DISEÑO E IMPLEMENTACIÓN DE UN PROCESADOR DE PROPÓSITO ESPECÍFICO SOBRE UNA FPGA PARA LA INVERSIÓN DE MATRICES.

CAMILO ANDRÉS RIBERO CELIS

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍA FISICO-MECÁNICAS
ESCUELAS DE INGENIERÍA ELÉCTRICA ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2014

DISEÑO E IMPLEMENTACIÓN DE UN PROCESADOR DE PROPÓSITO ESPECÍFICO SOBRE UNA FPGA PARA LA INVERSIÓN DE MATRICES.

CAMILO ANDRÉS RIBERO CELIS

Trabajo de grado para optar al título de ingeniero electrónico.

Director

CARLOS AUGUSTO FAJARDO ARIZA, PhD(c)

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍA FISICO-MECÁNICAS
ESCUELAS DE INGENIERÍA ELÉCTRICA ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2014

DEDICATORIA

A Dios, por brindarme el entendimiento y la perseverancia durante el desarrollo de este trabajo. A mi madre y mi hermano, por el apoyo incondicional recibido en todo momento.

AGRADECIMIENTOS

Al director CARLOS A. FAJARDO ARIZA por su compromiso, disposición, asesoría y paciencia durante el desarrollo de este trabajo. A mis profesores que con su labor me ayudaron a construir como profesional y como persona. A mis compañeros CARLOS A. REYES M. y EDDIE A. PORRAS N., a quienes agradezco su ayuda en los momentos críticos. A la Universidad Industrial De Santander, por permitirme vivir este proceso de formación profesional.

CONTENIDO

LISTA DE FIGURAS.

LISTA DE TABLAS.

LISTA DE ANEXOS

RESUMEN

Título: Diseño e implementación de un procesador de propósito específico sobre una FPGA para la inversión de matrices[1].

Autor: Camilo Andrés Ribero Celis[2].

Palabras clave:

FPGA, Gauss-Jordan, Inversión de matrices, Pivoteo parcial.

Descripción:

La inversión de matrices es costosa computacionalmente, debido al gran número de operaciones aritméticas que deben realizarse en un algoritmo secuencial, como el algoritmo ejecutado en software. La inversión de matrices aparece con mucha frecuencia en problemas científicos y de ingeniería entre los cuales se pueden citar, las comunicaciones, problemas de optimización, sistemas oscilatorios, circuitos eléctricos, procesamiento de imágenes, mecánica inelástica, problema de Navier Stokes en una cavidad cúbica[3]. Este trabajo inició buscando como usar FPGAs para la solución de ecuaciones diferenciales con métodos numéricos, como el caso de las *Diferencias finitas*. Esto condujo a la necesidad de solucionar sistemas de ecuaciones lineales y esto a su vez, a la inversión de matrices. Se utilizó una metodología de diseño de tipo TOP-DOWN estableciendo tres niveles abstracción y posteriormente se usó el lenguaje VHDL para hacer la descripción en hardware del sistema. Este trabajo representa el diseño e implementación de un procesador de propósito específico usado para la inversión de matrices en una FPGA y basado en el algoritmo de Gauss-Jordan con pivoteo parcial. El dispositivo FPGA usado XC3S700AN permitió implementar cinco módulos de procesamiento paralelo en un sistema que puede operar a una frecuencia máxima de 108.69 MHz. En el capítulo 3, se realiza un análisis de desempeño en donde se muestran las expresiones que predicen el comportamiento del sistema, donde $CR = N^2[1.5 + 1.5fbm] + N[46.5 + 26.5fbm] - 0.5fbm^2 + 0.5fbm + 6$, es el número de ciclos de reloj necesarios para completar la inversión de una matriz. Finalmente como trabajo futuro se propone una modificación en la arquitectura que proporciona a GJ-FPGA[4] un aumento del desempeño de alrededor del 33%.

[1] Título de grado.
[2] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: PhD (c) Carlos Augusto Fajardo Ariza.

[3] Claude-Louis Navier y George Gabriel Stokes, Conjunto de ecuaciones en derivadas parciales que describen el comportamiento de un fluido.
[4] Algoritmo de Gauss – Jordan con pivoteo parcial implementado en hardware sobre un FPGA.

ABSTRACT

Title: Design and implementation of a special-purpose processor into an FPGA for matrix inversion[5].

Author: Camilo Andrés Ribero Celis[6].

Keywords:

FPGA, Gauss-Jordan, Matrix Inversion, Partial pivoting.

Description:

The matrix inversion is a highly computationally expensive process, given the large number of arithmetic operations that are required to execute the inversion process. The matrix inversion appears frequently in scientific and engineering problems, including communications, optimization problems, oscillatory systems, electrical circuits, image processing, inelastic mechanics, Navier Stokes problem in a cubic cavity[7]. This work began with a search for a way to use the FPGA devices to solve differential equations using finite differences. This search led to the need to solve systems of linear equations by the matrix inversion. We present the design and implementation of a special-purpose processor for the matrix inversion in a FPGA and based on the Gauss-Jordan algorithm with partial pivoting.

A TOP-DOWN methodology was used, which use three different levels of abstraction: system, algorithm and RTL level. The description of the system at RTL level was done in VHDL. The implementation of the design was done in the XC3S700AN FPGA. We implemented five modules working in parallel. The system can operate at a maximum frequency of 108.69 MHz. The design requires $N^2[1.5 + 1.5fbm] + N[46.5 + 26.5fbm] - 0.5fbm^2 + 0.5fbm + 6$, clock cycles to complete the inversion of a matrix of N files. Finally, as future work, we propose a new architecture GJ-FPGA[8], which could increment in the performance in about 33%.

[5] Degree project
[6] Faculty of Physical-Mechanical Engineering.Electrical, Electronics and Telecommunications School.
Advisor: PhD (c) Carlos Augusto Fajardo Ariza.
[7] Claude-Louis Navier and George Gabriel Stokes, set of partial differential equations that describe the behavior of a fluid.
[8] Algorithm Gauss - Jordan with partial pivoting implemented in hardware on a FPGA.

INTRODUCCIÓN.

El avance incontenible de la investigación en las diferentes disciplinas de la ciencia, exige a los miembros que pertenecen a una comunidad académica el desarrollo de más y mejores herramientas enfocadas al fortalecimiento de la investigación misma, constituyéndose en una relación simbiótica que tiene como resultado inevitable la construcción de conocimiento.

El interés de realizar este trabajo tiene su origen en la búsqueda de una forma para solucionar las ecuaciones diferenciales (E.D.) de forma numérica. En ese escenario aparece un método que puede expresar una ecuación diferencial en un conjunto de ecuaciones lineales (E.L.), ese es el caso del método de las *diferencias finitas*. De esta manera la E.D. puede ser resuelta de forma numérica, el único inconveniente es que normalmente los órdenes de estas matrices resultantes suelen ser muy grandes y por lo tanto es necesario contar con una herramienta que sea capaz de resolver estos sistemas de E.L. de una manera eficiente.

Posiblemente existan en el mercado herramientas de cálculo numérico con la capacidad suficiente para resolver dichos sistemas, solo que quizás están lejos del alcance de muchos investigadores debido a sus altos precios. No se puede afirmar que este trabajo sea la solución definitiva a este problema de cálculo numérico, solo que es un intento por contribuir de alguna manera a la solución usando las herramientas tecnológicas que tenemos a nuestro alcance en el contexto universitario. Es en este punto donde nace la inquietud de construir una herramienta que cumpla con esa necesidad y sea fácil de acceder a nivel económico para algún investigador interesado en los temas de cálculo numérico para la solución de sistemas de E.D. El tema es demasiado extenso para ser cubierto en este trabajo, por eso se decidió entonces atacar el problema por su

punto más neurálgico y proponer la arquitectura de un procesador de propósito específico que pudiera invertir matrices.

Este trabajo es la oportunidad perfecta como reto personal y profesional para hacer una implementación que permita aplicar los conocimientos adquiridos durante la formación académica y se constituya como un aporte que ojalá pueda algún día usarse en bien de la naciente comunidad de investigadores en las diferentes ramas de la ingeniería colombiana.

