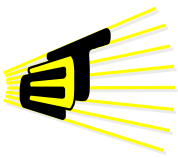
EVALUACIÓN DE UNA HERRAMIENTA DE COMPILACIÓN C TO VHDL PARA LA IMPLEMENTACIÓN DE PROCESADORES DE PROPÓSITO ESPECÍFICO SOBRE FPGA

JORGE HUMBERTO BEDOYA OJEDA
CARLOS ARTURO BOADA QUIJANO



Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

EVALUACIÓN DE UNA HERRAMIENTA DE COMPILACIÓN C TO VHDL PARA LA IMPLEMENTACIÓN DE PROCESADORES DE PROPÓSITO ESPECÍFICO SOBRE FPGA

JORGE HUMBERTO BEDOYA OJEDA

CARLOS ARTURO BOADA QUIJANO

Trabajo de Investigación para optar al título de
Ingeniero Electrónico

Director

MSc. CARLOS AUGUSTO FAJARDO ARIZA

Co-Director

MSc. SERGIO ALBERTO ABREO CARRILLO

Escuela de Ingenierías
Eléctrica, Electrónica
y de Telecomunicaciones

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2012

Agradecimientos

Agradecemos primeramente a Dios, padre todo poderoso por darnos la vida, la fortaleza y sabiduría para poder llevar nuestras metas a feliz término.

A Impulse Accelerated Technologies y en especial a Brian Durwood por facilitarnos la licencia de evaluación de Impulse C.

A nuestro director Carlos Augusto Fajardo Ariza por sus orientaciones y consejos profesionales y personales.

A todos nuestros maestros quienes dejaron una huella imborrable en nuestra formación como Ingenieros y en especial a William Alexánder Salamanca Becerra por sus conocimientos y su calidad profesional como personal.

Índice

Lista de Figuras

Lista de Tablas

Resumen

TITULO: EVALUACIÓN DE UNA HERRAMIENTA DE COMPILACIÓN C TO VHDL PARA LA IMPLEMENTACIÓN DE PROCESADORES DE PROPÓSITO ESPECÍFICO SOBRE FPGA.*

AUTORES: JORGE HUMBERTO BEDOYA OJEDA y CARLOS ARTURO BOADA QUIJANO**

PALABRAS CLAVE: Impulse C, FPGA, matriz de covarianza, Montecarlo, C to VHDL.

Para algunos problemas de cómputo intensivo, las FPGA han demostrado ser una solución superior a arquitecturas como las CPU e incluso las GPU. Sin embargo, la implementación de un algoritmo sobre esta tecnología requiere la "traducción" del mismo, a un Lenguaje de Descripción de Hardware (HDL), lo cual, consume un mayor tiempo de diseño y requiere de conocimientos en diseño digital.

Con el fin de facilitar la implementación de algoritmos sobre las FPGA, recientemente se han desarrollado compiladores que hacen el trabajo de "traducción". Este tipo de compiladores se conocen con el nombre de Compiladores *C to HDL*. El objetivo de este proyecto es evaluar la herramienta de compilación Impulse C en tres aspectos específicos: tiempo de diseño, rendimiento computacional y tamaño.

Para realizar la evaluación se han utilizado dos algoritmos, el primero de ellos calcula la matriz de covarianza para un conjunto de tres variables y el segundo realiza una integral de orden dos por medio del método de Montecarlo. La metodología de evaluación consiste en implementar los dos algoritmos de tres formas diferentes: 1) desarrollo en lenguaje C e implementación en una arquitectura CPU, 2) desarrollo mediante el compilador Impulse C e implementación en una arquitectura FPGA y 3) la implementación en una FPGA mediante metodologías tradicionales de diseño digital.

Los resultados del presente trabajo muestran que la herramienta evaluada puede reducir hasta 10 veces el tiempo de diseño; también logra desempeños computacionales de hasta 120 veces cuando se comparan con la implementación en la arquitectura CPU y desempeños similares cuando se compara con la implementación en la arquitectura FPGA realizada en forma tradicional; en cuanto a la administración de los recursos lógicos se observó una fuerte dependencia con el tipo de aplicación.

* Trabajo de Investigación

** **Facultad:** Facultad de Ingenierías Físico Mecánicas. **Escuela:** Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. **Grupo de Investigación:** CPS. **Director:** MSc. Carlos Augusto Fajardo Ariza

Abstract

TITLE: EVALUATION OF A COMPILATION TOOL C TO VHDL FOR THE IMPLEMENTATION OF APPLICATION-SPECIFIC PROCESSORS ON FPGA.*

AUTHOR: JORGE HUMBERTO BEDOYA OJEDA and CARLOS ARTURO BOADA QUIJANO**

KEY WORDS: Impulse C, FPGA, covariance matrix, Monte Carlo, C to VHDL.

For computationally intensive problems, FPGAs have shown to be a superior solution to architectures like CPUs or even GPUs. However, the implementation of an algorithm on this technology requires "translation" to a Hardware Description Language (HDL), which is more time-consuming and requires knowledge about digital design. In order to facilitate the implementation of algorithms on FPGA, *C to HDL* Compilers have recently been developed to make the "translation" task.

The objective of this project is to evaluate the Impulse C compiler tool in three specific areas: design time, computational performance and size. Two algorithms were used to perform the evaluation: the first computes the covariance matrix for a set of three variables while the second computes a second order integral through the Monte Carlo method.

The methodology for the evaluation consist of implementing both algorithms in three different ways: development and implementation in C on a CPU architecture, development in Impulse C compiler and implementation in FPGA architecture, and finally the implementation in an FPGA using traditional methodologies digital design.

The results of this study evidence that the tool can be evaluated up to 10 times the design time. It also achieves computational performance of up to 120 times when compared to the CPU architecture implementation and similar performance when compared to the implementation in the architecture FPGA performed in traditional manner. Moreover, in terms of logical resource administration, there was a strong dependence on the type of application.

* Research Work

** **Faculty:** Physico-mechanical Engineering Faculty. **School:** School of Electrical, Electronics and Telecommunications Engineering. **Research group:** CPS. **Director:** MSc. Carlos Augusto Fajardo Ariza

Capítulo 1

Introducción

Actualmente existe la necesidad de facilitar la implementación de algoritmos computacionales sobre dispositivos lógicos programables, como por ejemplo las FPGAs[3]. Al respecto existen principalmente dos propuestas: la primera está relacionada con los lenguajes de alto nivel para el diseño de hardware como por ejemplo Handel C[4] de la empresa Mentor Graphics[5] y System C[6] desarrollado por la compañía Accellera Systems Initiative[7]. A pesar de que estas herramientas son de gran utilidad a la hora de desarrollar algoritmos sobre hardware, este tipo de enfoque posee como desventaja que los diseñadores requieren conocimientos en el área de sistemas digitales y adicionalmente los programadores deben reescribir sus códigos, ya que semánticamente hablando, son nuevos lenguajes de programación[8]. El segundo enfoque es utilizar compiladores C to HDL, los cuales son herramientas capaces de convertir un algoritmo desarrollado en lenguaje C a un lenguaje de descripción de hardware.

Este trabajo se centra en el segundo enfoque y su objetivo es medir la eficiencia del compilador Impulse C[9], al ser usado para generar procesadores de propósito específico, que posteriormente puedan ser implementados en un FPGA. La estructura del artículo es la siguiente: En la primera parte se dará una breve introducción sobre qué es un compilador y cómo trabaja. Se hará una breve descripción sobre el compilador C to VHDL llamado Impulse C, que será la herramienta a evaluar. En la siguiente sección se hablará de la metodología utilizada para hacer la evaluación de la herramienta y los algoritmos seleccionados para tal fin. Se trata específicamente del cálculo de la matriz de covarianza[10] y de la solución de una integral de orden dos por medio del método Montecarlo[11]. Para ello se desarrollaron tres enfoques:

1. Desarrollo de los algoritmos seleccionados mediante un programa computacional como Matlab o DevC con el fin de determinar los tiempos de ejecución sobre una arqutectura CPU.

2. Compilación del programa en C con el compilador C to VHDL a evaluar, analizar su comportamiento tanto en simulación como al ser implementado en el FPGA. Para tal fin se usará la herramienta ISE[12] y en especial Chipscope[13].

3. Desarrollo manual de los algoritmos seleccionados, mediante un lenguaje de descripción de hardware (VHDL), simulación e implementación de los mismos en una FPGA y al igual que en el caso anterior la evaluación se realizará mediante Chipscope.

Por último, se muestra una sección de resultados y conclusiones en las cuales se podrá observar el comportamiento de los diferentes enfoques usados para evaluar la herramienta, en términos de recursos utilizados, tiempo de ejecución y tiempo de desarrollo. Con base en dichos resultados se determinó si la herramienta cumplía con las características que se desean en un diseño digital y que la descripción de hardware que arroja como resultado de la compilación pueda ser empleada, de una manera rápida y confiable.

Capítulo 2

Compiladores

En palabras sencillas un compilador es aquel programa capaz de tomar una entrada de texto de un algoritmo escrito en cierto lenguaje (lenguaje fuente) y producir como salida el texto de este algoritmo en otro lenguaje (lenguaje destino) el cual respeta el significado del texto original. Este proceso se conoce como traducción[14]. En el caso de los compiladores C to HDL el lenguaje fuente será C o C++ y el lenguaje destino será un lenguaje de descripción de hardware, de preferencia VHDL.

2.1 Estructura del Compilador

El proceso de compilación se encuentra estructurado mediante dos etapas: el análisis y la síntesis.

El análisis es la parte encargada de dividir el programa de entrada en componentes y darles una estructura gramatical. También permite detectar si el programa fuente está mal escrito en relación con la sintaxis ó si la semántica no es consistente y en tal caso presentar un mensaje informativo para que el usuario pueda corregirlo, además permite recopilar información sobre el programa de entrada, la cual es almacenada en una estructura de datos llamada tabla de símbolos.[15].

La síntesis construye el programa destino a partir de la representación intermedia y la tabla de símbolos.[16].

2.2 Compiladores C to HDL

Son herramientas de conversión de código C (lenguaje de alto nivel) hacia un lenguaje HDL (*Hardware Description Language*). Estos tipos de compiladores surgen de la necesidad de evitar en lo posible la tarea de implementar un algoritmo en una FPGA mediante un lenguaje de descripción de hardware, ya que para esto se necesitan conocimientos relacionados con el diseño digital avanzado[8].

Estas herramientas de conversión de código C a un lenguaje de descripcion de hardware HDL, en su mayoría son usadas en aplicaciones en las cuales los tiempos de ejecución son demasiado altos en arquitecturas CPU, razón por la cual se implementan en FPGAs con el fin de obtener una "aceleración hardware"[17]. Algunas de estas aplicaciones se encuentran en los campos de la Bioinformática, en la Dinámica de fluidos, en Análisis Estadístico y en la construcción de imágenes del subsuelo, en procesos de búsqueda de hidrocarburos[18][19].

Actualmente existe gran variedad de estos compiladores C to HDL [8][20]. Por un lado se encuentran los compiladores de licencia libre dentro de los cuales se resaltan SPARK[21], ROCCC[22] y DWARV[23], y los de licencia paga como CATAPULT-C[24], DIME-C[25], IMPULSE C[9] y Vivado Design Suite[26]. El compilador C to HDL seleccionado es Impulse C, esto debido a que la empresa que lo desarrolló facilitó el uso de una versión de evaluación de la herramienta, además, no hubo respuesta a la solicitud de este tipo de licencia por parte de las otras casas desarrolladoras. En cuanto a las herramientas de compilación disponibles para software libre se presentaron problemas a la hora de adquirir ciertas librerías sin las cuales los compiladores no trabajaban de manera satisfactoria. Las características principales de la herramienta de compilación Impulse C se muestran a continuación.

Impulse C

Impulse C[9] es un software de herramientas avanzadas con las cuales se obtienen aplicaciones con alto rendimiento sobre plataformas basadas en FPGAs. Estas herramientas permiten ejecutar mediante un entorno gráfico, procesos que pueden ser algoritmos o programas desarrollados en estándar C o C++, para ser traducidos a un lenguaje de descripcion de hardware. Adicionalmente la herramienta cuenta con una biblioteca de funciones y tipos de datos relacionados que proporcionan un entorno de programación para facilitar la conversión del código de lenguaje C a HDL. Su principal característica es un compilador cruzado C to HDL que le permite generar aplicaciones que pueden ser implementadas en plataformas hardware[1].

