de la señal A , como su frecuencia f_0 y su fase inicial ϕ , son desconocidas y, como dijimos anteriormente, se modelizan como parámetros determinísticos. Como resultado se obtiene que deberemos optar por hipótesis \mathcal{H}_1 : señal presente si [5]:

$$\max_{f_0, n_0} \frac{1}{N} \left| \sum_{n=n_0}^{n_0+N-1} x[n] \exp(-j2\pi f_0 n) \right|^2 > \gamma . \quad (1)$$

donde γ es el umbral de decisión dado. El problema es que es imposible su implementación en el dominio discreto ya que f_0 pertenece a los reales. Considerando $n_0 = 0$, una aproximación a este test se implementó en el DSP, por medio de la transformada discreta de Fourier (TDF). Es decir que optamos por \mathcal{H}_1 si:

$$\max_k \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] \exp(-j2\pi \frac{k}{N} n) \right|^2 > \gamma . \quad (2)$$

donde N es el largo de la FFT y $k \in \{0, 1, 2, \dots, N/2 - 1\}$. Debe tenerse en cuenta que hay una pérdida en el desempeño del detector, cuando la frecuencia de la señal f_0 no es un múltiplo de $1/N$. En nuestra implementación, dada la aplicación, no estamos interesados en estimar el valor del retardo n_0 , por lo que solo evaluamos para distintas frecuencias.

Una vez detectado un pico que supere el umbral, el valor de la frecuencia obtenida es el estimado de máxima verosimilitud de la señal recibida, si la frecuencia de la señal coincide con un bin de la TDF [6]. Con el valor de frecuencia encontrado, se inicializa el NCO del lazo de portadora, de manera que el error de frecuencia inicial sea pequeño. Esto disminuye el

tiempo de adquisición del mismo. Luego de detectada la señal de alguna DCP, se normalizan las muestras de la señal a la entrada del lazo de portadora, con la amplitud estimada. Esto último para evitar la modificación de los parámetros del lazo. Mientras no se detecta ninguna señal, la información se utiliza para fijar el control automático de ganancia de la etapa de radiofrecuencia. Debido a que muy probablemente, durante la detección, la señal no esté presente durante el tiempo en que se realizan las N observaciones, se decide por la hipótesis \mathcal{H}_1 si en dos salidas del espectro de potencia contiguas, se detecta un pico que supere el umbral, y además, si su valor de frecuencia estimada no difiere en más de un *bin* respecto a la frecuencia del primero. Esto último significa que la diferencia entre sus frecuencias no supere f_m/N [Hz], siendo f_m la frecuencia de muestreo. Consideraremos estacionaria a la señal durante el tiempo en que se calcula el espectro de potencia.

El procesador almacena las muestras de la señal y permanece en estado de espera. Cuando recolecta $N = 1024$ muestras, calcula el módulo cuadrado de su transformada discreta de Fourier, sin perder ninguna muestra de la señal de entrada. Obtenido un pico válido (el mayor que supere el umbral de detección) se anota su frecuencia y se espera que en el próximo bloque de datos el pico sea consistente. Hallado el segundo pico que cumple con la condición, se estima la amplitud de la señal \hat{A} y su frecuencia.

El estimado de la amplitud utilizando la FFT varía con la frecuencia f_0 de la señal, y con ello entonces el comportamiento del detector. La pérdida mayor se produce cuando el valor de la frecuencia f_0 se corresponde al valor intermedio entre dos *bins*. Para mitigar este efecto sin hacer ningún tipo de modificación de las muestras de la señal de entrada, se estima la amplitud de la señal de la siguiente manera:

$$\hat{A} = \frac{2 \sqrt{P_{i-1} + P_i + P_{i+1}}}{N} . \quad (3)$$

donde P_i es el valor del valor pico del espectro de potencia y $P_{i\pm 1}$ el valor de sus vecinos.

B. Unidad de Recuperación de Datos

1) *Lazo de Portadora*: Luego de la etapa de detección, lo que sigue es sincronizarse con la portadora y sincronizarse con el símbolo y la trama para, finalmente, recuperar el mensaje. La etapa de detección entrega un estimado de la frecuencia de la portadora, para que el error de frecuencia inicial del lazo sea a lo sumo 15,625Hz. El detector de fase implementado utiliza como entrada las señales en fase y cuadratura y en su salida se obtiene una señal que es función del coseno del error de fase, sin agregar el término de doble frecuencia de un detector multiplicativo. Esto es una ventaja dado que debido al efecto Doppler, la portadora recibida puede encontrarse en cualquier lugar dentro de una banda, en nuestro caso, de 32kHz, si esta frecuencia es muy baja, la componente de doble frecuencia también lo será y no podrá ser eliminada por el filtro de lazo.

El NCO (numerically controlled oscillator) se implementó mediante una tabla de 256 palabras de 8 bits (de donde se extraen las muestras del seno y el coseno). La inicialización del mismo se realiza a partir de la frecuencia estimada por la etapa anterior. Se utilizaron como constantes del lazo un

factor de amortiguamiento $\xi = 0.55$ y una frecuencia natural $\omega_0 = 250$ rad/s.

La Fig. 4 nos muestra la señal de salida del detector de fase, para un error inicial de 17 Hz y una relación señal a ruido de 10 dB en 400 Hz.

2) *Sincronismo de Símbolo y de Trama:* Como sincronizador de bit se utiliza un esquema denominado SCCL (Sample Correlate Choose Largest) [7]. El sincronizador óptimo realizaría un test de hipótesis, utilizando el criterio de máximo a posteriori (MAP) sobre las $L=80$ hipótesis (80 muestras por bit en nuestro caso, es decir, 80 posibles posiciones para el bit). En el DSP se implementó una versión sub-óptima, considerando solo tres hipótesis (*reloj de bit atrasado, a tiempo y adelantado*).

El algoritmo utiliza la salida del filtro adaptado (FA) en lugar de utilizar un correlador. Lo que hace el DSP es medir el valor absoluto de la salida del filtro cada 40 muestras (medio tiempo de bit). Se toman tres valores, el actual, el anterior y el posterior. Dado que el preámbulo incluye 15 bits cuyo valor es uno, como muestra la Fig. 2, la salida del filtro adaptado presentará picos cada medio tiempo de bit, con lo que obtenemos información acerca del estado actual del reloj de bit. La ecuación en diferencias recursiva del FA resulta:

$$y[k] = y[k - 1] - x[k] + 2x[k - N/2] - x[k - N] . \quad (4)$$

donde $y[k]$ y $x[k]$ son la salida y entrada al filtro respectivamente. Cada $L/2$ ciclos de reloj, se observa el valor actual del módulo de la salida del FA, $|y[k]|$ y además el anterior $|y[k - 1]|$ y posterior $|y[k + 1]|$. Dependiendo de cual de ellos resulta mayor, no se modifica, se adelanta en uno o se atrasa en uno el reloj de bit. El sincronizador hará que la fase del reloj se mueva de a un estado por vez, por lo que en ausencia de ruido, necesitaremos un máximo de $L/2$ pasos para llegar al estado correcto (el que produce el menor error en el reloj de bit). Puede ser que el estado uno, no coincida

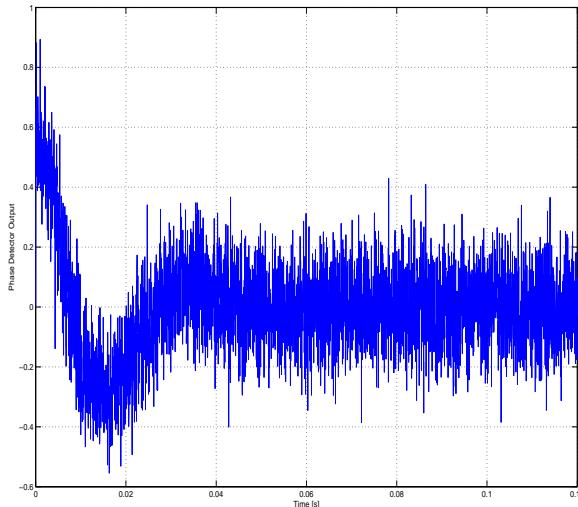


Fig. 4. Señal de salida del detector de fase, para un error inicial de 17Hz.

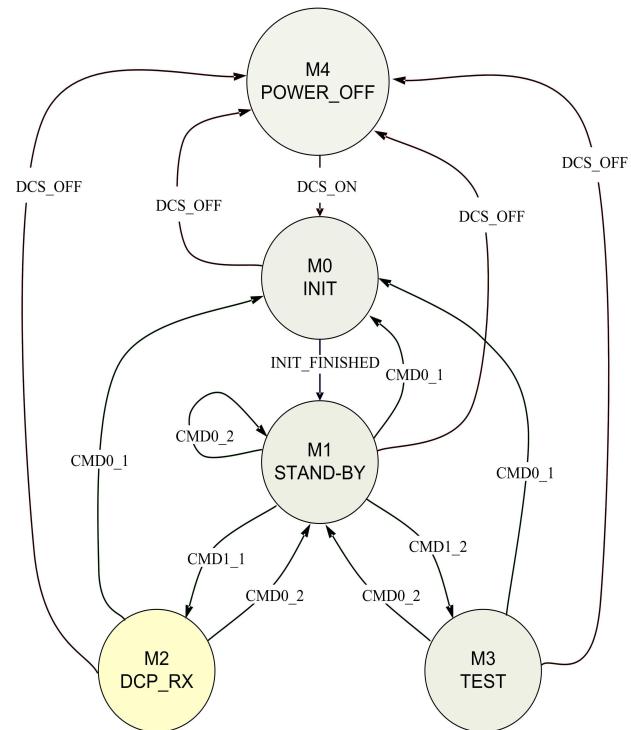


Fig. 5. Máquina de estados implementada en el ADSP21060 y los diferentes modos de funcionamiento del receptor.

con el instante exacto del comienzo del bit, dado que se toman muestras de una correlación continua.

Se debe recordar que existe una ambigüedad de π en la fase de la señal recibida. Esto se soluciona mediante el sincronismo de trama. Para ello, el DSP correlaciona los datos recibidos en el estado 1 y el 40 con los de una secuencia conocida que se transmite en el preámbulo. La secuencia a comparar se forma con el último uno de la secuencia de quince, los ocho bits siguientes y el uno correspondiente a la inicialización ('1000101111'). Adquirido el sincronismo de trama, sabemos en qué lugar dentro del mensaje nos encontramos y a continuación se extrae el largo del mensaje, la identificación de la plataforma y por último los datos. La finalización se da cuando se obtiene el último bit de datos (proceso de cuenta). Luego de adquirida la trama y mientras se extraen los datos, se sigue corrigiendo el reloj de bit, pero ahora, se mide el módulo de la salida del FA al comienzo del bit y se corrige en la muestra correspondiente a la mitad del bit.

C. Unidad de Interfaz de Comando y Datos

Esta unidad se implementa como una máquina de estados, que permite mediante comandos externos e internos, establecer el modo de operación del receptor. En la Fig. 5 se muestran los distintos modos de funcionamiento y los comandos permitidos en cada uno de ellos. Luego del encendido del sistema, el receptor pasa automáticamente al Modo M0 INIT de inicialización, para continuar en el modo M1 STAND-BY en donde es posible fijar la frecuencia de recepción y pasar a los modos

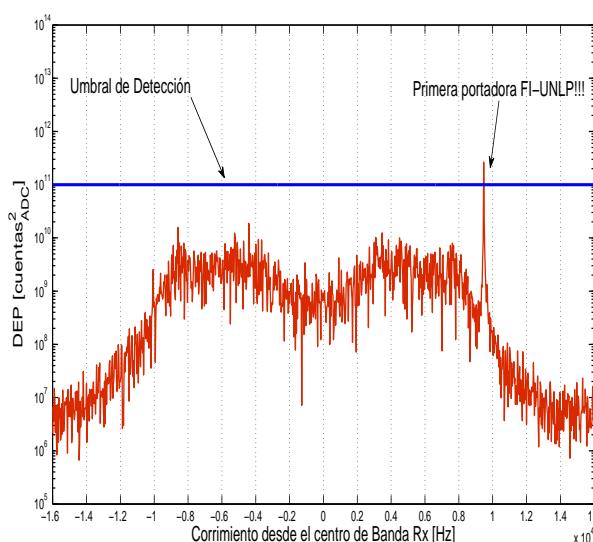


Fig. 6. Primera portadora detectada.

de operación principales M2 DCP RX de recepción y M3 de TEST.

IV. PRUEBAS EN TIERRA

Luego de ensamblado el receptor en la sala limpia de la FI-UNLP, se realizaron los test de vibración en el laboratorio del GEMA (Grupo de Ensayos Mecánicos Aplicados - FI-UNLP). Concluido este test se trasladó el equipo a ETC, donde se realizaron las pruebas de termo-vacío y EMI-EMC (solo el receptor). En INVAP, Bariloche se concreta la integración a la plataforma de servicio, junto con los demás instrumentos. Allí se realizan varios test funcionales, como por ejemplo, un día en la vida. A mediados del año 2010, se lleva el satélite a Laboratorio de Integración y Test, del INPE en Sao Jose dos Campos, Brasil, donde se realiza ya a nivel satélite, los test ambientales. Pasados estos últimos con éxito, se traslada el satélite a la base de la fuerza aérea de Vandenberg en Lompoc, EEUU. Previamente y con posterioridad a su integración al lanzador (Delta II de United Launch Alliance), se realizaron las pruebas funcionales correspondientes. Finalmente, el 10 de junio de 2011, se produjo la puesta en órbita del SAC-D.

V. PRIMEROS DATOS EN VUELO

Pasada la etapa de adquisición de órbita y estabilización, el 31 de agosto de 2011, a las 10:42 hs. UTC, se encendió con éxito el receptor DCS. El comportamiento térmico en vuelo fue el esperado, obteniéndose una sobreelevación de temperatura de alrededor de 3 grados Celsius en el DSP, en comparación con la temperatura de la base del receptor, por donde el calor se disipa a la plataforma. Durante las primeras órbitas, se adquirieron muestras de ruido en la banda de recepción. Luego se transmitió solo portadora desde la DCP patrón en la UNLP. Esta fue detectada en el satélite Fig. 6. Luego se transmitieron datos en la frecuencia nominal,

obteniéndose 276 mensajes recibidos en la pasada del 3 de setiembre de 2011, Fig. 7. Se puede observar el corrimiento por efecto Doppler de la frecuencia de portadora recibida en el satélite. Los valores positivos representan las transmisiones recibidas mientras disminuía la distancia entre la DCP y el receptor. En la actualidad, nos encontramos recibiendo transmisiones de plataformas pertenecientes a los sistemas SINDA y Argos sobre Argentina y Brasil.

VI. CONCLUSIONES

Durante el desarrollo del trabajo se introdujo al Sistema Argentino Satelital de Recolección de Datos Ambientales. Se mostró el diseño del receptor ubicado en el satélite, haciendo énfasis en las etapas de procesamiento digital. Se enumeraron algunos de los ensayos que el receptor tuvo que superar y se mostraron los primeros datos obtenidos en vuelo. El comportamiento del receptor es acorde a lo esperado, siendo su mayor ventaja su alta sensibilidad, lo que permite a las plataformas en tierra transmitir con baja potencia (menor a 30 dBm) permitiendo así su instalación en sitios remotos con alimentación autónoma.

VII. AGRADECIMIENTOS

El presente trabajo fue financiado por un convenio entre Conae y la Facultad de Ingeniería de la Universidad Nacional de La Plata. Los autores queremos agradecer a todas las personas involucradas en la misión SAC-D/Aquarius, las que con mucho empeño, lograron realizar con éxito todas las tareas que culminaron con la puesta en órbita del satélite. En especial al fundador del GrIDComD, Ing. Hugo Lorente, a dos de sus primeros integrantes: Dr. Pablo Costanzo Caso y Dr. Laureano Bulus Rossini y al Ing. Juan Ignacio Fernández Michelli por su necesaria participación.

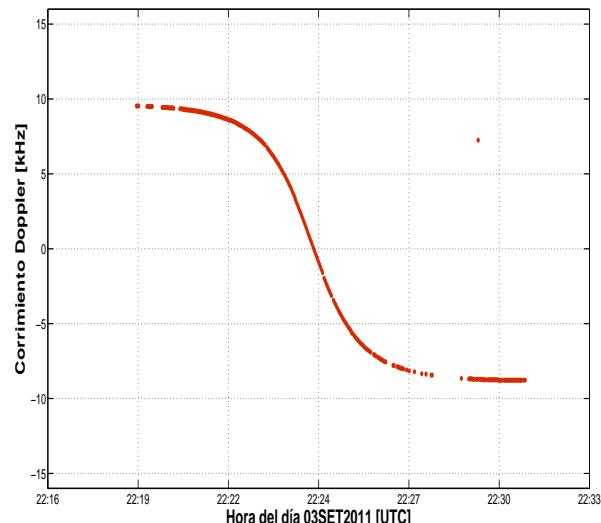


Fig. 7. Variación Doppler de la frecuencia de portadora recibida de 276 mensajes durante una pasada.



REFERENCIAS

- [1] Carlotto A., Lorente H.: *Análisis de un Sistema de Recolección de Datos Satelital*. Memorias de AADECA 2006, XX Congreso Argentino de Control Automático, Buenos Aires, Argentina, 28-30 de agosto 2006. ISBN 978-950-99994-4-2. (2006)
- [2] Clark, D. *Overview of the Argos System*, ; OCEANS '89 Proceedings, 1989. Volume: 3, Page(s): 934 - 939.
- [3] Ortega C. *Argos second and third generations: enhancements finely tuned to oceanographic applications*, OCEANS '98 Conference Proceedings , 1998 , Page(s): 845 - 848 vol.2.
- [4] Yamaguti W., et al. *Sistema Brasileiro de Coleta de Dados Ambientais: Status e planos futuros*, Anais XIV Simpósio Brasileiro de Sensoriamento Remoto, Natal, Brasil, 25-30 abril 2009, INPE, p. 1633-1640.
- [5] Kay, Steven M.: Fundamentals of Statistical Signal Processing: Detection Theory. Prentice-Hall PTR. ISBN 0-13-504135-X (1998).
- [6] Kay, Steven M.: Fundamentals of Statistical Signal Processing: Estimation Theory. Prentice-Hall PTR. ISBN 0-13-345711-7 (1993).
- [7] Chen, Kwang-Cheng (1992). *Analysis of a New Bit Tracking Loop—SCCL*, IEEE Trans. on Communications, Vol.40, No.1.



Foro Tecnológico

Pósters

DSPs





CODEC DE AUDIO PARA SISTEMA DE BAJOS RECURSOS DISEÑO, ANÁLISIS E IMPLEMENTACIÓN EN MATLAB® Y DSPIC

Esp. Ing. Gabriel R. Caballero.
Departamento Telecomunicaciones, Área I+D.
Tratamiento Digital de Señales
Universidad Blas Pascal
Córdoba, Argentina
Email: gcaballero@invap.com.ar

Ariel F. Reynoso, Carlos N. Liendo
Tratamiento Digital de Señales
Universidad Blas Pascal
Córdoba, Argentina
Emails: areynoso@gmail.com,
carlosnicolasliendo@gmail.com

Resumen: En este artículo se desarrolla un *CODEC* para compresión de audio con pérdida, para sistemas de bajos recursos, basado en el modelo psicoacústico y las propiedades de las transformadas de tiempo discreto. Este método permite obtener resultados que se aproximan al 80% de compresión con una calidad de audio definida por un MOS mayor o igual a 3. El algoritmo de compresión esta diseñado para su implementación en un sistema de bajos recursos basado en un *DsPIC*.

Palabras Claves: DSP, Compresión, FFT, DCT, Codificación de Huffman, MOS, Relación de Compresión.

1. INTRODUCCIÓN

Actualmente en el mercado se dispone de muchas soluciones encapsuladas y/o plataformas propietarias para la compresión /descompresión de audio dificultando su estudio e imposibilitando la implementación sin dependencia de módulos externos. Si bien existen soluciones libres las mismas apuntan a plataformas de gran rendimiento, las que quizás exceden las necesidades de algunas aplicaciones en particular. En este proyecto inicialmente se desarrolla el compresor en Matlab® para analizar su desempeño (radio de compresión y calidad) y posteriormente se verifica la factibilidad de su implementación en un sistema de bajos recursos.

1.1 Codificación

La codificación busca reducir el número de bits utilizados para almacenar o transmitir una información dada. En general consta de tomar una serie de símbolos y transformarlos en códigos. Si la codificación es efectiva, los códigos resultantes tendrán un tamaño menor que los símbolos originales. Existen varias técnicas de compresión, una de las más eficientes es la codificación aritmética pero requiere de un gran poder de cómputo. La codificación de Huffman es la más utilizada, por tratarse de un algoritmo simple que en comparación solo pierde un 6% de efectividad en relación con la anterior.

El propósito de este codificador es eliminar la redundancia, mientras más redundante sea la señal de entrada, mayor será el coeficiente de compresión logrado. [1]

El concepto de compresión queda ilustrado en el siguiente ejemplo. Sean los símbolos de entrada con su frecuencia de ocurrencia relativa:

Tabla 1. Frecuencia de símbolos.

Símbolo	Frecuencia
A	24
B	12
C	10
D	8
E	8

En total son 186 bits, considerando 3 bits por símbolo. Aplicando el algoritmo de Huffman se genera el árbol que se muestra en la figura 1.

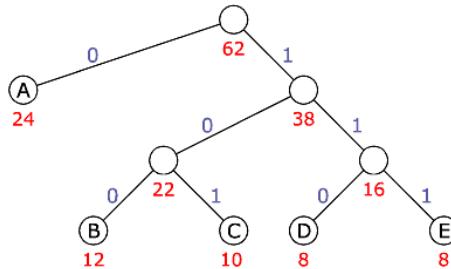


Fig. 1: Árbol de la codificación.

Del árbol puede construirse el diccionario que se utiliza en la codificación, resultando:

Tabla 2. Diccionario de Huffman

Símbolo	Frecuencia	Código	Bits	Total
A	24	0	1	24
B	12	100	3	36
C	10	101	3	30
D	8	110	3	24
E	8	111	3	24

El total ahora es de 138 bits, logrando una compresión del 25,8% y se puede ver como los símbolos más frecuentes se almacenan con menos bits y la redundancia se elimina. Considerando esta propiedad, se aplican las siguientes técnicas para que la codificación de Huffman sea efectiva: El modelo psicoacústico (PA) y la transformada discreta del coseno.



1.2 Modelo psicoacústico

El modelo psicoacústico [2] involucra una gran cantidad de conceptos aplicables a la compresión. Se hace uso de los siguientes:

Respuesta en Frecuencias del oído Humano

El modelo determina que el oído humano se comporta como un filtro pasa banda entre 20Hz y 20Khz con mayor sensibilidad a las frecuencias entre 2 y 4Khz. Esto define la tasa de muestreo para el procesamiento digital. Según el teorema de Nyquist:

$$\begin{aligned} f_s &= 2f_{\max} \\ f_s &= 2 \times 20000 \text{ Hz} = 40000 \text{ Hz} \end{aligned} \quad (1)$$

Siendo que la frecuencia de Nyquist es un valor ideal y debido a la imperfección de las técnicas de recuperación, se adopta 44100Hz, como frecuencia de muestreo a ser utilizada por el presente CODEC.

Enmascaramiento simultáneo

El estudio del modelo PA, además determina que la percepción del oído humano puede dividirse en 24 bandas de frecuencias. Dentro de cada banda es muy difícil que el oído pueda determinar con precisión la ubicación exacta de un sonido particular. [3]

Si existe una componente dominante, esta introducirá un efecto de enmascaramiento a todas las demás frecuencias dentro de la misma banda, el objetivo es llevar a cero las demás componentes imperceptibles y generar redundancia sin perder calidad. [4]

1.3 Análisis en frecuencia

La representación de una función periódica en el dominio de la frecuencia requiere para su almacenamiento menos coeficientes que en el dominio temporal, así mismo, descartar valores en el dominio de la frecuencia introduce una distorsión menor que descartar valores en el dominio temporal. Debido a que las señales de audio son de naturaleza periódica, es conveniente tratar las mismas en el dominio de la frecuencia. Una transformada que reúne estas características es la Transformada de Fourier que utiliza como base series de senos y cosenos para la representación en amplitud y fase de cada una de las componentes espectrales de la señal de entrada. En este caso el tratamiento es digital y se utiliza la transformada discreta de Fourier (DFT), la cual transforma una secuencia de valores a su representación en frecuencia mediante la expresión (2).

$$X_K = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n} \quad (2)$$

Estos coeficientes pueden calcularse mediante el algoritmo FFT (Fast Fourier Transform por sus siglas en inglés)

Si bien esta transformada es útil para realizar el análisis del espectro, en este trabajo se utiliza la Transformada Discreta del Coseno DCT [5] (Discrete Cosine Transform por sus

siglas en Inglés) que es una variación de la DFT pero utiliza cosenos como base. Donde sus coeficientes están dados por:

$$X_K = \sum_{n=0}^{N-1} x_n \cos \left[\frac{\pi}{N} \left(n + \frac{1}{2} \right) k \right] \quad (3)$$

$$k = 0, \dots, N-1$$

Su principal ventaja con respecto a la DFT es que luego de aplicar la transformada, la energía se concentra en grupos reducidos de coeficientes. Esto permite llevar a cero los demás valores que se han tornado irrelevantes en comparación a aquellos donde se encuentra concentrada la energía. Por otro lado se pueden cuantificar los valores obtenidos de tal manera que exista mayor probabilidad de que se repitan valores. De esta forma se genera redundancia y al aplicar la transformada inversa, la señal se recupera sin cambios perceptibles. [6]

En las *figuras 2 y 3* se puede apreciar cómo, para la misma entrada, la DFT tiene baja redundancia ya que todos los coeficientes son distintos, mientras que en la DCT solo hay 4 valores distintos y se aprecia la redundancia del valor cero que se repite 7 veces.

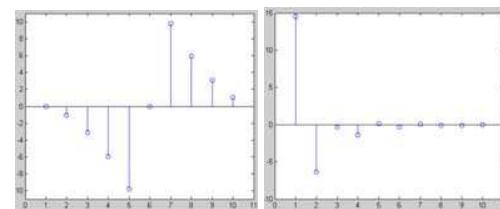


Fig. 2 Transformada DFT

Fig. 3 Transformada DCT

1.4 Objetivo

El objetivo de este trabajo es desarrollar un CODEC de audio mediante procesos matemáticos simples y mediante un análisis de recursos computacionales demostrar la factibilidad de implementación en un sistema de bajos recursos.

2. METODOLOGIA

Se describe el diagrama en bloques del sistema de compresión que es utilizado como guía para el desarrollo del CODEC. Luego se analiza cómo afectan los diversos parámetros del CODEC en el coeficiente de compresión y su relación con la calidad final, finalmente se diseña el descomprimidor para una plataforma dsPIC para comprobar su desempeño.

2.1 Variables

Las siguientes variables están involucradas en el sistema y se utilizan para el análisis del comportamiento del CODEC:

- Señal de entrada (bytes)
- Valoración de entrada (adimensional)
- Umbral de decisión (%)
- Cuantificación (bits)
- Señal de salida (bytes)
- Valoración de salida (adimensional)
- Coeficiente de compresión
- Tiempo de cómputo (ms)

La valoración de entrada y salida está basada en MOS, el cual consiste en un test que permite obtener a partir del punto de vista del usuario la calidad de audio del sistema. Esta es una

medida subjetiva promedio dada por los oyentes en base a encuestas de puntuación en una escala del 1 al 5.
El resto de las variables son de observación directa.

2.2 Implementación del compresor

En la *figura 4* se muestra el diagrama del sistema de compresión.



Fig. 4 Diagrama del sistema de compresión

2.2.1 Señal de entrada

Entre los posibles formatos de entrada de audio se elije PCM, 16 bits, 44100Hz ya que conserva la calidad del audio analógico con mayor fidelidad. Esta no es una restricción pero se toma como punto de partida.

En Matlab:

```
[oy,Fs,nbits]=wavread('filename');
```

Donde *oy* devuelve la señal de entrada muestreada, *Fs* la frecuencia de muestreo y *nbits* los bits por muestra.

2.2.2 Separación en frames

Debido a las limitaciones computacionales y de memoria, es difícil trabajar con la señal de audio completa en tiempo real, entonces es necesario tomar pequeños fragmentos que son procesados individualmente y llamarlos frames. Este sistema trabaja con frames de 26ms, que es un valor típico en las aplicaciones de audio y teniendo en cuenta la velocidad de muestreo, se adoptan 1024 muestras.

$$T_{frame} = 1024 \times \frac{1}{f_s} = 1024 \times 2,26 \mu s \quad (4)$$

$$T_{frame} = 23,21 ms$$

Puede apreciarse que el valor del tiempo del frame se approxima a los 26ms.

En Matlab:

```
Frame=oy(1+FrameSize*(FrameActual-1):FrameSize*FrameActual)
```

Tendremos en *Frame* la parte de *oy* correspondiente al *FrameActual*

2.2.3 Modelo psicoacústico

Para aplicar el concepto de enmascaramiento simultáneo se obtiene el espectro en frecuencia del frame mediante la FFT. Se debe tener en cuenta que antes de realizar la transformada es necesario aplicar un filtro de ventana para reducir los espúrios del espectro.

En Matlab:

```
Ventana=hamming(FrameSize);
FrameF=FFT(Frame.*Ventana);
```

Al espectro obtenido se lo divide en las bandas críticas, dentro de cada banda se determina si existe una componente, cuyo valor sea mayor a la media de dicha banda multiplicada por un factor de umbral.

En la *figura 5* se muestra un fragmento del espectro en donde se puede observar dos componentes, en la banda 3 y 4 respectivamente, que producen el efecto de enmascaramiento.

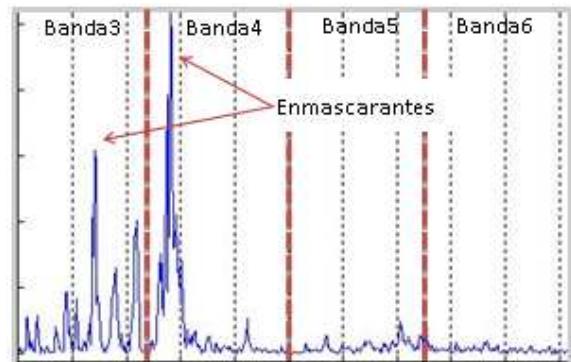


Figura 5. Fragmento del espectro

Una vez localizadas las componentes enmascarantes, se diseña un filtro pasa banda, selectivo en frecuencia para cada componente en particular, que será parte del banco de filtros que finalmente se aplicará al frame completo.

Es importante destacar que no se debe aplicar un filtro por banda sino que se debe filtrar el frame completo, incluso si no se detectase una componente predominante debe diseñarse un filtro que permita pasar toda la banda para que el retraso introducido sea uniforme en cualquier condición.

En Matlab:

```
Media=mean(abs(Banda));
[Maximo indice]=max(abs(Banda));
if (Maximo>Media*FactorUmbral)
  FiltroPB(indicebanda,indice);
else
  FiltroPlano(indicebanda);
end if
```

Donde *FiltroPB* es una función que toma un filtro prediseñado y lo centra en “índice” y *FiltroPlano* es un filtro prediseñado de ganancia unitaria que ocupa una banda entera. Estos filtros se irán sumando al banco de filtros.

Los filtros en Matlab® pueden ser implementados de diversas maneras dependiendo los requerimientos computacionales, una buena opción para este caso es utilizar la función *FilterPM*. [7] En la figura 6 se puede observar la respuesta en frecuencia del banco de filtros que sería utilizado para el ejemplo mostrado en la figura 5.

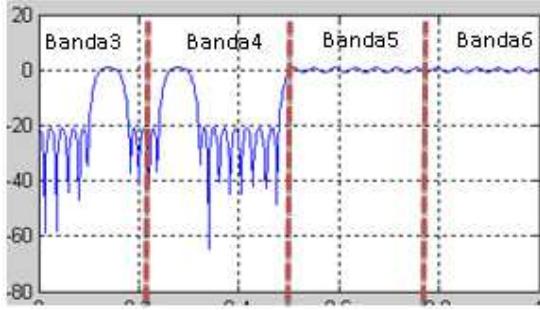


Figura 6. Diseño del filtro

2.2.4 Filtrado

Una vez construido el banco de filtros es necesario filtrar el frame, el cual es solo una porción del audio final, por lo tanto es necesario que el filtro tenga persistencia del estado de los frames filtrados en el pasado para eliminar el tiempo de estabilización requerido por los filtros. La función “*filter*” de Matlab® posee la opción de hacer “*Persist Memory*”.

En Matlab:

```
Hd = dfilt.dffir(FilterCoef);
Hd.PersistentMemory=true;
filter(Hd,Frame);
```

Donde *filterCoef* son los coeficientes del banco de filtros.

En la figura 7 se puede observar la señal filtrada luego de aplicar el filtro de la figura 5. Se puede apreciar el aumento de redundancia en las bandas 3 y 4.

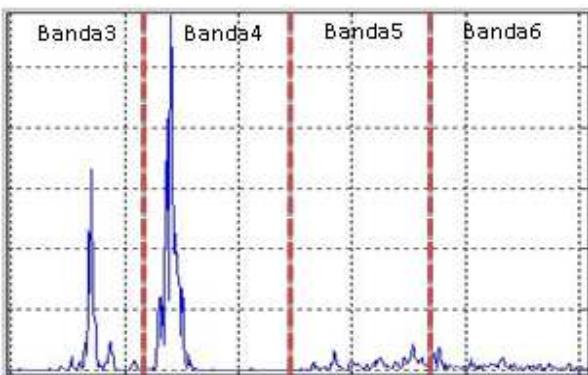


Figura 7. Señal filtrada

2.2.5 La DCT y la Cuantificación

Aplicar la DCT es una parte central del proceso, como fue mencionado anteriormente la misma concentra la energía y esto permite eliminar valores irrelevantes. Al resultado de la DCT se lo cuantifica en niveles correspondientes a la cantidad de bits elegidos para almacenar la información. Así mismo, se establece un umbral de decisión, por encima de dicho umbral se conservan los valores y por debajo se los iguala a cero.

En matlab:

```
fydct=dct(FrameFiltrado);
fydct=fydct/max(abs(fydct));
fydct=round(fydct*2^7);
fydct=fydct.*(abs(fydct)>max(abs(fydct))*Umbral);
al;
```

Donde *fydct* contiene el valor cuantificado, para aplicar el codificador de Huffman. *Umbral* es el porcentaje del pico máximo que se utiliza para descartar valores insignificantes.

Es importante tomar una relación de compromiso para buscar el número de bits a utilizar en la cuantificación para mantener la calidad y el coeficiente de compresión adecuado según la aplicación. En las pruebas se utilizan 8 bits, buscando aproximarse a las características de los sistemas de bajos recursos. [8]

En la figura 8 se observa que las amplitudes de la DCT son del tipo reales y el conjunto presenta baja redundancia.

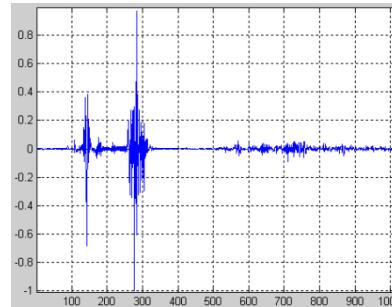


Fig 8. DCT sin cuantificar

En contraposición en la figura 9 se aprecia que gran parte de los coeficientes son iguales a cero, lo que genera redundancia y optimiza el proceso de codificación de Huffman.

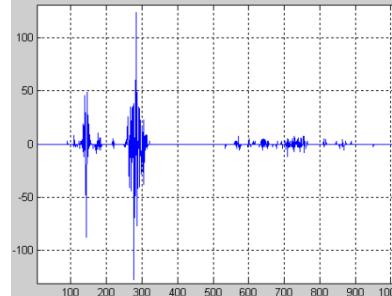


Figura 9. DCT cuantificada

2.2.6 Codificación de Huffman

Luego de aplicar los procesos anteriores, la codificación de Huffman es muy eficiente debido a la gran redundancia obtenida. A la salida del bloque codificador, es donde se puede apreciar el fenómeno de compresión, ya que se almacenan con menos bits los símbolos más frecuentes.

Matlab® no posee el algoritmo de Huffman para calcular el código y realizar la codificación. Se deberá implementar una función que como entrada reciba el frame procesado y como salida entregue el diccionario y los datos codificados.

2.2.7 Multiplexado

Después de aplicar el codificador de Huffman, se cuenta con la información codificada más su diccionario para ser recuperada, ahora es necesario escribir esta información en un formato que pueda ser interpretado por el receptor. Se implementa una estructura simple en donde cada frame está representado de la siguiente forma:

Tabla 3. Formato de la Información

Longitud (bytes)	Nombre	Detalle
1	Factor	Factor para desnormalizar los valores de la DCT
2	OSize	Tamaño original de la array sin comprimir
1	DicSize	Cantidad de entradas del diccionario
Variable	Dictionary	Par llave/valor correspondiente a cada entrada del diccionario
Variable	Huffman	Array de datos comprimidos

2.3 Implementación del descomprimidor

En la figura 10 se muestra el diagrama del sistema de descompresión:

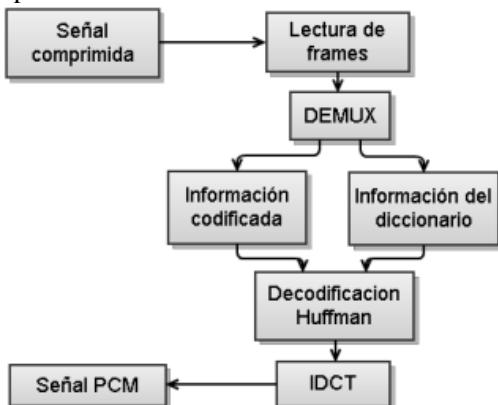


Fig 10. Diagrama del sistema de descompresión.

Este sistema está pensado para implementarse en dispositivos de bajos recursos, para las pruebas se utilizó un DsPIC de Microchip y se pretende evaluar la factibilidad del procesamiento en tiempo real.

Debido a la simetría en los procesos de compresión y descompresión en este trabajo solo se expone el pseudocódigo del descomprimidor.

En la figura 11 se muestra el diagrama lógico del código:

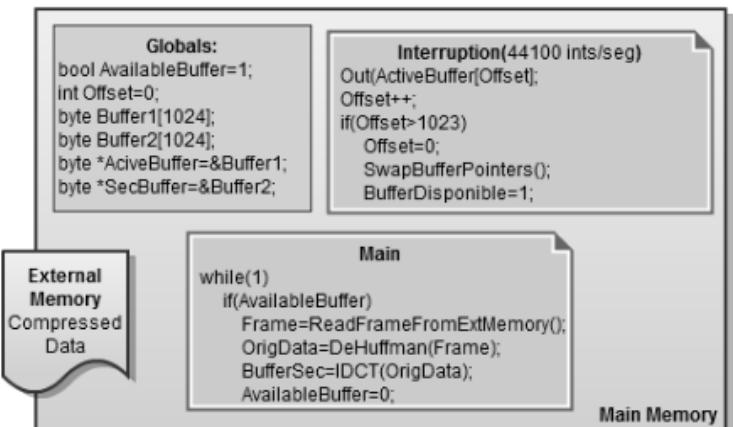


Fig 11. Diagrama lógico del código

2.3.1 Análisis de Factibilidad

Todas las pruebas se realizaron utilizando una plataforma de desarrollo Explorer 16 de Microchip, basada en un dsPIC33f256 con un cristal de 10MHz.

