SEP **TECNOLÓGICO NACIONAL MÉXICO**

INSTITUTO TECNOLÓGICO DE TIJUANA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROCESADOR DE USO DEDICADO GSGP COMO DISPOSITIVO DE IoT

TRABAJO DE TESIS PRESENTADO POR:

ING. RUBEN DARIO SALAS VILLEGAS

M1921052

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA INGENIERÍA

DIRECTORA DE TESIS:

DRA. YAZMIN MALDONADO ROBLES

TIJUANA, BAJA CALIFORNIA, MÉXICO

16 DE JUNIO DEL 2022

Tijuana Baja California, 16/Junio/2022
Asunto: Autorización de impresión de trabajo de tesis

DR. GUADALUPE HERNÁNDEZ ESCOBEDO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

En lo referente al trabajo de tesis, **"Procesador de uso dedicado GSGP como dispositivo de IoT"** presentado por el **Ing. Rubén Darío Salas Villegas,** alumno del programa de Maestría en Ciencias de la Ingeniería, con número de control **M1921052** informamos a usted que después de una minuciosa revisión e intercambio de opiniones, los miembros del comité manifiestan **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias, por lo que se autoriza al interesado para que proceda de inmediato a la impresión del mismo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®

DRA. YAZMIN MALDONADO ROBLES
DIRECTORA DE TESIS

DR. LEONARDO TRUJILLO REYES
MIEMBRO DEL COMITÉ

DR. PAUL ANTONIO VALLE TRUJILLO
MIEMBRO DEL COMITÉ

DR. DANIEL EDUARDO HERNÁNDEZ MORALES
MIEMBRO DEL COMITÉ

ccp. Archivo
Dr. José Ricardo Cárdenas Valdez – Coordinador Académico de la Maestría en Ciencias de la Ingeniería.

Tijuana, Baja California, 16/junio/2022
OFICIO No. 066/DEPI/2022
Asunto: Autorización de Impresión de Tesis

MARÍA MAGDALENA SERRANO ORTEGA
JEFA DEL DEPARTAMENTO DE SERVICIOS ESCOLARES
PRESENTE

En lo referente al trabajo de tesis, "Procesador de uso dedicado GSGP como dispositivo de IoT". Presentado por C. **Rubén Darío Salas Villegas**, alumno de la Maestría en Ciencias de la Ingeniería con numero de control **M1921052.**; informo a usted que a solicitud del comité de tutorial, tengo a bien **Autorizar la impresión de Tesis**, atendiendo las disposiciones de los Lineamientos para la Operación de Estudios de Posgrado del Tecnológico Nacional de México.

Sin más por el momento le envió un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE TIJUANA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

GUADALUPE HERNÁNDEZ ESCOBEDO
JEFE DE DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

ccp. Archivo

GHE/lap

CARTA DE CESIÓN DE DERECHOS

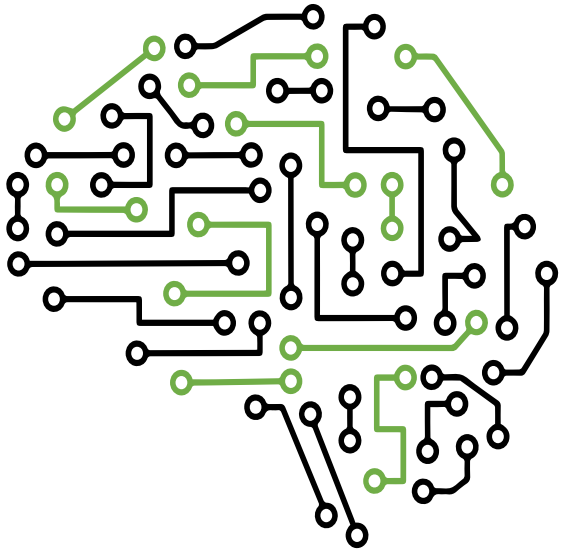
En la ciudad de Tijuana, Baja California, el día **16** del mes de **junio** del año **2022**, el que suscribe **Rubén Darío Salas Villegas**, con número de control **M1921052** alumno de **Maestría** del programa de Posgrado en Ciencias de la Ingeniería, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección de la **Dra. Yazmin Maldonado Robles** cede los derechos para su difusión, en su totalidad o en partes, con fines académicos o de investigación del documento de tesis titulado **"Procesador de uso dedicado GSGP como dispositivo de IoT"** al Tecnológico Nacional de México.

Los usuarios de la información no deben reproducir el contenido textual, gráficas, código, fórmulas o datos del trabajo sin permiso expreso del autor o directores del trabajo. Este debe ser obtenido escribiendo a cualquiera de las siguientes direcciones de correo electrónico **yaz.maldonado@tectijuana.edu.mx** o bien, dirigirse a las instalaciones del Instituto Tecnológico de Tijuana en Calzada del Tecnológico S/N Esq. Av. Castillo de Chapultepec y calle Cuauhtemotzin, Fracc. Tomás Aquino C.P. 22414, Tijuana, Baja California, conmutador 664-6078400.

Si se otorga el permiso, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo como lo indique el autor intelectual o el director del trabajo de Tesis.

ATENTAMENTE

RUBÉN DARÍO SALAS VILLEGAS
ALUMNO DEL POSGRADO EN CIENCIAS DE LA INGENIERÍA



Agradecimientos

Quiero expresar un enorme agradecimiento a....

A mis profesores por ser pilares en el proceso de mi formación.

A mis amigos, Iván, Diego y Ale que siempre estuvieron ahí para apoyarme, y siempre me animaron a continuar de frente, gracias por toda su ayuda.

A mis compañeros de maestría y laboratorios con los cuales pase momentos inolvidables.

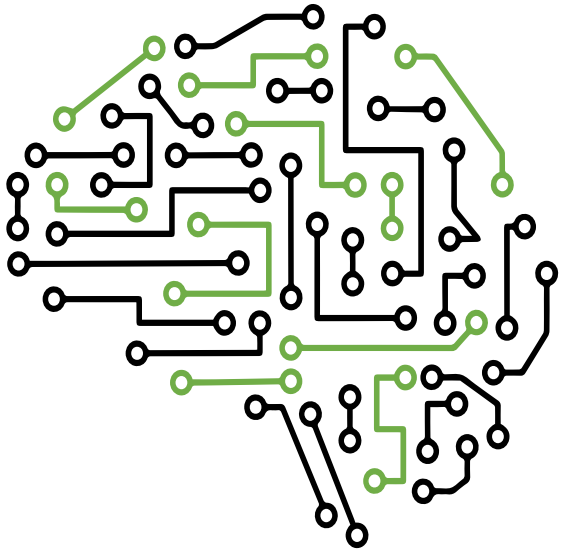
A mi Sol mi increíble diseñadora gráfica miniatura. Por enseñarme que existe grandeza donde menos te lo esperas.

A mi asesora, mi sensei, por permitirme ser parte de su equipo, y nunca perder la paciencia y la amabilidad a pesar de todos mis errores y tropiezos, fue grandioso trabajar con usted.

A mi familia, mi todo, los que han estado y siempre estarán a mi lado en cada camino que tome, muchísimas gracias.

Al Instituto Tecnológico de Tijuana por permitirme ser parte de su posgrado.

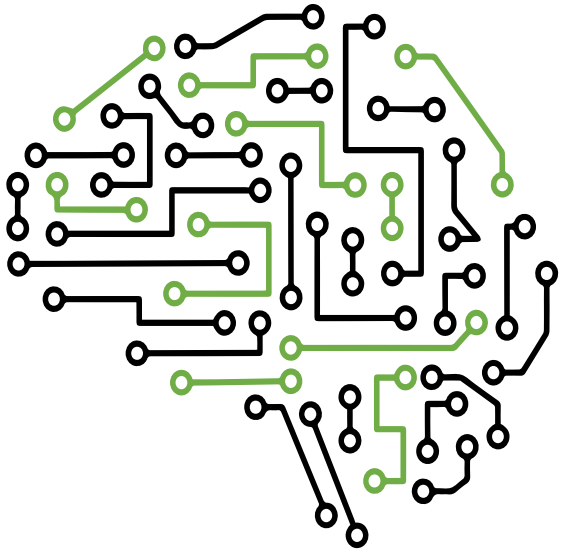
Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el gran apoyo con la beca No. 993254.



Contenido

Contenido (Continuación)

Capítulo Página



Lista de figuras

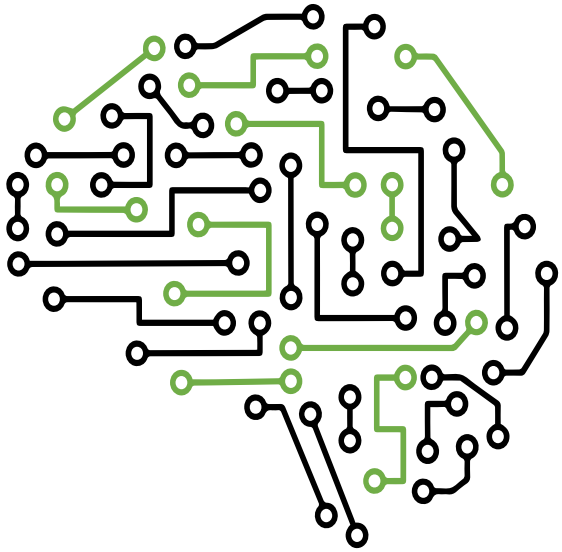
Figura Página

Lista de figuras (Continuación)

Figura Página

Lista de figuras (Continuación)

Figura Página

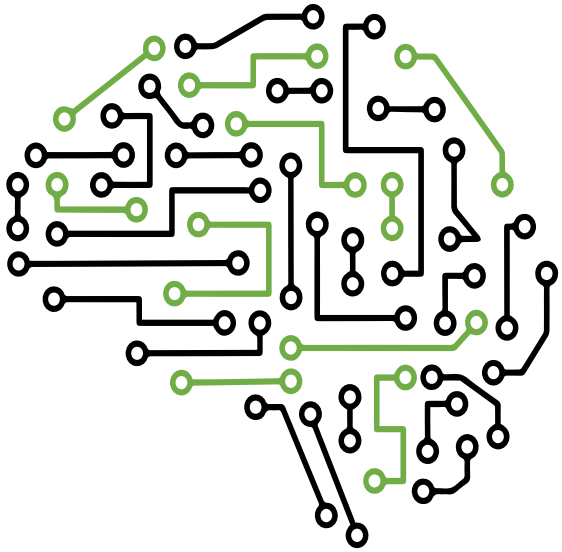


Lista de tablas

Tabla Página

Lista de tablas (Continuación)

Tabla Página



Resumen

Genetic Programming (GP) es una técnica dentro del campo de aprendizaje máquina que pertenece al área de inteligencia artificial. *Geometric Semantic Genetic Programming* (GSGP) es una variante de GP que ha logrado un rendimiento de vanguardia en distintos dominios, particularmente en problemas de regresión simbólica.

Internet of Things (IoT) es un nuevo paradigma que está aumentando en popularidad. puesto que ofrece conectividad avanzada de dispositivos, sistemas y servicios que va más allá de las comunicaciones de máquina a máquina y cubre una variedad de protocolos, dominios y aplicaciones. IoT produce gran volumen de datos, requiere de gran velocidad por lo que preservar la calidad de los datos es una tarea difícil y desafiante. Técnicas de aprendizaje máquina como GSGP resultan interesantes para el análisis de datos de IoT.

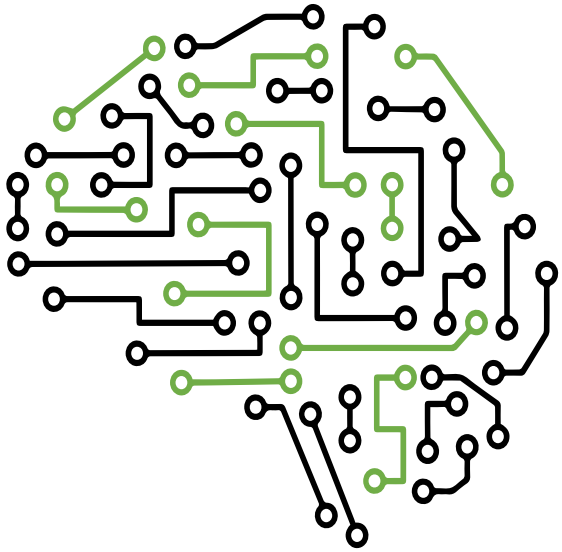
Por otro lado, los *Field Programmable Gate Arrays* (FPGAs) son dispositivos programables eficientes dado que permiten la programación paralela mediante *Hardware Description Language* (HDL) .

En este trabajo de tesis se diseñó un algoritmo en HDL de una variante GSGP para su implementación paralela en un dispositivo FPGA, el cual lleva por nombre, procesador dedicado GSGP-VHDL. GSGP-VHDL consta de 5 etapas, las cuales son: generación de la población, evaluación inicial, mutación de los individuos, evaluación de calidad y selección del mejor individuo.

Para determinar el rendimiento del GSGP-VHDL se realizaron experimentos con 6 problemas del *estado-del-arte* de regresión simbólica, housing, energy heating, energy cooling, concrete, yatch y tower, estos experimentos se hicieron con 2 y 12 variables. Los resultados experimentales muestran que el tiempo de ejecución del GSGP-VHDL es del orden de los microsegundos, en comparación con el GSGP-Software que es de segundos e incluso minutos.

Los resultados obtenidos demuestran que el procesador dedicado GSGP-VHDL es un sistema competitivo en rendimiento, por lo que se considera factible su aplicación en resolución de problemas de IoT.

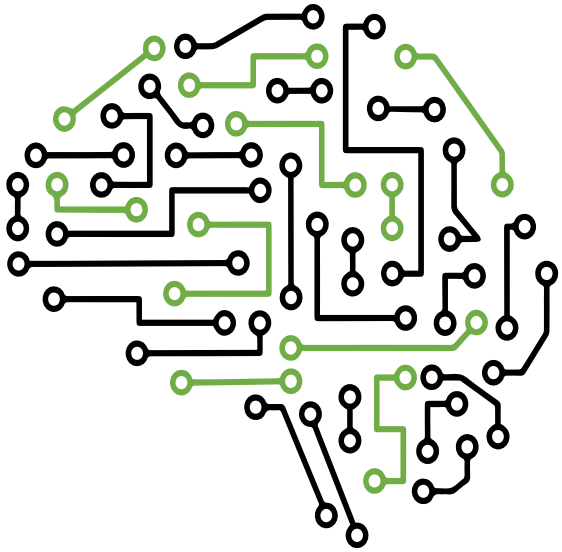
Palabras clave: Internet de las cosas, Aprendizaje máquina, GSGP, FPGA, VHDL.



Lista de acrónimos

ALU	Arithmetic Logic Unit
ABEL	Advanced Boolean Expression Language
BigData	Macrodata, great speed great volume and great variety of data
BMS	Building Management System
CAD	Computer Aided Design
CLB	Configurable Logic Block
CLK	clock cycles
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Computer Processor Unit
DSP	Digital Signal Processor
EA	Evolutionary Algorithm
EC	Evolutionary Computation
FF	Flip-Flop
FPGA	Field Programmable Gate Array
GP	Genetic Programming
GPU	Graphics Processing Unit
GSGP	Geometric Semantic Genetic Programming
GSM	Geometric Semantic Mutation
HDL	Hardware description Language
IDE	Integrated Design Environment
IEEE	Institute of Electrical and Electronics Engineers
IOB	Input-Output Block
IoT	Internet of Things
LUT	Look Up Table
MAE	Mean Absolute Error

ML	Machine Learning
NLP	Natural Language Processing
OMS	Optimal Mutation Step
RAM	Random Access Memory
RFID	Radio Frequency Identification
RTL	Register Transfer Level
STD	Standard Desviation
VHDL	Acronim of VHSIC and HDL
VHSIC	Very High Speed Integrated Circuit



Capítulo 1

Introducción

Internet of Things (IoT) conecta muchos dispositivos inteligentes, como sensores, teléfonos móviles y otros [33]. Estos dispositivos inteligentes pueden comunicarse entre sí e intercambiar información. Los dispositivos que interactúan con los usuarios generando y recuperando información sobre y desde su entorno, también contienen hardware que les permite controlar las salidas (como relés, conmutadores y puertos digitales) como se muestra en la Figura 1.1.

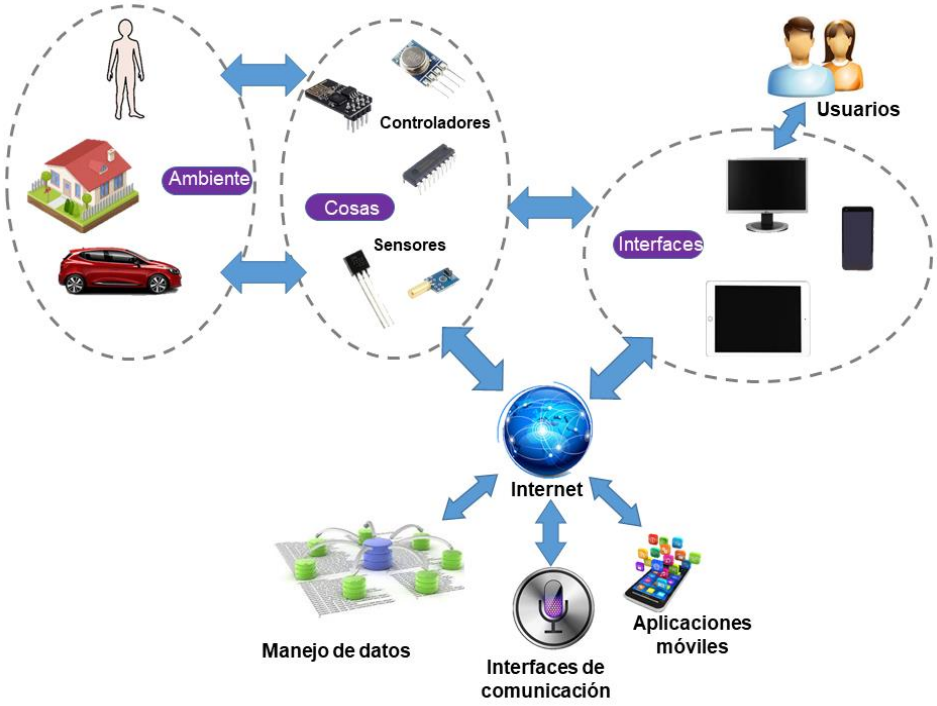


Figura 1.1: Ejemplo de IoT.

Según el informe estadístico de dispositivos conectados a internet, hay más de 50 mil millones de dispositivos IoT en el mundo. Al recopilar los datos de estos dispositivos IoT y analizarlos para detectar y comprender el entorno, los sistemas complejos se pueden construir para mejorar la calidad de vida, como el diagnóstico de la condición del cuerpo humano, seguridad, entre otros [8]. El propósito de IoT es desarrollar un entorno más inteligente y un estilo de vida simplificado ahorrando tiempo, energía y dinero. A través de esta tecnología, se pueden reducir los gastos en diferentes industrias. Las enormes inversiones y muchos estudios que se ejecutan en IoT son una tendencia creciente en los últimos años. IoT es un conjunto de dispositivos conectados que pueden transferir datos entre sí para optimizar su rendimiento, estas acciones se producen automáticamente y sin conciencia humana o aportes [28]. IoT incluye cuatro componentes principales:

- sensores,
- redes de procesamiento,
- análisis de datos y
- monitoreo del sistema.

Primero, los sensores recopilan la información del entorno. A continuación, se debe extraer el conocimiento de los datos sin procesar. Luego, los datos estarán listos para ser transferidos a otros objetos, dispositivos o servidores a través de internet [22]. Dado que IoT produce gran volumen, velocidad rápida y variedades de datos, preservar la calidad de ellos es una tarea difícil y desafiante [2]. Aunque se han investigado soluciones para resolver estos problemas, hasta el momento, ninguna de ellas puede manejar todos los aspectos de las características de los datos de manera precisa debido a la naturaleza distribuida de las soluciones de administración de datos masivos y las plataformas de procesamiento en tiempo real.

Genetic Programming (GP) [27] es una técnica dentro del campo de aprendizaje máquina y este a su vez pertenece al área de investigación de inteligencia artificial, dadas sus ventajas en cuanto resolución de problemas, GP se presenta como un candidato para resolver los inconvenientes presentes en IoT. No obstante, desde sus inicios no ha tenido una

buena aceptación por parte de la comunidad debido a algunos problemas que limitaban su eficiencia como el alto costo computacional.

Geometric Semantic Genetic Programming (GSGP) [36][6] es una variante de GP que ha logrado un rendimiento de vanguardia en distintos dominios, principalmente en problemas de regresión simbólica, sin embargo, al igual que GP el costo computacional es alto.

Por otro lado, existen los dispositivos de descripción de hardware, particularmente los *Field Programmable Gate Arrays* (FPGAs) que dada su estructura interna son muy eficientes ya que permiten la programación paralela y recurrente mediante *Hardware Description Language* y *Very High Speed Integrated Circuit* (VHDL) [37].

1.1. Planteamiento del Problema

En los últimos años, IoT se ha convertido en una de las tecnologías más importantes, se estima que para el 2030, existan 500 mil millones de dispositivos conectados a internet [32]. Uno de los desafíos que esta tecnología enfrenta es el uso óptimo de los recursos de energía disponibles dada la vida útil de la batería del dispositivo, además, el procesamiento masivo de datos en la nube de forma permanente, ya que esos datos son utilizados para que la inteligencia artificial y el *Machine Learning* (ML) realicen análisis casi de inmediato [20]. Contar con dispositivos que ataquen estos problemas es una tarea pendiente.

1.2. Hipótesis

Dada la problemática actual, es posible implementar un sistema embebido basado en ML y FPGA que permita resolver problemas de análisis de datos, consumo de energía y/o reducir la necesidad de servidores web.

1.3. Objetivo

1.3.1. Objetivo General

Desarrollar un sistema de evaluación experimental para un procesador dedicado GSGP implementado en un FPGA para uso en dispositivos de IoT.

1.3.2. Objetivo Específicos

- Desarrollar un sistema en FPGA que evalúe el consumo de recursos en cada proceso del GSGP.
- Implementar una metodología experimental en VHDL para evaluar el rendimiento del procesador dedicado GSGP en el FPGA.
- Identificar las mejores prácticas para el uso en aplicaciones IoT del procesador GSGP.

1.4. Justificación

Con la popularidad y el uso generalizado de IoT, los sensores y dispositivos de almacenamiento están generando datos masivos, lo que produce que se desarrollen nuevas aplicaciones de IoT para proporcionar más servicios precisos y seguros a los usuarios. Sin embargo, el procesamiento masivo de datos puede procesarse y analizarse aún más para ofrecer mejores resultados a los proveedores y usuarios de servicios de IoT [8].

ML es un conjunto de técnicas, que ya se han aplicado en múltiples campos, como visión por computadora, gráficos por computadora, procesamiento de lenguaje natural, reconocimiento de voz, toma de decisiones y control inteligente [41]. Del mismo modo, ML también puede aportar un beneficio potencial a la red informática. Algunas investigaciones estudiaron como utilizar el aprendizaje máquina para resolver problemas de redes, incluido el enrutamiento, la ingeniería de tráfico, la asignación de recursos y la seguridad [3]. La aplicación de ML para IoT permite a los usuarios obtener análisis profundos y desarrollar aplicaciones de IoT inteligentes; Esto se debe a que el ML puede proporcionar

soluciones factibles para extraer la información y las características ocultas en los datos de IoT [8].

Por otra parte, la velocidad de datos y la carga útil del dispositivo afectan directamente el consumo de energía. La alta velocidad y el tamaño de los datos de escritura conducen a un bajo consumo de energía. Entonces, en el nivel de protocolos para conectividad de corto alcance, los protocolos WiFi de baja potencia son la mejor solución [33].

1.5. Aportación al *estado-del-arte*

Con el desarrollo de esta tesis, se participó en los foros académicos:

- El 7mo encuentro estatal de jóvenes investigadores Baja California 2020, con el tema "Implementación de circuitos especializados en FPGA".
- El 7mo congreso internacional de investigación de Tijuana con el artículo "Representación de matrices en FPGA y su aplicación como árboles de sintaxis".

En adición a este último se cuenta con la publicación del artículo "Representación de matrices en FPGA y su aplicación como árboles de sintaxis" en la revista Aristas de ciencia básica y aplicada de la Universidad Autónoma de Baja California con ISSN 2007-9478, el cual puede ser encontrado en el siguiente link: *http* : *//revistaaristas.tij.uabc.mx/index.php/revista*$_a$*ristas/article/view/*104

1.6. Estructura de la tesis

La organización del documento está compuesto por el presente Capítulo de Introducción y cinco Capítulos centrales, un Capítulo de Conclusiones y Trabajo futuro, además de las Referencias y Anexos. A continuación se resumen los Capítulos centrales.

- **Capítulo 2**: Marco teórico sobre dispositivos de descripción de hardware, específicamente FPGAs, así como lenguajes de programación y arquitectura.
- **Capítulo 3**: *Estado-del-arte* de la programación genética, programación genética geométrica semántica, espacio semántico y su representación.

- **Capítulo 4**: En este apartado, se presenta la parte medular de este trabajo de tesis, el diseño en VHDL de cada etapa del GSGP y su validación mediante simulación.
- **Capítulo 5**: En este Capítulo se detallan los experimentos realizados y los resultados obtenidos. Las pruebas se realizaron con problemas del *estado-del-arte* y con estos resultados se visualiza que el procesador dedicado GSGP-VHDL es candidato a resolver problemas de IoT.

Capítulo 2

Dispositivos de Descripción de Hardware

Como consecuencia de la creciente necesidad de integrar un mayor número de dispositivos en un solo circuito, se desarrollaron nuevas herramientas de diseño que auxilian al ingeniero a integrar sistemas de mayor complejidad. Esto permitió que en la década de los cincuenta aparecieran los *Hardware Description Language* (HDLs) como una opción de diseño para el desarrollo de sistemas electrónicos complejos[21].

Los HDLs son utilizados en los ciclos de diseño de sistemas digitales asistidos por herramientas de *Computer Aided Design* (CAD) electrónico. Al principio surgieron una serie de lenguajes que no llegaron a alcanzar un éxito que permitiera su consolidación en el campo industrial o académico. En los años 80 aparecen los lenguajes Verilog y VHDL que, aprovechando la disponibilidad de herramientas hardware y software cada vez más potentes y asequibles y los adelantos en las tecnologías de fabricación de circuitos integrados, logran imponerse como herramientas imprescindibles en el desarrollo de nuevos sistemas [11].

Una de las principales características de estos lenguajes radica en su capacidad para describir en distintos niveles de abstracción (funcional, transferencia de registros *Register Transfer Level* (RTL) y lógico) algún diseño en particular. Los niveles de abstracción se emplean para clasificar modelos HDLs según el grado de detalle y precisión de sus descripciones [26].

Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

- Algorítmico: se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final.
- Transferencia de registros RTL: Consiste en la distribución del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque.
- Lógico: el circuito se expresa en términos de ecuaciones lógicas o de compuertas.

2.1. VHDL

VHDL es descrito como un lenguaje formal destinado a utilizarse en todas las fases de la creación de sistemas digitales. Debido a que es compatible con el desarrollo, verificación, síntesis y prueba de diseños de hardware; la comunicación de datos de diseño de hardware; y el mantenimiento, modificación y adquisición de hardware [30].

VHDL se originó en el departamento de defensa estadounidense, quien reconoció que tenían un problema en sus programas de adquisición de hardware. El problema era que recibían diseños en lenguajes de descripción de hardware patentados, lo que significaba que no solo era imposible transferir datos de diseño a otras empresas para su segundo abastecimiento, sino que tampoco había garantía de que estos lenguajes sobrevivieran durante la vida útil del hardware en cuestión.

La solución fue tener un lenguaje de descripción de hardware estándar único, que pudiera ser usado frecuentemente en el futuro. La especificación de dicho lenguaje se llevó a cabo como parte del programa de VHSIC a principios de la década de 1980. Por esta razón, el lenguaje se denominó más tarde lenguaje VHDL.

Si el lenguaje hubiera continuado como un requisito para la adquisición militar, muy posiblemente seguiría siendo un lenguaje confidencial de interés solo para los contratistas del departamento de defensa. Sin embargo, la importancia del desarrollo del lenguaje, y especialmente la importancia de la estandarización del lenguaje, fue reconocida por la comunidad de ingenieros electrónicos [30]. Y así, el lenguaje formativo pasó al dominio público al ponerlo a disposición del *Institute of Electrical and Electronics Engineers* (IEEE) en 1986. IEEE procedió a consolidar el lenguaje en un estándar que fue ratificado como estándar IEEE número 1076 en 1987 [17].

El lenguaje VHDL fue creado con el propósito de especificar y documentar circuitos y sistemas digitales utilizando un lenguaje formal. Las principales características del lenguaje VHDL se explican en los siguientes puntos:

Descripción textual normalizada: VHDL es un lenguaje de descripción que especifica los circuitos electrónicos en un formato adecuado para ser interpretado tanto por computadoras como por personas, normalizado, es decir, existe un único modelo para el lenguaje, cuya utilización está abierta a cualquier grupo que quiera desarrollar herramientas basadas en dicho modelo, garantizando su compatibilidad con cualquier otra herramienta que respete las indicaciones especificadas. Es, por último, un lenguaje ejecutable, lo que permite que la descripción textual del hardware se materialice en una representación del mismo utilizable por herramientas auxiliares.

Amplio rango de capacidad descriptiva: El lenguaje VHDL posibilita la descripción del hardware con distintos niveles de abstracción, pudiendo adaptarse a distintos propósitos y utilizarse en las sucesivas fases que se dan en el desarrollo de los diseños. Además, es un lenguaje adaptable a distintas metodologías de diseño y es independiente de la tecnología.

Otras ventajas: Además de las ventajas ya reseñadas también es destacable la capacidad del lenguaje para el manejo de proyectos de grandes dimensiones, las garantías que comporta su uso cuando, durante el ciclo de mantenimiento del proyecto, hay que sustituir componentes o realizar modificaciones en los circuitos [11].

2.1.1. Tipos de datos

Datos predefinidos

VHDL permite utilizar tipos de datos predefinidos, así como otros definidos por el usuario. Los tipos predefinidos más comunes son:

- **Bit**, tipo de dato que utiliza valores de 0 y 1 lógico. Para hacer una asignación a un objeto tipo bit el valor binario tiene que aparecer entre comas simples (‘0’ o ‘1’).
- **Bit_vector (rango)**, el rango siempre entre paréntesis, indica el número de bits del vector, éstos sólo pueden estar formados por ceros y unos. Para un vector de N bits el rango será N-1 downto 0, donde el bit más a la izquierda es el más significativo y

el bit más a la derecha el menos significativo (notación binaria estándar). Para hacer una asignación el valor tiene que aparecer entre comillas (por ejemplo: “1100”).

- **Boolean**, tipo de dato que define valores de verdadero o falso en una expresión.
- **Character**, tipo de dato que representa símbolos que corresponden generalmente al código ASCII.
- **String**, tipo de dato que representa una cadena de caracteres.
- **Integer (rango)**, tipo de dato que representa cualquier número entero dentro del rango, aquí el rango no va entre paréntesis, puede expresarse como 0 to MAX.
- **Natural (rango)**, tipo de dato que utiliza enteros no negativos dentro del rango.
- **Positive (rango)**, tipo de dato que utiliza enteros positivos dentro del rango.
- **Real (rango)**, tipo de dato que representa un número real dentro del rango.
- **Std_Logic**, tipo de dato predefinido en el estándar IEEE 1164, es el estándar en la industria puede tomar 9 valores los cuales están definidos en la Figura 2.1, así mismo, en la Tabla 2.1 se muestra el significado de los distintos valores en este tipo de dato.
- **Std_Logic_Vector**, representa un vector de Std_Logic, posee las mismas reglas de asignación y definición del rango que el tipo Bit_vector pero con un mayor número de valores [31].