1. PLANTEAMIENTO Y DEFINICIÓN DEL PROBLEMA.

Los problemas de investigación son cada vez más complejos, situación que demanda sistemas de procesamiento de datos más eficientes. Típicamente estos problemas no pueden ser resueltos de manera analítica, por lo que se hace necesario el uso de sistemas de cálculo numérico, para resolver las ecuaciones que componen los modelos.

Algunos ejemplos de estos fenómenos son, el modelado de sistemas *biológicos*, *moleculares*, *poblacionales*, *electromagnéticos*, *análisis estructural, teoría de circuitos, predicción del clima* de *problemas de la mecánica del continuo*. La resolución de grandes sistemas de ecuaciones algebraicas lineales subyace en la solución numérica, llegando a constituir en muchos casos el principal factor de costo computacional [1].

Como alternativa actual es posible usar chips programables, que ponen características muy interesantes a disposición de un diseñador de sistemas digitales. Es el caso de las FPGAs[9], de poco costo, flexibles y absolutamente configurables por el usuario.

[9] FPGA: Field Programmable Gate Array.

Se toma la decisión de construir un procesador económico, orientado exclusivamente a la inversión de matrices, usando un dispositivo FPGA y basado en algún algoritmo que contara con una gran potencialidad de explotación del procesamiento paralelo. Se recopila información y se revisan arquitecturas implementadas en FPGAs para inversión de matrices. Arquitecturas como *Descomposición QR* [2], *Descomposición LU y de Cholesky* [3] *y Gauss - Jordan* [4], [5], [6].

El algoritmo escogido fue el de Gauss-Jordan con pivoteo parcial, por presentar el mayor potencial de paralización.

1.1 OBJETIVOS.

1.1.1 Objetivo general.

Diseñar e implementar un procesador de propósito específico sobre una FPGA dedicado a la inversión de matrices.

1.1.2 Objetivos específicos.

- Diseñar una arquitectura para el procesador de propósito específico tomando como base el algoritmo de Gauss – Jordan con pivoteo parcial.
- Configurar las unidades IP (Intellectual Property) de cálculo flotante de Xilinx, requeridas para el diseño del procesador de propósito general.
- Describir en VHDL de cada una de las unidades.
- Integrar las unidades funcionales y las unidades IP en el diseño.
- Medir el desempeño del procesador de propósito específico.

2. METODOLOGÍA DE DISEÑO Y CONSTRUCCIÓN.

2.1 EL ALGORITMO DE GAUSS-JORDAN CON PIVOTEO PARCIAL

El algoritmo de Gauss-Jordan con pivoteo parcial se compone de cinco procesos principales que se muestra en la Fig. 1, en donde los cuadros superpuestos simbolizan procesos paralelos. El avance del algoritmo se controla de acuerdo al número de iteraciones y para completarlo son necesarias tantas iteraciones como filas posea la matriz.

2.1.1 Descripción de procesos.

El proceso de *carga de parámetros* se utiliza para inicializar el sistema, cargando en registros información necesaria, como por ejemplo el número de filas de la matriz.

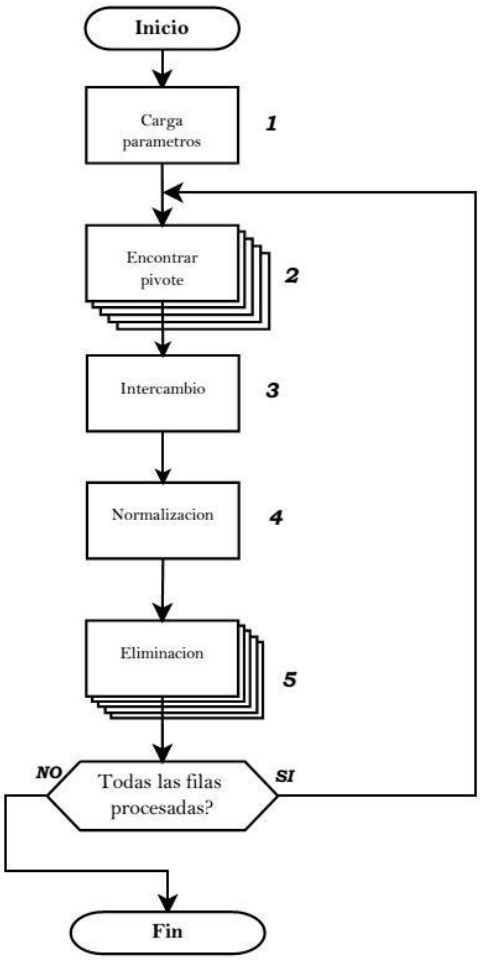
Encontrar pivote, es la implementación del pivoteo parcial, el cual consiste en encontrar el elemento de mayor magnitud en los elementos de una columna en particular. Este elemento es el denominado *pivote,* el cual una vez encontrado deberá ser debidamente registrado, tanto en su valor, como su localización.

Si, la posición de la fila en la cual se ha hallado el pivote no coincide con el número de iteración del algoritmo, la fila debe ser reubicada a dicha posición, de tal manera que se da un proceso de *intercambio* de filas. En caso contrario, se continúa normalmente.

La *normalización* consiste en tomar el elemento pivote y dividir todos los elementos de la fila que lo contiene. De esa manera aparece el *uno cabeza de fila* característico en este algoritmo.

El proceso de *eliminación* consiste en substraer de cada fila de la matriz, los valores de la fila normalizada pre multiplicada por un factor; excluyendo de esta substracción a la fila inmediatamente normalizada. Estos factores, son los elementos que se encuentran por encima y por debajo del *uno cabeza de fila*, en la fila normalizada. Dicho de otra manera, son los elementos que se van a eliminar.

Fig. 1. Algoritmo Gauss-Jordan con pivoteo parcial.



2.2 METODOLOGÍA DE DISEÑO GJ-FPGA.

En este diseño se usa la metodología Top-Down [7], la cual fracciona la complejidad del sistema, al descomponerlo descendentemente en tres niveles jerárquicos de abstracción. El nivel de sistema, el nivel algorítmico y el nivel RTL[10].

La aplicación de esta metodología en la construcción del sistema GJ-FPGA[11], consta de cinco pasos, que se ilustran en el diagrama de la Fig. 2.
Se utilizó un diagrama de bloques para la descripción a nivel sistema, múltiples diagramas ASM[12] lo definen a nivel algorítmico [8] y la descripción VHDL como nivel RTL.

2.2 DESCRIPCIÓN DEL DISEÑO GJ-FPGA.

En consecuencia con la metodología, el sistema GJ-FPGA está conformado por cinco bloques de procesos, cuatro del sistema de memoria y cinco bloques complementarios para un total de 14. La Fig. 3, muestra las relaciones entre dichos bloques.

La principal característica de del sistema GJ-FPGA es la paralelización de los procesos de *búsqueda y eliminación*. Para que esto sea posible la matriz no puede ser almacenada en un único bloque de memoria, sino que debe almacenarse en bloques separados.

[10] RTL: Register Transfer Level
[11] GJ-FPGA, es una abreviatura utilizada en este trabajo para referirse al algoritmo Gauss-Jordan implementado en hardware.
[12] Algorithmic State Machine.

Fig. 2. Aplicación de la metodología Top-Down a GJ-FPGA.