Se procede a analizar los tiempos de ejecución:

- Implementación de la DCT para 1024 muestras proporcionada por Microchip:
 - Tiempo requerido promedio: 6 ms
- Implementación de filtro FIR proporcionada por microchip (orden: 75) para 1024 muestras.
 - Tiempo requerido promedio: 9 ms
- Implementación propia del código Huffman más acceso a memoria escrita en C (sin optimizar):
 - Tiempo requerido promedio: 3.5 ms

En la descompresión, utilizando IDCT y Huffman se tiene un total de procesamiento de 9.5ms, tomando un margen de 2ms da como resultado 11.5ms.

Se obtiene un frame completo cada 23ms:

$$\frac{1}{44100 \text{Hz}} \times 1024 = 23 \text{ms} \quad (5)$$

Si el procesamiento toma 11.5ms por frame, se disponen: 23ms - 11.5ms = 11.5ms ociosos.

2.3.2 Conversión PCM-Analógico

La conversión D/A y su posterior filtrado se realizan mediante la placa Microchip Audio Pictail Daughter Card.

En la figura 12 se puede apreciar el setup de implementación del descomprimidor.

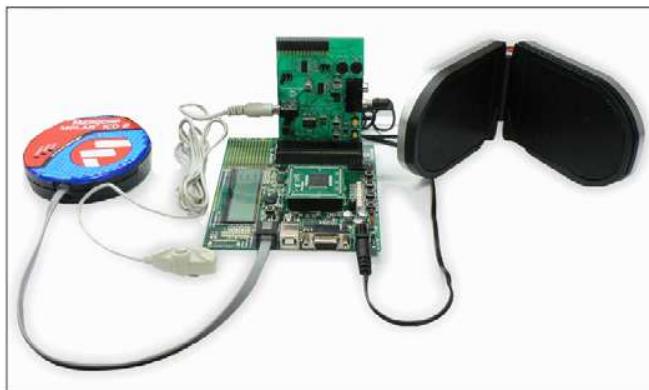


Fig 12. Setup de Implementación del Descompresor

3. RESULTADOS OBTENIDOS

Se utiliza un archivo de audio en formato FLAC (lossless-audio codec, 44100Hz, 16bit por muestra) para garantizar que el audio original no tenga ya alguna distorsión propia de los procesos de compresión. Se adopta una señal de entrada de 2Mbytes de extensión con una valoración de 5.

Pruebas:

Tabla 4. Resultados Obtenidos

#	Modelo PA	Umbral (%)	Bits de cuantif. (bits)	Salida (KB)	Val	Radio (%)
1	no	2	8	453,38	3,5	77,31
2	si	2	8	428,97	3,5	78,53
3	si	5	8	367,52	2,5	81,61
4	si	0	8	534,48	4	73,25
5	si	0	16	1753,3	4,5	14,3

A continuación se presentan las observaciones más relevantes obtenidas de las encuestas realizadas. Para obtener la valoración, en todos los casos, se contempla que la opinión es algo subjetivo, solo se anotan aquellas observaciones comunes a la mayoría de las encuestas realizadas.

1 y 2: Se puede ver que el Modelo Psicoacústico proporciona una ganancia del 1.22% de compresión pero no afecta a la valoración.

3: Se introduce ruido ajeno al sonido original y se aprecian distorsiones. Las componentes muy agudas y de baja intensidad desaparecen.

4: No se escuchan ciertas sutilezas del audio original pero no hay distorsión perceptible. Se preservan las componentes muy agudas de baja intensidad.

4. CONCLUSIONES

Tras la evaluación de los resultados se puede concluir que es posible desarrollar un CODEC para la

compresión/descompresión de audio con perdida, basándose en el modelo psicoacústico y las propiedades de las transformadas discretas de Fourier y del Coseno, para lograr una compresión de hasta el 78% conservando una calidad mayor o igual a una valoración de 3. Este proceso puede ser implementado mediante operaciones matemáticas que cualquier dispositivo DSP dispone, pudiendo incluso realizar el tratamiento de la señal en tiempo real. Sin embargo, debido a que el modelo psicoacústico repercute tan solo en un 1.22% del tamaño total, su factibilidad de implementación queda determinada en base a los recursos computacionales disponibles.

Queda pendiente someter el experimento a tests de hipótesis para demostrar la validez estadística de los resultados, así mismo queda pendiente realizar mejoras al algoritmo del modelo psicoacústico para aumentar su rendimiento.

5. BIBLIOGRAFIA

- [1] Mark Nelson, The Data Compression Book, University of Bahrain, College of Applied Studies
- [2] Robinson D., Hawksford M., Psychoacoustic Models and Non-Linear Human Hearing, The University of Essex, Centre for Audio Research and Engineering.
- [3] Howard C., Comparison of psychoacoustic Principles and genetic algorithms in audio compression, California State University at San Bernardino 2005.
- [4] Raissi R., The theory behind MP3, December 2002
- [5] Khayam S., The discrete cosine transform, Department of Electrical & Computer Engineering, March 2003.
- [6] Blinn J., What's the deal with the DCT?, California Institute of Technology, July 1993.
- [7] Losada R., Practical FIR filter design in Matlab®, Math Works, Inc., March 2003.
- [8] Blinn J., Quantization error and dithering, California Institute of Technology.



Artículos

FPGA y HDLs





Cancelación de Ruido Aplicando Estadística de Orden Superior y Sistemas Multiprocesador sobre FPGA

Miguel Enrique Iglesias Martínez

Dpto. de Investigación y Desarrollo

Centro de Desarrollo de la Electrónica y la Automática,

CDEA

Pinar del Río, Cuba

mgi@cdea.co.cu

Fidel E. Hernández Montero

Dpto. de Telecomunicaciones y Electrónica

Universidad de Pinar del Río, UPR

Pinar del Río, Cuba

fidel@tele.upr.edu.cu

Abstract—Las necesidades de cómputo de los sistemas electrónicos modernos aumentan cada día. Los sistemas multiprocesador en chip (*MPSoC*) son una buena opción para satisfacer las demandas de las aplicaciones modernas. En este trabajo se propone la utilización de sistemas multiprocesador sobre arquitecturas reconfigurables para la estimación de características estadísticas (cumulantes) de orden superior, y su aplicación en tareas de reducción o cancelación de ruido. Se aportan los fundamentos teóricos que soporta el trabajo realizado, que fue validado mediante el sistema digital implementado sobre el FPGA y con la utilización de señales simuladas (generadas en el entorno Matlab). Los resultados obtenidos, altamente satisfactorios, demuestran la efectividad del uso de sistemas multinúcleo con tecnología de hardware reconfigurable y la utilización de características estadísticas de orden superior, para cancelar ruido

Keywords-Cancelación de Ruido, Cumulantes, Multiprocesador, FPGA

I. INTRODUCCIÓN

Las fluctuaciones de un proceso de señal completamente aleatorio o la distribución de una clase de señales aleatorias en el espacio no pueden ser modeladas por una ecuación predictiva, pero pueden describirse en términos de los estadísticos de la señal y modelados por una función de distribución de probabilidad en un espacio de señal multidimensional [1]. Numerosos métodos han sido desarrollados buscando precisamente la extracción de señales perturbadoras algunos de estos se pueden agrupar de la siguiente manera:

- Métodos clásicos de cancelación de ruido, aplicados esencialmente en la rama de las comunicaciones eléctricas, cuando la señal deseada presenta un comportamiento primordialmente periódico.
- Métodos clásicos de procesamiento estadístico de señal.
- Métodos adaptativos de cancelación de ruido.

• Métodos de Inteligencia Artificial

En la actualidad los sistemas electrónicos modernos demandan cada vez mayor capacidad de cómputo. Por ejemplo, dispositivos capaces de ofrecer un buen rendimiento a velocidades de transferencia del orden de gigabits por segundo. Actualmente los FPGAs ofrecen gran cantidad de recursos al diseñador, millones de puertas lógicas equivalentes, bloques de memoria, bloques DSP, e incluso las más modernas incluyen uno o varios procesadores dentro del propio FPGA. Esto puede ser de gran utilidad en tareas de reducción o cancelación ruido a la hora de calcular la función o funciones correspondientes del proceso de señal contaminada la cuales pueden consumir excesivamente demasiando tiempo en su ejecución.

El presente trabajo muestra el uso de sistemas multiprocesador en tareas de reducción o cancelación de ruido como metodología para la aceleración del cómputo de funciones estadísticas de orden superior como los cumulantes de tercer y cuarto orden de la señal. En este trabajo se lleva a cabo la validación del diseño primeramente en la herramienta de simulación Matlab, empleando como muestras útiles, señales simuladas; posteriormente se realiza la verificación en un sistema para la reducción de ruido empleando hardware reconfigurable y sistemas multiprocesador.

El trabajo se ha organizado de la siguiente manera: Las secciones II y III, como parte del desarrollo del mismo, presentarán los aspectos generales sobre la estimación basada en estadísticos de orden superior de la señal, así mismo las secciones de la IV hasta la IX, presentarán además, las soluciones propuestas y los resultados obtenidos de la investigación conjuntamente con las plataformas de trabajo utilizadas, para finalmente exponer las conclusiones de la investigación y las referencias consultadas.

II. ESTADÍSTICA DE ORDEN SUPERIOR

De los métodos empleados para la eliminación del ruido y la interferencia, en esta sección se abordará uno basado en el cálculo de los estadísticos de orden superior de la señal. Se



tomará como señal útil un coseno de amplitud A , de frecuencia f_k y fase ϕ_k , el cual es afectado por ruido blanco gaussiano.

Existen varias motivaciones generales detrás del uso de la estadística de orden superior en el procesamiento de señales, dentro de las cuales destaca [1] [2]:

- Eliminar el ruido coloreado aditivo gaussiano de espectro de potencia desconocida.
- Identificar los sistemas de fase no mínima o reconstrucción de señales de fase no mínima.
- Extraer la información debido a las desviaciones de Gaussianidad.
- Detectar y caracterizar las propiedades no lineales de las señales, así como identificar los sistemas no lineales.

Sea entonces $\{X(k)\}, k = 0, \pm 1, \pm 2, \pm 3, \dots$ una señal real estacionaria discreta en el tiempo donde sus momentos de orden superior existen, luego:

$$m_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) = E\{X(k)X(k+\tau_1)\dots X(k+\tau_{n-1})\} \quad (1)$$

representa el momento de orden n de una señal estacionaria, el cual depende solo de los diferentes espacios de tiempo $\tau_1, \tau_2, \dots, \tau_{n-1}, \tau_i = 0, \pm 1, \dots$ para todo i . Claramente se puede apreciar que el momento de segundo orden $m_2^x(\tau_1)$ es la función de autocorrelación de $\{X(k)\}$, del mismo modo que $m_3^x(\tau_1, \tau_2)$ y $m_4^x(\tau_1, \tau_2, \tau_3)$ representan los momentos de tercer y cuarto órdenes, respectivamente. $E\{\cdot\}$ denota el operador esperanza matemática de la señal [2].

El cumulante de orden n de una señal aleatoria no gaussiana estacionaria, $\{X(k)\}$, puede escribirse solamente para $n = 3$ o $n = 4$ como:

$$C_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) = m_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) - m_n^G(\tau_1, \tau_2, \dots, \tau_{n-1}) \quad (2)$$

Donde $m_n^G(\tau_1, \tau_2, \dots, \tau_{n-1})$ es el momento de orden n de una señal gaussiana equivalente que presente el mismo valor medio y función de autocorrelación que $\{X(k)\}$. Es evidente que si es Gaussiana, $m_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) = m_n^G(\tau_1, \tau_2, \dots, \tau_{n-1})$, por tanto $C_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) = 0$. Obsérvese, que aunque la ecuación (1) es solo verdadera para $n=3$ y $n=4$, $C_n^x(\tau_1, \tau_2, \dots, \tau_{n-1}) = 0$, para todo n si $\{X(k)\}$ es gaussiano.

Como se ha comentado con anterioridad el cumulante de segundo orden de un proceso de ruido es distinto de cero. Por otra parte, de lo tratado en [3] y [4] se puede decir que los cumulantes de tercer orden se anulan para funciones cuya densidad probabilística sea simétrica, por tanto no tiene sentido su aplicación para cancelar el ruido en señales de tipo

cosinusoide. Por esta razón, se decide pasar a la estimación del cumulante de cuarto orden.

III. MÉTODO DE ESTIMACIÓN BASADO EN EL CUMULANTE DE CUARTO ORDEN

El cumulante de cuarto orden para el proceso análogo a (1) se puede calcular de la siguiente manera:

$$C_{4y}(\tau_1, \tau_2, \tau_3) = C_{4x}(\tau_1, \tau_2, \tau_3) + C_{4w}(\tau_1, \tau_2, \tau_3) \quad (3)$$

Donde x, y y w corresponden a la señal útil, la señal de salida y el ruido respectivamente.

$$\begin{aligned} C_{4y}(\tau_1, \tau_2, \tau_3) &= E\{x(t) \cdot x(t+\tau_1) \cdot x(t+\tau_2) \cdot x(t+\tau_3)\} \\ &\quad - C_2(\tau_1) \cdot C_2(\tau_2 - \tau_3) \\ &\quad - C_2(\tau_2) \cdot C_2(\tau_3 - \tau_1) \\ &\quad - C_2(\tau_3) \cdot C_2(\tau_1 - \tau_2) \end{aligned} \quad (4)$$

lo que da como resultado, aplicando el mismo principio para el cálculo del cumulante de tercer orden:

$$C_{4y}(\tau_1, \tau_2, \tau_3) = -\frac{A^4}{8} \left[\begin{array}{l} \cos w_c(\tau_1 - \tau_2 - \tau_3) + \\ + \cos w_c(\tau_2 - \tau_3 - \tau_1) + \\ + \cos w_c(\tau_3 - \tau_1 - \tau_2) \end{array} \right] \quad (5)$$

Si se hace $\tau_1 = \tau_2 = 0$, se obtiene lo siguiente:

$$C_{4y}(\tau, 0, 0) = -\frac{3}{8} A^4 [\cos w_c(\tau)] \quad (6)$$

Si $w(n)$ es ruido blanco gaussiano, entonces $C_{4w}(\tau_1, \tau_2, \tau_3) = 0$.

A. COMPONENTE UNIDIMENSIONAL DEL CUMULANTE DE CUARTO ORDEN

Haciendo un análisis de los resultados obtenidos hasta el momento, se puede notar que con tan solo utilizar la componente $(\tau, 0, 0)$ del cumulante de cuarto orden de la señal contaminada, se pueden recuperar los parámetros fundamentales de la muestra útil de señal. Si se realiza un análisis de lo descrito con anterioridad, análogamente a (4), haciendo $\tau_2 = \tau_3 = 0$ se puede obtener lo siguiente:

$$C_{4y}(\tau_1, 0, 0) = E\{x(t)^3 \cdot x(t+\tau)\} - 3 \cdot C_{2y}(\tau_1) \cdot C_{2y}(0) \quad (7)$$

Siendo $C_{2y}(\tau_1)$ la función de autocorrelación de la señal.

IV. RESULTADOS EXPERIMENTALES

Después de esta descripción teórica se pasa a comprobar la efectividad del método mediante la herramienta de simulación Matlab, utilizando las señales de muestra para el análisis de las técnicas descritas. Como se puede observar en la figura 1b, la señal contaminada posee una componente espectral en 50 Hz (0.05 normalizada); las demás son componentes de frecuencia del ruido. Teniendo en cuenta que la relación señal/ruido de entrada es de -7.1332db, se puede notar la efectividad del método empleando los cumulantes de orden superior ya que a la salida se obtiene una estimación de la relación señal/ruido de 21.2659db.

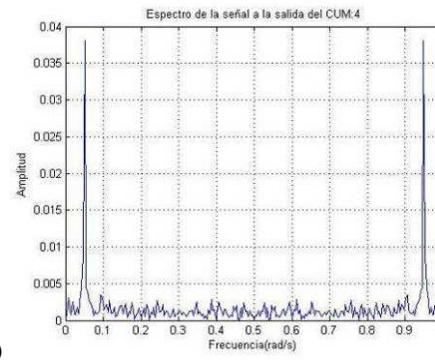
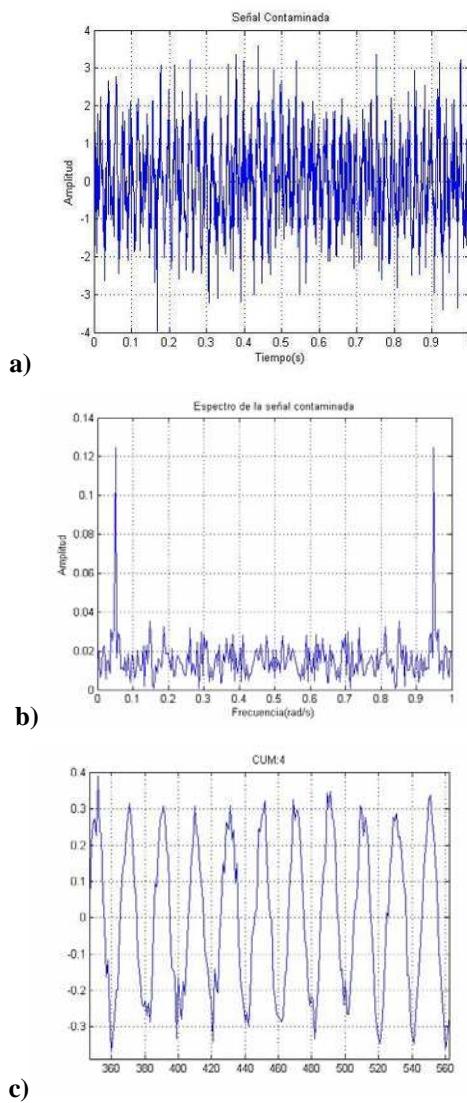


Figura 1 Ilustración de: a) Señal Útil-Ruido-Señal-Contaminada b) Espectro de Frecuencias de la señal contaminada (normalizada a π rad/s) c) Componente Unidimensional del cumulante de cuarto orden de la señal contaminada d) Espectro de Frecuencias de la señal de salida (normalizada a π rad/s)

Como se observa en la figura 1c y 1d respectivamente, el ruido presente en la señal de salida disminuye con respecto al presente en la señal contaminada (figura 1a y 1b). Otro aspecto relativo a la efectividad de este método radica en que el mismo no necesita tener información alguna de la señal de entrada (manejar una señal de referencia que esté correlacionada con el ruido) para obtener parámetros fundamentales de la señal, como son: la amplitud y frecuencia, los cuales pueden verse afectados cuando la señal está contaminada por ruido.

V. IMPLEMENTACIÓN SOBRE FPGA

A partir de lo mencionado anteriormente se inicia la implementación de la arquitectura para el algoritmo, según los parámetros de señal para los cuales el sistema deberá responder de acuerdo a las características de ruido presente en la misma. Todo el sistema fue sintetizado mediante la arquitectura FPGA de Xilinx SPARTAN-3AN, la cual posee 700 mil compuertas, 20 multiplicadores embebidos y un reloj externo de 50 MHz, frecuencia base a la cual se trabaja el diseño. Esto conduce a usar EDK (*Embedded Development Kit*), plataforma de la misma compañía para sistemas embebidos en sus productos, que a su vez se integra con ISE (*Integrated Software Environment*), herramienta que se encarga de los procesos de síntesis e implementación.

En el proceso de configuración del hardware embebido en el FPGA se usa como herramienta fundamental el asistente BSB (Base System Builder), el cual pertenece a la plataforma XPS (Xilinx Platform Studio) que automatiza las tareas básicas de configuración del hardware y software para la mayoría de los diseños basados en FPGA de Xilinx. A continuación se listan los módulos y periféricos usados, así como su configuración:

- MicroBlaze0: Frecuencia de reloj de 50 MHz; se usó depuración, memoria interna BRAM 4 kB; no se usó cache; se habilitó FPU e interfaz PLB para acceso a memoria externa.
- MicroBlaze1: Frecuencia de reloj de 50 MHz; se usó depuración, memoria interna BRAM 16 kB; no se usó cache; se habilitó FPU, no se habilitó interfaz PLB.



- XPS UARTLITE: Razón de baudios de 115200, 8 bits de datos, no se usó paridad, no se habilitó interrupción.
- XPS MCH EMC: Controlador de memoria RAM externa.
- XPS TIMER: Tamaño de 32 bits, con 2 temporizadores presentes; se usó interrupción.

El dispositivo estándar de Entrada y de Salida para MicroBlaze0 (STDIN y STDOUT) es el puerto RS232 y para MicroBlaze1, es el módulo MDM (Microblaze Debugger Module).

La comunicación entre procesadores se realiza a través del bus FSL de alta velocidad y se establece un mecanismo de arbitraje semafórico, definido por software mediante señalizaciones que indican cuándo un procesador finalizó el cálculo y está listo para enviar los datos.

El entorno de desarrollo de XPS se utilizó para generar el algoritmo correspondiente a la componente unidimensional del cumulante de cuarto orden de la señal, además de la arquitectura hardware que compone el sistema para el cálculo de la misma. Además, sobre esta plataforma se ejecutó y verificó todo el sistema de cancelación de ruido, incluyendo tanto las señales simuladas, como las reales utilizadas. Conjuntamente con el diseño en XPS, se elaboró una variante del algoritmo en hardware utilizando AccelDSP, la cual es una herramienta de síntesis proporcionada por Xilinx que permite transformar un diseño en punto flotante desarrollado en Matlab en un módulo *hardware* que puede ser implementado en un FPGA. Posee además una interfaz de usuario fácil de usar que controla un ambiente integrado con otras herramientas de diseño, tales como: Matlab, Xilinx ISE, System Generador, y otras como simuladores de código HDL y sintetizadores lógicos [5].

AccelDSP se utilizó para generar el bloque que calcula la componente unidimensional del cumulante de cuarto orden de la señal contaminada y es usado posteriormente como modelo de comparación con la arquitectura multiprocesador propuesta.

VI. TOPOLOGÍA DE RED MULTIPROCESADOR

Antes de implementar el diseño en una arquitectura multiprocesador se hace necesario discernir cual “topología de red” emplear según la eficiencia de esta y las potencialidades de cada una, según el diseño tratado para conectar varios procesadores paralelamente. Existen varias topologías de red que pueden ser materializadas con enlaces punto a punto a través del bus FSL [6] [7].

Para el caso particular del diseño presentado en este trabajo se escogió una topología simple Maestro-Esclavo como muestra la figura 2, aunque pudiera haberse empleado otro tipo de topología más eficiente, para equiparar aún más las cargas computacionales a la hora de implementar el algoritmo propuesto para la cancelación de ruido. El diseño se ve limitado al dispositivo FPGA con que se cuenta, ya que un procesador Microblaze consume aproximadamente entre el 22% y el 25% de los recursos del mismo.



Figura 2. Topología de Red Multiprocesador usada en el diseño

VII. INCREMENTO DE LA VELOCIDAD Y EFICIENCIA

La aceleración de un algoritmo paralelo (Ac) se define como el tiempo necesario para resolver el problema con un solo procesador (T_s) dividido por el tiempo necesario para resolver el problema con p procesadores (T_p), esta relación se muestra en la expresión 10. En un sistema ideal y con un algoritmo paralelo ideal, la aceleración sería igual a la cantidad de procesadores p . En el mundo real, siempre es más bajo debido a los gastos generales en el proceso de comunicación y transferencia de datos entre los procesadores [8].

$$Ac = \frac{T_s}{T_p} \quad (10)$$

Para el caso de este trabajo la aceleración del sistema con respecto al uso de un solo procesador es aproximadamente dos veces mayor. Con respecto a la eficiencia del diseño se puede plantear que esta es inversamente proporcional al número de procesadores presentes en el mismo, lo cual significa que se debe tratar de obtener la mayor eficiencia posible que corresponda a la aceleración máxima que se necesita.

La eficiencia como depende del número de procesadores esta íntimamente relacionada con el consumo de recursos del FPGA y al contar con una tarjeta de gama baja no es posible incrementar la cantidad de procesadores todo lo requerido por lo cual la aceleración del diseño se queda por debajo de la máxima posible.

VIII. ARQUITECTURA DE DISEÑO

Una de las ventajas del trabajo con hardware reconfigurable es la posibilidad de la aceleración de algoritmos debido al procesamiento paralelo que presentan los dispositivos FPGAs. Sin embargo, en ocasiones el procesamiento matemático de una determinada función no es realizable del todo paralelamente ya que el FPGA se ve limitado en cuanto a recursos o unidades de multiplicación-acumulación (MAC), y por otra parte, existen funciones que en sí, su procesamiento o modo de computación es secuencial; ejemplo de ello es la función de autocorrelación de una señal, la cual, en ese trabajo, es necesario realizar, para todos los desplazamientos posibles, en particular, hasta la dimensión de vector que contiene la señal contaminada.

En esta investigación se elige como arquitectura de diseño el uso de un sistema multiprocesador que trabaja en paralelo y realiza por separado partes del algoritmo correspondiente a la componente unidimensional del cumulante de cuarto orden de la señal contaminada. Esto equipara las cargas computacionales y aprovecha una de las posibilidades que brinda el diseño con

FPGAs, relativo al uso de sistemas multiprocesadores embebidos. En la figura 3 se puede observar el algoritmo propuesto para el cómputo de la componente $(\tau,0,0)$ del cumulante de cuarto orden de la señal contaminada sobre dos núcleos de procesamiento, teniendo previamente como primer paso, el proceso de digitalización de señal. Una vez digitalizada esta, se adiciona ruido blanco de naturaleza gaussiana y se elimina el valor medio de la muestra a procesar. Concluida esta etapa se almacenan todas las muestras de señal contaminada en un espacio de memoria RAM de cada uno de los procesadores, comenzando el proceso de cómputo paralelamente.

El microprocesador MB1 se encarga de ejecutar el término del algoritmo asociado al cálculo de la función de autocorrelación así como la multiplicación de esta por el cumulante de segundo orden en el origen (ecuación (7)); el resultado obtenido se transfiere a través del bus FSL hacia el microprocesador MB2. Este último calcula el término $E[x(t)^3 \cdot x(t+\tau)]$ de la ecuación (7) y le substrae el resultado transferido por el microprocesador MB1.

Una vez cumplimentadas estas funciones, se envía el resultado de las operaciones del microprocesador MB2 hacia MB1, a través del bus FSL, y este a su vez envía el resultado vía RS232 hacia el PC. Como se puede notar existe un mayor uso de memoria por parte del microprocesador MB1 con respecto a MB2; por esta razón, desde el diseño, a este se le asigna un tamaño mayor de memoria de programa, ejecutándose la aplicación correspondiente al mismo en memoria externa. Como se observa en la figura 3, el objetivo es que ambos núcleos por separado calculen partes del algoritmo para posteriormente ejecutar la substracción del resultado de ambos y enviar lo obtenido vía RS232 hacia Matlab, para su comprobación con el diseño establecido mediante simulación.

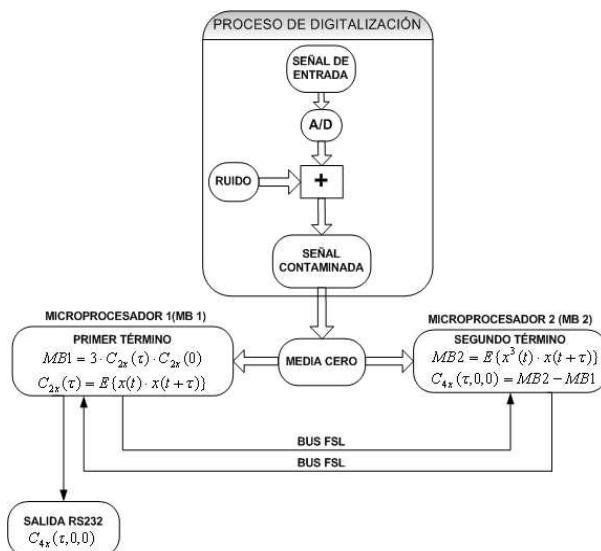


Figura 3. Algoritmo para el cálculo de la componente unidimensional del cumulante de cuarto orden

En la figura 4 se puede observar el diagrama en bloques de la arquitectura hardware generada a través de la herramienta XPS, la cual está constituida por dos núcleos de procesamiento

con unidades independientes de aritmética en punto flotante y un bus FSL de alta velocidad para el intercambio de información entre los procesadores. Además, la arquitectura posee una unidad de memoria DDR2-SDRAM y dos buses PLB independientes para cada procesador, utilizados en el acceso a memoria y el control de periféricos en el diseño, además de un controlador de puerto RS232 para la transmisión de datos hacia el PC. En cuanto al consumo de recursos del sistema, la tabla 2 resume su utilización en el diseño.

TABLA 2. CONSUMO DE RECURSOS DEL SISTEMA.

Sumario de Diseño Arquitectura Spartan-3AN xc3s700an		
Número de BSCANs	1 de 1	100%
Número de BUFGMUXs	6 de 24	25%
Número de DCMs	2 de 8	25%
Número de MULT18X18SIOs	14 de 20	70%
Número de RAMB16BWEs	15 de 20	75%
Número de Slices	4329 de 5888	73%
Número de Slice Flip Flops:	4284 de 11776	36%
Número de 4 input LUTs:	6059 de 11776	51%

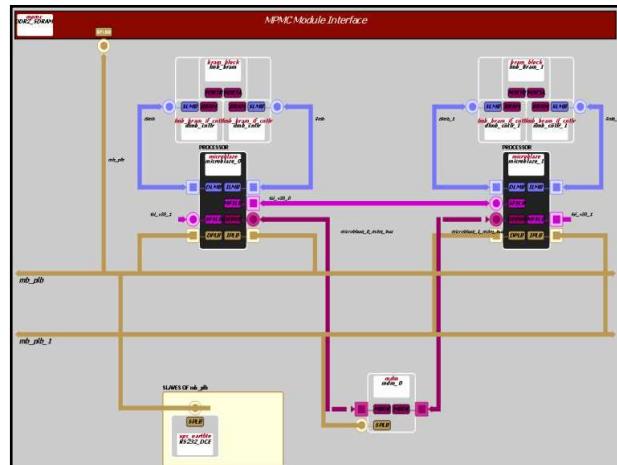


Figura 4. Diagrama en Bloques de la Arquitectura Hardware generada a través de la Herramienta XPS

IX. ANÁLISIS DE RENDIMIENTO

Para hacer un análisis del rendimiento del sistema, se realiza una comparación con la implementación del estadístico de orden superior (componente $(\tau,0,0)$ del cumulante de cuarto orden), mediante la herramienta de síntesis AccelDSP.

Como análisis, se muestra en la figura 5 la gráfica del comportamiento de los recursos críticos consumidos por ambas implementaciones lo cual muestra un excesivo consumo de recursos en cuanto a los multiplicadores embebidos en el diseño basado en AccelDSP. Por otro lado, los demás recursos consumidos por ambas implementaciones son similares dando validez a la arquitectura propuesta, cuyo objetivo es obtener un modelo óptimo para calcular funciones bidimensionales y tridimensionales del análisis estadístico de orden superior, sin

que su ejecución lleve aparejado un mayor consumo de recursos hardware además de los disponibles en el FPGA y solo dependa de la optimización de la función en software y de la frecuencia máxima de trabajo de los procesadores.

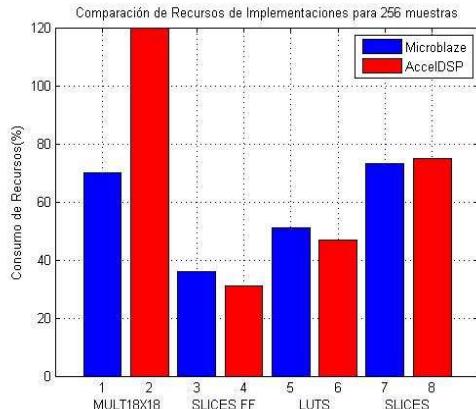


Figura 5. Consumo de recursos de implementaciones en AccelDSP y XPS

Como última comparación en el diseño propuesto, se establece un análisis del tiempo de ejecución de las implementaciones del sistema, empleando un solo procesador, para distintas funciones. Como se observa en la figura 6 la aceleración del sistema propuesto con respecto a uso de un solo procesador es casi el doble para las funciones descritas en el trabajo como son la autocorrelación de la señal y la componente unidimensional del cumulante de cuarto orden.

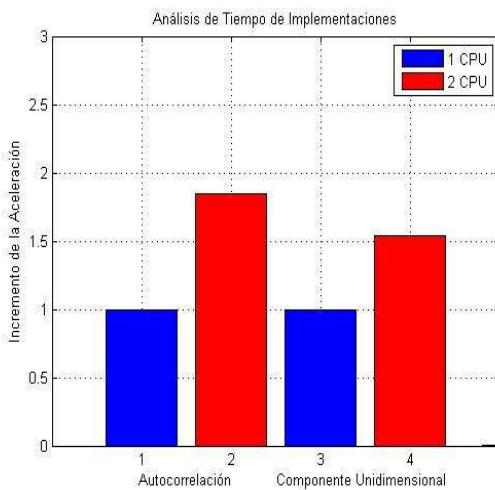


Figura 6. Comparación del rendimiento usando 1 y 2 procesadores respectivamente

En la figura 7 se puede observar la señal a la salida del sistema utilizando XPS (rojo) y su comparación con la respuesta simulada en Matlab (azul), para 256 muestras de la señal contaminada lo cual concluye la efectividad del cálculo empleando sistemas multinúcleo sobre FPGA.

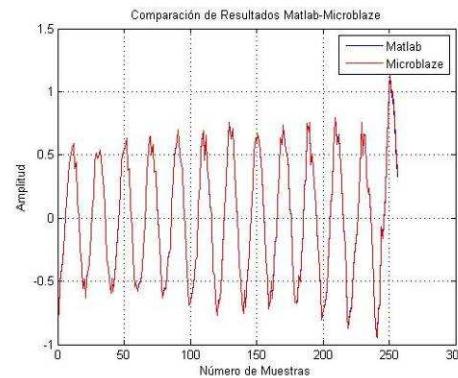


Figura 7. Comparación de Resultados Matlab-Microblaze

CONCLUSIONES

El resultado más importante de este trabajo deviene de la aplicación de sistemas multiprocesador con arquitectura paralela en tareas de reducción o cancelación de ruido fundamentado en el uso de funciones estadísticas de alto orden, lo cual posibilita obtener un modelo que pueda describir las características de señal útil, sin tener conocimiento alguno de esta o de la muestra contaminante. Estos resultados han sido verificados además, para muestras provenientes de sensores de vibración, así mismo para señales multitonos y modulaciones del tipo ASK y BPSK contaminadas por ruido blanco gaussiano con una relación señal/ruido similar a la expuesta.

El trabajo constituye además un acercamiento a la utilización de técnicas de cancelación de ruido fundamentadas en las propiedades estadísticas del proceso analizado, lo cual puede ser útil para efectuar comparaciones en cuanto a algoritmos y distintos tipos de señales, logrando así una mejor comprensión del ruido contaminante en la señal, además de tener como base, la utilización de una arquitectura u otra y el algoritmo de cómputo mas eficiente para esta, según la naturaleza del ruido incidente en la señal.

REFERENCIAS

- [1] Roy M. Howard. Principles of Random Signal Analysis and Low Noise Design, John Wiley & Sons Ltd. 2002
- [2] Signal Processing with Higher-Order Spectra, IEEE Signal Processing Magazine Julio 1993.
- [3] Izzet Ozcelik Blind Deconvolution of Music Signals using Higher Order Statistics, 17th European Signal Processing Conference (EUSIPCO 2009), Glasgow, Scotland, August 24-28, 2009.
- [4] Josep M. Salavedra Molí, Técnicas de Speech Enhancement considerando Estadísticas de Orden Superior, Tesis Doctoral, Barcelona, Junio 1995.
- [5] AccelDSP Synthesis Tool User Guide obtenido de: http://www.xilinx.com/aceldsp_user.pdf.
- [6] P.HUERTA, J.CASTILLO, J.I.MÁRTINEZ, V.LÓPEZ. Multi MicroBlaze System for Parallel Computing obtenido de: <http://www.etsii.urjc.es/~phuerta/pdf/MMB.pdf>. 2011
- [7] Sistemas MPSoC en FPGA obtenido de: <http://www.etsii.urjc.es/~phuerta/pdf/442.pdf>. 2011
- [8] A Microblaze Based Multiprocessor SoC obtenido de: <http://www.etsii.urjc.es/~phuerta/pdf/MBMPSOC.pdf>. 2011

Implementación en FPGA de generadores de números pseudoaleatorios utilizando registros de desplazamiento realimentados

Claudio M. González, Carlos A. Gayoso, Leonardo J. Arnone y Miguel R. Rabini

Laboratorio de Componentes Electrónicos

Universidad Nacional de Mar del Plata, Argentina

cmgonzal@fimdp.edu.ar

Resumen—Este trabajo consiste en el estudio e implementación de generadores de números pseudoaleatorios (PRNG) utilizando Registros de Desplazamiento Realimentados (LFSR). Las implementaciones se realizaron en dispositivos lógicos programables (FPGA). Primeramente se presentan las distintas configuraciones de LFSRs detallando convenciones y características generales, luego se describen implementaciones usuales en FPGA de PRNGs utilizando LFSRs y los parámetros de comparación utilizados, más adelante se proponen implementaciones superadoras utilizando los conceptos de Generadores de Geffe generalizados y generadores de salto hacia adelante (*leap ahead*).

I. INTRODUCCIÓN

El presente trabajo consiste en el estudio e implementación de generadores de números pseudoaleatorios (PRNG) utilizando Registros de Desplazamiento Realimentados (LFSR). Las implementaciones se realizaron en dispositivos lógicos programables (FPGA).

El trabajo está organizado de la siguiente manera: en el punto II se presentan las distintas configuraciones de LFSRs detallando convenciones y características generales. En el punto III se describen implementaciones usuales en FPGA de PRNGs utilizando LFSRs y los parámetros de comparación empleados, también se proponen implementaciones superadoras comparando los resultados obtenidos. Finalmente en el punto IV se expresan las conclusiones obtenidas a partir del trabajo realizado.

II. REGISTROS DE DESPLAZAMIENTO REALIMENTADOS

Los Registros de Desplazamiento Realimentados Linealmente, LFSR (*Linear Feedback Shift Registers*), son registros de desplazamiento en los que se realimenta a la entrada una función lineal del estado previo [1][2].

La función más utilizada es una XOR, por lo tanto el bit de entrada estaría determinado por una XOR de todos o algunos bits de los estados del registro de desplazamiento. El esquema general de un LFSR de n etapas es mostrado en la figura 1 a), en dicha figura los coeficientes $(c_1, c_2, \dots, c_{n-1})$ indican el esquema de conexiones realizado. En la figura 1 b) se muestra el caso en que el número de etapas es $n = 8$ y $c_7 = c_5 = c_3 = 1$, siendo el resto de los coeficientes iguales a 0.

Convencionalmente se puede expresar en forma abreviada el LFSR como un conjunto de las conexiones activas, agregando

las de la entrada y la salida que siempre están conectadas ($c_0 = c_n = 1$), dispuestas entre corchetes. Por ejemplo el circuito de la figura 1 b) se puede definir como : $[8, 7, 5, 3, 0]$. Este conjunto que representa al LFSR suele denominarse secuencia de conexiones o *Tap Sequence*.

Los LFSR pueden también ser representados por un polinomio módulo 2 del tipo $c_n x^n + c_{n-1} x^{n-1} + \dots + c_1 x + c_0$, en el ejemplo mencionado tal polinomio es: $x^8 + x^7 + x^5 + x^3 + 1$.

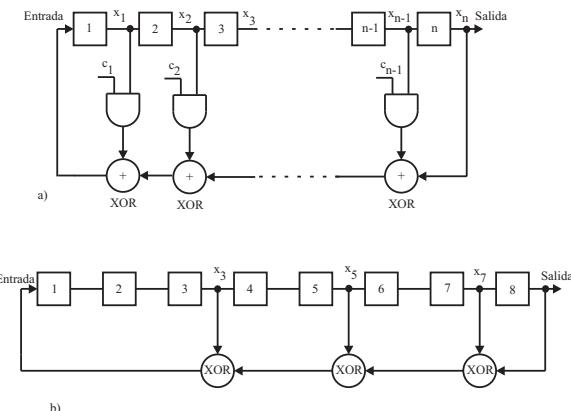


Figura 1. Configuración Fibonacci. Esquema y ejemplo para $n=8$.

