PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DEL MÉTODO GRADIENTE CONJUGADO EN UN FPGA ARQUITECTURA SPARTAN 6

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Stefano André Sosa Cordova

ASESOR: Paul Antonio Rodriguez Valderrama

Lima, Junio del 2014

A Patricia, mi compañera, amiga y amante. Por sacrificar tu tiempo para que te pudiera alcanzar y así avanzar juntos en nuestro proyecto de vida, por tu bondad y alegría que cada día me hace ser mejor para ti y para nuestro hijo.

A mi amado hijo André, mi principito, concebido del amor más sublime y sincero que podría existir. Tus sonrisas y tus abrazos me brindaban toda la paz y tranquilidad que necesitaba para pensar con claridad, cuando todo era tan turbulento y complicado.

A mis amados padres, Jaime y Giuliana, que hicieron todo para que yo pudiera culminar esta etapa de mi vida. Por darme su fuerza y respaldo cuando sentía que ya no podía más, y las lecciones de la vida, que valen mucho más que las que uno aprende en los libros.

A Daniel, Jaime y Renán, miembros del grupo de Procesamiento Digital de Señales, por el conocimiento, los buenos momentos y por los espacios de reflexión que me brindaron.

A Edward, Sammy y Guillermo, miembros del grupo de Microelectrónica, por brindarme la oportunidad de conocer aquella rama del conocimiento que más me apasiona y permitirme ser parte de un grupo cuya calidad humana es insuperable.

A Kristty, Silvera y Nathaly, mis colegas industriales, quienes me acercaron a Dios, y siempre me acompañaron en mi alegría y mi tristeza como unas hermanas.

A Misely, Mayra y Mario, a quienes considero un modelo a seguir por su fuerza, carisma, inteligencia y gran calidad profesional. Por todo el apoyo emocional que me brindaron a lo largo de la carrera, por haberme obsequiado su sincera amistad, por haber estado a mi lado cuando no parecía haber solución y siempre me mostraban la salida.

Agradezco infinitamente a cada uno de los mencionados por su colaboración en el desarrollo de esta tesis, pues entre discusiones académicas, conversaciones amicales y momentos de ocio surgieron las ideas que me permitieron culminar con éxito este trabajo.

Resumen

Resolver un sistema de ecuaciones lineales simultáneas es un problema fundamental en el algebra lineal numérica, y una de las etapas elementales en simulaciones científicas. Ejemplos son los problemas de ciencias e ingeniería modelados por ecuaciones diferenciales ordinarias o parciales, cuya solución numérica está basada en métodos de discretización que conducen a sistemas de ecuaciones lineales. Estos sistemas pueden ser resueltos de manera directa; sin embargo, cuando el orden del sistema es demasiado grande el costo computacional se incrementa. Ante esta situación se emplean métodos iterativos, los cuales son más eficientes y tienen una menor demanda computacional (p.e: Jacobi, Gauss-Seidel, Gradiente Conjugado, etc.).

En el presente trabajo se presenta un sistema digital basado en un procesador, un coprocesador y una memoria externa que desarrolla el método del Gradiente Conjugado. El sistema fue implementado en la arquitectura Spartan-6, la cual cuenta con un *softprocessor* de 32 bits llamado MicroBlaze y el FPGA propiamente dicho. MicroBlaze dirige el flujo del algoritmo, además de desempeñar las operaciones más sencillas (sumas vectoriales, productos internos, divisiones, etc). En tanto, en el FPGA se implementó un coprocesador, el cual fue descrito en VHDL, que se encarga de la operación de mayor costo computacional: el producto Matriz - Vector. El procesador y el coprocesador se comunican mediante interfaces unidireccionales basadas en unidades *FIFO* llamadas *Fast Simplex Link* (FSL). Se empleó el entorno EDK (*Embedded Development Kit*) de la empresa Xilinx, para configurar el procesador, los periféricos y el coprocesador; y se empleó la plataforma Atlys de la empresa Digilent para implementar el sistema propuesto. La implementación final es aproximadamente 2 veces más rápida y tiene una eficiencia de 0.25, respecto de la implementación de referencia que se desarrolló empleando solo el procesador.

El orden que sigue la tesis es el siguiente: En el primer capítulo se presenta el contexto de la tesis y se define puntualmente el problema que se desea resolver. En el segundo capítulo se cubre la mayoría de aspectos teóricos necesarios. La arquitectura propuesta, y los detalles de los componentes del sistema se especifican en el capítulo tres. Por último, se presentan los resultados en el capítulo cuatro, seguido de las conclusiones.

Índice general

Índice de figuras

Introducción

Existen varias aplicaciones de ciencias e ingeniería que se modelan como sistemas de ecuaciones lineales (p.e.: el análisis estructural [1], el entrenamiento de sistemas difusos [2, 3], el entrenamiento de *Support Vector Machine* (SVM) [4], el análisis de materiales [5], etc.) Para resolver estos sistemas existen dos clases de métodos: los métodos directos y los métodos iterativos.

Los métodos directos (p.e.: métodos de eliminación y métodos de factorización) calculan la solución exacta en un determinado número de pasos que depende del tamaño del sistema; sin embargo, cuando en la aplicación en cuestión involucra matrices de tamaños muy grandes el costo computacional de las operaciones involucradas aumenta en forma cúbica [6]. Otro inconveniente de estos métodos, es que si la matriz presenta muchos elementos nulos (sparse) se debe recurrir a técnicas de pivoteo parcial o completo, lo cual incrementa aún más el número de operaciones. Por otro lado, los métodos iterativos (p.e.: Jacobi, Gauss-Seidel, Chevyshev, etc.) determinan una aproximación de la solución exacta [7]. Estos fijan una solución inicial y a partir de ella calculan una secuencia de vectores, la cual converge a la solución exacta. La iteración termina cuando la aproximación tiene una precisión que cumple con una tolerancia determinada.

Uno de los métodos iterativos más conocidos es el método de Gradiente Conjugado (CG acrónimo en inglés), el cual fue propuesto por primera vez por Hestenes y Stiefel en 1950 como un algoritmo iterativo para resolver sistemas lineales con matrices definidas positivas [8]. Entre otras aplicaciones CG es la operación base utilizada en métodos de eliminación de ruido y deconvolución (p.e.: *Total Variation* [9]) así como métodos de descomposición adaptativa de datos con diccionarios sobre-completos (p.e.: *Basis Pursuit* [10]). Por lo general, los métodos iterativos son más rápidos que los métodos directos y su implementación en paralelo es más sencilla [11].

En el presente trabajo, se presenta un sistema digital que desarrolla el método de Gradien-

te Conjugado, el cual está basado en una de las nuevas metodologías de diseño, el codiseño software-harware. Se utilizará la arquitectura Spartan6 [12], la cual está conformada por un microprocesador y un FPGA. En princio, el algoritmo será implementado empleando solo el microprocesador y servirá como referencia. Por otro lado, en el FPGA se implementará el co-procesador que realizará el producto Matriz-Vector, el cual tendrá una jerarquía de 3 etapas: entradas, cómputo y salidas; y estará compuesto principalmente de núcleos de propiedad intelectual que realizan operaciones aritméticas en coma flotante de 32 bits.

Capítulo 1

Problemática

1.1. Descripción y formulación del problema

Los sistemas de ecuaciones lineales están presenten en varias aplicaciones de ingeniería. Típicamente, estos sistemas son simétricos y definidos positivos. La razón matemática es debido a la forma $B = A^T A$. Por otro lado, la razón física es que la expresión $\frac{1}{2}u^T Bu$ por lo general está asociada a procesos que provienen del análisis de interacción de fuerzas, o análisis energético y dado que estas magnitudes nunca son negativas se dice que son sistemas definidos positivos [1].

Es por esto que un aspecto muy importante en las aplicaciones de ingeniería son los métodos de resolución de sistemas lineales. Existen dos clases de métodos para resolver un sistema lineal: directos e iterativos. Los métodos directos se caracterizan por calcular la solución exacta en una determinada cantidad de pasos que, por lo general, depende del tamaño del problema; sin embargo, estos no tienen un buen rendimiento con sistemas de gran tamaño. Por su parte, los métodos iterativos (p.e.: Succesive Over Relaxation (SOR), Steepest Descent, Gradiente Conjugado, etc.) brindan una serie de vectores, la cual converge a una aproximación de la solución deseada [13]. Esto lo consiguen a un menor costo computacional, pero con menor precisión que los métodos directos.

Una de las dificultades asociadas a métodos iterativos como SOR, Chebyshev y otros métodos relacionados es que dependen de parámetros que la mayoría de veces son difíciles de fijar de manera manual [7]. Debido a esto, el método del Gradiente Conjugado resulta ser una alternativa más eficiente debido a que sus parámetros son fijados de manera interna, su rango de aplicación es más general y su costo computacional es menor comparado al de la mayoría de métodos iterativos.

1.2. Importancia y justificación del asunto de estudio

Esta tesis se enfoca en mejorar la eficiencia computacional del método Gradiente Conjugado, mediante el uso de un coprocesador. El problema radica en identificar de forma precisa los cuellos de botella de la aplicación que se desea optimizar para luego diseñar un nuevo algoritmo, típicamente en paralelo, que permita disminuir el tiempo de ejecución del mismo. Previamente, o durante el diseño del algoritmo, se debe seleccionar el paradigma que se empleará como solución.

Existen varios paradigmas de diseño de algoritmos en paralelo que se especializan en alguna plataforma específica para mejorar los resultados de la mejor manera posible. Debido a esto, se debe conocer bien las características de la aplicación y los alcances de implementación requeridos para poder seleccionar la plataforma adecuada y diseñar un algoritmo paralelo.