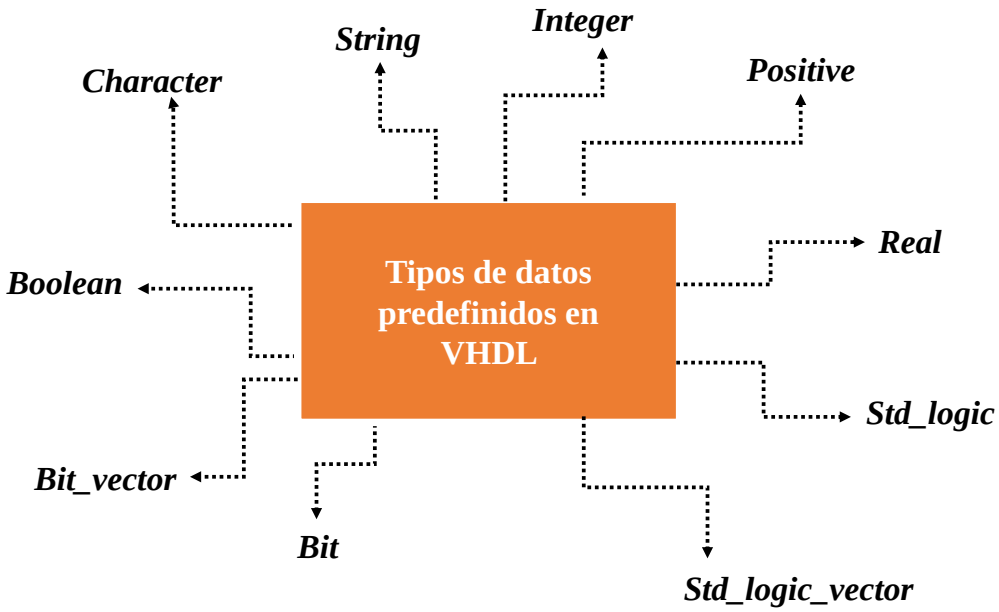


Figura 2.1: Tipos de datos predefinidos: *Bit*, utiliza valores de 0 (cero lógico) y 1 (uno lógico). *Bit_vector*, es un vector formado por elementos de tipo Bit. *Boolean*, define valores de true (verdadero) o false (falso) en una expresión. *Character*, representa símbolos que corresponden generalmente al código ASCII. *String*, es un vector de character (arreglo de 1D x 1D). *Integer*, representa un número entero. *Natural*, representa enteros no negativos. *Positive*, representa enteros positivos. *Real*, representa un número real. *Std_Logic*, el primero es un tipo de dato "resuelto". *Std_Logic_Vector*, define un vector de Std_Logic [10].

Tabla 2.1: Significado y posible uso para los nueve símbolos Std_Logic. La principal característica del tipo de dato Std_Logic, comparado con el tipo de dato Bit, es la inclusión de los valores 'Z' y '-'. El primero permite la construcción de buffer de tercer estado y el segundo una mejor optimización de tablas de búsqueda.

Valor	Significado
'U'	No inicializado (Uninitialized)
'X'	Desconocido (Forcing unknown)
'0'	Bajo (Forcing low)
'1'	Alto (Forcing high)
'Z'	Alta impedancia (High impedance)
'W'	Desconocido débil (Weak unknown)
'L'	Bajo débil (Weak low)
'H'	Alto débil (Weak high)
'-'	No importa (Don´t care)

VHDL permite utilizar todos los tipos de datos mencionados, sin embargo, para la conexión de los puertos se recomienda utilizar el tipo de dato Std_Logic o Std_Logic_Vector porque de esta manera se puede simular un circuito real y para realizar operaciones lógicas y aritméticas es conveniente que las operaciones o procesos se realicen con este tipo de dato o conjuntos de los mismos [31].

Datos Compuestos

Tanto los vectores como las matrices (arreglo de vectores) en VHDL son datos compuestos o como es mencionado en [16], son un conjunto o colección de datos indexados de un mismo tipo. Cada elemento de un vector o matriz puede ser accedido por medio de uno o más índices. La forma de definirlo es la siguiente:

TYPE identificador **IS ARRAY** (rango) **OF** tipo;

con esto, se define el tipo de dato, "identificador "es un arreglo que tiene un rango o número de elementos de un cierto tipo de dato, los tipos de datos pueden ser tanto los predefinidos en VHDL como los creados por el diseñador. La asignación de un valor a una posición del arreglo se realiza mediante números enteros. Con base en [31], una

vez definido el tipo compuesto y un nombre, se podrá definir cualquier señal, variable o constante como correspondiente a este nuevo tipo de dato.

Un ejemplo de definición de estos tipos de datos se muestra en la Figura 2.2

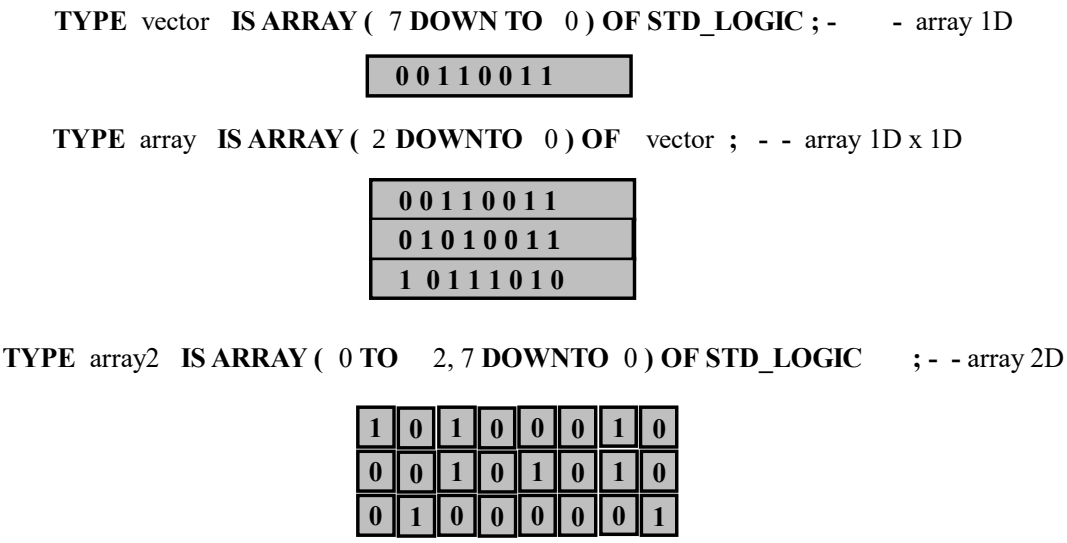


Figura 2.2: Ejemplo de definición de tipos de datos. En el primer renglón se define un nuevo tipo de dato llamado vector que es un arreglo de 1 fila por 8 columnas, en el segundo renglón se define un nuevo tipo de dato llamado array que es un arreglo compuesto de 3 elementos de tipo vector, por último en el tercer renglón se define un nuevo tipo de dato llamado array2 que es un arreglo compuesto de 3 filas por 8 columnas, estos dos últimos tipos son similares en los datos contenidos pero muy diferentes en sus definiciones.

2.1.2. Bibliotecas y paquetes

Un modelo VHDL se compone de un conjunto de unidades de diseño. Una unidad de diseño es la mínima sección de código compilable separadamente. Es un concepto muy simple, puede entenderse recurriendo a conocimientos básicos sobre lenguajes de programación por ejemplo: no se puede editar un fichero, escribir únicamente una sentencia *IF* o *CASE* y compilar el fichero, porque una sentencia no es una unidad de código compilable. Podrá compilarse un código, una función o un conjunto de declaraciones, o cualquier cosa que el lenguaje especifique.

Las unidades de diseño VHDL se construyen combinando construcciones del lenguaje

(sentencias, declaraciones, etc). En VHDL existen cinco tipos distintos de unidades de diseño, cada una de las cuales está pensada para el desempeño de una determinada función en el modelado del hardware. Las unidades se clasifican en primarias y secundarias (de acuerdo con las relaciones de dependencia jerárquica que mantienen entre sí, una unidad secundaría está asociada siempre a una unidad primaría), y son las siguientes:

1. La declaración de entidad: Es la unidad primaria del lenguaje que identifica a los dispositivos, definiendo su interfaz (nombre, terminales de conexión y parámetros de instanciamiento). Puede decirse que las entidades son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.) que forman de manera individual o en conjunto un sistema digital.

2. Arquitectura: Es una unidad secundaria del lenguaje. Está asociado a una determinada declaración de entidad, describiendo el funcionamiento lógico del dispositivo identificado por ésta. Aporta una información equivalente a la de las tablas de verdad o los cronogramas de los circuitos lógicos.

3. La declaración de paquete: Los paquetes desempeñan en VHDL funciones similares a las librerías en lenguajes de programación de alto nivel, sin embargo se comportan mas como bibliotecas. La declaración de paquete es una unidad primaria que contiene la “vista pública” de los paquetes.

4. El cuerpo de paquete: Es una unidad secundaria asociada a una declaración de paquete. Se utiliza, si resulta necesario, para definir los elementos declarados en éste.

5. La declaración de configuración: Es una unidad primaria que sirve para manejar el emplazamiento de componentes en modelos estructurales.

En el desarrollo de códigos VHDL pueden utilizarse o no tres de las cinco unidades de diseño, pero dos de ellas (la declaración de entidad y la arquitectura) son indispensables en la estructuración de un código. La declaración de entidad y el cuerpo de arquitectura exigen un mayor esfuerzo de desarrollo durante las tareas de diseño, las unidades con que

se construyen los paquetes VHDL (la declaración y el cuerpo de paquete) y los propios paquetes VHDL que son las unidades de código cuyo contenido puede utilizarse desde otras unidades mediante una serie de cláusulas de visibilidad.

La quinta unidad de código, la declaración de configuración, se utiliza rara vez en modelos de baja o media complejidad y su conocimiento no resulta imprescindible para iniciarse en el lenguaje [11].

Una parte importante en la programación con VHDL radica en el uso de bibliotecas y paquetes que permiten declarar y almacenar estructuras lógicas, seccionadas o completas que facilitan el diseño.

Una librería o biblioteca es un lugar al que se tiene acceso para utilizar las unidades de diseño predeterminadas por el fabricante de la herramienta (paquete) y su función es agilizar el diseño. En VHDL se encuentran definidas dos bibliotecas llamadas *ieee* y *work*, se muestran en la Figura 2.3. Como puede observarse, en la biblioteca *ieee* se encuentra el paquete *std_logic_1164*, mientras que en la biblioteca work se hallan *numeric_std*, *std_arith* y *gatespkg*.

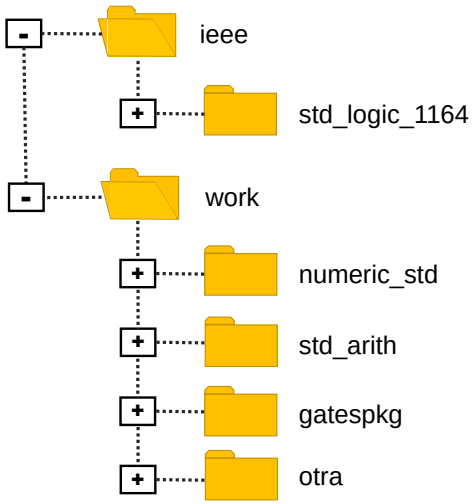


Figura 2.3: Contenido de las bibliotecas VHDL. La biblioteca ieee contiene el paquete std_logic_1164, en la bibilioteca work se encuentran los paquetes numeric_std, std_arith y gatespkg.

En una biblioteca también se permite almacenar el resultado de la compilación de un diseño, con el fin de utilizar en uno o varios programas. La biblioteca work es el lugar

establecido donde se almacenan los códigos que el usuario va generando. Esta biblioteca se encuentra siempre presente en la compilación de un diseño y los diseños se guardan en ella mientras no se especifique otra. La carpeta otra mostrada en la Figura 2.3 representa esta situación.

Un paquete es una unidad de diseño que permite desarrollar un código en VHDL de una manera ágil, debido a que contiene algoritmos preestablecidos (sumadores, restadores, contadores, etc.) que ya tienen optimizado su comportamiento. Por esta razón, el diseñador no necesita caracterizar paso a paso una nueva unidad de diseño si ya se encuentra almacenada en algún paquete (en cuyo caso basta con llamarla y especificarla en el código). Por lo tanto, un paquete no es más que una unidad de diseño formada por declaraciones, programas, componentes y subprogramas, que incluyen los diversos tipos de datos (bit, booleano, std_logic, etc.), empleados en la programación en VHDL y que suelen formar parte de las herramientas en software.

Por último, cuando en el diseño se utiliza algún paquete es necesario llamar a la biblioteca que lo contiene. Para esto se utiliza la siguiente declaración:

library ieee;

Lo anterior permite el uso de todos los componentes incluidos en la biblioteca ieee. En el caso de la biblioteca de trabajo (work), su uso no requiere la declaración library, dado que la carpeta work siempre está presente al desarrollar un diseño, es también en esta biblioteca donde se almacenan los paquetes creados por el usuario. El paquete std_logic_1164 (estándar lógico_1164) que se encuentra en la biblioteca ieee contiene todos los tipos de datos que suelen emplearse en VHDL (std_logic_vector, std_logic, entre otros). El acceso a la información contenida en un paquete es por medio de la sentencia use, seguida del nombre de la biblioteca y del paquete, respectivamente:

use nombre_librería.nombre_paquete.all;

por ejemplo:

use ieee.std_logic_1164.all;

En este caso ieee es la biblioteca, std_logic_l164 es el paquete y la palabra reservada all indica que se pueden usar todos los componentes almacenados en el paquete.

- El paquete numeric_std define funciones para realizar operaciones entre diferentes tipos de datos (sobrecargado); además, los tipos pueden representarse con o sin signo.
- El paquete numeric_bit define tipos de datos binarios con o sin signo.
- El paquete std_arith define funciones y operadores aritméticos, como igual (=), mayor que (>), menor que (<), entre otros.

En lo sucesivo, se usarán a menudo las bibliotecas y paquetes de los programas desarrollados en el texto [21].

2.2. FPGAs

Los dispositivos FPGAs se basan en lo que se conoce como arreglos de compuertas, los cuales consisten en la parte de la arquitectura que contiene tres elementos configurables: bloques lógicos configurables , bloques de entrada y de salida y canales de comunicación. La densidad de los FPGAs se establece en cantidades equivalentes a cierto número de compuertas [21].

2.2.1. Arquitectura de un FPGA

Por adentro, un FPGA está formado por arreglos de *Configurable Logic Block* (CLBs), que se comunican entre ellos y con los *Input-Output Block* (IOBs) por medio de conexiones llamados canales de comunicación. Cada FPGA contiene una matriz de bloques lógicos idénticos, por lo general de forma cuadrada, conectados por medio de líneas metálicas que corren vertical y horizontalmente entre cada bloque.

La función principal del FPGA es implementar una lógica programable que los clientes finales puedan utilizar para crear nuevos dispositivos de hardware. La matriz de un FPGA a menudo se conoce como la estructura lógica programable o simplemente la estructura. En los bordes hay IOBs programables diseñados para interconectar las señales de la estructura con el mundo exterior. Fue este conjunto de innovaciones lo que encendió la industria de FPGA. La Figura 2.4 muestra una arquitectura básica de un FPGA.

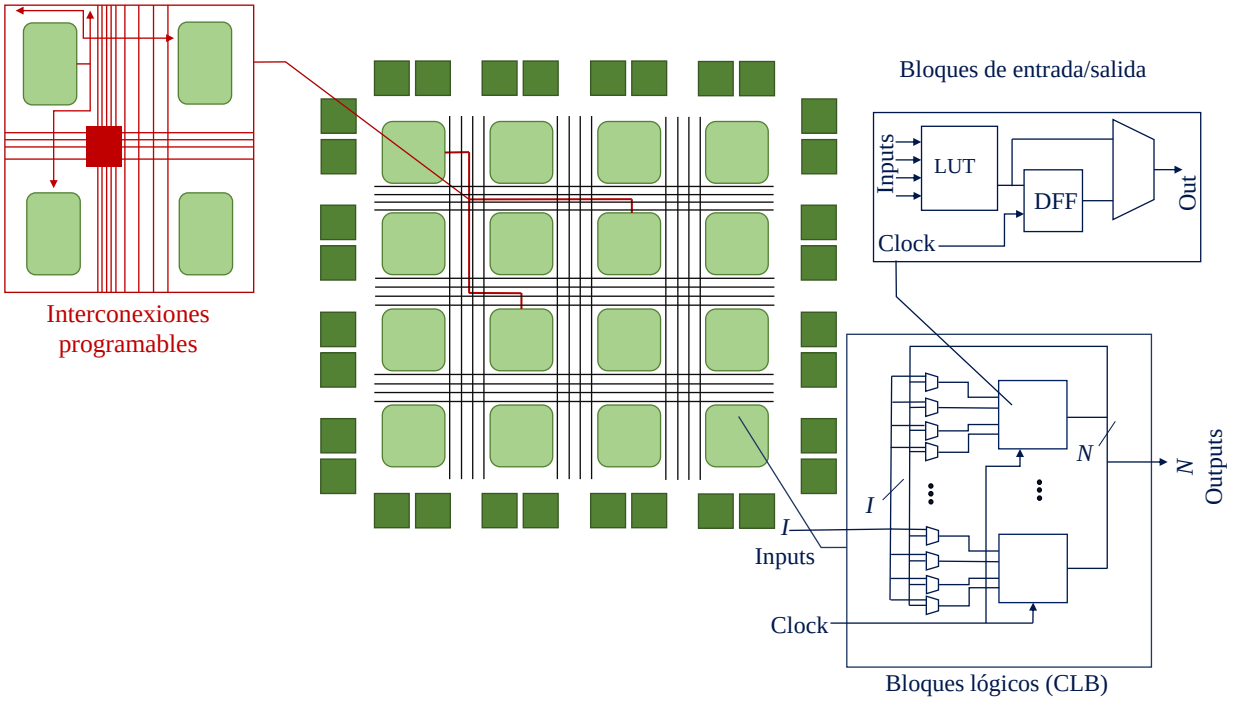


Figura 2.4: Arquitectura interna de un FPGA. Este circuito muestra a detalle la configuración interna de cada uno de los componentes principales que conforman este dispositivo, los CLBs, y los IOBs.

Dentro de cualquier familia de FPGA, todos los dispositivos compartirán una arquitectura de estructura común, pero cada dispositivo contendrá una cantidad diferente de lógica programable. Esto permite al usuario hacer coincidir sus requisitos lógicos con el dispositivo FPGA del tamaño correcto. Los dispositivos FPGAs están disponibles en dos o más tamaños de paquetes que permiten al usuario hacer coincidir los requisitos de IOBs de la aplicación con el paquete del dispositivo. Los dispositivos FPGAs también están disponibles en múltiples grados de velocidad y múltiples grados de temperatura, así como también en múltiples niveles de voltaje. Los dispositivos de mayor velocidad suelen ser un 25% más rápidos que los dispositivos de menor velocidad. Al diseñar con los dispositivos de menor velocidad, los usuarios pueden ahorrar costos, pero el mayor rendimiento de los dispositivos más rápidos puede minimizar el costo a nivel del sistema. Los FPGAs modernos suelen operar entre 100 y 500 MHz. En general, la mayoría de los diseños lógicos que no están destinados a arquitecturas FPGAs se ejecutarán en el rango de frecuencia más bajo, y los diseños destinados a FPGA se ejecutarán en el rango de

frecuencia media. Los diseños de frecuencia más alta suelen ser diseños *Digital Signal Processor* (DSP) construidos específicamente para aprovechar los bloques DSP.

La siguiente lista muestra las características de la arquitectura de un FPGA:

Interconexión Programable

Tejido a través de la estructura lógica del FPGA hay un conjunto de cables que se pueden conectar entre sí para dos bloques cualesquiera en un FPGA. Esto permite que el usuario construya redes lógicas arbitrarias. La arquitectura de los cables de interconexión varía de un FPGA a otro.

CLB

Una matriz de bloques lógicos configurables está integrada en la interconexión programable. Estos se denominan CLBs en los dispositivos Xilinx. En la actualidad, cada bloque lógico consta de una o más funciones lógicas programables implementadas como una tabla de búsqueda configurable de 4 a 6 bits, una cadena de transporte configurable y registros configurables. La palabra configurable se utiliza para indicar un bloque que puede configurarse a través de la memoria de configuración del FPGA y utilizarse como parte de la lógica del usuario. Por ejemplo, si el diseño del usuario requiere un registro con reloj habilitado, el registro está configurado para tener habilitado el reloj y conectado a la señal del reloj del usuario. La Figura 2.5 ilustra la arquitectura de un CLB, la cual consta de *Look Up Table* (LUTs), *Flip-Flops* (FFs) y las interconexiones.

IOB

Una de las capacidades clave de los FPGAs es que interactúan con señales externas de diferentes tipos como digitales y analógicas, mediante IOBs. Para admitir estos diversos requisitos, los FPGAs modernos contienen un bloque especial llamado IOB. Este bloque contiene potentes buffers para sacar señales externas de los FPGAs y los receptores de entrada, junto con registros para señales de IOB y habilitaciones de salida. Las IOBs suelen admitir señales con tecnología *Complementary Metal-Oxide-Semiconductor* (CMOS) de 1.2 a 3.3 V. Los IOBs se abstraen del diseño RTL y HDL del usuario y, por lo general, se configuran mediante un archivo de texto para especificar el estándar de señalización de cada IOB. Los IOBs suelen ser un recurso limitado, y los FPGAs están disponibles en varios tamaños para permitir que el usuario use FPGAs con diferentes cantidades de IOBs. La Figura 2.5 muestra el circuito de un CLB.

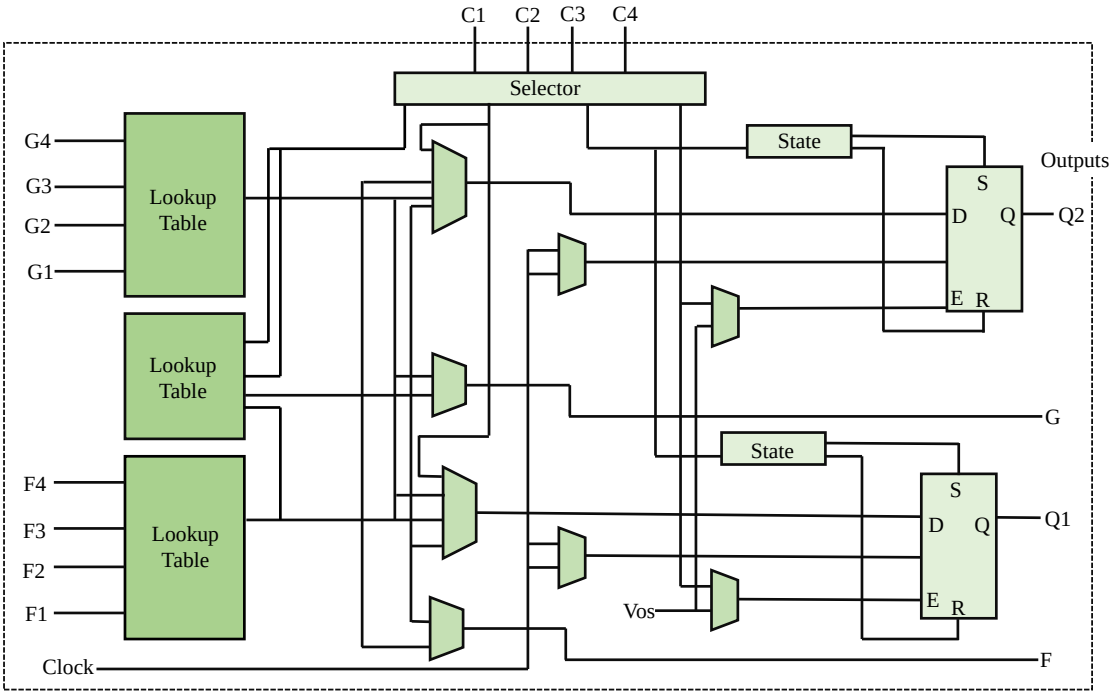


Figura 2.5: Circuito CLB.

2.2.2. AMD (Xilinx)

Xilinx fue una de las compañías líder en soluciones de lógica programable, incluyendo circuitos integrados avanzados, herramientas en software para diseño, funciones predefinidas y soporte de ingeniería. Xilinx fue la compañía que inventó los FPGAs y en la actualidad sus dispositivos ocupan más de la mitad del mercado mundial de los dispositivos lógicos programables [40]. A inicios del año 2022, la compañía fue comprada por AMD reuniendo así un conjunto altamente complementario de productos, clientes, mercados combinados y talento de clase mundial para crear el líder en computación adaptativa y de alto rendimiento de la industria, la Figura 2.6 muestra el logo actual de la compañía.

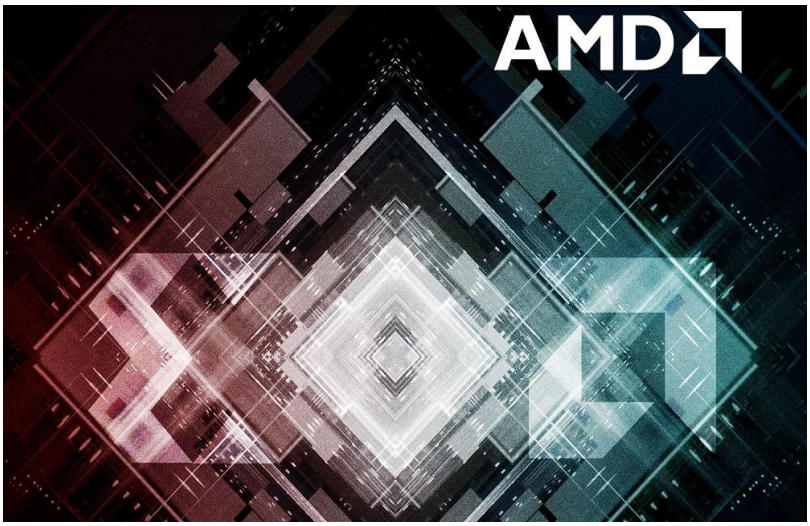


Figura 2.6: Nuevo Logo Xilinx y AMD [1].

Los dispositivos de esta compañía reducen de manera significativa el tiempo requerido para desarrollar aplicaciones en las áreas de computación, telecomunicaciones, redes, control industrial, instrumentación, aplicaciones militares y para el consumo general [1]. Las familias de CPLD XC9500 y XC9500XL proveen una larga variedad de dispositivos programables con características que van desde los 5 a 3.3 volts de operación, 36 a 288 macroceldas, 34 a 192 terminales de entrada y salida, y programación en sistema. Los dispositivos de las familias XC4000 y XC1700 de FPGA manejan voltajes de operación entre los 5 y 3.3 V, una capacidad de integración arriba de las 40,000 compuertas y programación en sistema.

En lo que se refiere a software, Xilinx desarrolló una importante herramienta llamada Foundation Series, que soporta diseños estándares basados en *Advanced Boolean Expression Language* (ABEL) y en VHDL. Esta herramienta se ofrece en versión estudiantil y profesional. De manera general, existe una amplia y variada gama de dispositivos lógicos programables disponibles en el mercado. La elección de uno u otro depende de los recursos con que cuenta el diseñador y los requerimientos del diseño [40].

Vivado Design Suite

Vivado Design Suite es un paquete de software producido por Xilinx para la síntesis y el análisis de diseños de lenguaje de descripción de hardware, que reemplaza a Xilinx ISE con características adicionales para el desarrollo de sistemas en un chip y síntesis de alto

nivel. Vivado representa una reescritura desde cero y un replanteamiento de todo el flujo de diseño (en comparación con Xilinx ISE).

Al igual que las versiones posteriores de Xilinx ISE, Vivado incluye el simulador lógico incorporado. Vivado también presenta síntesis de alto nivel, con una cadena de herramientas que convierte código C en lógica programable.

Vivado se presentó en abril de 2012 y es un entorno de diseño integrado con herramientas de nivel de sistema construidas sobre un modelo de datos escalable compartido y un entorno de depuración común. Vivado incluye herramientas de diseño de nivel de sistema electrónico para sintetizar y verificar algoritmos basados en C; empaquetamiento basado en estándares de algoritmos y RTL para su reutilización. Una versión gratuita WebPack Edition de Vivado proporciona a los diseñadores una versión limitada del entorno de diseño.

Componentes

El compilador Vivado High-Level Synthesis permite que los programas C, C++ y SystemC se apunten directamente a los dispositivos Xilinx sin necesidad de crear RTL manualmente. Vivado HLS se revisa ampliamente para aumentar la productividad del desarrollador y se confirma que admite clases, plantillas, funciones y sobrecarga de operadores de C++. Vivado 2014.1 introdujo soporte para convertir automáticamente kernels OpenCL a bloques algorítmicos para dispositivos Xilinx. Los núcleos OpenCL son programas que se ejecutan en varias plataformas de *Computer Processor Unit* (CPU), *Graphics Processing Unit* (GPU) y FPGA.

Vivado Simulator es un componente de Vivado Design Suite. Es un simulador de lenguaje compilado que admite scripts Tcl de lenguaje mixto, IP encriptada y verificación mejorada.

Vivado IP Integrator permite a los ingenieros integrar y configurar rápidamente bloques desde la biblioteca IP de Xilinx. El integrador también está ajustado para los diseños de MathWorks Simulink creados con System Generator for DSP y Vivado High-Level Synthesis de Xilinx.

Vivado Tcl Store es un sistema de secuencias de comandos para desarrollar complementos para Vivado y se puede usar para agregar y modificar las capacidades de Vivado. Tcl es el lenguaje de programación en el que se basa Vivado. Todas las funciones

subyacentes de Vivado se pueden invocar y controlar a través de scripts Tcl.

Soporte de dispositivos

Vivado admite la serie 7 de Xilinx y todos los dispositivos más nuevos (series UltraScale y UltraScale+) [3]. Para el desarrollo dirigido a dispositivos más antiguos, se debe usar el ISE de Xilinx ya descontinuado.

El conjunto de herramientas de diseño de Vivado contiene servicios que admiten todas las fases de los diseños de FPGA, desde la entrada del diseño, la simulación, la síntesis, el lugar y la ruta, la generación de flujo de bits, la depuración y la verificación, así como el desarrollo de software dirigido a estos FPGAs. Puede interactuar con el entorno de Vivado de varias maneras. Esto incluye una interfaz para usuarios interactivos, así como una interfaz de línea de comandos si prefiere usar el modo por lotes. Estos múltiples modos de interacción también se pueden combinar de diferentes maneras para adaptarse a las necesidades exactas de los usuarios [7].

Basys 3

El dispositivo FPGA Artix -7, con numero de serie XC7A35TCPG236-1 o mejor conocido cómo Basys 3, tiene una frecuencia de operación de 100MHz. Las especificaciones del XC7A35TCPG236-1 mencionadas en [39] muestran las características de este dispositivo, estos son:

- Cada CLB comprende cuatro segmentos interconectados. Un segmento incluye dos generadores de funciones LUT y dos elementos de almacenamiento, junto con lógica adicional. LUT es un generador de funciones basado en *Random Access Memory* (RAM) y es el principal recurso para implementar funciones lógicas. Además, las LUTs se pueden configurar como RAM distribuida o como un registro de desplazamiento de 16 bits.

- La salida de la LUT puede conectarse a la lógica del multiplexor, la lógica de acarreo y aritmética, o directamente a una salida CLB o al elemento de almacenamiento CLB.

- El elemento de almacenamiento CLB, que se puede programar como un flip-flop tipo D, proporciona un medio para sincronizar datos con una señal de reloj, entre

otros usos.

- Una almohadilla de reloj dedicado maneja un buffer de reloj global que se conecta a través de recursos de enrutamiento global a entradas de reloj en FFs y otros elementos sincronizados.

En la Figura 2.7 se muestran algunas características relevantes del dispositivo Basys 3.