La disposición circuital de la Fig. 1 es la denominada configuración de Fibonacci. Otra configuración posible es la denominada de Galois y es mostrada en la Fig. 2 a). En la figura se ve que el bit de salida se realimenta directamente a la entrada y en cada registro por medio de una XOR con la salida de la etapa previa en los casos en que los coeficientes g_i ($i = 1, \dots, n - 1$) correspondientes sean igual a 1. En forma similar a la configuración de Fibonacci se define un conjunto de coeficientes : $(g_1, g_2, \dots, g_{n-1})$ con $g_0 = g_n = 1$ siempre conectados. En la figura Fig. 2 b) se muestra un ejemplo con $n = 8$ y $g_1 = g_3 = g_5 = 1$, siendo el resto de los coeficientes iguales a 0. Puede también expresarse en este caso, la secuencia de conexiones en forma abreviada como: $[8, 5, 3, 1, 0]$.

Se puede demostrar que si se utiliza una secuencia de conexiones de Galois tal que cada componente se obtenga de la resta entre n (n° de etapas) y los componentes de una

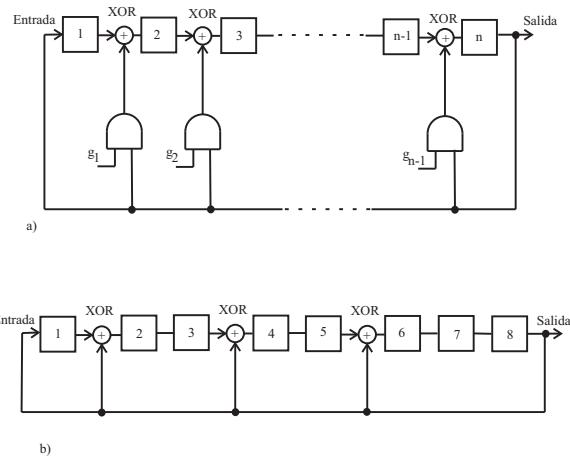


Figura 2. Configuración Galois. Esquema y ejemplo para $n=8$.

secuencia de conexiones de Fibonacci, las salidas de ambos LFSRs son equivalentes [2]. De tal forma los ejemplos de las Figs. 1 b) y 2 b) generan secuencias de salida equivalentes.

A partir de una condición inicial, denominada semilla, los LFSR generan una secuencia de bits determinística. Como el número de estados es finito necesariamente hay un período de repetición.

Sin embargo estos circuitos son muy utilizados como generadores de bits pseudoaleatorios, ya que con una cantidad de etapas y una elección de secuencias de conexiones adecuadas se puede lograr un período muy largo y un comportamiento estadístico similar al de una secuencia aleatoria.

De hecho si se utiliza como secuencia de conexiones a las provenientes de un polinomio primitivo de grado n , el período resultante es el máximo posible, $P = 2^n - 1$, con la particularidad de que se presentan todos los estados posibles, exceptuando el nulo, sin repetición dentro de dicho período [2]. Puede verse, además, que el estado nulo, en que todas las salidas de los registros sean 0s, es un estado prohibido, ya que el sistema queda bloqueado en él.

Los LFSRs pueden utilizarse como: generadores de secuencias pseudoaleatorias, generadores de secuencias de pseudo ruido, en comunicaciones, en criptografía entre otras aplicaciones.

Pueden implementarse fácilmente tanto en software como en hardware. De los esquemas generales mostrados en las Figs. 1 y 2 se ve que la implementación en hardware es particularmente sencilla ya que sólo se necesitan registros y compuertas XOR, pudiéndose lograr en principio poco retardo y por lo tanto gran velocidad de funcionamiento.

La configuración Galois aparece como la de mayor velocidad de funcionamiento, debido a que sólo hay en el peor de los casos, una compuerta XOR entre registros, teniendo la configuración Fibonacci en cambio que sortear varios niveles de retardo en la realimentación.

Sin embargo si los LFSR se realizan en FPGAs, y pudiendo utilizar secuencias de conexiones tales que la función de realimentación pueda ser implementada en una única celda, la

configuración Fibonacci es la más adecuada ya que se utilizan registros de desplazamiento ya optimizados como bloques de biblioteca.

Teniendo en cuenta este último criterio en este trabajo se utilizarán LFSRs en su configuración Fibonacci, asegurándose que las conexiones no tenga más de tres XORs (cuatro conexiones), debido a que se usarán FPGAs de la familia Cyclone II de Altera [3] que tienen tablas de búsqueda de cuatro variables en su celda básica.

III. IMPLEMENTACIÓN DE PRNGs UTILIZANDO LFSRs

Los generadores de números pseudoaleatorios (PRNG) son utilizados en muchas áreas del conocimiento tales como Ingeniería, Economía, Física, Estadística, etc. En particular en simulaciones, método de Monte Carlo, criptografía y telecomunicaciones. Ninguna secuencia producida por una máquina de estados finitos puede ser realmente aleatoria, debido a que son determinísticas por naturaleza y su salida es predecible. Por lo tanto, por estos medios, se generan secuencias con un período finito, pero si dicho período es lo suficientemente largo, las secuencias generadas pueden ser utilizadas satisfactoriamente en muchas de estas aplicaciones [4] [5] [6].

Los LFSRs generan secuencias de bits pseudoaleatorias, sin embargo las salidas de sus registros tomadas en paralelo no pueden ser utilizados como generador de números pseudoaleatorios, ya que existe una estrecha correlación entre un estado y el siguiente.

En esta sección se implementan PRNGs utilizando LFSRs en dispositivos lógicos programables de la familia Cyclone II de Altera (EP2C35F672C6)[3], generando números de 32 bits. Para evaluarlos se utilizan los siguientes parámetros:

- Período de repetición de estados.
- Test básicos de aleatoriedad. Para determinar las características de la secuencia generada se realizaron cinco pruebas estadísticas básicas: de frecuencia, serie, poker, rachas y autocorrelación. Estas pruebas son condiciones necesarias pero no suficientes para demostrar que una secuencia “luzca aleatoria” [7] .
- Test Diehard de Marsaglia [8]. Batería de 17 tests estadísticos, actuando sobre secuencias de más de 2500000 números de 32 bits.
- Frecuencia máxima de generación de palabras de 32 bits y cantidad de celdas lógicas utilizadas (Elementos lógicos, LEs [3]) en EP2C35F672C6 Cyclone II de Altera.

III-A. Implementaciones usuales

La manera más sencilla de generar números de n bits a través de un LFSR, es la de acumular n resultados de la secuencia de bits de salida. Teniendo la consecuencia lógica de la disminución de la frecuencia de generación de números en un factor de $1/n$.

En este sentido se implementó un LFSR de 32 bits (denominado LFSR32) con una secuencia de conexiones [32, 30, 26, 25, 0]. La frecuencia máxima de reloj del sistema

es de $312,30\text{MHz}$, por lo que se obtuvo una frecuencia de generación de números $f_n = \frac{312,3\text{MHz}}{32} = 9,8\text{MHz}$, con una utilización de 104 LEs. El período de repetición es de $P = (2^{32} - 1) \times 32 \approx 1,37 \times 10^{11}$.

Para aumentar la frecuencia de generación de números f_n , se implementó un generador, denominado LFSRPAR, que consiste en 8 LFSRs de 8, 9, 10, 11, 12, 13, 14 y 15 bits respectivamente funcionando en paralelo. Utilizando los bits de salida de cada LFSR, se genera un bus de 8 bits, $X = (X_8, X_7, \dots, X_1)$. Si se acumulan 4 estados consecutivos de este vector se obtiene un número de 32 bits (X_{32}) cada cuatro ciclos del reloj general. En la Fig. 3 se muestra el esquema circuital del generador LFSRPAR. La frecuencia máxima de reloj es de 302,7 MHz y la $f_n = 75,7\text{MHz}$, utilizando 167 LEs. El período de repetición de estados de la señal X es el mínimo común múltiplo (mcm) de los períodos individuales, es decir: $P_X = \text{mcm}(2^8 - 1, 2^9 - 1, \dots, 2^{15} - 1) \approx 2,4 \times 10^{22}$, siendo el período de la secuencia acumulada $P = P_X \times 4$.

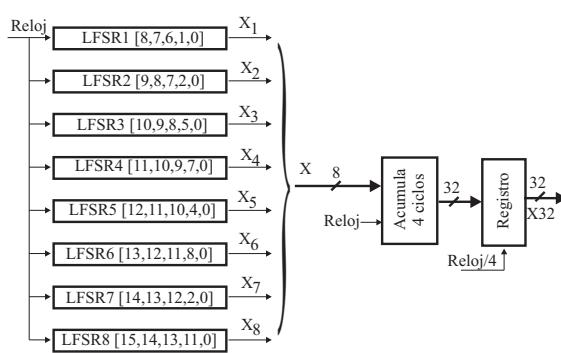


Figura 3. Generador LFSRPAR.

Combinando LFSRs se pueden obtener mejoras en el período y en las características estadísticas de las secuencias generadas. En la Fig. 4 se puede ver una implementación del generador de Geffe (GEFFE32)[9], en dicha figura la secuencia de bits de salida G se obtiene de la salida de un multiplexor controlado por la salida Z(30) del LFSR3 y cuyas entradas son X(32) del LFSR1 e Y(40) del LFSR2. La frecuencia máxima del sistema fue de 300,05MHz y la frecuencia de generación de números G32 es $f_n = \frac{300,0\text{MHz}}{32} = 9,4\text{MHz}$, se utilizaron 176 LEs. El período de repetición de la señal G se puede calcular como el mínimo común múltiplo de los períodos individuales de LFSR1, LFSR2 y LFSR3, es decir $P_G = \text{mcm}((2^{32} - 1), (2^{40} - 1), (2^{30} - 1)) \approx 1,94 \times 10^{25}$, mientras que el período total $P = P_G \times 32$.

Se simuló numéricamente a los generadores implementados hasta aquí, pasando satisfactoriamente los test básicos de aleatoriedad, pero no la batería de tests Diehard de Marsaglia.

III-B. Implementaciones propuestas

En este punto se proponen combinaciones e implementaciones superadoras, con respecto a las características estadísticas

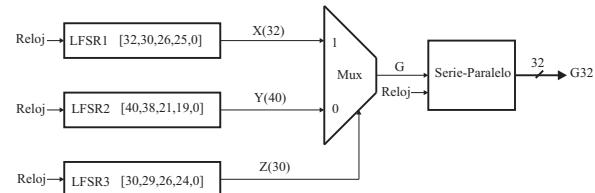


Figura 4. Generador de Geffe.

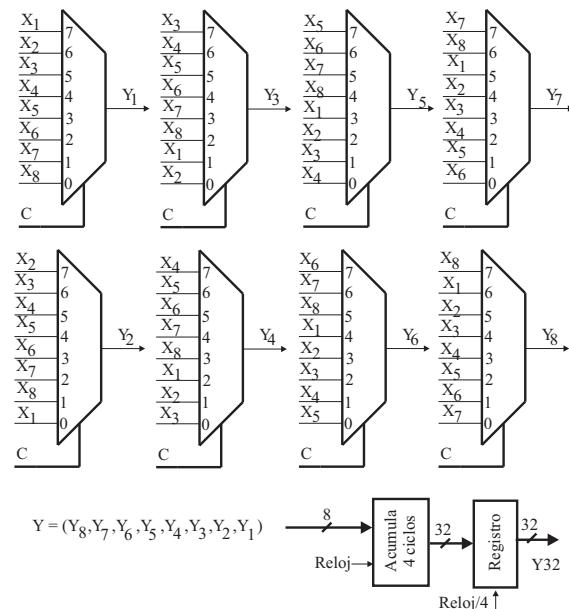
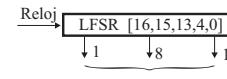


Figura 5. Generador de Geffe Generalizado.

cas, a la frecuencia de generación de números e intentando mantener períodos de repetición de secuencias largos. Esto se puede lograr con el lógico incremento de los recursos de hardware involucrados. Utilizando la señal $X = (X_8, X_7, \dots, X_1)$ del generador LFSRPAR, mostrada en la Fig. 3 y utilizando el concepto de generador de Geffe, ya mostrado, generalizado, se puede implementar un generador como el ilustrado en la Fig. 5, denominado GEFFEGEN. En la figura se muestra que se utiliza un LFSR adicional de 16 etapas ([16, 15, 13, 4, 0]) que tiene la función de controlar 8 multiplexores de 8 líneas a 1, la señal de control C se forma con las salidas de las etapas 16, 8 y 1 del mencionado LFSR. En función de la evolución de C se mezcla la señal X para dar Y, luego Y, que tiene 8 bits, se acumula obteniendo la salida denominada Y32. Esta combinación trabaja con una frecuencia de reloj de 314,9 MHz y una frecuencia de generación de números de 78,2 MHz. Se utilizaron 200 LEs, siendo el período de repetición de Y $P_Y \approx 6,2 \times 10^{24}$ y el período de la secuencia acumulada $P = P_Y \times 4$. Esta combinación pasa satisfactoriamente los tests Diehard de Marsaglia.

Los estados consecutivos de los LFSR están muy correlacionados entre sí, por lo que no pueden utilizarse directamente como generadores de números pseudoaleatorios. En cambio si se utiliza un estado cada m transiciones del reloj, se logra una marcada mejoría en las características estadísticas de la secuencia. Esto traería una baja en la frecuencia de generación de números en un factor de $1/m$. Se puede evitar esta baja en la frecuencia si se utiliza el concepto de salto hacia adelante (*leap ahead*)[10].

La expresión de próximo estado de un LFSR de n etapas es: $X(t+1) = A \times X(t)$ [10], donde, $X = (X_1, X_2, \dots, X_n)$ es el vector de estados y A la matriz de transiciones construida del siguiente modo:

$$\begin{bmatrix} c_1 & c_2 & \dots & c_{n-1} & c_n \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

En esta matriz la primera fila se construye con los coeficientes del LFSR, en la posición (2,1) se ubica una matriz identidad $I_{(n-1) \times (n-1)}$ y en la posición (2,n) se ubica un vector columna nulo de tamaño $n-1$.

El estado $X(t+2)$ puede expresarse entonces como $A \times (A \times X(t)) = A^2 \times X(t)$. Generalizando se pueden saltar m estados, por lo que se obtiene la expresión $X(t+m) = A^m \times X(t)$ o la equivalente $X'(t+1) = A^m \times X'(t)$, es de notar que las operaciones involucradas son en módulo 2. Resolviendo esta expresión se pueden obtener las ecuaciones de diseño para saltar m estados en cada transición del reloj principal.

Por ejemplo si tomamos el LFSR de $n = 8$ etapas de la Fig. 1 b), con secuencia de conexiones [8, 7, 5, 3, 0], y utilizamos un salto de $m = 3$ estados, la matrices obtenidas son:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Por lo tanto las ecuaciones de próximo estado se derivan de $A^3 \times X$, es decir:

$$\begin{aligned} X_1 &\leftarrow X_1 \oplus X_3 \oplus X_5 \oplus X_6 \\ X_2 &\leftarrow X_2 \oplus X_4 \oplus X_6 \oplus X_7 \\ X_3 &\leftarrow X_3 \oplus X_5 \oplus X_7 \oplus X_8 \\ X_4 &\leftarrow X_1 \\ X_5 &\leftarrow X_2 \\ X_6 &\leftarrow X_3 \\ X_7 &\leftarrow X_4 \\ X_8 &\leftarrow X_5 \end{aligned}$$

Con esta configuración se pueden tomar números de m bits a la frecuencia del reloj del sistema. Los m bits utilizados son las salidas de las m últimas etapas. El período de repetición de estados será: $P = \frac{mcm(2^n - 1, m)}{m}$, y se ve que no se produce un cambio significativo en los recursos de hardware, ni en la frecuencia máxima de trabajo, con respecto a los LFSRs convencionales.

Con este criterio se diseñó un LFSR *Leap Ahead* de $n = 32$ etapas, con conexiones [32, 30, 26, 25, 0] y con un salto de $m = 17$ estados, denominado LA3217. En este caso sólo se utilizaron los últimos 8 bits de los 17 disponibles, acumulando 4 estados consecutivos para lograr los números de 32 bits. En estas condiciones se logró una frecuencia máxima de reloj de 287,9 MHz y una frecuencia de generación de números de 72,0 MHz, utilizando 172 LEs. La secuencia generada pasa los tests Diehard de Marsaglia, pero el período de repetición de estados es: $P = \frac{mcm((2^{32}-1), 17)}{17} \times 4 \approx 1,0 \times 10^9$, siendo este valor demasiado bajo.

Para superar este inconveniente se utilizaron cuatro generadores del tipo *Leap Ahead*: LA3217, LA3615, LA4015 y LA4415. En la Fig. 6 se muestra el esquema circuital del generador denominado LAPAR, con el detalle de las conexiones, el número de etapas y de saltos en cada generador individual. Se utilizaron las 8 salidas más significativas de cada uno de ellos generando un número de 32 bits en cada transición del reloj general. Se utilizaron 395 LEs, con una frecuencia de reloj y de generación de números de 369,7 MHz. El período de repetición es alto, $P \approx 2,2 \times 10^{39}$, pasando satisfactoriamente los tests Diehard de Marsaglia.

Si se utiliza el concepto de generador de Geffe generalizado con LFSRs *Leap Ahead*, se pueden mejorar aun más las características de los PRNGs. El generador LAGEFFE de la Fig. 7 es un ejemplo de ello. Allí se utilizan las salidas LA1, LA2, LA3 y LA4 del generador LAPAR (Fig. 6) como entradas de 4 multiplexores controlados por una señal C generada por otro generador *Leap Ahead*, LA2815. Dicha señal se compone con los dos bits más significativos de LA2815. La evolución de C mezcla convenientemente a las señales provenientes de LAPAR, generando una salida de 32 bits de buenas características estadísticas. Se utilizaron 515

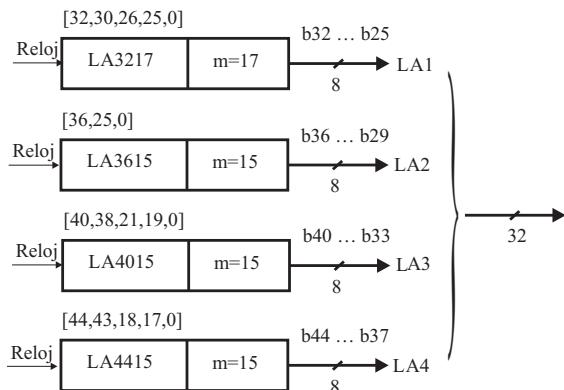


Figura 6. Generador LAPAR.

LEs, siendo la frecuencia máxima de reloj y de generación de números de 390,0 MHz. Pasa satisfactoriamente la batería de tests Diehard con un período de repetición $P \approx 4,0 \times 10^{46}$.

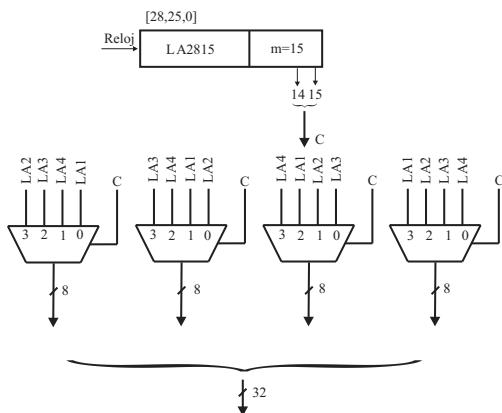


Figura 7. Generador LAGEFFE.

III-C. Comparación de resultados

La máxima fecuencia teórica en la que puede trabajar un LFSR puede establecerse, en una determinada tecnología, con un circuito compuesto por un flip-flop D realimentado a través de un negador. Haciendo esta prueba se obtuvo $f_{max} = 420,17\text{MHz}$.

En el Tabla I se resumen los resultados obtenidos, en él se ve que los generadores LAPAR y LAGEFFE se acercan bastante a ese límite, teniendo además, un período de repetición muy largo y pasando satisfactoriamente los tests Diehard de Marsaglia. Utilizando, como contrapartida, mayor cantidad de LEs que las otras implementaciones, sin embargo esta cantidad de recursos involucrados ($< 2\%$ del total del integrado utilizado), no son una cantidad importante con la tecnología actual.

Los generadores GEFFEGEN y LA3217 pasan Diehard, sin embargo sus frecuencias f_n no son tan altas como en los casos

Tabla I
COMPARACIÓN DE LOS DISTINTOS GENERADORES IMPLEMENTADOS. f_{max} Y f_n EN MHZ.

PRNG	f_{max}	f_n	LEs	Período	Diehard
LFSR32	312,3	9,8	104	$1,4 \cdot 10^{11}$	✗
LFSRPAR	302,7	75,7	167	$9,6 \cdot 10^{22}$	✗
GEFFE32	300,0	9,4	176	$6,2 \cdot 10^{26}$	✗
GEFFEGEN	314,9	78,2	200	$2,5 \cdot 10^{25}$	✓
LA3217	287,9	72,0	172	$1,0 \cdot 10^9$	✓
LAPAR	369,7	369,7	395	$2,2 \cdot 10^{39}$	✓
LAGEFFE	390,0	390,0	515	$4,0 \cdot 10^{46}$	✓

antes mencionados. Y en el caso de LA3217 su período es demasiado bajo.

Los otros tres generadores tienen el inconveniente de no pasar los tests exigentes de aleatoriedad, pero tienen períodos aceptables e implementaciones sencillas, con frecuencias de generación de números bajas.

IV. CONCLUSIONES

Se realizó un estudio y la implementación en FPGAs de generadores de números pseudoaleatorios mediante la utilización de registros de desplazamiento realimentados. Se analizaron en primera instancia implementaciones usuales, proponiendo luego topologías de diseño superadoras. Se realizaron las comparaciones correspondientes entre los generadores implementados y se llegó a que dos de los generadores propuestos, denominados LAPAR y LAGEFFE, poseen muy buenas características estadísticas, de frecuencia de generación de números y de período de repetición de estados, utilizando una cantidad de recursos de hardware no muy elevada.

REFERENCIAS

- [1] Bruce Schneier. *Applied Cryptography*. John Wiley & Son, 1996.
- [2] A. Menezes; P. Van Oorschot; S. Venstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [3] http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf. Cyclone II Device Handbook. *On Line*.
- [4] G. Marsaglia. A Current View of Random Number Generators. *Computer Science and Statistics: 16th Symposium on the Interface, Atlanta*, 1984.
- [5] P. L'Ecuyer. *Random Number Generation, Chapter 3 of Elsevier Handbooks in Operations Research and Management Science: Simulation*. Elsevier Science, 2005.
- [6] P. L'Ecuyer. Uniform Random Number Generation. *Annals of Operations Research*, (53):77–120, 1994.
- [7] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. Mc Graw-Hill, Inc., 1991.
- [8] G. Marsaglia. DIEHARD, Statistical Test. *On Line* <http://stat.fsu.edu/geo/diehard.html>.
- [9] M. Soriano y R. Díaz. *Generación y Análisis de Secuencias Pseudoaleatorias*. Ediciones UPC, Barcelona, 1999.
- [10] Gu Xiao-chen and Zhang Min-xuan. Uniform Random Number Generator Using Leap Ahead LFSR Architecture. *International Conference on Computer and Communications Security, 2009. ICCCS '09*, pages 150–154, Dec 2009.

Plataforma genérica de conversión de interfaces para buses de datos en aviónica

Leandro Aguirre
Redimec S.R.L.
www.redimec.com.ar
Tandil, Argentina
laguierre@redimec.com.ar

Lucas Leiva
INCA/INTIA
Facultad de Cs. Exactas, UNCPBA
Tandil, Argentina
lleiva@exa.unicen.edu.ar

Abstract— Tanto las aeronaves comerciales como militares están conformados por un gran número de sistemas electrónicos críticos que deben ser verificados periódicamente. Estos sistemas requieren de una máxima atención tanto en el normal funcionamiento como en mantenimiento. La mayoría de las interfaces actuales aeronáuticas basan sus comunicaciones en el estándar ARINC 429. Los equipos de análisis o testeo deben ser adquiridos en el mercado internacional, lo cual genera una dependencia de empresas del exterior. Por este motivo se presenta el desarrollo de un producto personalizable que permite la conversión de datos ARINC 429 a las interfaces más utilizadas, como por ejemplo RS232, RS422, USB. El producto presenta características que lo hacen apto para brindar otro tipo de soluciones, como por ejemplo, almacenado de datos. La plataforma desarrollada cuenta con un FPGA Spartan 3 de Xilinx, en donde se emplaza el procesamiento y conversión de datos.

Keywords-FPGA; IP core; conversión ARINC 429 a digital

I. INTRODUCCIÓN

Uno de los principales proveedores de soluciones en ingeniería de aviación del mundo es Aeronautical Radio, Incorporated (ARINC). Dentro de sus líneas de servicios ofrecen soporte en salud, aeronaves, aeropuertos, defensa, gobierno, redes, seguridad y transporte.

La actividad relacionada a la industria aeronáutica se encuentra en manos de tres comités (AEEC, AMC, y FSEMC), y éstos cooperativamente establecen por consenso los estándares referente a la aviación. AEEC (Airlines Electronic Engineering Committee) desarrolla estándares técnicos y de ingeniería que son ampliamente utilizados en equipos electrónicos de aviación, por su parte AMC (Avionics Maintenance Conference) desarrolla estándares de mantenimiento técnico y FSEMC (Flight Simulator Engineering and Maintenance) los relacionados a simulación y entrenamiento.

La AEEC surgió luego de la segunda guerra mundial con la intención de capitalizar el crecimiento en la industria de electrónica de a bordo. Este comité realiza investigaciones, evalúa y desarrolla estándares que competen a todos los segmentos de la aviación. La mayoría de los equipamientos presentes en las aeronaves, tanto comerciales, civiles y militares cumplen con los estándares ARINC. Éstos son usados

como base para el diseño, inversión, adquisición, soporte del ciclo de vida o decisiones de negocio. Por otra parte, provee mecanismos que permiten realizar de manera simple el reemplazo/actualización de equipos.

Los estándares propuestos por la AEEC se clasifican en las siguientes categorías:

- *Características*: definen la forma, ajustes, funciones e interfaces de los sistemas que componen a la aeronave (sistemas de cabina, de aviática y red de comunicación).
- *Especificaciones*: definen el empaquetado o montaje físico del equipamiento de la cabina; define estándares de seguridad de datos, comunicación y redes.
- *Reportes*: provee una guía de buenas costumbres aplicadas a la industria de la aviación, muy relacionada al mantenimiento, e ingeniería de simuladores de vuelo.

En la tabla 1 se presentan las diferentes series de estándares ARINC de acuerdo a la clasificación mencionada anteriormente, asociadas al tipo de sistemas en el cual se aplican (redes de aeronaves, aeronaves digitales y simuladores de vuelo, o aeronaves analógicas y simuladores de vuelo)

TABLA I. CLASIFICACIÓN DE SERIES DE ESTÁNDARES ARINC

	Red de aeronaves	Aeronave Digital Simulador	Aeronave Analógica Simulador
Características	Serie 700	Serie 700	Serie 500
Especificaciones	Serie 800	Serie 600 Serie 400	Serie 400
Reportes	Serie 800	Serie 600 Serie 400	Serie 400

El estándar ARINC 429 [1][2] [1] establece una descripción técnica del bus de comunicación en equipos de aviación. El bus se encuentra incorporado en la mayoría de las empresas aéreas comerciales y de transporte [4]. En su descripción, detalla las interfaces físicas y eléctricas de un bus *two-wire*, y el protocolo de datos que debe soportar una red local de una aeronave.

Durante las fases de desarrollo o mantenimiento de equipos con interfaces ARINC 429, es necesario contar con



instrumental específico que permita el análisis de los paquetes transmitidos por el bus. Asimismo, es recomendable la utilización de decodificadores o analizadores que permitan verificar que los valores contenidos sean correctos. Los analizadores poseen capacidad de recolectar, analizar, decodificar o almacenar información.

Existen soluciones comerciales que atacan este problema, pero el grado de flexibilidad de conversión soportado es escaso. En la siguiente tabla se presenta el estudio de mercado de algunos proveedores de conversores ARINC429 a RS232 o USB, detallando números de parte y costos (expresados en US\$ FOB).

TABLA II. CONVERSORES ARINC 429 COMERCIALES

Fabricante	P/N	Costo US\$ FOB
Nginuity [5]	AP9113 (USB)	2251
RTX System [6]	RTXSPA429	1595
Shadin [7]	LP 933612-02	2861
Shadin [7]	LP 933614	1454
Yed Limited [8]	A429R1-SERIAL-T1	3243

Aunque los costos de los productos presentados no son excesivos, surge la oportunidad de desarrollar capacidad y valor agregado a partir de la flexibilidad y reconfiguración en distintos tipos de salidas y protocolos que se obtienen. Esto finalmente redundaría en la sustitución de importaciones, posibilitando productos de tecnología de bandera argentina.

En este trabajo se detalla el desarrollo de una plataforma genérica configurable, junto con el diseño de un IP Core de recepción/transmisión ARINC 429, tal como los ofrecidos comercialmente [9][10][11]. El objetivo de este desarrollo es ofrecer una alternativa para la importación, a un costo menor y además, brindando capacidades de interfaz personalizable (USB, RS232, RS485, RS422, Señal Sincrónica, Señales Resolver, MIL-STD 1553).

El uso de un dispositivo reconfigurable como procesador principal de la plataforma, provee la flexibilidad suficiente para ajustarse a las necesidades del usuario sin impacto en el hardware. Asimismo el IP Core posee una interfaz digital, que posibilita la integración de este con diversos IP Cores de comunicación.

En la sección II, se presenta las características principales del estándar ARINC 429, la sección III detalla al IP Core desarrollado y la plataforma utilizada para la evaluación. La sección IV los resultados experimentales del desarrollo y finalmente las conclusiones del trabajo en la sección V.

II. ESTÁNDAR ARINC 429

En la norma [12][13][14] se detallan características del hardware y formato de datos. El bus está compuesto por un par trenzado, que permite establecer una comunicación simple de punto a punto entre equipos de aeronaves comerciales y de transporte. Los paquetes poseen una longitud de 32 bits y son transmitidos a una frecuencia de 12,5 o 100 kbps en forma serial utilizando una modulación bipolar con retorno a cero (BPRZ) como se presenta en la figura 1.

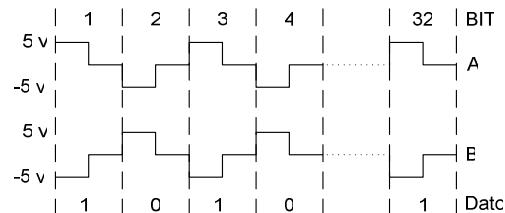


Figura 1. Ejemplo de codificación de bits ARINC 429

Los paquetes pueden ser binarios (BNR), codificación decimal (BCD) o en formato ASCII y están organizados en 5 campos (Fig. 2): Paridad, SSM, datos, SDI y Label. Los bits se encuentran numerados de 1 (LSB) a 32 (MSB) por convención.

32	31	30	29	1	10	9	8	1
P	SSM	DATA MSB	DATA MSB	PAD	DISCRETES LSB	SDI	LABEL	

Figura 2. Organización de datos ARINC 429

El bit más significativo del paquete se corresponde con la paridad, generalmente impar. Los primeros ocho bits se corresponden con la etiqueta del paquete que permite asociar a los datos con el tipo. El identificador del paquete se corresponde con el valor contenido en este campo en base octal. El campo SSM (Sign/Status Matrix) contiene información acerca de la condición del equipo, validez de datos o modo operacional. Los datos se hallan presentes entre los bits 29 y 11, y su formato depende del tipo de paquete antes mencionado. El ancho de bits de los datos puede ser superior, llegando a 20 bits, si se utilizan los bits del campo SSI, dejando a este campo sin uso. Los bits 30 y 31, en el caso de utilizarse se corresponden con el identificador de origen/destino (Source/Destination Identifier or SDI). Se utiliza para identificar el receptor del paquete para el cual está destinado. Asimismo puede ser usado para identificar el origen de la transmisión.

El orden de transmisión de los paquetes es el siguiente: 8, 7, 6, 5, 4, 3, 2, 1, 9, 10, 11, 12, 13 ... 32.

III. DESARROLLO

El desarrollo del IP Core está descrito en VHDL y comprende dos módulos (recepción/transmisión) que implementan la conversión de datos entre una interfaz ARINC 429, y una digital. Estos módulos pueden ser acoplados a la interfaz de salida/entrada deseada.

El componente de recepción se encuentra formado por un subcomponente encargado de realizar el procesamiento del paquete recibido (ARINC_RX), un subcomponente que la compara el paquete recibido y los paquetes que son de interés (LABEL_MATCH), y una FIFO que almacena los datos para su posterior lectura (ARINC_FIFO). En la figura 3 se presenta un diagrama simplificado en el cual se omiten los puertos de entrada/salida, así como también las conexiones de señales de reloj y reset.

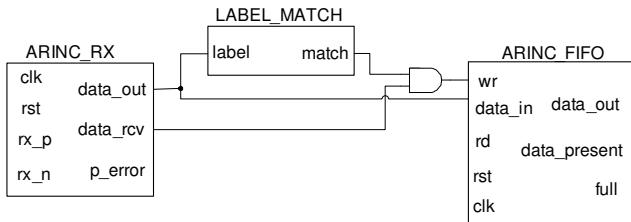


Figura 3. Módulo de recepción ARINC 429

El módulo ARINC_RX está descrito como una máquina de estados, la cual se activa ante la presencia de un inicio de paquete. Una vez finalizada la transmisión se controla la validez de la paridad del paquete y activa la señal *data_rcv*. La frecuencia de transmisión (12.5 o 100 kbps) junto con la frecuencia de operación del sistema son utilizados como constantes en el código fuente, adaptándose a la plataforma/uso.

El componente LABEL_MATCH está implementado de manera combinacional mediante una ROM la cual permite identificar si el paquete recibido forma parte del conjunto de paquetes que se desean retransmitir. La señal de salida estará activa cuando esta correspondencia se cumpla.

La señal de indicación del paquete junto con la señal *match*, activan la escritura del componente ARINC_FIFO, el cual almacena la cantidad deseada de paquetes (definida en etapa pre-síntesis) de 32 bits. Esta estructura informa la presencia de datos (*data_present*), así como también cuando se ha alcanzado la capacidad máxima (*full*) de la memoria. Los datos pueden ser extraídos de la estructura mediante una activación en la señal de lectura (*rd*).

Por otra parte, el módulo de transmisión (Fig. 4) se compone de un subcomponente que está implementado como otra instancia del componente ARINC_FIFO, que almacena los paquetes a ser transmitidos, y un componente encargado de realizar la transmisión del dato. La transmisión es llevada a cabo con la activación de la señal de presencia de datos de la unidad ARINC_TX, implementada como una máquina de estados que se activa ante una solicitud de envío. No se requiere un cálculo anticipado de la paridad del paquete a transmitir, debido a que el módulo cuenta con la lógica necesaria para determinar el valor de este campo.

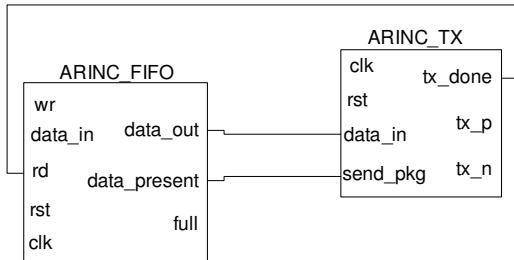


Figura 4. Módulo de transmisión ARINC 429

Al igual que en el componente de recepción, la frecuencia de operación y la de transmisión deben ser definidos en tiempo de pre-síntesis. Una vez finalizada la transmisión, se activa una señal que indica el fin de la operación, y esta es utilizada por el

buffer de paquetes ARINC, para permitir la lectura del próximo paquete.

Para la evaluación del Core ARINC 429 se desarrolló una placa prototipo (Fig. 5) que cuenta con un dispositivo de lógica programable (FPGA Spartan 3 de Xilinx - XC3S250E-4VQ100C) como dispositivo primario en la cara superior. Además cuenta con un microcontrolador de 32 bits (ARM Cortex-M3 LPC1768) y una memoria flash de 8 Mbyte en la cara inferior. La plataforma prototipo posee interfaces de comunicación USB, RS232/RS422, conversores digitales/analógicos y ARINC 429.

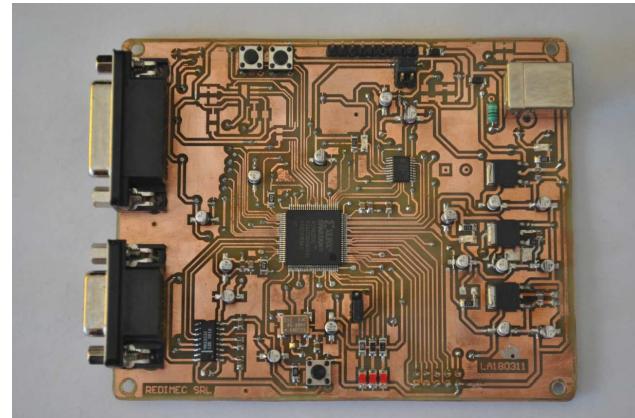


Figura 5. Prototipo de placa de conversor ARINC-429

La comunicación entre la FPGA con el microcontrolador está implementada mediante una conexión SPI (Serial Peripheral Interface). La memoria flash puede ser accedida tanto por el FPGA como por el microcontrolador. Esta característica brinda la posibilidad de almacenar datos que provienen desde un bus ARINC 429, y permite realizar el *logging* de información de vuelo que son generados por una unidad inercial, GPS y otros dispositivos.

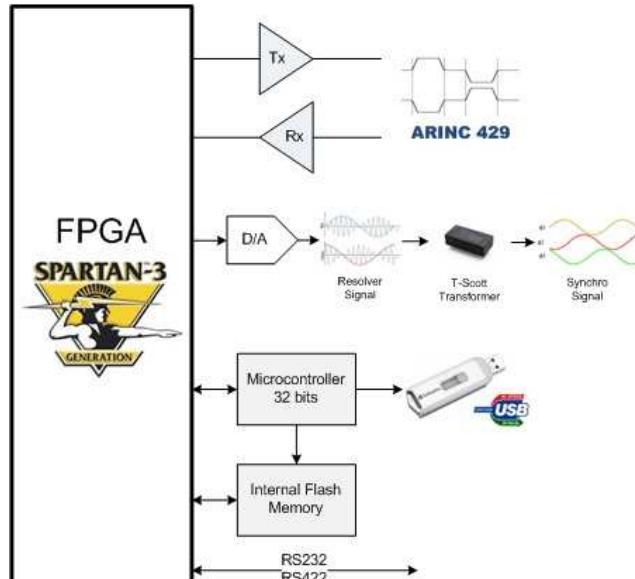


Figura 6. Diagrama en bloques del sistema



El hardware está desarrollado presentando atributos de flexibilidad. El mismo permite implementar variantes y soluciones a una amplia gama de dispositivos, como por ejemplo conversor de ARINC 429 a RS232/RS422/USB, almacenamientos de datos y la posibilidad de descarga a través de un dispositivo de almacenamiento masivo. En la figura 6 se presenta un esquema general del sistema.

En la industria aeronáutica conviven sistemas de naturaleza digital y analógica. Esta coexistencia hace imperiosa la adaptación para generar el menor impacto posible en la instalación de equipos en una aeronave. Es por ello que surge la necesidad de la conversión de señales digital (ARINC 429) a señales de sincrónicas (11.8V 400Hz) de acuerdo al estándar ARINC 407-1. Asimismo también es posible la conversión de señales sincrónica a digital.