En la actualidad, si bien existen plataformas de software que permiten mejorar la performance de un algoritmo (p.e SIMD, CUDA, MPI, etc. [14]), el diseño se está cada vez enfocando más a tener plataformas dedicadas y de bajo consumo de potencia, en lugar de ordenadores de propósito general, sobretodo cuando la aplicación será de campo. Una de las plataformas que proporcionan la sencilles del trabajo en software, junto con la flexibilidad y la capacidad de procesar datos en paralelo son los FPGAs, los cuáles han sido las plataformas de desarrollo más comunes para acelerar una variedad de aplicaciones. Por ejemplo, el robot Curiosity emplea FPGAs para su control de motores [15]; el satélite FedSat emplea un arreglo de FPGAs como parte de su unidad de cómputo de alto rendimiento [16]; el Radio Observatorio de Jicamarca posee una unidad de procesamiento paralelo para un radar de 16 canales basado en un FPGA [17].

Otro punto importante de esta tesis, además de buscar optimizar el algoritmo del gradiente conjugado, es el diseño del coprocesador que desarrollará la operación Matriz-Vector, dado que existe una gran variedad de algoritmos de DSP basados en esa operación. La idea de tener un sistema mixto donde las operaciones menos costosas son sencillas de programar y la operación costosas ya está implementada en hardware dedicado podría permitir extrapolar este trabajo a otras aplicaciones y no solo CG.

1.3. Justificación de la metodología

Diseñar arquitecturas de sistemas embebidos es complicado, y no hay método general, ni reglas para hacerlo. En los últimos años, este problema se ha incrementado debido a la gran

variedad de dispositivos programables que han surgido (p.e FPGA, CPLD, ASIP, DSP, etc.) [18].

Por un lado, los ASIP (*Application Specific Instruction Set Processor*) y los DSP (*Digital Signal Processor*) son plataformas en las que el diseño se da casi enteramente a nivel software. El diseño a nivel software ofrece la facilidad del entorno de desarrollo y la flexibilidad de un lenguaje de programación sencillo de adaptarse a los cambios; sin embargo, esto no garantiza una explotación al máximo de los recursos y las capacidades de la plataforma. Por otro lado, los CPLDs y FPGAs son plataformas de diseño de hardware. Diseñar un *custom hardware* brinda una serie de beneficios; primero, el rendimiento es típicamente más rápido debido a la demanda de menos ciclos de reloj, o debido ciclos de reloj más cortos como resultado de unidades funcionales más sencillas [19]. Por otro lado, se garantiza que se optimice el uso de los recursos disponibles; sin embargo, debido a los mayores requerimientos y conocimientos tienden a demandar más experiencia de diseño que la metodología de software [20]. Como una alternativa ante estos dos aportes está el codiseño Software-Hardware la cual ha sido propuesta como el punto de partida para diseñar sistemas embebidos. El codiseño ayuda al diseñador a pensar en términos de intercambio entre flexibilidad y rendimiento.

La idea central de esta metodología es fundir dos procesos de diseño. Por un lado, el diseño de hardware emplea la descomposición espacial y está adecuada para mejorar el rendimiento empleando paralelismo. Por su parte, el diseño de software emplea la descomposición temporal y está adecuada para desarrollar aplicaciones flexibles. Debido a estas razones, el codiseño software/hardware simplifica el proceso de desarrollo e implementación de arquitecturas dedicadas a algoritmos específicos [21].

1.4. Objetivos

El objetivo principal del presente trabajo es implementar el método de Gradiente Conjugado en la arquitectura Spartan6, empleando un procesador que se encargue de las operaciones más sencillas del algoritmo y un coprocesador que efectúe el producto Matriz-Vector en paralelo de manera que se consiga mejorar la performance del algoritmo. Con este propósito, se desarrollarán objetivos específicos. Primero, se implementarán las dos propuestas de CG en [8] en el lenguaje de alto nivel Matlab con la finalidad de aprender las propiedades básicas del algoritmo y analizar los resultados. Una vez conseguido esto, se implementará la versión que presente mejores resultados en el lenguaje C estándar para próximamente implementar el

código en el procesador MicroBlaze. Concluido el proceso de implementación del algoritmo, se procederá al análisis de la operación Matriz-Vector para diseñar el algoritmo paralelo que será implementado en el FPGA. Para la verificación del funcionamiento del coprocesador se realizarán simulaciones y depuraciones en tiempo real. Finalmente, ambos subsistemas serán conectados empleando las interfaces FSL.

1.5. Limitaciones

La plataforma de desarrollo Atlys solo cuenta con una memoria externa de 128MBytes, debido a esto solo se probarán tamaños de problema que no excedan su capacidad. Por otro lado, el sistema que se propone es síncrono y debe funcionar a la misma frecuencia, en este caso el elemento limitante es el procesador MicroBlaze que opera a 100MHz. Del mismo modo, el FPGA solo presenta recursos suficientes para implementar unidades que operen 16 datos en simultáneo, por esta razón su operación se realizará en bloques de 8, y al estar condicionado de esta forma los tamaños de problema de los sistemas deberán ser múltiplos de 8. Otra limitación es el rango de valores que pueden tomar los elementos del sistema, pues como se trabajará en precisión simple se deberá procurar que los valores del sistema estén escalados a un rango que evite que se produzcan los comodines *Inf* o *Nan*. Finalmente, dado que no existen trabajos que sigan la metodología empleada en este trabajo, no solo se restringe la información de la que se puede partir, sino que no se podrán realizar comparaciones con trabajos similares; sin embargo, sí existen implementaciones discretas de CG en FPGAs las cuáles serán una referencia para diseñar el coprocesador.

Capítulo 2

El Método del Gradiente Conjugado

2.1. Antecedentes

Los métodos iterativos para resolver sistemas lineales son propuestos como una alternativa a los métodos directos ante la gran demanda computacional de estos para sistemas de gran tamaño y los casos de las matrices que contienen gran cantidad de elementos nulos. Estos métodos calculan la solución del sistema en forma de límite de una secuencia de ciertos vectores, los cuáles son construidos usando un proceso uniforme [6]. Uno de estos métodos, es el Gradiente Conjugado el cual se puede derivar a partir de otros métodos (p.e Steepest Descent, Lanczos, General Search Directions) como se muestra en [13], [22] y [7]. El CG rápidamente se vuelve más popular que sus predecesores debido al menor costo computacional, y dado que varias aplicaciones de ingeniería involucran la resolución de sistemas lineales, el CG es tomado como operación base para varios procesos.

Adicionalmente, con el surgir de nuevas tecnologías y paradigmas de programación se busca la posibilidad de mejorar el rendimiento del algoritmo, mediante la ejecución en paralelo de sus operaciones más costosas. En este trabajo se pondrá un particular enfásis al empleo de FPGAs, dada su flexibilidad, su capacidad de procesar datos en paralelo y bajo consumo de potencia.

Implementaciones del Método del Gradiente Conjugado basadas en FPGAs se pueden encontrar en [23], [24] y [25]; sin embargo, todos estos trabajos son sistemas digitales discretos, no sistemas mixtos como el que se propone en este trabajo, a pesar de eso las arquitecturas presentadas fueron un referente para el diseño de la arquitectura del coprocesador. Durante la investigación no se encontraron trabajos que propusieran alguna implementación del método del Gradiente Conjugado empleando el codiseño Software-Hardware.

2.2. Definición

El método del gradiente conjugado es un método iterativo que sirve para resolver sistemas lineales

$$Ax = b \tag{2.1}$$

donde A es una matriz simétrica definida positiva de $n \times n$. El CG es el más antiguo y conocido de los métodos no estacionarios. El método procede al generar secuencias de vectores y residuos, y direcciones de búsqueda las cuales emplea para actualizar a los vectores y residuos durante cada iteración. Apesar de que dichas series pueden ser grandes, solo un pequeño grupo de estos vectores son necesarios en memoria. En cada iteración del método, se realizan actualizaciones de escalares que han sido definidos para cumplir ciertas condiciones de ortogonalidad. En un sistema simétrico definido positivo estas condiciones implican que la distancia a la solución esperada se minimiza en alguna norma [26].

2.3. Características

La característica principal del CG es que es iterativo, lo cual significa que su solución es una aproximación que proviene de una secuencia de vectores que converge a la solución final. Por otro lado, el CG está considerado dentro de los métodos iterativos estacionarios lo cual implica que los cálculos involucran información que cambia en cada iteración. Por lo general, las contantes involucradas se calculan tomando productos internos de residuos u otros vectores provenientes del método [26]. Otra propiedad resaltante del CG es su propiedad para generar un conjunto de vectores conjugados, sin necesidad de saber todos los elementos de la secuencia, lo cual es bastante útil en términos de memoria y cálculo [8]. Finalmente, otras de sus propiedades más útiles y que lo distingue de la mayoría de métodos es que la matriz de coeficientes no se ve alterada durante todo el desarrollo del algoritmo y que converge en un máximo de n iteraciones donde n es el tamaño del problema.