Algunas de las características principales de Impulse C son[1]:

- Generación optimizada de una descripción RTL(Register Transmition Level) en VHDL o Verilog a partir de C.
- Partición de los algoritmos que utilizan múltiples procesos (Programación Paralela).
- Optimización Interactiva mediante un entorno grafico, *Pipeline* y Paralelismo de la aplicación obtenida en VHDL a partir de C para alcanzar un alto rendimiento.
- Describe y genera las entradas y salidas, en forma de *streams* (cadena de caracteres), señales, memorias y registros.

El manejo de los datos es llevado a cabo mediante FIFOs que se especifican y configuran en la herramienta. Esto hace posible escribir aplicaciones paralelas en un nivel mayor de abstracción, sin la necesidad de sincronización que de otro modo seria necesaria[1].

La versión de Impulse C con la cual se trabajó este proyecto es la 3.70.b disponible desde el 28 de Julio del 2011.

Capítulo 3

Metodología

Para llevar a cabo la evaluación del compilador se seleccionaron dos aplicaciones con alta probabilidad de ser paralelizadas, para observar el comportamiento de la herramienta y analizar si la descripción de hardware que se obtenía como resultado, cumplía con ciertos parámetros de evaluación seleccionados, los cuales son el tiempo de desarrollo, área ocupada en la FPGA y tiempo de ejecución.

Las aplicaciones seleccionadas fueron el cálculo de la matriz de covarianza[10] y el método Montecarlo para la solución numérica de integrales de orden superior[11]. En la Figura 3.1 se muestra la metodología que se empleó para la evaluación del compilador. Para ello se realizaron tres implementaciones para cada una de las dos aplicaciones: una implementación en software, una desarrollada usando la herramienta Impulse C y por último una descripción de hardware realizada de manera tradicional utilizando VHDL.

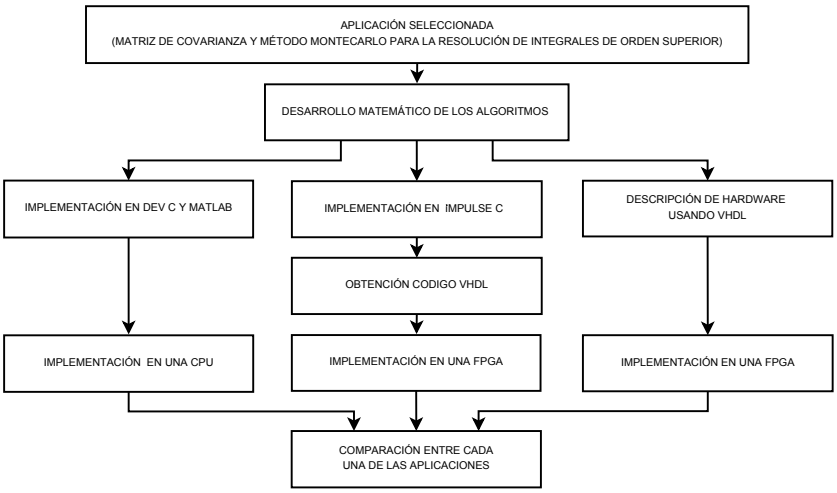


Figura 3.1: Diagrama de la Metodología Aplicada.

3.1 Covarianza

La primera aplicación desarrollada fue el cálculo de la covarianza, la cual es una medida estadística que expresa la relación existente entre dos variables. Se dice que dos variables están relacionadas sí varían de forma conjunta. Esta variación conjunta es positiva si las variables se mueven en la misma dirección una con respecto a la otra, es decir, que los valores bajos de ambas variables están asociados; por otro lado, la variación conjunta es negativa, cuando los valores más altos de una variable se asocian con los más bajos de la otra y una covarianza igual a cero está asociada con la falta de relación lineal entre las variables[10].

La covarianza se representa mediante la Ecuación(3.1)[27].

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(Xi - \overline{X})(Yi - \overline{Y})}{n-1} \tag{3.1}$$

Matriz de Covarianza

La matriz de covarianza es una forma de agrupar los resultados obtenidos al calcular la covarianza a un conjunto de datos de más de dos variables. Para un conjunto de tres variables la matriz de covarianza se representa mediante la Ecuación(3.2)[27].

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix} \tag{3.2}$$

La matriz de covarianza es simétrica respecto a su diagonal por lo tanto $cov(a,b) = cov(b,a)$ y su diagonal representa la varianza de cada variable.

El primer paso realizado para el desarrollo del módulo de covarianza consistió en buscar una ecuación equivalente que disminuyera el tiempo de ejecución. Esto debido a que en la Ecuación (3.1) es necesario hallar primero la media aritmética de X y de Y para poder realizar el resto de operaciones, lo que implica que se deba realizar dos veces el proceso de lectura de los datos. Para tal fin se desarrolla el producto presente en el numerador de la Ecuación(3.1). Mediante la fórmula de la media aritmética se agruparon términos semejantes y después de una serie de procedimientos matemáticos el resultado es la Ecuación (3.3).

$$cov(X,Y) = \frac{n\sum_{i=1}^{n} XiYi - \sum_{i=1}^{n} Xi \sum_{i=1}^{n} Yi}{n(n-1)} \tag{3.3}$$

El segundo paso fue la elaboración de un algoritmo basado en la Ecuación (3.3), este algoritmo se implementó en Dev-C++ y en Matlab. El diagrama de flujo para este algoritmo se muestra en la Figura 3.2.

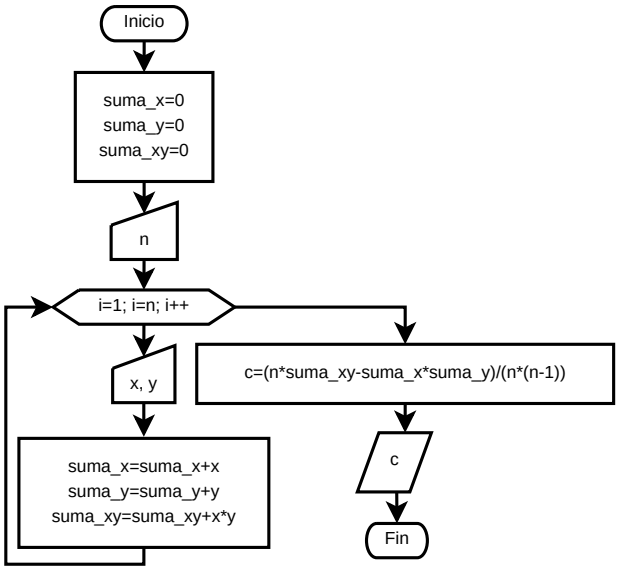


Figura 3.2: Diagrama de Flujo Algoritmo Covarianza.

Una vez implementado el algoritmo en lenguaje C, se comprobó su funcionamiento ingresando un conjunto de tres variables con 100 datos cada una, con el fin de generar la matriz de covarianza. Se midieron los tiempos de ejecución en una plataforma compuesta por un procesador Intel Core I7 a 2.0 GHz con 6 GB de RAM.

Se procedió a desarrollar el algoritmo en Impulse C. Mediante el C to VHDL se realizaron varias versiones preliminares las cuales dependiendo de los resultados parciales obtenidos tuvieron un fin diferente, algunas sólo llegaron a la fase de diseño, otras a la síntesis-simulación y las mejores se procedieron a implementar sobre una FPGA, convirtiéndose en versiones finales.

Para el módulo de covarianza el número de datos es representado por N. Esta variable tiene un tamaño de 8 bits por lo cual es posible tener un conjunto de datos máximo de 255, los valores de entrada se ingresan en formato de punto fijo sin signo *6,10* esto quiere decir que 6 bits son para la parte entera y 10 bits forman la parte fraccionaria, por lo tanto es posible tener un rango de datos entre 0 y 63,99902.

Las variaciones presentes en el código desarrollado en Impulse C con respecto al creado para Dev-C++ y Matlab se presentan en las operaciones aritméticas, esto es debido a que Impulse C [1] necesita hacer uso de ciertos macros para desarrollar las operaciones aritméticas. La multiplicación se realiza por medio de *FXMUL32U(a,b,d)*, *a* y *b* representan los operandos y *d* es la cantidad de bits que representa la parte fraccionaria, para este caso de 10. Para la suma se emplea *FXADD32(a,b,d)*, la resta se realiza con *FXSUB32(a,b,d)*, la división es representada por *FXDIV32(a,b,d)* y el manejo de constantes a un formato equivalente en punto fijo se realiza por medio de *FXCONST32(a,d)*.

Cabe resaltar que los macros aritméticos integrados a Impulse C pueden diferenciarse por el tipo de formato manejado. Para el formato de punto fijo el macro comienza con *FX* seguido de la operación pertinente *MUL, ADD, SUB, DIV* y finalizando con el tamaño en bits para el resultado de la operación. También se debe agregar una *U* al final macro para hacer referencia a que el formato no posee signo. Para el formato entero se puede remitir a la Tabla 3.1.

Tabla 3.1: Tipos de Datos y Macros para Operaciones Aritméticas[1].

Tipos Impulse C	**Modelo**	**Macros de operaciones**
co_int2 - co_int8	int8	IADDn(a,b), ISUBn(a,b), IMULn(a,b), IDIVn(a,b)
co_int9 - co_int16	int16	IADDn(a,b), ISUBn(a,b), IMULn(a,b), IDIVn(a,b)
co_int17 - co_int32	int32	IADDn(a,b), ISUBn(a,b), IMULn(a,b), IDIVn(a,b)
co_int33 - co_int64	int64	IADDn(a,b), ISUBn(a,b), IMULn(a,b), IDIVn(a,b)
co_uint1	uint8	UADDn(a,b), USUBn(a,b), UMULn(a,b), UDIVn(a,b)
co_uint2 - co_uint8	uint8	UADDn(a,b), USUBn(a,b), UMULn(a,b), UDIVn(a,b)
co_uint9 - co_uint16	uint16	UADDn(a,b), USUBn(a,b), UMULn(a,b), UDIVn(a,b)
co_uint17 - co_uint32	uint32	UADDn(a,b), USUBn(a,b), UMULn(a,b), UDIVn(a,b)
co_uint33 - co_uint64	uint64	UADDn(a,b), USUBn(a,b), UMULn(a,b), UDIVn(a,b)

La primera versión preliminar generada en Impulse C se caracterizó por hacer uso de la sentencia *if* y un contador en lugar del *for* presente en el algoritmo de la Figura 3.2. Con esta configuración no se obtienen buenos resultados en comparación a los obtenidos al ejecutar el algoritmo en una plataforma de computo CPU, por lo tanto se generó otra versión respetando el algoritmo planteado en la Figura 3.2. Esta versión presentó problemas en el tiempo de lectura de datos ya que la división castiga el proceso en su totalidad, limitando los momentos en los cuales es posible leer datos ya que sólo será posible hacerlo cuando la división se realiza al final de cada calculó de las covarianzas que conforman la matriz. El tiempo de ejecución de esta operación en Impulse C como para las otras operaciones que son posibles de implementar mediante el C to VHDL están regidas por la Tabla 3.2.

Tabla 3.2: Unidad de Retardo para Operaciones: Aritméticas, Lógicas, Relacionares y bit a bit[1].

Tipo de Operador	Operadores	Unidad de Retardo (Ciclos de Reloj)
Bit a Bit	& ^ >> << ~	1
Bit a Bit Asignación	&= |= ^=	1
Aritmético	+ − ∗ / %	Ancho de bit del mayor operando
Aritmético Asignación	+= −= ∗= /= %= ++ −−	Ancho de bit del mayor operando
Aritmético (unitario)	+ −	Ancho de bit del mayor operando
Lógico	&& || !	Ancho de bit del mayor operando
Relacional	< > <= >= == !=	Ancho de bit del mayor operando

Para poder realizar el proceso de lectura de datos sin necesidad que la división haya terminado es necesario separar esta operación del resto del algoritmo. Para tal fin se crea un nuevo proceso en Impulse C encargado de realizar sólo la división y mediante VHDL se unen los dos módulos o *cores* generados por Impulse C. Cabe aclarar que este proceso de unión también es posible de realizar mediante Impulse C, pero por facilidad se realiza en la herramienta encargada de implementar los *cores* en la FPGA (ISE10.1).

El esquema generado para los dos *cores* en Impulse C se muestra en la Figura 3.3. Para estos *cores* el tamaño de las FIFO para las entradas X e Y es tres, debido a que este es el valor mínimo requerido por ellas para garantizar la captura datos cada ciclo de reloj. El resto de FIFOs para las demás entradas y salidas es uno, porque estas solo se usan al principio y al final del proceso. El bloque *covar* es un procesador de propósito específico encargado de capturar los datos y realizar gran parte del cálculo. Sus salidas son el numerador y el denominador para la división así como el signo del resultado de la covarianza.