A continuación se enumera las ecuaciones (1) para describir la naturaleza de las señales del tipo resolver sin considerar desplazamiento de fase [15].

$$\begin{aligned} V_x &\propto \sin \theta \sin \omega t \\ V_y &\propto \cos \theta \sin \omega t \end{aligned} \quad (1)$$

Siendo $\omega = 2\pi f$, f la frecuencia de referencia (26V 400Hz en aviónica) y θ el ángulo a convertir.

Normalmente, los instrumentos analógicos son excitados con señales sincrónicas. Para tal fin se utiliza un transformador T-Scott cuyo propósito es adaptar las señales resolver a sincrónicas. La terna de ecuaciones (2) se describe a continuación:

$$\begin{aligned} V_{s1} &\propto \sin \theta \sin \omega t \\ V_{s2} &\propto \sin(\theta - 120^\circ) \sin \omega t \\ V_{s3} &\propto \sin(\theta - 240^\circ) \sin \omega t \end{aligned} \quad (2)$$

En la figura 7 se muestra el resultado de la simulación ECAD para el diseño emplazado y ruteado de la plataforma final desarrollada. La misma cuenta con filtrado LC en 28VDC y conversores DC-DC para generar las alimentaciones del sistema.



Figura 7. Simulación de placa de conversora ARINC-429

Posee dos canales independientes de ARINC 429 con la posibilidad de recepción y transmisión de información en 12,5 kbps y 100 kbps.

El banco de memorias Flash tiene una configuración de 64 Mbyte que permite almacenar hasta tres horas de datos ARINC 429 en 100 kbps, considerando 80 paquetes distintos con frecuencias entre 20 y 100 Hz.

Como interfaz de comunicación cuenta con transceivers de RS422, RS232 y USB como UART. El USB del microcontrolador se utiliza como USB Host. Esto permite conectar un memory stick y acceder de manera simple a los datos del vuelo. Los datos son utilizados para debriefing, posibilitando el análisis de la navegación en tierra. El USB también puede ser utilizado con otros fines dependiendo de la configuración deseada por el cliente (CDC, Communication Data Class; HID, Human Interface Device, Bulk, etc.).

Un conversor digital analógico de dos canales, con una resolución de 16 bits, permite conformar las ondas moduladas para una señal resolver. Finalmente, se rutearon pines del FPGA y del microntrolador a conectores genéricos de alimentación, para futuras ampliaciones de configuración .

En la figura 8 se muestra el renderizado del producto final RED0100A-XX (Redimec S.R.L.). El producto presentado se adapta a una gran posibilidad de configuraciones:

- Conversor ARINC 429 - RS232, RS422, USB.
- Conversor NMEA 0183 - ARINC 743A (GPS).
- Conversor ARINC 429 a ARINC 407-1.
- ARINC 429 datalogging.

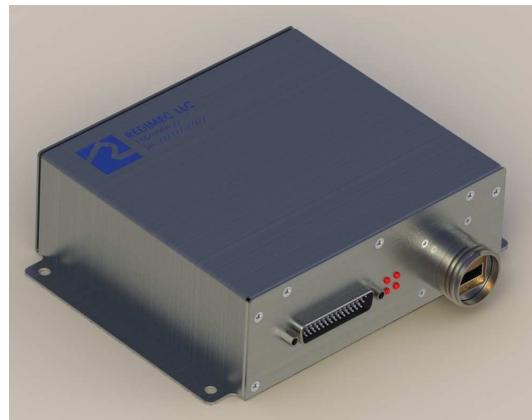


Figura 8. Renderizado del producto final conversor ARINC 429

IV. RESULTADOS EXPERIMENTALES

El core de recepción/transmisión ARINC 429 fue implementado sobre un FPGA Spartan 3 de Xilinx. La síntesis se realizó utilizando el XST (Xilinx Synthesis Technology), mientras que la implementación física se llevó a cabo mediante la herramienta ISE de Xilinx (Integrated Software Environment) versión 12.2.

El core posee un retardo máximo de pista de 8.240 ns, con lo cual acepta frecuencia de reloj máxima de entrada de 121 MHz. Los detalles del costo en área se muestran en la tabla III.

TABLA III. COSTO EN ÁREA DEL CORE RECEPTOR/TRANSMISOR

Slices	195
LUTs de 4 entradas	351
Usadas como lógica	281
Usadas como ruteo	70
RAMB16s	2

En los datos presentados sólo se muestran los resultados obtenidos de la implementación del módulo recepción y transmisión. Los costos asociados al protocolo al cual se desea convertir se omiten debido a que dependen de la aplicación requerida. Asimismo, los buffers tanto de entrada como de salida se configuraron con un tamaño máximo de 64 paquetes. El componente de LABEL_MATCH fue excluido de la síntesis de manera que todos los paquetes recibidos fuesen retransmitidos. Se observa que el diseño propuesto es compacto y no insume excesivos recursos del dispositivo programable.

Tanto el diseño como la plataforma fueron evaluados utilizando un banco de análisis que provee soporte ARINC 429. La consistencia de los datos en la recepción ARINC, se evaluó a partir de la generación de paquetes de datos desde el BUS Master 2500 [16]. Estos datos fueron convertidos a RS232, y analizados en una PC. De forma análoga se realizó el testeo en cuanto a la conversión a ARINC 429. Diversos paquetes de datos fueron generados por una PC, y se analizó la correcta conversión en el banco de análisis (Fig. 9).

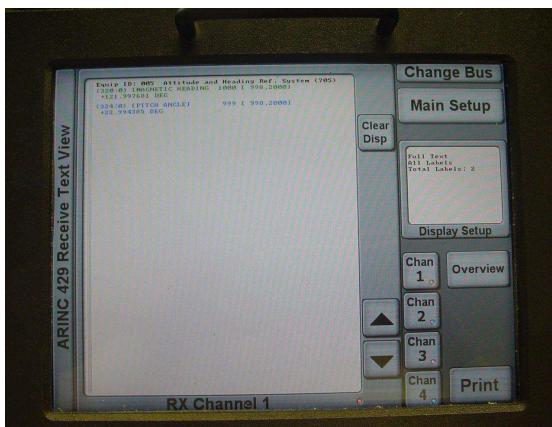


Figura 9. Evaluación de conversión a ARINC 429

La verificación comportamental se llevó a cabo variando las condiciones de transmisión, los test incluyeron a transmisiones intensivas (12,5 y 100 kbps), realizando barridos sobre el conjunto de paquetes soportado y rampas de valores para los datos contenidos en los paquetes. Las pruebas se realizaron tanto para la conversión ARINC 429- RS232, como RS232 – ARINC 429, Los resultados en todos los casos reportaron un 0% de pérdida de información.

Se comprobó además la correctitud en la forma de onda de las salidas generadas por el dispositivo programable, utilizando para esto un osciloscopio digital. Este análisis permitió determinar la correcta frecuencia de transmisión, así como también una presencia poco apreciable de ruido en la línea (Fig. 10).

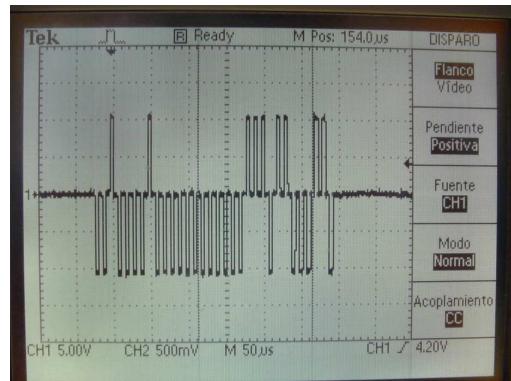


Figura 10. Análisis de canal de comunicación ARINC 429

La plataforma será homologada bajo las normas ambientales DO-160D [17] y el software fue descrito siguiendo los lineamientos de la DO-178 [18]. De esta manera se pretende que el producto sea un equipo certificado de aeronáutica.

V. CONCLUSIONES

En este trabajo se presenta un producto argentino para la conversión desde el estándar ARINC 429 y sus variantes, utilizando como unidad principal de procesamiento un FPGA Spartan 3 de Xilinx. Para cumplir el objetivo se implementó un IP Core de recepción/transmisión que cumple con el estándar. El IP Core permite acoplarse con otros de comunicación (USB, RS232, RS485, RS422, Señal Sincrónica, Señales Resolver, MIL-STD 1553) desarrollados.

El producto brinda capacidades tanto para el funcionamiento normal, mantenimiento, como para el desarrollo de electrónica de a bordo, logrando una sustitución de importación e independencia respecto a los productos provenientes principalmente del exterior.

La utilización del FPGA como módulo de procesamiento permite el desarrollo y personalización *ad hoc* de una nueva línea de productos como ser módulo de logging de datos en vuelo, conversor de múltiples canales ARINC-429, señales sincrónicas, todo esto referido al hardware. Sin embargo, también se genera un producto adicional en la posibilidad de exportación de software del IP Core desarrollado.

En definitiva, el desarrollo permite ser un producto a medida como característica diferencial, ya sea en la interfaz de comunicación que se requiera y paquetes de interés, así como en la diversidad de sus salidas. Estas características permite al usuario final del producto establecer cuáles son sus necesidades y para tal fin, generar una solución. El objetivo es proveer un producto argentino para satisfacer el mercado interno y con proyección internacional, cuya fortaleza sea una flexibilidad superior que los dispositivos importados que compiten en el mismo segmento.

REFERENCIAS

- [1] ARINC Specification 429P1-17 Mark 33 Digital Information Transfer System (DITS), Part 1, Functional Description, Electrical Interface, Label Assignments and Word Formats, 2004.



- [2] ARINC Specification 429P2-16 Mark 33 Digital Information Transfer System (DITS), Part 2 - Discrete Data Standards, 2004.
- [3] Steve Woodward (July 11, 2002). Bill Travis. ed (PDF). Circuit transmits ARINC 429 data. EDN Magazine.
- [4] Holt Integrated Circuits, ARINC 429, <http://www.holtic.com/category/352-arinc-429.aspx>
- [5] Nginuity Ltd., ARINC-429 to USB Converter Module (AP9113), http://www.nginuity.com/page/Nginuity_Products/AP-SERIES/AP9113/
- [6] RTX Systems, 4 Channel ARINC 429 Serial Port Adapter, <http://www.rtxsystems.com/SPA.html>
- [7] Shadin Avionics, Legacy converters, <http://www.shadin.com/products/converters/legacy.html>
- [8] Yeoved Electronics Components, A429R1-SERIAL-T1 ARINC429 to RS232/RS422 Converter (\$GPRMC & \$GPGGA), <http://www.jtelec.fr/dl/YED/A429R1-Serial-T1.pdf>
- [9] ACTEL Corporation, ARINC 429 Bus Interface, www.actel.com/ipdocs/CoreARINC429_DS.pdf
- [10] ARINC 429 IP Core for FPGAs, http://www.sitaltech.com/pdf/ARINC429_Brochure_1109.pdf
- [11] Digeratti Systems, Digi-ARINC-IP protocol core Technical Data Sheet, <http://digidigitate-systems.net/>
- [12] Alta Data Technologies LLC, "ARINC Protocol Summary", www.altadt.com.
- [13] AIM GmbH, "ARINC 429 Specification Tutorial", Nov 2010 v2.1, www.aim-online.com
- [14] SBS Technologies, Incorporated, "ARINC 429 Commentary", <http://www.resource.sbs.com>
- [15] Synchro/Resolver Conversion Handbook. DDC. Fourth Edition (Electronic Version).
- [16] AE Test Solutions, Inc. Bus Master 2500, <http://www.aeroexpress.com/products/pdfs/busmaster2500.pdf>
- [17] RCTA Inc, "DO-160E, Environmental Conditions and Test Procedures for Airborne Equipment", September, 2004
- [18] RCTA Inc, "DO-178B, Software Considerations in Airborne Systems and Equipment Certification". January, 1992.



Memory-mapped I/O aprovechando las Memorias Dual Port BRAM de una FPGA

Rodrigo A. Melo, David M. Caruso, Salvador E. Tropea
Laboratorio de Desarrollo Electrónico con Software Libre
Centro de Electrónica e Informática
Instituto Nacional de Tecnología Industrial
Buenos Aires, Argentina
Email: {rmelo,david,salvador}@inti.gob.ar

Resumen—En la actualidad, *Direct Memory Access* (DMA) es uno de los mecanismos más utilizados para transferir datos entre un procesador y sus periféricos. Otra posibilidad es mapear los periféricos directamente en el espacio de memoria del procesador, lo cual tiene la desventaja de precisar memorias de doble puerto cuando el dispositivo maneja grandes cantidades de datos. Este es el caso típico de tarjetas de video y red. En este trabajo proponemos el uso de memorias *dual port* BRAM, normalmente disponibles en FPGAs modernas, para implementar un *core* utilizando *Memory mapped I/O* (MMIO). Como caso de estudio, presentamos el desarrollo de un *core* microcontrolador AVR que incorpora un *Media Access Controller* (MAC) Ethernet, capaz de correr el *stack* TCP/IP uIP, con un *web server* como aplicación de ejemplo. Adicionalmente, se discuten las ventajas de mover el programa del microcontrolador a una memoria externa que usa el estándar *Common Flash Interface* (CFI). Este diseño fue simulado con herramientas de Software Libre y verificado en hardware utilizando una FPGA Virtex 4 de Xilinx.

Index Terms—MMIO, DMA, BRAM, FPGA, AVR, uIP, CFI, MAC, Ethernet, Web server.

I. INTRODUCCIÓN

Direct Memory Access es un mecanismo ampliamente utilizado para transferir datos entre memorias, procesadores y sus periféricos. Esto permite altas tasas de transferencia de datos y velocidades de operación, liberando de tareas al procesador, a expensas de requerir un controlador el cual es responsable de arbitrar los accesos a los buses.

Otro mecanismo muy útil es llamado *Memory-mapped I/O*, en el cual las entradas y salidas de un periférico se mapean directamente en el espacio de memoria del procesador. Las memorias *dual port* son adecuadas para su implementación. Esta opción era frecuentemente utilizada en placas de video, pero debido a los altos costos de implementación fue remplazada por el uso de SDRAMs.

En este trabajo se propone aprovechar las memorias *dual port* BRAM, comúnmente disponibles en FPGAs modernas, para implementar sistemas con MMIO en lugar de DMA. Como caso de estudio de la metodología propuesta, se interconectaron dos *cores* previamente desarrollados en nuestros laboratorios: un microcontrolador AVR [1] y un controlador MAC Ethernet [2]. Para testear la arquitectura, implementamos un *web server*, utilizando el *stack* TCP/IP llamado uIP [3], el cual es Software Libre. Adicionalmente, se realizaron optimizaciones tanto en *hardware* como en *firmware* para reducir la

cantidad de BRAMs utilizadas.

Este trabajo se estructura de la siguiente manera: en la sección II se realiza una breve discusión acerca de MMIO y la arquitectura propuesta. La sección III presenta el *core* desarrollado como caso de prueba mientras que la sección IV contiene información acerca del *stack* TCP/IP utilizado, las adaptaciones realizadas para que funcione sobre el microcontrolador AVR y algunas herramientas de Software Libre utilizadas. Las optimizaciones en el uso de BRAMs son detalladas en la sección V. Las secciones VI y VII describen las pruebas realizadas y los resultados obtenidos respectivamente. Finalmente, las conclusiones son expuestas en la sección VIII mientras que la sección IX propone trabajo futuro a realizar.

II. METODOLOGÍA PROPUESTA

II-A. Background

Memory-mapped I/O es un método para llevar a cabo transferencias de entrada/salida entre un procesador y sus periféricos. El bus de direcciones se utiliza para direccionar tanto memorias como dispositivos, por lo cual las instrucciones utilizadas por el procesador son las mismas para acceder a uno u otro. Esto simplifica el diseño del sistema y conlleva un hardware más sencillo y rápido, una ventaja particularmente útil en sistemas embebidos. En síntesis, para acceder a un dispositivo el procesador simplemente realiza operaciones de lectura o escritura dentro del espacio de direcciones asignadas.

Para implementar MMIO cada dispositivo precisa una memoria de doble puerto. En un circuito impreso no es una solución viable, dado los costos y el área de ocupación en la placa. Sin embargo, las *dual port* BRAMs están normalmente disponibles en las FPGA de hoy en día, con lo cual es una alternativa sencilla y sin costos extras cuando hay suficientes recursos disponibles. Debe tenerse en cuenta que una implementación como memoria distribuida no es conveniente, debido al gran consumo de área de la FPGA que esto provocaría.

Otra cuestión a recalcar es que esta metodología conviene aplicarla sobre dispositivos que transfieran al menos cientos de bytes de datos. Dichos dispositivos suelen precisar memoria para almacenar los datos, con lo cual el único recaudo es utilizar memorias *dual port* en lugar de *single port*.

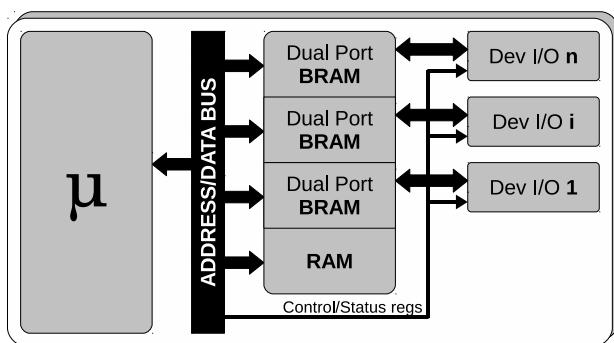


Figura 1. Arquitectura propuesta

II-B. Arquitectura

La Fig. 1 muestra un esquema simplificado de la arquitectura propuesta. En la misma, el procesador está solamente conectado al espacio de memoria donde la RAM del sistema y un puerto de cada BRAM están mapeados. El segundo puerto de cada BRAM es controlado por el periférico asociado.

En esta arquitectura el procesador puede acceder a la memoria de sistema o a la memoria del periférico con sólo especificar su dirección en el bus. El control y lectura de estado se implementa utilizando registros, mapeados en el mismo espacio de memoria o utilizando un espacio separado de I/O.

III. CASO DE ESTUDIO: EL IP CORE DESARROLLADO

III-A. Implementación

Se implementó un *core* microcontrolador AVR el cual incluye un *core* controlador MAC Ethernet. Ambos *cores* se interconectaron utilizando la metodología MMIO.

El *core* fue implementado utilizando el estándar VHDL 93, y se desarrolló con herramientas y bajo los lineamientos recomendados por el proyecto FPGALibre [4] [5]. El diagrama en bloques del microcontrolador obtenido, denominado ATeth, puede apreciarse en la Fig. 2.

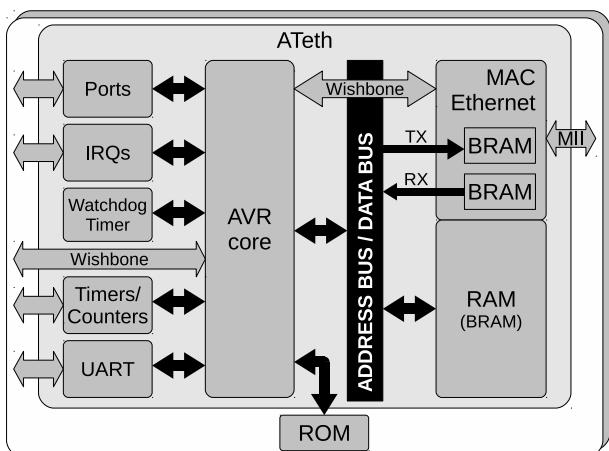


Figura 2. Diagrama en bloques de ATeth

Cuadro I
MAPEO EN MEMORIA

(a) Opción 1			(b) Opción 2		
15	14	Device	15	14	13
0	0	RAM	0	0	0
1	0	RX	1	1	0
1	1	TX	1	1	1

El *core* AVR fue instanciado con una configuración equivalente a un ATmega32. Todos sus periféricos se habilitan mediante *generics* y sus características son similares al microcontrolador presentado en un trabajo previo [1].

El *core* MAC Ethernet se modificó respecto a un trabajo previo [2] donde se usaban FIFOs y registros de control y estado. Los *buffers* de recepción y transmisión fueron mapeados en el espacio de memoria y los registros en una interfaz WISHBONE [6] esclava. El WISHBONE maestro fue implementado como bus interno en lugar de utilizar el externo ya disponible, para evitar limitaciones de direccionamiento externo y mantener compatibilidad con desarrollos previos. Por lo tanto se tienen dos maestros WISHBONE, mapeados en registros de I/O que comparten el registro de direcciones, pero poseen registros independientes para los datos.

Para una decodificación más sencilla de la memoria de sistema y los periféricos mapeados, se propuso el uso de los *bits* más significativos del bus de direcciones. Se consideraron dos posibles casos, uno utilizando los dos *bits* MSBs (cuadro I(a)) y el otro utilizando los tres MSBs (cuadro I(b)). La primer opción tiene un problema: en la arquitectura AVR, los primeros 96 bytes se utilizan para los *General Purpose Registers File* y los *I/O Registers*, y a continuación se ubica la RAM interna. El *stack pointer* se inicializa apuntando a la última posición de memoria, la cual por ejemplo en un ATmega32 es 2K+96. Si el bit 15 es utilizado para seleccionar la RAM, habrá disponibles 32KB, pero esto puede provocar el solapamiento entre el *stack pointer* y el canal de recepción que se ubica a partir de la dirección 32K. Para evitar este problema, se adoptó la segunda alternativa.

III-B. Modo de uso del core MAC Ethernet

Los registros WISHBONE del *core* MAC Ethernet pueden apreciarse en la Fig. 3. El registro 0 se utiliza para configuración y control. Si los *bits* TXerr o RXerr están en 1 tras una transferencia, significa que hubo un error y el detalle del mismo se puede obtener del registro 1. Los registros 2 y 3 se utilizan en conjunto para indicar el número de *bytes* recibidos (lectura) o a transmitir (escritura). Sólo se utilizan 11 *bits* (2KB) debido a que el tamaño máximo de un *frame* Ethernet es 1514 *bytes*.

Un *frame* Ethernet consta de 6 *bytes* para la dirección MAC destino, 6 *bytes* para la dirección MAC origen, 2 *bytes* para el campo *length/type*, la información contenida y finalmente el CRC, el cual es automáticamente agregado en transmisión y descartado en recepción.

- Configuración: los *bits* TXen, RXen, Full y Prom del

7	6	5	4	3	2	1	0	
RXerr	RXctrl	TXerr	TXctrl	Prom	Full	RXen	TXen	R0: Control
RD	R/W	RD	R/W	R/W	R/W	R/W	R/W	
7	6	5	4	3	2	1	0	
RXfo	RXlen	RXftl	RXfts	RXcrc	Rxalg	TXfo	TXal	R1: Status
RD	RD	RD	RD	RD	RD	RD	RD	
7	6	5	4	3	2	1	0	
-	-	-	-	-	Len10	Len9	Len8	
Len7	Len6	Len5	Len4	Len3	Len2	Len1	Len0	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Figura 3. Registros del core MAC Ethernet

- registro de Control son respectivamente utilizados para habilitar/deshabilitar los canales de transmisión y recepción, el modo *full duplex* y el modo promiscuo.
- Transmisión: para determinar si en canal esta listo para una nueva transmisión, el usuario debe asegurarse que el *bit* TXctrl este en 0. Los datos a ser transmitidos deben copiarse a la memoria de transmisión y el número de *bytes* involucrados debe ser especificado en los registros 2 y 3. La transmisión comienza cuando se coloca un 1 en el *bit* TXctrl. Si ocurre un error, el *bit* RXerr se coloca en 1 y puede obtenerse información detallada de los *bits* 0 y 1 del registro de Estado.
 - Recepción: habrá datos disponibles cuando el *bit* RXctrl este en 0. La cantidad de datos recibidos se obtiene de los registros 2 y 3. Si ocurrió un error, el *bit* RXerr estará en 1 y los detalles del mismo pueden obtenerse de los *bits* 2 a 7 del registro de Estado. Finalmente, es necesario indicar que se procesaron los datos, colocando un 1 en el *bit* RXctrl.

IV. FIRMWARE

IV-A. El stack TCP/IP

Se realizó una búsqueda de *stacks* TCP/IP que tuvieran bajos requerimientos de *hardware* y una licencia libre. Se seleccionó el *stack* denominado uIP versión 1.0, el cual en la actualidad forma parte del sistema operativo Contiki [7]. Este *stack* está escrito en lenguaje C, provee conectividad TCP/IP a microcontroladores de 8 *bits* y es compatible con los documentos RFC correspondientes. Como aplicación de pruebas se seleccionó un *web server* sencillo, utilizado para servir 4 páginas diferentes.

Las siguientes características hicieron a uIP adecuado para nuestro proyecto:

- Posee muy pocos requerimientos de *hardware*.
- Es utilizado ampliamente en sistemas embebidos.
- Había una versión implementada con un AVR, por lo cual se sabía con antelación que probablemente funcionaría en ATeth.
- Es Software Libre, con lo cual se puede usar, modificar y distribuir sin la necesidad de pagar regalías.
- El código fuente es estructurado y modular.

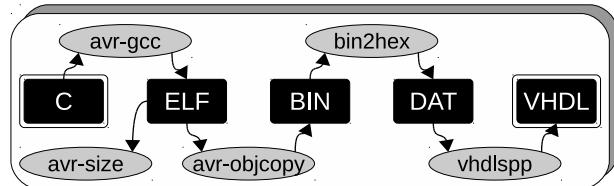


Figura 4. Obtención de la memoria de programa

- Está bien documentado.
- Hay ejemplos disponibles, incluido el *web server* seleccionado.

IV-B. Adaptaciones

Para utilizar uIP con ATeth, fue necesario escribir un controlador para el *core* MAC Ethernet. Básicamente, se implementaron funciones para inicializar, enviar y recibir datos, siguiendo los pasos descriptos en la sección III-B.

El ejemplo utilizado para el *web server* fue diseñado simulándolo con interfaces de red virtuales bajo GNU[8]/Linux. Fue necesario modificar dicho ejemplo para que funcione con una interfaz Ethernet real.

Por otra parte, los 2KB de memoria de datos del ATmega32 no eran suficientes para contener las páginas *web* embebidas. Nuestra implementación sobre FPGAs soporta la configuración del tamaño de la memoria. El único cuidado especial a tener en cuenta es mover la dirección de inicio del *stack pointer* en nuestro caso a la dirección 0x405F (16K+96 *bytes*).

IV-C. Herramientas de Software

El *core* implementado soporta el juego de instrucciones completo del procesador original y una configuración equivalente a un microcontrolador existente, por lo cual es posible utilizar herramientas de *software* disponibles, tales como el avr-gcc, una versión del compilador de C de GNU [9].

La Fig. 4 muestra la cadena de herramientas utilizada para obtener la descripción de *hardware* de la memoria de programa en base al código fuente del *web server*. Comenzando por las fuentes en C (y sus encabezados .h) se obtienen el ejecutable (elf) utilizando el avr-gcc. Luego, la memoria de programa (bin) es extraída mediante el avr-objcopy. Las herramientas bin2hex y vhdlsp fueron desarrolladas por nuestro laboratorio como parte del proyecto FPGALibre: bin2hex convierte un archivo bin en valores hexadecimales para asignaciones VHDL (dat) y luego vhdlsp (*VHDL Simple Pre Processor*) los combina con el esqueleto de una ROM, dando como resultado el archivo a utilizar junto a ATeth.

Se utilizó la herramienta avr-size en la determinación de la cantidad de memoria de programa necesaria para la aplicación resultante, y con esta información se dimensionó correctamente la memoria del sistema. La herramienta muestra la cantidad de *bytes* utilizados por las secciones *text*, *data* y *bss*. La suma de las secciones *text* (código ejecutable) y *data* (valores de las variables inicializadas) es igual al número de *bytes* utilizados por la memoria de programa, mientras que la suma de *data* y

bss (variables no inicializadas) es la cantidad de *bytes* de la memoria de datos.

V. REDUCCIÓN DEL USO DE BRAMs

Las BRAMs son recursos frecuentes en una FPGA, pero finitos, por lo cual deben ser utilizados con moderación. Para optimizar su utilización, se analizaron diferentes alternativas, realizando cambios tanto en el *hardware* como en el *software*.

V-A. Colocación de la ROM en una memoria flash paralela

Se observó que gran parte de las BRAMs eran utilizadas como memoria ROM por el AVR, debido a que el código del *stack* TCP/IP era de aproximadamente 20KB. Se decidió mover la ROM del microcontrolador a una memoria *flash* paralela externa, preexistente en el *kit* de desarrollo utilizado. La misma contaba con interfaz CFI [10], un estándar abierto desarrollado por AMD, Fujitsu, Intel y Sharp, el cual puede proveer información a través de secuencias predefinidas de comandos. Esta información incluye detalles tales como el tamaño de la memoria, tiempos de borrado, voltajes utilizados, etc. Existen muchos tipos de comandos para dialogar con las memorias CFI, cada uno definido por su fabricante. Los mas comunes son *AMD/Fujitsu Command Set* e *Intel/Sharp Command Set*. Nuestra memoria CFI utilizaba el segundo.

Para trabajar con esta clase de memorias, nuestro laboratorio desarrolló un *core* controlador CFI. En resumen, es una *Finite State Machine* (FSM) en *hardware*, que permite leer y escribir la memoria *flash* con comandos enviados a través del USB de una PC. Los pasos necesarios para cambiar el programa de ATEth son: primero, la FPGA tiene que ser configurada con el controlador CFI. Luego, la memoria *flash* es grabada mediante USB. Finalmente, la FPGA se configura nuevamente, pero esta vez con ATeth.

La memoria utilizada tiene un tiempo de lectura de página mayor al tiempo de acceso a datos. Dado que ATeth no puede insertar estados de espera por sí mismo, debía leer la memoria a una velocidad compatible a los cambios de página, lo cual implicaba trabajar a menor frecuencia que en el caso previo a utilizar la *flash* externa. Para mejorar el rendimiento durante operaciones de lectura, se utilizó el *core Flash ROM Interface* [11]. El mismo, inserta estados de espera al microcontrolador cuando se está efectuando un cambio de página, y el resto del tiempo permite extraer los datos a la máxima velocidad. La Fig. 5 muestra un esquema de conexión entre el microcontrolador, el *core Flash ROM Interface* y la memoria *flash*.

V-B. Optimizaciones del firmware

El *firmware* del uIP fue desarrollado para ser independiente del dispositivo donde se ejecuta, por lo cual, no posee consideraciones de *hardware* particulares.

El AVR usa una arquitectura Harvard modificada donde el programa y los datos son almacenados en espacios separados de memoria, los cuales utilizan diferentes espacios de direcciones. Por defecto, las instrucciones toman los datos desde la RAM.

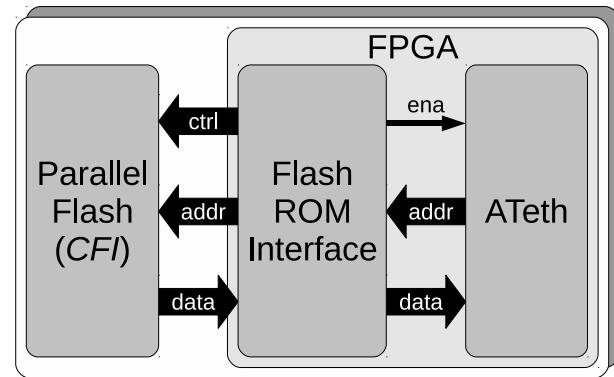


Figura 5. Interconexión entre ATeth, Flash ROM Interface y la memoria

Las cadenas constantes de datos, son almacenadas en la ROM y copiadas a la RAM antes de pasar el control a la función *main()*. Esta redundancia de datos requiere mas memoria de la realmente necesaria.

El compilador gcc provee un mecanismo especial para evitar esta duplicación. Esto se logra mediante el uso de atributos, una extensión del estándar ANSI C. Como efecto secundario, el código debe copiar la constante en la memoria RAM antes de llamar a la función que la utiliza. Un *buffer* es utilizado para mantener sólo una de las constantes necesarias en RAM.

Se aplicaron estos atributos principalmente sobre las constantes que representan las páginas web, las cuales son una gran cantidad de datos, y se realizaron algunas modificaciones al código del uIP, basadas en una adaptación a un AVR [12].

VI. TESTEO Y VALIDACIÓN

VI-A. Banco de pruebas

Para la simulación del *core* se utilizó GHDL [13] 0.29. Se realizó un banco de pruebas muy sencillo para probar la interconexión entre el AVR y el MAC Ethernet:

- El programa responde paquetes de tipo ARP (*Address Resolution Protocol*) e ICMP (*Internet Control Message Protocol*) a través de la interfase MAC Ethernet. Llamamos a este programa loopback_eth.
- Para generar estímulos, se usaron programas originalmente desarrollados para el proyecto del MAC Ethernet. Se generaron dos archivos en base a datos aleatorios: el primero con datos de recepción y el segundo con los correspondientes a la transmisión.
- El banco de prueba inyecta datos del archivo de recepción y compara la salida obtenida con los datos del archivo de transmisión, abortando en caso de detectarse diferencias.

VI-B. Validación en hardware

Este desarrollo se validó utilizando una FPGA Virtex 4 (XC4VLX25) de Xilinx y el *software* ISE WebPack 11.3 - L57. La memoria CFI externa es una Intel Strata Flash TE28F640. Se utilizaron computadoras personales corriendo el sistema operativo Debian [14] GNU/Linux.



Cuadro II
RESULTADOS DE LA SÍNTESIS

(a) El IP core desarrollado

Caso	FFs	LUTs	Slices	Fmax (MHz)	BRAMs	PING (ms)	Text	Data	bss	ROM	RAM
1	739	2558	1486	62	27	0,25	13640	6774	7045	20414	13819
2	736	2411	1417	56	11	0,74					
3	754	2567	1499	61	11	0,32					
4	727	2612	1488	61	5	0,32					

(b) Comparación con otros cores

Caso	Procesador	Tipo	Slices	GCLKs	BRAMs
xapp433	MicroBlaze	Softcore	3737	5	8
xapp434	PowerPC	Hardcore	4383	5	12
ATeth	AVR	Softcore	1488	3	5

La primer prueba consistió en utilizar el programa *loopback_eth* sobre el AVR. Se comprobó el funcionamiento del sistema mediante el envío y recepción de paquetes ICMP (comando *ping*), y luego se analizaron los paquetes intercambiados entre la PC y el *core* con el programa *wireshark* [15].

Las siguientes pruebas incluían el programa uIP. Realizamos un programa que descarga las páginas *web* servidas y las compara con la versión de desarrollo. Las 4 páginas *web* servidas fueron también navegadas para detectar la ocurrencia de fallos y finalmente, se volvió a probar el comportamiento frente al comando *ping*, pero esta vez en *hardware*.

VII. RESULTADOS

VII-A. Síntesis

El cuadro II(a) muestra los resultados de la síntesis de ATeth para cada modificación y mejora descriptas en la sección V. Adicionalmente, se muestra en el mismo cuadro la cantidad de memoria utilizada en cada caso. En el caso 1, la ROM está implementada sobre BRAMs, mientras que en el caso 2 y los siguientes, se encuentra en la memoria *flash* externa (CFI). En los casos 1, 3 y 4, la frecuencia de operación fue 50 MHz, la cual corresponde a uno de los relojes disponibles en la placa de evaluación de Virtex 4 utilizada. En el caso 2, el sistema corre a 8 MHz utilizando un *Digital Clock Manager* (DCM), debido a que la memoria *flash* tiene una demora de 120 ns cuando realiza un cambio de página.

Comparando los casos 1 y 2, el consumo de BRAMs disminuyó en un 60 % y los *slices* tuvieron una ligero decremento del 6 % cuando la ROM se movió a la *flash*. Por otro lado, el tiempo de respuesta frente al comando *ping* se triplicó. Por lo tanto, en el caso 2 se consume menos *hardware* pero el tiempo de respuesta es peor.

En el caso 3 se utilizó el *Flash ROM Interface* para mejorar los tiempos de acceso a la *flash*. En comparación al caso 1, los resultados dan un área y tiempo de respuesta similares, pero las BRAMs utilizadas se mantienen estables respecto al caso 2.

Finalmente, el caso 4 corresponde a optimizaciones en el *firmware*, con la intención de reducir las redundancias en RAM. 5.5K *bytes* de las páginas *web* se movieron a la memoria de programa. En este caso la RAM se reduce 90 % y consecuentemente las BRAMs utilizadas un 55 % respecto

al caso 3 y un 80 % respecto al caso 1. La cantidad de *bytes* en ROM aumentó ligeramente un 15 % debido al uso de funciones especiales para leer las constantes desde esta memoria en lugar de la RAM.

VII-B. Otras implementaciones

Se encontraron 2 notas de aplicación [16] [17] de Xilinx, las cuales pueden ser utilizadas para contrastar algunos resultados. El sistema diseñado en estas notas de aplicación tiene las siguientes características

- Se utiliza una FPGA Virtex 4 (XC4VFX12).
- Un procesador (*softcore* o *hardcore*) es conectado a 16 Kb de BRAM.
- Se utiliza una memoria SRAM externa para contener los 3.8KB de páginas *web*.
- Se utiliza una memoria DDR SDRAM externa para el código ejecutable.
- Se conectan *cores* UARTLite, Ethernet 10/100 MAC y General Purpose I/O.
- Se utiliza el *On-chip Peripheral Bus* (OPB) para las interconexiones.
- Se utiliza el *stack lwIP* [18].

Los resultados extraídos de las notas, como así también los de este trabajo, se encuentran en el cuadro II(b). La misma muestra que nuestra implementación utiliza 2,5 veces menos *slices*, ocupa menos BRAMs y usa 3 relojes globales en lugar de 5.

VIII. CONCLUSIONES

La metodología MMIO, resultó ser de muy sencilla implementación. Se evitó la necesidad de un arbitro y memoria *cache*, necesarios para una implementación similar con DMA. Una vez adaptado el *core* MAC *Ethernet* y mapeado sus registros a *WISHBONE*, la interconexión con el AVR y manejo desde el *firmware* fueron triviales. MMIO resulta entonces como una metodología recomendada para el desarrollo rápido y ágil sobre FPGAs que cuenten con memorias *dual port* BRAM, que en la actualidad son mayoría.

Si bien no es posible realizar una comparación totalmente justa con otras implementaciones, es posible apreciar que nuestro desarrollo es más compacto. Ocupa menos de la mitad del área de la FPGA que los ejemplos de las notas



de aplicación con similares características, así como también menor cantidad de BRAMs.

La estrategia de mover la memoria de programa de las BRAM internas a *flash* externa, incidió notablemente sobre el número de BRAMs utilizadas, pero acarreo el degradamiento de la frecuencia de operación del sistema, que se reflejó en la velocidad de respuesta del *stack*, como se puede observar en las contestaciones al comando *ping*. La utilización del *core Flash ROM Interface* mejoró la velocidad de respuesta. Utilizar los *string* de datos directamente desde la ROM, disminuye el tamaño de la sección *data*, liberando así más BRAMs.

La adaptación y utilización del *stack* TCP/IP uIP y el *web server* de ejemplo sobre ATeth fueron sencillas y demostraron la versatilidad del mismo para su utilización en sistemas embebidos.

La utilización de lenguaje VHDL 93 estándar, permite que el *core* sea sintetizable en una FPGA de cualquier fabricante. La utilización de las herramientas propuestas por el proyecto FPGALibre demostró ser adecuada para un proyecto de estas características.

IX. TRABAJO FUTURO

Como trabajo futuro, sería recomendable implementar un sistema con múltiples procesadores y dispositivos, utilizando tanto MMIO como DMA, para obtener una comparación precisa de los recursos involucrados, dificultad de implementación, rendimiento, máximas frecuencias de operación, etc.