2.4. Algoritmos

2.4.1. Forma básica del Método Gradiente Conjugado

La primera propuesta para el método de Gradiente Conjugado brindada en 1952 [8], indica que la solución del sistema se debe aproximar acercándose por la dirección conjugada del

negativo del gradiente. Entonces, la solución del sistema implica calcular tres series de vectores: la solución en la iteración k, la dirección conjugada k, y el residuo en la iteración k. En el CG, por lo general, la solución inicial se fija en el vector cero; asimismo, cada dirección conjugada es elegida como una combinación lineal del residuo y la dirección anterior de manera que sea un paso descendente de la misma; mientras que el residuo es el resultado de multiplicar la matriz A por la solución aproximada restada del vector b. Esta versión es útil para estudiar las propiedades esenciales del gradiente conjugado. La primera propuesta del método de Gradiente Conjugado se muestra a continuación:

algorithm 1 CG Versión Preliminar

Data: A, b

Result: x

$x_0 \leftarrow 0$

$r_0 \leftarrow Ax_0 - b$

$p_0 \leftarrow -r_0$

$k \leftarrow 0$

while $\frac{\|r\|^2}{\|b\|^2} > \epsilon$ **do**

$\alpha \leftarrow -\frac{r_k^T p_k}{p_k^T A p_k}$

$x_{k+1} \leftarrow x_k + \alpha_k p_k$

$r_{k+1} \leftarrow Ax_{k+1} - b$

$\beta_{k+1} \leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k}$

$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$

$k \leftarrow k + 1$

end

2.4.2. Forma práctica del Método Gradiente Conjugado

Aplicando teoría de optimización se pueden conseguir expresiones más sencillas para las constantes α y β del algoritmo [8]. Luego de ejecutar estos cambios, se obtiene la forma estándar del método del Gradiente Conjugado:

Las implementaciones de este algoritmo, sobrescriben los valores de *x*, *r* y *p* en la memoria. La tarea computacional de mayor costo realizada en cada paso es el cálculo del producto Matriz-Vector Ap_k, el cálculo de los productos internos $p_k^T A p_k$ y $r_{k+1}^T r_{k+1}$ y las tres sumas de vectores. Es recomendable solo emplear el método CG para grandes problemas, pues para

algorithm 2 CG Versión Final

Data: A, b
Result: x
$x_0 \leftarrow 0$
$r_0 \leftarrow Ax_0 - b$
$p_0 \leftarrow -r_0$
$k \leftarrow 0$
while $\frac{\|r\|^2}{\|b\|^2} > \epsilon$ **do**
  $\alpha \leftarrow -\frac{r_k^T p_k}{p_k^T A p_k}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} \leftarrow r_k + \alpha_k A p_k$
  $\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$
  $k \leftarrow k + 1$
end

casos de problemas pequeños es preferible emplear eliminación Gaussiana u otros algoritmos de factorización ya que son menos sensibles a errores de rendondeo.

2.5. Aplicaciones

Existe una gran variedad de aplicaciones de ingeniería en la que se puede emplear CG como operación base. Por ejemplo, en [27] se propone un algoritmo para el entrenamiento de una red neuronal multicapa, el cual está basado en el CG. La implementación final actualizaba los pesos de entrada de cada neurona de manera rápida y eficiente; por otro lado, el rendimiento del algoritmo resultó superior al de los convencionales. Otro ejemplo se puede encontrar en [4], en el que CG es empleo como operación base para el entrenamiento de una SVM con fines de encontrar un clasificador de patrones. Asimismo, CG también es empleado en otras especialidades de ingeniería como ingeniería química e ingeniería de materiales [5].

Capítulo 3

Diseño del Sistema Propuesto

3.1. Metodología de Diseño

En el presente trabajo se empleará la metodología de co-diseño Software-Hardware, pues permite integrar ambos aportes y mejorar el rendimiento empleando tecnologías específicas. Por ejemplo, las soluciones en hardware presentan un mejor rendimiento, pues permiten la ejecución de operaciones en paralelo y aceleración en términos de frecuencia de operación; sin embargo, la mayoría de desarrollos en hardware son lentos y los recursos de los dispositivos que lo proporcionan están limitados. Por otro lado, las soluciones en software, son pensadas para procesadores de alto rendimiento, los cuales están disponibles a bajo costo debido a los elevados volúmenes de producción; a pesar de esto, la ejecución serial de operaciones puede disminuir el rendimiento de la aplicación deseada [28].

Para desarrollar la metodología se presentan dos aproximaciones una *orientada a software* y otra *orientada a hardware*. La primera, propone implementar una solución total en software, para posteriormente determinar cuáles son las operaciones más costosas, y finalmente proponer una implementación en hardware de éstas, junto con las respectivas interfaces de comunicación entre los subsistemas. La aproximación *orientada a hardware*, por su lado, propone desarrollar un sistema discreto de la aplicación para luego desarrollar una análisis de síntesis, recursos y tiempo de ejecución para finalmente hacer la transición a software de aquello que sea conveniente [18]. En este trabajo se empleó la primera aproximación, dado que los los algoritmos presentados en el capítulo 2, son sencillos de implementar y análizar en lenguajes de alto nivel como Matlab.

3.2. Diagrama de bloques del Sistema

El diagrama de bloques general del sistema propuesto se presenta en la figura 3.1. Primero, el procesador se encargará de generar el sistema lineal como una operación previa al algoritmo y que no será considerada como parte del método. Luego, el procesador ejecutará la función principal que consiste en el CG propiamente dicho, durante la ejecución del algoritmo, en cada momento que se desarrolle el producto Matriz-Vector, el procesador enviará los datos de manera serial al coprocesador, el cual ejecutará la operación en bloques de 8 y enviará el resultado de manera serial al procesador para que continúe con la ejecución del resto del algoritmo.

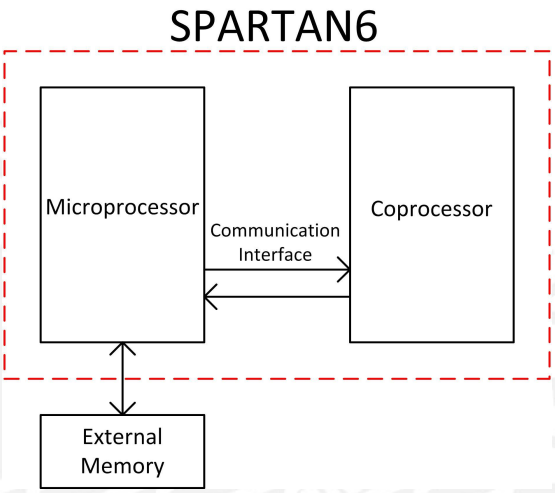


Figura 3.1: Elementos principales del Sistema

3.3. Interfaz de Coprocesamiento

Dado que la aplicación requiere de un alto *throughput* entre el software y el hardware, tiene sentido que en vez de emplear un bus del chip, se emplee una interfaz dedicada. Como se ilustra en la figura 3.2, la interfaz emplea puertos dedicados del procesador, los cuales son controlados por un conjunto restringido de instrucciones: las instrucciones del coprocesador. Este conjunto de instrucciones es diferente para cada tipo de procesador, pues dependen de como sea la estructura de la interfaz y de las propiedas del hardware. Emplear una interfaz de co-procesamiento puede ofrecer varias ventajas, como las más usuales están el alto *throughput* y una latencia fija. En el caso del *throughput* resalta que estas interfaces no están restringidas a la longitud de palabra del procesador. Por ejemplo, un procesador de 32 bits, podría soportar interfaces de 64 bits, o 128 bits y así mandar más de un dato por cada instrucción. Por otro

lado, tener una latencia fija permite que el *timing* de ejecución del software y el hardware sea conocido y predecible. Esta característica permite simplificar el diseño e implementación de los mecanismos de sincronismo [18]. Por contraparte, emplear un bus *On-Chip* implicaría que el mecanismo de comunición sería compartido por varios periféricos, lo cual hubiera causado que el *timing* fuera más complejo e impredecible, y así el diseño de las unidades de control del coprocesador se hubiera complicado.

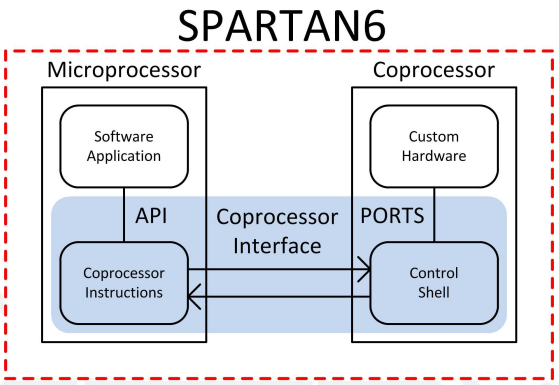


Figura 3.2: Interfaz de Co-procesamiento

3.4. Coprocesador

3.4.1. Algoritmo Paralelo de la operación Matriz Vector

El coprocesador se encarga de ejecutar el producto Matriz-Vector. La operación es realizada con una precisión de 32 bits en coma flotante, la cual fue seleccionada debido a que el tope de longitud de palabra de la interfaz era 32. Por otro lado, el coprocesador realiza la operación en bloques de 8, magnitud seleccionada experimentalmente, pues se intentaron diversos tamaños en potencias de 2, desde el 4 hasta el 16. Se observó en los reportes de síntesis de las herramientas que no habían suficientes LUTS para implementar un bloque de 16, que hubiera sido lo ideal. Los datos son enviados del procesador de manera serial y el coprocesador retorna su resultado al procesador de la misma forma.

A continuación se muestra el pseudocódigo de operación del coprocesador:

En la figura 3.3 se puede apreciar de forma gráfica el modo de operación del coprocesador. En (a) se han envíado el bloque de 8 elementos de A y de x para calcular el resultado parcial del primer elemento del vector b, luego en (b) se enviaron los 2 bloques faltantes y se calcula el resultado final del primer elemento de b. En (c) y (d) se repite el procedimiento para calcular

algorithm 3 Producto Matriz-Vector

Data: N, A, x
Result: b
$Store(N)$
$k \leftarrow 0$
while $k \leq \frac{N}{8}$ **do**
    $Store(x_{block})$
    $Store(A_{block})$
    $p \leftarrow Multiply(x_{block}, A_{block})$
    $s \leftarrow BinaryReduction(p)$
    $b \leftarrow Accumulate(s)$
    $k \leftarrow k + 1$
end
$Write(b)$

el siguiente elemento del vector b, y así hasta terminar toda la operación Matriz-Vector.