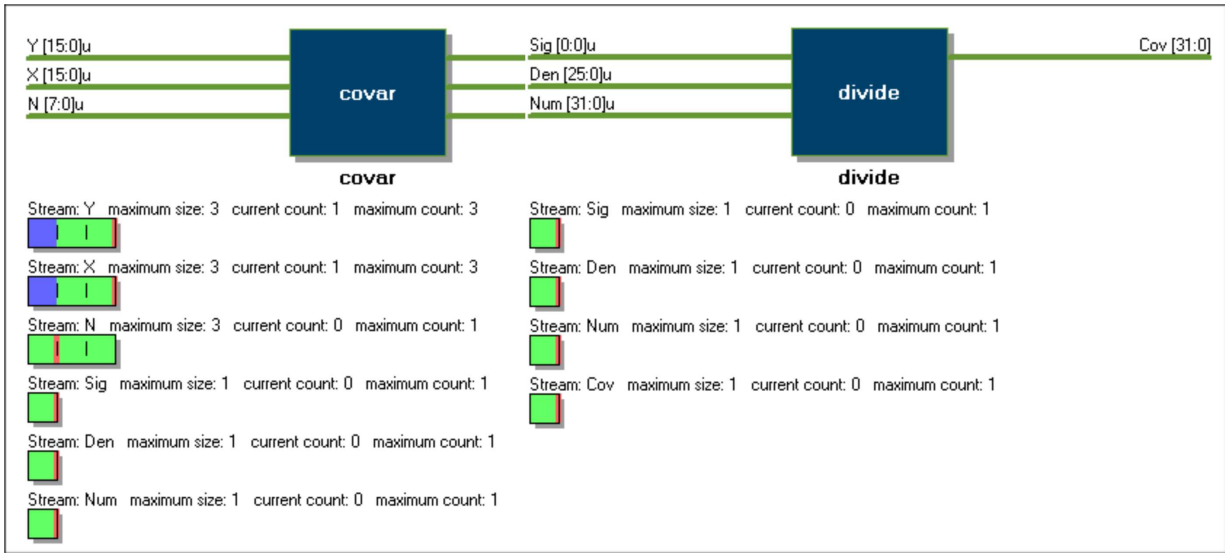


Figura 3.3: Esquema de *Cores* en Impulse C Versión Uno.

Otra forma para calcular la covarianza implementada en Impulse C se muestra en el esquema de la Figura 3.4. El procesador *covar* está encargado de leer los datos y realizar el proceso de acumulación, es decir, realiza sólo la parte del algoritmo encerrada por el *for* de la Figura 3.2, mientras que el *core divide* está encargado del resto de operaciones del algoritmo, esta modificación se realizó buscando aumentar la velocidad de lectura de los datos.

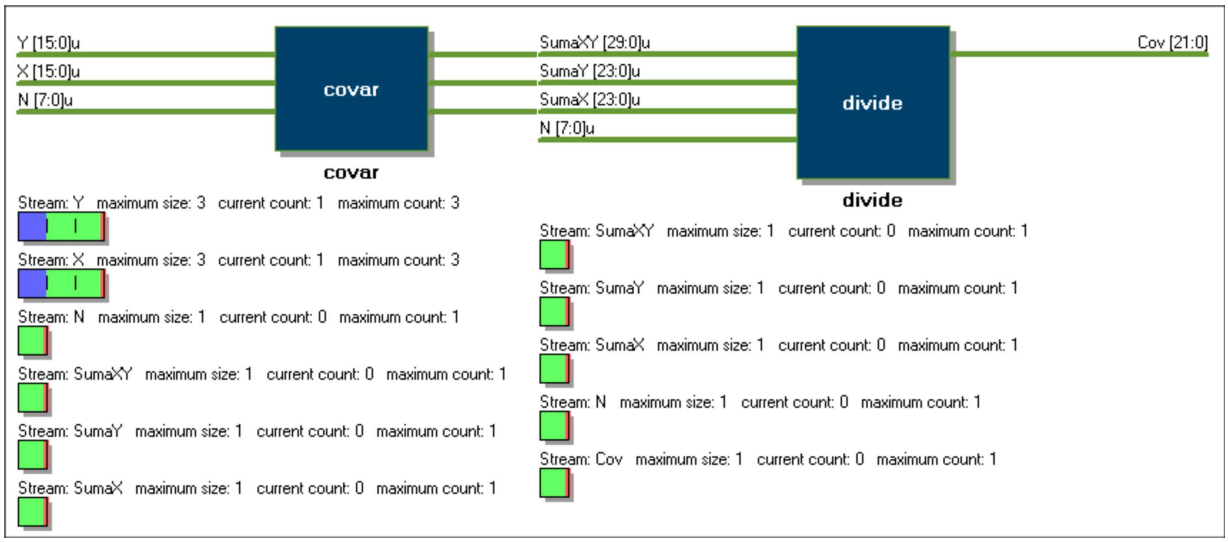


Figura 3.4: Esquema de *Cores* en Impulse C Versión Dos.

El último paso, consistió en realizar el módulo de covarianza de forma manual mediante la descripción en VHDL, usando como base la metodología Fajardo-Ramon[28], en la cual se diseña un procesador de propósito específico haciendo uso de un *Datapath* controlado por una máquina de estados (FSM), técnica que se conoce como *Finite State Machine With Datapath* (FSMD)[29].

El objetivo de este diseño digital es realizar el cálculo de la covarianza mediante el algoritmo presente en la Figura 3.2. El primer inconveniente que se presenta es la realización de la división presente en la parte final del diagrama, debido a que la división no es una operación que se encuentre en la librería aritmética del lenguaje de descripción de hardware utilizado, por ende el primer paso fue realizar un procesador de propósito específico que calcula la división de dos números en formato de punto fijo sin signo cuya parte entera está representada por 11 bits y la fraccionaria por 10 bits. El diagrama ASM[29] para la división se muestra en la Figura 3.5.

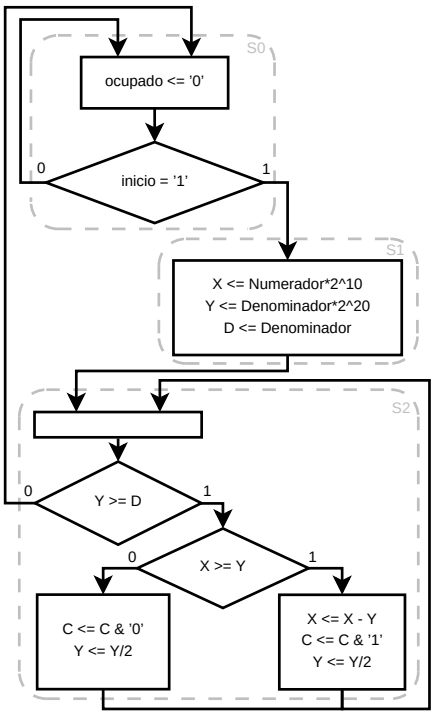


Figura 3.5: Diagrama ASM para la División.

El *datapath* es diseñado para la división mediante el diagrama ASM, también brinda información sobre la máquina de estados a implementar. Para el caso de la división la FSM cuenta con tres estados. En la Figura 3.6 se aprecia la FSMD. El bloque de la parte izquierda representa la máquina de estados y el de la derecha el *datapath*.

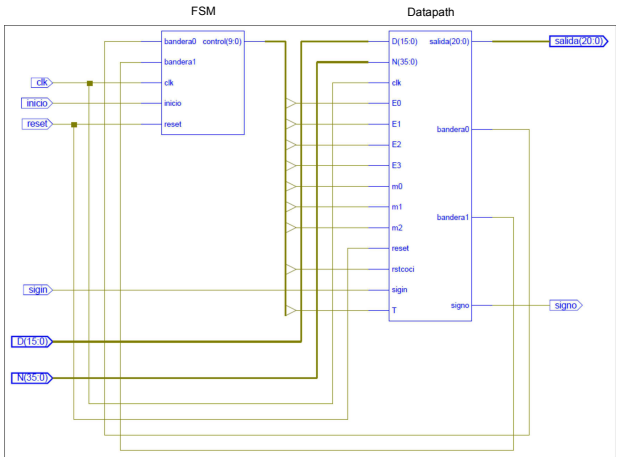


Figura 3.6: Procesador de Propósito Específico para la División (FSMD).

Luego se desarrolló el procesador *Covar* encargado de leer los datos y calcular el numerador y denominador que después serán enviados al módulo de la división. El diagrama ASM para el *core* Covar se muestra en la Figura 3.7.

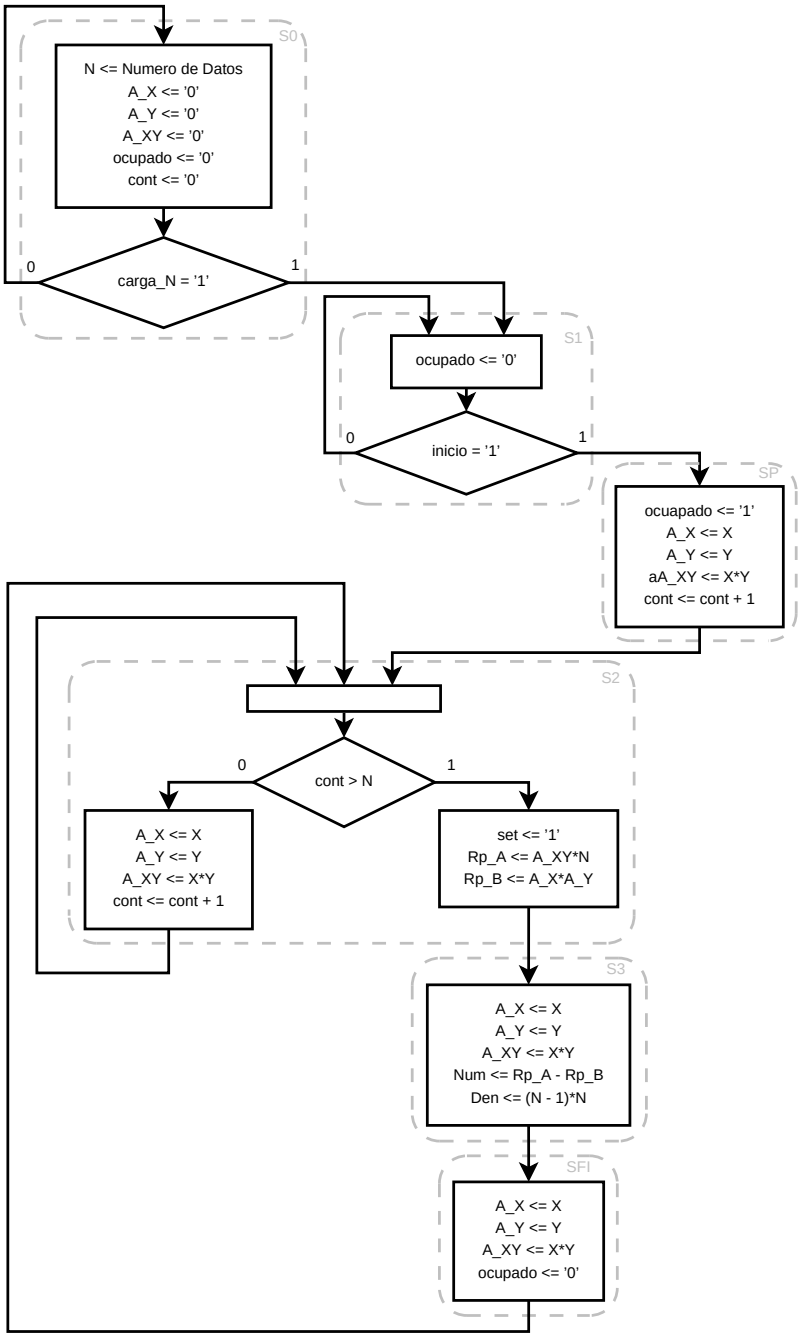


Figura 3.7: Diagrama ASM para el *core* Covar.

La máquina de estados para el *core* covar cuenta con seis estados, el conjunto de FSM y *Datapath* se aprecian en la Figura 3.8.