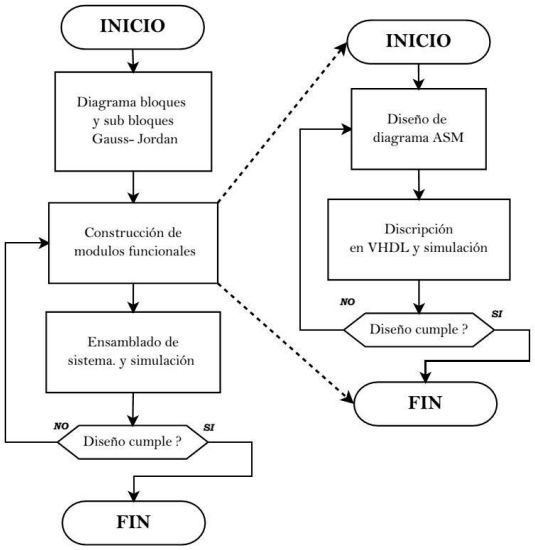
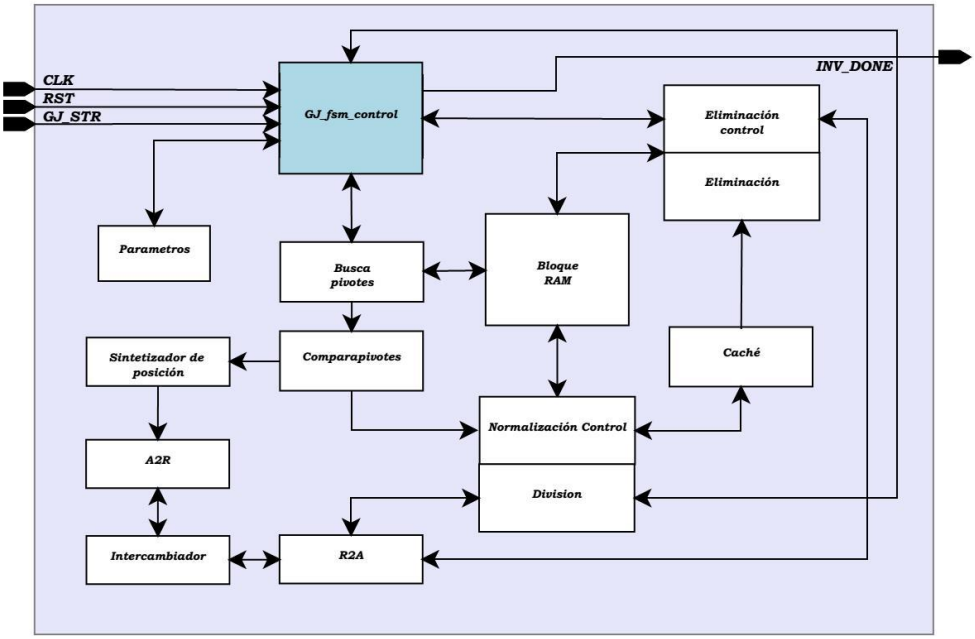


Fig. 3. Diagrama relacional de bloques.

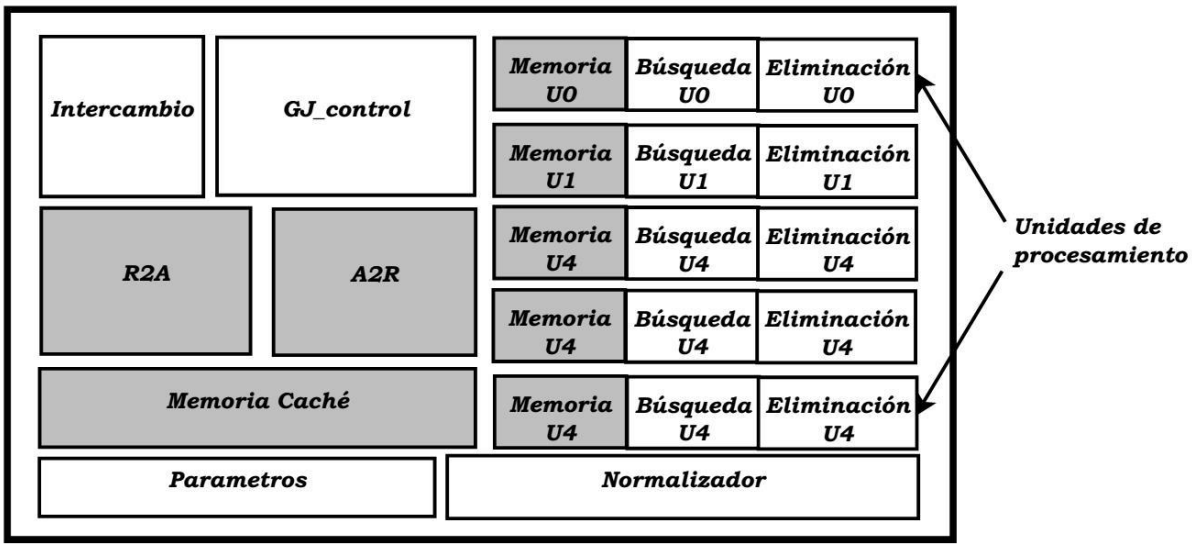


En la Fig. 4 se muestra el diagrama de bloques del diseño donde se muestran los bloques del sistema mostrando su arquitectura, en donde se han suprimido las conexiones para facilitar la comprensión.

Es importante destacar dentro de la arquitectura un grupo de unidades denominadas unidades de procesamiento, las cuales son las encargadas de las labores de paralelismo. Estas unidades están conformadas por tres bloques que son, un bloque de memoria, uno de búsqueda y uno de eliminación.

El número de unidades de procesamiento que pueden existir en una implementación en particular depende de la cantidad de recursos lógicos disponibles en un dispositivo. Para el caso de esta implementación el número de unidades es cinco. En adelante se usará una U para referenciarlas en este documento.

Fig. 4 Diagrama de bloques de GJ-FPGA.



2.2.1 Descripción del sistema de memoria.

El sistema de memoria está conformado por los bloques de memoria de cada unidad de procesamiento, R2A, A2R, y la memoria caché. En la Fig. 4, se encuentra resaltado en un color más.

Los bloques de memoria RAM permiten en conjunto albergar la matriz a resolver, almacenando individualmente in cierto número de filas que serán procesadas únicamente por la unidad que las contiene.

La distribución de las filas debe de ser uniforme para obtener el máximo rendimiento del sistema. En algunos casos esta distribución no será posible, dado que el número de filas debe ser un número entero para cada bloque, por lo tanto; sucederá que algún bloque o bloques tengan que albergar una fila adicional. El término *filas por bloque máximas (fbm)*, sirve para definir el número de filas totales que contienen estos bloques de memoria. Cuando la distribución es uniforme *fbm* es la misma para todos los bloques RAM. La ecuación 1, define el valor para *fbm*, donde N es el número de filas y U el número de unidades de procesamiento disponibles en el diseño.

$$fbm = \begin{cases} \frac{N}{U} & si\ U\ divide\ exactamente\ a\ N \\ \left[\left[\frac{N}{U}\right]\right] + 1 & en\ caso\ contrario. \end{cases} \quad (1)$$

Adicionalmente, existe un bloque llamado memoria *caché*, el cual tiene como función, almacenar los valores normalizados de una fila en particular. La memoria caché permite acceder a estos datos de manera rápida, manteniendo esta información disponible para todos las unidades de procesamiento. Si no existiera este bloque, los datos de la fila normalizada deberían ser leídos de un bloque de memoria y para cada unidad de manera independiente, lo cual sería altamente

inconveniente, dado que generaría un cuello de botella en la ejecución del proceso de eliminación, destruyendo por completo el concepto de procesamiento paralelo.

2.2.1.1 Posición absoluta y relativa.

La posición relativa es la posición natural de las filas de una matriz y que se nota con un número entero de cero a N-1, siendo N el número de filas de la matriz. La posición absoluta es la ubicación real dentro del sistema de memoria en el que se encuentra cada fila en GJ-FPGA. Para poder ubicar una fila en particular, es necesario tener como datos, el número de bloque y la posición que tiene esa fila dentro de dicho bloque. Esta arquitectura de memoria fraccionada obedece exclusivamente a la necesidad del paralelismo, porque permite acceder a distintas filas de la matriz de forma simultánea, situación que no sería posible si la matriz estuviera almacenada en una única memoria.

Básicamente R2A *(entiéndase como Relativa a Absoluta)*, es una memoria que contiene las posiciones absolutas de cada fila, asociadas a una dirección de memoria que corresponde con la posición relativa. El contenido de cada sector de memoria contiene dos valores, el número de bloque y el número de fila dentro del bloque, para cada una de las filas de la matriz. Al finalizar el proceso de inversión la matriz inversa se recupera consultando las posiciones de la memoria R2A.