3.4.2. Estructura del Coprocesador

Dado que el procesador enviaría constantemente los datos al procesador, era necesario que este tenga una estructura que permita un traslape de comunicación y cómputo. Con este fin, se empleó una estructura jerárquica como la que se muestra en la figura 3.4, la cual permite un control independiente al almacenamiento de las entradas, los cálculos y el almacenamiento de las salidas. El intérprete de comandos analiza las señales que llegan de la interfaz de coprocesamiento y los traduce a señales de control para las máquinas de estado de niveles más bajos, que en términos simples son solo bits de inicio/fin para *handshakes* entre las etapas del coprocesador. A este esquema de diseño se le denomina *block-level pipelining* [18].

3.4.3. Registros de Entradas

Este módulo recibe como entrada los datos seriales que provienen de la interfaz de coprocesamiento y permite que se puedan operar en paralelo. El módulo opera a la misma frecuencia que el procesador y que la interfaz de coprocesamiento. La unidad está compuesta de Flip-Flops tipo D de 32 bits con habilitador. Adicionalmente, cuenta con un contador que se habilita cada vez que recibe un dato. Así, cuando el registro ya ha recibido todos los datos que soporta se activa una señal *full* que le indica a su unidad de control que ya puede empezar a operar datos. En la figura 3.5 se pueden apreciar en (a) y (b) la entidad y arquitectura del módulo respectivamente.

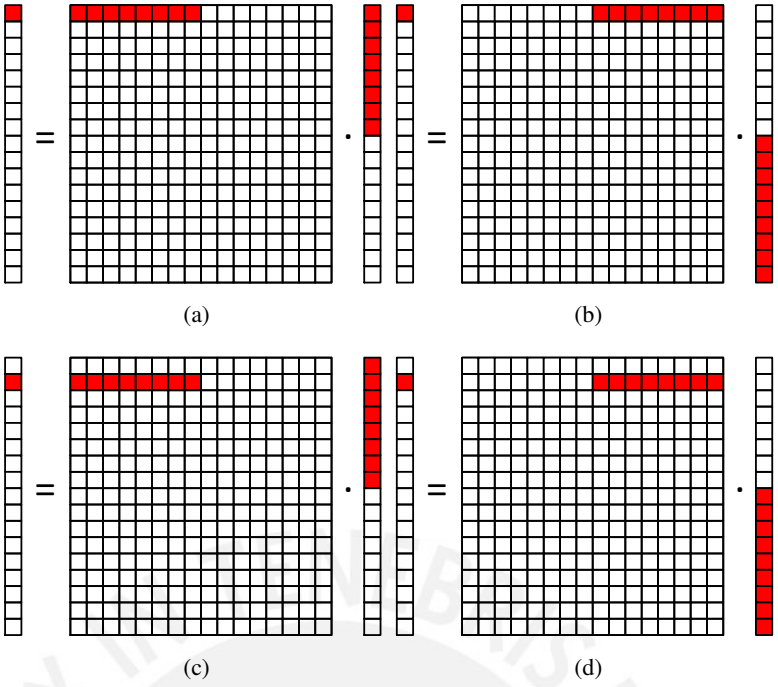


Figura 3.3: Descripción gráfica del algoritmo para el producto Matriz-Vector

3.4.4. Kernel de Cómputo

El Kernel es la unidad del coprocesador que se encarga del cálculo matemático. En este caso, el Kernel está compuesto de tres etapas de operación. En la primera etapa se multiplican los bloques de A y x en paralelo, la segunda etapa consiste en una reducción binaria de 3 etapas, en las cuáles se operan 8, 4 y 2 datos en cada etapa respectivamente. Finalmente, los resultados parciales son acumulados hasta que se haya terminado de operar. En la figura 3.6 se muestra un diagrama de las etapas del Kernel.

3.4.4.1. Multiplicadores

El bloque de multiplicadores está basado en núcleos de propiedad intelectual (LogiCORE IP) de Xilinx que operan en coma flotante de 32 bits. Los multiplicadores pueden operar a una frecuencia máxima de aproximadamente 300MHz, tiene una latencia de 10 ciclos de reloj y cuentan con señales de *handshake* de dato nuevo y listo las cuales facilitan el control de esta etapa [29].

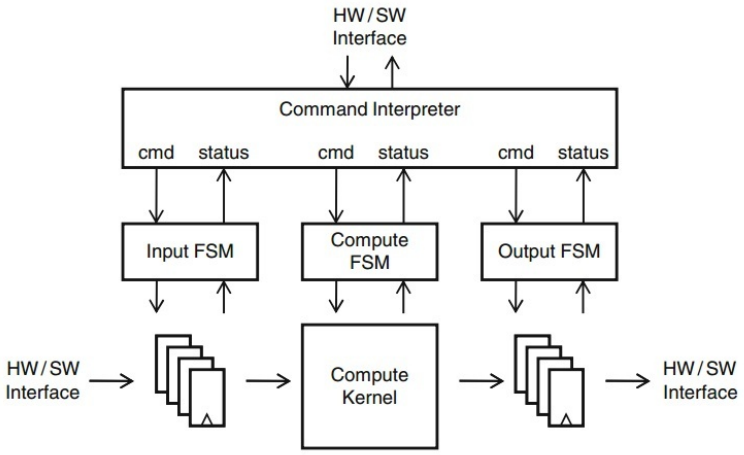


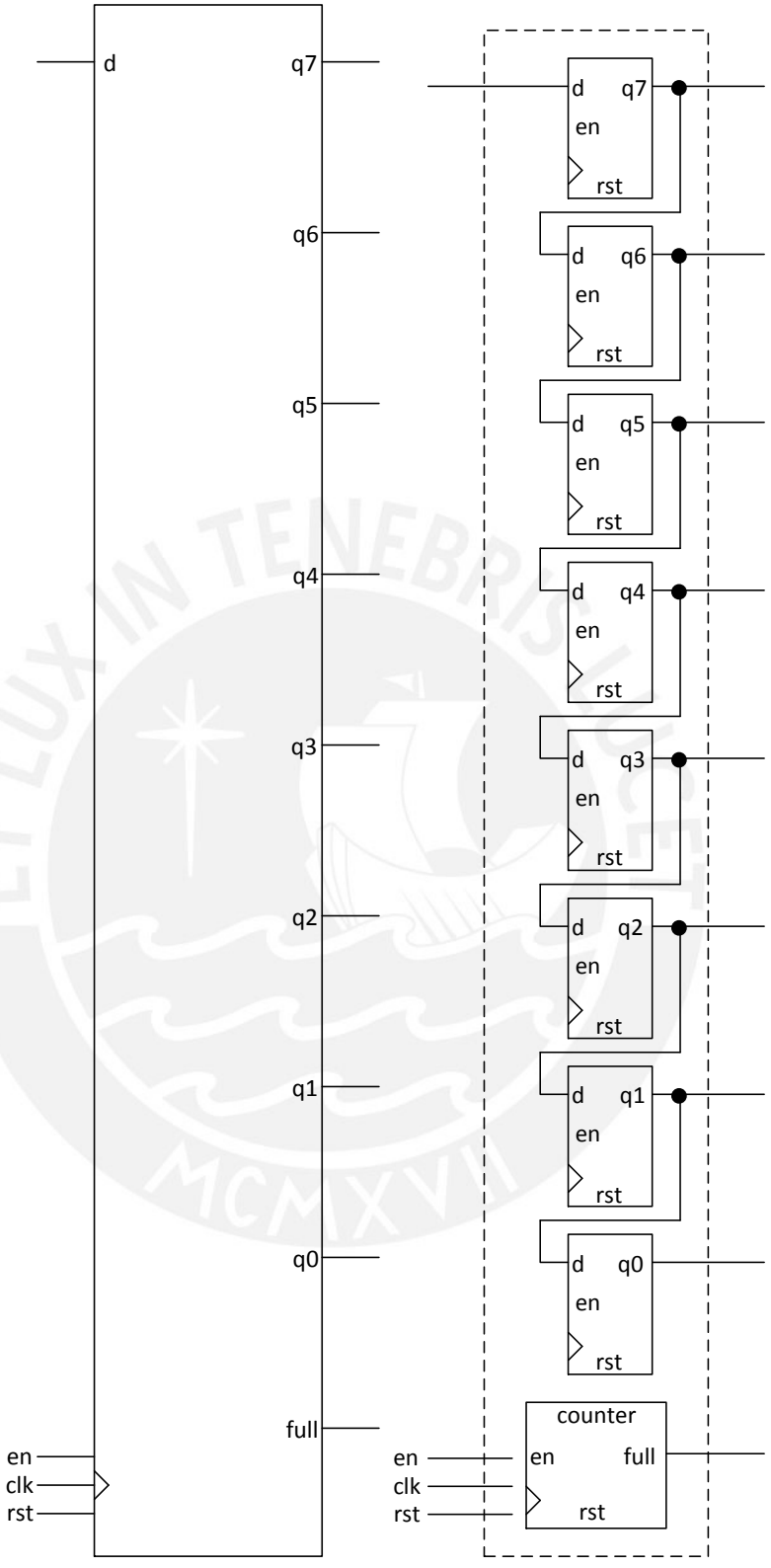
Figura 3.4: Estructura Jerárquica de un Coprocesador
[18]

3.4.4.2. Reducción Binaria

El bloque de reducción binaria, al igual que el de multiplicadores, está basado en LogiCORE s de Xilinx. Los sumadores que operan en esta etapa tienen una frecuencia de operación máxima de 300MHz, una latencia fija de 12 ciclos de reloj y poseen las mismas señales de *handshake* que los multiplicadores.