REFERENCIAS

- [1] S. E. Tropea and D. M. Caruso, "Microcontrolador compatible con AVR, interfaz de depuración y bus wishbone," in *Proceedings of the FPGA Designer Forum 2010*, Ipojuca, Brazil, 2010, pp. 1–6.
- [2] R. A. Melo and S. E. Tropea, "IP core MAC Ethernet," in *Proceedings of the FPGA Designer Forum 2011*, Córdoba, Argentina, 2011, pp. 1–4.
- [3] A. Dunkels. (2011, Oct.) uIP TCP/IP stack. Networked Embedded Systems group / Swedish Institute of Computer Science. [Online]. Available: "<http://www.sics.se/~adam/old-uip/>"
- [4] S. E. Tropea, D. J. Brengi, and J. P. D. Borgna, "FPGALibre: Herramientas de software libre para diseño con FPGAs," in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 173–180.
- [5] INTI Electrónica e Informática *et al.*, "Proyecto FPGA Libre," <http://fpgalibre.sourceforge.net/>.
- [6] Silicore and OpenCores.Org. (2011, Oct.) WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores. [Online]. Available: http://prdownloads.sf.net/fpgalibre/wbspec_b3-2.pdf?download
- [7] (2011, Oct.) The operating system for the internet of things. Cisco, Redwire LLC, SAP, SICS, and others. [Online]. Available: <http://www.contiki-os.org/>
- [8] "GNU project," <http://www.gnu.org/>, Jun. 2010.
- [9] (2009, Nov.) GCC, the GNU compiler collection. [Online]. Available: <http://gcc.gnu.org/>
- [10] JEDEC Solid State Technology Association. Common Flash Interface (CFI). [Online]. Available: <http://www.jedec.org/download/search-jesd68-01.pdf>
- [11] D. M. Caruso and S. E. Tropea, "Comparación del desempeño de microcontroladores AVR de 4ta generación," in *Congreso Argentino de Sistemas Embebidos - Libro de Trabajos*, Buenos Aires, Argentina, 2011, p. 179.
- [12] T. Harbaum. (2012, Feb.) Wlan for avr. [Online]. Available: <http://www.harbaum.org/till/spi2cf/index.shtml>
- [13] T. Gingold. (2010, Jun.) A complete VHDL simulator. [Online]. Available: <http://ghdl.free.fr/>
- [14] I. Murdock *et al.* (2010, Jun.) Debian GNU/Linux operating system. [Online]. Available: <http://www.debian.org/>
- [15] G. Combs and contributors. (2010, Jun.) Network protocol analyzer. [Online]. Available: <http://www.wireshark.org/>
- [16] (2011, Oct.) Embedded system example: Web server design using microblaze soft processor. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp433.pdf
- [17] (2011, Oct.) Web server reference design using a powerpc-based embedded system. Xilinx. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp434.pdf
- [18] L. W. Adam Dunkels *et al.* (2011, Oct.) The lwIP TCP/IP Stack. Swedish Institute of Computer Science. [Online]. Available: "<http://www.sics.se/~adam/lwip/>"



Inyección de Fallas utilizando memorias de dos puertos:

Un nuevo y versátil método para emular SEUs

Eduardo A. Sanchez¹, Pablo A. Ferreyra^{2,3}, Marcos J. Oviedo⁴, Carlos A. Marqués¹

1: Facultad de Matemáticas, Astronomía y Física – U.N.C.

2: Facultad de Ciencias Exactas, Físicas y Naturales –U.N.C.

3: Posgrado de Sistemas Embebidos – I.U.A.

4: Argentina Software Design Center - Intel

Córdoba, Argentina

{esanchez, marques, ferreyra}@famaf.unc.edu.ar; marcos.j.oviedo@intel.com

Resumen—En este trabajo se presenta un novedoso método para inyectar fallas tipo *Single Event Upsets* (SEUs) en un procesador implementado en FPGA de tipo *Soft Processor*. A diferencia de otros métodos existentes, la inyección se hace de manera absolutamente transparente al conjunto de aplicaciones que se estén ejecutando, en cualquier parte del ciclo de ejecución de una instrucción, en cualquier elemento de almacenamiento interno. Se basa en el uso de memorias de dos puertos compartidas entre el dispositivo bajo prueba (DUT) y un procesador maestro que controla el funcionamiento del DUT. Todos los bloques constitutivos se encuentran embebidos en la misma FPGA. El procesador maestro se comunica con el exterior mediante el protocolo *TCP/IP*, logrando un sistema totalmente versátil, y adaptable a cualquier tipo de procesador, sistema operativo y aplicación bajo estudio.

Palabras Clave: *fault injection, microprocessor, microcontroller, bitflip, single-event upset, field programmable gate arrays*

I. INTRODUCCIÓN

La sección transversal estática intrínseca de un procesador, denotada por “ σ ”, viene dada por la relación entre la cantidad de inversiones de *bits* que se producen (“*Single Event Upsets*” o “SEUs”) y la fluencia de las partículas que atraviesan el área sensible del mismo [1][2]. Se sabe que la sección transversal estática, conduce a una sobre estimación grosera de la cantidad de errores que se producirán en la aplicación, durante su ejecución [3][4]. Esta sobreestimación se puede corregir multiplicando la sección transversal estática por un factor conocido como la “tasa de errores”, denotado por “ τ ”. Éste último parámetro es la relación entre los errores que se producen en la aplicación y la cantidad SEUs que ocurren mientras que se ejecuta la misma. Existen muchos métodos para estimar la tasa de errores. Uno de los primeros trabajos en ese sentido se basa en el cálculo de los ciclos de trabajo de los elementos de almacenamiento de información [5]. Este método utiliza únicamente herramientas de “*software de debugging*”. Si bien, en principio se podría acceder a todos los elementos de almacenamiento del procesador, el problema de este método y de toda la familia de métodos que se basan en herramientas de *software*, son los tiempos de simulación extremadamente largos que los convierten en inaplicables al estudio de la

confiabilidad o robustez de sistemas de media o alta complejidad. Por tal motivo surgen métodos que combinan técnicas *hardware-software* para acelerar la inyección de fallas. Un método representativo de esta familia se conoce como “CEU” [6], por sus siglas en francés (*Code Emulateur D’Upsets*). El método CEU se basa en la ejecución de rutinas de interrupción asíncronas que ejecutan porciones de código que permiten invertir la información en registros o posiciones de memoria. Si bien CEU permite obtener una estimación rápida de la tasa de errores, está muy limitado en relación a los instantes de tiempo y a las posiciones que se pueden inyectar fallas. Además, agregan tiempos de ejecución y distorsionan las medidas tales como tiempos medios entre fallas. Estos problemas se extienden a toda la familia de métodos que se conocen hasta el momento.

La posibilidad de implementar varios procesadores en FPGA (*Soft Processors*) ha abierto nuevas estrategias inimaginables o inaplicables en los procesadores monolíticos, que resuelven los problemas anteriores. En particular, en este trabajo se presenta un novedoso método que se basa en el uso de memorias de dos puertos compartidas entre el DUT y un procesador maestro que controla el funcionamiento del DUT. Todos los bloques constitutivos se encuentran embebidos en la misma FPGA. El procesador maestro se comunica con el exterior mediante el protocolo *TCP/IP*, logrando un sistema totalmente versátil, y adaptable a cualquier tipo de procesador, sistema operativo y aplicación en el cliente. A diferencia de las familias de métodos mencionados más arriba, la inyección de fallas se hace de manera absolutamente transparente al conjunto de aplicaciones que se estén ejecutando, en cualquier parte del ciclo de ejecución de una instrucción, en cualquier elemento de almacenamiento interno. Esto representa sin lugar a dudas una contribución importante al estado del arte actual en el área.

En la sección II se da una descripción a nivel de sistemas y las tecnologías utilizadas por este nuevo método de inyección de fallas. En la sección III, se escoge un *hardware* DUT para demostrar el método. La sección IV, explica la arquitectura del *software* mostrando sus tres niveles, que permiten fácilmente adaptarse a cualquier tipo de dispositivo bajo prueba. La sección V, precisamente muestra los comandos requeridos para la simulación que son manejados a un nivel de abstracción tal

que permite la adaptación a cualquier procesador. En la sección VI se valida el concepto mediante el estudio de un caso de prueba sencillo. Finalmente en la sección VII, se elaboran conclusiones y proponen líneas de trabajo futuros asociadas al uso de este novedoso sistema de inyección de fallas.

II. DESCRIPCIÓN DETALLADA DEL MÉTODO DE INYECCIÓN DE FALLAS

Este trabajo pone foco en el modelo de fallas a nivel *bit* (conocido como *bit-flip*) [7][8], es decir en la modificación del contenido de un elemento en memoria durante la ejecución del DUT. En particular, se estudian los *bitflips* que ocurren en el módulo de memoria RAM del DUT únicamente, dejando sin efecto los que pueden ocurrir en los registros internos y en la memoria ROM del mismo. Existen numerosas herramientas que permiten, ya sea vía JTAG [9][10][11] o reconfiguración parcial [12][13], la modificación de los *bits* en memoria; aunque rara vez lo hacen sin parar la ejecución del DUT para realizar el cambio en la memoria. Se propone un método más eficiente donde un microprocesador (μ P) embebido dentro de la FPGA controle el uso de una memoria de doble puerto compartida. La inyección de fallas al DUT es realizada por *hardware* [14] mediante accesos del μ P a los registros de dicha memoria, tal como puede observarse en la Fig. 1. El uso de memorias SRAM multi-puerto permite que el mismo dato pueda ser leído y escrito vía dos puertos a distintas frecuencias de *clocks*. Si bien es factible que ocurran múltiples lecturas y escrituras simultáneamente; en el diseño propuesto se eliminan las colisiones al restringir, por *hardware*, el uso de este recurso compartido. En todo momento los accesos son secuenciados y manejados por el μ P maestro con prioridad para realizar cambios.

Teniendo en mente los recursos con los que cuenta una placa de desarrollo *Xilinx University Program Virtex-5* (XUPV5), se diseñó la arquitectura de *hardware* del método propuesto tal como se representa en la Fig. 2. Cabe destacar que la FPGA Virtex-5 posee recursos fijos para manejar la memoria multi-puerto, por lo que no es necesario diseñar un *hardware* específico propio en VHDL o Verilog para la implementación de dicha memoria.

La computadora *host* es utilizada para recibir datos de *debugging* vía consola serial y de manejar el sistema embebido mediante “comandos” enviados a través de paquetes TCP/IP vía *ethernet*, tal como se presenta en la sección V.

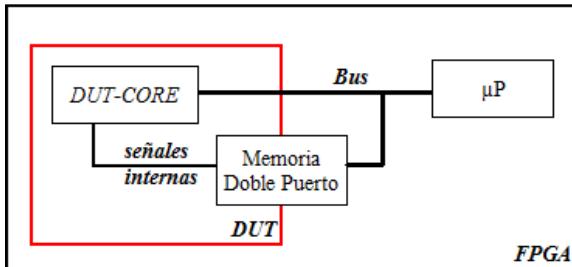


Figura 1. Conexiones DUT-Memoria Multipuerto- μ P

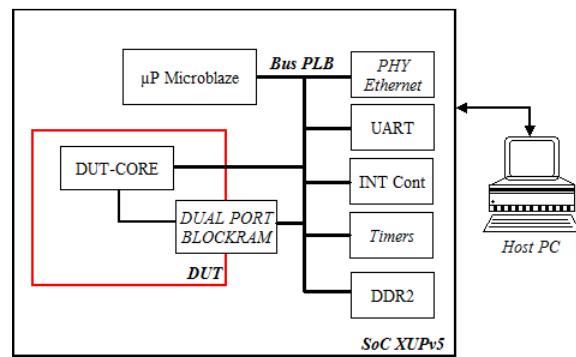


Figura 2. Diagrama en bloques de la arquitectura propuesta

III. HARDWARE ESPECÍFICO EN EL COMPONENTE DUT-CORE

Una prueba de concepto completa necesita un *DUT-CORE* concreto con el que trabajar. En la actualidad, el código funcional del microcontrolador (μ C) 8051 es provisto por *Oregano Systems* de forma libre y gratuita bajo licencia LGPL en lenguaje VHDL [15]. Se selecciona este tipo de μ C, debido a su gran aceptación y utilización en satélites para controlar sistemas más complejos y poder recuperarlos de fallas, inclusive existen esfuerzos en robustecerlo [16][17][18]. Si bien hay que realizar ciertas adaptaciones al diseño de *Oregano*, es de gran ayuda tener ejemplos y código testeado funcional con el que trabajar. Como el diseño original coloca el programa en memoria ROM descripta en lenguaje VHDL, se la reemplaza por una *multi-port blockram* para lograr mayor flexibilidad y eficiencia de los recursos, tal como puede observarse en la Fig. 3.

Esta pequeña modificación permite que el programa a correr en el μ C 8051 pueda ser actualizado sin la necesidad de una recopilación total o parcial de *hardware*, lo que se traduce en una ganancia de velocidad muy importante.

IV. ARQUITECTURA DE SOFTWARE DEL MÉTODO DE INYECCIÓN DE FALLAS

Debido a la arquitectura de *hardware* implicada en el diseño de la Fig. 2, pueden determinarse tres “niveles” o categorías de *software* heterogéneos tal como indica la Fig. 4.

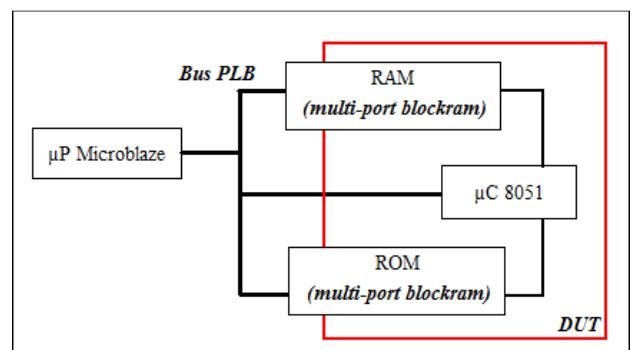


Figura 3. Arquitectura de *hardware* DUT específica

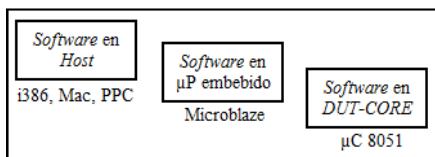


Figura 4. Categorías de software heterogéneo

A. El software en Host y en μ P embebido

Ambos van de la mano, ya que deben compartir el protocolo de comunicación entre ellos. La arquitectura del sistema se diseña siguiendo los lineamientos del, ya conocido, modelo cliente/servidor. Donde el cliente es el que envía un “comando” a ejecutarse en el servidor, éste lo procesa y retorna un resultado. La comunicación del sistema embebido al mundo exterior se realiza mediante paquetes *TCP/IP*; permitiendo enviar información de *debugging* vía la consola serial (Fig. 5).

El servidor “embebido” en la placa de desarrollo, es diseñado con la capacidad de inicializar, monitorear, programar, ejecutar programas y elegir el punto exacto donde cambiar el valor de un *bit* en la memoria del *DUT-CORE*. Con escasos recursos en la placa siempre es mejor mantener un diseño simple y “liviano” del mismo. El servidor no debe guardar el estado de la comunicación ni de la ejecución del *DUT* y solo debe esperar “comandos” que le envía el cliente, y reaccionar a los mismos; sin tener en cuenta que comando ejecutó con anterioridad. De esta manera, el estado y la complejidad se manejan íntegramente del lado del cliente, lo cual no es un problema teniendo en cuenta los recursos con los que cuentan las computadoras actuales. Se propone una opción “liviana” al colocar un *stack TCP/IP*, a correr sobre el único hilo de ejecución, llamado *Lightweight TCP/IP stack (lwIP)* [19][20]. Su propiedad más importante, es que puede correr directamente sobre el μ P Microblaze sin la necesidad de colocar un sistema operativo. *lwIP* es una implementación pequeña del protocolo *TCP/IP*, desarrollada en una primera instancia por Adam Dunkels del Instituto Sueco de Ciencias de la Computación y en la actualidad abierta a la comunidad. El foco es reducir al mínimo el uso de los recursos pero siempre manteniendo *TCP/IP*, lo que hace posible su fácil integración en sistemas embebidos. Cabe destacar que *lwIP* no es el único en su especie, y que existen otras opciones de servidores “livianos” a explorar como μ C/TCP-IP o CW-TCP/IP; siendo este último más orientado a redes *wireless* [21].

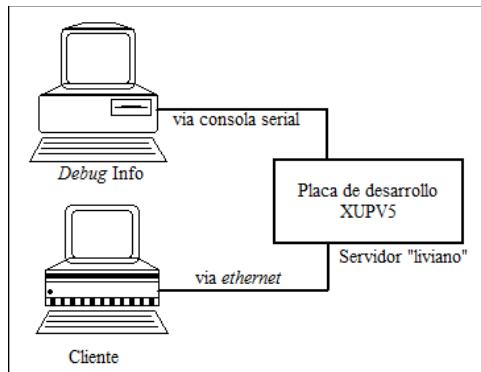


Figura 5. Comunicación del SoC con el exterior

B. El software que corre en DUT-CORE

Existen diversas herramientas que permiten el desarrollo, compilación y *debugging* de código para μ C 8051. El código se compila a formato *Intel HEX*, por lo que debe ser traducido a un formato adecuado para trabajar. Con el fin de lograr una mayor flexibilidad del sistema, el programa a correr en el *DUT-CORE* se compila en la PC, se lo traduce a formato propio, se lo empaqueta y envía mediante un “comando” especial al servidor, para que el μ P Microblaze lo coloque en la memoria compartida.

V. COMANDOS PARA LA INYECCIÓN DE FALLAS

Tanto el cliente como el servidor deben compartir un protocolo específico de “comandos” que permitan la ejecución real de una ronda de inyección de fallas. El servidor debe ser capaz de responder estos comandos sencillos muy rápidamente. Veamos algunos ejemplo de estos comandos: *correr_dut* (*N* cantidad de ciclos), *parar_dut*, *correr_dut_un_paso*, *habilitar_memorias*, *retornar_resultados*, *cargar_memoria_rom*, *resetear_dut*, etc. es deber del cliente organizarlos y enviarlos de manera que se logre una ejecución completa. Si se precisa un cambio de bit en una posición de memoria, existe un comando especial llamado *configurar_bitflip* donde se indica donde y cuando realizar el *bitflip* deseado. Se define, entonces, una “corrida de ejecución” al conjunto de los comandos anteriores que debe enviar el cliente desde el primer comando necesario hasta la obtención de los resultados del *bitflip* (ver Fig. 6). Se define una “ronda de inyección de fallas” a un conjunto de las corridas de ejecución.

Cabe aclarar que tanto el *overhead* de reconexión como el de la escritura de un mismo programa pueden eliminarse corriendo este tipo de “rondas”, como indica la línea punteada de la Fig. 6.

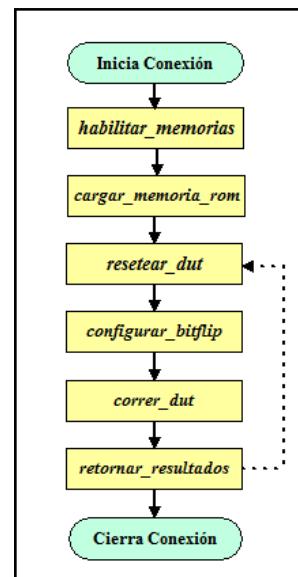


Figura 6. Diagrama de comandos necesarios para un *bitflip*



VI. CASO DE ESTUDIO Y RESULTADOS OBTENIDOS

La aplicación escogida a ser testeada es muy simple, ya que sólo se desea probar el concepto de la máquina de inyección de fallas y no la robustez de una aplicación en particular. El programa realiza la suma de un vector de cinco elementos, originalmente escrito en *assembly* para tener completo control de las posiciones de memoria afectadas. Los elementos se encuentran en posiciones contiguas desde la dirección *0x40h*; colocando el resultado en la posición *0x50h* y *0x51h*. El µC 8051 necesita 98 ciclos para ejecutar la aplicación y obtener el resultado. Si bien existen múltiples versiones de este programa, se escogió una aplicación que utilice la menor cantidad de registros internos. Luego la inyección de fallas puede ocurrir en cualquiera de esos 98 ciclos, en cualquier *bit* de cualquier dirección de memoria RAM.

En la Tab. 1 se encuentran los resultados de 10 rondas de 100 inyecciones de falla cada una. Cada ronda además incluye los resultados de la ejecución sin fallas (*Golden Run*) con los que se van comparar el resto. En los casos donde la falla inyectada produce un error en el sistema (i.e. cambios en el resultado final de la aplicación) se considera como *failure*, y si no tiene un efecto sobre los resultados como *silent* [9].

VII. CONCLUSIONES Y MEJORAS FUTURAS

Los resultados obtenidos en la sección anterior, permiten validar el concepto de este novedoso método de inyección de fallas. En efecto, al tratarse de una aplicación tan sencilla, puede verificarse fácilmente que su tasa de errores “ τ ” es del orden del tres por ciento.

La factibilidad tecnológica y la conveniencia del uso de las memorias de dos puertos para inyección de fallas ha sido ampliamente demostrada en este trabajo. En efecto, dado que Los pulsos de reloj del procesador esclavo (DUT) son generados por medio del sistema de interrupciones y temporizadores del µP maestro, los accesos a memorias compartidas pueden hacerse dentro de cualquier parte del ciclo de reloj del procesador bajo estudio y en cualquier posición de memoria de manera totalmente transparente para el DUT. Además de las memorias, se puede utilizar para simular fallas en cualquier registro de almacenamiento de datos del sistema bajo estudio. A nuestro entender, estas posibilidades representan una importante contribución al estado del arte actual en el área.

Este trabajo continuará con la aplicación de este método a aplicaciones reales tales como sistemas de distribución de energía dentro de un satélite controlado por un µC 8051. Por otro lado se prevé extender su uso a aplicaciones más complejas; tales como algoritmos de control de actitud y órbita, encargados de la estabilización y orientación del satélite, a ejecutarse sobre un procesador León III.

AGRADECIMIENTOS

A la Secretaría de Ciencia y Tecnología de la Universidad Nacional de Córdoba por otorgar becas y subsidios para permitir el desarrollo de este trabajo.

TABLA I. CLASIFICACIÓN DE FALLAS INYECTADAS

#Ronda	#Failure	#Silent
1	2	98
2	3	97
3	2	98
4	4	96
5	2	98
6	3	97
7	3	97
8	3	97
9	1	99
10	4	96
Total	27 (2,7%)	973 (97,3%)

REFERENCIAS

- [1] R. Koga, W.A. Kolasinski, M.T. Marra, W.A. Hanna, “*Techniques of Microprocessor Testing and SEU Rate Prediction*”, IEEE Transaction on Nuclear Science, vol. NS-32, N°6, December 1985.
- [2] R. Harboe-Sorensen, L. Adams, E.J. Daly, C. Sansoe, D. Mapper, T.K. Sanderson, “*SEU Risk assessment of Z80A, 8086 and 80C86 microprocessors intended for the use in a low altitude polar orbit*”, IEEE Transaction on Nuclear Science, vol. 33, N°6, December 1986.
- [3] R. Velazco, S. Karoui, T. Chapuis, D. Benezech, L.H. Rosier, “*Heavy ion tests for the 69020 microprocessor and the 68882 Coprocessor*,” IEEE Trans. on Nucl. Sci., vol. 39, N° 3, Dec. 1992.
- [4] F. Bezerra, R. Velazco, A. Assoum, and D. Benezech, “*SEU and latchup results on transputers*”, in Trans. on Nucl. Sci.: IETNAE, 1996, pt. I, vol. 43.
- [5] J. H. Elder, J. Osborn, W.A. Kolasinsky, R. Koga, “*A Method for Characterizing Microprocessor’s Vulnerability to SEU*”, IEEE Transaction on Nuclear Science, vol. 35, N°6, December 1988.
- [6] R. Velazco, S. Rezgui, R. Ecoffet, “*Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection*”, IEEE Transactions on Nuclear Science, Vol. 47, Issue 6, Part 3, Dec. 2000.
- [7] V. Asenek, C. Underwood, R. Velazco, S. Rezgui, M. Oldfield, Ph. Cheynet, R. Ecoffet, “*SEU induced errors observed in microprocessor systems*”, IEEE Transactions on Nuclear Science, Vol. 5, Issue 6, Part 1, Dec. 1998.
- [8] S. Rezgui, R. Velazco, R. Ecoffet, S. Rodriguez, J.R. Mingo, “*Estimating error rates in processor-based architectures*”, IEEE Transactions on Nuclear Science, Volume 48, Issue 5, Oct. 2001.
- [9] M. Sonza Reorda, L. Sterpone, M. Violante, M. Portela-Garcia, C. Lopez-Ongil, L. Entrena, “*Fault Injection-based Reliability Evaluation of SoPCs*”, ETS ’06 Proceedings of the Eleventh IEEE European Test Symposium, pp. 75 - 82, 2006.
- [10] M. Portela-García, C. López-Ongil, M.G. Valderas, L. Entrena, “*Fault Injection in Modern Microprocessors Using On-Chip Debugging Infrastructures*”, IEEE Transactions on Dependable and Secure Computing, vol. 8, pp. 308-314, 2011.
- [11] M. Pignol, “*Methodology and Tools Developed for Validation of COTS-based Fault-Tolerant Spacecraft Supercomputers*”, 13th IEEE International On-Line Testing Symposium, pp.85-92, 2007.
- [12] U. Legat , A. Biasizzo and F. Novak, “*Automated SEU fault emulation using partial FPGA reconfiguration*”, In Proceedings of the IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 24-27, 2010.
- [13] L. Sterpone, M. Violante, “*A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs*”, IEEE Transactions on Nuclear Science, Part 2, 2007, p. 965-970.
- [14] G.C. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli, R. Velazco, “*Bit flip injection in processor-based architectures: a case*



- study”, Proceedings of the Eighth IEEE International On-Line Testing Workshop, pp.117-127, 2002.
- [15] Oregano Systems, “8051 IP”, Internet: <http://www.oreganosystems.at>, Aug. 2004 [Jun, 2011].
- [16] F. G. de Lima, E. Cota, L. Carro, M. Lubaszewski, R. Reis, R. Velazco, S. Rezgui, “Designing a Radiation Hardened 8051-Like Micro-Controller”, Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, pp. 255–260, September 2000.
- [17] J.W. Howard, K.A. LaBel, M.A. Carte, C. Seidleck, J.W. Gambles, S.L. Ruggles, “Validation and testing of design hardening for single event effects using the 8051 microcontroller”, Proceedings of the 2005 Radiation Effects Data Workshop IEEE, pp. 85-92, 2005.
- [18] E. Cota, F. Lima, S. Rezgui, L. Carro, R. Velazco, M. Lubaszewski, R. Reis, “Synthesis of an 8051-Like Micro-Controller Tolerant to Transient Faults”. 1st IEEE Latin America Test Workshop (LATW), 2000.
- [19] A. Dunkels, “IwIP - A Lightweight TCP/IP stack”, Internet: <http://savannah.nongnu.org/projects/iwip> Oct. 2002 [Dec. 2011].
- [20] Q. Chuanfei, L. Junxian, Z. Wei, X. Guanghui, W. Dawei, “Design of Online Reconstructable SOPC System Based on TCP/IP”, 2010 International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), vol. 1, pp. 1022 - 1025, 2010.
- [21] In-Su Yoon, Sang-Hwa Chung, Jeong-Soo Kim, “Implementation of Lightweight TCP/IP for Small, Wireless Embedded Systems”, Proceedings of the 2009 International Conference on Advanced Information Networking and Applications, 2009.

Implementación de funciones básicas del amplificador Lock-in en FPGA

Layla Martínez Guevara, Esteban Peláez, C. Sosa Paez

Facultad de Ciencias Físico Matemáticas y Naturales

Universidad Nacional de San Luis

Email: {lmartinez,epelaez,cfsp}@unsl.edu.ar

Abstract—Los amplificadores Lock-in permiten estudiar señales débiles, con fuerte presencia de ruido, mediante la modulación y posterior detección por fase de dicha señal. Estos dispositivos utilizan la técnica conocida como Phase Sensitive Detection (PSD) para aislar la componente de la señal de una determinada frecuencia y fase de referencia. Se propone la implementación de las funciones básicas de un amplificador digital Lock-in, en una FPGA de gama media y con su propio generador de señal, el cual tiene la ventaja de ser económico y de pequeñas dimensiones. El trabajo propuesto, está orientado a detectar de manera automática, la frecuencia de resonancia de un sensor implementado en un sistema MEMs.

I. INTRODUCCIÓN

Cuando se trabaja con muestras de tamaño micro y nanométricos de nanotubos de manganita individual, uno de los principales objetivos es medir la respuesta magnética [1]. Para lograr esto, no sólo se necesita fabricar este tipo de muestras sino que también es fundamental implementar instrumentos suficientemente sensibles que permitan medir las pequeñas señales que se generan en dicho experimento. La implementación de micro-osciladores mecánicos de silicio como dispositivos de medición es indispensable para cumplir con el objetivo. Estos forman parte de lo que se conoce como Micro Electro Mechanical Systems (MEMs). Los mismos tienen una alta sensibilidad y un tamaño reducido.

Un micro-oscilador presenta una paleta central unida a dos resortes tipo serpentina los cuales se encuentran anclados al sustrato mediante pilares en sus extremos (Fig. 1). Los resortes proveen una fuerza de restitución y la paleta provee la mayor parte del momento de masa inercial del oscilador mecánico. Por debajo de ella se encuentran dos electrodos que son utilizados para detección y excitación del movimiento. Un modo torsional de oscilación puede producirse mediante una tensión alterna aplicada en alguno de los dos electrodos, con valores de amplitud entre 80 mV y 150 mV. El micro-oscilador en esta configuración oscila a dos veces la frecuencia de excitación. La frecuencia de resonancia de estos varía entre 20 kHz y 70 kHz dependiendo de varios factores, entre ellos de las dimensiones de la paleta y la longitud de la serpentina. El cuadrado de la frecuencia natural, de un oscilador torsional sin disipación es proporcional a la constante elástica de las serpentinas, e inversamente proporcional al momento de inercia del sistema.

Un método de detección del modo torsional de los micro-osciladores es el capacitivo. Por debajo de la paleta (Fig. 1) se



Fig. 1. Foto de un micro-oscilador

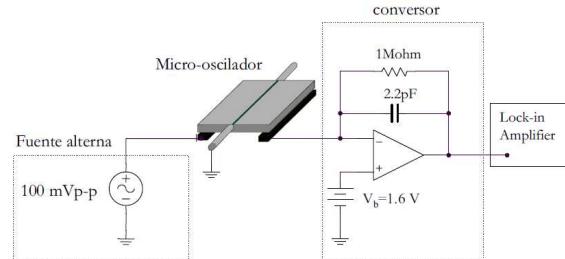


Fig. 2. Esquema del sistema

encuentran dos electrodos fijos, uno de excitación al cual se le aplica una tensión alterna y otro de detección. A medida que se produce el movimiento de la paleta, se sienta la capacidad eléctrica entre el electrodo fijo de detección y el oscilador utilizando el sistema compuesto por un convertidor de corriente tensión y un amplificador Lock-in (Fig. 2).

Los amplificadores Lock-in se utilizan para detectar y medir señales muy pequeñas de tensión alterna. Con ellos es posible detectar fase y amplitud a una frecuencia de referencia específica. Las señales de ruido con frecuencias que difieren a la de referencia son rechazadas.

La respuesta en frecuencia de un MEM, como el de la Fig. 3, posee un ancho de banda estrecho que puede variar entre 10 Hz y 50 Hz a la frecuencia de resonancia, dependiendo del material y las condiciones del experimento.

II. DESCRIPCIÓN FUNCIONAL

En el proceso de transmisión, las señales siempre se ven mezcladas con señales no deseadas. En general cualquier proceso impuesto sobre alguna señal tiende a introducir perturbaciones indeseables, denominadas ruido. La medición de

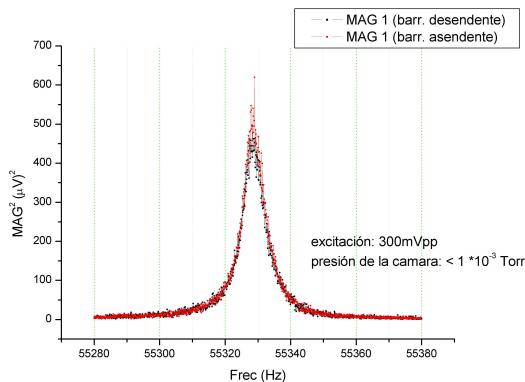


Fig. 3. Gráfica frecuencia de resonancia

señales eléctricas de bajo nivel es una tarea difícil. Ya sea ruido AC o variaciones de DC causan que la lectura sea inestable y aumente la incertidumbre en la precisión de la misma. Existen diversas técnicas que atenúan este problema, una de las más efectivas es la “Detección Sensible a Fase” [2][3].

A. Detección Sensible a Fase

La detección sensible a fase (PSD) consiste en la multiplicación de la señal de entrada por una señal de referencia de una frecuencia constante conocida. Algunos amplificadores Lock-in utilizan una señal de referencia cuadrada, mientras que otros usan una senoidal. Una referencia cuadrada contiene armónicos impares de la fundamental lo cual causa que el ruido a estas frecuencias armónicas sea detectado y esto es normalmente indeseable. En cambio, una referencia senoidal perfecta arroja como resultado la detección de sólo la componente fundamental y es por esta razón que se decidió utilizar este tipo de señal para el diseño.

$$V_{PSD} = V_{sig} \sin(\omega_{sig} t + \theta_{sig}) \times V_{ref} \sin(\omega_{ref} t + \theta_{ref}) \quad (1)$$

Cuando las frecuencias ω_{sig} , ω_{ref} y ω son iguales y la salida del PSD pasa por un filtro pasa bajo se obtiene en la salida una señal DC proporcional a la magnitud de la señal de entrada.

$$V_{PSD} = 0,5 \times V_{sig} \times V_{ref} \times \cos(\theta_{sig} - \theta_{ref}) \quad (2)$$

B. Magnitud y Fase

Un Lock-in de fase simple utiliza una sola PSD, la desventaja es que presenta una dependencia de fase entre la señal de entrada y de referencia. Esta dependencia puede ser eliminada por la adición de un segundo PSD, que multiplica la señal de entrada con el oscilador de referencia desplazado por 90° . A la salida del filtro se obtienen las siguientes señales.

$$X = V_{PSD1} = 0,5 \times V_{sig} \times V_{ref} \times \sin(\theta_{sig} - \theta_{ref}) \quad (3)$$

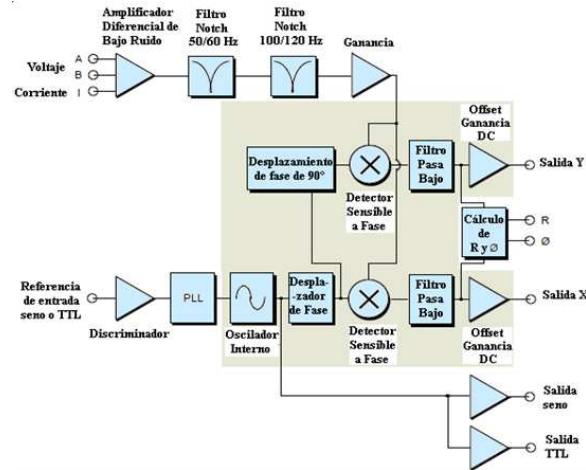


Fig. 4. Diagrama de bloques del Lock-in de fase doble

$$Y = V_{PSD2} = 0,5 \times V_{sig} \times V_{ref} \times \cos(\theta_{sig} - \theta_{ref}) \quad (4)$$

Estas dos cantidades representan la señal como un vector en relación con el oscilador de referencia del Lock-in. X se llama la componente **en fase** e Y la componente en **cuadratura**. Mediante el cálculo de la magnitud del vector señal (R), la dependencia de fase se elimina.

$$R = \sqrt{X^2 + Y^2} \quad (5)$$

III. DISEÑO DEL LOCK-IN

El amplificador Lock-in del tipo comercial, es utilizado en laboratorios especializados cuya desventaja es el costo elevado que hace en ocasiones difícil su adquisición. Puede poseer referencia interna o externa, y no sólo mide la amplitud del vector sino también el ángulo de desfasaje; el diagrama básico se puede ver en la Fig. 4.

Por dicho motivo, se propone la implementación de las funciones básicas de un amplificador digital Lock-in, en una FPGA de gama media y con su propio generador de funciones. La decisión de utilizar una FPGA está basada en su arquitectura especializada que permite reprogramarla una gran cantidad de veces [4], lo que posibilita no sólo realizar mejoras en el diseño sino también implementar otro instrumento totalmente diferente en caso de ser necesario. Además permite ejecutar operaciones complejas a alta velocidad, las cuales son necesarias para llevar a cabo la técnica PSD que emplea el Lock-in.

En la Fig. 5 se muestra el diagrama en bloques del amplificador Lock-in diseñado y su ubicación en el sistema general. La idea central del instrumento es inyectar al experimento una señal de frecuencia fija y conocida mediante el sistema de control. Esta señal, generada por un DDS (Sintetizador Digital Discreto), es una onda senoidal que es utilizada para comandar un detector de fase. Dado que la salida del PSD es proporcional a la diferencia de fase entre la señal de entrada y

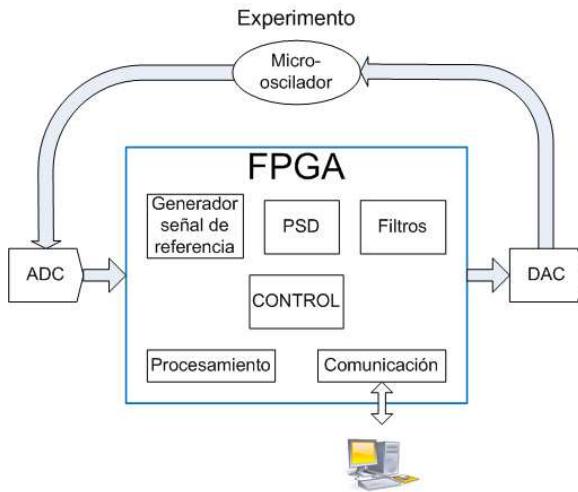


Fig. 5. Diagrama simplificado de la implementación

la de referencia, se utiliza un segundo PSD para poder medir la amplitud de la señal independizándose de dicha diferencia. Posteriormente se utiliza un filtro pasa bajo de banda angosta. Sólo la señal a la frecuencia de referencia resultará en una salida DC proporcional a la magnitud de entrada y no será afectada por el filtro pasa bajo. Para visualizar los resultados obtenidos se utiliza una interfaz a una computadora. La señal de entrada, a ser analizada, debe ser acondicionada por una etapa analógica diseñada para tal fin. El rango de tensión, de dicha señal, debe adaptarse al del conversor A/D de la placa. Además ésta señal debe ser filtrada con un filtro pasa-bajos (filtro antialiasing), con una frecuencia de corte inferior a la mitad de la frecuencia de muestreo configurada en el conversor A/D. Ésta etapa de acondicionamiento, no forma parte de este trabajo.

Teniendo en cuenta las características del experimento a realizar, se han establecidos los siguientes requisitos que debe cumplir el diseño:

- Rango de frecuencia: 20 kHz a 70kHz.
- Resolución de la frecuencia: 0,1 Hz
- Sentido del Barrido: Ascendente.
- Cantidad de Barridos: 5

Los instrumentos comerciales cuentan con la posibilidad de establecer el tiempo en que se permanece en la frecuencia de trabajo, que es la misma para cada una del barrido, pero en este diseño cada frecuencia contiene la misma cantidad de períodos, por lo tanto el tiempo de permanencia en cada frecuencia es distinta.