La memoria A2R *(entiéndase como Absoluta a Relativa),* contiene la las posiciones relativas de cada fila, asociadas a una dirección de memoria generada a partir del bloque y la fila, por un bloque complementario llamado sintetizador de Posición. El contenido de cada sector de memoria contiene el valor de la posición relativa de cada fila.

Las memorias R2A y A2R funcionan como un mecanismo de conversión entre los dos tipos de direcciones. Cuando el proceso de búsqueda encuentra un pivote, se

apoya en la información del bloque y la fila asociados al pivote, para averiguar a través de A2R cuál es la posición relativa que tiene dicha fila en la matriz, para poder determinar si hay o no lugar a intercambio de filas.

El *intercambio* de filas es un proceso que reubica virtualmente las filas de acuerdo con cada iteración del algoritmo. La reubicación de las filas no mueve físicamente los datos, solamente modifica los valores de las posiciones relativas y sus equivalentes absolutas.

2.2.1.2 Estructura de datos dentro de los bloques de memoria.

Los datos dentro de los bloques de memoria están organizados por filas y de manera consecutiva. Cada fila tiene concatenada su correspondiente fila de la matriz identidad en concordancia con el algoritmo de Gauss-Jordan.

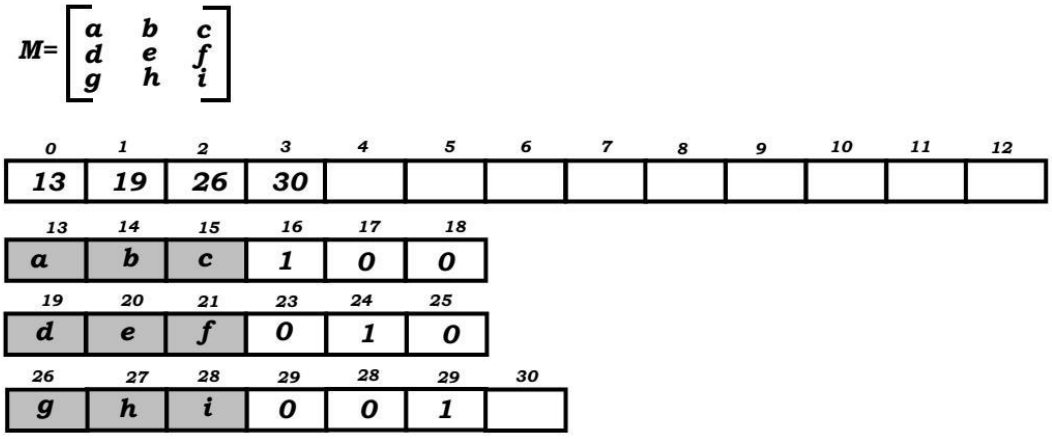
Los primeras 13 posiciones de memoria de cada bloque están reservadas para ser punteros. Estos punteros indican la posición de inicio de cada fila. Es necesario un puntero adicional para indicar en donde termina la última fila. De esta manera, almacenar un numero X de filas en un bloque de memoria requiere X + 1 punteros. La Fig. 5 representa la estructura de datos de los bloques de memoria.

Se ha representado en la Fig. 5, la forma como se almacena una matriz M de tres filas. La matriz inversa, se encontrará ubicada en las posiciones de memoria en donde inicialmente se encuentra la matriz identidad.

2.2.3 Descripción de bloques complementarios

GJ-control es un bloque que coordina, la secuencia del algoritmo, y es una máquina de estados que a través de señales de control indica el comienzo y verifica el final de cada proceso.

Fig. 5. Estructura de datos.



El *sintetizador de posición*, es un bloque encargado de generar una dirección que pueda ser aplicable a la memoria A2R, en función del bloque y la fila dentro de un bloque de memoria para los pivotes hallados, con la finalidad de poder obtener su posición relativa.

Comparapivotes es un bloque complementario a los bloques de búsqueda de pivotes incluidos dentro de cada unidad de procesamiento. Su finalidad es seleccionar uno de los cinco pivotes candidatos, para ser el pivote de una columna en particular.

El bloque de *división* es complementario al proceso de normalización. Por uno de sus dos puertos ingresan los datos de una fila particular y por el otro el valor del elemento pivote. El resultado de esta división se almacena en la memoria caché.

La *unidad de eliminación* es complementaria al bloque del proceso de *eliminación*. Contiene dos unidades de cálculo flotante, una para producto y una para resta.

2.2.2 Ejecución del algoritmo.

El primer proceso en ejecutarse es *parámetros*, que inicializa el sistema, cargando en registros la información para cada unidad U del número de filas, el número máximo de filas por bloque (*fbm*) y el número de filas de la matriz (*N*) a invertir. El término *número máximo de filas máximo fbm*, aparece debido a que no todos los módulos de procesamiento U, tendrán asignadas las mismas filas por bloque, por lo tanto; *fbm* es el mayor número de filas asignadas a algún bloque.

Cada unidad U tiene asociado un bloque de *búsqueda*, así se logra una búsqueda global mediante búsquedas parciales en una misma columna. De esta manera cada unidad encontrará el elemento de mayor magnitud de las filas por bloque asignadas previamente. Esto ocasiona que al final se tenga un valor parcial por cada unidad U, de los cuales se escogerá nuevamente el de mayor magnitud en una última *comparación* y este valor se considera como *pivote*.

Una vez hallado el pivote se verifica la posición de la fila en cual se encuentra, para saber si hay lugar a intercambio. El mecanismo de verificación consiste en comparar la posición relativa de la fila en donde se halla el pivote con el valor de la iteración del algoritmo. Este valor se halla en un registro llamado contador de pivotes ó PCO. Si la posición relativa de la fila en donde se halla el pivote es diferente, entonces debe haber intercambio. Si hay lugar a *intercambio de filas,* se ejecuta el proceso para reubicar la fila en la posición correcta, de lo contrario se continúa con normalmente. El intercambio de filas, modifica solamente la posición relativa, no mueve físicamente los datos.

El intercambio de filas en GJ-FPGA, es la implementación del pivoteo parcial, que busca mejorar la precisión del resultado al reducir la propagación de los errores de aproximación. Es una característica difícil de pero resulta muy positiva para el sistema [5].

La *normalización* convierte en uno el pivote que fue hallado anteriormente y que toma la fila en donde se ha hallado el elemento pivote y realiza una operación aritmética de división entre la fila completa y el valor del elemento pivote hallado previamente, convirtiendo así, al elemento que antes era el pivote, en un *uno* cabeza de fila. Este conjunto de datos es almacenado en una memoria *caché*, la cual permite tener disponible los datos normalizados, sin tener que acceder nuevamente a la memoria tras cada eliminación. Este concepto de caché ahorra tiempo durante el proceso de *eliminación.*

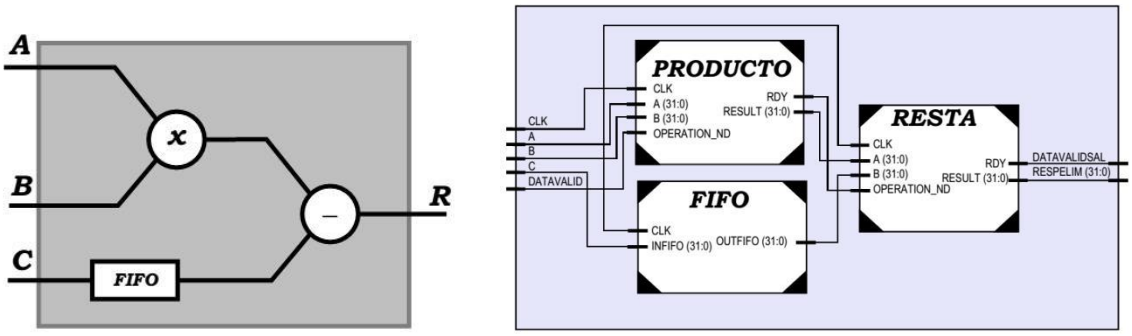
Finalmente el proceso de *eliminación* convierte en ceros los elementos de la columna, por encima y por debajo de la posición de la fila en donde se halló el elemento pivote, como es habitual en la eliminación de Gauss-Jordan, pero en este caso realizada paralelamente.