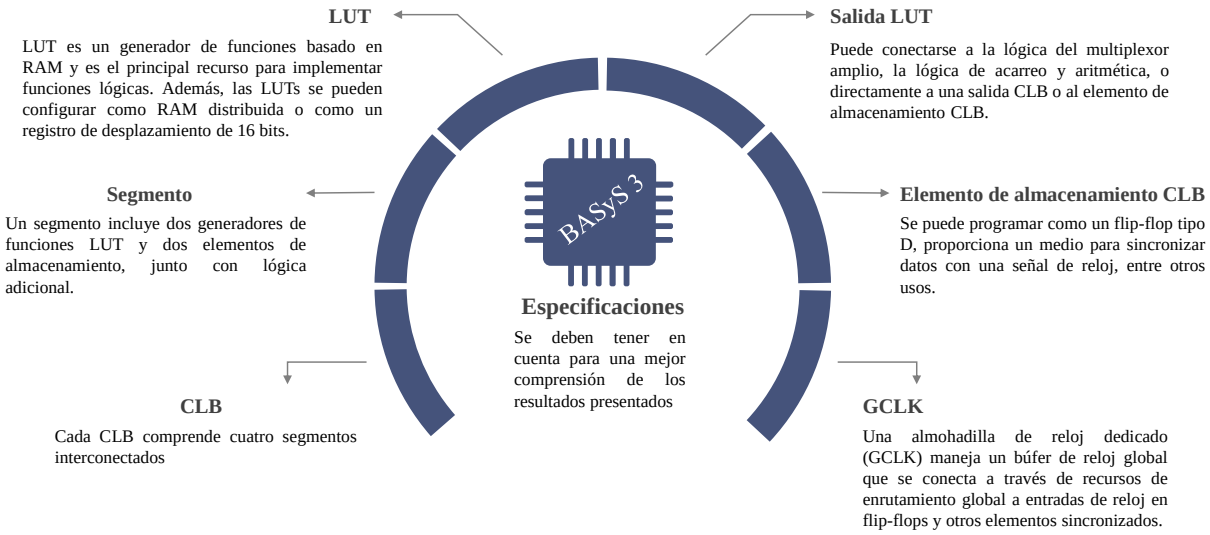


Figura 2.7: Especificaciones Basys 3.

2.2.3. Intel (Altera)

En junio del 2015 Intel adquiere a Altera, una de las compañías más importantes en el negocio de los semiconductores.

Intel® FPGAS para análisis de datos

Intel ha estado desarrollando marcos y bibliotecas de análisis de *Macrodata*, *great speed great volume and great variety of data* (big data) basados en infraestructura definida por software con bloques de construcción estándar abiertos. Desde plataformas de software abiertas listas para la empresa hasta componentes básicos de análisis, optimizaciones de tiempo de ejecución, herramientas, puntos de referencia y casos de uso, el software Intel® hace que los grandes datos y el análisis sean más rápidos, más fáciles y más

perspicaces. Los ejemplos incluyen marcos optimizados Apache Hadoop* y Spark*, Intel® Data Analytics Acceleration Libraries (Intel® DAAL) y BigDL: Distributed Deep Learning on Apache Spark* que se ejecuta sobre Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL- DNN).

Estos marcos y bibliotecas de Intel se están integrando con las opciones de aceleración Intel® FPGA. Los clientes pueden ejecutar aplicaciones no modificadas que, en tiempo de ejecución, pueden realizarse en la plataforma Intel® Xeon®, Intel® FPGA u otras plataformas Intel. Intel también proporciona marcos de aceleración con FPGAs, virtualización y seguridad de extremo a extremo. Intel ofrece aceleración de almacenamiento de datos no estructurados, NoSQL* y relacionales con Intel® FPGA individuales multifunción, que aceleran los flujos de datos, las redes, el acceso a datos y los algoritmos [18].

Quartus

El revolucionario software de diseño Intel Quartus Prime incluye todo lo que se necesita para diseñar en dispositivos lógicos programables complejos, desde la entrada del diseño y la síntesis hasta la optimización, la verificación y la simulación. Las capacidades dramáticamente aumentadas en dispositivos con varios millones de elementos lógicos están brindando a los diseñadores la plataforma ideal para cumplir con las oportunidades de diseño de próxima generación.

Software de diseño Quartus® Prime v20.0

El software de diseño Intel Quartus Prime v20.0 cuenta con una mayor reducción de tiempo de compilación y reducción en los requisitos de memoria virtual. Intel presenta el software Quartus Prime v20.0 que ofrece mejoras a través de las tres áreas clave que preocupan en gran medida a los diseñadores: el rendimiento, la productividad y la facilidad de uso. La edición v20.0 del software Quartus Prime Pro Edition de Intel admite dispositivos Intel Stratix® 10 TX, MX, SX, y GX. Permite acelerar el desarrollo de FPGAs con tiempos de compilación más rápidos de los diseños Intel Stratix 10 en v20.0 en comparación con versiones anteriores. Los diseños más grandes de Intel Stratix 10 muestran una mayor reducción de tiempo de compilación.

El soporte de análisis concurrente proporciona la capacidad de analizar los resultados

de un diseño de compilación en marcha. Además, la reconfiguración parcial permite a los usuarios reprogramar dinámicamente una porción del FPGA mientras que el diseño FPGA restante sigue funcionando. Con la edición v20.0 del software Quartus Prime Pro Edition de Intel, los usuarios pueden acelerar sus aplicaciones con las herramientas de programación FPGA Intel en la nube para programar FPGA en un entorno informático de alto rendimiento suministrado por Nimbix. Más información en la página de web de servicios de nube [19].

Características

- Tiempo de compilación más rápido.
- Reducción de la memoria virtual pico.
- Análisis simultáneos.
- La reconfiguración parcial permite a los usuarios configurar dinámicamente una porción del FPGA.

Cyclone

La familia FPGA Cyclone amplía el liderazgo de la serie FPGA Intel® Cyclone al proporcionar FPGA de bajo consumo con opciones de transceptor. Ideal para aplicaciones sensibles a los costos y de gran volumen, los FPGAs Cyclone le permiten satisfacer los crecientes requisitos de ancho de banda.

Reduce el consumo eléctrico

Construidos sobre un proceso optimizado de bajo consumo de 60 nm, los FPGAs Cyclone reducen el voltaje del núcleo, lo que reduce la energía total en un 25 por ciento en comparación con el predecesor.

Los FPGAs Cyclone® de Intel están optimizadas para el consumo de energía más bajo, ayudándole a administrar mejor los requisitos térmicos. Como resultado, puede reducir o eliminar los costos de enfriamiento del sistema y también extender la vida útil de la batería para aplicaciones portátiles.

2.3. Conclusión

La información mostrada en este Capítulo muestra las características de los dispositivos de descripción de hardware, en específico de los dispositivos FPGAs de diferentes fabricantes, estos son dispositivos con un alto rango de procesamiento de datos, por ejemplo dispositivos como el FPGA XC7A35TCPG236-1 o mejor conocido cómo Basys 3 o la tarjeta de desarrollo Cyclone que tienen ventajas, en unidad de procesamiento, calidad y precio que los hacen candidatos para albergar el desarrollo de un sistema que permita resolver algunos de los problemas que presenta el análisis de datos de IoT presentado en el Capítulo 1.

Capítulo 3

Programación Genética Geométrica Semántica

Evolutionary Computation (EC) es un área de investigación dentro de las ciencias de la computación. Como sugiere el nombre, es un área especial de la computación, que se inspira en la teoría de la evolución natural propuesta por Darwin. No sorprende que algunos hayan elegido la evolución natural como fuente de inspiración: el poder de la evolución en la naturaleza es evidente en las diversas especies que componen nuestro mundo, cada una diseñada para sobrevivir bien en su propio nicho. La metáfora fundamental de la EC relaciona la evolución natural con un estilo particular de resolución de problemas: el de prueba y error.

Considerando la evolución natural simplemente como un entorno lleno de una población de individuos que se esfuerzan por sobrevivir y reproducirse. La aptitud de estos individuos está determinada por el entorno y se relaciona con qué tan bien logran alcanzar sus objetivos. En otras palabras, representa sus posibilidades de supervivencia.

En el contexto de un ensayo y error estocástico también conocido como proceso de resolución de problemas de estilo generar y probar, tenemos una colección de soluciones candidatas. Su calidad determina la posibilidad de que se mantengan y se utilicen como padres para construir otra población de soluciones candidatas en la Figura 3.1 se puede apreciar un ejemplo básico de este proceso [9].

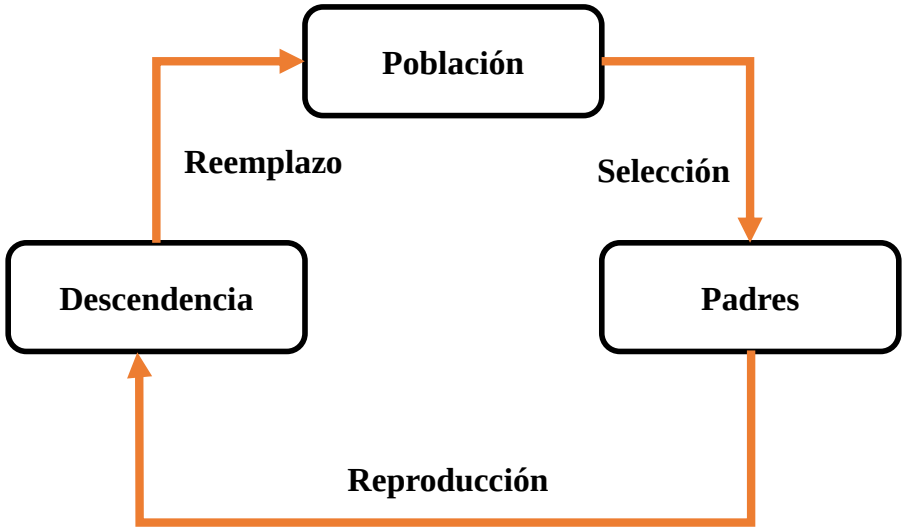


Figura 3.1: Esquema del proceso evolutivo, dada una población inicial de individuos se selecciona una cantidad finita de padres, los cuales son reproducidos para obtener descendencia (hijos), luego se someten a una evaluación la cual permite reemplazar a la primera población con los individuos más aptos encontrados.

El desarrollo de potentes técnicas de búsqueda y optimización es de gran importancia en la ciencia y la ingeniería, particularmente en el mundo actual que requiere que los investigadores y profesionales aborden una variedad de problemas desafiantes del mundo real a medida que la tecnología se convierte en un aspecto cada vez más importante de la vida cotidiana. Hay dos campos bien establecidos y ampliamente conocidos que abordan estos problemas: (i) técnicas tradicionales de optimización numérica y (ii) heurísticas bio-inspiradas comparativamente recientes, como algoritmos evolutivos y programación genética. Ambos campos han desarrollado enfoques con sus fortalezas y debilidades únicas, lo que les permite resolver algunos problemas desafiantes mientras que a veces fallan en otros.

3.1. Programación Genética

GP es una técnica de computación evolutiva que resuelve problemas automáticamente sin que el usuario tenga que conocer o especificar la forma o estructura de la solución de antemano. En el nivel más abstracto, GP es un método sistemático e independiente del

dominio para hacer que las computadoras resuelvan problemas automáticamente a partir de una declaración de alto nivel de lo que se debe hacer.

En GP se evoluciona una población de programas. Es decir, generación tras generación, GP transforma estocásticamente poblaciones de programas en poblaciones de programas nuevas, con suerte mejores, la Figura 3.2 representa el proceso básico de un algoritmo GP. GP, como la naturaleza, incluye proceso aleatorio y nunca puede garantizar buenos resultados. La aleatoriedad esencial de GP, sin embargo, puede llevarlo a escapar de óptimos locales por las cuales los métodos deterministas pueden ser capturados. Al igual que la naturaleza, GP ha tenido mucho éxito en la evolución de individuos nuevos e inesperados.

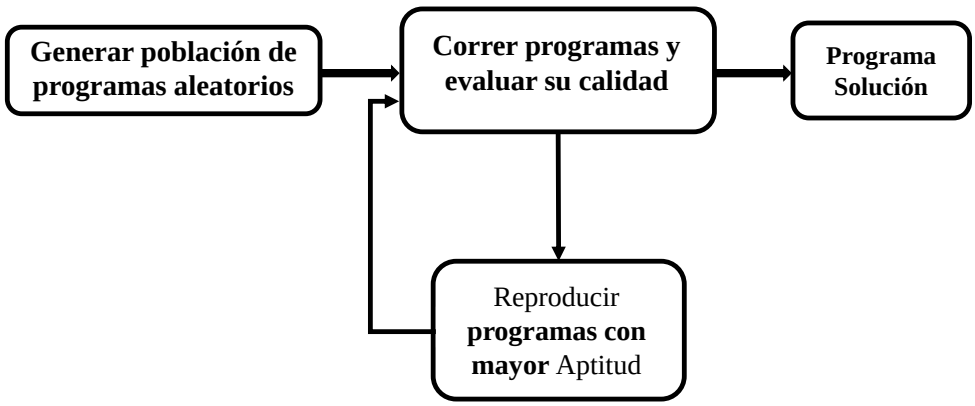


Figura 3.2: Proceso básico de GP, en el cual se genera una población de programas, luego por un numero n de generaciones se corren estos programas, se evalúa su calidad, posteriormente se reproducen los individuos más aptos y por último se selecciona al individuo más apto.

3.1.1. Representación de árbol

En algunas técnicas evolutivas de aprendizaje máquina como GP los individuos son usualmente representados por árboles. La representación de árbol permite crear

cualquier expresión a través de un conjunto de primitivas, el cual está conformado por un subconjunto de Funciones F y Terminales T. Por ejemplo, en la Figura 3.3 se muestra la representación de la función: $(x+x)/(x-(3*y))$ [27].

Las variables y constantes del programa $(x, y, 3)$ son las hojas del árbol. Mientras que los nodos internos son las operaciones aritméticas o funciones $(*, /, +, -)$.

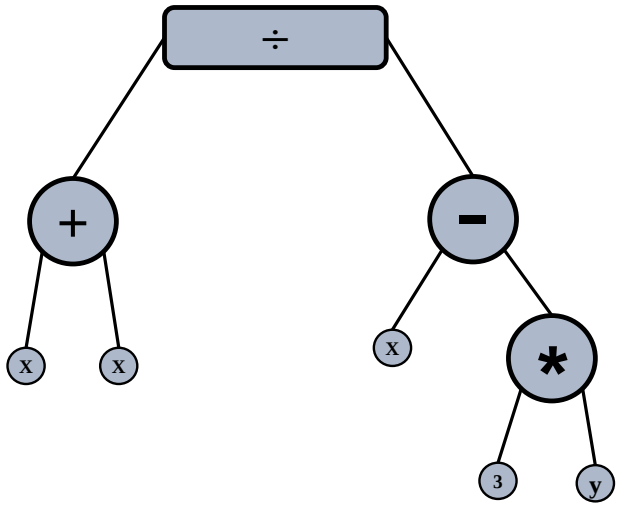


Figura 3.3: Función matemática representada en un árbol, se representa la función $(x+x)/(x-(3*y))$ en forma de árbol de sintaxís.

Para describir programas en GP se tienen que definir el conjunto de funciones y terminales. El conjunto de terminales puede estar compuesto por entradas externas de programas. Por lo general, toman las variables nombradas (por ejemplo, x, y). Frecuentemente se busca tener una terminal constante fija, generalmente se crean como parte del proceso de inicialización de la población. Esto se logra mediante la introducción de una terminal que representa una constante aleatoria efímera. El conjunto de funciones utilizado en GP generalmente está impulsado por la naturaleza del dominio del problema. En un problema numérico simple, por ejemplo, el conjunto de funciones puede consistir simplemente en las funciones aritméticas $(+, -, /, *)$. Sin embargo, se pueden usar todo tipo de otras funciones y construcciones que generalmente se encuentran en los programas de computadora, en la Figura 3.4, se muestra un ejemplo de terminales y funciones.

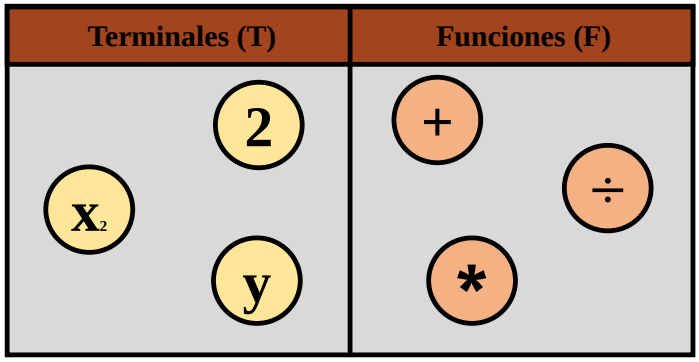


Figura 3.4: Ejemplo de terminales y funciones

GP es una técnica que hace uso de distintos subprocesos para poder cumplir con el ciclo evolutivo. A continuación se realiza un descripción de estos subprocesos de GP [27].

Inicialización

Esta fase es la encargada de generar todos los individuos que existirán en la población inicial. Donde cada individuo estará compuesto por un conjunto de Funciones (F) y terminales (T).

Evaluación

Este proceso depende del problema dado a GP, sin embargo, a cada uno de los individuos se le debe evaluar. La evaluación servirá para determinar la calidad de cada individuo, con respecto al objetivo dado. Para poder calcular esta evaluación llamada también función aptitud, se necesita:

- Evaluar el cromosoma en cada uno de los datos del conjunto de entrenamiento, este resultado comúnmente se le conoce como semántica.
- Calcular la función objetivo cuyo resultado es conocido como la aptitud del individuo.
- Obtener el error esperado, esto es calcular la diferencia entre la salida obtenida y la salida esperada con algún estimador de error como el error cuadrático medio por mencionar alguno.

Mutación

El operador de mutación más utilizado en GP es la mutación de árbol, que funciona seleccionando un nodo dentro de un individuo de manera aleatoria para reemplazarlo con un subárbol generado de manera aleatoria. La creación de este subárbol debe tener la misma configuración con la que se creó la población inicial.

Cruce

El cruce genera nuevos individuos intercambiando material genético a través de dos padres seleccionados. La forma de cruce más utilizada en GP es el cruce de subárbol. Dados dos padres, el cruce de subárbol selecciona un punto de cruce aleatoriamente en cada árbol padre. La descendencia se crea al reemplazar el subárbol conectado en el punto de cruce en una copia del primer padre con una copia del subárbol conectado en el punto de cruce en el segundo padre.

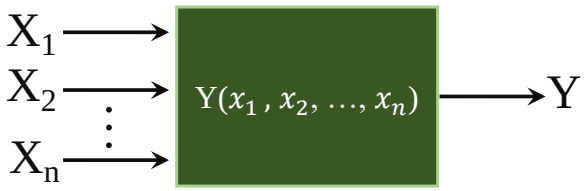
Selección

El operador de selección funciona de la siguiente manera, de todo el conjunto de individuos en la población, únicamente un subconjunto será seleccionado probabilísticamente basados en su aptitud para generar los individuos de la siguiente generación. Esto garantiza que los mejores individuos pueden llegar a tener más descendientes. En la literatura se reportan diferentes técnicas de selección, la técnica comúnmente utilizada es selección por torneo. En este tipo de selección, se elige una cantidad aleatoria de individuos de la población. Estos se comparan entre sí, a través de la calidad de su aptitud y se elige a los mejores individuos como padres. La selección de torneo solo analiza qué solución es mejor que otra, no necesita saber cuánto mejor [27].

3.1.2. Regresión simbólica

El análisis de regresión simbólica, es una técnica estadística que busca deducir el patrón de una serie de datos o investigar la relación estadística entre una variable dependiente Y y una o más variables independientes, el resultado es una expresión algebraica del tipo $Y = F(X\_1 , X\_2, \ldots , X\_n)$. El tipo de análisis de regresión más usual es la regresión lineal que tiene una variable independiente $Y = F(x)$.

La regresión simbólica (una aplicación de GP) tiene el mismo objetivo de la regresión lineal pero con un espectro mucho mayor de búsqueda y menos restricciones: Dados los datos, buscará el patrón (expresión simbólica) que identifique el comportamiento de estos accediendo a todo tipo de funciones y combinaciones algebraicas un ejemplo se puede observar en la Figura 3.5 [25].



X_1	X_2	…	X_n	Y
0.324	0.344	…	0.424	2.224
0.124	0.144	…	0.394	0.824
0.228	0.248	…	0.595	3.224
⋮	⋮		⋮	⋮
0.098	0.588	…	0.335	1.982

Figura 3.5: Ejemplo de Regresión Simbólica.

3.2. Programación Genética Geométrica Semántica

En 2012, se introdujeron nuevos operadores genéticos para GP, conocidos como semánticos geométricos [23], que proporcionan un paisaje de aptitud marcado por la ausencia de soluciones locales subóptimas (paisaje de aptitud unimodal) para todo tipo de problema de aprendizaje, donde la aptitud es una diferencia (o error) entre un conjunto de valores objetivo y el conjunto de salidas calculadas (esta es la situación típica, por ejemplo, de clasificación o problemas regresivos).

Desde sus inicios, GSGP ha despertado el interés de los investigadores de GP, quienes probablemente se sientan atraídos por la ventaja potencial en términos de capacidad evolutiva proporcionada por un panorama de aptitud unimodal.

Al mismo tiempo, los investigadores han intentado superar un significativo inconveniente en los operadores geométricos semánticos, es decir, el rápido crecimiento de los individuos en términos de tamaño de código. Debido al tamaño del individuo, GSGP es complicado de usar en la práctica. La solución a esta dificultad se introdujo en 2013, con una nueva implementación de GSGP que lo hizo no solo utilizable, sino también muy eficiente [5, 35].

La Figura 3.6 muestra el proceso básico de un algoritmo GSGP, al igual que en GP primero se genera una población de programas o individuos aleatorios, posteriormente se realiza una primera evaluación en la cual se obtiene la semántica de los individuos, una vez obtenida la semántica se realiza un ciclo de n repeticiones donde se evalúa la aptitud de los individuos mientras mutan cada generación y por último se selecciona al individuo más apto.

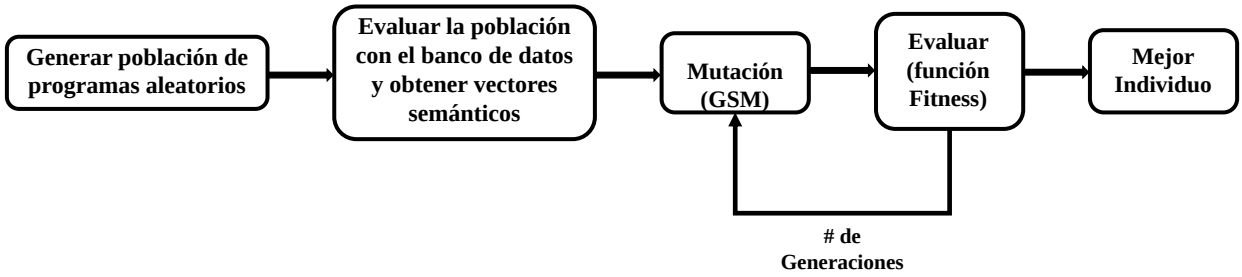


Figura 3.6: Ejemplo del proceso GSGP, primero se genera una población de programas o individuos aleatorios, posteriormente se realiza una primera evaluación en la cual se obtiene la semántica de los individuos, una vez obtenida la semántica se realiza un ciclo de n repeticiones donde se evalúa la aptitud de los individuos mientras mutan cada generación y por último se selecciona al individuo más apto.

Espacio Semántico

La semántica de un individuo o programa P se define como la salida del programa, donde, para cada vector de entrada $\overrightarrow{X_i}$ devuelve el valor escalar $P(\overrightarrow{X_i})$ en otras palabras la semántica de P es $\overrightarrow{S_P} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), ..., P(\overrightarrow{x_n})]$. En este contexto, se puede definir la semántica de un individuo como un punto en un espacio multidimensional, denominado *espacio semántico*.

Los nuevos operadores implementados en GSGP son capaces de inducir un efecto

semántico a través de operaciones sintácticas, y permiten a GSGP explorar un paisaje cónico semántico de aptitud, que puede optimizarse eficientemente mediante la búsqueda evolutiva. Estos operadores permiten definir modificaciones en la sintaxis de individuos GP que tienen una correspondencia precisa sobre su semántica, así pues, tienen la propiedad de inducir un paisaje de aptitud unimodal para cualquier problema que consista en generar una coincidencia entre los datos de entrada con los resultados objetivo.

La Figura 3.7 muestra los dos espacios de búsqueda, del lado izquierdo se tiene el espacio de búsqueda tradicional de GP, el cual se realiza mediante la manipulación de la sintaxis de los individuos. Del lado derecho, se puede observar el comportamiento de un programa una vez que se ejecuta, o más particularmente el conjunto de sus valores de salida sobre datos de entrenamiento de entrada.

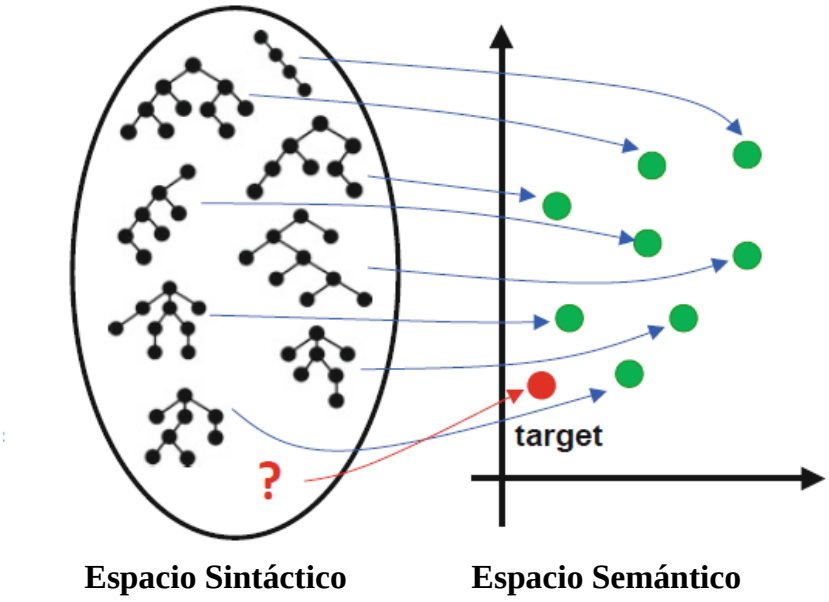


Figura 3.7: Ejemplo de espacios, el espacio sintáctico, en el que los individuos están representados por sus estructuras, y el espacio semántico, en el que los individuos están representados por puntos, que son su semántica. El espacio semántico se representa en 2D, lo que corresponde al caso irreal en el que solo existen dos instancias de entrenamiento [36].

GSGP utiliza un esquema generacional con selección por torneo, cruce y mutación

respectivamente que siempre están a disposición del usuario. GSGP básicamente tiene similitudes con un GP tradicional o un *Evolutionary Algorithm* (EA), por el hecho de solo integrar nuevos operadores genéticos, el resto del algoritmo funciona de manera similar a los tradicionales.

Mutación Geométrica Semántica

El objetivo de la *Geometric Semantic Mutation* (GSM) es el de generar una transformación en la sintaxis de los individuos GP que tenga el mismo efecto en su semántica que la mutación de caja. La situación se ejemplifica en la Figura 3.8. Representa una cadena de posibles individuos que podrían generarse aplicando GSM varias veces, con su correspondiente semántica. Dado que la semántica de los individuos generados por mutación puede ser cualquier punto dentro de una caja de un lado dado centrado en la semántica misma, GSM tiene siempre la posibilidad de crear un individuo cuya semántica sea más cercana al objetivo, en la Figura 3.8 se representa por el símbolo de estrella. Como consecuencia directa, el panorama del fitness no tiene soluciones localmente óptimas [36].

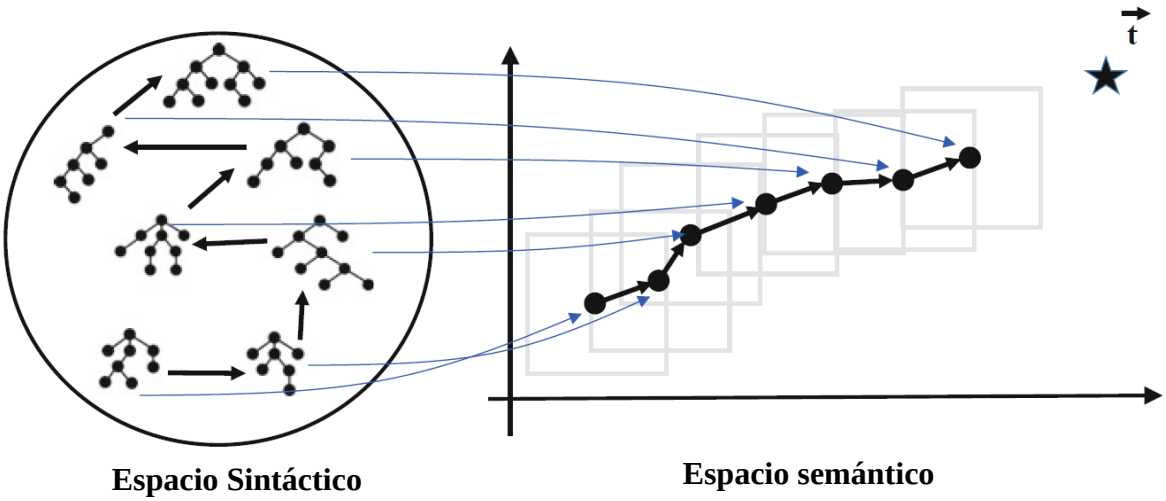


Figura 3.8: Representación gráfica simple de individuos generados por la GSM en el espacio sintáctico, de una cadena de soluciones [36].

No es difícil tener una intuición del hecho de que GSM es una perturbación "débil" del elemento correspondiente en la semántica del vector semántico de padre. De manera

informal, se define esta perturbación como "débil"porque está dada por una expresión aleatoria centrada en cero (la diferencia entre dos árboles aleatorios). Sin embargo, se puede ajustar el "paso"de la mutación y, por lo tanto, la importancia de esta perturbación [36].

La GP tradicional utiliza operadores que manipulan la sintaxis de programas ignorando su efecto en la salida del programa o la semántica; es decir, la búsqueda es ciega a los efectos sobre la aptitud. Por el contrario, GSGP busca directamente en el espacio de la semántica subyacente de los programas por medio de sus operadores especiales definidos en [24].

La GSM es definida en [24] como una perturbación aleatoria de las coordenadas de un individuo en el espacio semántico, es conocida también como mutación de bola y consiste en la perturbación de la semántica de un individuo donde su mutación se encuentra dentro de un circulo o radio centrado en el padre, la Figura 3.9 ejemplifica este tipo de mutación.

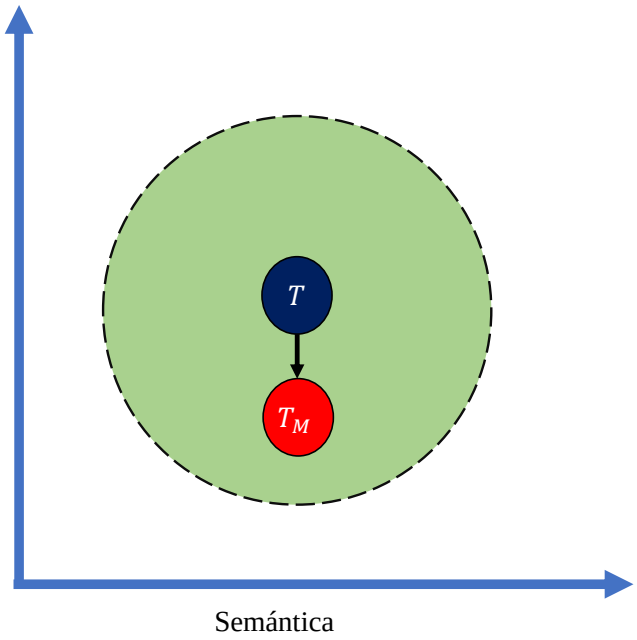


Figura 3.9: Mutación de bola.