3.4.4.3. Acumulador

El bloque acumulador se encarga de sumar lo resultados parciales obtenidos hasta que se haya concluido de operar la respectiva fila de la matriz A. El sumador empleado en esta etapa es el mismo que el de la etapa de reducción binaria y posee las mismas características.



(a) Entidad del Registro de Entradas (b) Arquitectura del Registro de Entradas

Figura 3.5: Registro Serie-Paralelo

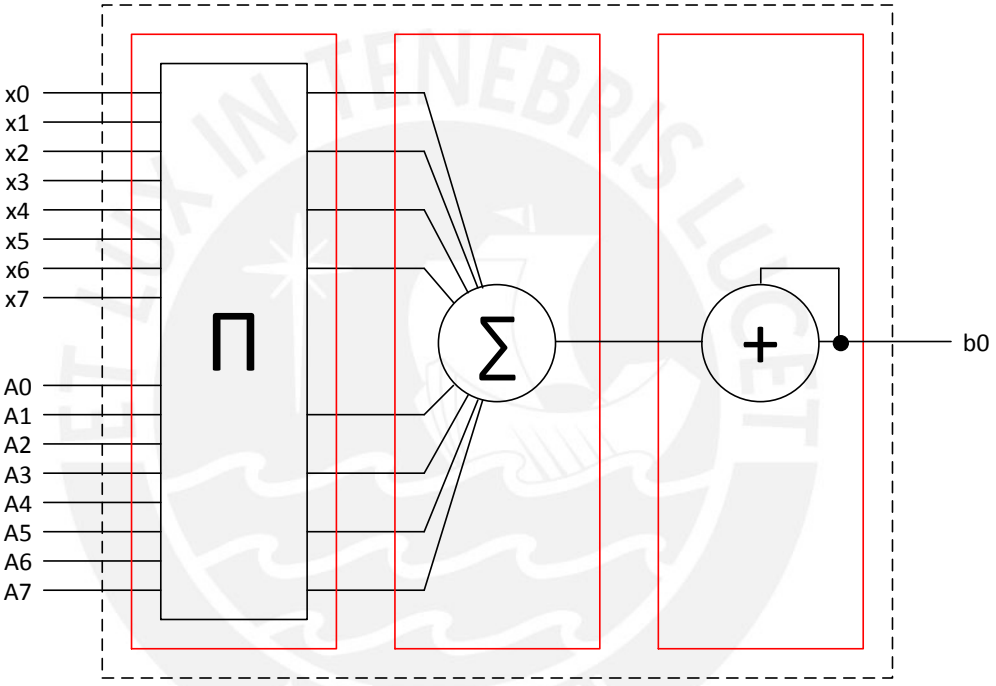
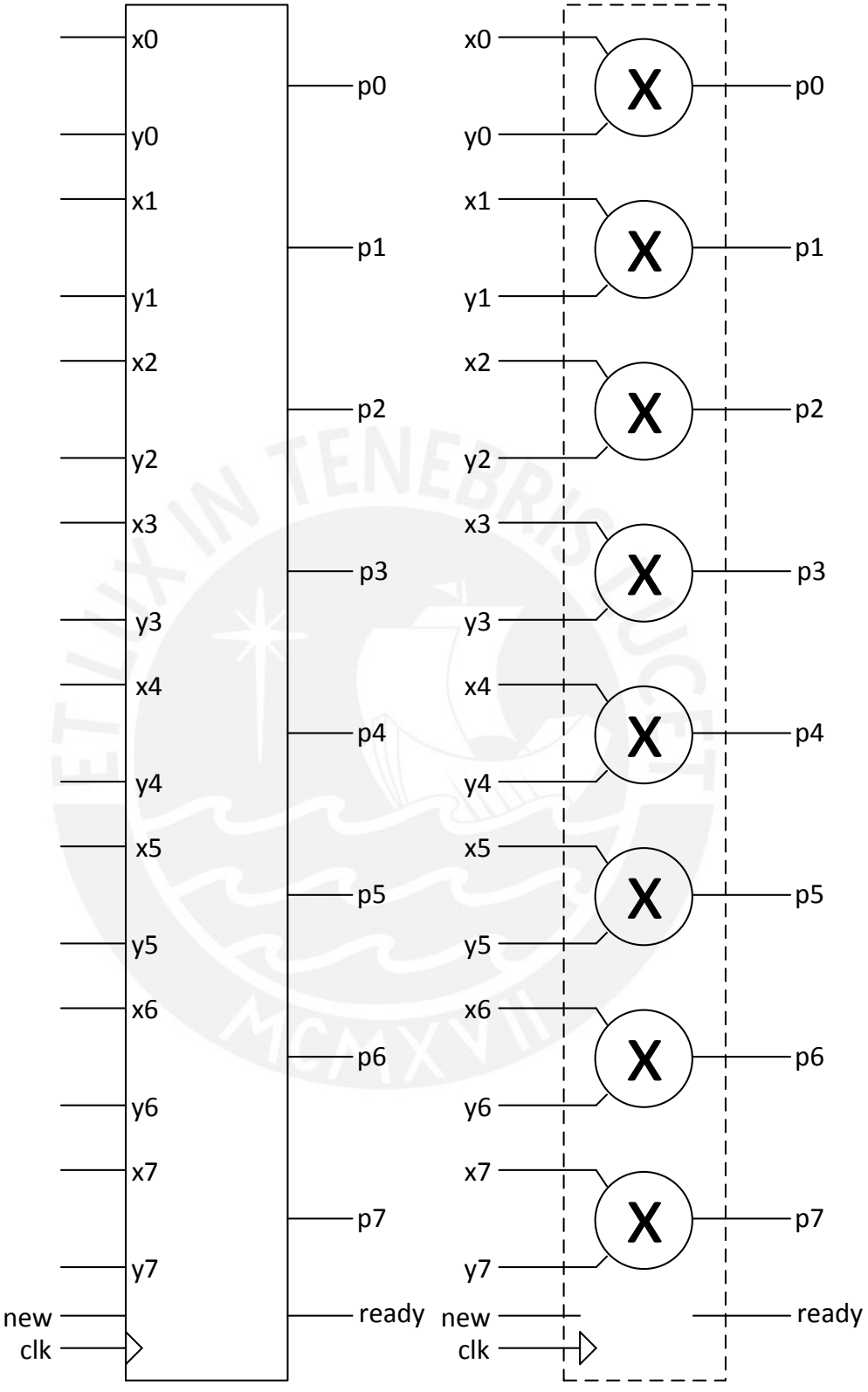


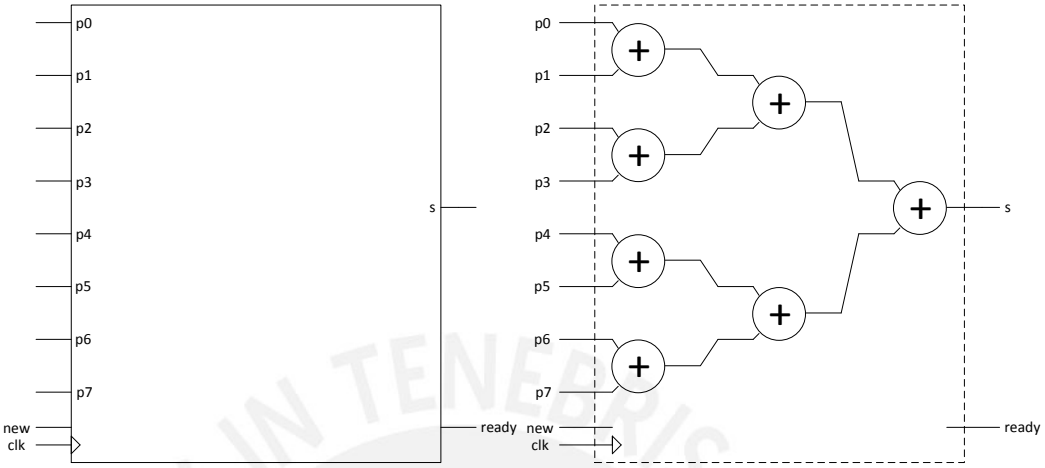
Figura 3.6: Etapas del Kernel de Computo



(a) Entidad del bloque de multiplicadores

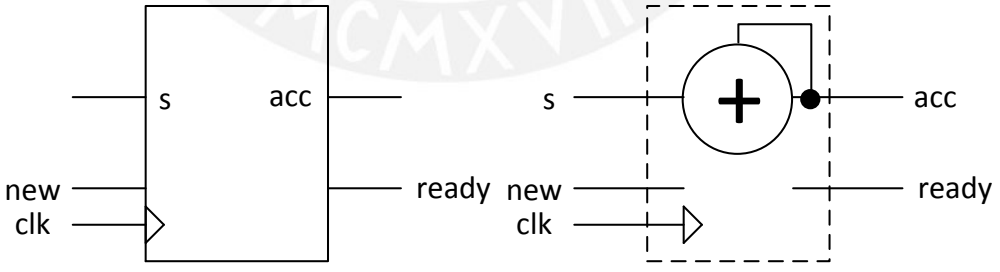
(b) Arquitectura del bloque de multiplicadores

Figura 3.7: Bloque de Multiplicadores



(a) Entidad del bloque de Reducción Binaria (b) Arquitectura del bloque de Reducción Binaria

Figura 3.8: Bloque de Reduccion Binaria



(a) Entidad del bloque Acumulador (b) Arquitectura del bloque Acumulador

Figura 3.9: Bloque Acumulador

Capítulo 4

Implementación y Resultados

4.1. Implementación del Sistema

El sistema propuesto fue implementado en la tarjeta de Desarrollo Atlys de la empresa Digilent. La tarjeta Atlys es una plataforma de desarrollo de sistemas basada en una arquitectura Spartan-6 LX45. El plataforma contiene, además del Spartan-6, una colección de periféricos que incluyen Gbit Ethernet, Video HDMI, una memoria DDR2 de 16 bits, puertos de audio y USB; los cuáles le dan el potencial para ser la plataforma ideal para el desarrollo de sistemas digitales, incluyendo diseños basados en el procesador MicroBlaze. La plataforma es compatible con todas las herramientas CAD (*Computer Aided Design*) de Xilinx, incluyendo ChipScope, EDK y ISE de manera que cualquier diseño puede ser realizado sin costo adicional [12].