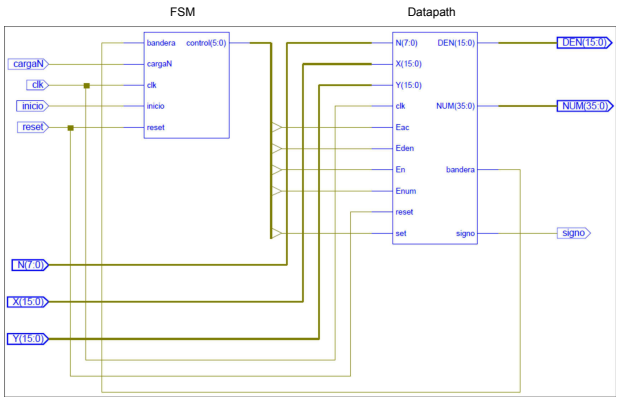


Figura 3.8: Procesador de Propósito Específico Covar (FSMD).

3.2 Montecarlo

La siguiente aplicación que se desarrolló fue la implementación del método Montecarlo para la resolución de integrales numéricas de orden superior. Lo primero que se debe tener en cuenta cuando se hace referencia al método Montecarlo es que se está hablando de una serie de técnicas cuantitativas de naturaleza estocástica, utilizadas para obtener soluciones rápidas y aproximadas de problemas complejos ya sean estos de naturaleza determinística o aleatoria. El método fue desarrollado por los científicos Stanislaw Ullan y John Von Newman alrededor de 1944, durante el desarrollo de investigaciones relacionadas con la creación de la bomba atómica, pero sólo fue hasta 1948, con el trabajo de Harris y Herman Kahn, que el método obtuvo un desarrollo sistemático[30]. Las técnicas de Montecarlo reciben su nombre de un famoso casino ubicado en el principado de Mónaco y que es considerado como un símbolo de los juegos de azar.[31]

Si bien es cierto que el método se puede aplicar a una integral de orden uno, no resulta eficiente debido a que requiere la generación de una cierta cantidad de valores aleatorios para evaluar la función, además existen otros métodos que convergen al resultado de forma más rápida. Pero cuando se habla de integrar múltiples dimensiones, Montecarlo, demuestra ser un método bastante útil, esto se debe a que para cualquier valor de iteración el error absoluto será inversamente proporcional a la raíz cuadrada de la cantidad de iteraciones realizadas, ver Ecuación (3.4), siendo independiente de la cantidad de dimensiones que se estén evaluando, cosa que no ocurre en otros métodos (Trapezoidal, Simpson, Gaussiana) en los cuales el error aumenta a medida que aumentan la cantidad de dimensiones [32].

$$error\_absoluto = \frac{1}{\sqrt{N}} \tag{3.4}$$

La aplicación del método se mostrará sobre una integral de orden uno y se extrapolará para las integrales de orden dos. Lo primero que debe hacerse es llevar la integral a un intervalo entre cero y uno como se puede apreciar en al Ecuación (3.5)[11].

$$\int_a^b G(x)\,dx \approx \int_0^1 G((a+(b-a)u))\,du \tag{3.5}$$

Una vez la integral se encuentra en el intervalo deseado y aplicando la ley de los grandes números [33] se obtiene como resultado la Ecuación (3.6); el término u_i representa una serie de números de carácter aleatorio con los cuales se evaluará la función tantas veces como el usuario lo determine, siendo M la cantidad de iteraciones a realizar.

$$\int_0^1 G((a+(b-a)u))\,du \approx \frac{(b-a)}{M}\sum_{i=1}^{M} G(a+(b-a)u_i) \tag{3.6}$$

Generación de Números Pseudo-aleatorios

Se puede decir que el corazón del método Montecarlo son los números aleatorios, pero dada la imposibilidad de generarlos en software se hace uso de métodos deterministas para obtener números que reciben el nombre de pseudo-aleatorios y aunque la mayoría de los programas de desarrollo de software poseen dentro de sus librerías y aplicaciones la función de generar dichos números, estas librerías no están disponibles en VHDL, razón por la cual se buscó un método para generarlos. Los números generados en la secuencia deben poseer las siguientes características para ser considerados como pseudo-aleatorios[34]:

- Estar uniformemente distribuidos.
- Ser reproducibles.
- Ser estadísticamente independientes.
- Poseer un periodo largo.
- Ser generados por métodos rápidos.
- No consumir demasiados recursos computacionales.

Dentro de las múltiples opciones que se evaluaron para desarrollar un generador de números pseudo-aleatorios, estaban el método de generación de Von Newman o medios cuadrados[35], el método aditivo de Fibonacci [36], la generación mediante *linear feedback shift register*(LFSR)[36], pero por último, se tomó la decisión de hacer uso del método congruencial aditivo de Green [35], debido a que este método cumple con varios de los criterios de prueba que debe satisfacer la secuencia de números generada para que sea considerada pseudoaleatoria, como se puede observar en la Tabla 3.3 [2].

Tabla 3.3: Resultado de los Test de Aleatoriedad[2].

Gen.\Test	**Monobit**	**Poker**	**Run**	**Long Run**	**Chi$\hat{2}$**
G.L.C. Multiplicativo	NO	NO	NO	NO	NO
G.L.C. Mixto	NO	NO	NO	SI	NO
Cuadrático	NO	NO	NO	SI	SI
Fibonacci	NO	NO	NO	SI	NO
Green	SI	SI	NO	SI	SI
Mitchel-Moore	SI	NO	NO	SI	SI
Inversa	NO	NO	NO	SI	NO
Método Mixto	SI	NO	NO	SI	SI
Registro y XOR	SI	NO	NO	SI	SI

En la Ecuación (3.7) se puede apreciar la manera en la que se van generando los números a partir de ciertos valores semilla almacenados en un vector de 17 elementos cantidad de datos necesaria para que el método de generación de Green cumpla con las características anteriormente expuestas[35].

$$X_{(n+1))} = [X_{(n)} + X_{(n-16)}] mod A \tag{3.7}$$

La Ecuación (3.7) y la Ecuación (3.8) son equivalentes y esta última fue la que se empleó en el desarrollo del método de generación de Green. Esto debido a que la función MOD presente en la Ecuacion (3.7) no es sintetizable en la herramienta ISE, porque no está presente en sus librerías.

$$X_{(n+1))} = [X_{(n)} + X_{(n-16)}] - K * A \tag{3.8}$$

Donde

$$K = \begin{cases} 1 & \text{si} \quad (X_{(n)} + X_{(n-16)}) > A \\ 0 & \text{si} \quad (X_{(n)} + X_{(n-16)}) \leq A \end{cases}$$

El siguiente paso fue seleccionar la integral mostrada en la Ecuación (3.9), para llevar a cabo la implementación del método:

$$F(x,y) = \int_0^1 \int_0^1 X^2 + Y^2 dxdy = \frac{2}{3} \tag{3.9}$$

Por último, los números están representados en un formato de punto fijo *0,16* y representan valores que están ubicados en un rango de 0 a 0.99998, dado que este es el intervalo que se debe evaluar cuando se está usando el método de Montecarlo.

A continuación, con base en la Ecuación (3.6) y siguiendo la metodología propuesta en la Figura 3.1 se desarrolló una implementación en software para determinar el comportamiento del método al ser evaluado en un computador. Se usó el algoritmo que se observa en la Figura 3.9.

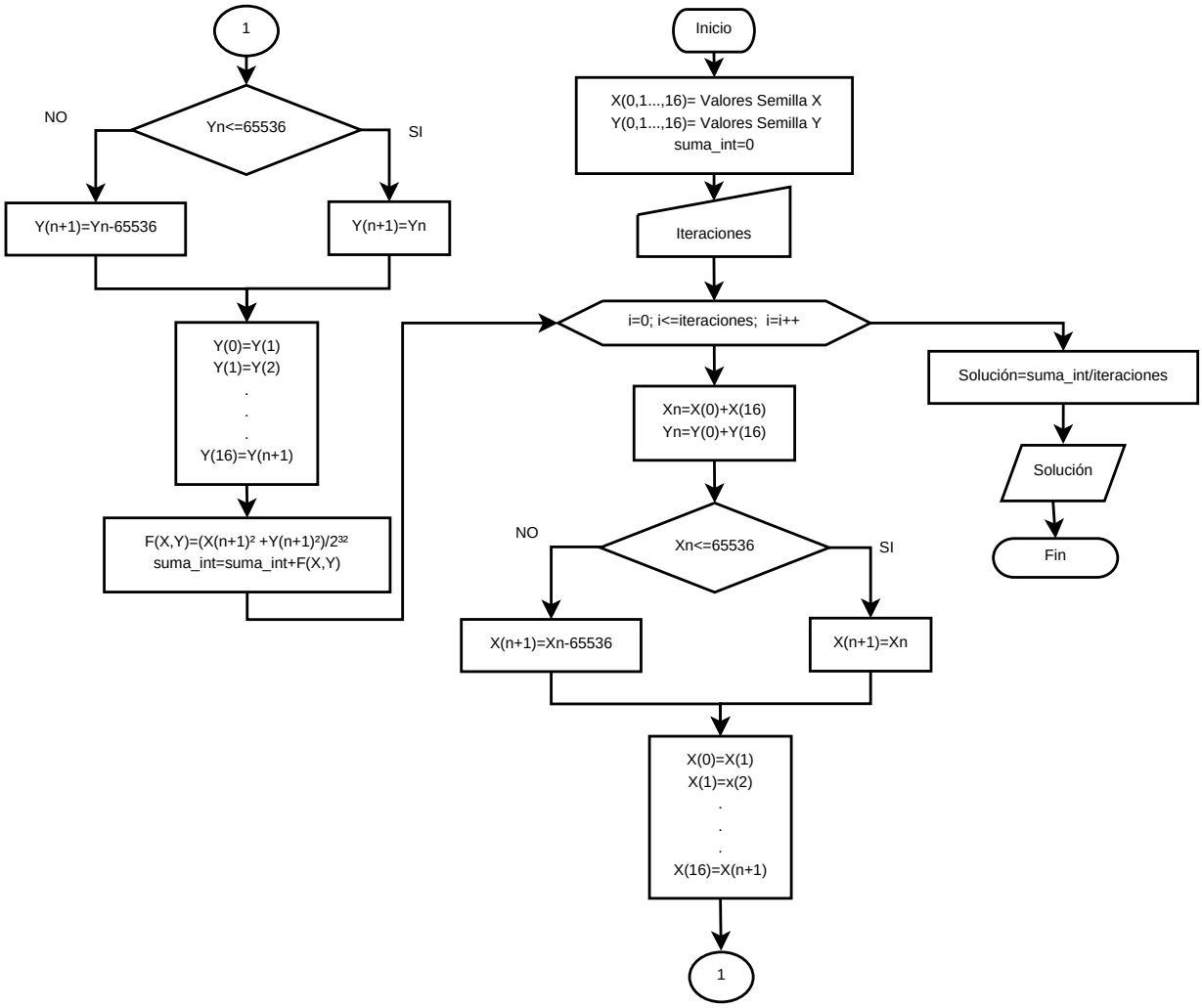


Figura 3.9: Diagrama de Flujo del Algoritmo Montecarlo para Resolver Integrales de Orden Superior.

Una vez desarrollado el algoritmo en software se hace uso de la herramienta Impulse C con la cual se compiló el programa desarrollado en lenguaje C del método de integración de Montecarlo. Las versiones preliminares para este método contaban con dos vectores semilla de 17 posiciones cada uno y mediante un *for* se realizaba el cálculo y movimiento correspondiente para el generador de Green encargado de generar las parejas X,Y que serán evaluadas en el método. Los resultados con esta estrategia no fueron los mejores debido a que el uso de vectores en los cuales se tenga múltiple acceso a ciertas posiciones limita fuertemente el rendimiento del proceso desarrollado en Impulse C. Con el fin de evitar esta limitación se cambiaron los vectores por un conjunto de 17 variables para cada vector.

Se desarrollaron tres versiones finales teniendo en cuenta los resultados preliminares para mejorar el calculó de la integral doble. La primera versión está basada en la Ecuación (3.8), la segunda versión usa la función *mod* incorporada en Impulse C con el fin de implementar la Ecuación (3.7). En la Figura 3.10 se muestra el *core* generado para un módulo Montecarlo para las versiones uno y dos.