Este tipo de mutación geométrica semántica se define en particular cómo:

$$T_M = T + ms \cdot (T_{R1} - T_{R2}) \tag{3.1}$$

donde T es el individuo seleccionado, T_{R1} y T_{R2} son 2 individuos aleatorios y ms es un paso de mutación.

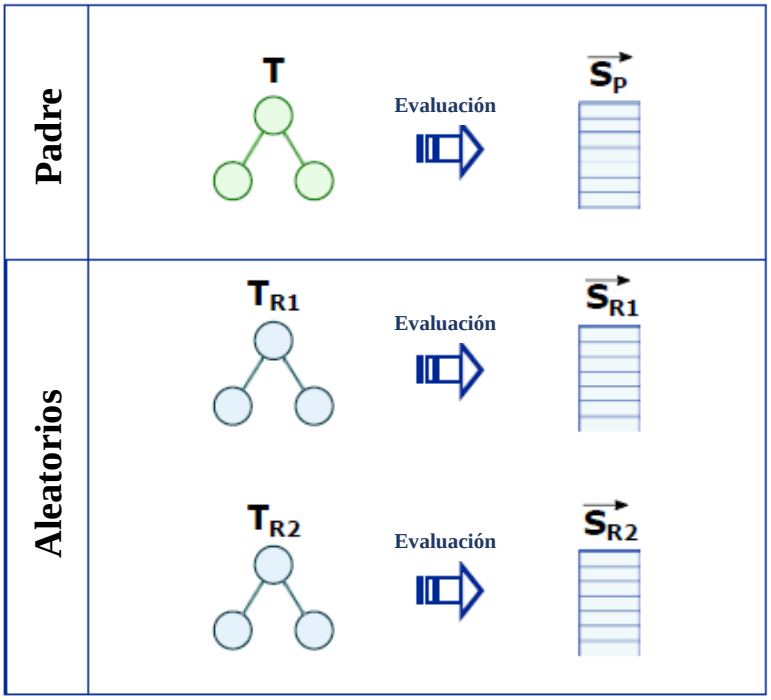


Figura 3.10: Vectores semánticos de individuos usados en mutación.

Si sustituimos los individuos de la Ecuación 3.1 por sus vectores semánticos como lo muestra la Figura 3.10 la Ecuación 3.1 puede reescribirse como:

$$\overrightarrow{S_M} = \overrightarrow{S_P} + ms \cdot (\overrightarrow{S_{R1}} - \overrightarrow{S_{R2}}) \tag{3.2}$$

donde $\overrightarrow{S_P}$ es el vector semántico del individuo seleccionado, $\overrightarrow{S_{R1}}$ y $\overrightarrow{S_{R2}}$ son los vectores semánticos de los 2 individuos aleatorios y ms es un paso de mutación.

3.2.1. Paso de mutación óptimo

El *Optimal Mutation Step* (OMS) puede desempeñar un papel en la reducción del riesgo de sobreajuste. Cuando se trata de aprender de manera más eficiente, la GSM se puede mejorar adaptando su paso. Es posible calcular de forma determinista el OMS para cada aplicación del operador.

Similar al enfoque descrito en [13] propone una modificación al operador GSM utilizando un OMS. La operación de mutación en la Ecuación 3.2, es de hecho una combinación lineal del vector semántico del padre $\overrightarrow{S_T}$ y y un vector semántico aleatorio $\overrightarrow{R_I}$ que es a su vez la diferencia entre los vectores semánticos de dos individuos aleatorios $\overrightarrow{T_{R1}}$ y $\overrightarrow{T_{R2}}$. Dado que $\overrightarrow{t}$ es el vector semántico del objetivo deseado, la ecuación se convierte en:

$$\overrightarrow{t} = \overrightarrow{S_P} + ms \cdot \overrightarrow{RI} \tag{3.3}$$

Resolviendo para ms,

$$ms = (\overrightarrow{t} - \overrightarrow{S_P}) \cdot \overrightarrow{RI}^{-1} \tag{3.4}$$

donde $\overrightarrow{RI}^{-1}$ es la pseudo-inversa del vector semántico $\overrightarrow{RI}$.

La implementación del OMS requiere el cálculo de la pseudo-inversa del vector $\overrightarrow{RI}$. En [13] se recomienda aplicar la pseudo-inversa de Moore-Penrose, pero como se menciona en [15], $\overrightarrow{RI}$ es siempre un vector de *m x 1*(siendo *m* el número de casos de entrenamiento), no se requiere un método general para calcular la pseudo-inversa para una matriz *m x 1*. En su lugar, se puede calcular la pseudo-inversa de Moore-Penrose para el caso especial cuando:

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \tag{3.5}$$

entonces la pseudo-inversa A^+ de A esta dada por

$$A^+ = \frac{1}{||A||^2} A^* = \frac{1}{||A||^2} (\overline{a_1}, \ldots, \overline{a_1}) \tag{3.6}$$

donde,

$$\frac{1}{||A||} = \sqrt{|a_1|^2 + \cdots + |a_m|^2} \tag{3.7}$$

Considerando a todos los elementos de RI como valores reales, entonces $\overline{a_i} = a_i$, y la Ecuación 3.6 puede ser reescrita como:

$$A^+ = \frac{(a_1, \ldots, a_m)}{|a_1|^2 + \cdots + |a_m|^2} = \frac{A^T}{A^T \cdot A} \tag{3.8}$$

3.3. Conclusión

En este Capítulo se revisó el *estado-del-arte* sobre programación genética y programación genética geométrica semántica, estas técnicas de computación evolutiva tienen un mejor rendimiento en problemas de regresión simbólica, en comparación con otros algoritmos de aprendizaje máquina, particularmente los basados en la teoría de la evolución, sin embargo, su principal desventaja se encuentra en el tiempo de ejecución del algoritmo y en la dependencia de una computadora independientemente del lenguaje de programación.

Debido a estos inconvenientes, se propone el uso de dispositivos de excelente capacidad de procesamiento como lo son los dispositivos HDLs, en específico los FPGAs, gracias a las ventajas de estos, es posible el desarrollo de un sistema basado en algoritmo GSGP implementado en un dispositivo FPGA que permita la solución a todos los inconvenientes descritos anteriormente. En el siguiente Capítulo se explica a detalle la implementación propuesta para el procesador dedicado GSGP-VHDL.

Capítulo 4

Propuesta de Implementación

GSGP tiene un mejor rendimiento en comparación con otros algoritmos de aprendizaje máquina, particularmente los basados en la teoría de la evolución, sin embargo, su principal desventaja se encuentra en el tiempo de ejecución del algoritmo el cual puede llegar a tardar días en encontrar una solución y en la dependencia de una computadora independientemente del lenguaje de programación [5]. Por lo tanto, se desarrolló un sistema basado en dispositivos de gran capacidad de procesamiento como lo son los FPGAs para resolver esta desventaja. Este Capítulo presenta la aportación de este trabajo al área de GP, se propone un algoritmo paralelo de GSGP para realizar la búsqueda de soluciones en el espacio semántico. En primer lugar, se muestran las etapas que componen al algoritmo propuesto, donde cada una de ellas se explica a detalle.

El algoritmo de GSGP consta principalmente de 5 etapas: La creación de la población inicial, Evaluación inicial, Mutación, Evaluación de aptitud y por último la selección del mejor individuo. La Figura 4.1 muestra a grandes rasgos las etapas del algoritmo GSGP.

Primeramente se crea una población de individuos y aprovechando la ventaja de los procesos concurrentes en VHDL se realizan las siguientes etapas a cada uno de estos individuos T_1 T_2, ..., T_n, empezando por una evaluación inicial, luego de obtener la semántica, se realiza la mutación semántica n cantidad de veces, posteriormente, se realiza la evaluación de la aptitud y por último, se compara la calidad de toda la población para elegir al mejor individuo.

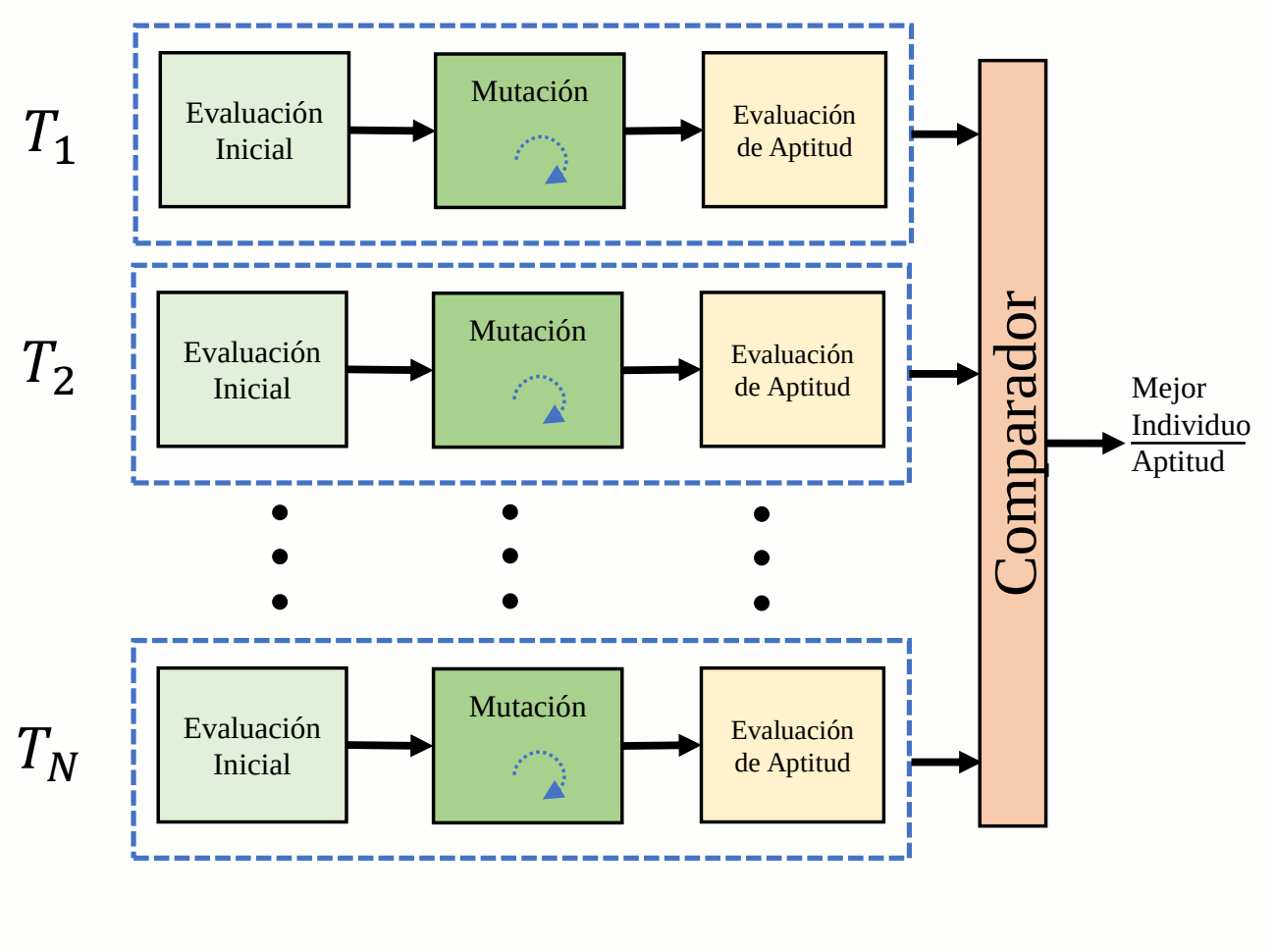


Figura 4.1: Propuesta de implementación. Mediante procesos concurrentes en VHDL, se crea una población de individuos, posteriormente una evaluación inicial, luego de obtener la semántica, se realiza la mutación semántica *n* veces, enseguida se evalúa la aptitud y por último, se compara la calidad de toda la población para elegir al mejor individuo.

4.1. Población inicial

En [14], se describe un generador automático de árboles de sintaxís en un FPGA. Se menciona que cuando se implementa un árbol en software, es habitual hacer uso de punteros o referencias para vincular los nodos, teniendo en cuenta que los punteros son una herramienta poderosa para realizar un seguimiento de todas las conexiones de un nodo, pero desafortunadamente las referencias o punteros no son compatibles con los lenguajes de descripción de hardware, por consiguiente se desarrolló la implementación mostrada en la Figura 4.2, cada nodo del árbol se representa como un elemento de una

matriz, las dimensiones de dicha matriz están representadas por la profundidad del árbol y su número de terminales u hojas.

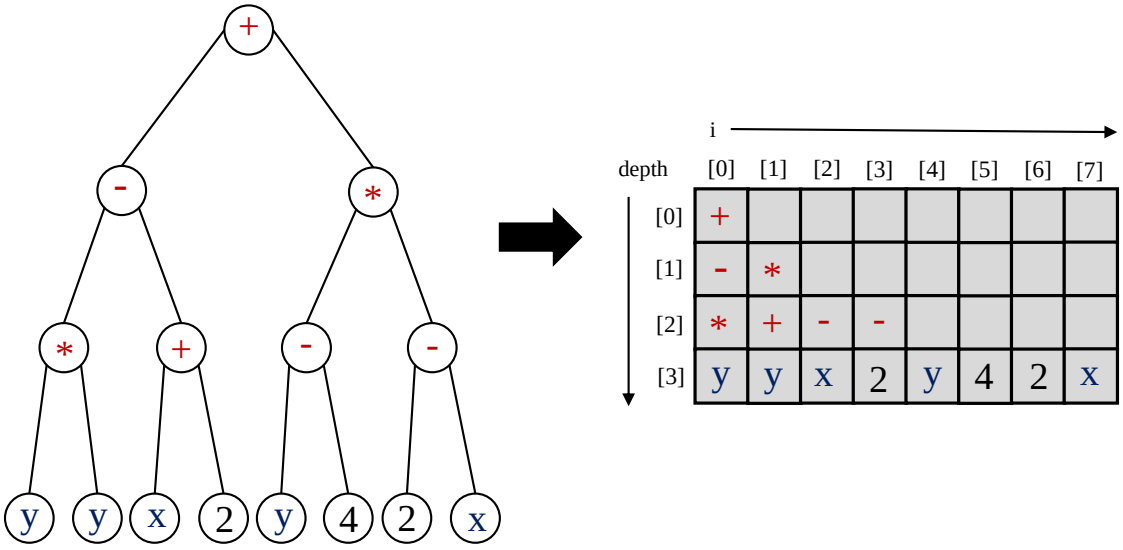


Figura 4.2: Representación de árbol como matriz.

Cada uno de los valores posibles que puede tomar cada nodo está definido en un conjunto de primitivas. La Tabla 4.1, muestra el conjunto de primitivas que incluye las funciones aritméticas $(+, -, *, /)$, variables (x, y) y constantes $(0, 1, 2, 9)$. Cada elemento del conjunto está asociado con un identificador (ID), por lo que cada ID representa solo a un elemento de las primitivas en la implementación. Estas primitivas o ID son por definición vectores o arreglos de datos tipo *std_logic* de 4 bits.

Tabla 4.1: Definición de primitivas.

OP/Var/Const	ID
+	0000
-	0001
*	0010
/	0011
x	0100
y	0101
0	0110
1	0111
2	1000
3	1001
4	1010
5	1011
6	1100
7	1101
8	1110
9	1111

Por lo tanto, el conjunto que se obtiene al sustituir los nodos de la matriz mostrada en la Figura 4.2 por las primitivas mostradas en la Tabla 4.1, se obtiene un conjunto de arreglos de vectores o una matriz de arreglos de vectores que se muestra en la Figura 4.3.

0000	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -
0001	0010	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -
0010	0000	0001	0001	- - - -	- - - -	- - - -	- - - -
0101	0101	0100	1000	0101	1010	1000	0100

Figura 4.3: Conjunto de arreglos de vectores.

Cómo podemos observar en la Figura 4.3, es una matriz del tipo *mxn* la cual se muestra en la Figura 4.4. En [14], se demostró que el diseño de un árbol de sintaxis en un matriz era la mejor opción en cuanto a menor consumo de recursos en el dispositivo FPGA, por lo que, en este caso, hay elementos que no forman parte de la implementación del árbol ya que no son utilizados.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Figura 4.4: Matriz *m x n*.

donde: a_{mn} : Es un vector tipo std_logic_vector de dimensión *mn*.

En el Capítulo 2, se mostró cómo definir un tipo de dato como una matriz, para luego crear objetos pertenecientes a este nuevo tipo de dato, la Figura 4.5 muestra el código VHDL que permite definir los datos para el árbol de la Figura 3.3.

```
TYPE matrix IS ARRAY (0 to 3,0 to 7) OF STD_LOGIC_VECTOR(3 downto 0);

SIGNAL TREE: matrix;
```

Figura 4.5: Definición de tipo de dato matriz y su asignación a una señal.

En la Figura 4.6, se observa la simulación de un árbol de sintaxis diseñado mediante una matriz, en la simulación se muestran 3 señales, (1) *clk* representa la señal del reloj de 100MHz de la tarjeta Basys 3, (2) *seed* de 7 bits, el cual es indispensable para generar un árbol de sintaxis pseudo-aleatorio y (3) la matriz *TREE* que representa el árbol de sintaxis. En la simulación se observa un cursor en color rojo indicando que la generación del árbol ha sido completada y señala aproximadamente 1240 nS, tiempo utilizado para generar el árbol, sin embargo, el simulador necesita 1000 nS para inicializarse, por lo tanto, en 240 nS o 24 ciclos de reloj se genera el árbol.

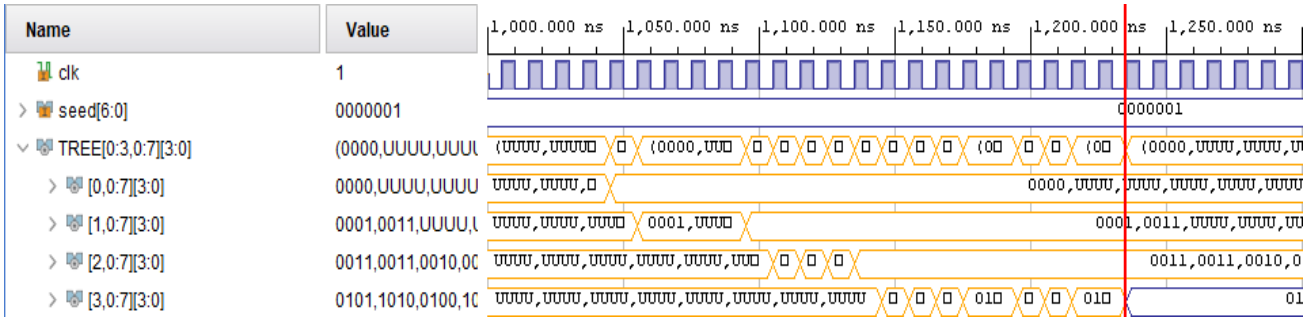


Figura 4.6: Simulación de un árbol en una matriz. La señal periódica en color morado, representa el reloj con frecuencia de 100MHz y periodo de 10 nS.

La Tabla 4.2, resume el consumo de recursos de hardware para la implementación del árbol de sintaxis en una matriz cuya simulación se muestra en la Figura 4.6, en la Tabla se observa que se necesitan 0.9% de LUTs, 0.13% de FFs y 50.94% de IOBs, por lo que son suficientes para la implementación. El árbol tiene un tiempo de ejecución de 24 ciclos de reloj, equivalentes a 240 nS.

Tabla 4.2: Recursos utilizados para un árbol de sintaxis.

Recursos	Utilización	Disponibles	Utilización %
LUTs	188	20800	0.9
FFs	54	41600	0.13
IOBs	54	106	50.94

La Tabla 4.3, compara el consumo de recursos para la implementación de 2, 10, 50 y 100 árboles de sintaxis diferentes, en este caso los recursos son suficientes, excepto por el

número de IOBs, esto se debe que se necesita el 50.94% para un árbol, por lo que, si se incrementa la cantidad de árboles, el número de IOBs también aumentará. Los recursos están sujetos a la configuración del árbol, pero estos cambiarán si la configuración también. En este caso se utilizó una cantidad fija de nodos y terminales por lo que cada árbol tendrá la misma profundidad. Otro punto importante es que el árbol será un módulo interno del FPGA, esto significa que la comunicación con otros módulos será mediante señales y por lo tanto, no se utilizarán las IOBs. Adicionalmente, el tiempo de ejecución incrementa en promedio 1 ciclo de reloj, para 2 árboles se necesitan 24 ciclos de reloj, mientras que para 100 árboles solo 28 ciclos de reloj que equivalen a 280 nS, esta ventaja es gracias a la implementación en paralelo de esta etapa.

Tabla 4.3: Recursos consumidos por 2, 10, 50 y 100 árboles de sintaxis.

Arboles\ Recursos (%)	2	10	50	100
LUTs	163 (0.78%)	2007(9.64%)	10250(49.27%)	20500(98.55%)
FFs	54(0.12%)	568(1.36%)	2700(6.49%)	5400(12.9%)
IOBs	107(100.9%)	531(500.94%)	2704(2,550%)	5354(5050%)
CLKs (10 nS)	24	25	27	28

En la Figura 4.7, se muestra el diagrama esquemático RTL del árbol de sintaxis, donde podemos observar dos entradas (*clk* y *seed*) y las salidas (*Operators* y *Terminals*) para los datos del árbol. la señal *TREE* mostrada en la Figura 4.6 no es mostrada debido a que es una señal y no una salida.

Los arreglos de vectores o matrices son tipos de datos que demuestran ser muy útiles en sistemas donde se requiera trabajar con conjuntos de datos como por ejemplo los métodos de optimización.

Otro punto importante es que los arreglos no necesariamente tienen que ser del tipo std_logic y std_logic_vector, ya que es posible crear arreglos de distintos tipos de datos e incluso definir arreglos de arreglos, además al ser su definición básica del tipo std_logic, pueden ser definidos como puertos en la entidad de una arquitectura permitiendo así la conexión con otros módulos mediante la instanciación.

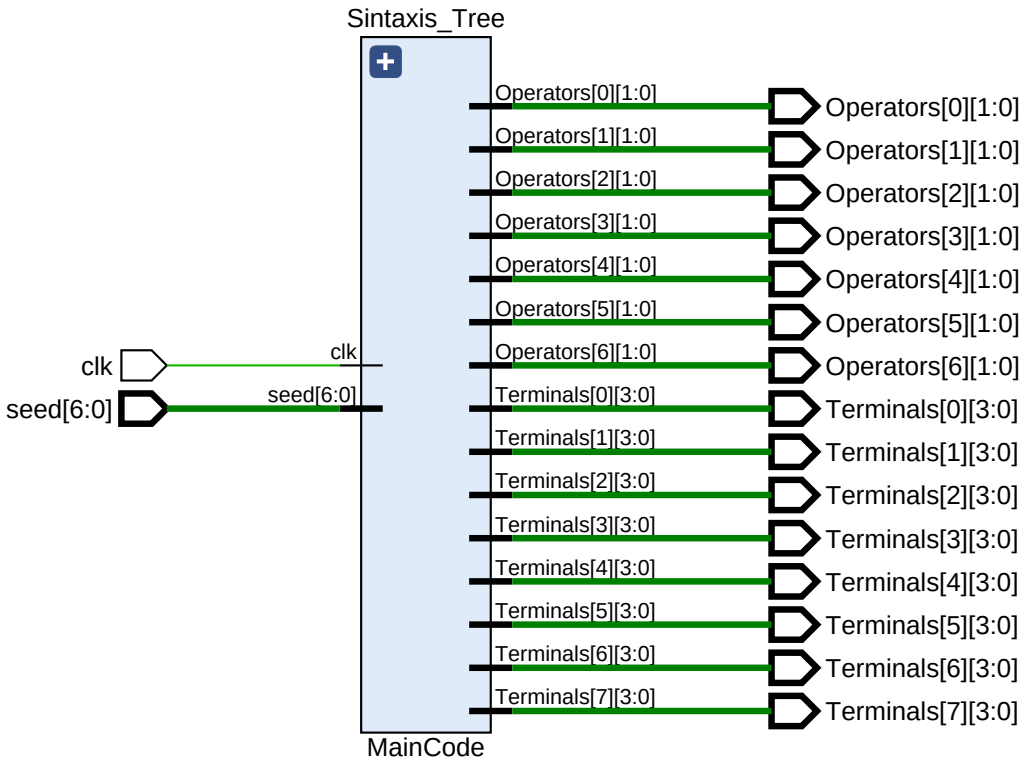


Figura 4.7: Diagrama esquemático del árbol de sintaxis.

Dados los datos mostrados en el Capítulo se observa que los recursos disponibles en el dispositivo XC7A35TCPG236-1 son suficientes para realizar un árbol de sintaxis, pero no para más, debido a la poca disponibilidad de IOBs, esto puede solucionarse, ya sea sustituyendo IOBs por señales o bien cambiar a un dispositivo que permita utilizar más IOBs.

4.2. Evaluación inicial

La semántica o vector semántico de un individuo o programa T se define como la salida del programa, donde, para cada vector de entrada $\overrightarrow{X_i}$ devuelve el valor escalar T $(\overrightarrow{X_i})$ en otras palabras la semántica de T es $\overrightarrow{S_T} = [T(\overrightarrow{x_1}), T(\overrightarrow{x_2}), ..., T(\overrightarrow{x_n})]$. Tomando en cuenta lo descrito anteriormente, el sistema GSGP propuesto basa su funcionamiento en trabajar directamente con la semántica de los individuos, puesto que utiliza operadores semánticos que modifican el vector semántico de cada individuo. Es prioritario contar con este vector antes de comenzar cualquier acción con los individuos por lo tanto se realiza la primera evaluación para obtenerlo. El vector semántico como lo indica la Figura 4.8,

se obtiene al evaluar al individuo o árbol de sintaxis, con los datos del problema que se este analizando, el vector semántico es el vector de resultados, al evaluar cada instancia del problema, por lo tanto cada renglón es la respuesta del individuo a dicha instancia, es por esto que el vector semántico tendrá el mismo numero de renglones (instancias) que el total de datos del problema analizado.

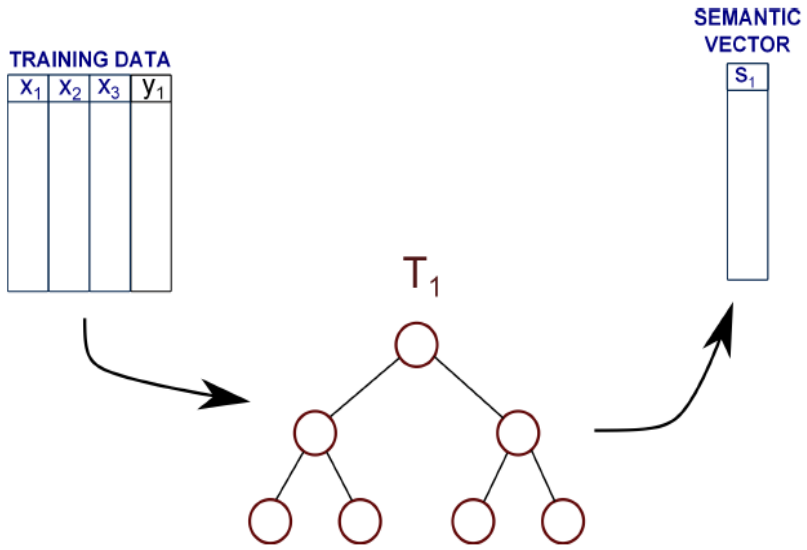


Figura 4.8: Evaluación inicial de un individuo. Se obtiene un vector semántico al evaluar el individuo con los datos del problema.

Una ventaja de la implementación propuesta del algoritmo GSGP, como se ha mencionado anteriormente, es que al ser implementado en un FPGA y programado en lenguaje VHDL se puede explotar una de las características más significativas de los dispositivos FPGA que son los procesos concurrentes o paralelos, si observamos la Figura 4.9 al evaluar una población *n* de individuos se obtienen *n* vectores semánticos, en otros lenguajes de programación para dispositivos programables como microcontroladores, los procesos serían secuenciales, de manera que se tendría que crear un individuo después de terminar el primero y por ende la evaluación seguiría el mismo principio. El algoritmo GSGP propuesto como lo muestra la Figura 4.1, aprovecha los beneficios de los procesos concurrentes y realiza todos los procesos al mismo tiempo para cada individuo, ahorrando así tiempo de ejecución.

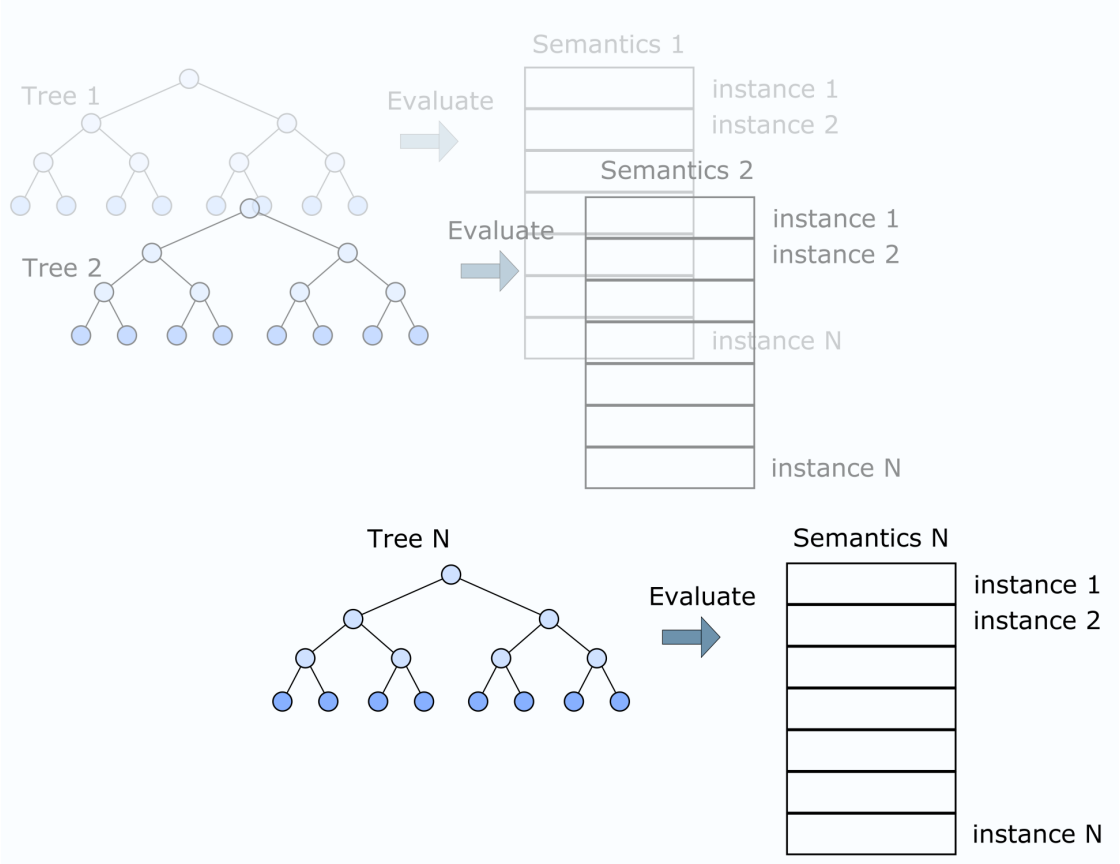


Figura 4.9: Semánticos de una población, se obtiene una población de vectores semánticos al evaluar a la población de individuos con los datos del problema.