La *eliminación* se lleva a cabo ingresando tres tipos de datos a la unidad de eliminación; La fila normalizada proveniente de la memoria caché, la fila a eliminar, proveniente del bloque de memoria y el factor de eliminación, que es el primer elemento no nulo de cada fila y que permanece estático durante la eliminación para cada fila.

En la Fig. 66 se muestra el esquema y el diagrama de bloques del módulo de eliminación, en donde el conjunto de datos de fila normalizada ingresan por el puerto **A,** el factor de eliminación por **B** y la fila a eliminar ingresan por el puerto **C**. El registro FIFO[13] utilizado en la unidad de eliminación es una manera sencilla de compensar el retraso de la latencia en la unidad de producto. Este retraso es una consecuencia de la ejecución segmentada propia de las unidades de coma flotante.

[13] First Input First Output

Fig. 6. Unidad de eliminación.



a) Esquema. **b) Bloques.**

El resultado obtenido en el puerto **R** se almacena nuevamente en los bloques RAM respectivos, sustituyendo los valores que la misma fila que ingresó por el puerto **C** previamente.

2.3 IMPLEMENTACIÓN DE GJ-FPGA.

El diagrama de la
Fig. **3** (Pág. 16), muestra el flujo de datos entre módulos del sistema de forma bastante abstracta y dada la complejidad el diagrama de bloques, se ha simplificado y fraccionado, mostrando las conexiones más relevantes en las Fig. 77 y Fig. 88.

Múltiples diagramas ASM fueron utilizados para el diseño de los módulos hardware a nivel algorítmico. Estos diagramas se encuentran disponibles en la sección de anexos.

En concordancia con la metodología, se describió el sistema GJ-FPGA a nivel RTL en base a los diseños ASM y usando el lenguaje VHDL[14]. La descripción en VHDL de GJ-FPGA también se puede encontrar en la sección de anexos.

La implementación de este diseño se ha hecho en el dispositivo FPGA XC3S700AN de la serie SPARTAN 3AN y el software de desarrollo ISE 10.1 de XILINX, Inc. Las características del dispositivo se muestran en la **¡Error! No se ncuentra el origen de la referencia.**1.

Como mecanismo de verificación de las matrices inversas, se utilizó la implementación de una *UART* [9] que tiene licencia LGPL[15]. Este módulo fue modificado y acoplado al sistema para enviar información de forma serial hacia una computadora. La transmisión se activa de forma manual una vez finalizada la inversión.

Tabla 1. Características XC3S700AN, Spartan 3AN [10]

Device	System gates	Equivalent logic cells	CLBs	Slices	Distributed RAM Bits	Block RAM Bits	Dedicated multipliers	DCMs	Maximum User I/O	Max Differential I/O Pairs	Bitstream Size	In-System Flash Bits
XC3S700AN	700K	13,248	1,472	5,888	92K	360K	20	8	372	165	2,669K	8M

2.4 CONFIGURACIÓN DE UNIDADES DE COMA FLOTANTE

Las unidades de producto, resta y división que realizan las operaciones en coma flotante fueron sintetizadas con una herramienta llamada *Core Generator* provista por XILINX, Inc. En la Tabla 22 se muestran los detalles de la configuración realizada para estas unidades.

[14] Very High Speed Integrated Circuit Description Language.
[15] Lesser General Public License

Fig. 7. Secciones de búsqueda e intercambio[16].

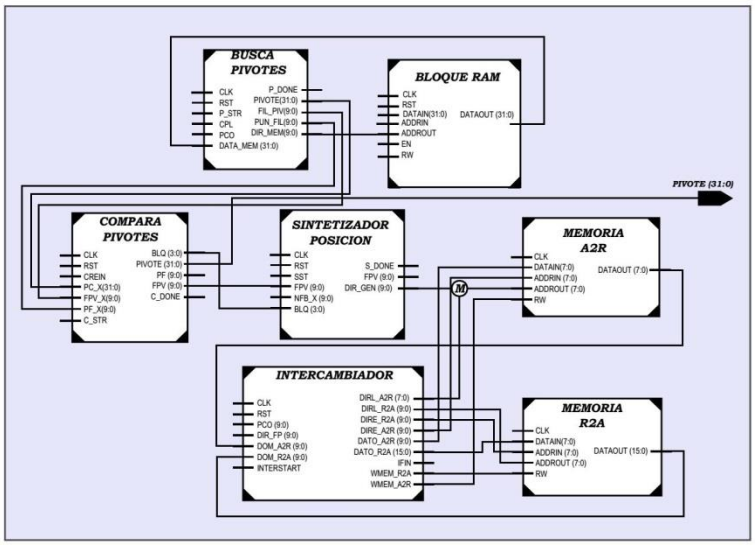


Fig. 8. Secciones de Normalización y eliminación.

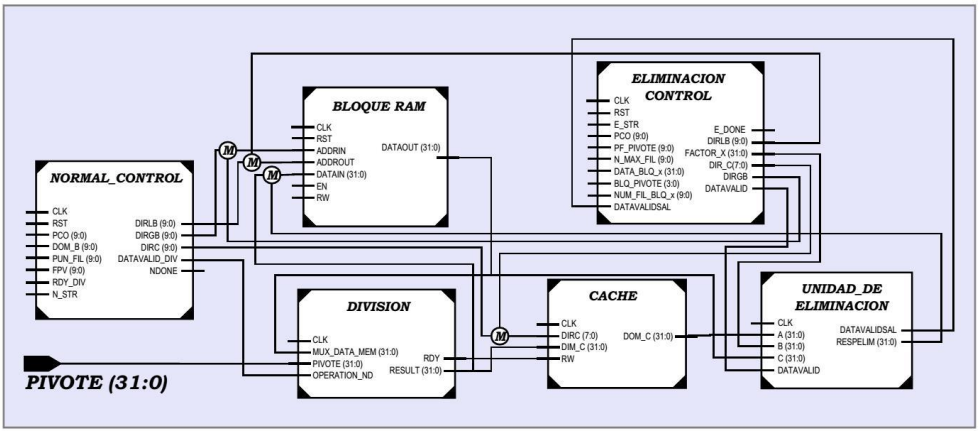


Tabla 2. Configuración de unidades de coma flotante.

Unidad	Número de bits	Ciclos por operación	Latencia[17]
Producto	32	1	6
División	32	1	28
Resta	32	1	13

[16] La M dentro del círculo simboliza multiplexores, que son controlados por el bloque GJ-control
[17] En ciclos de reloj

La latencia de una unidad segmentada es el número de ciclos de reloj que se tarda en completar una operación. El valor de la latencia usado en esta implementación es el máximo permitido por el software de configuración, con el fin de obtener la máxima frecuencia de operación del sistema.

Los ciclos por operación son los ciclos de reloj que debe haber entre el ingreso de un dato y el siguiente, es decir; que para el caso de GJ-FPGA se pueden ingresar un par de datos a cada ciclo de reloj. El valor de los ciclos por operación pudiera ser mayor, pero afectaría el desempeño, ya que se requieren más pulsos para procesar un par de datos.

2.5 RESULTADOS DE LA SÍNTESIS DE GJ-FPGA

La descripción de todos los módulos mencionados, algunos módulos auxiliares y las unidades de como flotante debidamente configuradas y encapsuladas en el diseño, dan como resultado de síntesis la información referida por ISE 10.1 que se muestra en la Tabla 33.

Tabla 3. Características de síntesis ISE 10.1

CARACTERÍSTICA	XC3S700AN-4	XC3S700AN-5
Frecuencia máxima	95.13 MHz	108.669 MHz
Slices	100 %	100 %
Slices Flip Flop	64 %	64 %
4 Inputs LUTs	73 %	73 %
Multiplicadores 18x18	100 %	100 %

Los números a continuación del guion en el nombre del dispositivo indican el grado de velocidad del mismo, que es función de la tecnología de fabricación del dispositivo.