4.1.1. Spartan-6

Spartan-6 es una familia de circuitos integrados que proporcionan las capacidades para desarrollar sistemas discretos y embebidos a bajo costo y con gran variedad de aplicaciones. La familia está conformada por 30 dispositivos, que contienen gran cantidad de celdas lógicas, consumen baja potencia y son más rápidos y sencillos de emplear que familias Spartan anteriores [31].

4.1.1.1. MicroBlaze

MicroBlaze es un soft-procesador embebido RISC (*Reduced Instruction Set Computer*) optimizado para aplicaciones científicas y de ingeniería. El soft-procesador MicroBlaze es configurable, lo que permite al diseñador el conjunto de características que requiera su aplicación.

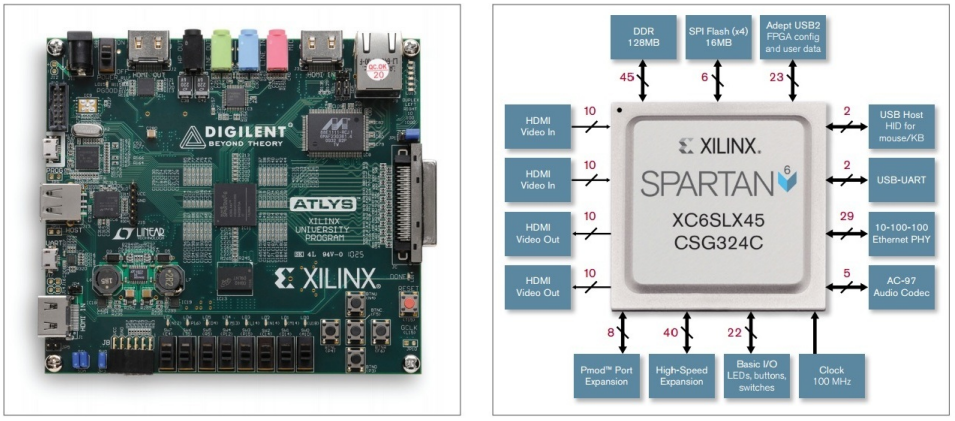


Figura 4.1: Targeta Atlys de Digilent con FPGA Spartan 6
[30]

MicroBlaze posee 32 registros de propósito general, instrucciones de 32 bits con tres operandos y tres modos de direccionamiento, buses de 32 bits y pipeline de 3 etápas. Adicionalmente, MicroBlaze está parametrizado para permitir que habiliten funciones adicionales a un nivel de análisis más bajo [32].

4.1.1.2. Interfaz Fast Simplex Link (FSL)

El IPcore Fast Simplex Link (FSL) es un bus de comunicación unidireccional punto a punto usado para desarrollar una rápida transferencia de datos entre dos subsistemas en un FPGA. La interfaz está disponible en el procesador MicroBlaze y se emplea para transferir datos desde el archivo de registros del procesador a un hardware en el FPGA y viceversa. El FSL proporciona un mecanismo para comunicación no arbitrada y no compartida, esto es empleado para una rápida transferencia de datos entre elementos maestro-esclavo. Asimismo, la interfaz proporciona un bit de control el cual se podría emplear para distinguir el inicio o fin de una trama o datos de intrucciones. El FSL puede operar tanto en modo síncrono como asíncrono, puede estar basado en FIFOs o BRAMs y tiene una capacidad de hasta 8K [33]. Un ejemplo de aplicación del FSL, junto con una descripción de sus modos de operación y demás potencialidades se brinda en [34].

4.2. Software de Implementación

Para la implementación del sistema se empleó el paquete ISE (*Integrated Software Environment*) de la empresa Xilinx versión 14.4. El entorno está desarrollado para la síntesis, diseño y

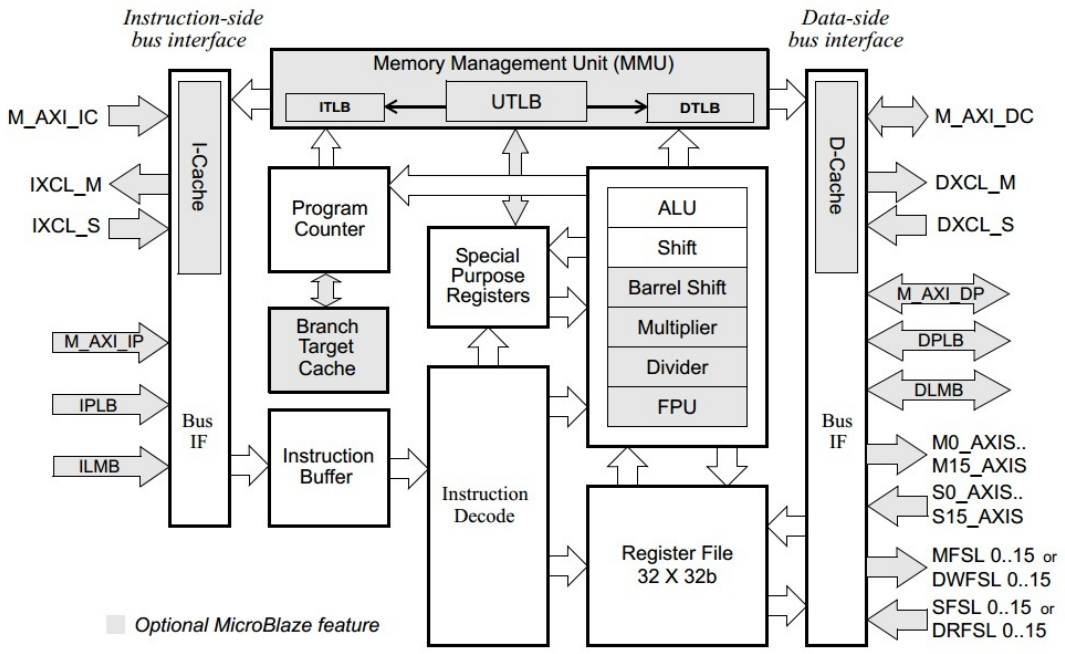


Figura 4.2: Estructura del procesador MicroBlaze
[32]

análisis de sistemas HDL (*Hardware Description Language*) [35]. Asimismo, también permite la evaluación de *timing* de los circuitos, el análisis de sus respectivos diagramas RTL (*Register Transfer Level*) y diferentes niveles de simulación y depuración [36].

4.3. Descripción de la implementación

4.3.1. Configuración e instancia del procesador MicroBlaze

MicroBlaze fue configurado con la herramienta XPS (*Xilinx Platform Studio*) que viene incluida en el paquete ISE. El sistema fue configurado para poseer un único núcleo y se le agregó los cores necesarios para poder emplear el depurador, el timer, la memoria externa y el UART. También se configuró para que se optimizara el área y el *throughput*, y se seleccionó 100MHz como frecuencia de operación. Adicionalmente, luego de que estás características fueran desarrolladas, se le agregó al procesador los puertos para que soportara dos interfaces FSL, una en modo maestro y otra en esclavo [37].

Figura 4.3: Diagrama de Bloques del Fast Simplex Link
[33]

4.3.2. Configuración e instancia de la interfaz FSL

La FSL fue configurada empleando el wizard para crear/exportar perifericos del XPS. Se seleccionó el modo síncrono, a una frecuencia de operación de 100MHz, debido a las limitaciones en frecuencia de operación de los IPcores de la unidad en coma flotante del coprocesador, pues el modo asíncrono operaba a frecuencias de 600MHz. Por otro lado, para poder tener una transferencia de datos sencilla fue implementada empleando FIFOs de ancho de 8 palabras de 32 bits cada una, pues se procesaban bloques de la misma longitud.

4.3.3. Configuración e instancia del coprocesador

El coprocesador fue implementado en VHDL siguiendo las especificaciones del capítulo 3 con el uso de la herramienta ISE-PN (*ISE Project Navigator*). Para integrar el coprocesador al sistema se empleó el wizar para configuración del XPS. Antes de integrar el coprocesador, se verificó que cumpliera con los requerimientos de frecuencia de operación y timing [38]. Para verificar la funcionalidad del coprocesador se desarrollaron simulaciones a 3 niveles: *Behavioral*, *Structural* y *Timing*. Con el primer nivel se verifica la lógica del circuito, con el segundo las capacidades físicas y con el tercero la fiabilidad de la implementación final. Para poder detectar errores o realizar depuraciones con el circuito final será necesrio hacer uso de herramientas como *ChipScope* o el *SDK-Debugger* las cuales permiten analizar tanto las variables del programa, como las señales del co-procesador en tiempo real. Referencias de como realizar estos procedimientos se pueden encontrar en [39] y [40].