4.3. Operaciones

Tomando en cuenta la Figura 4.2 para realizar la evaluación de un árbol de sintaxis, es necesario realizar las operaciones que represente cada operador que se presente en el penúltimo nivel de nuestra matriz con sus respectivas terminales. Otro punto importante, recordemos que los arboles de sintaxis se generan en nuestro algoritmo con un sistema de codificación mostrado en la Tabla 4.1 donde cada identificador ID esta asignado a un valor real, tomando en cuenta esto antes de realizar cualquier operación es necesario decodificar estos identificadores, esto ese realiza por un módulo independiente que identifica los ID y los remplaza por los valores asignados, por ejemplo el identificador ¨*0110*¨ es sustituído por el número 0. Para el caso particular del árbol mostrado en la Figura 4.2 se utiliza la configuración de 2 variables, por lo tanto cuando se detecta el identificador de cualquiera de las 2 variables estas son remplazadas por los datos reales del problema analizado.

4.3.1. Evaluación mediante ALU

Para realizar las operaciones matemáticas se utiliza una *Arithmetic Logic Unit* (ALU), la cuál cuenta con las siguientes características:

- Las entidades que resuelven las operaciones básicas de suma, producto y cociente han sido diseñadas como módulos independientes entre sí, con el objeto de permitir la implementación separada de cada una de ellas.
- Se cuenta con la implementación de un control lógico que gestione el enrutamiento de los datos y que permita obtener operaciones derivadas de los bloques básicos.
- Inclusión de un registro en el que se almacena el último resultado obtenido para que pueda usarse como operador en la siguiente operación, con objeto de reducir la cantidad de ciclos de escritura-lectura.

La Figura 4.10 muestra el diseño de la ALU la cual realiza todas las operaciones posibles al mismo tiempo para las 2 terminales y por medio de un selector de operación se decide cual es el resultado que será enviado como salida, este selector toma el identificador del operador y decide cual es el resultado mostrado.

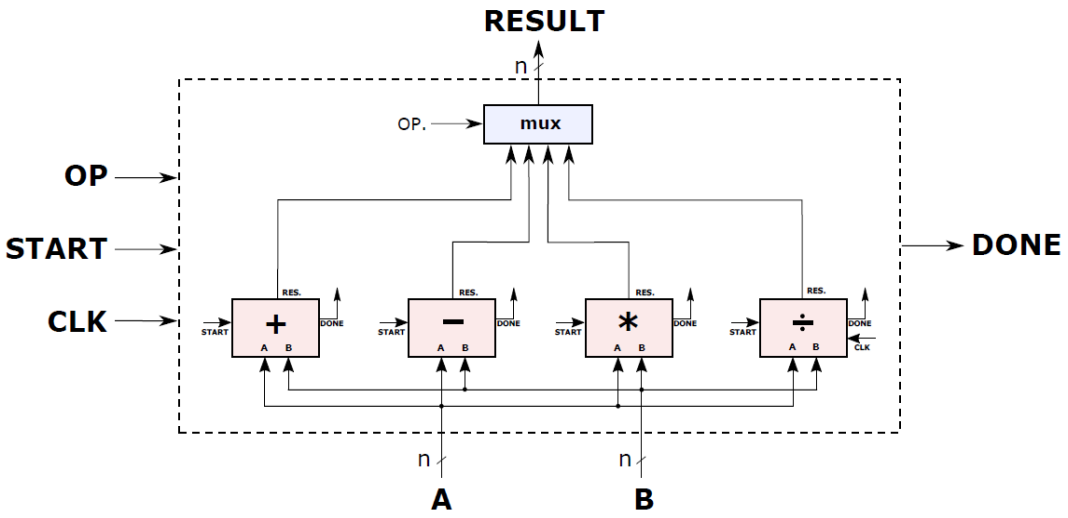


Figura 4.10: ALU, realiza todas las operaciones posibles al mismo tiempo y por medio de un selector de operación se decide cual es el resultado que será enviado como salida.

La Figura 4.10 nos muestra el diseño de la ALU que realiza las operaciones individuales, es decir, que solo involucra un operador y 2 terminales del árbol de sintaxis completo, en la Figura 4.11 se muestra el diseño del árbol de evaluación completo, se observa que está compuesto por módulos ALU interconectados siguiendo la representación de un árbol de sintaxis común.

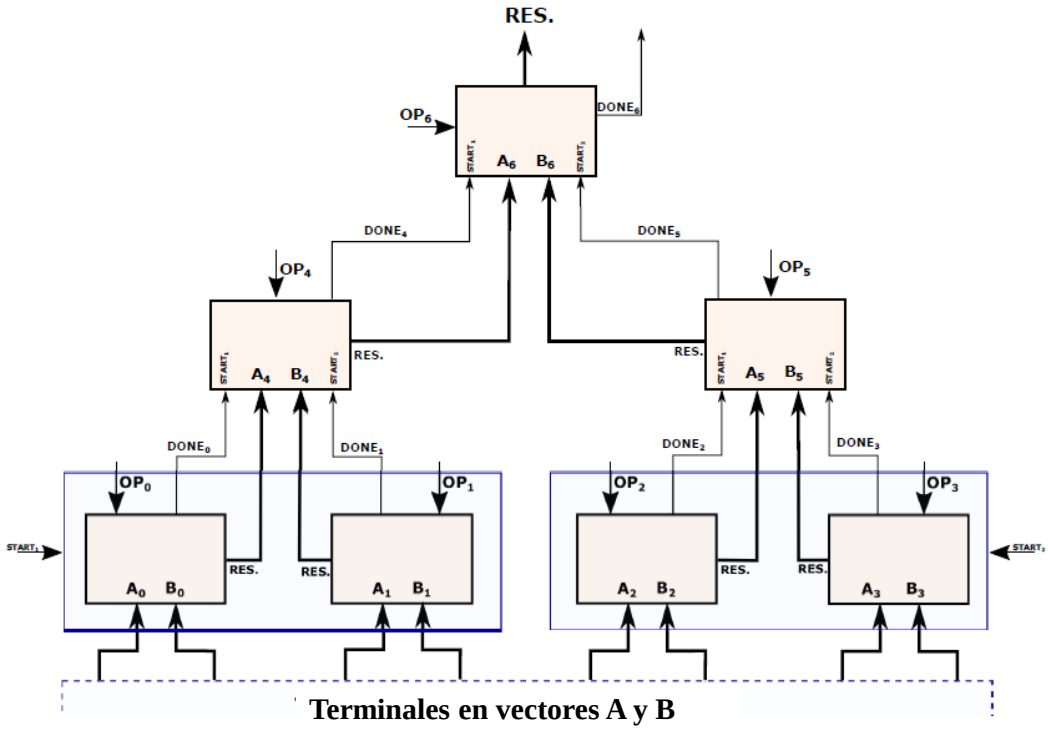


Figura 4.11: Árbol de evaluación completo basado en ALU. Cada operación se realiza por niveles en un proceso semi-secuencial.

4.3.2. Representación de punto fijo

El algoritmo GSGP-VHDL se ha implementado para manejar números de punto fijo con signo. En esta implementación, n es el grupo de bits reservado para la parte fraccionaria del número, mientras que *BIT_WIDTH* se refiere a los bits generales que representan el número de punto fijo (entero + partes fraccionarias), como se indica en la Figura 4.12.

A diferencia de otras operaciones aritméticas como la suma o la multiplicación que requieren un número finito de dígitos para representar los resultados, la división puede requerir muchos más dígitos que los operandos para representar el cociente con precisión

y quizás un número infinito de lugares decimales. Es debido a esto que para mantener una mayor precisión se optó por utilizar la representación de punto fijo para la totalidad del sistema.

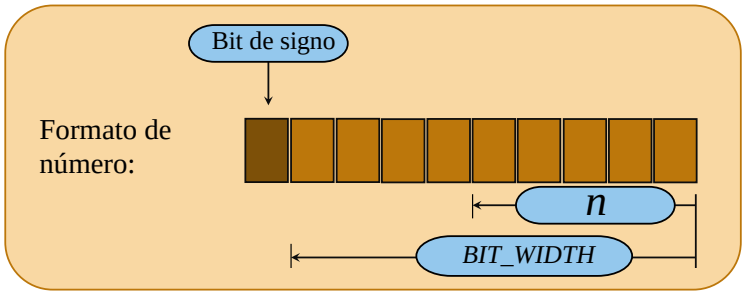


Figura 4.12: Representación de número en formato de punto fijo con signo.

Operandos con signo

Existen varios métodos para representar números binarios negativos; los más comunes son el complemento a uno, el complemento a dos y el bit de signo. Todos los bits se invierten en el complemento a uno, mientras que en el complemento a dos, el primer paso es tomar el complemento a uno del número y luego agregar un 1 al bit menos significativo. En la implementación propuesta, el método para representar el signo de un número es usar el bit más significativo, si el bit es 0, entonces es un número positivo; de lo contrario, es un número negativo, como lo muestra la Figura 4.12. Las operaciones se realizan excluyendo el bit de signo, esto significa que solo los números positivos entran en el proceso de la operación. La función *XOR* se utiliza para calcular el signo de operaciones como el producto o la división, como se muestra en la Figura 4.13. Las entradas de la función lógica son los bits de signo de los operandos (dejando de lado el resto de los bits) y la salida es el signo del resultado, en el caso de la suma o resta el bit solo servirá para determinar cual numero es mayor o menor y determinar el signo resultante.

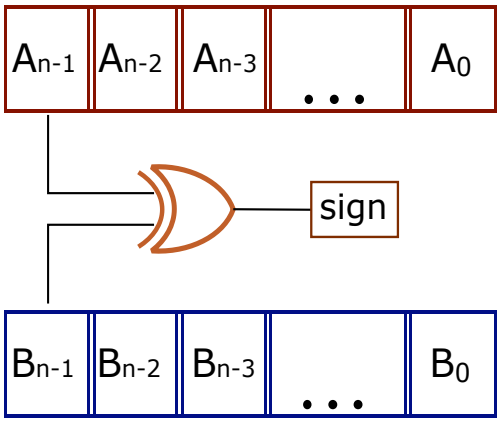


Figura 4.13: Operandos con signo.

4.3.3. Implementación

La Figura 4.14 muestra el diagrama RTL del módulo de la Evaluación inicial, se observan 3 módulos interconectados, el primero llamado *Generador_árbol* es el módulo que se encarga de crear el árbol de sintaxis, este módulo esta seguido por un módulo llamado *Config_datos* este módulo se encarga de sustituir los identificadores por los datos reales y enviar estos datos al siguiente módulo, por último se muestra el módulo *operaciones* en el cual está la implementación del árbol de evaluación mostrado en la Figura 4.11 y se encarga de realizar las operaciones pertinentes del árbol de sintaxis.

En la Figura 4.15 se muestra el proceso concurrente para la evaluación de una población en este caso de 2 individuos, se observa que cada uno necesita una semilla de entrada para la creación del árbol, mientras que como salida se tiene el vector semántico, además de una pequeña población de mini-arboles necesarios en el módulo de mutación.

En la Figura 4.16, se observa la simulación del módulo de Evaluación inicial, en la simulación se muestra la entrada *clk*, que es la señal del reloj de 100MHz de la tarjeta Basys 3, la señal *seed* que se compone de dos semillas de 7 bits cada una, y los vectores semánticos de cada individuo por generación, para esta simulación de 1 individuo, se observa el cursor en color rojo que indica que aproximadamente en 1680 nS (menos 1000 nS de la inicialización del simulador) la salida *resultado* tiene un valor debido a que la salida *doneOut* se activó, por lo que se interpreta que al sistema le toma 68 ciclos de reloj (10 nS equivalen a un ciclo de reloj) completar el proceso de evaluación inicial.

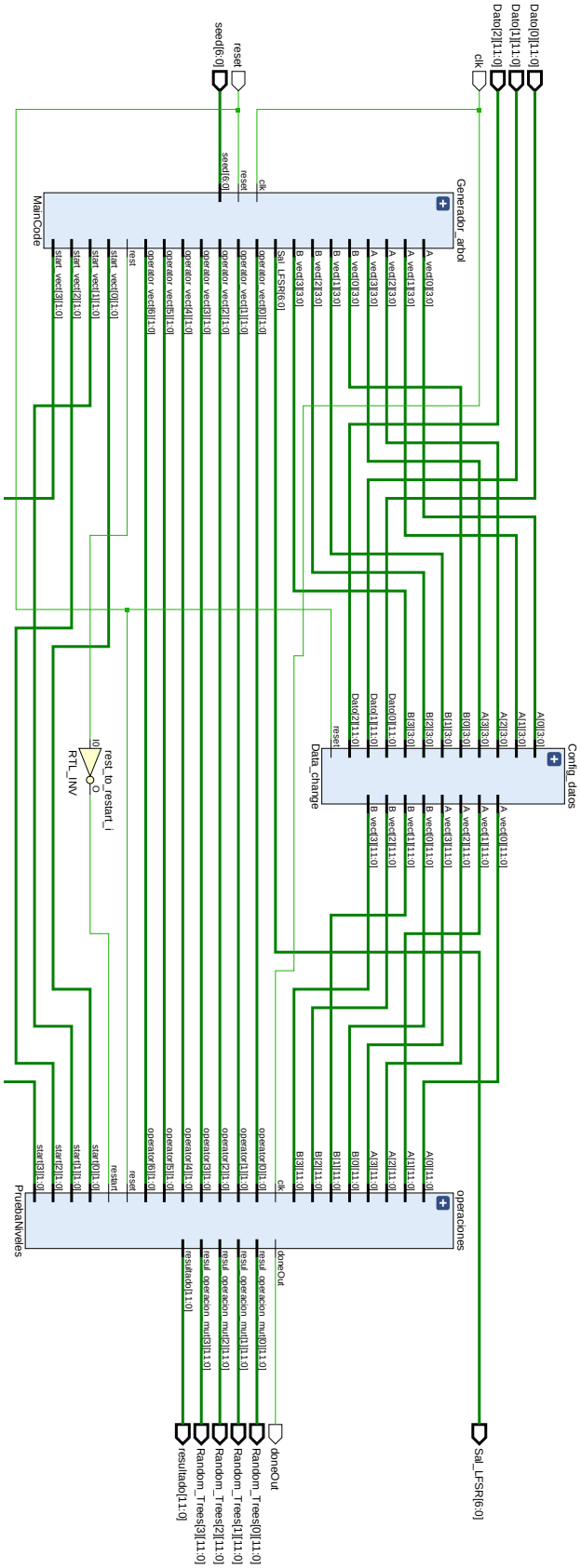


Figura 4.14: Evaluación de un individuo.

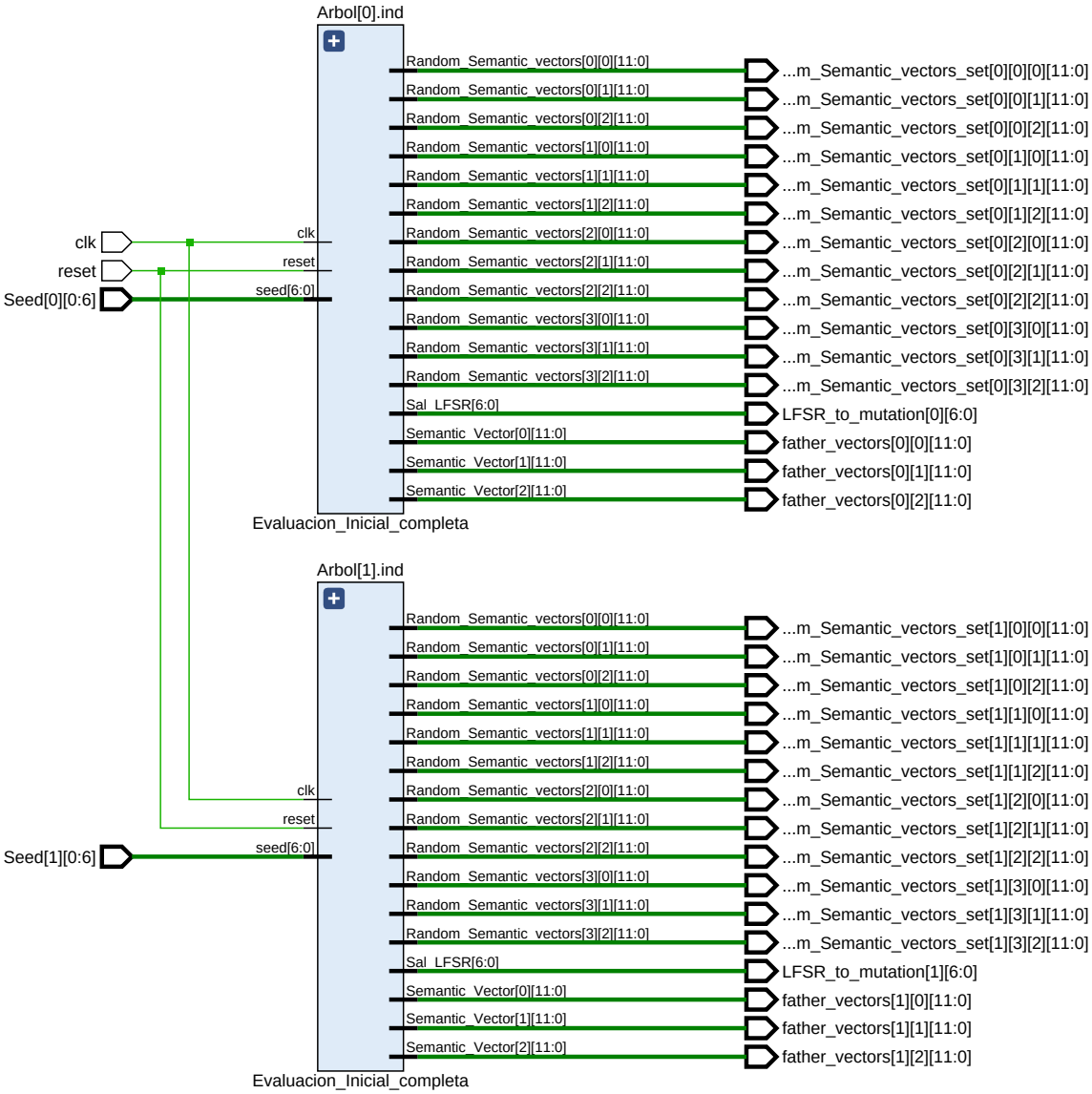


Figura 4.15: Evaluación de una población de 2 individuos.

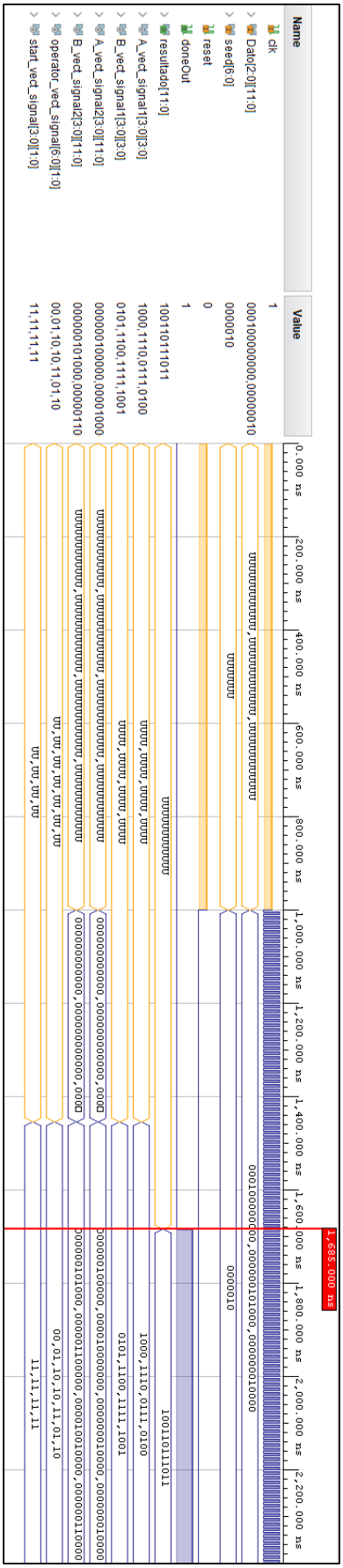


Figura 4.16: Simulación de Evaluación de 1 individuo.

4.4. Mutación Geométrica semántica

Debido a que la implementación se realiza en un dispositivo FPGA, la Ecuación 3.2 debe ser entendida como un circuito, de esta manera será más sencillo de implementar en el lenguaje VHDL, la Figura 4.17, muestra el circuito propuesto para la implementación de la Ecuación 3.2, se observan los diferentes vectores semánticos así como el paso de mutación.

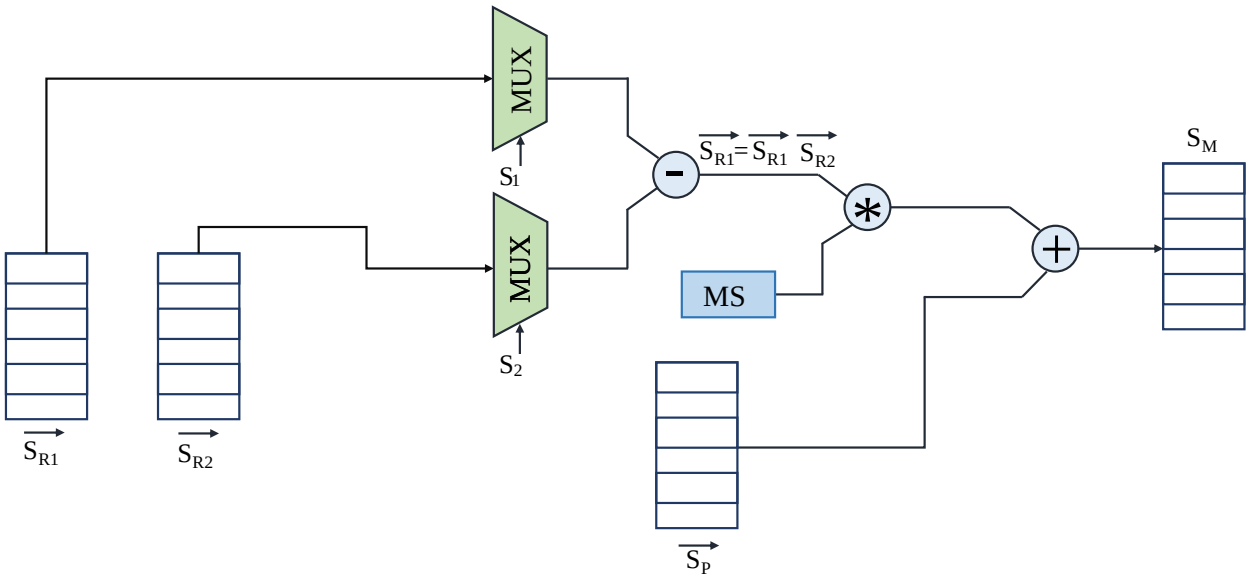


Figura 4.17: Circuito para la implementación de la Ecuación 3.2.

La Figura 4.18, muestra el diagrama RTL de una sección del módulo de mutación, en donde es posible observar varios de los componentes mostrados en el circuito de la Figura 4.17, por ejemplo, los módulos que contienen los vectores semánticos tanto de la diferencia de los individuos aleatorios, así como el del padre o individuo seleccionado.

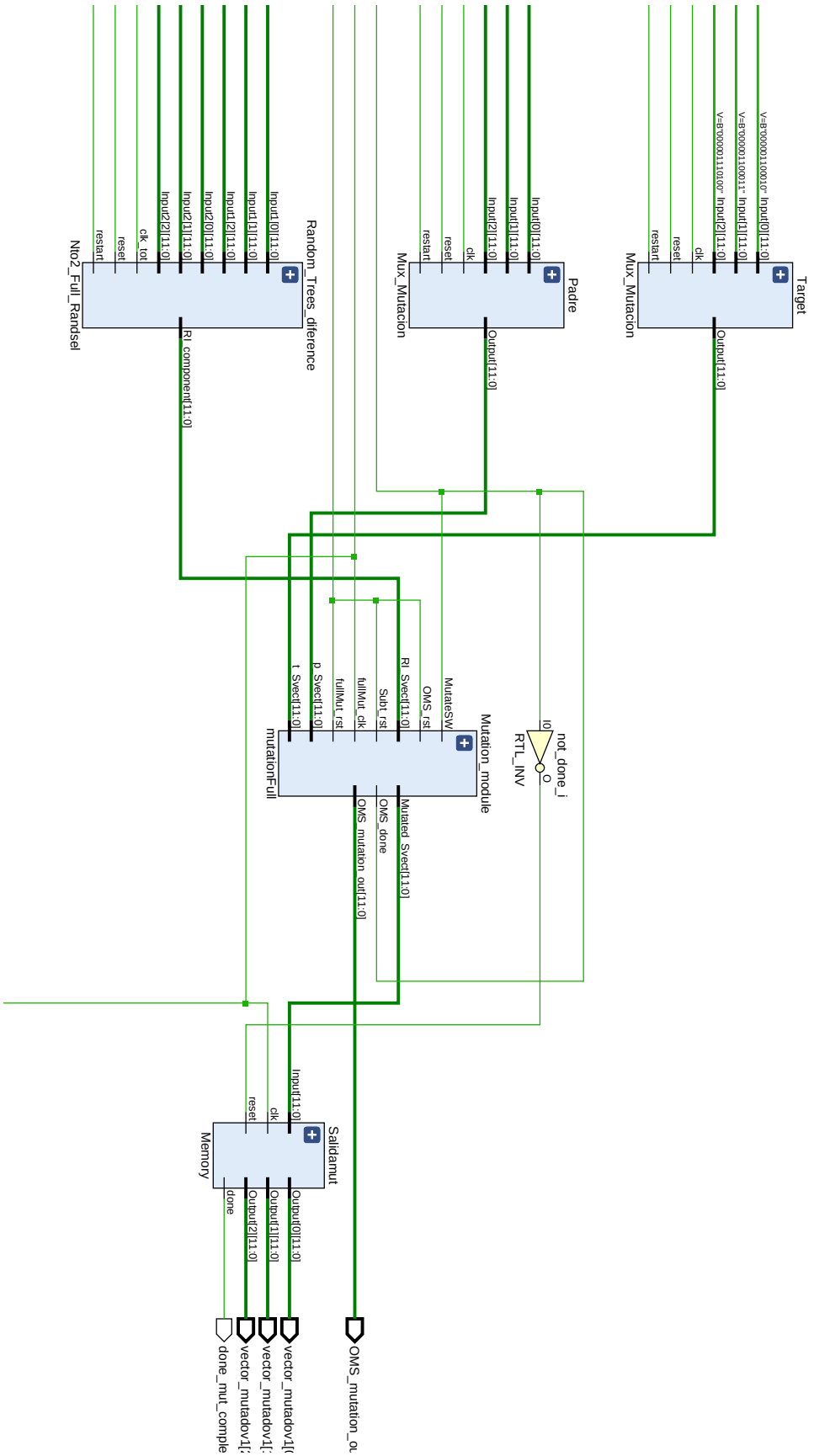


Figura 4.18: Diagrama RTL de una sección del módulo de la mutación.

4.4.1. Paso de mutación óptimo

Usando la Ecuación 3.8, se requiere menos recurso computacional en comparación con un método numérico complejo. Así, la Ecuación 3.8 solo requiere cálculos aritméticos, la Figura 4.19 muestra el circuito diseñado para la implementación de la Ecuación 3.8.

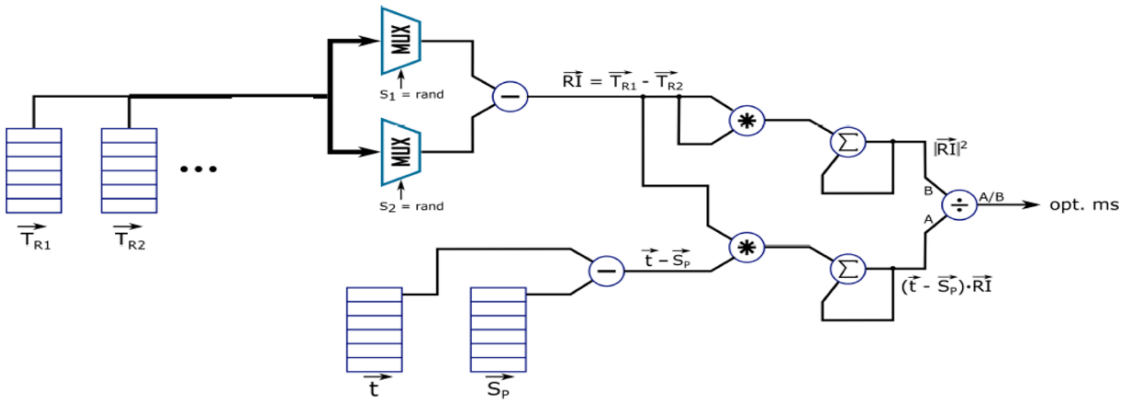


Figura 4.19: Circuito para la implementación del paso de mutación óptimo.

En la Figura 4.20, se muestra el diagrama esquemático RTL del módulo del paso de mutación óptimo, aquí se encuentran los diferentes módulos internos, tanto del módulo de mutación como del módulo del OMS.

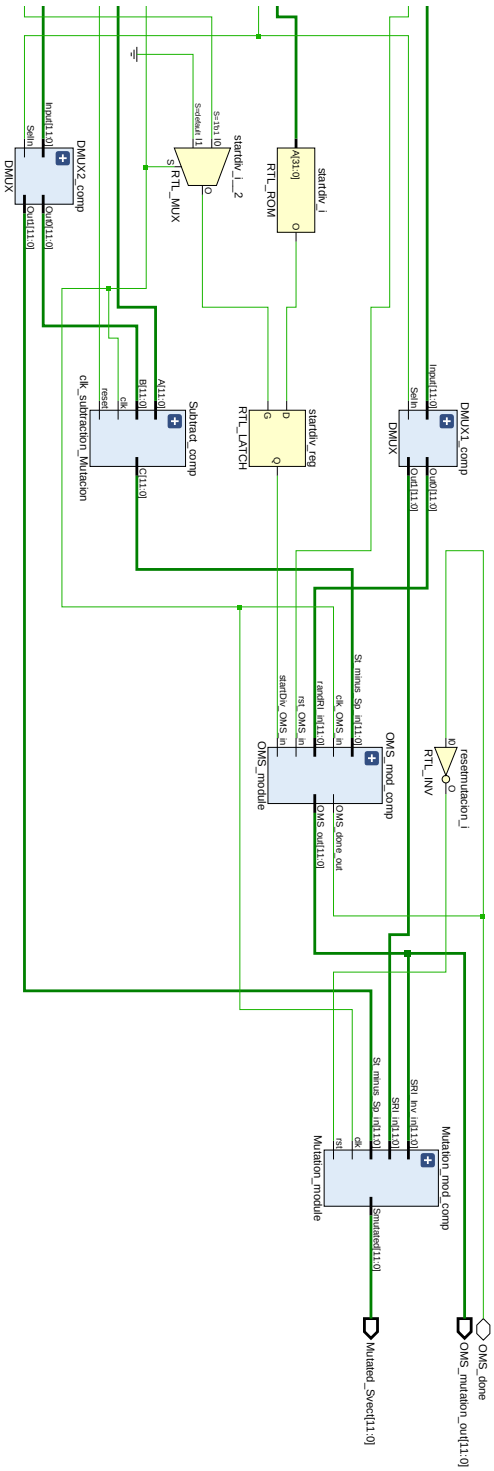


Figura 4.20: Módulo del paso de mutación óptimo.

Como se puede ver en la Figura 4.1, la mutación tiene que ser cíclica, de modo que este módulo se debe repetir *n* cantidad de veces, para lograr esto en el lenguaje VHDL se realizó una instanciación en serie del mismo módulo, de tal manera que

cuando uno termine su proceso, el vector semántico resultante será utilizado como entrada de la siguiente instancia del mismo módulo, la Figura 4.21 muestra el diagrama esquemático RTL del módulo de mutación instanciado 2 veces, es decir una mutación de 2 generaciones. En la Figura 4.22, se observa la simulación del módulo completo de mutación, en la simulación se muestra la entrada *clk* que es la señal del reloj de 100MHz, la señal *seed* que se compone de dos semillas de 7 bits, y los vectores semánticos de cada individuo por generación, en la simulación se observa el cursor en color rojo el cual indica el instante de tiempo en que la salida *done_complete* se activa, por lo tanto, el resultado para dos individuos y dos generaciones se obtiene en aproximadamente 2960 nS (menos 1000 nS de la inicialización del simulador) o 196 ciclos de reloj.