3. ANÁLISIS DE DESEMPEÑO.

El diseño implementado sobre este dispositivo está condicionado por los recursos lógicos que este posee y en consecuencia el sistema completo GJ-FPGA que se logra sintetizar en el dispositivo XC3S700AN tiene cinco unidades de eliminación U. Invertir una matriz en este sistema requiere distribuir el número de filas de la matriz (N), en el número de unidades U para alojarlas en la memoria RAM que posee cada unidad. De esta manera, cada unidad U procesará las filas que se hallen en su bloque de memoria. La ecuación (1) muestra esta relación, donde *fbm* es el número entero máximo de filas por cada unidad U.

$$fbm = \begin{cases} \frac{N}{U} & si\ U\ divide\ exactamente\ a\ N \\ \left[\left[\frac{N}{U}\right]\right] + 1 & en\ caso\ contrario. \end{cases} \quad (1)$$

3.1 DESCRIPCIÓN ANALÍTICA DEL ALGORITMO.

Cada proceso tiene una duración en ciclos de reloj que es función *N* y *fbm*. En la Tabla 44 se encuentran las ecuaciones que permiten calcular la duración en ciclos de reloj totales para cada proceso durante la ejecución del algoritmo. La suma de estas duraciones parciales da como resultado el caso más desfavorable de GJ-FPGA.

3.2 DESEMPEÑO REAL Y TEÓRICO.

La ecuación (7) es la expresión que sirve para calcular el total de ciclos de reloj empleados por GJ-FPGA en el cálculo de matrices inversas y que se muestra en gráficamente en la Fig. 99. La diferencia que se aprecia, se debe a que el desempeño real depende de cada matriz en particular y la ecuación (7) calcula el peor caso; por lo tanto, el número de ciclos real será menor o igual al teórico.

Tabla 4. Resumen de las ecuaciones. Ciclos de reloj por proceso.

PROCESO	DURACIÓN (ciclos)	
Parámetros	6	
Búsqueda de pivotes	$4fbm + \frac{fbm(fbm+1)}{2} + (N - fbm) * (4 + fbm)$	(2)
Comparación de pivotes	$6N$	(3)
Intercambio de filas	$6N$	(4)
Normalización	$31N + 2N^2 - \frac{N(N+1)}{2}$	(5)
Eliminación	$\left[26N + 2N^2 - \frac{N^2}{2} - \frac{N}{2}\right] fbm$	(6)
Total Ciclos de reloj	$CR(N, fbm) = N^2[1.5 + 1.5fbm] + N[46.5 + 26.5fbm] - 0.5fbm^2 + 0.5fbm + 6$	(7)

El error relativo cometido al calcular el número de ciclos de reloj con la ecuación (7) se representa en la Fig. **10**10. Puede verse que a mayor tamaño de la matriz el error relativo disminuye llegando a ser menor al 1% para los datos mostrados.

Fig. 9. Desempeño real y teórico.

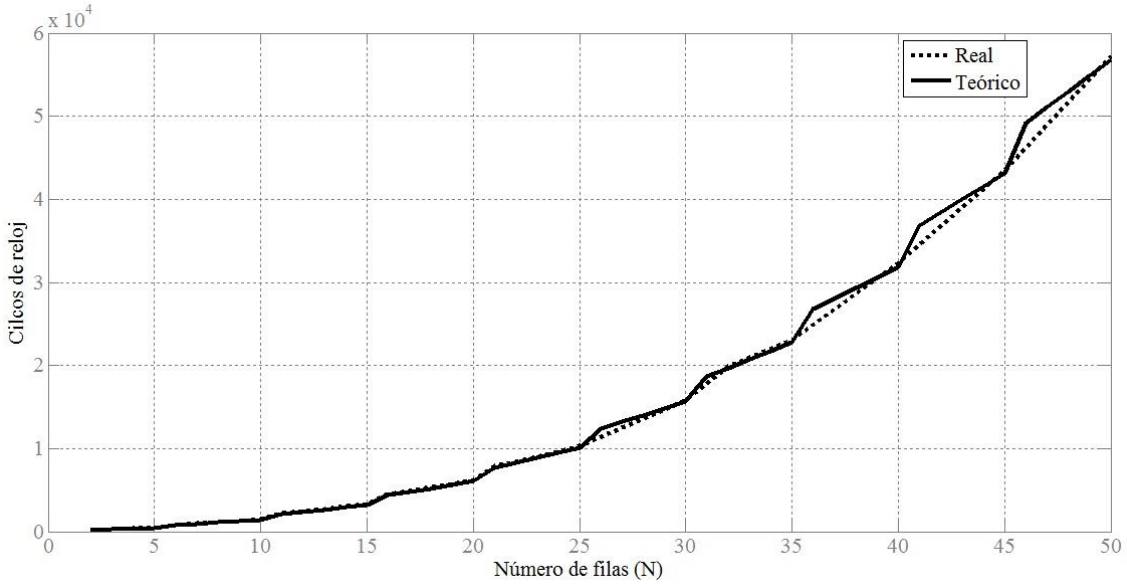
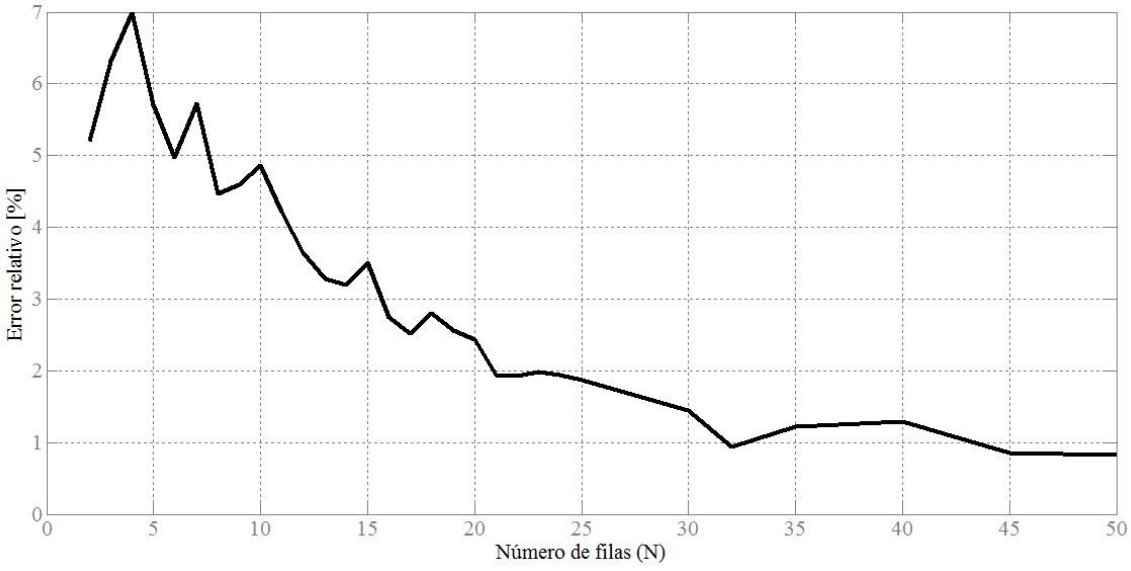


Fig. 10. Error relativo porcentual.



3.3 LA FRECUENCIA Y EL DESEMPEÑO.

El tiempo de ejecución del algoritmo T_e es función del número de ciclos y la frecuencia como lo muestra la ecuación 8.