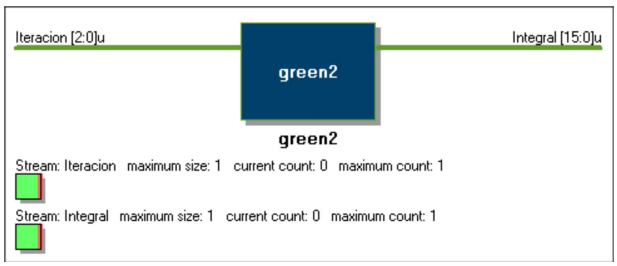


Figura 3.10: *Core* generado en Impulse C para el Método Montecarlo.

En la parte izquierda de la Figura 3.10 se encuentra la entrada, la cual toma valores de 0 a 7 que corresponden a valores de iteracion que van desde 2^{10} hasta 2^{17}.

Por medio de Impulse C se crearon cuatro módulos como el mostrado en la Figura 3.10, cuya única diferencia radica en las semillas para el método de Green, esto con el fin de paralelizar los procesos. Los cuatro módulos generan 4 salidas que serán promediadas. Para esto es necesario crear un proceso encargado de sumar los resultados parciales de los cuatro módulos y promediarlos con el fin de obtener el valor de la integral de forma paralelizada. Para ello se implementó dicho sumador promedio en Impulse C. El esquema del *core* generado se muestra en la Figura 3.11.

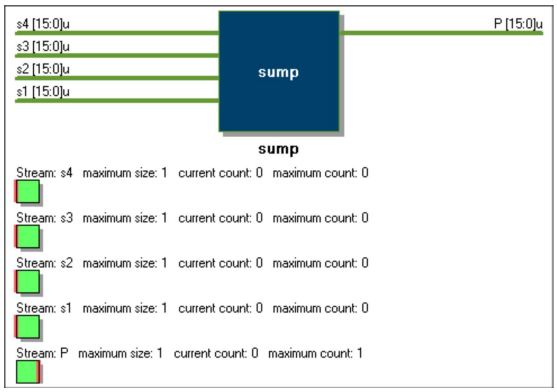


Figura 3.11: Sumador Promedio para cuatro Módulos Montecarlo.

La última versión realizada en Impulse C consiste en unir dos generadores de Green, cada uno generando una pareja X,Y y mediante las combinaciones $(x1, y1), (x2, y2), (x1, y2), (x2, y1)$ es posible obtener 4 valores de la integral los cuales se suman y se obtiene el valor promedio.

El esquema propuesto en la Figura 3.12, muestra el procedimiento que se empleó para llevar a cabo la aplicación en VHDL de los módulos que conforman la implementación, los cuales son: un generador de números pseudo-aleatorios (X,Y), un módulo Función, donde se evalúa la función a integrar y por último un módulo llamado Solución, que realiza la sumatoria de los valores obtenidos en el módulo Función y que dependiendo de la cantidad de iteraciones que se realizaron arroja un resultado. Estos módulos están controlados por una FSM que genera las señales que controlan el procesador de propósito específico anteriormente descrito. El resultado obtenido puede ser apreciado en las Figuras (3.13 y 3.14).

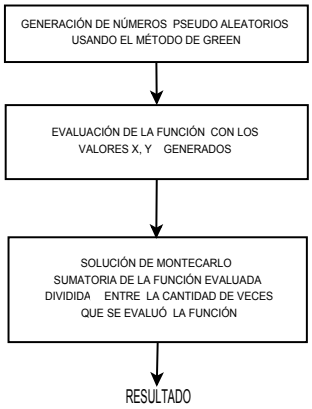


Figura 3.12: Procedimiento para Implementar el Método Montecarlo en la Resolución de Integrales de Orden Superior.

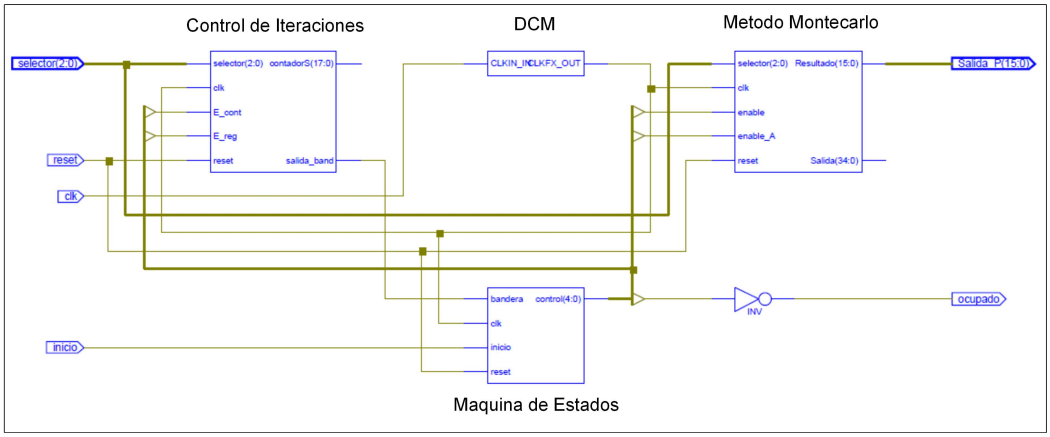


Figura 3.13: Módulos del Procesador de Propósito Específico que Resuelve una Integral de Orden Dos Mediante el Método Montecarlo.

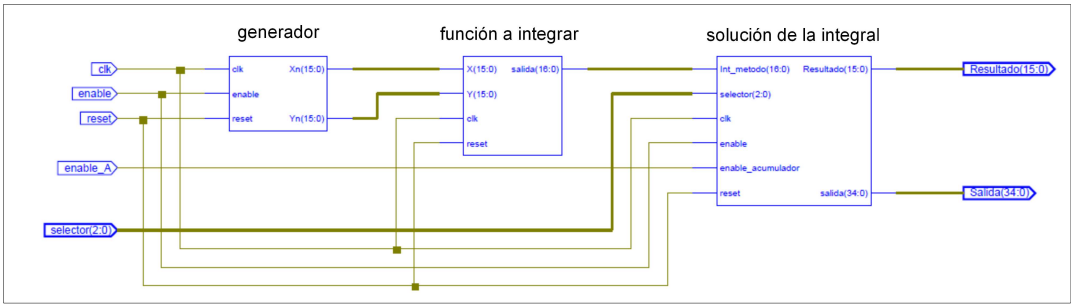


Figura 3.14: Módulos que Conforman el Método Montecarlo.

Capítulo 4

Resultados

Las plataformas de cómputo donde se implementaron los algoritmos son un computador portátil con procesador Intel Core I7 a 2.0 GHz con 6 GB de RAM y una FPGA Spartan-3AN con referencia 3S700ANFGG484-4. Los *cores* se implementaron en la FPGA mediante ISE10.1 usando una configuración de optimización para velocidad y los datos generados por la FPGA son capturados mediante ChipScope[13].

4.1 Resultados Módulo Covarianza

El conjunto de datos con los cuales se calculó la matriz de covarianza están compuestos por tres vectores con tamaño de 100 datos generados de forma aleatoria en un rango de 0 a 63,99902. Estos vectores son almacenados para ser utilizados como datos de entrada para el algoritmo en Matlab y las implementaciones en la FPGA.

Datos Obtenidos

En la Figura 4.1 se representa la matriz triangular superior de la matriz de covarianza calculada mediante Matlab en formato de punto flotante de doble precisión.

```
ans =

  330.9912  -11.6685  -48.7331
         0  359.6076   10.9436
         0         0  329.6602

Elapsed time is 0.000162 seconds.
```

Figura 4.1: Matriz de Covarianza Función en Matlab corriendo sobre un procesador I7 a 2 GHz.

En Matlab también se trabajó con el formato de precisión sencilla pero el rendimiento con este formato decrecía con respecto al formato de doble precisión al ser ejecutado el algoritmo sobre el procesador Intel Core I7 por lo cual estos datos no se tomaron en cuenta para las comparaciones.

Para tener una comparación visual con los procesos realizados mediante Impulse C y la descripción manual, los resultados de la matriz de covarianza en Matlab son convertidos a formato de punto fijo como se muestra en la Figura 4.2.

```
>> floor(ans*2^10)

ans =

      338934      -11949      -49903
           0      368238       11206
           0           0      337572
```

Figura 4.2: Matriz de Covarianza en Formato Punto Fijo.

El cálculo de la matriz de covarianza en la FPGA para un módulo covarianza, se realizó de forma serial de la siguiente manera: cov(x,x), cov(x,y), cov(x,z), cov(y,y), cov(y,z), cov(z,z). El resto de valores no se calculan debido a la simetría de la matriz de covarianza, como también se puede apreciar en los resultados de Matlab. En las Figuras(4.3, 4.4, 4.5, 4.6) se muestran los resultados arrojados por la FPGA mediante ChipScope al implementar un módulo covarianza de las diferentes versiones realizadas mediante Impulse C y la realizada de forma manual.

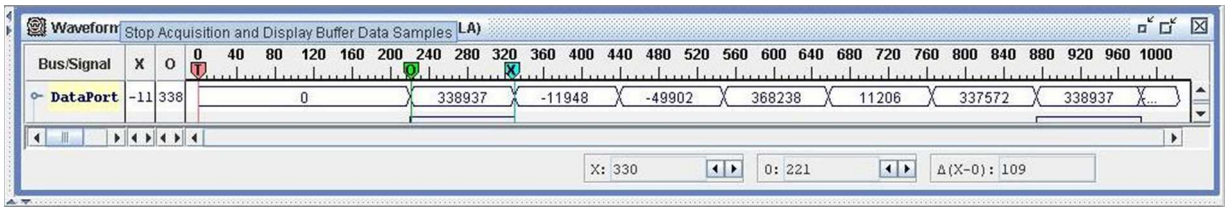


Figura 4.3: Matriz de Covarianza Impulse C con un Módulo Covarianza V1.0.

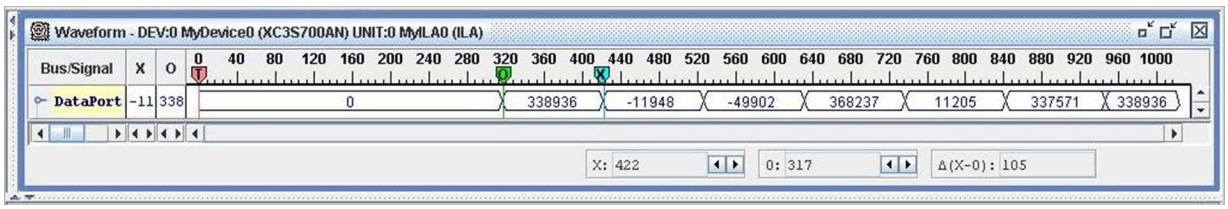


Figura 4.4: Matriz de Covarianza Impulse C con un Módulo Covarianza V2.0.

Mediante Impulse C se habilita la opción de usar doble reloj para la versión 2.0 del módulo de covarianza,

de modo que los cores de *covar* y *divide* trabajen a sus máximas frecuencias generando la versión 2.1. Los resultados se muestran en la Figura 4.5.

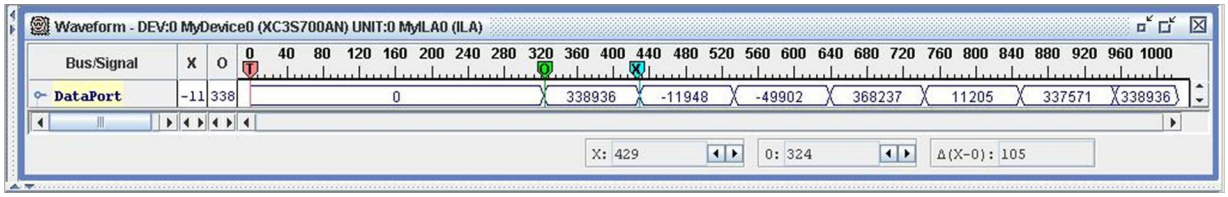


Figura 4.5: Matriz de Covarianza Impulse C con un Módulo Covarianza V2.1.

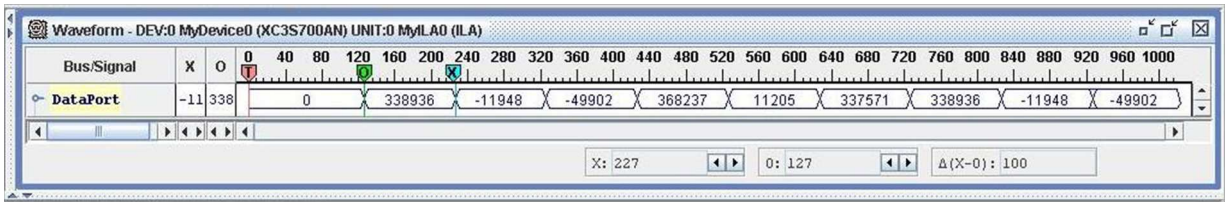


Figura 4.6: Matriz de Covarianza Método Manual con un Módulo Covarianza.