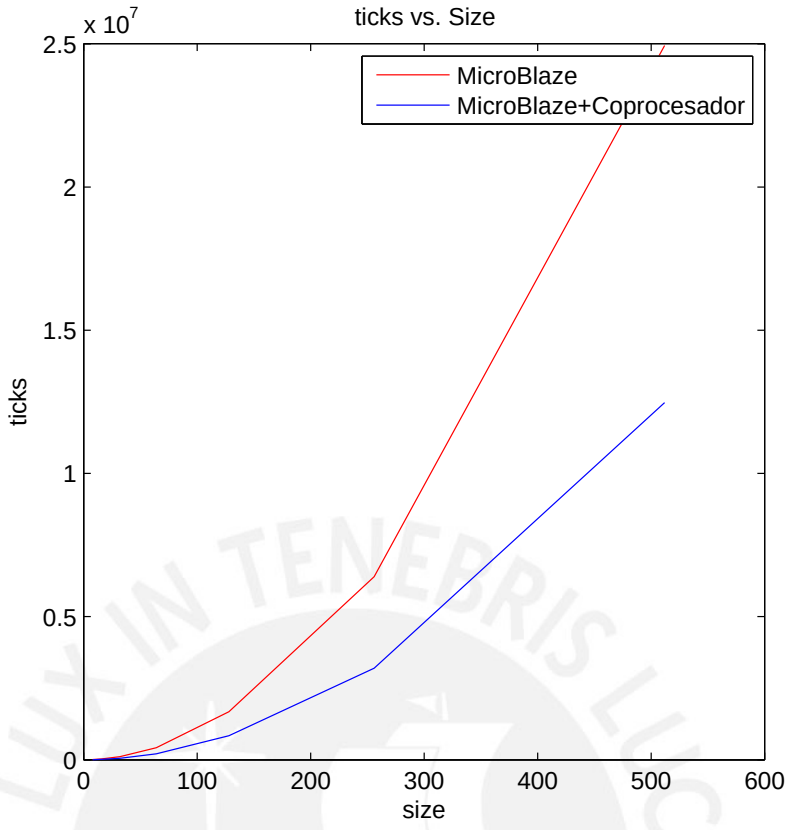


Figura 4.4: Número de ticks vs. Tamaño del problema

4.4. Resultados Computacionales

El método del CG presentado en el capítulo 2 fue implementado empleando solo el procesador y su respectiva mejora. Para la comparación de los resultados de ambas implementaciones y validar la mejora se emplearon 4 métricas: El número de ciclos de reloj (*ticks*), los megaflops, el speedup y la eficiencia [11]. Con fines estadísticos se realizaron pruebas para tamaños potencias de 2 desde el 8 hasta el 512 y por cada tamaño se realizaron 10 pruebas.

En la figura 4.4 se aprecia la comparación entre el número de ciclos de reloj del procesador y el sistema completo. Los ticks se definen desde el inicio hasta el fin del algoritmo. Las ventajas de emplear esta métrica, en vez del tiempo de respuesta es que toma en cuenta las condiciones de operación del sistema que se desea evaluar.

Otra métrica empleada fueron los MFLOPS (Millones de operaciones en coma flotante por segundo). Los MFLOPS se calculan de la siguiente manera:

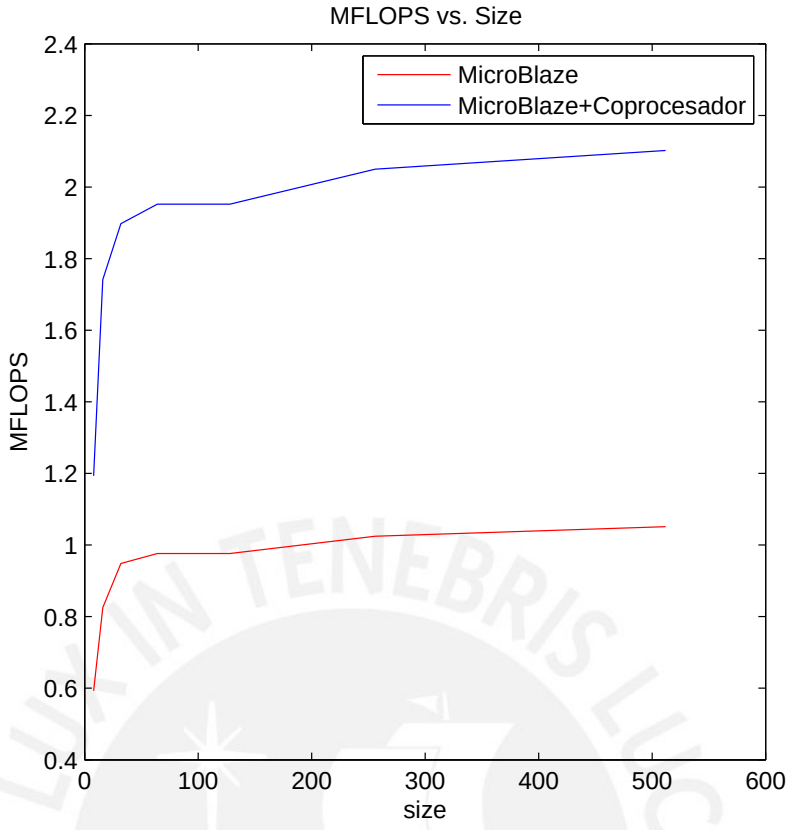


Figura 4.5: Número de operaciones en coma flotante vs. Tamaño del problema

$$MFLOPS(A) = \frac{n_{flp-op}(A)}{T_{CPU(A)},10^6} \tag{4.1}$$

donde $n_{flp-op}(A)$ es el número de operaciones en coma flotante ejecutadas por A. Los MFLOPS están basados en la cantidad de valores calculados por la ejecución de instrucciones aritméticas en coma flotante, por lo que valores más altos de MFLOPS corresponden a tiempos de ejecución menores. En la figura 4.5 se puede apreciar la comparación de MFLOPS entre la implementación de referencia y la versión mejorada.

Por otro lado, para medir el beneficio obtenido por el paralelismo se emplea el speedup. El speedup de un programa paralelo se calcula de la siguiente manera:

$$S_p(n) = \frac{T(n)}{T_p(n)} \tag{4.2}$$

donde p es el número de elementos de procesamiento en paralelo empleados para un problema de tamaño n. $T(n)$ es el tiempo de ejecución de la mejor implementación secuencial que

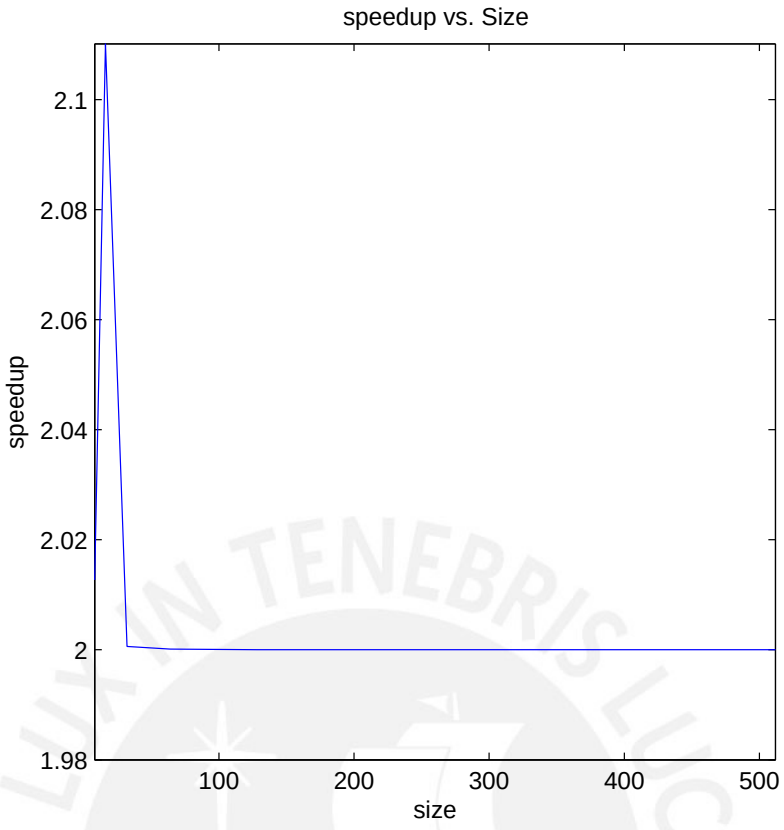


Figura 4.6: Speedup vs. Tamaño del problema

resuelve el mismo problema. En la figura 4.6 se puede observar el speedup obtenido, el cual tenía una valor aproximado de 2 y que se mantuvo aproximadamente constante para todos los tamaños.

Finalmente, la última métrica empleada para medir el rendimiento del programa paralelo es la eficiencia. La eficiencia se puede entender como la fracción de tiempo por el que es empleado útilmente un elemento de procesamiento. La eficiencia puede ser calculada de la siguiente manera:

$$E_p(n) = \frac{S_p(n)}{p} \tag{4.3}$$

donde $S_p(n)$ es el speedup y p es el número de elementos de procesamiento paralelo. En la gráfica 4.7 se puede apreciar que la implementación final presenta una eficiencia de 0.25.