Se emplean cinco barridos para determinar la frecuencia de resonancia. Dado que la respuesta en frecuencia de un material es de un ancho de banda aproximado de 20 Hz, dependiendo del material, se consideró conveniente realizar un primer barrido en el que la frecuencia cambiara de 20 a 70 kHz con una variación de 2 Hz; los restantes se basan en la frecuencia de resonancia aproximada obtenida en el barrido anterior para determinar la nueva frecuencia mínima

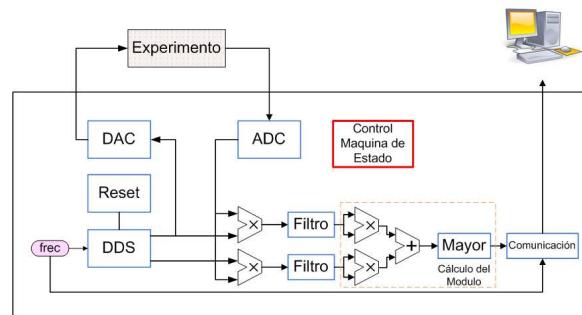


Fig. 6. Diagrama esquemático del diseño

y máxima. Para ello, las fórmulas que se utilizan son las siguientes:

$$f_{min} = (10 \times \Delta f) - f_{resonancia}$$

$$f_{max} = (10 \times \Delta f) + f_{resonancia}$$

Los Δf correspondientes a cada barrido son:

- 1º barrido: 2 Hz
- 2º barrido: 1 Hz
- 3º barrido: 0,5 Hz
- 4º barrido: 0,2 Hz
- 5º barrido: 0,1 Hz

Sólo los valores de frecuencia y amplitud del último barrido son transmitidos a la computadora, que se encarga de graficar la respuesta en frecuencia.

El diseño cuenta con una máquina de estado que realiza el control global. El funcionamiento cumple los siguientes pasos:

- 1) Inicializar los filtros y el DDS.
- 2) Calcular la frecuencia de trabajo y configurar al DDS.
- 3) Inicializar el contador de períodos de la frecuencia de trabajo.
- 4) Habilitar el PSD y los filtros.
- 5) Calcular la magnitud del vector y máximo valor de amplitud.
- 6) Calcular el barrido que se realizará.
- 7) Transmitir los datos.

En la Fig. 6 se muestran todos los bloques a implementar en la FPGA.

IV. SINTETIZADOR DISCRETO DIGITAL

El DDS es el bloque encargado de generar una onda senoidal y cosenoide a la frecuencia deseada. Se utilizó el modulo IP CoreDDS de la empresa Actel [5], que es un generador de RTL que produce un sintetizador digital directo optimizado para FPGAs. Este genera digitalmente una onda sinusoidal de valores complejos o reales. Debido a la naturaleza digital de la funcionalidad del mismo, ofrece un cambio rápido entre las frecuencias de salida, buena resolución de frecuencia, y operaciones sobre un amplio rango de frecuencia.

EL DDS se puede implementar en tres arquitecturas de hardware: CORDIC Paralelo, Small LUT y Big LUT, siendo la Small LUT la elegida para este diseño, dado que es la que ocupa menos área de la FPGA y cumple con las

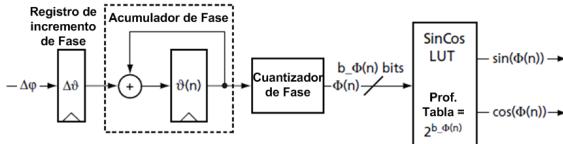


Fig. 7. Diagrama de bloques de DDS basado en LUT

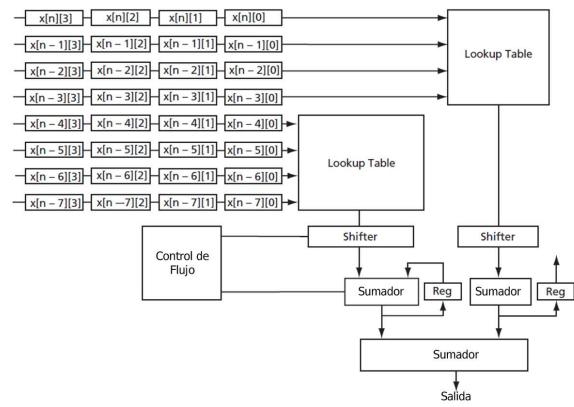


Fig. 8. Filtro FIR de arquitectura de Aritmética Distribuida

especificaciones requeridas. Las LUTs almacenan muestras de una onda seno y coseno uniformemente espaciadas, que representan un único ciclo de una senoidal compleja. El ciclo contiene $N = 2^{b_{\vartheta}(n)}$ muestras complejas y corresponden a valores específicos del argumento de la sinusoida.

A. Frecuencia de Salida

La frecuencia de salida, f_{out} , de la forma de onda del DDS, es una función de la frecuencia de reloj del sistema, f_{clk} , el número de bits en el acumulador de fase, $b_{\vartheta}(n)$, y el valor de incremento de fase, $\Delta\varphi$. La frecuencia de salida en Hertz se define como se muestra en (6).

$$f_{out} = \frac{f_{clk} \Delta\varphi}{b_{\vartheta}(n)} \quad (6)$$

B. Resolución de frecuencia

La resolución de la frecuencia del sintetizador es una función de la frecuencia de reloj y el número de bits empleados en el acumulador de fase y se puede determinar usando la ecuación 7.

$$2^{b_{\vartheta}(n)} = \log_2\left(\frac{f_{clk}}{\Delta f}\right) = \log_2\left(\frac{20 \times 10^6 \text{ Hz}}{0,1 \text{ Hz}}\right) = 27,575 \quad (7)$$

Teniendo en cuenta que la frecuencia de trabajo en la FPGA es de 20 MHz, y la resolución mínima deseada es de 0,1 Hz, el ancho de palabra mínimo del acumulador debería ser de 27,575 bits, por lo que se tomó la cantidad de 28 bits, reemplazando este valor se obtiene una resolución de 0,0745058 Hz.

Con esta resolución se trabajará en el diseño del Lock-in, pero existe la posibilidad de incrementarla como máximo a 0,0045661 Hz, aumentando el número de bits utilizados en el acumulador de fase a 32 bits. En base a estos parámetros, el DDS se configuró para generar una onda sinusoidal compleja de frecuencia variable y fase constante.

V. FILTROS DIGITALES

El PSD y el filtro paso bajo sólo detectan señales cuyas frecuencias están muy cerca a la de referencia. El ruido en las frecuencias lejanas a la de referencia se atenúa a la salida del filtro paso bajo, en cambio en las frecuencias cercanas a la frecuencia de referencia, da lugar a salidas de AC de frecuencia muy baja de la PSD ($|\omega_{noise} - \omega_{ref}|$ es pequeño). Su atenuación depende del ancho de banda del filtro de paso bajo y de la banda de transición. Un ancho de banda más estrecho eliminará fuentes de ruido muy cercanas a la

frecuencia de referencia, un mayor ancho de banda permitirá que estas señales pasen. Sólo la señal a la frecuencia de referencia se traducirá en una verdadera salida de CC y no será afectada por el filtro paso bajo. Esta es la señal que se quiere medir.

Debido a que este Lock-in se implementará en FPGA de gama media, el filtro a utilizar se debe escoger teniendo en cuenta el ancho de banda del mismo y la cantidad de recursos utilizados.

Se realizaron varias pruebas hasta llegar al óptimo, ya que junto con el DDS, es el componente de mayor importancia en el diseño y el que debe cumplir más requerimientos. Los primeros filtros fueron generados con el bloque FIR y la herramienta FDA TOOL que ofrece Synphony DSP, pero un filtro de este tipo de orden 64 ocupa el 70% del área de la FPGA y dado que se requieren dos para el diseño, fue necesario buscar otra opción.

Como alternativa para generar el componente se utilizó el COREFIR 4.0 que es un generador de RTL de Actel optimizado para FPGA [6]. Permite generar dos tipos de arquitectura: coeficientes constantes (CF) o de aritmética distribuida (AD) (Fig. 8). La arquitectura de coeficientes constante es una arquitectura basada en multiplicadores para obtener un throughput más alto, mientras que la arquitectura AD utiliza sumadores y operaciones de desplazamientos para realizar los cálculos en una pequeña área. Es por este motivo que se decidió utilizar la arquitectura AD [7]. Esta implementación ocupa sólo el 34% de los recursos de la FPGA.

A. Características de los Filtros

La característica de diseño de los filtros se definió teniendo en cuenta el ancho de banda, la cantidad de coeficientes y los recursos utilizados.

- Tipo: FIR
- Orden: 64
- Frec. corte inferior de la banda de transición: 450 Hz
- Frec. corte superior de la banda de transición: 6000 Hz
- Frecuencia de muestreo: 300 kHz
- Número de bits de entrada: 20

TABLA I
CARACTERÍSTICAS DE LA LP DATA CONVERSION DAUGHTER BOARD

	ADC	DAC
Canales	2	2
MSPS	20	0,9
Bits	10	14
Interfaz	Serie	Paralelo

- Número de bits de coeficientes: 13
- Tipo de entrada y salida: sin signo

La cantidad de bits de entrada se debe a que la señal a filtrar es el producto entre la señal del DDS y la proveniente del experimento las cuales son de 10 bits cada una. Como la señal de mayor frecuencia generada por el DDS es de 70 kHz, a la salida de los multiplicadores la frecuencia será de 140 kHz, por lo cual la frecuencia de muestreo se eligió mayor que el doble de esta frecuencia de manera tal de cumplir con el teorema de Nyquist.

B. Determinación de los coeficientes del Filtro

Los coeficientes utilizados se obtuvieron mediante la herramienta Toolbox de Procesamiento de Señales de MATLAB en conjunto con los bloques de la herramienta Symphony que incluye bloques FIR e IIR, que proveen una interfaz al software FDATool [8].

El FDATool genera automáticamente los coeficientes del filtro. Symphony proporciona una función para incorporar automáticamente los coeficientes generados por el FDATool al filtro seleccionado, pero como en este diseño no se usa un filtro de Symphony los coeficientes debieron exportarse al archivo de configuración del COREFIR previa conversión al formato Q.

VI. IMPLEMENTACIÓN

Este diseño puede ser implementado en la placa RVI Prototype Board [9] o Fusion Embedded Development Kit [10], utilizando en ambos casos la placa de expansión LP Data Conversion Daughter Board. Las mismas poseen los componentes básicos necesarios para el diseño del amplificador, pero no poseen filtro antialiasing. Las características de estas placas en cuanto celdas disponibles y Block RAMs son iguales.

A. LP Data Conversion Daughter Board

Esta placa posee un conversor AD y DA, con las características que se enumeran en la tabla I. La máxima frecuencia de clock del bloque DAC es de 22,5 MHz. Dado que se requieren 25 ciclos para poner un dato completo en la salida, la tasa máxima de conversión es de 0,9 MHz [11].

Tomando un margen de seguridad, la frecuencia de clock en la entrada de este bloque, se fijó en 20 MHz. Por lo tanto, el clock out de este módulo es de 800 kHz (20 MHz/25).

VII. RESULTADOS OBTENIDOS

Cada bloque se simuló primero por separado, luego se integraron y se diseñó el bloque de control. Una vez finalizado el diseño, para tener la seguridad de que el módulo Lock-in funcionaba en forma correcta, se realizaron las tres simulaciones posibles: antes de ser sintetizado, luego de ser sintetizado y luego de realizar el proceso de posicionamiento-ruteo.

Para la simulación del diseño completo, por razones de una mejor visualización y para disminuir el tiempo de duración, se redujo el barrido de frecuencias a un primer barrido de entre 20 kHz y 25 kHz con un delta de frecuencia de 1 kHz, y un segundo con delta de 100 Hz. Las simulaciones fueron realizadas con una señal de estímulo sinusoidal de 23 kHz a la entrada del AD. Para simular dicha señal se utilizó un archivo de texto con valores obtenidos de una onda senoidal de amplitud 1 con componente de continua, diseñada en MATLAB. La frecuencia de la misma puede variarse desde el testbench modificando la velocidad con que son leídos los datos.

Para comprobar que se cumplían con los requisitos establecidos en el diseño, se procedió a realizar una serie de mediciones para los siguientes bloques: DDS, filtro, máquina de control, sistema de comunicación e interfaz gráfica.

Del bloque DDS se verificó la exactitud de la frecuencia generada con respecto a la deseada, y se determinó que hay un desviación de 0,017 Hz. Esta verificación se realizó de la siguiente manera: se generaron señales de distintas frecuencias y se midió el valor obtenido con un multímetro digital (HP34401 A). También se midió si la variación en frecuencia mínima es igual a 0,1 Hz de acuerdo a la configuración elegida, determinando de este modo que la resolución de frecuencia obtenida es 0,0782 Hz.

El correcto funcionamiento del filtro se verificó realizando un barrido de frecuencia con un generador de señales en el rango de 0 Hz a 70 kHz y analizando la respuesta en frecuencia con el amplificador Lock-in Signal Recovery 7280.

Para verificar el funcionamiento integral del diseño, se procedió a la medición de la frecuencia de resonancia de dos circuitos RLC paralelo con valores de Q alto. Se realizó un único barrido de 20 a 70 kHz, con un incremento de frecuencia de 500 Hz. Los resultados se contrastaron con los obtenidos utilizando el Lock-in 7280. En la tabla 2 se muestran los valores de la frecuencia de resonancia, calculados y medidos con los dos Lock-in, la diferencia observada entre dichos resultados es que el Lock-in digital no cuenta con la etapa de acondicionamiento adecuada para las señales de entrada y salida.

El reporte de síntesis de la lógica de la descripción del hardware, implementada en la FPGA se indica en la tabla 3.

A. Limitaciones

La limitación más importante en esta implementación son los conversores utilizados. El conversor A/D de la placa LP Data Conversion Daughter Board, posee una resolución de 10 bits de ancho de palabra. Esta característica es insuficiente



TABLA II
CIRCUITOS DE PRUEBA

Frec. de Resonancia	Circuito 1	Circuito 2
Calculada	29399,5 Hz	47387,69 Hz
Medida Lock-in 7280	29399,4 Hz	47387,7 Hz
Medida Lock-in diseñado	29399,8 Hz	47387,2 Hz
	Fig. 9	Fig. 10

TABLA III
RECURSOS UTILIZADOS EN LA FPGA

	FPGA	DDS	Filtros
Celdas	1500000	10%	68%
Block RAM	60	4	14

para poder detectar diferencias de amplitud muy pequeñas. El problema sólo puede ser superado cambiando el conversor A/D, modificando componentes del hardware por uno de mayor resolución.

En caso de que el Lock-in quisiera utilizarse para otra aplicación distinta de la que fue diseñada, la frecuencia máxima de un barrido estaría limitada por la frecuencia de conversión del conversor D/A, el cual es muy lento porque trabaja en serie, y si bien no es un factor limitante en el rango de frecuencias de 20 a 70 kHz, puede serlo si se utilizan frecuencias muy altas, ya que puede alcanzar una frecuencia de muestreo máximo de 800 kHz con un clock de 20 MHz.

VIII. CONCLUSIONES

Se ha diseñado un Lock-in con funcionalidades básicas, que puede ser implementado en una plataforma basada en FPGA M1AFS1500 de la familia Fusion o en la M1A3PE1500 de la familia ProASIC3E [12]. Este equipo usa, como interfaz con el usuario, una computadora personal desde donde se seleccionan los parámetros de trabajo y se muestran los resultados. La comunicación con la FPGA es vía USB. Existe la posibilidad de agregar otras funciones al diseño con esta misma plataforma de hardware, ya que se dispone de casi un 30% de lógica programable y un 70% de memoria interna. El hecho de

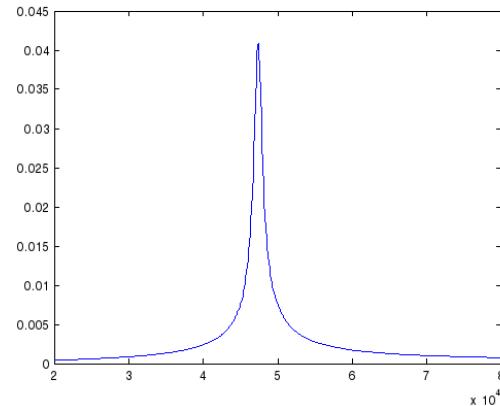


Fig. 10. Frecuencia de Resonancia Circuito RLC 2

incorporar dentro de la FPGA el DDS hace al diseño muy compacto y económico, si se compara con cualquier equipo comercial.

A. Trabajos Futuros

Agregar a la interfaz gráfica la posibilidad de configurar una mayor cantidad de parámetros por parte del usuario tales como:

- Frecuencia máxima y mínima de barrido.
- Cantidad de barridos
- Variación de frecuencia de cada barrido
- Cantidad de periodos de la señal de salida o tiempo de permanencia en la frecuencia de la señal de salida.
- Sentido del barrido (ascendente o descendente)

REFERENCIAS

- [1] M. I. Dolz, "Medición de superconductores mesoscópicos mediante micro-electro-máquinas," Ph.D. dissertation, -Universidad Nacional de Cuyo - Instituto Balseiro, Bariloche, 2009. [Online]. Available: <http://www.ib.edu.ar/>
- [2] S. R. Systems. (2006) About lock-in amplifiers, application note 3. [Online]. Available: www.thinkSRS.com
- [3] M. Meade, *Lock-in Amplifiers: Principles and Applications*. New York, NY: Peter Peregrinus LTD, ch. 2.
- [4] A. Cicuttin, M. L. Crespo, A. Shapiro, and N. Abdallah, "Building an evolvable low-cost hw/sw educational platform-application to virtual instrumentation," in *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, ser. MSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 77-78. [Online]. Available: <http://dx.doi.org/10.1109/MSE.2007.26>
- [5] *CoreDDS Handbook*, Microsemi, 2006.
- [6] *CoreFIR v4.0 Handbook*, Microsemi, 2006.
- [7] P. Longa and A. Miri, "Area-efficient fir filter design on fpgas using distributed arithmetic," *International Symposium on Signal Processing and Information Technology*, vol. 0, pp. 248-252, 2006.
- [8] (2012) The Synphony website. [Online]. Available: <http://www.synopsys.com/>
- [9] (2007) Rvi prototype board. ICTP-INFN. Microprocessor Laboratory. [Online]. Available: <http://mlab.ictp.it/research/rvi.html>
- [10] (2009) Fusion embedded development kit datasheet. Microsemi. [Online]. Available: <http://www.actel.com/products/hardware>
- [11] (2012) Digital-to-analog converters (dac) LTC1654. Linear Technology. [Online]. Available: <http://www.linear.com/product/LTC1654>
- [12] (2006) Proasic3e flash family fpgas datasheet. Microsemi. [Online]. Available: <http://www.actel.com/products/pa3series>

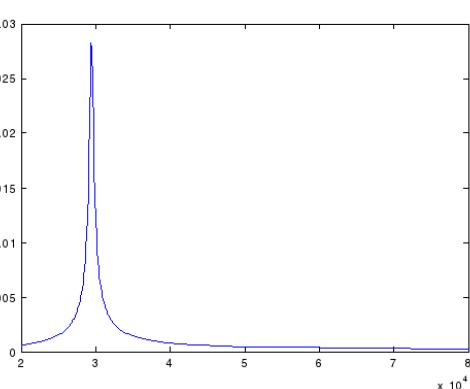


Fig. 9. Frecuencia de Resonancia Circuito RLC 1



Foro Tecnológico

Pósters

FPGA y HDLs



Single Event Upset en FPGA (SRAM)

Revisión de Técnicas de Mitigación

Pablo A. Salvadeo

Laboratorio de Computación Reconfigurable
FRM – UTN
Mendoza, Arg.
pablo.salvadeo@frm.utn.edu.ar

Ángel Veca

Instituto de Automática
FI – UNSJ
San Juan, Arg.
aveca@inaut.unsj.edu.ar

Rafael Castro Lopez

Instituto de Microelectrónica de Sevilla
CNM – CSIC
Sevilla, Esp.
castro@imse-cnm.csic.es

Resumen—La intención de este trabajo es explorar los efectos del SEU (Single Event Upset) sobre FPGA con tecnología SRAM. Tal problema afecta las celdas de memoria produciendo un comportamiento indeseado, dado que modifica la configuración del dispositivo. Se presentan: las causas del SEU, su impacto en las diferentes tecnologías FPGA, y técnicas de mitigación relacionadas. Además se evalúan de forma comparativa los resultados de éstas, teniendo en cuenta la robustez en función del costo.

Palabras claves-SEU; FPGA-SRAM;

I. INTRODUCCIÓN

Las tecnologías usadas en sistemas digitales cambian constantemente aumentando su densidad, su velocidad, disminuyendo su consumo, reduciendo las dimensiones del transistor y el voltaje de alimentación. En 1965 Gordon E. Moore observa que desde la invención del circuito integrado hasta ese año, el número de componentes en un chip se duplica cada año, y con ello se logra el costo mínimo, además predice que esta tendencia continuará por diez años más [1]. Tal proyección se conoce, desde 1970, como Ley de Moore [2]. En 1975 su creador indica que alrededor de ese año la tasa de crecimiento cambiará, se duplicará la cantidad de componentes cada dos años, y así seguirá otra década [3]. Se observa, que el crecimiento aún es exponencial pero su pendiente es menor. Esto sucede dado que la tecnología se vuelve el parámetro dominante, en particular el tamaño de la pastilla semiconductora y del transistor. En cambio, en la primera década la innovación en el diseño es el factor de mayor peso. La Fig. 1 muestra, utilizando escala semilogarítmica, la tendencia descripta por la Ley de Moore, que sigue siendo válida hasta el día de hoy, y se la formula comúnmente usando el término transistor en vez de componente. Tal validez, se debe a que la industria de los semiconductores la ha adoptado como regla a seguir debido a la creación del ITRS (International Technology Roadmap for Semiconductors) [4]. En 1992 se funda en EEUU la SIA (Semiconductor Industry Association), y en 1998 se asocia con sus homónimas: europea, japonesa, coreana, y taiwanesa, conformándose el ITRS. Este se publica los años impares y los pares sus revisiones, siendo su objetivo principal

estimar las necesidades de investigación y desarrollo que permitan a la industria de los semiconductores mantener sus tendencias los próximos quince años. Tales tendencias son seis: nivel de integración, costo por función, consumo de potencia, compactabilidad, y funcionalidad. Las cinco primeras están hoy en día relacionadas con la Ley de Moore, dado que ésta asegura la cantidad de transistores en un chip con costo mínimo. Pero considerando la tecnología reinante, CMOS, al volverse el transistor más pequeño también consume menos, conmuta más rápidamente y es más económico. Esta afirmación se conoce como Ley de Crecimiento de Dennard, presentada por primera vez en 1972 y más en detalle en 1974 [5, 6, 7]. En [8] se encuentran, además de los ya citados, varios artículos relacionados con el impacto de ésta ley en la industria de los semiconductores.

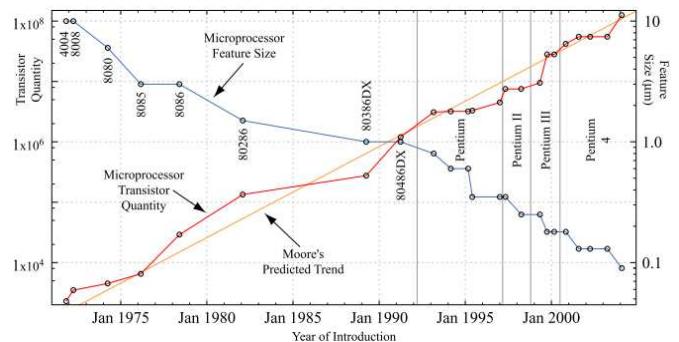


Figura 1. Ley de Moore sobre los microprocesadores de Intel [9].

El problema es que todas estas mejoras, por su naturaleza, incrementan la probabilidad de que se produzcan SEUs a causa de la radiación espacial. La mayoría de los dispositivos usados en aplicaciones terrestres, aéreas y espaciales, como las memorias, microprocesadores, ASICs (Application Specific Integrated Circuit) y FPGAs (Field Programmable Gate Array), pueden ser afectados por partículas ionizadas. Sin embargo, construir tales dispositivos con alta fiabilidad es costoso, y por tanto se necesitan técnicas de mitigación que disminuyan los efectos de la radiación sin elevar el costo del dispositivo.

Este trabajo fue parcialmente financiado por la Universidad Tecnológica Nacional.

A continuación, se discutirán las causas y efectos de los SEUs, su influencia sobre las diferentes tecnologías de FPGA, y las distintas técnicas de mitigación.

II. CAUSAS Y EFECTOS

La radiación espacial está compuesta por partículas cargadas, como por ejemplo: electrones, protones, e iones pesados, su interacción con los átomos causa excitación e ionización. Tales partículas provienen de: las llamaradas solares (protones, partículas alfa, iones pesados), el viento solar (electrones y protones), los rayos cósmicos (protones, iones pesados), dentro de los anillos de Van Allen (protones) y fuera de estos (electrones). Estas partículas cargadas, pueden causar efectos indeseados en el funcionamiento de los FPGA, al impactar con ellos. Entre los efectos se encuentra el denominado SEE (Single Event Effect) que se divide en dos categorías: soft (suaves) y hard (duros). Los suaves son dos: el SEU (Single Event Upset), que consiste en la conmutación de un bit en un elemento de lógica secuencial, y el SET (Single Event Transient), que implica un pulso transitorio de voltaje que afecta la lógica combinacional. En la tabla 1 se presenta una comparativa de las diferentes tecnologías de FPGA en relación a su robustez ante los SEEs suaves. Las familias comparadas son: SX-A (Anti-Fusible) [10] y ProASIC3 (Flash) [11] de Microsemi, y Virtex IV (SRAM - Static Random Access Memory) de Xilinx [12]. Se aprecia que los dispositivos SRAM están más adelante siguiendo la Ley de Moore, por lo tanto son: más económicos, más flexibles y permiten diseñar sistemas más complejos. Sin embargo, esta tecnología es más susceptible a la radiación espacial en comparación con la Flash y la Anti-Fusible, dado que es sensible al SEU en su memoria de configuración y de usuario. El problema de la corrupción de datos de usuario puede solucionarse usando códigos de corrección de errores, en cambio si se corrompe la memoria de configuración puede cambiar el comportamiento lógico del sistema, dado que esta memoria define la lógica y su interconexión. En la siguiente sección, se tratarán las técnicas de mitigación de SEU para la memoria de configuración del FPGA (SRAM), debido a la viabilidad económica de estos dispositivos.

TABLA I. COMPARATIVA DE LAS TECNOLOGÍAS DE FPGA.

Tecno.	Re-config.	Volatil	SEU RAM		PE ^a [Kb]	FF ^b [Kb]	RAM [Mb]	DC ^c [nm]
			Usuario	Config.				
Anti-Fusible	no	no	no	no	40	20	0,5	150
Flash	sí	no	sí	no	75	75	0,5	130
SRAM	sí	sí	sí	sí	180	180	6	90

a. Puertas Equivalentes (Look-Up Tables de 4 entradas). b. Flip-Flops. c. Dimensión del Canal.

III. TÉCNICAS DE MITIGACIÓN

Si se aplicara a la memoria de configuración de un FPGA (SRAM) el mismo esquema de mitigación de SEU, empleado en una RAM, se requeriría de una memoria del doble de capacidad o más, lo que implicaría un aumento sustancial en el costo del dispositivo haciéndolo inviable. Sin embargo si el sistema soporta periodos de inactividad, del orden de las decenas de microsegundos hasta los segundos, es posible

comprobar si el bitstream ha sido corrompido y si es así realizar una reconfiguración. El tiempo que implica ésta depende del área a cubrir, por ello se tiende a realizar reconfiguraciones parciales más rápidas que una total [13]. En la Fig. 2 se muestra una gráfica que relaciona el tamaño del bitstream y el tiempo de configuración para varios métodos. Así, se puede leer el bitstream cada cierto tiempo y calcular su checksum, si éste no coincide con el almacenado, realizar una reconfiguración parcial on-line utilizando un script con las utilidades de ISE: FPGA Editor y BitGen [14, 15], o seguir una de las técnicas off-line presentadas en [16]. En el caso de un SoC (System on Chip) se podría calcular el checksum de cada core y reconfigurar el corrompido, por ejemplo, siguiendo la técnica presentada en [17]. Esta se basa en la utilización de REPLICA (Relocation per online Configuration Alteration), un filtro capaz de realizar tareas de manipulación del bitstream, durante la reconfiguración, haciendo foco sólo en un core objetivo.

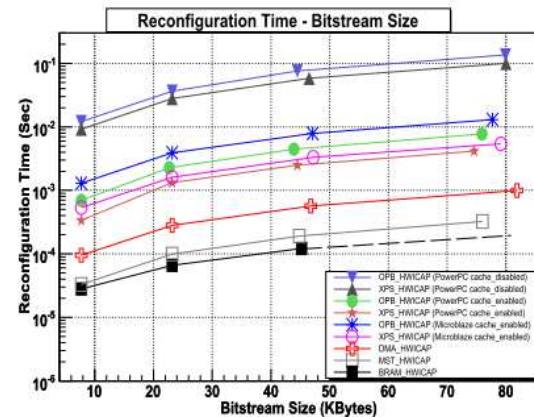


Figura 2. Tiempo de reconfiguración vs. tamaño de bitstream para distintos métodos con la interfaz ICAP [13].

En [16] se presentan, para FPGAs de la familia Virtex, dos técnicas de mitigación denominadas: Detección y Corrección de Marco (Frame), y Scrubbing, ambas basadas en la utilización de la interface SelectMAP que permite recuperar el bitstream (Readback) y reconfigurar el dispositivo. La memoria de configuración está compuesta por tres grupos de marcos, dos para la memoria de usuario, BRAM (Block RAM), y uno para los CLBs (Configurable Logic Blocks), siendo el direccionamiento y las operaciones de lectura/escritura de los primeros independiente de los segundos. La información de los marcos CLB define la configuración de las matrices de interconexión, los bloques de entrada y salida, y los valores de las LUTs (Look-Up Tables). Así, este método puede utilizarse para realizar una reconfiguración parcial sin interrumpir el sistema, siempre y cuando no se utilicen las LUTs como memoria distribuida o registros de desplazamiento, para ello pueden utilizarse los BRAMs. La primera técnica actualiza un cuadro sólo cuando detecta que es erróneo, ello requiere hardware adicional para implementar el algoritmo de verificación y memoria para almacenar constantes. La segunda, en cambio, no requiere hardware extra dado que simplemente reescribe sin verificar, sin embargo no lo hace cuadro a cuadro si no que reconfigura todo el conjunto de marcos CLB. La verificación bit a bit del bitstream requiere un archivo para almacenar el

actual y una máscara (del mismo tamaño), lo que triplica la memoria del sistema y por ello no se utiliza en aplicaciones espaciales. En cambio, se pude almacenar una palabra CRC (Cyclic Redundancy Check) de longitud fija para cada marco y luego compararla con la actual, reduciendo así sustancialmente la cantidad de memoria necesaria. Además, el cálculo del CRC se puede hacer en paralelo con el readback, por lo tanto todo el proceso no lleva mucho más tiempo que el empleado para recuperar el bitstream; por ejemplo: para un XC2VP100 (3500 marcos y 9792 bits/marco) el readback lleva 90ms. Si no es necesario actualizar el sistema, los valores CRC pueden ser almacenados en una ROM, en cambio si se aceptan actualizaciones deben estar en una RAM y ser actualizados junto con el sistema. Así, la Detección y Corrección de Marco necesita de un controlador compuesto por una FSM (Finite State Machine) que siga una secuencia para calcular y comparar los valores CRC y se comunique con la interface SelectMAP, en cambio el Scrubbing implica una FSM mucho más sencilla, un contador, y un decodificador para interactuar con la interface. En la Fig. 3 se muestran esquemas de los dos controladores.

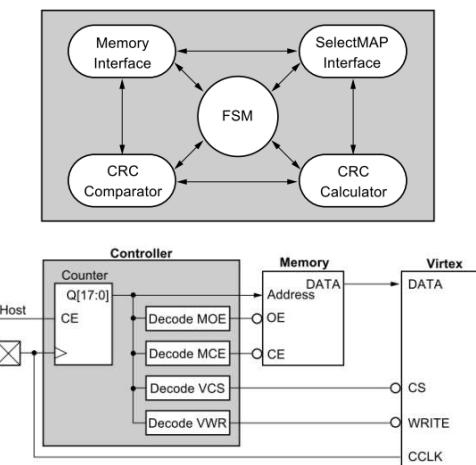


Figura 3. Controladores para: Detección y Corrección de Cuadro (arriba), y Scrubbing (abajo) [16].

La tasa de scrubbing puede ser determinada en función de la tasa esperada de SEU para una aplicación en particular, dependiendo esta: de la duración de la misión, la zona/s de operación y la sección de cruce estática por bit (determinada experimentalmente) multiplicada por la cantidad de bits del dispositivo. Sin embargo, la sección de cruce estática no determina la tasa de errores en un FPGA como lo hace en un ASIC, dado que un SEU en el primero puede o no implicar un error en el sistema, por tanto hay que considerar una sección de cruce dinámica. Una regla sencilla para obtener la tasa de scrubbing es multiplicar por diez la tasa de SEU, o sea diez scrubbing por cada SEU.

En la familia Virtex IV se introduce lógica ECC (Error Correction Code), basada en código Hamming, para detectar errores en uno o dos bits en los marcos, la cual se puede emplear en conjunto con las interfaces: SelectMAP, ICAP o JTAG, para mitigar SEUs [18]. Así, durante el readback la lógica calcula una palabra denominada síndrome empleando los bits del marco incluidos los ECC. Si los bits están intactos el síndrome es igual a cero, si ha cambiado uno, el último bit

del síndrome es uno y el resto indican la posición del cambio, si dos cambiaron, el último bit es 0 y los restantes no importan, si más de dos cambiaron, el síndrome es indeterminado. Si uno o dos bits son erróneos la salida de la lógica ECC se pone en alto para indicarlo. En [19] se utiliza esta lógica y la interface ICAP para construir un controlador capaz de detectar y corregir SEUs de un bit sobre Virtex IV y V, el mismo ocupa un BRAM y 182 slices (0,50% sobre 35.840 slices) en la primera y 60 slices (0,17% sobre 34.560 slices) en la segunda, siendo veinte veces más veloz que los controladores propuestos en [20] y [21]. En la Fig. 4 aparece el diagrama del controlador SEU desarrollado en [19], se aprecia que el BRAM está configurado como memoria de doble puerto.

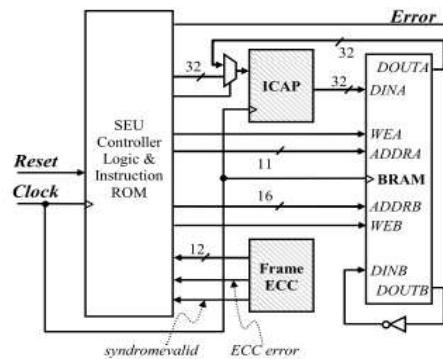


Figura 4. Controlador SEU sobre Virtex IV y V [19].

A través de los dos métodos mencionados anteriormente, se pueden mitigar los SEUs sin intervenir en el proceso de diseño del sistema en sí mismo. Sin embargo, aunque intervenir en el diseño conlleve un aumento de los recursos empleados, se ha mostrado que la combinación de ambas aproximaciones brinda mayor robustez ante los impactos de partículas cargadas [23, 24]. El método mayormente utilizado para ello es la Triple Module Redundancy (TMR) con votación, este consiste en triplicar la lógica, interconectarla para que ejecute en paralelo la misma tarea, y colocar a la salida un circuito de votación. En la Fig. 5 se aprecia la aplicación del TMR a un flip-flop D. Aplicar la técnica simplemente consiste en reemplazar todos los flip-flops D del diseño por el esquema mostrado.

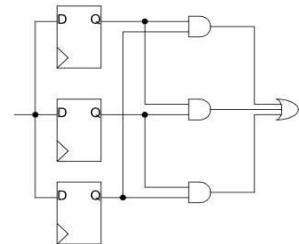


Figura 5. TMR con votación sobre un flip-flop D.

La TMR puede aplicarse considerando distintos niveles de abstracción del diseño, por ejemplo: dispositivos, cores, módulos, lógica. El primero requiere triplicar el número de FPGAs y agregar un dispositivo votador, lo cual implica un alto costo y además dependiendo de las interfaces usadas para la interconexión de los dispositivos, una gran complejidad. El segundo implica triplicar todos o algunos cores del sistema, si este



ocupa más de un tercio del área deberán repartirse los cores sobre diferentes dispositivos, si ocupa menos de la mitad se pueden duplicar los cores y colocar dos FPGA sin requerir de un chip extra para la votación. El tercer y cuarto nivel de abstracción requieren definir puntos críticos del diseño donde deba emplearse la técnica de mitigación. La ventaja de usar TMR en el cuarto nivel es que se puede mantener la sincronización entre los elementos redundantes si se realimenta la votación, esto permite detectar el error dentro de un periodo de reloj, así por ejemplo, el estado de un flip-flop no difiere del correcto por más de un ciclo. El TMR duplica aproximadamente el área y la potencia consumidas, lo que implica un importante aumento del costo. Sin embargo, si se emplea en el cuarto nivel, puede lograrse una reducción del costo aplicándolo sólo en las partes críticas del diseño, siempre y cuando ésta sea una porción pequeña del sistema, tal técnica se denomina TMR Selectivo y se presenta en [22, 23]. Esta origina un compromiso entre la robustez del sistema y el área ocupada, dado que al contrario del TMR que logra una tasa de error cero, éste no lo hace; por ejemplo, para un combinacional de 189LUTs el número de SEUs que indujeron errores fueron 83, con TMR el error cae a cero y el área crece un 200% (567LUTs), en cambio con TMR Selectivo el área sólo creció el 62,43% (307LUTs) pero el error disminuyó un 66,27% (28 errores) y no el 100%. El compromiso entre el área y la robustez del sistema es función de un umbral de probabilidad, que indica el valor por debajo del cual la probabilidad de la señal es cero, los errores anteriores son para un umbral de 0,5.

La implementación de TMR, al estar en el nivel de diseño, puede hacerse directamente en VHDL (VHSCIC Hardware Description Language), por ejemplo describiendo la lógica de votación y utilizando la sentencia `for generate` para triplicar los elementos. En [24] se presenta una metodología para aplicar la técnica mencionada combinada con scrubbing a BRAMs, sobre Virtex II y IV, empleando un programa denominado TMRTTool. La metodología se basa, en la utilización de un macro personalizable en VHDL (`bram_scrubber_pkg.vhd` y `bram_scrubber_bram_inst_bitw.vhd`), el cual consiste en una máquina de scrubbing y los BRAMs, y en el empleo del programa mencionado para aplicar al macro la técnica TMR.

IV. CONCLUSIONES

Se han presentado las razones para el empleo de los dispositivos FPGA basados en tecnología SRAM en misiones espaciales y su debilidad ante los SEUs en tales ambientes, mostrando además las técnicas actuales de mitigación. Estas son: a nivel de configuración Detección y Corrección de Marco (Frame) y Scrubbing, y a nivel de diseño TMR y TMR Selectivo. Se observa también que la aplicación en ambos planos mejora la relación consumo de recursos versus robustez. Sin embargo, queda en evidencia que deben hacerse nuevos aportes en el tratamiento de los SEUs sobre FPGA (SRAM) que alcancen mejores relaciones robustez - costo.

AGRADECIMIENTOS

Se agradece a los Dres. Adrián Faigon y José Lipovetzky de la Facultad de Ingeniería de la Universidad de Buenos Aires, por la organización del curso Ciencias Básicas aplicadas a la Microelectrónica, dictado en el marco de la EAMTA (Escuela Argentina de Microelectrónica, Tecnología y Aplicaciones).