En la Tabla 4.1 se evidencian los datos obtenidos para la matriz de covarianza arrojados por Matlab, Impulse C y la descripción manual en VHDL para un módulo de covarianza sobre la FPGA. Los datos obtenidos de la FPGA son convertidos a formato decimal y junto a ellos se muestran los errores relativos, tomando como valor verdadero el arrojado por Matlab en formato de punto flotante de doble precisión.

Tabla 4.1: Datos en Formato Decimal y Errores con un Módulo Covarianza.

Un Core	**Matlab Dato**	**Impulse V1**		**Impulse V2 y V2.1**		**Manual**	
		Dato	**% Error**	**Dato**	**% Error**	**Dato**	**% Error**
cov(x,x)	330,99116	330,99316	0,00061	330,99219	0,00031	330,99219	0,00031
cov(x,y)	-11,66853	-11,66797	-0,00478	-11,66797	-0,00478	-11,66797	-0,00478
cov(x,z)	-48,73313	-48,73242	-0,00146	-48,73242	-0,00146	-48,73242	-0,00146
cov(y,y)	359,60764	359,60742	-0,00006	359,60645	-0,00033	359,60645	-0,00033
cov(y,z)	10,94356	10,94336	-0,00184	10,94238	-0,01076	10,94238	-0,01076
cov(z,z)	329,66024	329,66016	-0,00003	329,65918	-0,00032	329,65918	-0,00032

Una vez realizada la implementación del cálculo de la matriz de covarianza mediante un módulo de covarianza se procedió a paralelizar el cálculo de la matriz de covarianza. Con este fin, se implementaron tres módulos de covarianza sobre la FPGA, cada módulo ésta encargado de calcular una fila de la matriz.

En las Figuras(4.7, 4.8, 4.9, 4.10), se muestran los resultados obtenidos al implementar tres módulos de covarianza para cada versión realizada en Impulse C y de forma manual.

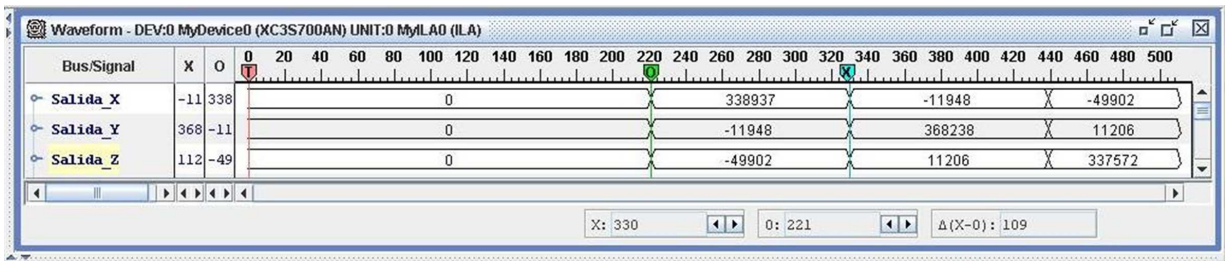


Figura 4.7: Matriz de Covarianza Impulse C con tres Módulos Covarianza V1.0.

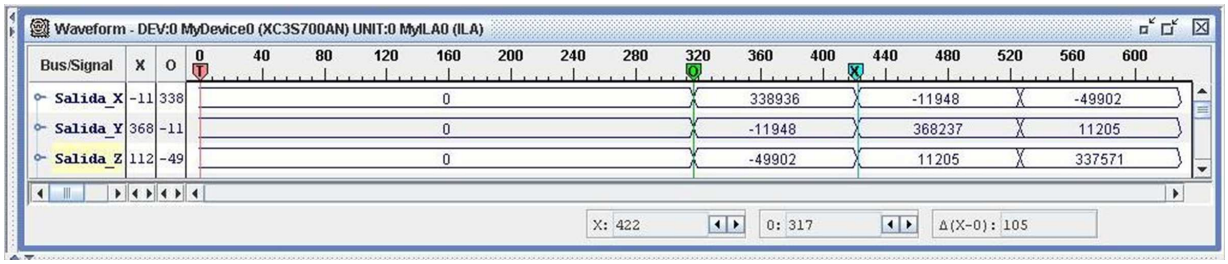


Figura 4.8: Matriz de Covarianza Impulse C con tres Módulos Covarianza V2.0.

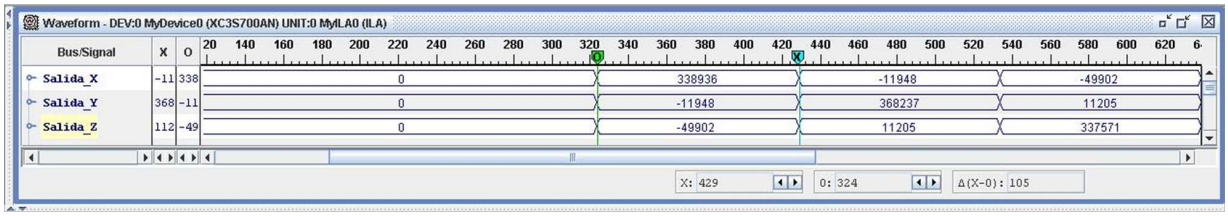


Figura 4.9: Matriz de Covarianza Impulse C con tres Módulos Covarianza V2.1.

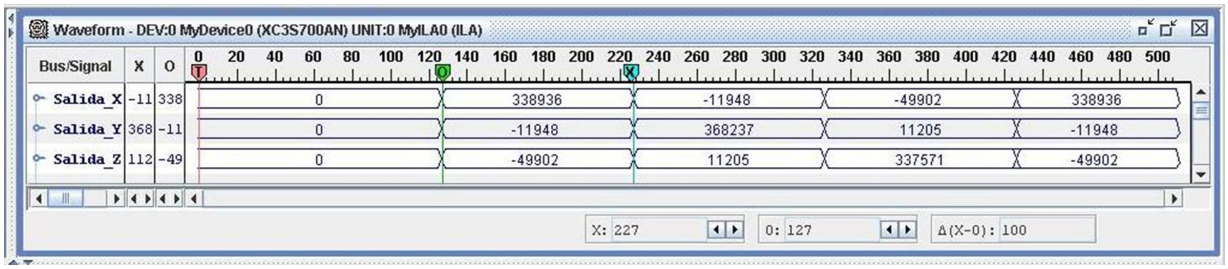


Figura 4.10: Matriz de Covarianza Método Manual con tres Módulos Covarianza.

Los resultados y sus errores son los mismos que se muestran en la Tabla 4.1.

Tiempos de Ejecución

La Tabla 4.2 evidencia los tiempos de ejecución para el cálculo de la matriz de covarianza mediante Matlab en un computador, así como la latencia, tiempo en ciclos de reloj, frecuencia y tiempo en segundos de las implemetaciones sobre una FPGA realizadas en Impulse C y de forma manual para un módulo covarianza.

Con base en estos resultados, se calcula la aceleración que se obtiene mediante la implementación en hardware, este cálculo se realizó dividiendo el tiempo de ejecución del algoritmo en el computador entre el tiempo de ejecución sobre la FPGA.

Tabla 4.2: Tiempos de Ejecución para un Módulo Covarianza.

Tiempos de Ejecución	**Matlab**	**Impulse V1**	**Impulse V2.0**	**Impulse V2.1**	**Manual**
Latencia [ciclos de reloj]	-	221	317	324	127
Tiempo [ciclos de reloj]	-	766	842	849	627
Frecuencia [MHz]	2000	70	80	150	75
Tiempo [μs]	162	10,94	10,53	5,66	8,36
Aceleración	1	14,80	15,39	28,62	19,38

En la Tabla 4.3, se muestran los tiempos de ejecución para el cálculo de la matriz de covarianza usando tres módulos de covarianza para las implementaciones sobre la FPGA.

Tabla 4.3: Tiempos de Ejecución para tres Módulos Covarianza.

Tiempos de Ejecución	**Matlab**	**Impulse V1**	**Impulse V2.0**	**Impulse V2.1**	**Manual**
Latencia [ciclos de reloj]	-	221	317	324	127
Tiempo [ciclos de reloj]	-	439	527	534	327
Frecuencia [MHz]	2000	70	80	150	75
Tiempo [μs]	162	6,27	6,59	3,56	4,36
Aceleración	1	25,83	24,59	45,51	37,16

Recursos de Hardware

En la Tabla 4.4 se encuentran los recursos utilizados de la FPGA al implementar las diferentes versiones realizadas mediante Impulse C y de forma manual para un módulo covarianza.

Tabla 4.4: Recursos utilizados en la FPGA para un Módulo Covarianza.

Recursos FPGA	Total FPGA	Impulse V1		Impulse V2.0		Impulse V2.1		Manual	
		Recursos	% Total	Recursos	% Total	Recursos	% Total	Recursos	% Total
Slices	5888	975	16,56	780	13,25	972	16,51	579	9,83
Flip-Flops	11776	842	7,15	613	5,21	813	6,90	402	3,41
LUTs	11776	1906	16,19	1736	14,74	1881	15,97	950	8,07
MULT18X18	20	11	55,00	11	55,00	8	40,00	8	40,00
DCMs	8	1	12,50	1	12,50	2	25,00	1	12,50

Los recursos utilizados en la FPGA para la implementación de los tres módulos de covarianza para las diferentes versiones en Impulse C así como la desarrollada de forma manual, se muestran en la Tabla 4.5.

Tabla 4.5: Recursos utilizados en la FPGA para tres Módulos Covarianza.

Recursos FPGA	Total FPGA	Impulse V1		Impulse V2.0		Impulse V2.1		Manual	
		Recursos	% Total	Recursos	% Total	Recursos	% Total	Recursos	% Total
Slices	5888	3105	52,73	2426	41,20	2885	49,00	1439	24,44
Flip-Flops	11776	2601	22,09	1908	16,20	2565	21,78	1147	9,74
LUTs	11776	6078	51,61	5360	45,52	5518	46,86	2287	19,42
MULT18X18	20	18	90,00	18	90,00	18	90,00	18	90,00
DCMs	8	1	12,50	1	12,50	2	25,00	1	12,50

Tiempo de Diseño

Para el tiempo de diseño sólo se tomó en cuenta el necesario para desarrollar la aplicación, más no el tiempo requerido para aprender a usar la herramienta de compilación Impulse C. En este caso, el tiempo de aprendizaje fue de más o menos 8 semanas, en las cuales se realizaron múltiples ejemplos y cuando se adquirió un conocimiento apropiado del comportamiento de la herramienta se procedió a realizar la aplicación para su evaluación. En la Tabla 4.6 se muestra el tiempo empleado para desarrollar el módulo de covarianza para las metodologías anteriormente expuesta. El tiempo esta dado en semana, cada semana contempla 5 días de 8 horas laborales.

Tabla 4.6: Tiempo de Desarrollo Módulo Covarianza.

Tiempo de Desarrollo	Matlab	Impulse C	Manual
Tiempo en Semanas	0,5*	1,5**	6***

* 1. Primera Función (Ecuación (3.1)): un día.

2. Segunda Función (Ecuación (3.3)): un día.

3. Ejecución en punto flotante de precisión doble y simple: un día.

** 1. Versiones preliminares: 3 días.

2. Core *covar* V1: un día.

3. Core *divide* V1: un día.

4. Core *covar* V2.0: un día.

5. Core *divide* V2.0: un día.

6. Core *covar* V2.1: medio día.

7. Core *divide* V2.1: medio día.

*** 1. Procesador de división: 2 semanas.

2. Segmentación y mejoras al procesador división: 1 semana.

3. Procesador *Covar*: 1 semana.

4. Segmentación y mejoras del procesador *Covar*: 1 semana.

5. Maquina de control para la unión de los dos procesadores, sincronización: 1 semana

4.2 Resultados Módulo Montecarlo

Datos Obtenidos

A continuación se podrán observar los resultados obtenidos para la solución numérica de integrales de orden superior usando el método Montecarlo. En las Figuras(4.11, 4.12) se muestran los resultados en ChipScope para la solución de la integral mediante Impulse C y de forma manual para un sólo módulo de integral con 1024 iteraciones. Dado que es un método iterativo se estimó un porcentaje de error o tolerancia de 2% para asumir que los resultados son acertados.