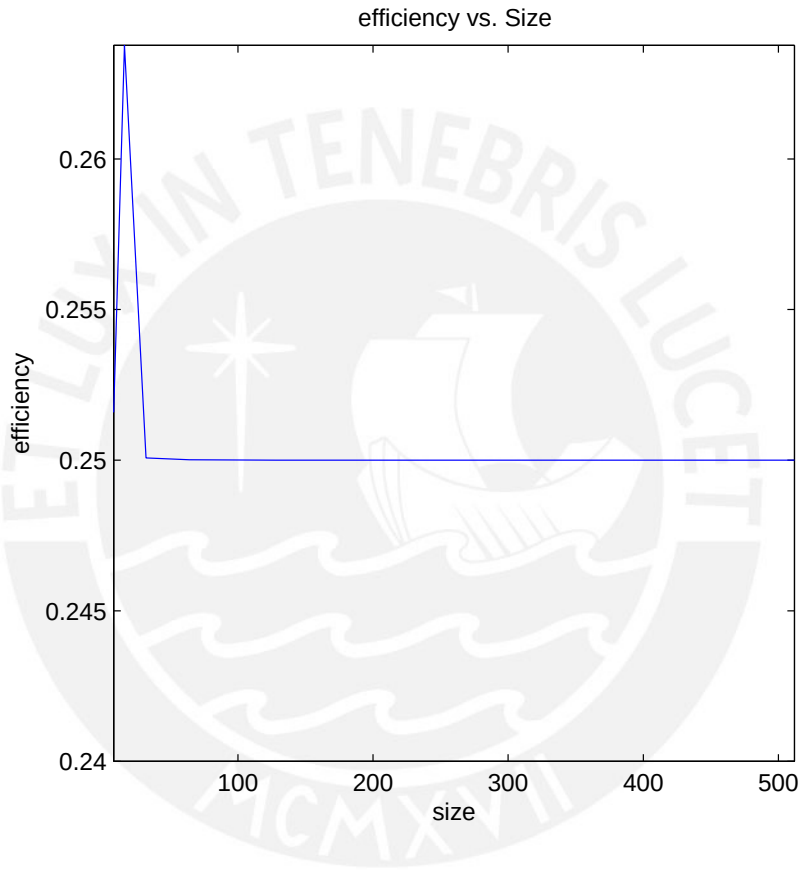


Figura 4.7: Eficiencia vs. Tamaño del problema

Conclusiones

En este trabajo, se propuso un sistema digital que desarrolle el método de Gradiente Conjugado, el cual estaba basado principalmente en un procesador y un coprocesador. En contraste a la mayoría de soluciones que emplean integramente software o hardware, la propuesta de solución presentaba un aporte de ambas metodologías. De manera específica, el trabajo central fue diseñar el coprocesador que efectúa el producto Matriz-Vector, pues ésta es la operación más costosa del CG, para así mejorar su rendimiento computacional. La implementación final presentó resultados favorables, pues el tiempo de ejecución resultó menor al de la implementación de referencia que se desarrolló en el procesador MicroBlaze. Asimismo, se obtuvo una ganancia en tiempo de ejecución (speedup) de valor 2 respecto de la implementación de referencia, la cuál resultó ser aproximadamente constante en la mayoría de tamaños de problema. En cuanto a la precisión de cálculo se presentan resultados adecuados (error relativo $\leq 0{,}05$) lo cual significa que la precisión empleada como tope en el método es adecuada.

Recomendaciones y Observaciones

Una de las desventajas del algoritmo propuesto para la operación Matriz-Vector es que el vector x es reutilizado durante todo el cálculo del vector b, por está razón sería conveniente que se agregará una memoria BRAM al coprocesador para almacenar todo el vector x o una porción considerable, como propone Fujimoto en [41]. De esta forma se reducirían las pérdidas en transferencia de datos. Asimismo, aprovechando que la matriz A no cambia durante todo el proceso del algoritmo, se podría emplear una BRAM adicional para almacenar parte de esta o toda en el mejor caso. Por otro lado, emplear el modo asíncrono permitiría que el coprocesador y procesador funcionaran a frecuencias diferentes. Si bien la plataforma sólo ofrece frecuencias de 100MHz y 600MHz, se podría emplear divisores de frecuencia para acondicionar el coprocesador a frecuencias que estén dentro de su capacidad, pero superiores a la frecuencia del procesador. Así se reduciría el tiempo de procesamiento del coprocesador. Por último, se menciona que la plataforma permite implementar un sistema que esté basado en dos procesadores MicroBlaze. Cada procesador soporta sus propias interfaces FSL, así se les podría conectar su respectivo coprocesador y distribuir los cómputos para obtener aún más ganancia.

Bibliografía

[1] G. Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 2003.

[2] L. Abbasabadi, "Solving non square fuzzy linear systems of equations by use of the generalized inverse," *International Conference on Applied Mathematics and Pharmaceutical Sciences*, 2012.

[3] P. Pandit, "Fully fuzzy system of linear equations," *International Journal of Soft Computing and Engineering*, 2012.

[4] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, pp. 1155–1178, 2007.

[5] A. Edelman, T. A. Arias, T. A. Arias, and S. T. Smith, "Curvature in conjugate gradient eigenvalue computation with applications to materials and chemistry calculations," in *In Proceedings of the 1994 SIAM Applied Linear Algebra Conference*, 1994.

[6] D. Fadeev and V. Fadeeva, *Computational Methods of Linear Algebra*. Freeman, 1963.

[7] G. Golub, *Matrix Computations*. The Johns Hopkins University Press, 1996.

[8] J. Nocedal, *Numerical Optimization*. Springer, 2000.

[9] P. Rodriguez and B. Wohlberg, "Iteratively reweighted norm algorithm for total variation regularization," 2006.

[10] K. Koh, S.-J. Kim, and S. Boyd, "An interior-point method for large-scale l1-regularized logistic regression," *Journal of Machine Learning Research*, vol. 8, pp. 1519–1555, 2007.

[11] T. Rauber and G. Runger, *Parallel Programming for Multicore and Cluster Systems*. Springer, 2010.

[12] Digilent, *Atlys Board Reference Manual*. Digilent, 2013.

[13] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. 1997.

[14] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. H. Still, "Exploring traditional and emerging parallel programming models using a proxy application," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, IEEE Computer Society, 2013. LLNL-CONF-586774.

[15] D. Ratter, "Fpga on mars," tech. rep., Xilinx, 2004.

[16] P. Bergsman, "Xilinx fpga blasted into orbit," tech. rep., Xilinx, 2003.

[17] J. Munoz, "Modulo controlador de radar 16 canales," tech. rep., Radio Observatorio de Jicamarca, 2008.

[18] P. Schaumont, *A Practical Introduction to Hardware/Software Co-Design*. Springer, 2010.

[19] F. Vahid and T. Givargis, *Embedded System Design - A Unified Harware-Software Approach*. Department of Computer Science and Engineering - University of California, 1999.

[20] P. Chu, *RTL Hardware Design with VHDL*. Wiley-Interscience, 2006.

[21] P. Schaumont, "Hardware/software codesign is a starting point in embedded systems architecture education," *Workshop on Embedded Systems Education*, 2008.

[22] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[23] A. Roldao and G. A. Constantinides, "A high throughput fpga-based floating point conjugate gradient implementation for dense matrices," *Transactions on Reconfigurable Technology and Systems*, vol. 3, pp. 1:1–1:19, 2010.

[24] T. Habbestad, "An fpga-based implementation of the conjugate gradient method used to solve large dense systems of linear equations," Master's thesis, Norwegin University of Science and Technology, 2011.

[25] J. D. Bakos and K. K. Nagar, "Exploiting matrix symmetry to improve fpga-accelerated conjugate gradient," *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 223–226, 2009.

[26] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, and V. Eijkhout, *Templates for the Solutions of Linear Systems: Building Blocks for Iterative Methods*. Software Environment Tools, 1994.

[27] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *Circuits, Devices and Systems, IEE Proceedings G*, 1992.

[28] N. Nedjah and L. de Macedo Mourelle, *Co-Design for System Acceleration*. Springer, 2007.

[29] Xilinx, "Logicore ip floating-point operator v5.0," tech. rep., Xilinx, 2011.

[30] Xilinx, "Connectivity and embedded processing power for digital logic designs, dsp, video and image processing," tech. rep., Xilinx, 2013.

[31] Xilinx, "Spartan-6 family overview," tech. rep., Xilinx, 2011.

[32] Xilinx, *MicroBlaze Processor Reference Guide Embedded Development Kit*. Xilinx, 2012.

[33] Xilinx, "Logicore ip fast simplex link (fsl) v20 bus (v2.11c)," tech. rep., Xilinx, 2010.

[34] H.-P. Rosinger, "Connecting customized ip to the microblaze soft processor using fast simplex link," tech. rep., Xilinx, 2004.

[35] Xilinx, *Xilinx Design Tools: Installation and Licensing Guide - Vivado Design Suite and ISE Design Suite*. Xilinx, 2012.

[36] Xilinx, *ISE In-Depth Tutorial*. Xilinx, 2012.

[37] Xilinx, *Embedded System Tools Reference Manual*. Xilinx, 2012.

[38] Xilinx, *Synthesis and Simulation Design Guide*. Xilinx, 2011.

[39] Xilinx, *Isim User Guide*. Xilinx, 2012.

[40] R. Griffith and F. Pang, *MicroBlaze System Performance Tunning*. Xilinx, 2008.

[41] N. Fujimoto, "Faster matrix-vector multiplication on geforce 8800 gtx," *IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8, April 2008.