REFERENCIAS

- [1] G. E. Moore, "Cramming more components onto integrated circuits", Electronics, vol. 38, no. 8, pp. 114-117, 1965.
- [2] G. E. Moore, "Excerpts from a conversation with Gordon Moore: Moore's Law", Intel, 2005.
- [3] G. E. Moore, "Progress in digital integrated electronics", Technical Digest, International Electron Devices Meeting, IEEE, pp. 11-13, 1975.
- [4] ITRS, Introduction, Executive Summary, pp. 1-14, 2009.
- [5] R. H. Dennard, F. H. Gaensslen, L. Kuhn y H. N. Yu, "Design of micron MOS switching devices", IEDM Tech. Dig., pp. 168-170, 1972.
- [6] R. H. Dennard, F. H. Gaensslen, H. N. Yu y V. L. Rideout, "Ion implanted MOSFETs with very short channel lengths", IEDM Tech. Dig., pp. 152-155, 1973.
- [7] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, y A. R. Leblanc, "Design of ion-implanted MOSFET's with very small physical dimensions", IEEE Journal of Solid-State Circuits, vol. 9, no. 5, pp. 256-268, 1974.
- [8] SSCS, "The impact of Dennard's Scaling Theory", Solid-State Circuits Society (SSCS) News, IEEE, vol. 12, no. 1, pp. 11-50, 2007.
- [9] Intel, "Microprocessor quick reference guide", 2002.
- [10] Microsemi, "SX-A Family FPGAs", 2007.
- [11] Microsemi, "ProASIC3 Flash Family FPGAs", 2011.
- [12] Xilinx, "Virtex-4 Family Overview", 2010.
- [13] Ming Liu, Kuehn, W., Zhonghai Lu, Jantsch, A., "Run-time Partial Reconfiguration speed investigation and architectural design space exploration", International Field Programmable Logic and Applications (FPL09) Conference, pp. 498-502, 2009.
- [14] C. Cameron, "Digital Duct Tape with FPGA Editor", Xcell, Xilinx, no. 66, pp. 54-57, 2008.
- [15] E. Eto, "Difference-Based Partial Reconfiguration", Application Note 290 (XAPP290), Xilinx, 2007.
- [16] C. Carmichael, M. Caffrey y A. Salazar, "Correcting Single-Event Upsets Through Virtex Partial Configuration", Application Note 216 (XAPP 216), Xilinx, 2000.
- [17] H. Kalte, G. Lee, M. Porrmann y U. Rückert, "REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems", 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), pp. 151-158, 2005.
- [18] Xilinx, "Virtex-4 configuration guide", Technical Report UG071, 2009.
- [19] B. F. Dutton y C. E. Stroud, "Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs", ISCA International Conf. on Computers and Their Applications, pp. 57-62, 2009.
- [20] L. Jones, "Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices", Application Note 714 (XAPP714), Xilinx, 2007.
- [21] J. Heiner, N. Collins, y M. Wirthlin, "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing", IEEE Aerospace Conf., pp. 1-10, 2008.
- [22] P. K. Samudrala, J. Ramos y S. Katkori, "Selective triple modular redundancy for SEU mitigation in FPGAs", In Military and Aerospace Applications of Programmable Logic and Devices (MAPLD), pp. 1-3, 2003.
- [23] S. Xiaoxuan y P. K. Samudrala, "Selective Triple Modular Redundancy for Single Event Upset (SEU) Mitigation", NASA/ESA Conference on Adaptive Hardware and Systems, pp. 344-350, 2009.
- [24] G. Miller, C. Carmichael y G. Swift, "Single-Event Upset Mitigation for Xilinx FPGA Block Memories", Application Note 962 (XAPP962), Xilinx, 2008.



Desarrollo del Ejercicio de Vigilancia Tecnológica en Dispositivos Lógicos programables Complejos CPLD

Ilber Adonayt Ruge Ruge

Docente Universidad de Cundinamarca
Grupo de Investigación GITEINCO
Fusagasugá, Colombia
iruge@mail.unicundi.edu.co

Diana Aponte Rada

Ingeniera Electrónica Universidad de Cundinamarca
Fusagasugá, Colombia
dianita_ram7@hotmail.com

Mayerly Benavides Lopez

Ingeniera Electrónica Universidad de Cundinamarca
Fusagasugá, Colombia
mayebenavidez@gmail.com

Resumen— Dentro del estudio realizado en este informe, se encuentra un análisis sobre las tendencias e inclinaciones del mercado actual en cuanto a los dispositivos lógicos programables complejos CPLD, además información detallada donde se relacionan temas de importancia, tales como: patentes encontradas desde el año 2001 hasta el año 2009; principales fabricantes, referenciando las familias y el número de dispositivos que pertenecen a las mismas; capacidades nacionales, resaltando los grupos de investigación en cuanto a instrumentación, control, automatización y microelectrónica, que aplican dentro de sus proyectos o investigaciones este dispositivo; y artículos que permiten medir, el volumen de aplicaciones y desarrollos que se han realizado y adelantan alrededor de esta tecnología en la industria electrónica (automovilismo, consumo, procesamiento de datos, etc.).

Palabras Clave— CPLD, lógica programable, patente, desarrollo, tendencias.

Abstract— Within the study in this report is an analysis of trends and inclinations of the current market in terms of complex programmable logic devices CPLD, plus detailed information which relate important issues, such as patents found since 2001 until today's leading manufacturers, referencing families and the number of devices that belong to them; national capacities, highlighting research groups in terms of instrumentation, control, automation and microelectronics, which apply within their projects or research this device; items to measure the volume of applications and developments made in advance about this technology in the electronics industry (auto racing, consumer, data processing, etc..).

Key Words— CPLD, programmable logic, patent, development, trends.

I. INTRODUCCIÓN

Debido al auge de los dispositivos lógicos programables y la marcada tendencia hacia la incorporación de nuevas tecnologías en el desarrollo de aplicaciones y productos existentes en el mercado, es preponderante para las empresas estar a la vanguardia de los cambios que se producen en su campo y sector de acción, con el fin de asegurar su permanencia en el mercado que les compete.

Para tal fin, se estableció un proyecto macro que permite desarrollar el estudio de vigilancia tecnológica (VT) orientado a la solución de necesidades que respectan al uso de dispositivos digitales en sistemas de control y/o instrumentación, bajo el nombre de *"Asociativamente establecer el servicio de vigilancia tecnológica en control e instrumentación del cluster de la industria electro electrónica de Bogotá y Cundinamarca"*, en ejecución por parte del CIDEI y la Universidad de Cundinamarca bajo el contrato 329037019598 firmado con Colciencias.

Dicho proyecto se divide en múltiples temas, abarcando en su totalidad, algunas tecnologías ya existentes y otras emergentes, entre las cuales se destaca como objeto principal de este artículo, el estudio de los dispositivos lógicos programables complejos (*Complex Programmable Logic Devices*, CPLD), permitiendo determinar las tendencias en cuanto a la incorporación de esta tecnología en los productos y/o aplicaciones presentes en el mercado actual de dispositivos electrónicos, dado que en un ambiente de globalización como el que actualmente se presenta, predominan las organizaciones que se anticipan al cambio, siendo importante precipitarse a las futuras innovaciones y desarrollos que sugiere la naturaleza de la economía cambiante, y por lo tanto estar la vanguardia de los cambios que se suceden en el actual mundo de la lógica programable, de ahí, la importancia de este trabajo.



II. ¿QUÉ ES VIGILANCIA TECNOLÓGICA?

La vigilancia es el esfuerzo sistemático y organizado por la empresa de observación, captación, análisis, difusión precisa y recuperación de información sobre los hechos del entorno económico, tecnológico, social o comercial, relevantes para la misma por poder implicar una oportunidad u amenaza para ésta. Requiere una actitud de atención o alerta individual. De la suma organizada de estas actitudes resulta la función de vigilancia en la empresa. En definitiva la vigilancia filtra, interpreta y valoriza la información para permitir a sus usuarios decidir y actuar más eficazmente [1].

La misión de la vigilancia tecnológica es informar sobre la aparición y evolución de nuevas tecnologías, el impacto posible sobre el mercado y la empresa, oportunidades y amenazas tecnológicas y de negocio y acciones futuras de los competidores

III. METODOLOGÍA

A. Factor Crítico de Vigilancia

El ejercicio de vigilancia tecnológica fue desarrollado mediante un proceso de estudio investigativo y analítico, que estableció de acuerdo a una metodología ya constituida, el estado del arte en cuanto a los CPLDs, basados principalmente, en la formulación de doce preguntas que abarcan de manera general, las posibles soluciones para realizar este estudio:

1. ¿Cuáles empresas son líderes (que patentan) en el uso de esta tecnología?.
2. ¿Existen grupos de investigación especializados en estos temas en las universidades?, de dónde?.
3. ¿Qué proveedores existen en el mercado?.
4. ¿Qué tipos de software son empleados?.
5. ¿Qué requerimientos mínimos se necesitan para emplear esta tecnología?.
6. ¿Qué aplicaciones son típicas con esta tecnología?.
7. ¿En qué otros sectores se emplea esta tecnología?.
8. ¿Quienes brindan capacitación?.
9. ¿Qué patentes existen en el tema?.
10. ¿Qué expertos nacionales o extranjeros hay en el tema?.

En la captura y procesamiento de dicha información, se consideró la utilización de los medios más adecuados para su búsqueda, con el fin de optimizar desde un comienzo, el proceso investigativo a desarrollar. Para cumplir este objetivo, se recurrió a un medio bastante conocido en la actualidad: herramientas de búsqueda basadas en Internet, que permiten obtener una representación muy útil y visual sobre cualquier tema de estudio, (remarcando la importancia creciente de Internet en la vigilancia tecnológica), tales como: motores de búsqueda, metabuscadores (generadores de cluster), servicios de

alerta, web invisible y bases de datos de artículos y patentes.

IV. RESULTADOS

A. Análisis de Patentes

El análisis de patentes se realiza como un indicador que permite observar las tendencias de los Dispositivos Lógicos Programables Complejos, CPLD en el mercado, con el objeto de identificar la evolución que presenta la tecnología, los expertos en el tema y las compañías que adelantan cambios y/o mejoras (innovaciones) significativas al respecto. Sobre un soporte de 4 bases de datos consultadas, especializadas en la búsqueda de patentes, USPTO, ESPACENET, FREEPATENTSONLINE y la WIPO (*World Intellectual Property Organization*) se han realizado las siguientes observaciones.

Recopilando la información de cada una de las bases de datos, en total se encontraron 86 patentes sobre CPLD en los que la tecnología es base o parte importante y 34 compañías a las cuales están asignadas. En la figura 1 pueden observarse estas estadísticas de acuerdo al número de patentes por compañía. Como lo indica la gráfica, la mayor cantidad de patentes se encuentran asignadas a empresas de poca participación en el mercado de los CPLD, es el caso de Advanced Micro Devices con el 6%, Samsung Electronics y LG Electronics, ambas con un 2% sobre el total de patentes encontradas. Además del 30% que representan esas otras compañías que han desarrollado una sola patente y que son menos conocidas.

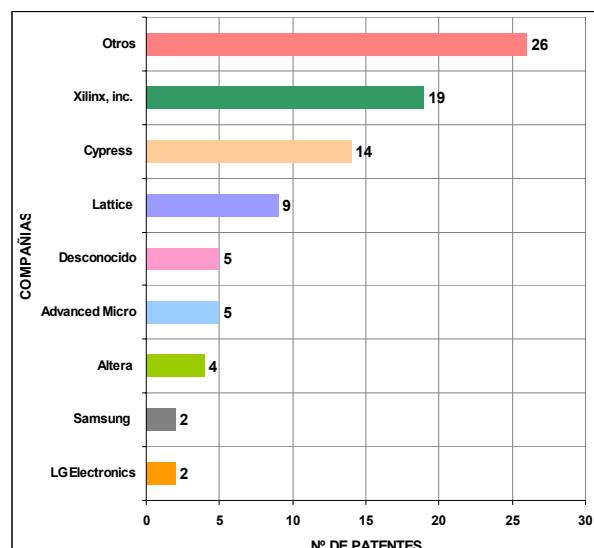


Figura 1. Número total de patentes por Compañía. Elaboración propia basada en datos de los sitios web de la USPTO, ESPACENET, FREEPATENTSONLINE y la WIPO

Por otra parte se identificaron cuatro fabricantes de CPLD reconocidos en el mercado que tienen asignadas patentes, Xilinx, Cypress, Lattice y Altera. En su orden, representan la cantidad de patentes que han desarrollado, Xilinx a la cabeza tiene el 22%, seguido de Cypress con el 16%, Lattice con un 10% y Altera con el 5%. Existe también un 6% que representa las patentes que no se han asignado a ninguna compañía.

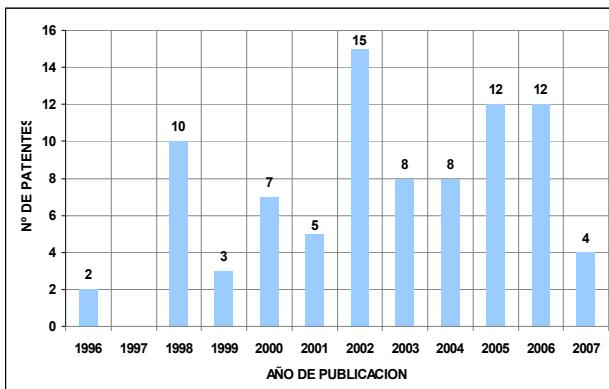


Figura 2. Número de patentes por año de publicación. Elaboración propia basada en datos de los sitios web de la USPTO, ESPACENET, FREEPATENTSONLINE y la WIPO

De acuerdo a la figura 2, desde el año 1996 hasta el año 2007 (parcial) el número de patentes publicadas por año ha sido notable, con un promedio de 5 patentes por año a excepción del año 1997 en el cual no se registro ninguna. Los años en los que se encontraron mayor número de patentes fueron el 2002 con un 17% y 2005 y 2006 ambos con el 14%. El aumento en el número de patentes para el 2002 fue bastante significativo con respecto al año inmediatamente anterior, triplicando la cantidad (de 5 año 2001 a 15 año 2002) que hasta ese momento se tenían. Para los años 2005 y 2006 aunque la cantidad de patentes se vio disminuida (20%) frente al año 2002, en esos dos períodos consecutivos se logro mantener el mismo margen de patentes, al igual que los años 2003 y 2004 solo que esta vez con un aumento del 50% sobre el número de patentes que se registraron en esos dos años. En el año 2007 lo que lleva transcurrido, la tendencia parece ser decreciente con una cantidad de 4 patentes, es decir, un 5% (parcial) sobre el total encontradas y una baja del 67% con respecto al año 2006. Sin embargo, esto no debe verse como el detrimento de la utilización de estos dispositivos ya que en general el balance de patentes publicadas por año es favorable y de acuerdo al comportamiento que se ha dado desde el año 1996 a 2007 y que se aprecia en la figura, luego de periodos en el que el número de patentes ha sido reducido, el siguiente suele ser de un aumento valioso.

Como se mencionó anteriormente se encontraron un conjunto de 34 compañías que han desarrollado patentes y el análisis de cada una revela un total de inventores que se encuentran vinculadas a estas, que asciende a 133. En la

figura 3 se indican el número de patentes que pertenecen a cada inventor.

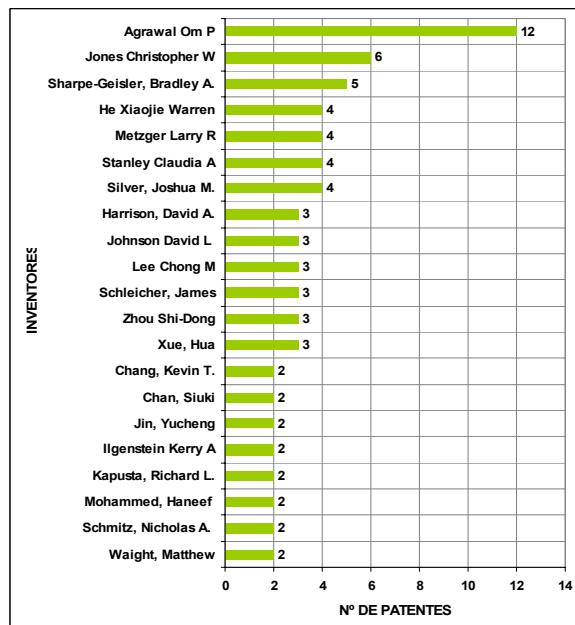


Figura 3. Número de patentes por inventor. Elaboración propia basada en datos de los sitios web de la USPTO, ESPACENET, FREEPATENTSONLINE y la WIPO

Finalmente para completar este análisis se ha realizado una clasificación de las patentes encontradas de acuerdo a la aplicación y según los datos del sitio www.wipo.int de la WIPO acordes a la clasificación internacional de patentes. Según esto las patentes encontradas clasifican dentro de las ramas de la física y la electricidad. De acuerdo a la clase que asigna el IPC (Clasificación Internacional de Patentes de la WIPO) versión 2007.01 el mayor número de patentes (36%) dentro de estas dos ramas se encuentran en el sector de aplicación de los circuitos electrónicos básicos. Los otros dos campos de aplicación en los que se concentra mayor número de patentes son cómputo y cálculo (30%) en donde se encuentran instrumentos, equipos y sistemas computarizados para el cálculo de condiciones existentes o anticipadas en el interior de un dispositivo o sistema real y el tratamiento o generación de datos de imagen, entre otros, que se estipulan en el sitio web de la WIPO.(figura 4)

B. Análisis de Fabricantes

El mercado de los dispositivos lógicos programables se divide en dos grandes grupos: Arreglos de Compuertas Lógicas programables en Campo, FPGA y Dispositivos lógicos programables Complejos, CPLD. Dentro de este mercado y de acuerdo a las estadísticas que se ponderan cada año, se establece que las compañías que lideran este segmento de los semiconductores son principalmente 7, XILINX, ALTERA, LATTICE, ACTEL, CYPRESS, QUICK LOGIC Y ATMEL, nombradas según el dominio que cada una posee y el grado de relevancia que esto les implica. De las 7 compañías fabricantes de PLD

mencionadas antes, se han identificado 5 que fabrican CPLD, estas son XILINX, ALTERA, LATTICE, CYPRESS Y ATMEL (figura 5). Las restantes se excluyen debido a que el fin de sus mercados se basa en FPGA y otros dispositivos como circuitos integrados de aplicación específica, ASIC, e IPcores (Es el caso de ACTEL que desarrolla una gama de dispositivos mixtos que integra cuatro CPLD y un FPGA, y que constituyen la familia de FPGA SX).

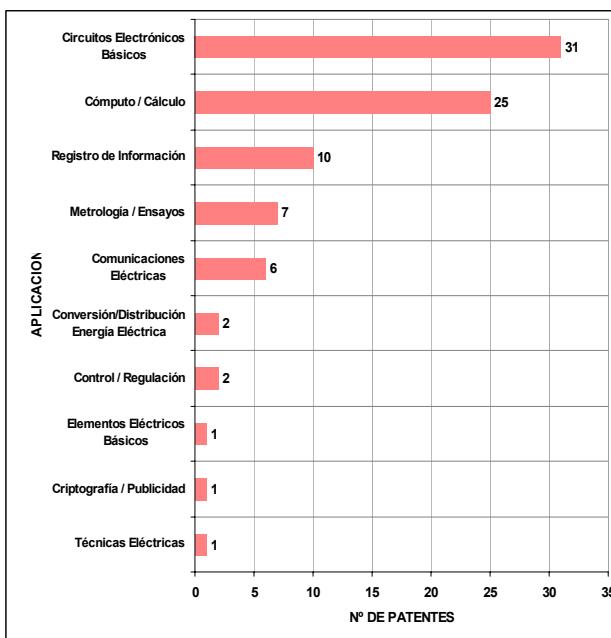


Figura 4. Número de patentes por aplicación de acuerdo la clase.
Elaboración propia basada en datos de los sitios web www.wipo.int

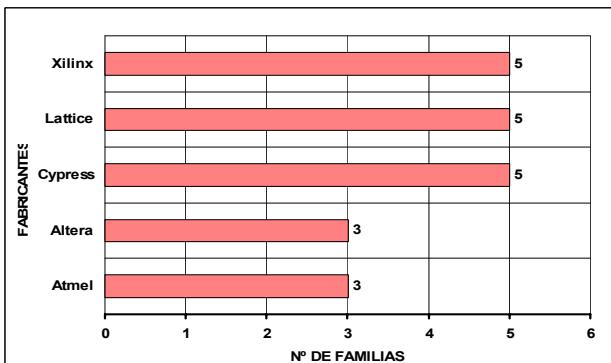


Figura 5. Fabricantes de CPLD y número de familias por fabricante.
Elaboración propia basada en datos del sitio Web de cada fabricante.

Actualmente se conocen 21 familias de CPLD que equivalen a un total de 119 dispositivos que se han desarrollado hasta este año. En la figura 6 se referencia las familias y el número de dispositivos que pertenecen a cada una de ellas. Sin embargo, en el mercado deben contarse solamente 18 familias y 109 dispositivos en total, debido a que Cypress ha anunciado que empezara a descontinuar sus familias de CPLD Boot EEPROM, FLASH370i y Quantum 38K.

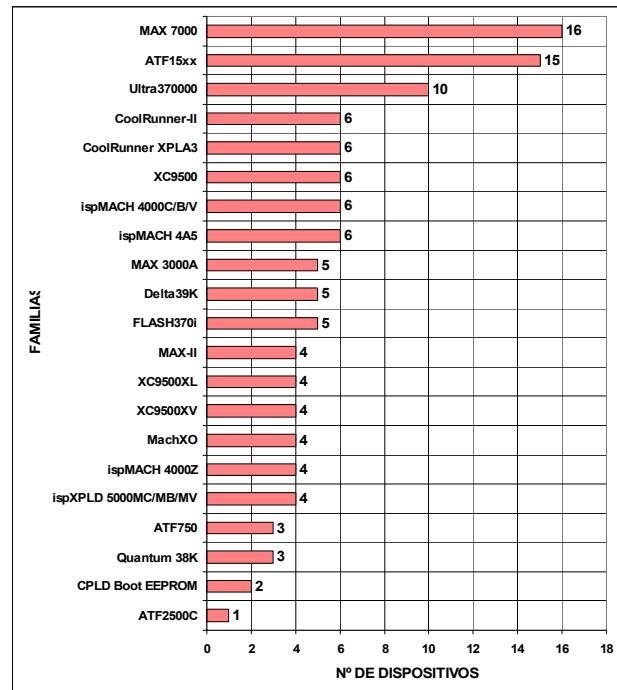


Figura 6. Familias de CPLD y número de dispositivos por familia.
Elaboración propia basada en datos del sitio web de cada fabricante

Este número de familias con respecto al año 2000 creció un 40% (figura 7), crecimiento que se dio como el resultado de cinco años de continuo movimiento alrededor de dicha tecnología. Así, en el año 2001 Lattice introduce al mercado las familias ispMACH 4000V/B/C seguido de Xilinx que en el 2002 lanza la familia CoolRunner-II.

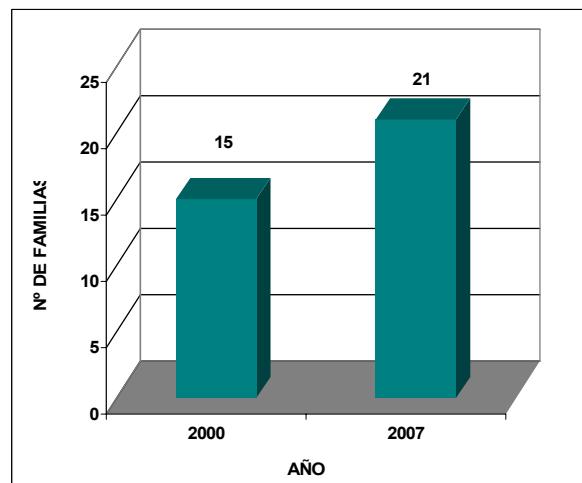


Figura 7. Número de familias de CPLD desde el año 2000 al 2007.
Elaboración propia basada en datos del sitio web de cada fabricante

Este incremento en el número de familias de CPLD marca un punto importante en la tendencia de estos dispositivos, el comportamiento que se viene dando desde el año 2000 deja ver dos aspectos interesantes: el primero que es el más claro, indica que la demanda de CPLD va en



crecimiento lo que ha obligado al desarrollo de nuevas familias. Si la demanda ha crecido es decir que el número de aplicaciones que se desarrollan sobre la base de CPLD también lo ha hecho, lo que deja entrever que esta tecnología pese a que compite con otras como lo son los FPGA y ASIC, esta brindando las mismas o mejores características que estos otros dispositivos para un amplio margen de aplicaciones y que son las que permite que mantenga firme su posición en el mercado.

V. ANÁLISIS DE RESULTADOS

Como resultado de la información consultada en las bases de datos examinadas, los sitios web de cada fabricante y los análisis elaborados al respecto, se encontraron las siguientes tendencias sobre los CPLDs.

De acuerdo al crecimiento del número de familias de CPLD que existen en el mercado, que pasó de ser 15 en el año 2000 a 21 en el año 2007 (parcial) se observa que la demanda de estos dispositivos se ha incrementando (si hay un mayor número de familias es porque existe una mayor utilización del dispositivo) y tiende a aumentar más. De esta forma se puede pensar que dentro de dos años o menos se lance al mercado una nueva familia de CPLD con mayores beneficios y menores restricciones que las que poseen las que existen actualmente. Ligada a estas características técnicas se encuentra que los CPLDs están evolucionando hacia una mayor densidad, menor potencia de consumo, mayor rendimiento y menor costo. Así, se han ido dejando atrás los dispositivos con un número de macroceldas por debajo de 32 y voltajes de alimentación de 5.0V y 3.3V. En contraste, las nuevas familias que se han ido desarrollando y que se encuentran en el mercado ofrecen densidades desde las 32 hasta 1700 macroceldas y núcleos de voltaje de 1.8V y 1.2V.

Al respecto de patentes se encontraron un total de 86 en las que la tecnología es base o parte importante, 34 compañías a las cuales están asignadas y alrededor de 133 inventores que individualmente o asociativa figuran en cada una de ellas. Según el comportamiento que se ha dado desde el año 1996 hasta el 2009 el número de patentes sobre CPLD es favorable para esta tecnología y el comportamiento se inclina a aumentar el número de patentes en el 2008 frente a las que se tienen hasta este momento para el año 2009. Además de esto, el hecho de que el número de las compañías que hacen el estudio de las patentes sea tan elevado se traduce en un mayor número de empresas que adelantan estudios de investigación e innovación alrededor de estos dispositivos, y por consiguiente marcan una tendencia de expansión de ese mercado con la inclusión de nuevas compañías (competidores).

Finalmente el aumento de familias y el balance positivo de patentes que se ponderan hasta el año 2007, refleja la ampliación del margen de aplicaciones que se soportan sobre la base de CPLD, conduciendo así a la reafirmación

de la importancia de esta tecnología dentro del mercado de la lógica programable a nivel mundial.

Sin embargo, las capacidades nacionales en Colombia dejan entrever que existe una gran desventaja con respecto al balance general proyectado en la industria internacional, sin desmeritar que existen instituciones como COLCIENCIAS, encargadas del desarrollo y la innovación en ciencia y tecnología, tanto para universidades como para pequeñas y grandes empresas del país.

En general, Colombia se proyecta hacia el crecimiento en grupos de investigación que fomenten el estudio, análisis, diseño e implementación de tecnologías como los CPLDs, al igual, que en empresas que utilicen este dispositivo como aplicación dentro de sus líneas de trabajo.

VI. CONCLUSIONES

Se muestra un incremento en el interés por el conocimiento de la tecnología de CPLD, adelantándose nuevas investigaciones e innovaciones por parte, no solo de las compañías que son líderes de ese mercado sino lo que es más importante por esas "otras" que integran y/o buscan integrar a la finalidad de sus mercados las soluciones y beneficios que brindan los CPLDs.

La mayor demanda de CPLD se presenta en aplicaciones móviles y portátiles como celulares, *Personal Digital Assistant*, PDA (ayudante personal digital), reproductores mp3, cámaras digitales, etc; y en segmentos del mercado como las comunicaciones y el procesamiento de datos.

En un balance global, las expectativas y resultados que se han obtenido sobre el crecimiento y penetrabilidad de los CPLDs dentro de la industria de los semiconductores y el mercado de los dispositivos lógicos programables, presupone un estado actual favorable de esta tecnología.

El alto número de patentes, distribuidores, familias y artículos encontrados al año 2007, reafirman el crecimiento y la acción integradora que han tenido los CPLDs en el mercado, que se traduce en un incremento de las aplicaciones y productos que giran entorno a los mismos. Sin embargo, no debe descartarse que este éxito signifique que en este punto la tecnología se encuentre en el pico máximo de su evolución y como consecuencia de aquí en adelante empiece a declinar, dejando así tomar ventaja a otras tecnologías como los *Digital Signal Processor*, DSPs (Procesador Digital de Señales).

A nivel nacional el conocimiento de esta tecnología no es nulo pero aun falta mucho por hacer alrededor de ésta, y proyectar actividades sistemáticas y de vigilancia que permitan estar al tanto de este tipo de tecnologías que posibiliten el crecimiento y fortalecimiento de la base empresarial de industria electro electrónica del país.



Pese a que la cuota nacional investigativa en cuanto a grupos y distribuidores no es grande, debe resaltarse que esta pequeña minoría permite promover el desarrollo científico, tecnológico e innovador de esta clase de dispositivos en el país y contribuir al crecimiento de la industria nacional.

VII. REFERENCIAS

- [1] COTEC. "Vigilancia tecnológica". Cotec, 1999.
- [2] Reporte anual Xilinx año 2007. Disponible On- Line: http://media.corporateir.net/media_files/irol/75/75919/XLN_X_AR_07/XILINX_AR07_Printed.pdf. Consultado en Agosto de 2007
- [4] Observatorio colombiano de ciencia y tecnología. Disponible On- Line: <http://www.ocyt.org.co/>. Consultado en Junio de 2007
- [5] Dispositivos Lógicos Programable, capítulo II. Iec Francisco Javier Torres Valle . Pág 10. Documento formato pdf.
- [6] LEADER Servicio de Diseño de Compañías Asociadas. Disponible On-Line: <http://www.latticesemi.com/support/designservices/index.cfm?source=topnav>. Consultado en Noviembre de 2007
- [7] ADI Engineering, Inc. Disponible On-Line: [http://www.adengineering.com/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY\\$0E\\$D7\\$](http://www.adengineering.com/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY$0E$D7$). Consultado en Noviembre de 2007
- [8] Digital Vortex, Inc. Disponible On-Line: [http://www.digitalvortex.us/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY\\$0E\\$D7\\$](http://www.digitalvortex.us/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY$0E$D7$). Consultado en Noviembre de 2007
- [9] EnCADIS Design. Disponible On-Line: [http://www.encadis.com/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY\\$0E\\$D7\\$](http://www.encadis.com/?CFID=27512904&CFTOKEN=12665000&jsessionid=ba304e3e787bY$0E$D7$) Consultado en Noviembre de 2007

Autores

Ruge Ruge, Ilber Adonayt. Ingeniero Electrónico egresado de la Universidad Pedagógica y Tecnológica de Colombia sede Sogamoso Boyacá en el año 2005, Magíster en Ingeniería de Control Industrial egresado de la Universidad de Ibagué Tolima en el año 2011. Actualmente se desempeña como docente en la Universidad de Cundinamarca en el Programa de Ingeniería Electrónica. Miembro del grupo de Investigación GITEINCO clase D ColCiencias. Temas de interés: Control inteligente, energías alternativas, dispositivos electrónicos programables entre otros.

Benavidez López, Nelly Mayerly. Nació en Quipile (Cundinamarca en Septiembre 22 de 1985). Ingeniera Electrónica, Universidad de Cundinamarca, Fusagasugá, Colombia, 2008. Miembro del grupo de investigación en Tecnologías de la Información y las comunicaciones de la Universidad de Cundinamarca GITEINCO.

Aponte Rada, Diana Marcela. Nació en Villega (Cundinamarca en Octubre 23 de 1985). Ingeniera Electrónica, Universidad de Cundinamarca, Fusagasugá, Colombia, 2008. Miembro del grupo de investigación en Tecnologías de la Información y las comunicaciones de la Universidad de Cundinamarca GITEINCO.

Diseño y Verificación On-Chip de un Multiplicador Serial Basado en Bases Normales sobre GF(2¹⁶³)

Fernando Aparicio Urbano-Molano

Departamento de Telemática, Facultad de Ingeniería Electrónica y Telecomunicaciones, Universidad del Cauca

Popayán, Colombia

faurbano@unicauca.edu.co

Abstract— Este artículo presenta el diseño y verificación funcional On-chip sobre un FPGA de un multiplicador serial basado en bases normales para criptosistemas de curvas elípticas sobre GF(2¹⁶³). El diseño esta descrito usando VHDL estructural/comportamental y sintetizado sobre una Stratix II (EP2S60F1020C3) usando Altera Quartus II, con el fin de comparar los resultados obtenidos con trabajos similares. La verificación y la depuración fueron sintetizadas sobre una Cyclone II (EP2C35F672C6). Los resultados de la simulación y la verificación en hardware muestran que el multiplicador diseñado presenta una buena relación área-velocidad comparado con trabajos similares y es apropiado para ser embebido en un criptosistema SoC.

Keywords- criptosistema de curvas elípticas, FPGA, multiplicadores en el campo de Galois (GF), Bases normales, Depuración, SignalTap II Logic analyzer.

I. INTRODUCCION

Las implementaciones de las operaciones aritméticas sobre los campos finitos determinan el desempeño en muchas aplicaciones como la teoría de codificación y los criptosistemas de clave-pública, en particular los basados en curvas elípticas (ECC) propuestos por Koblitz [1].

La implementación hardware de los ECCs puede llevarse a cabo en tres niveles de la jerarquía: la aritmética de campo finito, la operación de grupo elíptico, y la multiplicación escalar (o de punto). Se puede considerar un nivel adicional, que se refiere a las aplicaciones, específicamente a los protocolos criptográficos (cifrado y firma digital). Por lo tanto, para alcanzar implementaciones eficientes en hardware es indispensable realizar la mejor implementación de cada nivel. En este caso, el desempeño de un criptosistema depende de la eficiencia en las operaciones de la aritmética de campo finito y en particular de la multiplicación sobre GF(2^m) que es la que más recursos consume. De esta manera, el multiplicador es el bloque funcional más importante y debe implementarse eficientemente.

Aunque todos los campos finitos de la misma cardinalidad son isomórficos, la eficiencia de la aritmética depende de la selección de la base usada para la representación del elemento finito. Las bases más usadas para representar la aritmética de campo finito son las Bases Normales (NB) y las Polinomiales (PB). Las bases normales son más adecuadas para las implementaciones en hardware que las bases polinomiales debido a que las operaciones aritméticas en la representación de las bases normales están principalmente basadas en la rotación, el desplazamiento y la adición. Esta última operación es basada en compuertas OR exclusivas, las cuales son eficientemente implementadas en

hardware [3]. Una ventaja de las bases normales es que la operación de elevar al cuadrado se implementa con una rotación a la derecha de la representación binaria. Sin embargo, la multiplicación es una desventaja debido a que requiere mayor tiempo de computación. En la literatura existen varios algoritmos para realizar la multiplicación, pero la gran mayoría son orientados para implementaciones en software, los cuales requieren mayor tiempo computacional que las implementaciones en hardware.

Los criptosistemas ECC están incluidos en varios estándares internacionales, tales como ANSI y NIST [4]. Su principal ventaja sobre otros sistemas de clave pública como RSA, es la utilización de parámetros de tamaño menor pero con el mismo nivel de seguridad computacional [2]. Entonces teniendo en cuenta lo anterior, los criptosistemas ECC se utilizan en aplicaciones donde los recursos de computación son limitados, tales como *Smart cards* y teléfonos celulares.

Varios trabajos relacionados con la implementación de diferentes arquitecturas hardware de multiplicadores en el campo finito binario se encuentran en la literatura consultada. En [5] comparan tres multiplicadores seriales: Berle-Kamp, Massey-Omura y un multiplicador en bases polinomiales, los cuales fueron implementados para pequeños campos finitos GF(2⁸) en VLSI. En [6] consideran la implementación en VLSI de multiplicadores paralelos sobre GF(2^m) con grados de extensión m= 8, 16, 24 y 32 que no son primos. En [7] se presenta una modificación del algoritmo Massey-Omura, que consistió en eliminar la redundancia en el campo GF(2^m) definido por un polinomio irreducible, de esta manera la nueva arquitectura presenta menor complejidad en el diseño. Sin embargo estos multiplicadores son complejos de implementar en hardware, ya que la multiplicación en bases normales está determinada por el producto cruz de los términos de GF(2^m). Cuando m crece, la complejidad del espacio es proporcional a m². En [8], se presenta la implementación de multiplicadores paralelos a nivel de dígito sobre GF(2¹⁶³) usando bases normales Gausianas basados en el Algoritmo de Reyhani-Masoleh, con una arquitectura mediante registros, compuertas lógicas y máquina de estados; en [9] el diseño de multiplicadores en bases polinomiales basados en el Algoritmo de Elia-Leone sobre GF(2²³³), y [11] el diseño de multiplicadores en bases polinomiales y normales Gausianas basados en el Algoritmo de Li-Zhang y Multiplicación Convencional sobre GF(2¹⁶³) respectivamente, que presentan una buena relación área-velocidad.

Teniendo en cuenta lo anterior, se presenta el diseño, implementación y verificación hardware de un multiplicador serial para bases normales sobre GF(2¹⁶³) basado en el algoritmo propuesto en [9]. Este multiplicador se describe usando VHDL estructural/genérico. Se sintetiza sobre el



FPGA EP2S60F1020C3 y verifica sobre el EP2C35F672C6 usando la tarjeta DE2 de Altera.

La organización de este artículo es la siguiente: En la sección II se presenta una descripción del algoritmo de multiplicación para bases normales en el cuerpo finito $GF(2^m)$. En la sección III, se describe el procedimiento de diseño del multiplicador, usando como ejemplo $GF(2^5)$. En la sección IV, se muestran los resultados de simulación y síntesis, así como las comparaciones de los resultados, con trabajos similares. En la sección V, se muestra el procedimiento de verificación y depuración *on-chip* usando *SignalTap II* y finalmente, las conclusiones.

II. ALGORITMO DE MULTIPLICACIÓN

En esta sección se presenta una descripción detallada del algoritmo de multiplicación usando bases normales presentado en [10].

A. Multiplicador Serial

Se tiene que la multiplicación está dada por $C = \sum_{i=0}^{m-1} c_s \alpha_s = AB$ y se puede representar por la ecuación (1).

$$c_s = \sum_{i,j} a_{i+s} b_{j+s} \lambda_{ij}^{(0)} = \sum_{i,j} a_{i+s} b_{j+s} \lambda_{ij} = \sum_{j=0}^{m-1} \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{ij} \right) b_{j+s} \quad (1)$$

Se define un elemento x_{st} , para $0 \leq s \leq m-1$, en $GF(2^m)$ como

$$x_{st} = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{it} \right) b_{t+s}, \quad (2)$$

Entonces la t -ésima columna del vector X_t de X es

$$X_t = (x_{0t}, x_{1t}, \dots, x_{m-1,t})^T \quad (3)$$

donde $(x_{0t}, x_{1t}, \dots, x_{m-1,t})^T$ es la transpuesta del vector fila $(x_{0t}, x_{1t}, \dots, x_{m-1,t})$. Además la suma de todos los vectores columna X_t , $t = 0, 1, \dots, m-1$, es exactamente

$$(c_0, c_1, \dots, c_{m-1})^T, \quad (4)$$

Ya que $\sum_{t=0}^{m-1} x_{st} = c_s$.