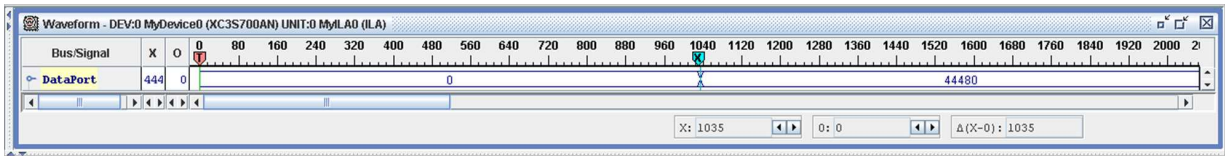


Figura 4.11: Metodo Montecarlo en Impulse C Versiones Uno y Dos para un Módulo.

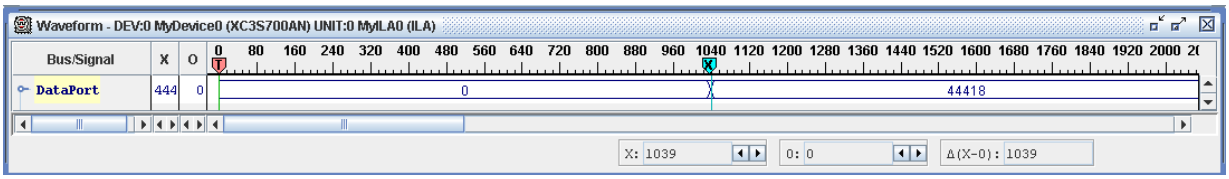


Figura 4.12: Metódo Montecarlo de forma Manual con un Módulo.

En la Tabla 4.7 se pueden observar los resultados obtenidos al implementar un módulo Montecarlo en Matlab, Impulse C y descrito de manera tradicional, además se puede apreciar el porcentaje de error que presentan las implementaciones propuestas con respecto al valor real de la integral.

Tabla 4.7: Resultados de la Integral en Formato Punto Fijo y Decimal con un Módulo Montecarlo.

Iteraciones	Matlab		Impulse V1 y V2			Manual		
	Dato	% Error	Dato PF	Dato	% Error	Dato PF	Dato	% Error
1024	0,67763	1,6447	44408	0,67761	1,6418	44418	0,67776	1,6647
2048	0,66687	0,0300	43702	0,66684	0,0259	43656	0,66614	-0,0793
4096	0,66782	0,1735	43765	0,66780	0,1701	43533	0,66426	-0,3609
8192	0,67309	0,9639	44110	0,67307	0,9598	43626	0,66568	-0,1480
16384	0,67137	0,7049	43993	0,67128	0,6920	43981	0,67110	0,6645
32768	0,66711	0,0661	43716	0,66705	0,0580	43666	0,66629	-0,0565
65536	0,66829	0,2429	43793	0,66823	0,2342	43657	0,66615	-0,0771
131072	0,66669	0,0039	43689	0,66664	-0,0038	43680	0,66650	-0,0244

En las Figuras(4.13, 4.14, 4.15) se muestran los resultados en ChipScope para la solución de la integral mediante Impulse C y de forma manual para cuatro módulos de integral con 1024 iteraciones.

Figura 4.13: Metodo Montecarlo en Impulse C Versiones Uno y Dos para cuatro Módulos.

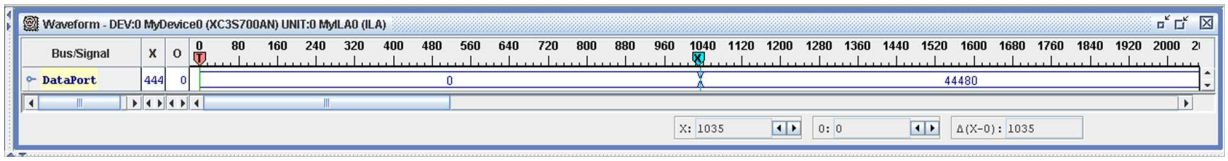


Figura 4.14: Metodo Montecarlo en Impulse C Versión Tres para cuatro Módulos.

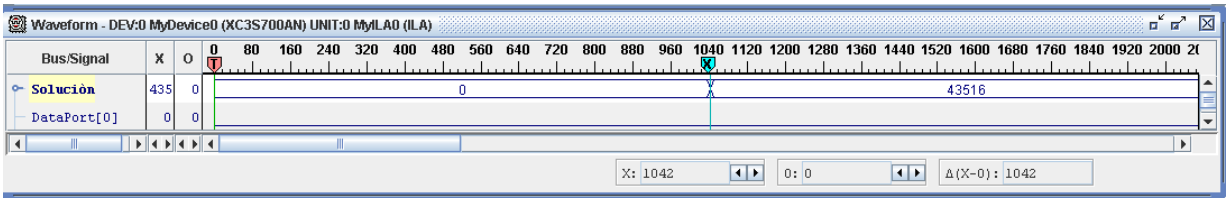


Figura 4.15: Metódo Montecarlo de forma Manual con cuatro Módulo.

En la Tabla 4.8 se observan los resultados obtenidos al paralelizar cuatro veces el proceso en hardware contra los resultados obtenidos en software.

Tabla 4.8: Resultados de la Integral en Formato Punto Fijo y Decimal con cuatro Módulos Montecarlo.

Iteraciones	Matlab		Impulse V1 y V2			Impulse V3			Manual		
	Dato	% Error	Dato PF	Dato	% Error	Dato PF	Dato	% Error	Dato PF	Dato	% Error
1024	0,67237	0,8560	44062	0,67233	0,8499	44480	0,67871	1,8066	43651	0,66606	-0,0908
2048	0,66609	-0,0870	43651	0,66606	-0,0908	43550	0,66452	-0,3220	43638	0,66586	-0,1205
4096	0,66957	0,4359	43875	0,66948	0,4219	43629	0,66573	-0,1411	43916	0,67010	0,5157
8192	0,66860	0,2905	43813	0,66853	0,2800	43730	0,66727	0,0900	43740	0,66742	0,1129
16384	0,66757	0,1350	43746	0,66751	0,1266	43780	0,66803	0,2045	44011	0,67155	0,7332
32768	0,66774	0,1605	43758	0,66769	0,1541	43809	0,66847	0,2708	43689	0,66664	-0,0038
65536	0,66786	0,1796	43766	0,66782	0,1724	43774	0,66794	0,1907	43775	0,66795	0,1930
131072	0,66680	0,0203	43696	0,66675	0,0122	43670	0,66635	-0,0473	43709	0,66695	0,0420

Tiempos de Ejecución

En la Tabla 4.9 se observan los tiempos de ejecución de las aplicaciones desarrolladas en Impulse C y de forma manual y se comparan con los tiempos de ejecución obtenidos al implementar el algoritmo en Matlab. La aceleración se calcula de la misma manera como se calculó en los resultados de la covarianza.

Tabla 4.9: Tiempos de Ejecución para un Módulo de Integral.

Tiempos de Ejecución		**Matlab**	**Impulse V1**	**Impulse V2**	**Manual**
Iteraciones	Frecuencia [MHz]	2000	140	140	160
1024	Tiempo [ciclos de reloj]	-	1035	1035	1039
	Tiempo [μs]	356	7,39	7,39	6,49
	Aceleración	1	48,15	48,15	54,82
2048	Tiempo [ciclos de reloj]	-	2059	2059	2063
	Tiempo [μs]	553	14,71	14,71	12,89
	Aceleración	1	37,60	37,60	42,89
4096	Tiempo [ciclos de reloj]	-	4107	4107	4111
	Tiempo [μs]	878	29,34	29,34	25,69
	Aceleración	1	29,93	29,93	34,17
8192	Tiempo [ciclos de reloj]	-	8203	8203	8207
	Tiempo [μs]	1578	58,59	58,59	51,29
	Aceleración	1	26,93	26,93	30,76
16384	Tiempo [ciclos de reloj]	-	16395	16395	16399
	Tiempo [μs]	2796	117,11	117,11	102,49
	Aceleración	1	23,88	23,88	27,28
32768	Tiempo [ciclos de reloj]	-	32779	32779	32783
	Tiempo [μs]	5675	234,14	234,14	204,89
	Aceleración	1	24,24	24,24	27,70
65536	Tiempo [ciclos de reloj]	-	65547	65547	65551
	Tiempo [μs]	11203	468,19	468,19	409,69
	Aceleración	1	23,93	23,93	27,34
131072	Tiempo [ciclos de reloj]	-	131083	131083	131087
	Tiempo [μs]	22161	936,31	936,31	819,29
	Aceleración	1	23,67	23,67	27,05

En la Tabla 4.10 se puede apreciar la aceleración obtenida por el desarrollo en hardware para las diferentes descripciones implementadas en la FPGA contra los resultados arrojados en software para cuatro módulos trabajando en paralelo.

Tabla 4.10: Tiempos de Ejecución para cuatro Módulos de Integral.

Tiempos de Ejecución		**Matlab**	**Impulse V1**	**Impulse V2**	**Impulse V3**	**Manual**
Iteraciones	Frecuencia [MHz]	2000	140	140	140	85
1024	Tiempo [ciclos de reloj]	-	1041	1041	1035	1042
	Tiempo [μs]	894	7,44	7,44	7,39	12,26
	Aceleración	1	120,23	120,23	120,93	72,93
2048	Tiempo [ciclos de reloj]	-	2065	2065	2059	2066
	Tiempo [μs]	1541	14,75	14,75	14,71	24,31
	Aceleración	1	104,47	104,47	104,78	63,40
4096	Tiempo [ciclos de reloj]	-	4113	4113	4107	4114
	Tiempo [μs]	2843	29,38	29,38	29,34	48,40
	Aceleración	1	96,77	96,77	96,91	58,74
8192	Tiempo [ciclos de reloj]	-	8209	8209	8203	8210
	Tiempo [μs]	5435	58,64	58,64	58,59	96,59
	Aceleración	1	92,69	92,69	92,76	56,27
16384	Tiempo [ciclos de reloj]	-	16401	16401	16395	16402
	Tiempo [μs]	10631	117,15	117,15	117,11	192,96
	Aceleración	1	90,75	90,75	90,78	55,09
32768	Tiempo [ciclos de reloj]	-	32785	32785	32779	32786
	Tiempo [μs]	21020	234,18	234,18	234,14	385,72
	Aceleración	1	89,76	89,76	89,78	54,50
65536	Tiempo [ciclos de reloj]	-	65553	65553	65547	65554
	Tiempo [μs]	37440	468,24	468,24	468,19	771,22
	Aceleración	1	79,96	79,96	79,97	48,55
131072	Tiempo [ciclos de reloj]	-	131089	131089	131083	131090
	Tiempo [μs]	82326	936,35	936,35	936,31	1542,24
	Aceleración	1	87,92	87,92	87,93	53,38

Recursos de Hardware

En la Tabla 4.11 se puede observar la cantidad de recursos usados en hardware cuando las descripciones obtenidas son llevadas a implementación sobre una FPGA para un módulo integral.

Tabla 4.11: Recursos Utilizados en la FPGA para un Módulo Integral.

Recursos FPGA	**Total FPGA**	**Impulse V1**		**Impulse V2**		**Manual**	
		Recursos	**% Total**	**Recursos**	**% Total**	**Recursos**	**% Total**
Slices	5888	525	8,92	524	8,90	841	14,28
Flip-Flops	11776	745	6,33	745	6,33	1400	11,89
LUTs	11776	981	8,33	981	8,33	969	8,23
MULT18X18	20	2	10,00	2	10,00	0	0,00
DCMs	8	1	12,50	1	12,50	1	12,50

En la Tabla 4.12 se aprecia la cantidad de recursos utilizados en la FPGA para llevar a cabo la implementación de los cuatro módulos en paralelo.

Tabla 4.12: Recursos Utilizados en la FPGA para cuatro Módulos Integral.

Recursos FPGA	**Total FPGA**	**Impulse V1**		**Impulse V2**		**Impulse V3**		**Manual**	
		Recursos	**% Total**	**Recursos**	**% Total**	**Recursos**	**% Total**	**Recursos**	**% Total**
Slices	5888	2210	37,53	2209	37,52	1292	21,94	3337	56,67
Flip-Flops	11776	3102	26,34	3102	26,34	1628	13,82	5552	47,15
LUTs	11776	4208	35,73	4208	35,73	2363	20,07	3627	30,80
MULT18X18	20	8	40,00	8	40,00	4	20,00	0	0,00
DCMs	8	1	12,50	1	12,50	1	12,50	1	12,50

Tiempo de Diseño

Nuevamente el tiempo de diseño en Impulse C no tiene en cuenta el tiempo empleado en aprender a manejar esta herramienta. El tiempo mostrado en la Tabla 4.13 representa la cantidad de semanas necesarias para llevar a cabo las implementaciones de los diferentes módulos realizados en una FPGA, se destaca el hecho que mientras se realizó la implementación mediante la descripción de hardware, en un periodo mucho menor se logró obtener tres implementaciones completamente funcionales mediante Impulse C.