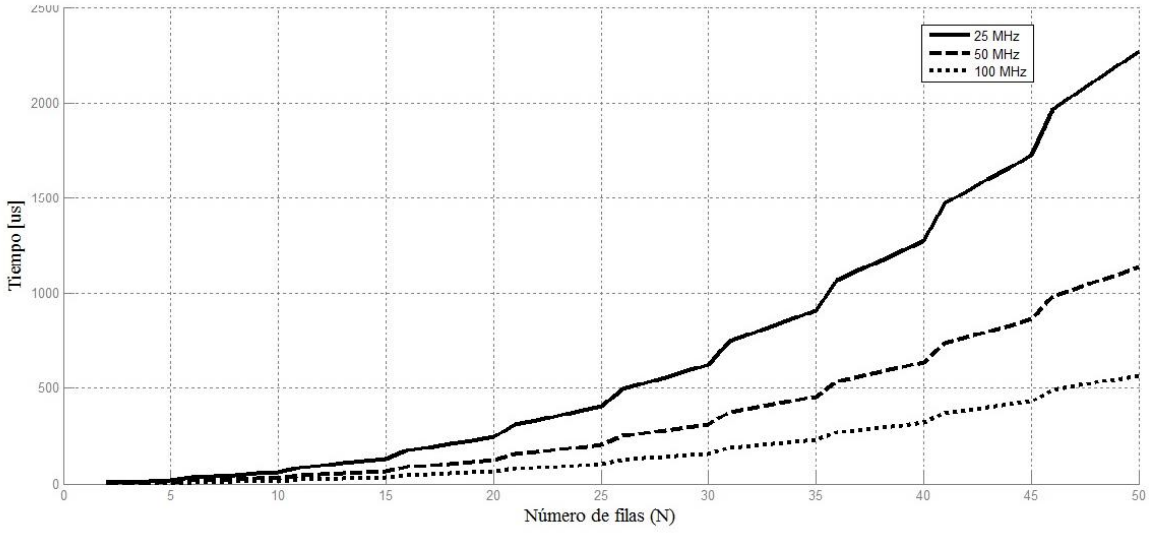
$$T_e = \frac{Ciclos\ de\ reloj}{Frecuencia} \tag{8}$$

En la Fig. 1111. Se ilustra el tiempo de ejecución para diferentes valores de frecuencia en base a un sistema con 5 unidades U y varios valores de N.

A partir de la ecuación (8) se puede encontrar que la ecuación (9) representa la reducción porcentual de tiempo de ejecución. Si duplicamos una frecuencia a partir de una frecuencia de referencia F, la reducción de tiempo es el 50%. Esto se ilustra en la Fig. 1111 para cinco unidades de eliminación U.

$$\frac{\Delta F}{\Delta F + F} * 100 = T_e red\% \tag{9}$$

Fig. 11. Reducción de tiempo de ejecución en función de la frecuencia.

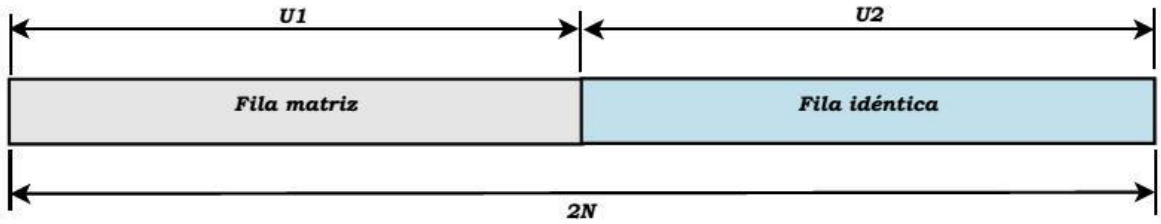


4. TRABAJOS FUTUROS.

4.1 EL CONCEPTO DE ELIMINACIÓN DUAL.

Es importante recalcar que el proceso más costoso en términos de cómputo es el de *eliminación*, por lo tanto, si se logra reducir este tiempo, se puede lograr un aumento adicional del desempeño del sistema. Hasta ahora se ha considerado el proceso de eliminación procesando varias filas simultáneamente, aun cuando las columnas se procesan una a una. Pero es posible usar el mismo concepto para procesar las columnas. Esto es factible debido a la independencia de los datos entre columnas. Se definirá entonces, la forma de eliminación donde una sola unidad U procesa toda la fila como *sencilla* y donde dos unidades U procesan una misma fila, como *dual*. En la Fig. 122 se ilustra este concepto.

Fig. 12. Eliminación dual



De esta manera una fila cualquiera se muestra siendo eliminada, en sus primeros N elementos, por U1 sus últimos N elementos por U2 simultáneamente. El objetivo de este arreglo, consiste en realizar la eliminación de la fila en la mitad del tiempo que en el caso de eliminación *sencilla*. Por lo tanto (6) se modifica como:

$$\left[26N + N^2 - \frac{N^2}{2} - \frac{N}{2}\right] fbm \tag{10}$$

Lo cual a su vez representa un cambio en (7) tal que:

$$N^2[1.5 + 0.5fbm] + N[46.5 + 26.5fbm] - 0.5fbm^2 + 0.5fbm + 6 \quad (11)$$

En la Fig. 133 se estima como cinco unidades *duales* podrían exhibir un mejor desempeño que 10 *sencillas*.

Fig. 13 Eliminación sencilla U=10 vs eliminación dual U=5.

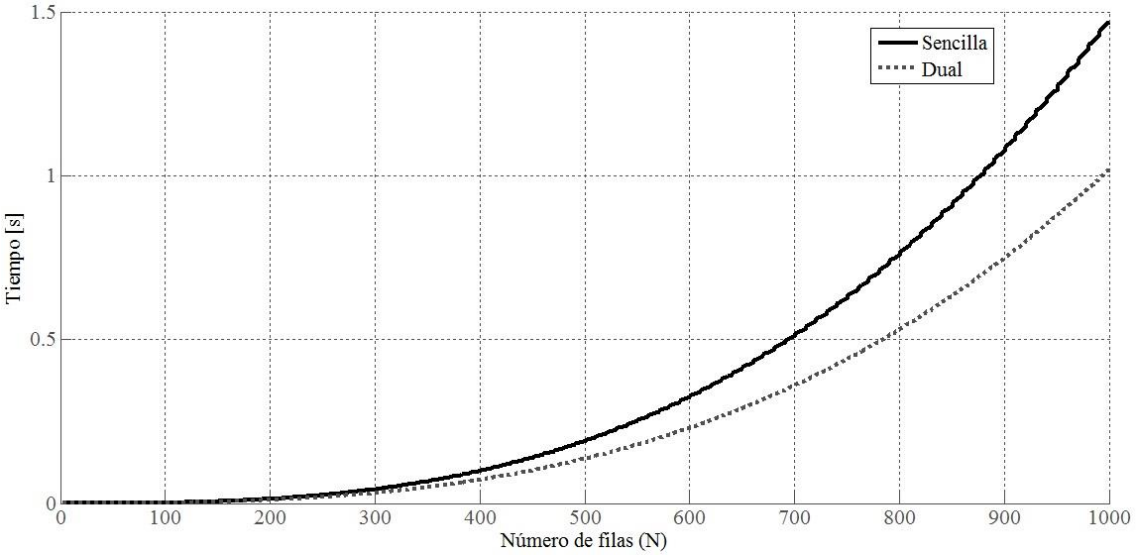
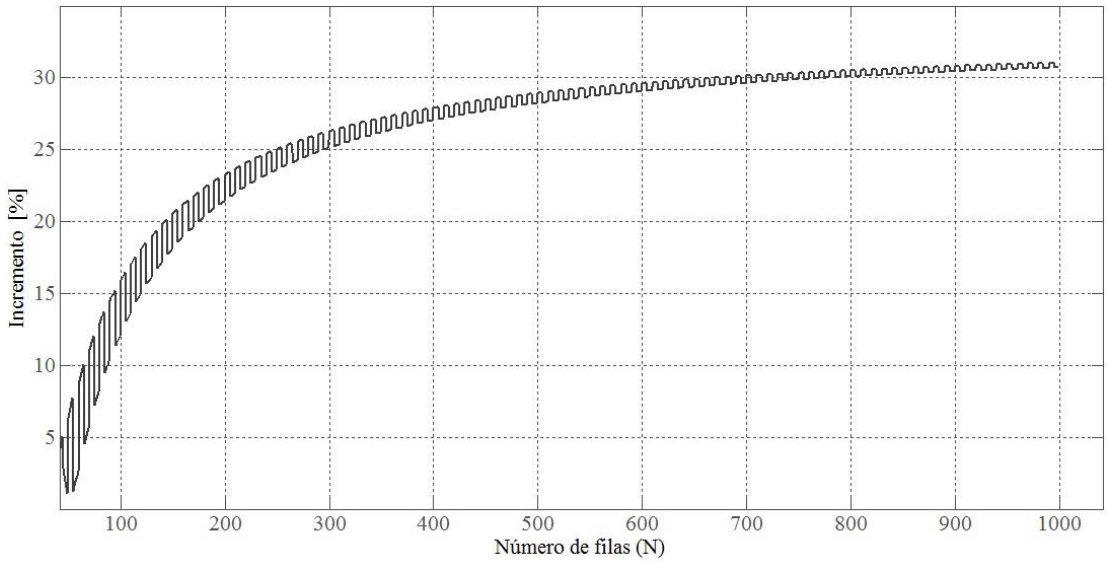


Fig. 14. Aumento del desempeño dual.