Con el propósito de reducir la complejidad de las compuertas del multiplicador, los vectores columna X_t , son reorganizados y se reutilizan las sumas parciales en el cálculo. Sea $m-1 = 2v$ y $Y = (y_{st})$ una matriz $m \times m$ definida por la permutación de los vectores columna de X como sigue: cuando v es impar, Y se define como:

$$X_v, \dots, X_3, X_1, X_{m-1}, X_{m-3}, \dots, X_{m-v}, X_{v-1}, \dots, X_2, X_0, X_{m-2}, \dots, X_{m-v+1} \quad (5)$$

y cuando v es par, Y se define como:

$$X_v, \dots, X_2, X_0, X_{m-2}, \dots, X_{m-v}, X_{v-1}, \dots, X_3, X_1, X_{m-1}, X_{m-3}, \dots, X_{m-v+1} \quad (6)$$

Entonces la suma de todos los vectores columna Y_t , $0 \leq t \leq m-1$ de Y . Donde con $Y_t = (y_{0t}, y_{1t}, \dots, y_{m-1,t})^T$ es el mismo que la suma de todos los vectores columna X_t , $0 \leq t \leq m-1$, de X los cuales son $(c_0, c_1, \dots, c_{m-1})^T$.

Con el objetivo de realizar una arquitectura hardware del algoritmo del multiplicador presentado en la Tabla I, se calcula la suma de los vectores de diagonales de Y desplazados, en lugar de calcular la suma de los vectores columna de Y . Esto puede hacerse debido a las siguientes consideraciones. En la expresión de la matriz Y , hay exactamente $t-1$ columnas entre los vectores X_t y X_{m-t} . Además, la s -ésima entrada de X_t y la $s+t$ -ésima entrada de X_{m-t} tienen el mismo término a_{is}^s en sus sumandos. En otras palabras, de (2), se obtiene

$$x_{s+t, m-t} = \left(\sum_{i=0}^{m-1} a_{i+s+t} \lambda_{i,-t} \right) b_s = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{i-t, -t} \right) b_s = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{it} \right) b_s \quad (7)$$

donde la tercera expresión se obtiene desde la reorganización de la sumatoria sobre el subíndice i y la última se obtiene cuando $\lambda_{ij} = \lambda_{i-j, -j}$.

Por lo tanto X_{st} y $X_{s+t, m-t}$ tienen el mismo término $\sum_{i=0}^{m-1} a_{i+s} \lambda_{it}$ en su expresión, por lo tanto ahorrará el número de compuertas XOR durante el cálculo de AB .

TABLA I. ALGORITMO DE MULTIPLICACION

ENTRADAS: $A, B \in GF(2^m)$	
SALIDAS: $C \in GF(2^m)$	
1	$A = \sum_{i=0}^{m-1} a_i \alpha_i$ y $B = \sum_{i=0}^{m-1} b_i \alpha_i$
	Se cargan en los registros de m -bits respectivamente. $D_0, D_1, \dots, D_{m-1} \leftarrow 0$
2	Para $t = 0$ hasta $m-1$ $y_{s,s+t} + D_{s+t} \rightarrow D_{s+t+1}$ Fin Para. Para todo $0 \leq s \leq m-1$.
3	Después de la m -ésima iteración, se tiene $D_i = c_i$ para todo $0 \leq i \leq m-1$, donde $AB = \sum_{i=0}^{m-1} c_i \alpha_i$.

En el primer ciclo de reloj ($t = 0$), los valores de $D_{s+1} = D_s + y_{ss}$ son calculados simultáneamente para todo $0 \leq s \leq m-1$, por ejemplo, $D_1 = y_{00}$, $D_2 = y_{11}$, ..., $D_0 = y_{m-1, m-1}$. Cuando $t = 1$, los valores de



$D_{s+2} = D_{s+1} = D_{s+1} + y_{s,s+1}$ son calculados simultáneamente para todo $0 \leq s \leq m-1$, por ejemplo

$$D_2 = D_1 + y_{01} = y_{00} + y_{01},$$

$$D_3 = D_2 + y_{12} = y_{11} + y_{12}, \dots, D_1 = D_0 + y_{m-1,0} = y_{m-1,m-1} + y_{m-1,0}$$

Finalmente, en el m -ésimo ciclo ($t = m - 1$), los valores de $D_s = D_{s-1} + y_{s,s-1}$ son calculados simultáneamente. Esto corresponde a:

$$D_0 = D_{m-1} + y_{0,m-1} = y_{00} + y_{01} + \dots + y_{0,m-1} = c_0$$

$$D_1 = D_0 + y_{10} = y_{11} + y_{12} + \dots + y_{10} = c_1$$

.....

.....

$$D_{m-1} = D_{m-2} + y_{m-1,m-2} = y_{m-1,m-1} + y_{m-1,0} + \dots + y_{m-1,m-2} = c_{m-1} \quad (9)$$

En otras palabras, para una s dada, el valor final D_s se calcula secuencialmente en el siguiente orden

$$D_s = \underbrace{y_{ss} + y_{s,s+1}}_{D_{s+1}} + y_{s,s+2} + \dots + y_{s,s+i} = \sum_{i=0}^{m-1} y_{s,s+i} = c_s \quad (10)$$

Desde la ecuación 9 se observa que $y_{s-1,s}$ y y_{ss} , $0 \leq s \leq m-1$, son de la misma columna y_{ss} de la matriz Y . Debido a que Y se obtiene por una permutación de una columna de una matriz X , se concluye que $y_{s-1,s} = x_{s-1,s}$ y $y_{ss} = x_{ss}$ para algunos s en función de s . Por otra parte desde (2), se obtiene $x_{ss} = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{is} \right) b_{s+s}$, y

$$x_{s-1,s} = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{is} \right) b_{s+s-1} \quad (11)$$

Lo cual implica que $x_{s-1,s} (= y_{s-1,s})$ se obtiene por una rotación cíclica de una posición de los vectores $a_i s$ y $b_i s$ de la expresión $x_{ss} (= y_{ss})$. Debido a que esto puede hacerse sin ningún costo extra desde el punto de vista del hardware, todas las compuertas necesarias para implementar el circuito desde el algoritmo presentado en la Tabla I, son suficientes para calcular el primer ciclo de reloj (por ejemplo, $t = 0$) del paso 2 del algoritmo,

$$D_{s+1} = D_s + y_{ss}, 0 \leq s \leq m-1 \quad (12)$$

Recordando que, para cada s , hay una correspondiente s' (debido a una permutación) tal que

$$y_{ss} = x_{ss} = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{is} \right) b_{s+s} \quad (13)$$

Por ejemplo si $s' \neq 0$, y también x_{ss} no está en la columna 0 de X , entonces desde (2) y (7), se encuentra que

las compuertas XOR necesarias para calcular x_{ss} y $x_{s+s,m-s}$ (que son las entradas de la matriz Y) pueden compartirse. Se puede observar que $x_{ss} = \left(\sum_{i=0}^{m-1} a_{i+s} \lambda_{is} \right) b_{s+s}$ puede calcularse mediante una AND y al menos $k-1$ XOR, ya que la matriz de multiplicación (λ_{ij}) de una base normal de tipo k tiene al menos k entradas no cero para cada columna (fila). Por lo tanto el número total de compuertas necesarias para calcular todos los $y_{ss} = x_{ss}$ con $s \neq 0$ son $m-1$ ANDs, más $\frac{m-1}{2}(k-1)$ XORs.

Cuando $s = 0$, entonces el número de entradas no cero de λ_{i0} , para $0 \leq i \leq m-1$, es uno porque $\alpha \alpha_0 = \alpha^2 = \alpha_1$. Por consiguiente se necesita una AND y ninguna XOR para calcular x_{ss} con $s = 0$. Debido a que la suma $D_s + y_{ss}, 0 \leq s \leq m-1$, en la ecuación (12) se necesita una XOR para cada $0 \leq s \leq m-1$, la complejidad total de compuertas del multiplicador es m ANDs más como mucho $m + \frac{m-1}{2}(k-1)$ XORs [10].

III. DISEÑO DEL MULTIPLICADOR SECUENCIAL

A. Arreglo del Multiplicador

Sea β una raíz primitiva de orden $p = 2m+1 = 11$ de la unidad en $GF(2^{10})$ y sea $\alpha = \beta + \beta^{-1}$ un elemento normal óptimo de tipo II en $GF(2^5)$. Los términos de la multiplicación $C = \sum_{i=0}^4 c_i \alpha_i$ de $A = \sum_{i=0}^4 a_i \alpha_i$ y

$$B = \sum_{i=0}^4 b_i \alpha_i, \text{ para } 0 \leq i \leq 4, \text{ son}$$

$$\begin{aligned} c_0 &= (a_3 + a_4)b_2 + a_1 b_0 + (a_1 + a_2)b_3 + (a_0 + a_3)b_1 + (a_2 + a_4)b_4 \\ c_1 &= (a_4 + a_0)b_3 + a_2 b_1 + (a_2 + a_3)b_4 + (a_1 + a_4)b_2 + (a_3 + a_0)b_0 \\ c_2 &= (a_0 + a_1)b_4 + a_3 b_2 + (a_3 + a_4)b_0 + (a_2 + a_0)b_3 + (a_4 + a_1)b_1 \\ c_3 &= (a_1 + a_2)b_0 + a_4 b_3 + (a_4 + a_0)b_1 + (a_3 + a_1)b_4 + (a_0 + a_2)b_2 \\ c_4 &= (a_2 + a_3)b_1 + a_0 b_4 + (a_0 + a_1)b_2 + (a_4 + a_2)b_0 + (a_1 + a_3)b_3 \end{aligned}$$

En este caso, dos registros de desplazamiento son requeridos para calcular $C = AB$ usando una base normal óptima (ONB) de tipo II en $GF(2^m)$ para $m = 5$. Las Fig. 1 y 2 muestran el arreglo y la celda R_i del multiplicador diseñado.

Los productos parciales subrayados son los primeros en ser calculados. Además, las entradas de la diagonal (desplazada) tienen los términos en común.

B. Diseño del Multiplicador

La arquitectura hardware del multiplicador se basa en un arreglo y un contador, tal como se muestra en la Fig. 3. El contador genera la señal de control Fin para finalizar la multiplicación después de 164 ciclos de reloj y el arreglo del multiplicador realiza los productos parciales. El multiplicador tiene cinco entradas y dos salidas. Las entradas son: En, señal para habilitar de todos los registros del multiplicador; clk, la señal de reloj; load, señal para cargar los datos a multiplicar; reset, señal para borrar todos los registros del multiplicador; A y B, son los datos de 163 bits. Las salidas son: C, el resultado y Fin, la señal de control para indicar cuando está lista la multiplicación.

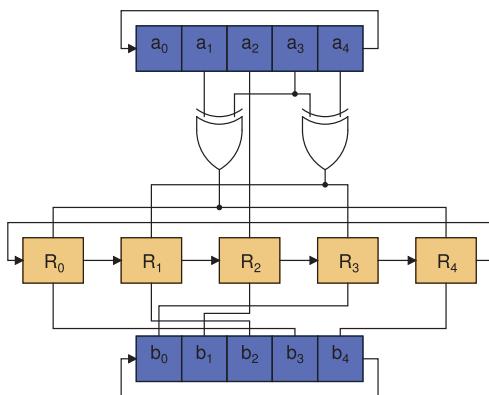


Figura 1. Arreglo del multiplicador usando una ONB de tipo II en $GF(2^5)$ para $m = 5$.

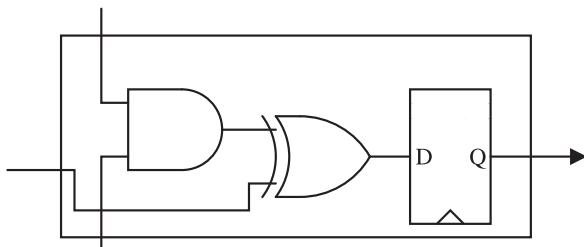


Figura 2. Bloque de la celda R_i

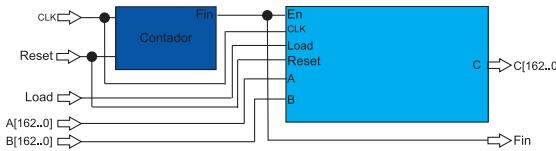


Figura 3. Diagrama de bloques del Multiplicador.

IV. RESULTADOS DE SIMULACIÓN Y SÍNTESIS.

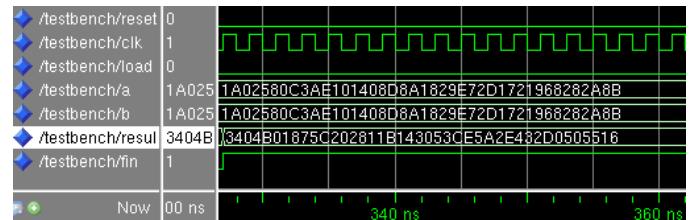
Con el propósito de verificar el funcionamiento del multiplicador se realizaron diferentes simulaciones en el cuerpo finito $GF(2^{163})$ usando varios vectores de prueba generados en *Maltab* al azar y utilizando *Modelsim Altera Starter Edition*.

La Fig. 4 presenta los resultados de simulación y estos son comparados con los obtenidos funcionalmente en *Matlab*, tal como se muestra en la Tabla II. En este caso, el vector de entrada es A.

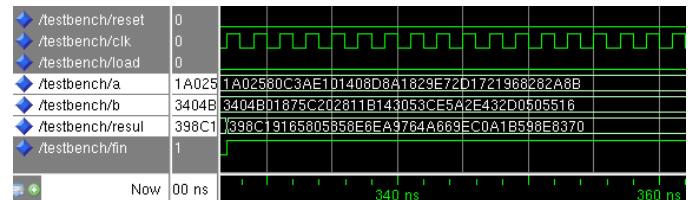
Con el propósito de conocer la relación entre el campo (m) y el área del multiplicador, en la Fig. 5 se muestran los recursos usados de acuerdo al campo binario m , es decir la cantidad de ALUTs y registros utilizados por el multiplicador.

TABLA II. RESULTADOS DE SIMULACIÓN PARA EL MULTIPLICADOR SOBRE $GF(2^{163})$

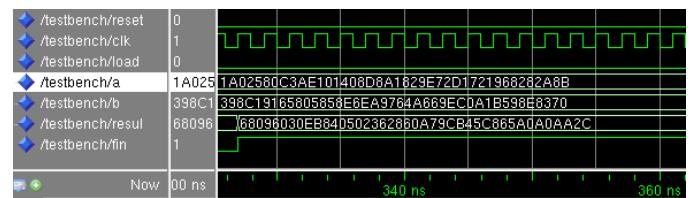
A	1A02580C3AE101408D8A1829E72D1721968282A8B
A^2	3404B01875C202811B143053CE5A2E432D0505516
A^3	398C19165805858E6EA9764A669EC0A1B598E8370
A^4	68096030EB840502362860A79CB45C865A0A0AA2C



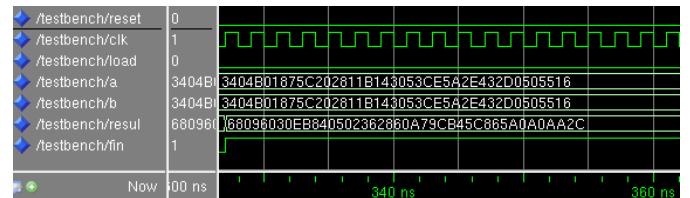
(a) Resultado de la simulación para $A \cdot A = A^2$



(b) Resultado de la simulación para $A \cdot A^2 = A^3$



(c) Resultado de la simulación para $A \cdot A^3 = A^4$



(d) Resultado de la simulación para $A^2 \cdot A^2 = A^4$

Figura 4. Resultados de las simulaciones y verificaciones del multiplicador de Kwon.

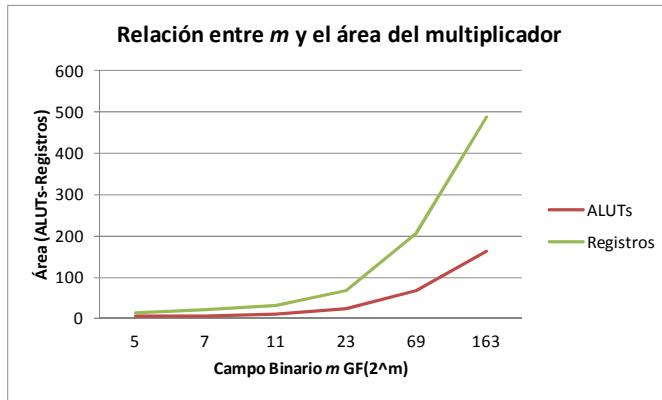


Figura 5. Relación entre el campo binario m y el área del multiplicador

En la Tabla III se presentan los resultados de la síntesis sobre el FPGA EP2S60F1020C3, debido a que no es posible realizar comparaciones entre diseños realizados con diferentes herramientas y FPGAs. Se comparan los resultados de [8], que para este caso se ha utilizado como referencia un multiplicador a nivel de dos dígitos, que realiza la multiplicación en aproximadamente 82 ciclos de reloj y [11]; Como se puede observar para un reloj de 100MHz, el diseño presentado utiliza menos recursos hardware, manteniendo una buena relación área – velocidad.

TABLA III. RESULTADOS DE LA SÍNTESIS PARA EL MULTIPLICADOR SOBRE $GF(2^{163})$ USANDO EL FPGA EP2S60F1020C3

	ALUTs	REGISTROS	$F_{MAX}(\text{MHz})$	TIEMPO DE OPERACIÓN (CLK = 10NS)
	TOTALES			
Este paper	175	497	392.83	1.635 us
[8]	903	652	NO REPORTA	842.4 ns
[11] Diseño 1	974	NO REPORTA	128.34	1.704 us
[11] Diseño 2	1249	NO REPORTA	178.13	1.67 us
[11] Diseño 3	1292	NO REPORTA	276.55	1.72 us
[11] Diseño 4	1269	NO REPORTA	386.12	1.74 us

V. VERIFICACIÓN Y DEPURACIÓN EN HARDWARE USANDO SIGNALTAP II LOGIC ANALYZER DE ALTERA.

Altera dispone de la herramienta software *SignalTap® II Logic Analyzer* para ayudar con el proceso de la depuración del diseño. Este analizador lógico es una solución que permite verificar el comportamiento interno de las señales, sin usar pines adicionales de E/S, mientras el diseño está corriendo sobre un dispositivo FPGA.

SignalTap® II Logic Analyzer es escalable, fácil de usar, y está disponible como un paquete único o incluido con la versión licenciada de *Quartus® II*. Este analizador lógico ayuda a depurar un diseño sobre un FPGA probando el estado interno de las señales en el diseño sin el uso de equipo externo. No requiere conexiones externas o cambios en los archivos de diseño para capturar el estado de los nodos internos (*signals*) o los pines de E/S de alguna familia específica de dispositivos tales como: Stratix®, Arria®, Cyclone® y HardCopy®. Todos los datos de las señales son capturados y convenientemente almacenados en un dispositivo de memoria, hasta que se deseen leer y analizar [12].

A. Descripción General del Sistema

Con el objetivo de mejorar el desempeño de los diseños hardware, en [13] se propone una verificación funcional *in-system*, como se muestra en la Fig. 6, el proceso es:

- A. Análisis temporal usando *RTL Viewer*.
- B. Mejoramiento temporal usando el *Floorplan Editor*.
- C. Depuración y verificación funcional *in-system* usando *SignalTap II*.

En este trabajo se utilizó únicamente la depuración y verificación funcional *in-system* usando *SignalTap® II*, ya que las versiones actuales de *Quartus® II* (versión 11), realizan un proceso de compilación y síntesis muy eficientes.

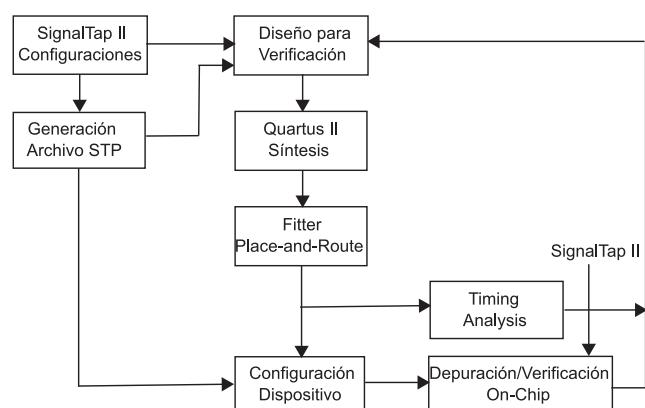
B. Resultados de la Depuración y Verificación usando SignalTap II

Siguiendo la metodología mencionada anteriormente, el proceso de depuración y verificación es muy simple, para este caso se usó la tarjeta Altera DE2 de *Terasic*.

La tarjeta de desarrollo Altera DE2 posee un FPGA Cyclone II 2C35 con 35000 elementos lógicos, el USB Blaster sobre la tarjeta para la programación, soporta el Modo JTAG Mode y el AS, un oscilador de 50 MHz, entre otros que la hacen apropiada para la depuración y verificación, debido al reducido tamaño del diseño.

Para la depuración y verificación se diseñó una máquina de estados que ingresara los datos de la Tabla II, con el objetivo de comprobar si los resultados eran equivalentes a los obtenidos en la simulación. Para tal fin la única entrada del sistema es el oscilador de 50 MHz y las salidas del sistema, el resultado de la multiplicación y la señal de finalización de la operación. Esta máquina de estados se simuló funcionalmente, con el fin de verificar y corroborar que los resultados esperados fueran correctos. Una vez comprobados todos los aspectos, se compila nuevamente, se asignan las señales y el oscilador, se programa el FPGA y se inicia la adquisición de datos con el *SignalTap® II*.

En la Fig. 7, se muestran los resultados de la verificación, utilizando un muestreo de 1K. Se pueden observar los resultados de las operaciones.





Type	Alias	Name	Value	click to insert
File		salida	0	3404B01975C202811B143053CE5A2E432D0505516h
File		Termina		

(a) Resultado de la verificación para $A^*A = A^2$

Type	Alias	Name	Value	click to insert
File		salida	166	398C19165806859E6EA9764A669EC0A1B598E8370h
File		Termina		

(b) Resultado de la verificación para $A^*A^2 = A^3$

Type	Alias	Name	Value	click to insert
File		salida	332	68096030EB840502362860A79CB45C865A040AA2Ch
File		Termina		

(c) Resultado de la verificación para $A^*A^3 = A^4$

Type	Alias	Name	Value	click to insert
File		salida	498	68096030EB840502362860A79CB45C865A040AA2Ch
File		Termina		

(d) Resultado de la verificación para $A^2*A^2 = A^4$

Figura 8. Resultados de las verificaciones funcionales on-chip del multiplicador de Kwon

En la Fig. 8, se puede observar que los resultados obtenidos con el SignalTap II son equivalentes a los de la Fig. 4 y por lo tanto, el multiplicador diseñado funciona correctamente y el resultado se obtuvo rápidamente.

VI. CONCLUSIONES.

Este artículo presenta la implementación y verificación en hardware del algoritmo de multiplicación presentado en [9] usando bases normales sobre $GF(2^{163})$. El multiplicador presenta una buena relación área - velocidad y puede ser usado en criptosistemas basados en curvas elípticas para soportar aplicaciones como *smart cards* y teléfonos móviles.

El multiplicador fue sintetizado sobre el FPGA EP2S60F1020C3 usando Quartus II 11.0 sp1 de Altera. El diseño fue verificado on-chip intensivamente para diferentes vectores de prueba sobre el FPGA y EP2C35F672C6 en la tarjeta Altera DE2 y los resultados fueron comparados con los obtenidos en ModelSim Altera Edition, logrando los mismos resultados. Se puede concluir, que el multiplicador es muy eficiente en área y velocidad, ocupando el 0.36% de ALUTs y el 1.02% de registros totales disponibles, frente a 1.88% de ALUTs y 1.35% de registros con respecto al de dos dígitos presentado en [8] y 2.62% de ALUTs aproximadamente para los presentados en [11]. Adicionalmente, la frecuencia máxima de operación dada por la herramienta de síntesis, es de 392.83MHz, obteniendo el resultado de la multiplicación en 417.48ns, mientras que en [11], la frecuencia máxima es 386.12MHz y el resultado se obtiene en 450.63ns, para el diseño más rápido pero con mayor área. Asumiendo, que todos los multiplicadores fueran seriales, el presentado en este artículo presenta una disminución considerable con relación a los recursos hardware utilizados y una mejora en la velocidad, utilizando la frecuencia máxima generada por Quartus II.

Adicionalmente, se comprueba que la depuración y verificación *on-chip* es un procedimiento bastante rápido comparado con los métodos de simulación utilizados.

El trabajo futuro, estará orientado a implementar nuevos algoritmos de multiplicación, en particular a nivel de dígito y criptoprocesadores basados en curvas elípticas sobre $GF(2^{163})$ y $GF(2^{233})$.

AGRADECIMIENTOS

Este trabajo ha sido patrocinado por *Altera Corporation* a través del programa universitario.

El autor agradece al Departamento de Telemática y al Grupo de Ingeniería Telemática de la Universidad del Cauca por brindar el tiempo para poder concluir satisfactoriamente este proyecto. A Vladimir Trujillo del Grupo de Bionanoeléctronica de la Universidad del Valle (Cali, Colombia), por sus aportes en el desarrollo de este trabajo y al director del grupo, el profesor Jaime Velasco Medina, por sus contribuciones.

REFERENCIAS

- [1] N. Koblitz, "Elliptic Curve Cryptosystems," Mathematics of Computation, vol. 48, Number 177, pp. 203–209, January 1987.
- [2] V. Trujillo, J. Velasco, J. López, "Multiplicador en el Cuerpo Finito $GF(2^{163})$ usando Bases Normales Gaussianas", Grupo de Bionanoeléctronica. EIEE. Universidad del Valle. X Workshop Iberchip, vol.1, fasc.1, 2004, p.1002 - 1012.
- [3] T. F. Al-Somani and A. Amin. "Hardware Implementations of $GF(2^m)$ Arithmetic using Normal Basis", Journal of Applied Sciences, Vol 6, Issue 6, 3rd ed., vol. 2. Asian Network for Scientific Information, 2006, pp.1362–1372.
- [4] "Digital signature standard," Tech. Rep. Technical Report FIPS PUB 186-3, National Institute of Standard and Technology, Junio 2009.
- [5] I. S. HSU, T. K. Truong, L. J. Deutsch and I. S. Reed, "A comparison of VLSI architecture of finite field Multipliers using dual, normal, or standard basis," IEEE Transactions on Computers, vol. 37, no. 6, june 1988, pp.735–739.
- [6] C. Paar y N. Lange, "A comparative VLSI Synthesis of Finite Field Multipliers", in Proceedings of the 3rd International Symposium on Communication Theory & Application, Lake District, UK, July 1995.
- [7] A. Reyhani-Masoleh and M. A. Hassan, "A New Construction of Massey-Omura parallel multiplier over $GF(2^m)$ ", IEEE Transactions on Computers, Vol. 51, No. 5, PP. 511-520, May 2002.
- [8] P. C. Realpe, V. Trujillo y J. Medina, "Implementación de un multiplicador paralelo a nivel de dígito sobre $GF(2^{163})$ usando bases normales gaussianas" En: Perú. 2007. Evento: XIII Workshop Iberchip, Editorial Hozlo Srl , p.253 - 256 , v.1 <, fasc.1.
- [9] V. Trujillo, J. Medina y J. López. "Design of polynomial basis multipliers over $GF(2^{233})$ " En: Perú. 2007. Evento: XIII Workshop Iberchip, Editorial Hozlo Srl , p.257 - 260 , v.1 <, fasc.1.
- [10] S. Kwon, K. Gaj, C. H. Kim and C. P. Hong, "Efficient Linear Array for Multiplication in $GF(2^m)$ using Normal Basis for Elliptic Curve Cryptography", En: M. Joye and J. J. Quisquater (Eds.): CHES 2004, LNCS 3156, pp. 76-91, 2004.
- [11] V. Trujillo, J. Medina y J. López. "Design of Gaussian Normal and Polynomial Basis Multipliers over $GF(2^{163})$ " En: Costa Rica. 2006. Evento: XII Workshop IBERCHIP Ponencia:Design of Gaussian Normal and Polynomial Basis Multipliers over $GF(2^{163})$ Libro: Diseño De Un Algoritmo En Hardware Para La Sincronización de Portadora, , p.174 - 176 , v.1 <, fasc.1
- [12] "Design Debugging Using the SignalTap II Logic Analyzer", Documentation: Quartus II Development Software. Volume 3: Verification. Section IV: Chapter 13. Noviembre, 2011.
- [13] J. M. Espinosa Durán , M. Vera Lizcano y J. Velasco Medina. "An Approach for Debugging and In-system Functional Verification for FPGA-Based System Design" Evento: 7º IEEE LATW, Buenos Aires, Argentina, Marzo 2006.



“Modulo Contador de Frecuencia en VHDL”

Guillermo M. Gancio.

Instituto Argentino de Radioastronomía I.A.R. CONICET.

ggancio@iar-conicet.gov.ar

Se describe el desarrollo de un contador de frecuencia implementado en VHDL; se decidió realizar la implementación en este lenguaje dadas las ventajas que ofrecen los dispositivos lógicos programables como las FPGA's ya que permiten utilizar diferentes aplicaciones operando de forma simultánea en un mismo dispositivo, brindando así una mayor integración y performance sobre la aplicación definitiva.

El modulo contador desarrollado en VHDL será aplicado sobre un hardware basado en una FPGA; el cual deberá tener la capacidad de trabajar en forma directa con frecuencias de hasta 150Mhz, trabajara con una resolución de 10Hz, capacidad de comunicación serie RS-232 y que pueda ser utilizado con una referencia de frecuencia tanto externa como interna. Para demostrar que el presente desarrollo es capaz de funcionar de la forma esperada se presentará la descripción del código VHDL, simulaciones de los diferentes sub-módulos y mediciones sobre el modulo implementado en una placa de desarrollo diseñada en el I.A.R. y basada en una FPGA de la firma XILINX.

Como nota final se muestra la implementación en un instrumento para el monitoreo de un sintetizador de frecuencia trabajando como segundo oscilador local en la frecuencia de 120Mhz. El cual es parte del radiotelescopio que opera el I.A.R. en una de sus antenas de 30mts de diámetro, utilizado para realizar observaciones radio-astronómicas en la línea espectral del hidrógeno neutro (HI, 1420Mhz).



Implementación en FPGA de un algoritmo para extrapolar funciones

Millán, I.; Villegas, R.; Salvadeo, P. A.

{ivanmillan36, ruyvillegas}@gmail.com; pablo.salvadeo@frm.utn.edu.ar

Laboratorio de Computación Reconfigurable

FRM – UTN

El trabajo consiste en la implementación del algoritmo de interpolación/extrapolación de Gregory-Newton sobre FPGA. La descripción se realizó utilizando VHDL. El método de extrapolación radica en generar una ‘tabla de diferencias hacia atrás’ a partir de muestras de una función. Dichas muestras se toman a intervalos de tiempo regulares. Luego se utiliza la fórmula de Gregory-Newton para predecir el valor de la función en el punto siguiente a la última muestra tomada. Para obtener valores de la función en puntos sucesivos, por ejemplo 5 lugares después de la última muestra tomada, se efectúan iteraciones hasta dar con el valor de la función en dicho punto. Para poder apreciar mejor el procedimiento se realizaron una serie de experimentos usando Excel, representando gráficamente cada función simultáneamente con su correspondiente predicción. También se hizo un programa en lenguaje C para probar la eficiencia del método según distintas variables. Esto permitió ajustar la cantidad de cálculos a efectuar para obtener una mejor predicción. Se demostró la efectividad del método al extrapolar con errores ínfimos el valor de funciones conocidas tales como: lineales, potenciales, trigonométricas. El método encuentra utilidad práctica al aplicarse en la extrapolación de señales físicas, las cuales son en principio desconocidas. La frecuencia de trabajo lograda permite seguir la tendencia de algunos fenómenos en tiempo real. La descripción en estilo algorítmico dio como resultado un consumo de recursos adecuado para que este core forme parte de un SoC (System on Chip).



“Adquisidor de señales implementado en VHDL y FPGA”

Guillermo M. Gancio.

1. Instituto Argentino de Radioastronomía I.A.R. CONICET.
ggancio@iar-conicet.gov.ar

El I.A.R. utiliza en una de sus antenas de 30mts de diámetro un receptor de uso radioastronómico para observar fenómenos en la línea espectral de HI - 1420Mhz; este receptor de tipo criogénico tiene en una de sus etapas de mezcla un detector cuadrático de potencia el cual entrega un pequeño nivel de tensión proporcional a la potencia recibida, esta pequeña señal es adquirida y procesada por un sistema adquisidor ADC-FPGA para realizar un monitoreo del estado del instrumento.

La sección analógica de este adquisidor está compuesta por un acondicionador de señal y un conversor analógico-digital; la sección digital esta implementada utilizando una FPGA de la firma XILINX®, la que permite realizar las tareas de adquisición y manejo del ADC, procesamiento de la señal mediante acumuladores, la detección de señales de control externas y manejo de la comunicación de forma serial mediante una interface RS-232 con la PC de monitoreo.

En el presente trabajo se expone una descripción del código implementado en la FPGA; el mismo está escrito en lenguaje VHDL, utilizando las herramientas de la firma XILINX®. La implementación del código fue dividida en distintos módulos que permiten evaluar de forma más sencilla y mejorar la performance del diseño; Estas evaluaciones se llevaron a cabo mediante simulaciones de los diferentes módulos previo a su implementación en hardware; Además se presentan medidas realizadas y observaciones radioastronómicas preliminares.



Codificación variable en el tiempo empleando mapa caótico

Maximiliano Antonelli y Luciana De Micco

maxanto@fimdp.edu.ar

Departamentos de Física y de Ingeniería Electrónica, Facultad de Ingeniería,
UNMDP. CONICET

En el diseño de un sistema de comunicaciones de datos inalámbrico tanto la confiabilidad de la transmisión como el nivel de privacidad son objetivos a cumplir. Han surgido últimamente técnicas de codificación que permiten además de aumentar la confiabilidad de la transmisión frente al ruido adicionar al sistema algún nivel de seguridad. En este trabajo se propone un esquema de codificación que cumple con ambos objetivos. Este sistema se basa en un mapa cuadrático bidimensional cuya salida presenta un comportamiento caótico y distintos atractores dependiendo de los coeficientes que se empleen.

Codificar significa básicamente tomar las 2^k palabras binarias de k bits que se pretende codificar, y asignarlas a algunos de los 2^n vectores de n bits. Esto se realiza como una función unívoca entre los 2^k y los 2^n vectores. Siendo regularmente $k < n$ existen más vectores de n bits que los que se tienen de k bits. Tradicionalmente el subgrupo de 2^n palabras código es fijo y la elección de los vectores de n bits se realiza empleando la menor redundancia, y maximizando la distancia o separación entre las palabras. En este trabajo la asignación de los vectores es variable ya que a cada una de las 2^k palabras a transmitir se le asigna un juego de coeficientes que genera salida caótica del mapa. Según la palabra a transmitir el sistema generará una salida determinada por los coeficientes y el valor inicial, esta será la palabra código correspondiente a la palabra a enviar. De esta forma, el subespacio de palabras código va cambiando a medida que la transmisión evoluciona.



“Implementación del proceso de transformación del espacio de color YCbCr al RGB en tecnología FPGA”

C. Gómez, Á. Anzueto y H. Mejía

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Instituto Politécnico Nacional

Distrito Federal, México

{cgomez10400, aanzuetor, hmejiaa0400}@ipn.mx

En el presente trabajo, se describe la implementación en tecnología FPGA la transformación del espacio de color YCbCr al espacio RGB basandonos en el estándar ITU-R BT.601-5.

Para la implementación del sistema se utiliza un sensor de video analógico tipo CMOS para adquirir imágenes con resolución de 640x480 en formato YCbCr. Una tarjeta digitalizadora VDEC-1 de la compañía Digilent, quien provee la información YCbCr en formato 4:2:2. Los datos digitales son ingresados a la tarjeta Spartan 3E, que contiene un FPGA XC3S500E, desarrollado por la compañía Xilinx, la cual, antes de realizar la transformación entre formatos de color, modifica el muestreo de los datos a 4:4:4.

Para cumplir los estándares de transmisión y codificación de señales de vídeo digital (ITU-R BT. 601) las tarjetas encargadas de la conversión de señal analógica a digital, entregan la señal de vídeo en formato digital en el espacio de color YCbCr. El estándar menciona una ecuación que realiza la transformación de RGB a YCbCr, la cual esta compuesta de una matriz de coeficientes re-normalizados de los vectores horizontales Y,Cb,Cr y una matriz de re-cuantificación de los vectores; ya que el estándar menciona que dichos vectores sólo pueden ocupar 220 niveles de luminancia y 225 niveles de crominancia, debido a que el resto de los valores son reservados para señales de sincronización en dispositivos analógicos, una re-cuantificación amplía los vectores a un rango de 256 valores. De acuerdo con esto se realiza una transformación inversa entre espacios de color como lo muestra la ecuación 1.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \left[\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix}^t \times \begin{bmatrix} 220/256 & 0 & 0 \\ 0 & 225/256 & 0 \\ 0 & 0 & 225/256 \end{bmatrix}^t \right]^{-1} \times \left(\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right) \quad (1)$$

Las operaciones de multiplicación de cada uno de los términos que integran a la ecuación 1 son desarrolladas en el FPGA utilizando bloques primitivos nombrados MULT18X18SIO. Para desarrollar todas las operaciones son necesarios 5 de este bloque de multiplicación y 5 bloques de sumatoria. Como tiempo máximo de retardo se tiene 12 ns. Este tiempo de retardo es el que transcurre a partir de que las señales en las entradas están presentes en el bloque de conversión y hasta que se obtienen los datos de salida.



“Sistema para el Acceso Remoto de Instrumentos Virtuales Reconfigurables”

Miguel A. Risco Castillo

Grupo de microelectrónica del Centro de Investigación y Desarrollo en Ingeniería (CIDI) de la Facultad de Ingeniería Electrónica y Mecatrónica, Universidad Tecnológica del Perú

mrisko@accesus.com

Los instrumentos virtuales reconfigurables pueden ser modificados con la intención de agregar nuevas funciones. Las innovaciones pueden darse al nivel del hardware o mediante el cambio de su software. Habitualmente el interfaz gráfico de usuario (GUI por sus siglas en inglés) se ejecuta en una PC que está físicamente conectada al hardware, esta conexión impone ciertas restricciones en cuanto a la distancia entre ambos componentes del instrumento. Existen soluciones en donde el hardware tiene un puerto de red, permitiendo que el GUI se ejecute remotamente según el alcance de la red a la que está conectado. Estas soluciones exigen que el hardware implemente el protocolo de comunicación lo que no siempre es una tarea sencilla. Además deja fuera la posibilidad de construir el instrumento virtual con tarjetas de costo reducido que no suelen contar con estos puertos.

Este trabajo propone un sistema para el acceso remoto a través de la red TCP-IP. El método no requiere que el dispositivo posea un puerto de red, sino, un puerto serie RS-232 o un puerto serie virtual por Bluetooth. Este es conectado a una computadora, en donde, un software especializado funciona como un servidor que provee información a los clientes conectados a una red Ethernet. Este sistema se ha probado en el desarrollo de instrumentos como trazadores de curvas, osciloscopios virtuales, generadores de onda, etc. Las librerías desarrolladas son de código abierto y pueden adaptarse a distintos tipos de hardware, la carga de trabajo es distribuida entre las tarjetas y la PC.