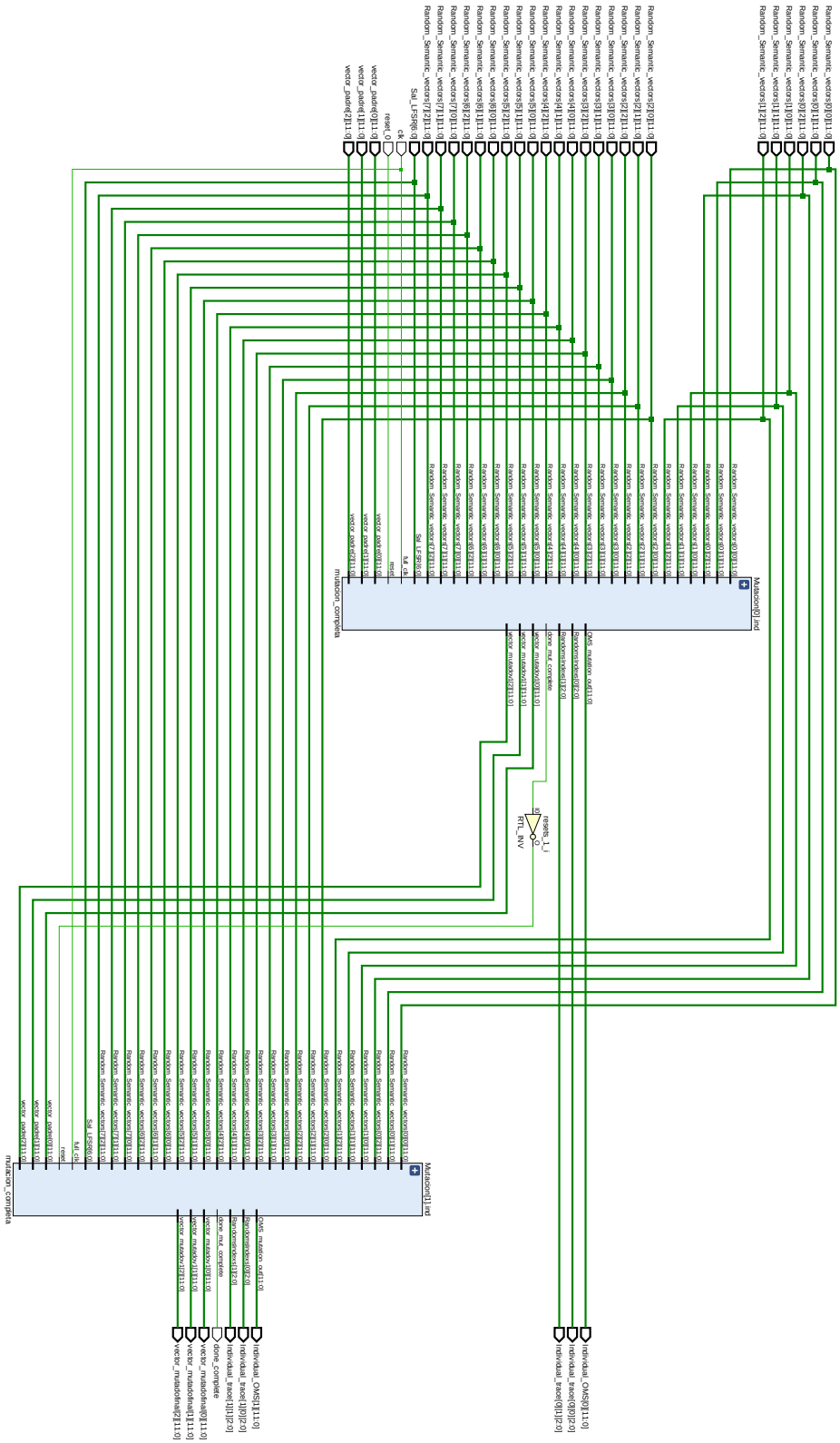


Figura 4.21: Diagrama RTL del módulo de mutación de 2 generaciones.

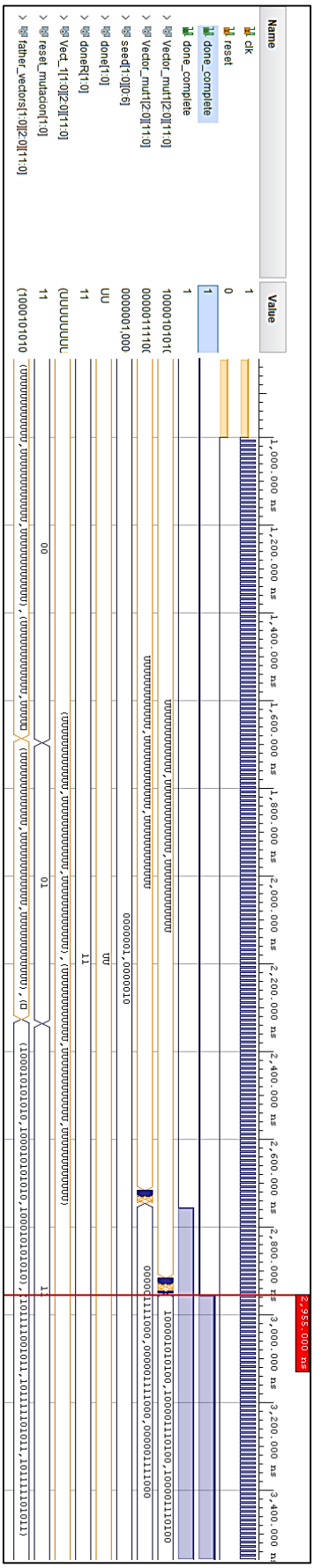


Figura 4.22: Simulación mutación generacional.

4.5. Fitness

Para evaluar la calidad de los individuos se hace uso del *Mean Absolute Error* (MAE), el error se calcula como un promedio de diferencias absolutas entre los valores objetivo en este caso el target y el vector semántico del individuo mutado o nuestro vector padre. El MAE es una puntuación lineal, lo que significa que todas las diferencias individuales se ponderan por igual en el promedio. Matemáticamente, se calcula utilizando la fórmula:

$$MAE = \sum_{i=1}^{n} \frac{|y_i - Y_i|}{n}. \tag{4.1}$$

Lo importante de esta métrica es que penaliza errores muy grandes, por lo tanto, no es tan sensible a los valores atípicos como el error cuadrático medio. La Figura 4.23, muestra el circuito propuesto para la implementación de la Ecuación 4.1, la cual esta conformada por la suma de la diferencia del vector padre y el target.

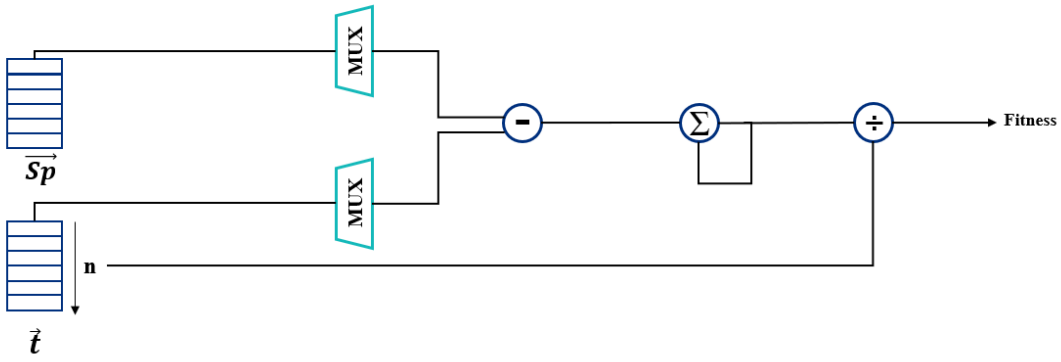


Figura 4.23: Circuito propuesto para la implementación del fitness MAE.

En la Figura 4.25, se ilustra la simulación del módulo fitness MAE, en la simulación se muestra la entrada *clk* es la señal del reloj de 100MHz, la salida *multiple_fitness* que contiene el valor de la aptitud de los individuos, en la Figura se observa, que el GSGP-VHDL calcula el fitness en 255 ciclos de reloj (quitando los 1000 nS de la inicialización del simulador), esto se visualiza con la posición del cursor color rojo, el cual indica que la salida *doneOut* y la salida *multiple_fitness* se activan en el mismo instante de tiempo.

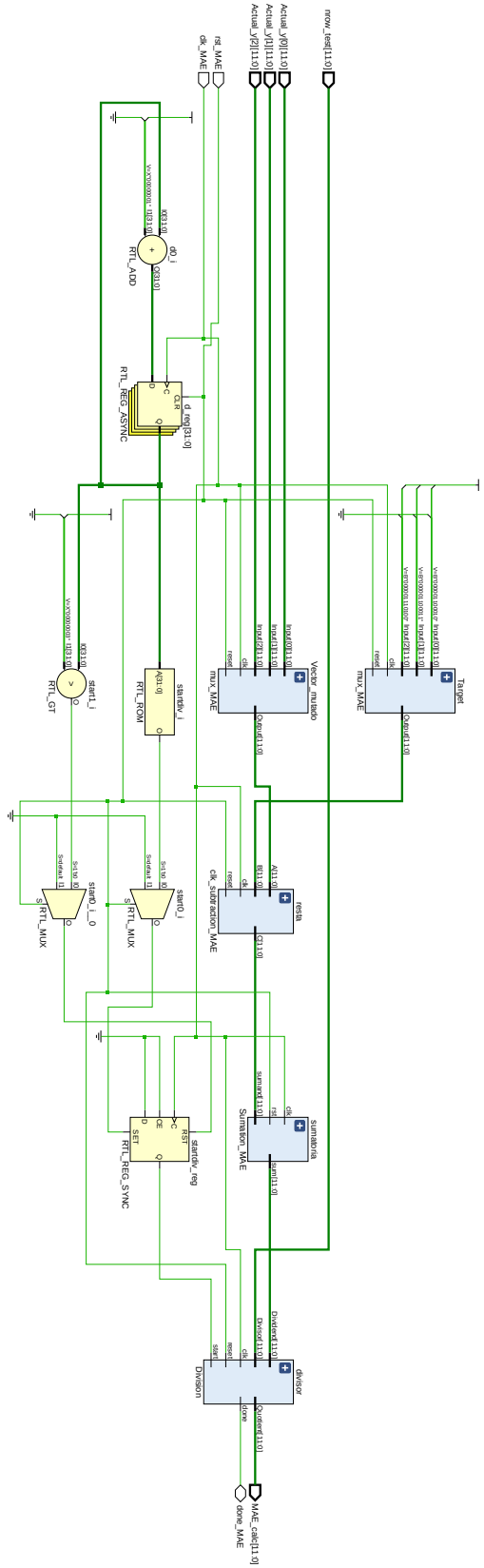


Figura 4.24: RTL fitness MAE.

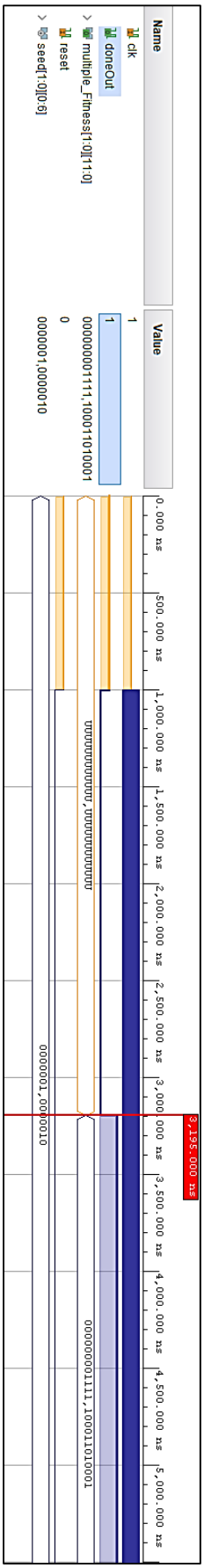


Figura 4.25: Simulación fitness MAE.

4.6. Sistema completo

El sistema completo llamado procesador dedicado GSGP-VHDL contiene instanciados todos los módulos diseñados en VHDL y explicados anteriormente. El procesador dedicado GSGP-VHDL aprovecha el paralelismo en los procesos. En la Figura 4.26, se muestran los módulos *individual[0]* e *individual[1]* que contienen todas las etapas por las que pasa un individuo, como lo mostrado en la Figura 4.1, por último en la Figura 4.26, se muestra el módulo *Comparador* que se encarga de encontrar el individuo con mejor aptitud utilizando los datos de entrenamiento. La Figura 4.27 muestra la simulación del sistema completo, destacan las señales *bestsolution* y *bestindex*, estas muestran el valor de la aptitud del mejor individuo y su posición en la población. Para esta simulación, se observa que la salida *DoneOut* se activa en aproximadamente 4920 nS (menos 1000 nS de la inicialización del simulador), equivalente a 392 ciclos de reloj para que el GSGP-VHDL obtenga una respuesta válida.

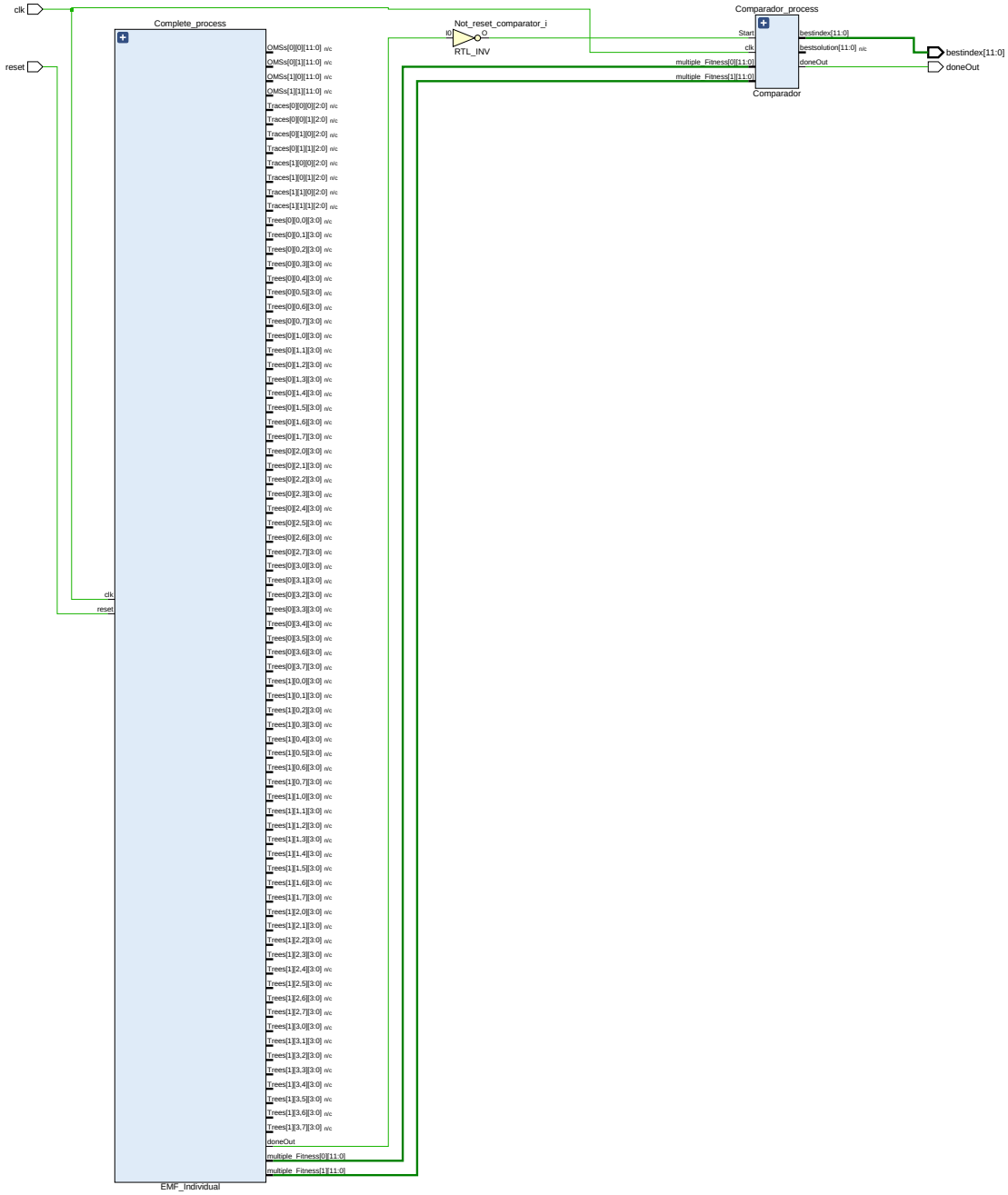


Figura 4.26: Diagrama RTL del GSGP-VHDL.

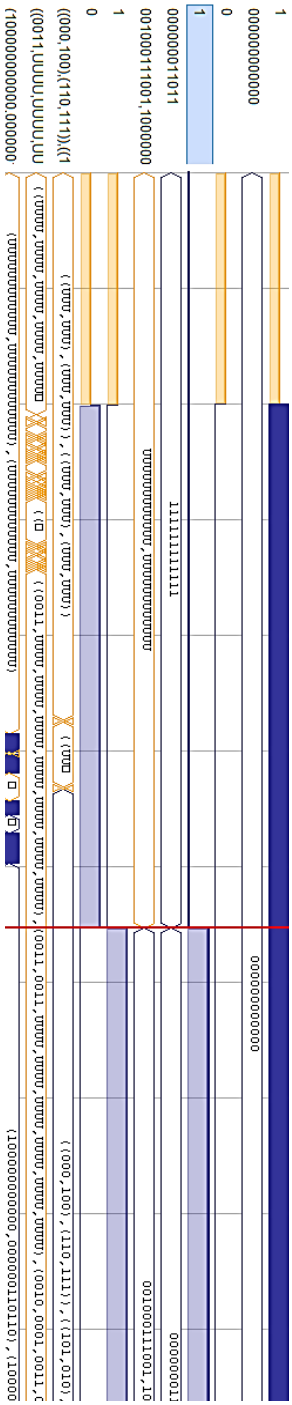


Figura 4.27: Simulación del GSGP-VHDL

4.6.1. Entrenamiento y prueba

Dada la naturaleza del GSGP, es necesario realizar un módulo para evaluar el entrenamiento y prueba del algoritmo. En la Figura 4.28, se muestra la etapa de evaluación del procesador dedicado GSGP-VHDL, en esta etapa una vez que el GSGP-VHDL obtenga al mejor individuo con los datos de entrenamiento, este módulo toma como entradas al individuo seleccionado, a todos los mini-arboles que fueron seleccionados en la mutación y también toma todos los pasos de mutación que fueron utilizados, de manera que sea posible reconstruir el mejor individuo.

El módulo realiza las operaciones para evaluar al individuo con los datos de prueba, la Figura 4.29 muestra la simulación de esta etapa, aquí resaltan la salida *fitness* puesto que es el valor de la aptitud de nuestro individuo evaluado con los datos de prueba.

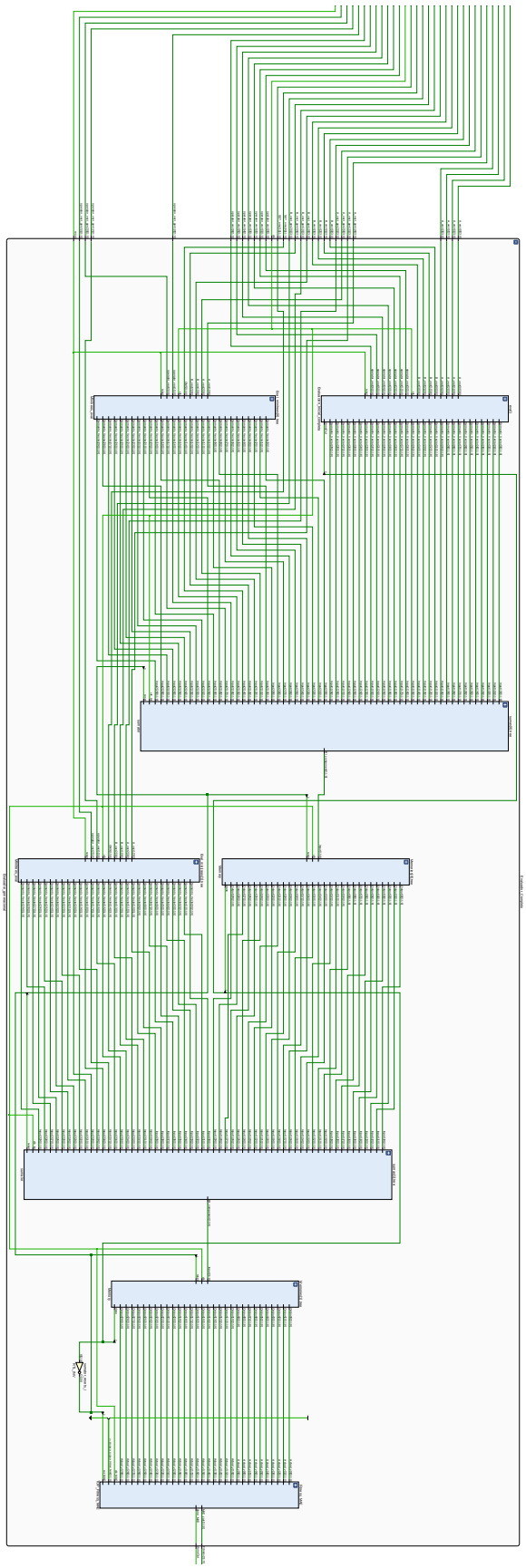


Figura 4.28: Diagrama RTL del sistema de evaluación del procesador dedicado GSGP-VHDL.

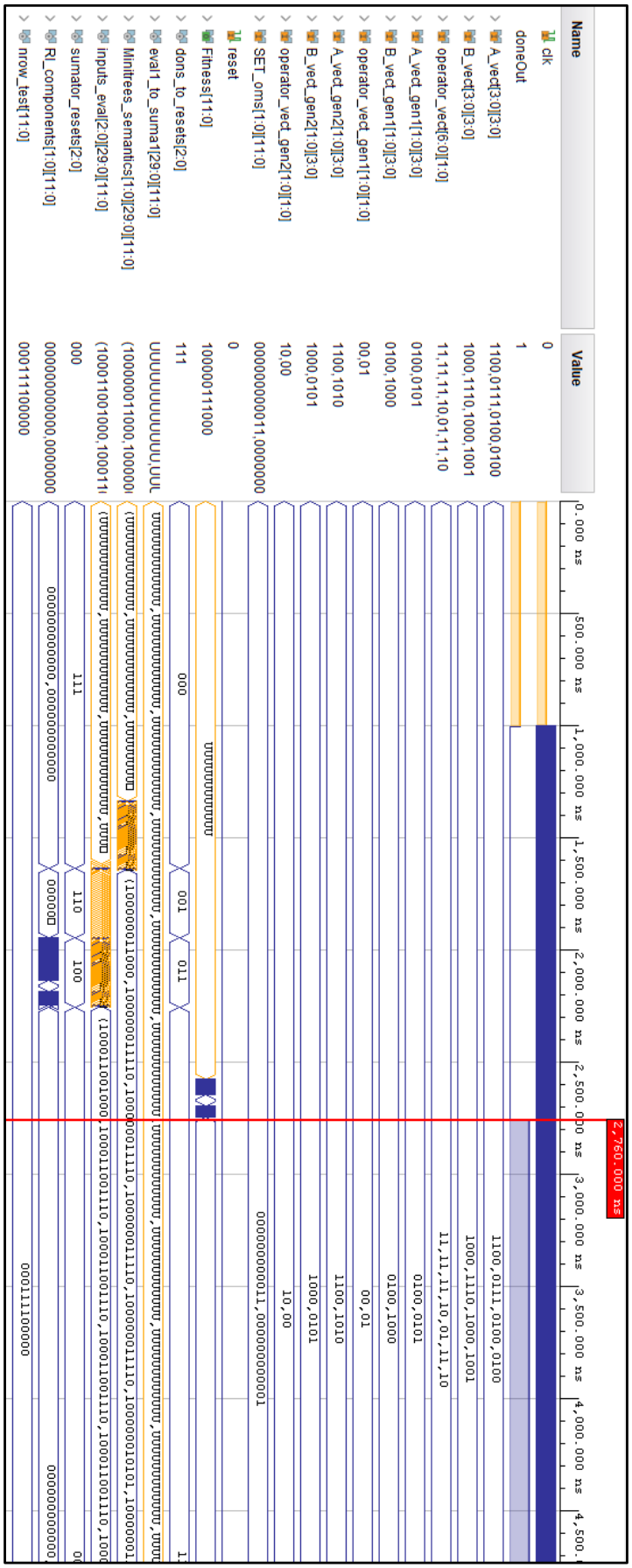


Figura 4.29: Simulación del sistema de evaluación del procesador dedicado GSGP-VHDL

4.7. Conclusión

La versatilidad del lenguaje VHDL al simular el comportamiento de un circuito en todos los módulos, hace que sea mucho más sencillo el reutilizar módulos creados anteriormente, en estructuras o diseños más complejos, por ejemplo, en el procesador dedicado GSGP-VHDL desarrollado y explicado en este Capítulo contiene varios módulos individuales que son reutilizados a lo largo de todos los procesos, como por ejemplo el módulo de suma, producto y división, estos módulos son usados en todas las etapas, además, el comportamiento simulado de un circuito permite compactar todo un sistema en un solo módulo, y por medio de una instanciación, clonarlo o multiplicarlo las veces que se requiera.

Analizando los resultados obtenidos, es posible decir que una vez diseñado el procesador dedicado GSGP-VHDL, se tiene la facilidad de que el procesador se vuelva más preciso, utilizando instanciaciones para crear varias copias del mismo sistema que contengan 1 o más procesadores GSGP-VHDL trabajando en paralelo, siempre y cuando los recursos del dispositivo lo permitan. Otro punto importante es el tiempo de ejecución, debido a que la cantidad de árboles no afecta el rendimiento del procesador dedicado GSGP-VHDL, por lo tanto, el procesador dedicado GSGP-VHDL resulta ser muy atractivo y competitivo.

Capítulo 5

Discusión de los Resultados

Para evaluar la calidad de la implementación del procesador dedicado GSGP-VHDL, se utilizó un conjunto de problemas de regresión simbólica del mundo real especificados en la Tabla 5.1, estos problemas se han utilizado ampliamente para evaluar algoritmos GP, donde el objetivo es minimizar la función aptitud. Además, para realizar la comparación del procesador dedicado GSGP-VHDL, se utilizó un diseño de GSGP en software (GSGP-Software) [4] con características equivalentes a lo del GSGP-VHDL, como por ejemplo, la representación de los datos en punto fijo con diferentes cantidades de bits. En la Tabla 5.2 se muestran los parámetros de configuración para la evaluación del GSGP en sus dos versiones.

Tabla 5.1: Problemas de regresión simbólica del mundo real.

Problema	**Descripción**
Concrete [42]	La resistencia a la compresión del hormigón es una función altamente no lineal de los parámetros de construcción.
Energy Cooling [34]	Evaluación de los requisitos de refrigeración de los edificios en función de los parámetros del edificio.
Energy Heating [34]	Evaluación de los requisitos de calefacción de los edificios en función de los parámetros del edificio.
Housing [29]	El valor de la vivienda en los suburbios de Boston.
Tower [38]	Un conjunto de datos industriales de medición de cromatografía de la composición de una torre de destilación.
Yacht [12]	Conjunto de datos de Delft, utilizado para predecir el rendimiento hidrodinámico de la navegación a partir de las dimensiones y la velocidad.

Tabla 5.2: Parámetros de la configuración de los problemas para evaluar el rendimiento del procesador dedicado GSGP-VHDL.

Parámetros	**Rangos**
Número de corridas	30
Tamaño de población	10 individuos
Numero de generaciones	3, 6, 8, 10
Probabilidad de mutación	1
Numero de variables	2, 12

5.1. Pruebas con 2 variables

Las Tablas y Figuras mostradas en esta sección se realizaron con los parámetros descritos en la Tabla 5.2, para la realización de esta prueba se configuró el procesador GSGP-VHDL con 2 variables y 3, 6, 8 y 10 generaciones.

Los resultados obtenidos se resumen en la Tabla 5.3, donde se muestra el rendimiento de ambos algoritmos con datos de entrenamiento, en la Tabla 5.4 se muestran los resultados de las pruebas con datos de prueba en cada unos de los algoritmos y en cada uno de los problemas. Estas Tablas muestran la media, la mediana, la desviación estándar (STD) en todas las corridas. En las Tablas se muestran las ejecuciones para datos de entrenamiento y datos de prueba, también se visualiza que GSGP-VHDL tiene un mejor rendimiento contra el diseño GSGP-Software en todos los experimentos, por lo tanto, se puede deducir que GSGP-VHDL es un excelente candidato para resolver este tipo de problemas.

Tabla 5.3: Comparación del GSGP-VHDL vs. GSGP-Software con 2 variables y datos de entrenamiento.

Generaciones	Problema	GSGP-VHDL			GSGP-SOFTWARE		
		Media	Mediana	STD	Media	Mediana	STD
3	Housing	6.954	7.000	0.628	8.943	9.217	1.303
	Heating	7.278	7.250	0.933	8.821	8.721	1.209
	Ecooling	7.007	7.007	0.687	8.943	8.915	1.297
	Concrete	9.065	9.125	0.638	15.226	15.237	0.641
	Yatcht	7.722	7.688	0.753	10.646	10.636	0.083
	Tower	222.951	221.625	15.146	273.366	271.258	15.858
6	Housing	6.433	6.433	0.567	7.668	7.732	0.904
	Heating	6.213	6.188	0.808	8.202	8.125	1.225
	Ecooling	5.919	6.063	0.704	8.001	8.099	0.613
	Concrete	8.032	8.032	0.651	14.170	14.290	0.566
	Yatcht	7.079	7.000	0.662	10.306	10.307	0.055
	Tower	209.035	212.500	10.408	239.547	242.978	13.584
8	Housing	5.271	5.250	0.906	6.897	6.798	0.869
	Heating	5.419	5.563	0.755	7.163	6.930	0.838
	Ecooling	4.986	4.875	0.709	7.358	7.353	0.681
	Concrete	7.523	7.523	0.624	13.105	13.061	0.432
	Yatcht	6.583	6.563	0.612	10.200	10.216	0.061
	Tower	190.896	190.896	15.058	225.338	224.069	14.333
10	Housing	4.590	4.590	1.072	6.627	6.682	0.863
	Heating	4.084	4.084	0.649	6.910	6.828	0.979
	Ecooling	3.949	3.949	0.598	6.669	6.741	0.769
	Concrete	7.056	7.125	0.629	12.783	12.641	0.471
	Yatcht	6.354	6.375	0.601	10.098	10.095	0.058
	Tower	176.576	180.250	15.198	214.720	215.631	10.726

Tabla 5.4: Comparación del GSGP-VHDL vs. GSGP-Software con 2 variables y datos de prueba.