Tabla 4.13: Tiempo de Desarrollo Módulo Integral.

Tiempo de Desarrollo	**Matlab**	**Impulse C**	**Manual**
Tiempo en Semanas	0,5*	1**	10***

* 1. Algoritmo Montecarlo con generación mediante *rand*: un día.

2. Algoritmo Montecarlo con generación mediante método de Green: un día.

3. Ejecución y toma de datos: medio día.

** 1. Versiones preliminares: dos días.

2. Modulo Integral Versión uno: un día.

3. Modulo Integral Versión dos: un día.

4. Modulo Integral Versión tres: un día.

*** 1. Generación de números pseudo-aleatorios: 6 semanas.

2. Módulo Función: 1 semana.

3. Segmentación y mejoramiento del módulo Función: 1 semana.

4. Descripción del módulo Solución: 1 semana.

5. FSM, unión de los módulos : 1 semana

Capítulo 5

Conclusiones

Para desarrollar la evaluación de la herramienta Impulse C se tomaron en cuenta tres aspectos: el tiempo de diseño, el tiempo de ejecución y la cantidad de recursos lógicos. Con base en esos resultados se determinó la conveniencia de usar está herramienta en el desarrollo de procesadores de propósito específico, que puedan ser implementados de forma eficiente y confiable sobre un dispositivo lógico programable. Las siguientes son las conclusiones que se surgen al respecto:

- La herramienta de compilación Impulse C permite reducir el tiempo de diseño y desarrollo de una implementación. Esta es una gran ventaja a la hora de implementar algoritmos computacionales sobre hardware y como se puede evidenciar en las Tablas (4.6) y (4.13), los tiempos de desarrollo de una descripción en Impulse C son cortos con relación a una descripción de hardware desarrollada de manera tradicional, llegando a ser hasta 10 veces menores como en el caso de la aplicación de Montecarlo y de 4 en el caso de la matriz de covarianza. Por ejemplo, es posible hacer mejoras en el algoritmo con sólo modificar algunas lineas de código en lenguaje C, compilar el nuevo código haciendo uso de Impulse C y obtener procesadores de propósito específico optimizados, caso contrario a lo que ocurre cuando el desarrollo de los procesadores de propósito específico se hace de manera tradicional, ya que cualquier alteración del algoritmo original hace que se produzcan cambios significativos en la descripción que se está realizando.

- Se pudo observar que en cuanto a tiempos de ejecución las aplicaciones implementadas sobre el FPGA presentan una aceleración hasta 120 veces mayor, si se comparan con las implementaciones desarrolladas en software, esto se debe a que los computadores se rigen por un tipo de arquitectura secuencial mientras que los procesadores de propósito específico permiten paralelizar procesos. Ahora, si se comparan los *cores* obtenidos con la herramienta de compilación con los obtenidos de forma tradicional, se puede apreciar que éstos últimos pueden trabajar a frecuencias más elevadas dependiendo de la implementación como en el caso de un sólo módulo del método Montecarlo

(Tabla 4.9). Pero, en el caso de la matriz de covarianza de un solo módulo (Tabla 4.2), la matriz de covarianza de tres módulos (Tabla 4.3) y en el caso de la implementación del método Montecarlo de cuatro módulos (Tabla 4.10) se observó que los tiempos de ejecución de los *cores* desarrollados mediante Impulse C fueron levemente mejores. Esto se debe en el primer caso, a que el cálculo de la matriz de covarianza con el C to VHDL está conformada por componentes de alta velocidad como lo son los Multiplicadores 18X18 y FIFOs que le permiten trabajar a diferentes frecuencias de reloj y en el segundo caso, a que cuando se desarrolló el cálculo del promedio de forma tradicional, se usaron operaciones de tipo combinacional que castigaron la frecuencia de operación. Sin embargo, se pudo concluir que a pesar de generar módulos que trabajan a una frecuencia de operación menor en algunos casos, los *cores* obtenidos con Impulse C logran acelerar los algoritmos computacionales implementados en lenguaje C.

- Con respecto a los recursos lógicos utilizados, se pueden apreciar diferencias notables entre las descripciones obtenidas por Impulse C y las desarrolladas de manera tradicional. En el caso del módulo de covarianza la cantidad de recursos utilizados en la implementación es menor en la descripción desarrollada de forma manual que las obtenidas con Impulse C (Tablas 4.4), sin embargo para la implementación del método de Montecarlo (Tabla 4.11) Impulse C arroja como resultados *cores* que administran de una forma más eficiente los recursos.

- Al observar los resultados obtenidos en cualquiera de las implementaciones realizadas ya sea de forma manual o mediante la herramienta C to VHDL Impulse C, estos se encuentran en un rango de tolerancia inferior al 2% y por lo tanto podemos decir que los errores relativos que se producen en dichas aplicaciones no son significativos.

- La curva de aprendizaje para la herramienta de compilación Impulse C, es bastante lenta ya que requiere en cierta medida una especie de ensayo y error para encontrar los criterios que mejor se adaptan a las descripciones que se desean implementar. También hay que tener en cuenta que es necesario aprender qué tipos de datos son soportados y cómo se pueden representar en esta herramienta, por lo cual, es necesario reescribir parte del código. Esta curva de aprendizaje no se compara con el tiempo, esfuerzo y conocimiento necesario para realizar una descripción de hardware similar a la obtenida mediante Impulse C. Como ventaja está el hecho que una vez realizado el proceso de aprendizaje de la herramienta, se torna fácil hacer uso de ella, permitiendo con ello realizar múltiples variantes de una misma aplicación, como se puede constatar en los resultados. Mientras se realizaba el proceso de describir un algoritmo en forma manual mediante VHDL fue posible mediante Impulse C realizar

3 versiones para cada una de las aplicaciones en no más de una semana.

- Por último, se puede concluir que Impulse C es una herramienta bastante completa que entrega descripciones de hardware, sintetizables y cumple con los criterios de evaluación que se establecieron a la hora de proponer esta investigación. Además los resultados obtenidos superan las expectativas que se tenían al momento de adquirir la versión de evaluación con cual se trabajó. Aunque, cabe anotar que si bien, la herramienta permite disminuir los tiempos de diseño y desarrollo de un proyecto, no implica que los desarrolladores de hardware queden relegados a un segundo plano, porque a pesar de ser una muy buena herramienta en ocasiones es necesario el criterio del diseñador de hardware que determina en qué ocasiones es mejor reducir área y maximizar frecuencia o viceversa.

Capítulo 6

Trabajo Futuro

- Evaluar otro tipo de compiladores C to VHDL como por ejemplo Catapult-C[24], Dime-C[25] y en especial la herramienta Vivado Design Suite[26] de Xilinx por ser el primero y mayor fabricante de dispositivos lógicos programables.

- Involucrar a estudiantes con conocimientos en C, para que hagan parte de el grupo de investigación de CPS y aprendan a manejar las herramientas de compilación C to HDL, que se encuentren disponibles en la escuela.

- Incursionar en el campo de los lenguajes de alto nivel para el diseño de hardware como Handel C[4] o System C[6], con el objetivo tener a la mano otras alternativas de desarrollo de hardware y no depender totalmente de los lenguajes HDL. Además estos lenguajes son una de las alternativas que existen actualmente para la implementar sobre FPGAs algoritmos desarrollados bajo el estándar ANSI C.

Bibliografía

[1] Impulse Accelerated Technologies. CoDeveloper from Impulse Accelerated Technologies. pages 1–136.

[2] Mª Goretti Sevillano Berasategui. Circuitos Digitales basados en FPGAs para Generación de Números Aleatorios.

[3] Eduardo Boemo Scalvinoni. Estado del arte de la tecnologia fpga. Technical report, 2005.

[4] Matthew Bowen. *Handel-C Language Reference Manual*.

[5] Mentor Graphics. http://www.mentor.com/products/fpga/handel-c, Octubre 2012.

[6] Waldo López and Santa Ana Javier. Introducción a SystemC. 2003.

[7] Accellera Systems Initiative. http://www.accellera.org/home, Octubre 2012.

[8] Arcilio J Virginia, Yana D Yankova, and Koen L M Bertels. An empirical comparison of ANSI-C to VHDL compilers : Spark, Roccc and DWARV. In *Anual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, pages 388–394, Veldhoven, Netherlands, 2007.

[9] Impulse Accelerated Technologies. http://www.impulseaccelerated.com, Octubre 2012.

[10] Pedro Morales Vallejo. Correlación y Covarianza. 2008.

[11] Patricia Saavedra Barrera. Integración numérica por Monte-Carlo. pages 1–10, 2008.

[12] Xilinx. http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm, Octubre 2012.

[13] Xilinx. ChipScope ILA Tools Tutorial (for ChipScope ILA). Technical report, 2003.

[14] Dick Grone. *Diseño de compiladores modernos*. McGraw Hill, 2007.

[15] Jeffrey D. Ullman Alfred V. Aho, Ravi Sethi. *Compiladores: Principios, técnicas y herramientas*. Pearson, 2007.

[16] Jacinto Ruiz Catalán. *Compiladores: Teoría e implementación*. Alfaomega, 2010.

[17] Pablo Huerta Pellitero. *Sistemas de multiprocesamiento simétrico sobre FPGA*. PhD thesis, Universidad Rey Juan Carlos, 2009.

[18] Carlos A Fajardo, Javier Castillo, Aparece En, and L A Tesis. Viailidad de acelerar la migración sísmica utilizando clusters heterogéneos. *Communications*, 2011.

[19] Abreo Carrillo, Sergio Alberto, Ramírez Silva, and Ana Beatriz. Viabilidad de acelerar la migración sísmica 2D usando un procesador específico implementado sobre un FPGA The feasibility of speeding up 2D seismic migration using a specific processor. 2010.

[20] Brian Holland, Mauricio Vacas, Vikas Aggarwal, Ryan Deville, Ian Troxel, and Alan D George. Survey of C-based Application Mapping Tools for Reconfigurable Computing. 2005.

[21] SPARK. http://mesl.ucsd.edu/spark/, Octubre 2012.

[22] ROCCC 2.0. http://www.jacquardcomputing.com/roccc/, Octubre 2012.

[23] Yankova Yana, Kuzmanov Georgi, Bertels Koen, Gaydadjiev Georgi, Lu Yi, and Vassiliadis Stamatis. DWARV : DELFT WORKBENCH AUTOMATED RECONFIGURABLE VHDL GENERATOR.

[24] CATAPULT-C. http://calypto.com/products/catapult/overview, Octubre 2012.

[25] DIME-C. http://www.nallatech.com/Development-Tools/dime-c.html, Octubre 2012.

[26] Vivado. http://www.xilinx.com/products/design-tools/vivado/index.htm, Octubre 2012.

[27] Lindsay I Smith. A tutorial on Principal Components Analysis. *Statistics*, 2002.

[28] Carlos Fajardo and Jorge Ramón. Descripción de una metodologia para diseñar procesadores de propósito específico implementados sobre FPGAs. In *XV SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISIÓN ARTIFICIAL - STSIVA 2010*.

[29] Gajski Daniel. *Principios de Diseño Digital*. Prentice Hall.

[30] Jorge Puertas Herranz. Estimación de Coste y Plazo en Proyectos de Túneles Ejecutados Mediante Excavación Convencional y Voladura. 2010.

[31] Universidad Nacional del Centro de la Provincia de Buenos Aires Facultad de Ciencias Exactas. Simulación Método Monte Carlo, 2005.

[32] German Prieto. Capitulo 3 : Integración númerica, 1998.

[33] Alejandro Ramirez. http://www.mat.puc.cl/ aramirez/clases/parte5.pdf.

[34] Alex Rosete. http://alexrosete.orgfree.com/materiales_2004/06Simulacion/Manual_Asignatura-Simulacion_a.pdf, Octubre 2012.

[35] Antonio Salmeron Cerdán and María Morales Giraldo. *Estadística Computacional*. 2001.

[36] William E Mahoney and Paul D Coddington. Evaluating Parallel Random Number Generators, 1993.