Sintetizar una unidad *dual*, es igual de costosa a sintetizar dos *sencillas* en cuanto a recursos lógicos se refiere. Por lo tanto el esquema de unidades duales puede verse como una redistribución de los recursos lógicos del dispositivo FPGA.

Si se representa gráficamente el incremento de desempeño del ejemplo de la Fig. 133, puede verse que tal incremento no es indefinido y se encuentra cerca del 33% para grandes valores de N, como se muestra en la Fig. 14.

Aunque no se muestra gráficamente, el incremento de desempeño para N=1e6 es de 33.33%, lo cual muestra el poco incremento que hay después del 33%, permaneciendo casi constante.

Sin embargo; este análisis no puede ser aprovechado para efectos del presente trabajo, dado que implica una reforma sustancial de la arquitectura del sistema, lo cual excede los alcances de este trabajo y se reserva para trabajos futuros.

Adicionalmente podría considerarse en replicar este análisis para hacer eliminación múltiple con más de *dos* unidades, pero se recomienda que el número de unidades sea par, con el objetivo de distribuir uniformemente los datos de la matriz original y los de la matriz inversa. Haciendo menos difícil el diseño del control de eliminación.

5. CONCLUSIONES.

- El número de ciclos de reloj CR para inversión de matrices en GJ-FPGA se puede calcular con la expresión (1) y (7).

$$fbm = \begin{cases} \frac{N}{U} & si\ U\ divide\ exactamente\ a\ N \\ \left[\left[\frac{N}{U}\right]\right] + 1 & en\ caso\ contrario. \end{cases} \quad (1)$$

$$CR(N, fbm) = N^2[1.5 + 1.5fbm] + N[46.5 + 26.5fbm] - 0.5fbm^2 + 0.5fbm + 6 \quad (7)$$

- La implementación de GJ-FPGA está hecha en coma flotante en precisión sencilla de 32 bits, pero el resultado puede ser ampliado a precisión doble (64 bits), sin mayores dificultades.

- Una modificación encaminada a mejorar el desempeño del sistema, debe interesarse por hacer más eficiente el proceso de eliminación, dado que es el que más tiempo consume.

- El sistema GJ-FPGA es apto para ser usado en algún sistema, que requiera inversión de matrices sin software. Como en el caso de las comunicaciones digitales del tipo MIMO-OFDM. En [11] y [12] se muestran implementaciones similares para este propósito.

- Una modificación del sistema GJ-FPGA bajo el concepto de eliminación doble, puede imprimir un aumento del desempeño cercano al 30%, en comparación con la eliminación sencilla.

REFERENCIAS BIBLIOGRÁFICAS

[1] V. Sonzogni, P. Sachez y M. Storti, RESOLUCION DE GRANDES SISTEMAS DE ECUACIONES EN UN CLUSTER DE COMPUTADORAS, Bariloche: G.Buscaglia, E.Dari, O.Zamonsky (Eds.), 2004.

[2] A. Irtuk, S. Mirzaei y R. Kastner, "An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition", March 9, 2009.

[3] J. Ketonen, "Equalization and Channel Estimation Algorithms and Implementations for Cellular MIMO-OFDM Downlink", Oulu, Finland: Centre for Wireless Communication at the University of Oulu, 2012.

[4] J. Arias, R. Pezzuol y M. Ayala, ”A SUITABLE FPGA IMPLEMENTATION OF FLOATING-POINT MATRIX INVERSION BASED ON GAUSS-JORDAN ELIMINATION”, Brasilia: Universidad de Brasilia : Departamentos de mecánica y Ciencia de la computación, 2011.

[5] R. Duarte, N. Horácio y M. Véstias, "Double-presicion Gaus-Jordan algorithm with partial pivoting on FPGAs", 12th Euromicro Conference on Digital System Design / Architectures, Methods and Tools, 2009.

[6] H. Neto y G. De Matos, ”On Reconfigurable Architectures for Efficient Matrix Inversion"., s.l. : FPL ’06. International Conference, 2006., 2006.

[7] R. A. Gelonch, "Procedimiento de diseño de circuitos digitales mediante FPGAs", Lleida: Universidad de Lleida, 2007.

[8] C. Fajardo y J. Ramón, "Descripción de una metodología para diseñar procesadores de propósito específico implementados sobre FPGAs", Bucaramanga: XV SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISIÓN ARTIFICIAL - STSIVA, 2010.

[9] R. Lange y P. Benett, «https://github.com/sd2k9/hdl_uart,» [En línea]. [Último acceso: 30 Septiembre 2014].

[10] X. inc, Xilinx., [En línea]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf. [Último acceso: 20 Septiembre 2014].

[11] M. Karkooti y J. R. Cavallaro, FPGA Implementation of Matrix Inversion Using, Miami: Center for Multimedia Communication, Department of Electrical and Computer Engineering.

[12] D. Wu, J. Eilert, D. Liu, D. Wang, N. Al-Dhahir y H. Minn, Fast Complex Valued Matrix Inversion for Multi-User STBC-MIMO Decoding, Sweden. Department of Electrical Engineering, USA, Department of Electrical Engineering.

BIBLIOGRAFÍA

A. Irtuk, S. Mirzaei y R. Kastner, "An Efficient FPGA Implementation of Scalable Matrix Inversion Core using QR Decomposition", March 9, 2009.

C. Fajardo y J. Ramón, "Descripción de una metodología para diseñar procesadores de propósito específico implementados sobre FPGAs", Bucaramanga: XV SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISIÓN ARTIFICIAL - STSIVA, 2010.

D. Wu, J. Eilert, D. Liu, D. Wang, N. Al-Dhahir y H. Minn, Fast Complex Valued Matrix Inversion for Multi-User STBC-MIMO Decoding, Sweden. Department of Electrical Engineering, USA, Department of Electrical Engineering.

H. Neto y G. De Matos, ”On Reconfigurable Architectures for Efficient Matrix Inversion"., s.l. : FPL ’06. International Conference, 2006., 2006.

J. Arias, R. Pezzuol y M. Ayala, ”A SUITABLE FPGA IMPLEMENTATION OF FLOATING-POINT MATRIX INVERSION BASED ON GAUSS-JORDAN ELIMINATION”, Brasilia: Universidad de Brasilia : Departamentos de mecánica y Ciencia de la computación, 2011.

J. Ketonen, "Equalization and Channel Estimation Algorithms and Implementations for Cellular MIMO-OFDM Downlink", Oulu, Finland: Centre for Wireless Communication at the University of Oulu, 2012.

M. Karkooti y J. R. Cavallaro, FPGA Implementation of Matrix Inversion Using, Miami: Center for Multimedia Communication, Department of Electrical and Computer Engineering.

R. A. Gelonch, "Procedimiento de diseño de circuitos digitales mediante FPGAs", Lleida: Universidad de Lleida, 2007.

R. Duarte, N. Horácio y M. Véstias, "Double-presicion Gaus-Jordan algorithm with partial pivoting on FPGAs", 12th Euromicro Conference on Digital System Design / Architectures, Methods and Tools, 2009.

R. Lange y P. Benett, «https://github.com/sd2k9/hdl_uart,» [En línea]. [Último acceso: 30 Septiembre 2014].

V. Sonzogni, P. Sachez y M. Storti, RESOLUCION DE GRANDES SISTEMAS DE ECUACIONES EN UN CLUSTER DE COMPUTADORAS, Bariloche: G.Buscaglia, E.Dari, O.Zamonsky (Eds.), 2004.

X. inc, Xilinx., [En línea]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf. [Último acceso: 20 Septiembre 2014].