Generaciones	Problema	GSGP-VHDL			GSGP-SOFTWARE		
		Media	Mediana	STD	Media	Mediana	STD
3	Housing	7.440	7.625	1.114	10.290	10.286	1.670
	Heating	7.921	7.921	1.155	10.345	10.371	1.583
	Ecooling	7.669	7.669	1.060	10.591	10.642	1.446
	Concrete	9.759	9.750	1.022	16.384	16.018	0.843
	Yatcht	7.625	7.625	0.690	9.636	9.601	0.092
	Tower	190.630	194.313	17.476	154.703	156.420	18.056
6	Housing	7.213	7.313	0.883	8.704	8.717	1.168
	Heating	7.134	7.134	1.166	9.461	9.187	1.384
	Ecooling	6.442	6.625	0.826	8.974	8.944	0.989
	Concrete	8.880	8.880	1.156	15.213	15.119	0.761
	Yatcht	6.991	6.938	0.782	9.514	9.505	0.055
	Tower	184.090	184.375	16.018	126.224	125.510	15.487
8	Housing	5.963	6.375	1.197	7.989	7.900	1.145
	Heating	6.093	6.250	1.292	8.204	8.162	0.762
	Ecooling	5.692	5.563	0.729	8.331	8.255	0.966
	Concrete	7.567	7.688	0.849	14.214	14.133	0.741
	Yatcht	6.532	6.438	0.694	9.374	9.366	0.042
	Tower	170.551	170.551	17.197	107.576	103.962	23.002
10	Housing	5.646	5.625	1.348	7.372	7.095	1.130
	Heating	4.900	4.900	0.926	7.911	8.155	1.019
	Ecooling	4.780	4.625	1.007	7.310	7.498	0.809
	Concrete	7.331	7.313	0.757	13.748	13.558	0.773
	Yatcht	6.350	6.438	0.641	9.276	9.270	0.039
	Tower	155.944	159.250	15.935	94.315	97.083	19.912

En las Figuras 5.1, 5.2, 5.3, 5.4, 5.5 y 5.6 se observan las comparaciones de ambos algoritmos en el comportamiento gradual de la aptitud conforme se incrementa la cantidad de generaciones desde 3 hasta 10, las gráficas muestran el fitness del algoritmo GSGP-VHDL y del algoritmo GSGP-Software con datos de entrenamiento (Train) y prueba (Test) para cada uno de los problemas.

Se puede apreciar en las distintas Figuras que los valores del fitness de GSGP-VHDL tienden a converger a 0 antes que los valores mostrados por GSGP-Software.

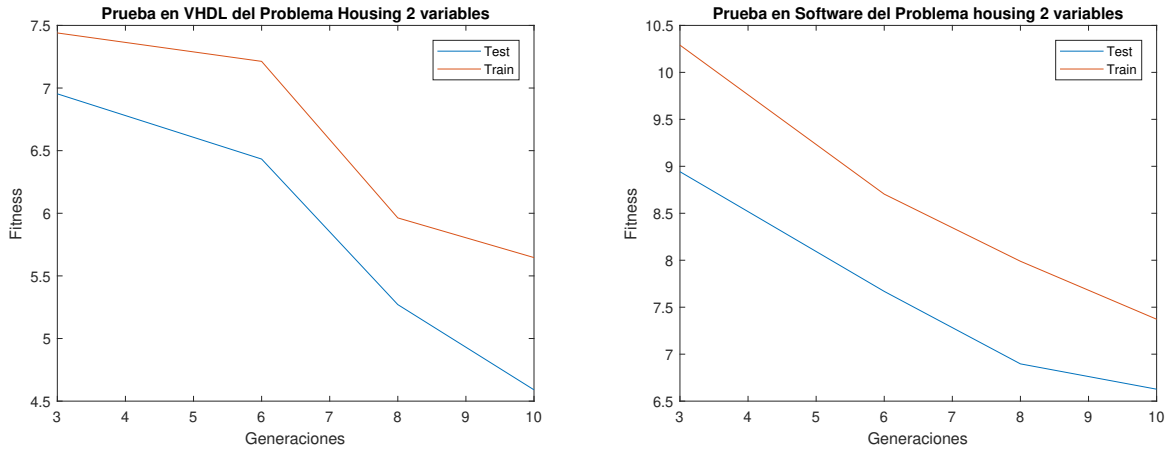


Figura 5.1: Fitness del Problema Housing a 2 variables, GSGP-VHDL Vs. GSGP-Software.

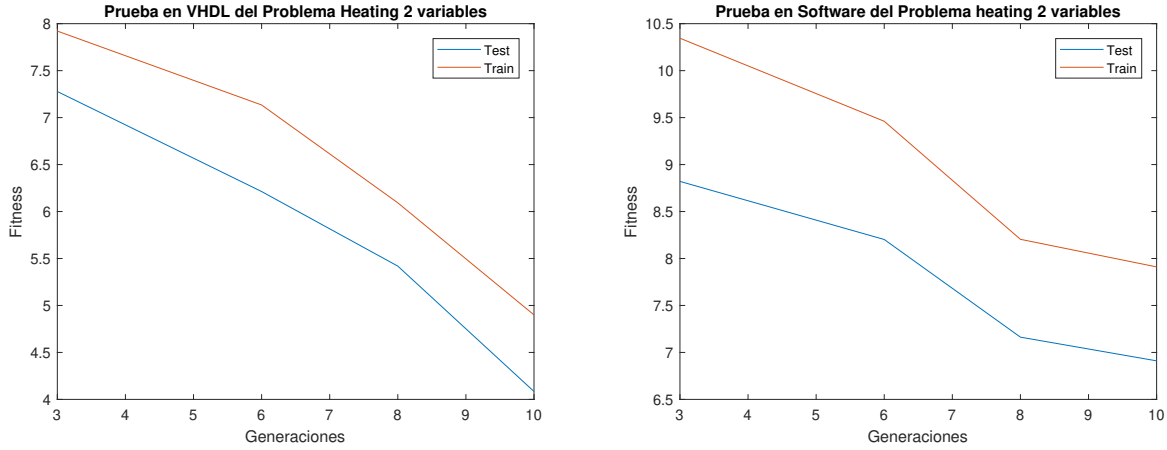


Figura 5.2: Fitness del Problema Heating a 2 variables, GSGP-VHDL Vs. GSGP-Software.

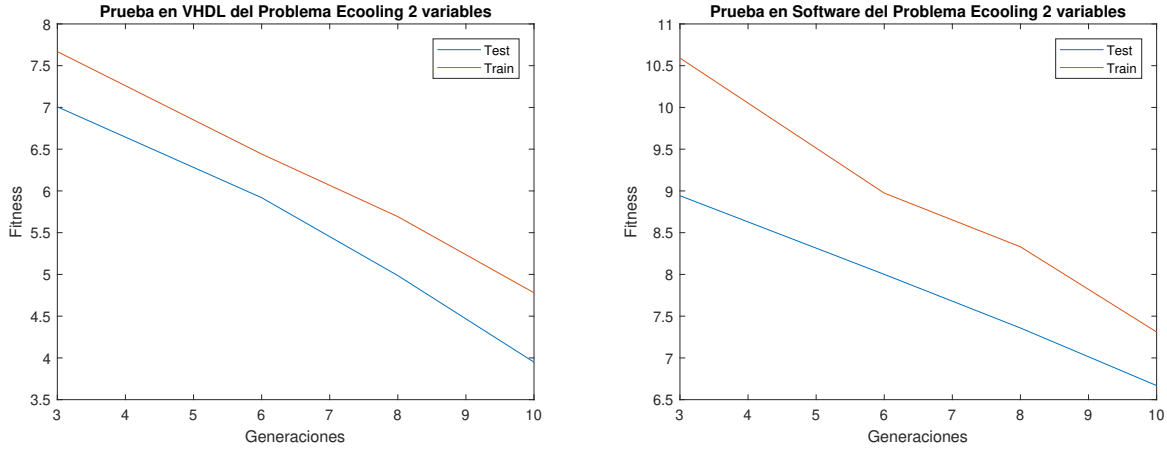


Figura 5.3: Fitness del Problema Ecooling a 2 variables, GSGP-VHDL Vs. GSGP-Software.

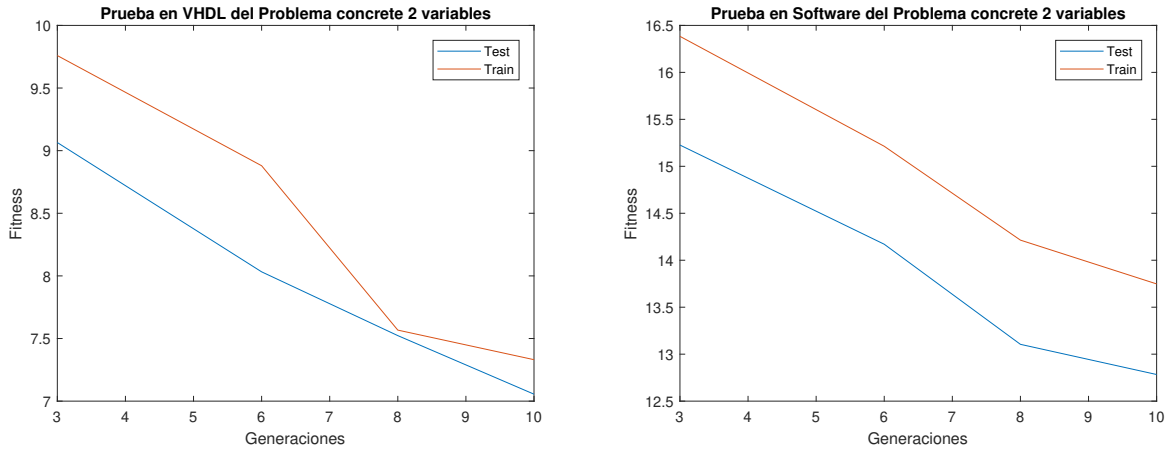


Figura 5.4: Fitness del Problema Concrete a 2 variables, GSGP-VHDL Vs. GSGP-Software.

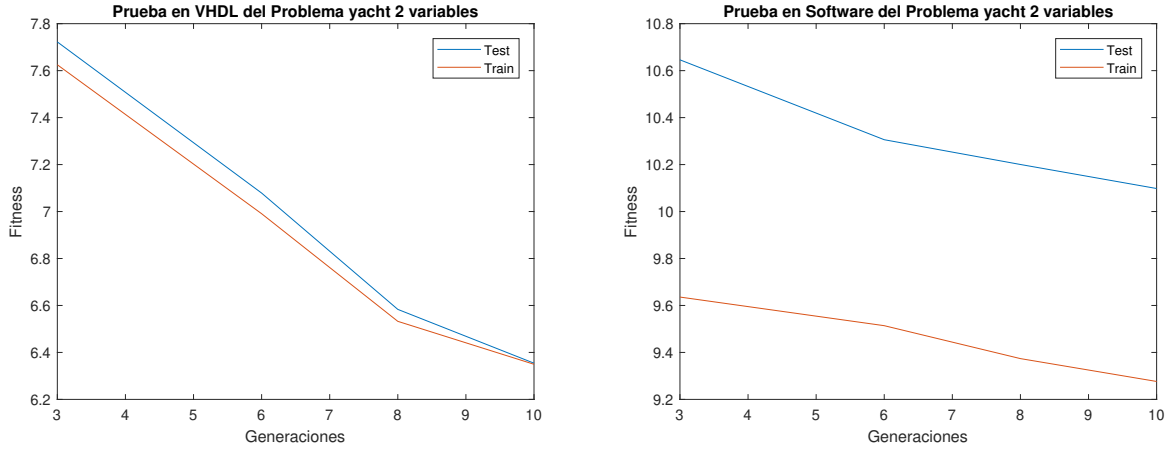


Figura 5.5: Fitness del Problema Yacht a 2 variables, GSGP-VHDL Vs. GSGP-Software.

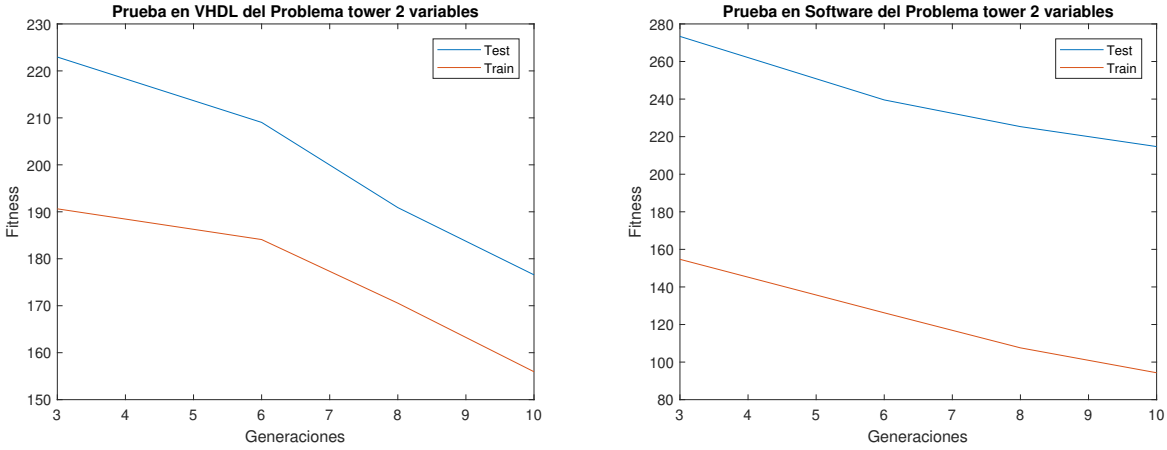


Figura 5.6: Fitness del Problema Tower a 2 variables, GSGP-VHDL Vs. GSGP-Software.

5.2. Pruebas con 12 variables

Las pruebas se realizaron con los parámetros descritos en la Tabla 5.2, para esta prueba se configuró el algoritmo con 3, 6, 8, 10 generaciones y 12 variables.

Los resultados obtenidos se incluyen en la Tabla 5.3, donde se muestra el rendimiento de ambos algoritmos con datos de entrenamiento, en la Tabla 5.4 se muestran los resultados de las pruebas con datos de prueba en cada unos de los algoritmos y en cada uno de los problemas. Al igual que en la prueba a 2 variables las tablas muestran la media, la mediana, STD en todas las corridas.

Al igual que el experimento mostrado anteriormente con 2 variables, se observa que GSGP-VHDL tiene un mejor rendimiento en comparación con el GSGP-Software, por lo tanto, se puede deducir que el procesador dedicado GSGP-VHDL es un mejor candidato para resolver este tipo de problemas.

Tabla 5.5: Comparación del GSGP-VHDL vs. GSGP-Software con 12 variables y datos de entrenamiento.

Generaciones	Problema	GSGP-VHDL			GSGP-SOFTWARE		
		Media	Mediana	STD	Media	Mediana	STD
3	Housing	18.403	18.403	3.158	20.402	20.953	4.953
	Tower	280.810	282.188	28.691	310.758	304.855	27.086
6	Housing	15.752	15.188	2.835	17.417	17.306	1.481
	Tower	256.074	262.125	28.243	283.664	289.433	26.157
8	Housing	11.549	11.375	2.251	12.442	12.266	1.431
	Tower	252.363	249.250	30.516	258.053	272.101	31.494
10	Housing	6.507	6.500	1.889	7.383	7.166	1.420
	Tower	217.208	214.313	25.900	219.950	220.504	28.533

Tabla 5.6: Comparación del GSGP-VHDL vs. GSGP-Software con 12 variables y datos de prueba.

Generaciones	Problema	GSGP-VHDL			GSGP-SOFTWARE		
		Media	Mediana	STD	Media	Mediana	STD
3	Housing	19.106	19.563	2.773	18.762	18.757	4.931
	Tower	292.683	289.063	29.965	332.957	328.133	30.318
6	Housing	16.343	16.375	2.895	17.026	16.890	1.948
	Tower	268.551	271.813	28.189	304.315	296.620	28.395
8	Housing	11.664	11.375	2.230	11.854	12.072	1.870
	Tower	264.940	264.940	30.471	273.675	288.564	32.951
10	Housing	6.366	6.366	1.975	6.804	6.538	2.590
	Tower	226.160	220.000	26.624	235.098	231.616	31.447

En las Figuras 5.7 y 5.8, se observan las comparaciones de ambos algoritmos en el comportamiento gradual de la aptitud conforme se incrementa la cantidad de generaciones desde 3 hasta 10, las gráficas muestran el fitness del algoritmo GSGP-VHDL y del algoritmo GSGP-Software con datos de entrenamiento (Train) y prueba (Test) para 2 de los problemas, esto con la finalidad de ver el comportamiento del algoritmo en situaciones mucho más complejas.

Se puede apreciar en las Figuras que los valores del fitness de GSGP-VHDL tienden a converger a 0 antes que los valores mostrados por GSGP-Software.

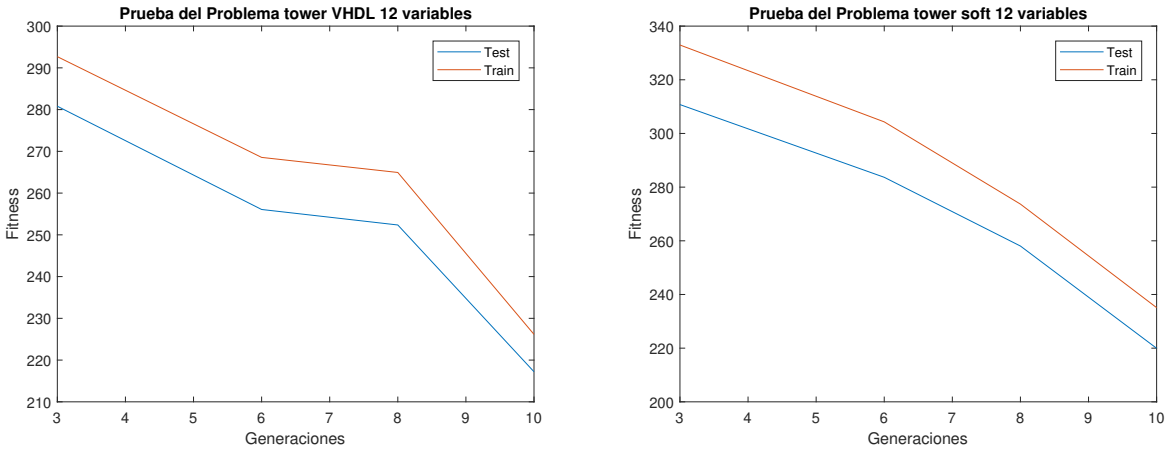


Figura 5.7: Fitness del Problema Tower a 12 variables, GSGP-VHDL Vs. GSGP-Software

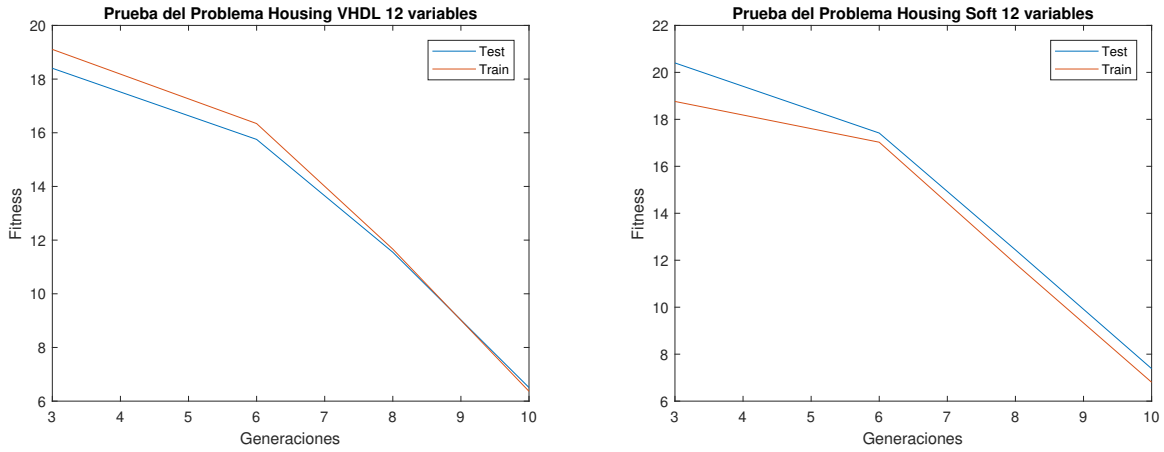


Figura 5.8: Fitness del Problema Housing a 12 variables, GSGP-VHDL Vs. GSGP-Software

Con base en los resultados obtenidos, se puede concluir que en términos de rendimiento y aptitud, el algoritmo GSGP-VHDL es bastante competitivo, logrando el error más bajo en todos problemas. Sin embargo, GSGP-Software logra un mejor resultado, con el error más bajo en uno de los seis problemas. Esta diferencia de rendimiento probablemente se deba al diferente enfoque que se le da a los arboles utilizados en la mutación por GSGP-VHDL y también el hecho de que el operador genético muta a toda población. Por lo tanto, es razonable suponer que el algoritmo GSGP-VHDL es muy eficaz en problemas de aprendizaje comunes, en términos de cantidad de instancias y características. Por lo que se considera al algoritmo GSGP-VHDL como una opción viable para abordar este tipo de problemas, teniendo en cuenta los resultados aquí reportados. En el Apéndice A, se muestran más resultados de experimentos con diferentes configuraciones.

5.3. Posibles aplicaciones en IoT del procesador dedicado GSGP-VHDL

De acuerdo a los resultados discutidos en este Capítulo, el procesador dedicado GSGP-VHDL es un dispositivo muy eficiente dada su rápidez en tiempo de ejecución en ciclos de reloj, mientras GSGP en una computadora tarda en resolver el problema en horas [6], a GSGP-VHDL le toma microsegundos y en algunos casos milisegundos, como se aprecia en las Tablas del Apéndice A. Con base en los resultados obtenidos, es posible proponer al

procesador dedicado GSGP-VHDL como candidato para aplicaciones de IoT como el caso de estudio presentado en [20].

El caso de estudio es un *Building Management System* (BMS) que tiene como objetivo automatizar la operación de las estructuras básicas de la instalación de un edificio, así como el control remoto de los edificios a través de una red WAN que consta de múltiples redes WPAN, una para cada piso del edificio. El estudio de caso describe un edificio de una empresa, que consta de habitaciones a las que los trabajadores tienen acceso durante las horas de trabajo (8.00 a 18.00).

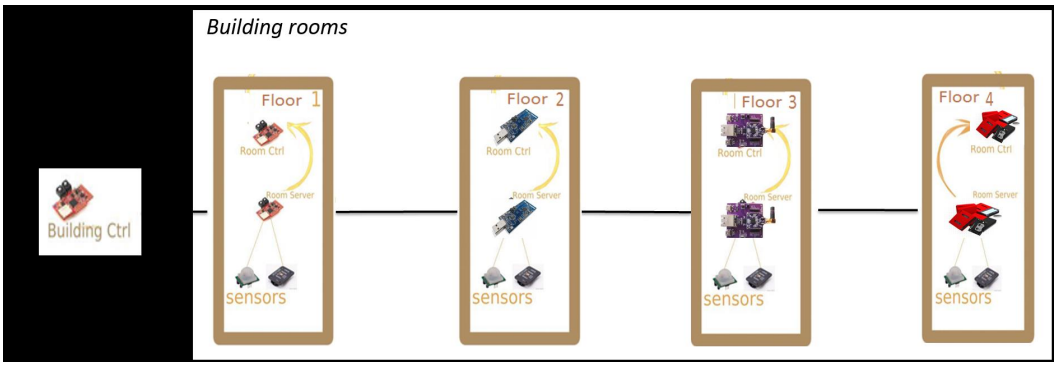


Figura 5.9: Topología de nodos en el sistema de gestión de edificios. [20].

La Figura 5.9 presenta la arquitectura heterogénea del sistema IoT, donde cada piso tiene dos dispositivos IoT únicos en las funciones de controlador de piso y servidor de piso. La heterogeneidad arquitectónica permite analizar el consumo de energía en una variedad de dispositivos IoT, como Zolertia Z1 (nivel 1), Sky mote (nivel 2), OpenMote (nivel 3) y SimpleLink Sensortag (nivel 4). Cada conjunto de dispositivos de piso comunica su estado al nivel inferior y el nivel más bajo (planta 1 en la Figura 5.9) reenvía todos los estados de planta al dispositivo de gestión de edificios ubicado en el nivel 1. Este dispositivo forma el supervisor del sistema central. Todos los controladores de habitación notifican su estado actual y se puede acceder o administrar de forma remota como parte de la red WAN. El estado de cada piso está determinado por un conjunto de recursos sensores/actuadores presentes en los dispositivos servidores de piso. [20].

Es en este tipo de problemas es que un procesador dedicado GSGP-VHDL puede ser una útil solución, dada la rapidez del procesador dedicado y que los datos se procesarían in situ.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En esta tesis se presentó el primer algoritmo de Programación Genética Geométrica Semántica, diseñado en VHDL para FPGAs. La implementación es un procesador dedicado GSGP-VHDL, el cual utiliza árboles de sintaxis con una representación de matriz *m x n*, permitiendo un mejor manejo de la población a la hora de paralelizar procesos, además como se mencionó en el Capítulo 4, los módulos se realicen de manera simultánea, aprovechando las ventajas de los FPGAs. En el Capítulo 5, se muestran las pruebas experimentales con 6 problemas sintéticos (Housing, Energy Heating, Energy Ecooling, Concrete, Yacht y Tower) de la literatura, se realizaron comparaciones entre el GSGP-Software y el procesador dedicado GSGP-VHDL, ambas versiones de GSGP fueron configuradas con diferentes variables y misma representación de punto fijo para los datos, obteniendo en promedio que el proceso completo del procesador dedicado GSGP-VHDL se realiza en microsegundos, mientras que el GSGP-Software es realizado en minutos e incluso en horas. Las pruebas experimentales en simulación del procesador dedicado GSGP-VHDL fueron realizadas en Vivado para un FPGA Basys 3, el rendimiento del procesador dedicado GSGP-VHDL provoca que el uso de recursos incremente, por lo que es importante elegir adecuadamente el FPGA que se va a utilizar.

Los resultados obtenidos demuestran que el procesador dedicado GSGP-VHDL es un sistema altamente competitivo en rendimiento, por lo que se considera factible su aplicación en resolución de problemas de IoT.

6.2. Trabajo Futuro

Actualmente se han llevado a cabo las pruebas de simulación del procesador dedicado GSGP-VHDL, sin embargo, se están realizando las pruebas de implementación en FPGAs Basys 3 y Cyclone de los módulos a partir de la evaluación inicial. Los resultados de estas pruebas permitirán mejorar el sistema en medida de rendimiento, diseño y código.

Las pruebas de simulación, fueron realizadas para problemas sintéticos del *estado-del-arte*, con datos almacenados en el FPGA, un reto pendiente es el desarrollo de un módulo que analice en tiempo real datos provenientes de distintas fuentes como sensores.

Otro reto es elegir problemas de IoT mediante el procesador dedicado GSGP-VHDL, como el propuesto en el Capítulo 5.

Dado que el procesador dedicado GSGP-VHDL fue diseñado en VHDL, puede ser implementado en cualquier dispositivo FPGA, por lo que realizar la implementación es una tarea pendiente.

Otro reto es la integración al procesador dedicado GSGP-VHDL, el módulo del evaluador explicado en el Capítulo 4, ya que por el momento los experimentos se realizaron por separado.

Referencias

[1] Advanced Micro Devices (AMD), I., https://www.amd.com 2022.

[2] Al-Fuqaha, A., Guizani, M., Soodkhah Mohammadi, M., Aledhari, M., and Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials 17* (2015), 2347–2376.

[3] Ayoubi, S., Limam, N., Salahuddin, M. A., Shahriar, N., Boutaba, R., Estrada Solano, F., and Caicedo Rendon, O. M. Machine learning for cognitive network management. *IEEE Communications Magazine 56* (2018), 158–165.

[4] Casildo, S. Estudio del algoritmo GSGP, 2019.

[5] Castelli, M., Silva, S., and Vanneschi, L. A C++ framework for geometric semantic genetic programming. *Genetic Programming and Evolvable Machines 16* (03 2014).

[6] Castelli, M., Vanneschi, L., Manzoni, L., and Popovič, A. Semantic genetic programming for fast and accurate data knowledge discovery. *Swarm and Evolutionary Computation 26* (2016), 1–7.

[7] Chakraborty, S. *Vivado Design Tools*. ResearchGate, Discover the world's research, 10 2017, pp. 17–21.

[8] Cui, L., Yang, S., Chen, F., Ming, Z., Lu, N., and Qin, J. A survey on application of machine learning for internet of things. *International Journal of Machine Learning and Cybernetics* (2018).

[9] Eiben, A., and Smith, J. Introduction to evolutionary computing. In *Natural Computing Series* (2003), vol. 45.

[10] Fernández López, S., López Péres, Alfonso, S., and Soto Campos, E. *Diseño de sistemas digitales con VHDL*. Editorial Paraninfo, 2002.

[11] Freire Rubio, M. *Introducción al lenguaje VHDL*, 1 ed. Universidad Politécnica de Madrid, 2010.

[12] García, J. G., and Lopez Ortigosa, R. A neural networks approach to residuary resistance of sailing yachts prediction. *Proceedings of the International Conference on Marine Engineering* (2007), 250.

[13] Goncalves, I., Silva, S., and Fonseca, C. On the generalization ability of geometric semantic genetic programming. In *Genetic Programming - 18th European Conference, EuroGP 2015, Proceedings* (2015), vol. 9025 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer-Verlag, pp. 41–52. 18th European Conference on Genetic Programming, EuroGP 2015 ; Conference date: 08-04-2015 Through 10-04-2015.

[14] Goribar, C., Maldonado, Y., and Trujillo, L. *Automatic Random Tree Generator on FPGA*. Springer International Publishing, Cham, 2017, pp. 89–104.

[15] Goribar-Jimenez, C., Maldonado, Y., Trujillo, L., Castelli, M., Gonçalves, I., and Vanneschi, L. Towards the development of a complete GP system on an FPGA using geometric semantic operators. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (2017), pp. 1932–1939.

[16] Grediaga, A. VHDL. Tipos de Datos. Universidad de Alcalante, 2007.

[17] IEEE. *The IEEE Standard VHDL Language Reference Manual*, 2000 ed. Institute of Electrical and Electronics Engineers, 2020.

[18] Intel, I., https://www.intel.la 2022.

[19] Intel, I. Software de diseño intel quartus prime, 2022.

[20] Lekidis, A., and Katsaros, P. Model-based design of energy-efficient applications for IoT systems. In *1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD) @ETAPS* (2018).

[21] Maxines, D., and Alcalá, J. *VHDL. El Arte de Programar Sistemas Digitales*, 1 ed. Continental, 2002.

[22] Mohammad, S., Mohammadreza, R., Mohammadamin, B., Peyman, A., Payam, B., and Amit, P. S. Machine learning for internet of things data analysis: a survey. *Digital Communications and Networks 4*, 3 (2018), 161–175.

[23] Moraglio, A., Krawiec, K., and Johnson, C. G. Geometric semantic genetic programming. In *ParallelProblem Solving from Nature, PPSN XII (part 1)* (2012), vol. 7491, Springer, Berlin, Heidelberg, p. 21–31.

[24] Moraglio, A., Krawiec, K., and Johnson, C. G. Geometric semantic genetic programming. *Parallel Problem Solving from Nature - PPSN XII* (2012), 21–31.

[25] Moreno Parra, R. A. Programación genética: La regresión simbólica. *Entramado* (2007).

[26] Olloz, S., Villar, E., Torroja, Y., and Teres, L. *VHDL Lenguaje estándar de diseño electrónico*, 1 ed. McGraw-Hill/Interamericana de España, S. A. U., España, 1998.

[27] Poli, R., Langdon, W. B., and Freitag, M. N. *A Field Guide to Genetic Programming*, 1 ed. Lulu Enterprises, United Kingdom, 2008.

[28] Qin, Y., Sheng, Q. Z., Falkner, N. J., Schahram, D., Wang, H., and Vasilakos, A. V. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications 64* (2016), 137–153.

[29] Quinlan, J. Combining instance-based and model-based learning. *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts* (1993), 236–243.

[30] Rushton, A. *VHDL For Logic Synthesis*, 3 ed. John Wiley & Sons, Ltd., United Kingdom, 2011.

[31] Sánchez Élez, M. *Introducción a la Programación en VHDL*, 1 ed. Facultad de informática, Universidad Complutense de Madrid, 2014.

[32] Shafique, K., Khawaja, B. A., Sabir, F., Qazi, S., and Mustaqim, M. Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *IEEE Access 8* (2020), 23022–23040.

[33] Shuker, M., and Mohamad, A. A. H. A study of efficient power consumption wireless communication techniques/ modules for internet of things (iot) applications. In *IoT 2016* (2016).

[34] Tsanas, A., and Xifara, A. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings 49* (2012), 560–567.

[35] Vanneschi, L., Castelli, M., Manzoni, L., and Silva, S. A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In *Proceedings of the 16th European conference on Genetic Programming* (04 2013), pp. 205–216.

[36] Vanneschi, L., Schütze, O., Trujillo, L., Legrand, P., and Maldonado, Y. *An Introduction to Geometric Semantic Genetic Programming*. Springer International Publishing, Cham, 2017, pp. 3–42.

[37] Vipin, K., and Fahmy, S. A. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. *ACM Comput. Surv. 51*, 4 (2018), 1–39.

[38] Vladislavleva, E. J., Smits, G. F., and Hertog, D. D. Order of nonlinearity as a complexity measure for models generated by symbolic re- gression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation 13*, 2 (2009), 333–349.

[39] Xilinx, I. 7 series FPGAs data sheet: Overview, 2020.

[40] Xilinx, I., www.xilinx.com 2022.

[41] Xu, Z., and Homer Saleh, J. Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. *Reliability Engineering and System Safety 211* (2021), 107530.

[42] Yeh, I.-C. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research 28*, 12 (1998), 1797–1808.

Apéndice A

Pruebas Completas

En este apartado, se muestran los resultados obtenidos de los experimentos realizados para los problemas mostrados en la Tabla 5.1, se muestra una comparación del GSGP-Software Vs. GSGP-VHDL.

Las tablas y gráficas mostradas en esta sección se realizaron con los parámetros descritos en la Tabla 5.2, para esta prueba se probó el algoritmo con 3, 6, 8, 10 generaciones y 12 variables.

A.1. Pruebas con 2 variables

Tabla A.1: Problema Housing resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.Housing VHDL 2 variables

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	3.8125	4.8125	1	5.875	6.625	1	7.125	6.625	1	6.3125	7.8125
2	5.9375	5.4375	2	6.4375	6.5	2	6.75	7.9375	2	6.9375	7.625
3	3.375	4.9375	3	5.1875	6.5625	3	5.875	7.625	3	7.875	9
4	5.8125	8.375	4	6.3125	5.875	4	7.1875	9.0625	4	7.0625	7.1875
5	5	6	5	4.8125	6.5	5	5.9375	6.875	5	7.3125	8.1875
6	3.5	3.9375	6	6.0625	7.6875	6	6.375	7.3125	6	6.8125	8.3125
7	3.9375	6.625	7	6.4375	7.5	7	6.0625	5.9375	7	6.25	6.4375
8	5.625	5.5	8	5.1875	6.125	8	5.9375	5.625	8	6.125	6.25
9	5.75	7.325	9	4.6875	5.75	9	7.0625	8.25	9	6.5625	8.3125
10	3.6875	6.625	10	5.25	6.5	10	5.75	5.875	10	6.4375	6.1875
11	3	2.9375	11	4.75	6.5625	11	7.4375	7.3125	11	6.25	8
12	3.125	3.9375	12	6.1875	7.9375	12	6.875	7.125	12	7	6.5625
13	5.875	6.25	13	4.8125	5.1875	13	6.125	7.3125	13	7	8.1875
14	3.3125	4.125	14	4.375	5.875	14	6.875	7	14	6.8125	6.4375
15	4.75	5.375	15	4.5	5.125	15	5.5	5.0625	15	6	6.4375
16	5.75	5.625	16	6.4375	6.375	16	5.5	7.3125	16	7	7.9375
17	4.5	7.1875	17	4.0625	4.625	17	6.1875	7.625	17	7.8125	9.25
18	3.125	4.125	18	4	3.875	18	6.5	6.8125	18	7.75	8.625
19	3	2.8125	19	6.3125	7.75	19	6.8125	6.875	19	7.6875	7.125
20	5.4375	5.625	20	6.5625	6.75	20	6	7.8125	20	7.9375	8.375
21	5.9375	6.0625	21	5.5	6.4375	21	6.0625	6.6875	21	7.875	9
22	5.125	5.125	22	5.9375	7	22	6.0625	7.8125	22	7.4375	7.0625
23	5	5.0625	23	4.125	4.6875	23	6.5625	8	23	6.5625	5.9375
24	5.9375	6	24	6.5625	7.3125	24	7.1875	6.75	24	6	6.0625
25	3.5	4.125	25	4.625	5.1875	25	5.875	6.4375	25	6.25	5.3125
26	5.4375	7.1875	26	6.4375	6.1875	26	7.0625	7.9375	26	7.5625	9.3125
27	4.3125	7.125	27	4.3125	3.8125	27	7	8.4375	27	7.0625	7.6875
28	5.6875	7.0625	28	4.4375	4.0625	28	6	7.625	28	6.25	6.625
29	3.9375	6.5	29	4	3.75	29	6.5	8.25	29	7.3125	8.9375
30	3.875	5.8125	30	5.625	6.5625	30	7.25	7.625	30	7.625	7.125
mean	4.5902778	5.6462963	mean	5.270833333	5.962963	mean	6.4328704	7.212963	mean	6.9537037	7.4398148
clk	10.505 us	9.745 us	clk	9.67 us	7.235 us	clk	7.605 us	6.515 us	clk	6.925 us	5.965 us

Tabla A.2: Problema Housing resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	6.417	6.846	1	6.469	8.408	1	7.665	9.155	1	8.958	9.574
2	7.359	7.855	2	7.181	8.79	2	8.946	11.15	2	10.614	11.576
3	7.203	7.548	3	6.441	6.597	3	7.687	9.474	3	9.243	10.299
4	6.238	6.982	4	8.51	10.329	4	6.763	7.268	4	9.19	10.154
5	5.669	7.094	5	7.424	8.598	5	7.382	9.509	5	10.616	13.331
6	6.703	7.758	6	6.323	6.385	6	7.945	8.871	6	9.555	11.077
7	7.936	9.012	7	6.006	7.154	7	6.471	7.561	7	7.972	8.84
8	7.362	8.349	8	6.022	6.062	8	7.58	8.386	8	10.176	11.465
9	7.603	8.907	9	6.985	7.981	9	8.746	8.975	9	9.332	11.47
10	6.459	7.038	10	8.248	9.484	10	6.689	6.744	10	10.702	12.291
11	5.892	6.043	11	8.068	9.67	11	8.907	8.939	11	6.904	7.744
12	6.395	6.616	12	6.243	7.211	12	8.825	9.72	12	10.626	11.124
13	6.149	6.37	13	5.742	6.686	13	6.291	8.445	13	6.306	7.919
14	6.661	6.715	14	7.975	8.386	14	6.25	8.439	14	9.699	11.556
15	7.553	8.965	15	7.323	8.056	15	7.796	8.338	15	10.194	11.933
16	5.202	5.926	16	5.757	7.481	16	8.062	8.203	16	8.016	9.616
17	6.89	7.735	17	6.101	7.573	17	8.68	9.637	17	7.053	7.525
18	6.975	7.096	18	7.024	7.874	18	7.188	8.103	18	9.056	10.526
19	7.487	8.967	19	7.715	8.846	19	6.773	7.274	19	9.779	12.002
20	7.954	8.705	20	5.952	7.926	20	6.631	7.529	20	6.826	7.843
21	5.477	5.491	21	6.811	7.522	21	8.691	8.796	21	8.96	9.923
22	5.299	6.381	22	6.785	6.847	22	7.48	8.448	22	10.541	12.076
23	5.857	6.185	23	8.069	9.886	23	8.505	10.856	23	9.749	12.025
24	7.549	8.991	24	8.145	9.266	24	8.933	11.213	24	8.717	9.22
25	6.024	6.221	25	6.096	7.871	25	8.605	9.447	25	9.289	10.273
26	7.478	7.745	26	7.787	9.135	26	6.822	7.586	26	9.11	9.315
27	7.436	8.78	27	6.019	7.368	27	8.373	8.638	27	7.161	7.29
28	7.809	8.839	28	7.367	9.2	28	6.695	7.674	28	7.868	10.245
29	5.311	6.067	29	6.046	6.395	29	8.186	10.551	29	7.441	7.893
30	5.562	6.066	30	5.667	6.515	30	7.776	9.852	30	10.615	13.158
mean	6.6270370	7.372	mean	6.896666667	7.9891481	mean	7.6683333	8.7037777	mean	8.942703704	10.290148

Tabla A.3: Problema Heating resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	3.875	4.8125	1	5.625	4.9375	1	6.3125	5.9375	1	8.8125	8.5
2	4.75	5.9375	2	6.688	7.8125	2	5.5	6.0625	2	7.3125	8.6875
3	3.5	3.3125	3	4.063	4.75	3	6.9375	5.25	3	8.5	7.8125
4	3	5.8125	4	4.563	4.3125	4	5.0625	6.25	4	7.375	8.375
5	5.0625	6.875	5	6.250	6.0625	5	5.5625	6.375	5	6.5625	7.3125
6	4.9375	5.25	6	5.563	7.6875	6	6.5625	8.8125	6	6.5625	6
7	3.0625	3.625	7	6.188	7.125	7	5.125	5.375	7	8.1875	9.5
8	4.4375	5	8	5.813	6.0625	8	5.8125	7.3125	8	8.8125	9.25
9	4.625	5.5	9	5.063	6.9375	9	5.4375	7	9	8.375	8.875
10	4.375	6.9375	10	6.063	6.875	10	6.4375	7.5625	10	7.6875	8.8125
11	3.6875	4.5	11	4.750	4.3125	11	6.1875	7.4375	11	7.8125	8.3125
12	3.8125	4.125	12	5.563	7.0625	12	7.6875	7.4375	12	7.25	9.9375
13	4.9375	4.8125	13	4.500	6.4375	13	5.1875	6.0675	13	6.875	6.9375
14	3.4	3.8125	14	5.563	5.3125	14	5.4375	7.5	14	6.6875	7
15	4.5	4.9375	15	4.313	6.9375	15	6.5625	7.1875	15	7.1875	7
16	3.9375	4.125	16	5.438	6.5625	16	5.25	7.1875	16	8.9375	9.125
17	4.0625	4.6875	17	5.375	6.25	17	5.6875	5.5625	17	6.75	7
18	4.75	4.9375	18	6.125	7.5	18	6.75	5.1875	18	8.375	8.6875
19	3.75	3.3125	19	5.500	6.375	19	6.875	8.5625	19	6.5	7.6875
20	3.625	4.5625	20	5.000	4.25	20	5.8125	6.375	20	6.25	5.875
21	4.6875	4.25	21	4.875	3.875	21	6.75	6.6875	21	6	8.6875
22	4.6875	3.6875	22	6.375	6.25	22	5.6875	6.5	22	6.625	7.8125
23	3.25	5.875	23	6.813	6.875	23	5.375	5.5625	23	8.875	8.875
24	3.0625	4.125	24	4.250	4.8125	24	7.875	8.6875	24	6.4375	6.8125
25	4.75	5.5625	25	4.313	4.3125	25	7.5	8.8125	25	6.625	7.75
26	4.1875	5.4375	26	4.750	4.1875	26	7.25	9.875	26	6.9375	5.0625
27	3.5	4.5	27	5.688	7.6875	27	6.0625	7.125	27	6.125	8.6875
28	4.625	5.0625	28	5.563	6.6875	28	7.125	7.25	28	7.375	7.875
29	3.125	5.4375	29	5.875	5.3125	29	5.9375	7.4375	29	6.6875	7.375
30	4.4375	5.542	30	6.188	8.4375	30	6.75	7.5	30	8.625	9.25
mean	4.08425926	4.8997037	mean	5.418981481	6.0925926	mean	6.21296296	7.13444444	mean	7.27777778	7.9212963
clk	10.445 us	9.725 us	clk	9.55 us	7.125 us	clk	7.435 us	6.525 us	clk	6.965 us	5.925 us

Tabla A.4: Problema Heating resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	6.53	8.227	1	6.051	6.704	1	6.669	6.963	1	9.782	12.621
2	6.788	7.106	2	6.408	7.457	2	6.525	8.986	2	6.633	9.428
3	5.459	6.856	3	7.733	7.742	3	6.341	8.321	3	6.822	7.271
4	7.582	8.009	4	7.464	8.304	4	8.091	8.438	4	9.058	11.884
5	7.553	8.335	5	6.674	8.655	5	9.964	12.004	5	7.241	10.212
6	7.24	7.634	6	8.028	8.811	6	6.845	9.277	6	7.415	9.158
7	5.414	6.906	7	7.229	9.01	7	6.739	8.515	7	8.26	9.65
8	6.964	8.339	8	6.339	7.417	8	8.901	10.035	8	10.234	12.941
9	7.293	8.568	9	6.808	7.979	9	8.291	9.657	9	10.139	10.255
10	5.44	5.644	10	7.312	8.3	10	8.465	9.01	10	8.276	10.831
11	6.083	6.145	11	6.419	8.178	11	9.947	10.667	11	9.374	11.175
12	8.126	9.453	12	7.864	9.342	12	8.311	8.837	12	10.332	11.206
13	8.475	8.733	13	6.114	7.569	13	6.24	7.298	13	8.697	9.448
14	6.463	8.152	14	8.229	8.252	14	9.899	10.879	14	9.351	9.534
15	6.693	8.158	15	6.131	7.112	15	8.484	9.579	15	7.964	8.859
16	5.603	7.337	16	6.774	8.145	16	8.128	8.449	16	9.058	9.344
17	8.431	8.861	17	8.044	8.634	17	9.406	11.401	17	7.581	9.226
18	5.59	7.056	18	8.586	8.705	18	8.122	8.262	18	7.495	7.66
19	6.868	7.57	19	6.399	7.918	19	6.343	6.472	19	8.51	10.624
20	7.185	8.596	20	7.84	9.102	20	7.874	9.824	20	9.18	11.664
21	7.197	8.398	21	6.14	6.632	21	7.144	9.38	21	7.619	8.66
22	7.63	8.751	22	7.051	8.736	22	9.689	11.582	22	9.669	11.561
23	8.463	8.814	23	6.291	6.805	23	7.487	9.097	23	8.571	11.226
24	5.943	6.786	24	8.729	8.758	24	8.461	10.093	24	6.511	7.606
25	6.685	7.295	25	7.254	7.459	25	6.254	7.67	25	10.072	10.53
26	5.269	5.978	26	6.136	8.084	26	8.715	10.462	26	8.345	11.345
27	6.39	6.964	27	6.766	7.711	27	9.826	10.974	27	10.82	13.776
28	7.957	8.539	28	8.03	8.766	28	7.355	8.688	28	10.104	11.628
29	7.511	9.675	29	6.011	6.881	29	7.331	8.091	29	9.541	10.487
30	6.532	8.89	30	8.732	10.251	30	9.152	10.803	30	8.745	8.829
mean	6.91037037	7.91059259	mean	7.162740741	8.204296	mean	8.20237037	9.46088889	mean	8.82081481	10.3451481

Tabla A.5: Problema Ecooling resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	3.5625	4.4375	1	4.938	6.063	1	6.75	6.938	1	7.1875	7.0625
2	4.25	5.375	2	4.750	5.250	2	6.9375	8.063	2	7.75	9.1875
3	4.3125	4.625	3	4.500	5.688	3	6.1875	7.063	3	6.6875	7.3125
4	3.5625	4.5	4	5.625	5.875	4	6.3125	6.625	4	7.9375	9.3125
5	3.1875	3.6875	5	5.875	5.875	5	5.75	5.688	5	5.8125	5.75
6	3.1875	3.3125	6	4.563	5.063	6	4.8125	5.250	6	7.125	7.1875
7	3.75	5.6875	7	4.500	4.438	7	7	7.188	7	7.8125	9.25
8	3.125	2.8125	8	5.000	5.500	8	5.1875	5.188	8	7.8125	7.9375
9	3	4.4375	9	5.875	6.563	9	6.0625	6.563	9	8.0625	8.5625
10	4.0625	5.625	10	3.875	5.000	10	6.0625	6.688	10	6.4375	6.5625
11	5	4.875	11	4.563	4.938	11	6.1875	6.125	11	6.875	7.25
12	3.5	4.3125	12	5.813	6.000	12	6	7.063	12	6.8125	6.625
13	3.125	2.875	13	4.125	5.563	13	5.125	5.563	13	6.5625	7.9375
14	4.6875	5.6875	14	4.063	5.438	14	6.5625	7.125	14	6.5	7.75
15	4.3125	5.625	15	4.313	4.688	15	4.875	6.000	15	6.25	7.625
16	4	4.5	16	5.875	7.188	16	6.75	7.063	16	7.0625	8.25
17	4.1875	4.375	17	4.875	5.438	17	5.5625	6.063	17	7.0625	8.4375
18	4.9375	6.6875	18	5.750	5.563	18	7	7.813	18	6.5625	6.5
19	3.875	5.5	19	4.313	5.688	19	6.5625	7.500	19	8.1875	9.4375
20	4.5625	6.5625	20	5.688	6.188	20	6.25	6.500	20	6.375	6.1875
21	4.5625	6.125	21	5.375	5.625	21	7.125	8.000	21	7.6875	8.375
22	3.5	4.125	22	4.188	5.125	22	5.6875	6.688	22	6.875	7.3125
23	4.0625	3.8125	23	5.625	6.438	23	5.4375	5.375	23	6.4375	6.75
24	4.75	4.625	24	6.188	7.188	24	5.0625	5.063	24	7.5625	8.625
25	3.375	4.6875	25	4.563	5.563	25	6.125	7.063	25	7.8125	9.25
26	4.4375	5.9375	26	4.688	6.063	26	5	6.000	26	7.6875	7.9375
27	4.875	5.625	27	3.938	4.688	27	5.1875	5.938	27	5.8125	6.5625
28	3.4375	5	28	6.000	7.375	28	5.6875	6.000	28	7.25	7.125
29	3.75	3.5	29	5.000	5.438	29	5.625	6.688	29	5.9375	5.9375
30	3.8125	4.5625	30	4.375	5.188	30	6.8125	7.125	30	6.875	8.0625
mean	3.94907407	4.78009259	mean	4.986111111	5.69212963	mean	5.91898148	6.44212963	mean	7.00694444	7.66875
clk	10.445 us	9.725 us	clk	9.55 us	7.125 us	clk	7.435 us	6.525 us	clk	6.965 us	5.925 us

Tabla A.6: Problema Ecooling resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	7.009	7.963	1	8.451	8.485	1	8.479	9.166	1	10.226	10.528
2	7.843	8.829	2	7.088	8.534	2	8.742	9.823	2	8.024	10.942
3	5.927	5.943	3	8.168	9.310	3	7.695	8.695	3	10.268	11.426
4	5.944	6.629	4	6.804	7.878	4	8.33	9.353	4	10.628	11.71
5	7.224	7.48	5	7.716	7.773	5	7.845	9.728	5	7.632	7.724
6	6.261	6.578	6	8.365	9.142	6	7.163	8.639	6	7.988	10.756
7	7.562	7.856	7	8.478	9.721	7	8.573	9.875	7	10.994	13.02
8	5.546	5.989	8	8.268	9.136	8	8.516	9.06	8	9.736	11.158
9	6.072	6.098	9	7.909	9.677	9	8.181	8.89	9	7.543	8.979
10	5.901	6.625	10	7.320	8.582	10	8.193	8.7	10	9.188	12.101
11	7.244	7.563	11	6.687	8.035	11	8.016	9.514	11	8.381	8.99
12	5.756	6.613	12	7.144	7.909	12	7.862	8.236	12	6.941	8.407
13	7.058	7.774	13	7.511	7.892	13	8.916	10.915	13	10.761	12.054
14	7.802	7.984	14	7.305	8.163	14	7.247	7.342	14	8.324	10.513
15	7.084	8.432	15	7.228	7.630	15	7.63	8.465	15	8.494	9.45
16	6.385	7.103	16	7.816	9.702	16	8.527	8.754	16	8.412	10.141
17	6.263	6.774	17	6.274	6.686	17	8.885	9.808	17	9.756	11.539
18	6.518	7.759	18	6.260	6.497	18	7.015	7.683	18	8.411	11.16
19	7.483	8.04	19	8.047	9.780	19	7.066	9.208	19	10.252	12.61
20	7.872	7.96	20	6.862	7.250	20	8.967	10.736	20	9.794	10.089
21	7.811	8.713	21	8.360	9.844	21	7.77	8.762	21	10.344	11.567
22	6.403	7.074	22	6.802	7.425	22	7.691	9.581	22	7.573	9.517
23	5.558	6.913	23	6.545	7.350	23	7.44	7.683	23	8.787	8.941
24	5.577	6.296	24	7.991	9.638	24	7.05	7.128	24	7.799	9.29
25	7.252	8.101	25	6.745	7.383	25	8.704	10.59	25	10.22	12.667
26	7.327	7.929	26	7.386	7.986	26	7.157	7.429	26	7.287	9.465
27	6.023	7.516	27	7.269	7.916	27	7.79	8.848	27	10.133	12.165
28	7.289	8.066	28	6.545	8.347	28	8.29	8.424	28	10.437	12.471
29	6.963	7.191	29	7.528	8.928	29	8.248	9.953	29	6.595	8.997
30	5.879	6.324	30	7.509	8.663	30	8.952	8.998	30	9.042	9.365
mean	6.66877778	7.31037037	mean	7.358296296	8.3308519	mean	8.00088889	8.97414815	mean	8.94266667	10.5914

Tabla A.7: Problema Concrete resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	7.3125	7.6875	1	7.4375	7	1	7.5625	8.125	1	9.4375	9.875
2	6.875	6.5	2	8.125	8.1875	2	7.8125	7.3125	2	10.0625	11.4375
3	7.5625	7.9375	3	6.875	6	3	7.4375	8.5	3	8.3125	9.75
4	6.3125	6.6875	4	8.0625	7.4375	4	7.125	8.375	4	9.75	9.3125
5	8.125	8.8125	5	7.875	8.1875	5	8.875	10.125	5	8.25	10.0625
6	7.875	8.1875	6	7.9375	8.1875	6	8.9375	9.25	6	8.8125	10.3125
7	7.75	7.3125	7	8	8.1875	7	8.125	8.125	7	8.875	10
8	5.9375	6.5625	8	6.8125	6.0625	8	7.4375	7.75	8	9.0625	8.75
9	6.75	7.0625	9	6.5	6.3125	9	8.5625	10.0625	9	9.8125	9.25
10	6.3125	6.25	10	8.25	7.75	10	8.9375	9.1875	10	8.6875	10.5625
11	5.875	6.25	11	7.4375	8.5625	11	6.875	6.25	11	8.5	9.125
12	7	6.75	12	8.125	8.125	12	7.375	9.1875	12	8.8125	10.5
13	6.5625	7.125	13	6.75	7.75	13	8.6875	8.8125	13	9.6875	9.1875
14	5.9375	6.25	14	7.6875	8.0625	14	7.1875	8.5625	14	10.125	9.375
15	7.8125	7.5	15	8.4375	9.375	15	8.6875	9.625	15	9.5625	11.3125
16	6.75	6.625	16	7.125	6.875	16	7.25	7.1875	16	9.1875	11.0625
17	6.5625	6.375	17	8.25	8.5625	17	8.125	9.5	17	9.4375	11.25
18	7.6875	8.625	18	6.5	7.625	18	7.875	7.625	18	8.375	10.3125
19	7.5	8.125	19	8	7.1875	19	8.5625	10.1875	19	8.0625	8.1875
20	7.25	7.5	20	6.6875	6.4375	20	8.3125	10.0625	20	9.6875	9.75
21	7.375	7.6875	21	7	7.1875	21	8.25	9.625	21	9.4375	8.6875
22	7.125	6.625	22	7.75	8.5625	22	7.0625	6.375	22	9.1875	8.5
23	7.6875	8.0625	23	6.5625	7.6875	23	8.4375	9.5625	23	9.5	10.125
24	7.5	7.875	24	6.9375	6.375	24	7.625	7.875	24	8.6875	9.25
25	7.75	8.375	25	8.4375	8	25	7.3125	7.75	25	9.875	11.75
26	7.3125	8	26	8.3125	7.75	26	8.75	10.25	26	8	8.9375
27	7.4375	8.0625	27	7.5625	7.0625	27	8.75	10.25	27	7.9375	9.4375
28	7	7.1875	28	7.4375	6.4375	28	8.4375	9.9375	28	9.125	8.5
29	7	7.1875	29	7.3125	6.75	29	7.3125	8.6875	29	9.625	11.5625
30	6.3125	6.875	30	7.375	7.8125	30	8	9.5625	30	8.6875	8.4375
mean	7.05555556	7.33101852	mean	7.52314815	7.56712963	mean	8.03240741	8.87962963	mean	9.06481481	9.75925926
clk	10.505 us	9.745 us	clk	9.67 us	7.235 us	clk	7.605 us	6.515 us	clk	6.925 us	5.965 us

Tabla A.8: Problema Concrete resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	12.067	13.034	1	13.857	15.538	1	13.214	14.345	1	14.07	15.811
2	12.083	13.911	2	12.505	13.054	2	13.217	14.556	2	14.632	14.904
3	12.02	12.261	3	12.791	14.523	3	14.794	16.085	3	15.112	15.533
4	12.952	13.548	4	12.942	14.575	4	14.612	14.978	4	15.302	17.269
5	13.255	14.005	5	13.041	13.809	5	14.251	14.798	5	14.676	15.979
6	12.495	12.64	6	13.146	14.209	6	14.388	15.039	6	15.926	17.85
7	12.567	14.408	7	13.65	15.504	7	13.705	15.487	7	15.446	17.266
8	12.306	13.567	8	13.89	15.275	8	14.256	15.796	8	15.686	17.529
9	12.259	13.482	9	12.567	14.165	9	13.165	14.097	9	15.927	15.977
10	13.287	14.947	10	12.91	14.422	10	14.331	15.39	10	15.997	17.658
11	13.072	13.385	11	13.636	15.614	11	15.084	16.514	11	15.278	15.331
12	13.074	13.24	12	12.698	13.934	12	14.348	16.258	12	15.833	16.206
13	12.631	13.163	13	12.546	14.389	13	14.622	16.407	13	15.12	16.72
14	13.485	15.002	14	12.67	13.778	14	13.331	14.248	14	14.409	14.876
15	12.54	14.496	15	13.538	14.879	15	14.784	15.647	15	15.413	15.843
16	12.422	13.511	16	12.611	13.617	16	14.396	15.128	16	15.683	17.498
17	12.651	14.448	17	13.294	13.952	17	13.249	14.172	17	14.695	16.056
18	13.023	14.198	18	12.523	13.222	18	14.359	14.417	18	14.084	16.029
19	13.469	14.448	19	12.576	13.79	19	13.324	13.92	19	15.228	16.531
20	13.03	14.378	20	13.147	13.844	20	14.589	15.947	20	15.89	16.007
21	12.153	13.764	21	12.838	13.263	21	13.909	14.774	21	15.204	15.494
22	13.4	14.831	22	13.596	15.388	22	14.358	14.695	22	14.034	16.03
23	12.069	12.234	23	13.142	14.101	23	14.106	15.51	23	15.713	17.285
24	13.308	14.02	24	13.85	14.623	24	14.139	15.132	24	14.038	15.798
25	12.397	13.539	25	13.09	13.528	25	14.324	14.715	25	14.673	15.766
26	12.22	12.326	26	13.105	14.675	26	14.16	15.109	26	15.943	17.535
27	12.835	13.237	27	12.906	13.147	27	15.071	16.216	27	15.245	15.801
28	12.961	14.11	28	13.08	13.645	28	14.597	16.27	28	14.751	15.863
29	13.006	13.28	29	12.989	14.019	29	13.787	14.872	29	15.79	16.483
30	12.286	12.987	30	13.848	14.414	30	13.356	15.223	30	15.126	15.679
mean	12.8543425	13.6812	31	13.16775862	13.853874	31	14.2117954	15.281069	31	15.1458989	16.1653103

Tabla A.9: Problema Yatch resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	5.5	5.6875	1	5.5625	5.4375	1	7.8125	7.625	1	7	7.5
2	7.125	6.875	2	6.25	6.5	2	7.9375	8.1875	2	8.5625	8.75
3	6.4375	6.5625	3	5.625	5.3125	3	6.875	7.1875	3	7.1875	7.6875
4	5.6875	5.8125	4	6.125	6.0625	4	8.125	7.875	4	7.1875	7.4375
5	5.5625	5.625	5	6.5625	6.5	5	7.5	7.4375	5	7.75	7.3125
6	7.1875	7.375	6	7.3125	7.125	6	6.75	6.625	6	8.75	7.875
7	6.25	5.9375	7	5.875	6.375	7	6.375	5.875	7	8.9375	8.5625
8	6.5625	6.5625	8	7.0625	6.5625	8	6.25	6.125	8	8.375	8.125
9	6.5	6.4375	9	6.1875	6.3125	9	6.875	6.5625	9	8.1875	8.125
10	7.125	7.25	10	7.25	7.1875	10	7.75	8.25	10	7.3125	7.6875
11	5.875	6	11	6	5.9375	11	6.8125	6.9375	11	7.25	7.25
12	5.875	5.6875	12	6.3125	5.875	12	6.625	6.25	12	7.875	8.25
13	6	6	13	6.4375	6	13	6.8125	6.375	13	7.6875	7.75
14	7	7.0625	14	6.75	6.4375	14	6.125	6.125	14	7	7.1875
15	7.4375	7.625	15	5.8125	5.4375	15	6.6875	6.625	15	6.4375	6
16	6.375	6.4375	16	6.9375	7.3125	16	8.0625	8	16	7.6875	7.4375
17	6.4375	6.5625	17	7.5	7.5625	17	7.8125	7.375	17	8.0625	7.625
18	5.625	5.4375	18	5.625	5.5	18	7.5625	7.9375	18	6.4375	6.8125
19	6	6.3125	19	7.3125	7.1875	19	6.25	5.75	19	8.1875	7.875
20	6.625	6.6875	20	6.125	5.875	20	7	6.9375	20	7.125	7.5
21	6.75	6.5	21	7.5	7.875	21	8	8.375	21	6.4375	6.0625
22	7.125	6.875	22	5.625	5.5625	22	7.0625	6.9375	22	9	8.75
23	5.625	5.375	23	7.1875	7.0625	23	8.0625	7.8125	23	7.3125	7.125
24	6.875	7	24	6.8125	6.375	24	6.8125	6.375	24	7.1875	7.625
25	5.5625	5.4375	25	6.75	6.8125	25	6.1875	6.6875	25	7.9375	8.4375
26	5.625	5.875	26	6.875	7.25	26	7.625	7.8125	26	9.25	8.75
27	6.9375	7.125	27	6.8125	6.5625	27	6.1875	6.0625	27	8.1875	8.375
28	7	6.9375	28	6.0625	6.4375	28	6.875	6.8125	28	7.6875	7.25
29	6.0625	5.875	29	5.9375	5.9375	29	7.9375	8	29	7.3125	7.125
30	5.875	5.625	30	7	7.25	30	7	6.8125	30	7.9375	7.5625
mean	6.35416667	6.34953704	mean	6.58333333	6.53240741	mean	7.0787037	6.99074074	mean	7.72222222	7.625
clk	10.505 us	9.745 us	clk	9.67 us	7.235 us	clk	7.605 us	6.515 us	clk	6.925 us	5.965 us

Tabla A.10: Problema Yatch resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	12.067	13.034	1	13.857	15.538	1	13.214	14.345	1	14.07	15.811
2	12.083	13.911	2	12.505	13.054	2	13.217	14.556	2	14.632	14.904
3	12.02	12.261	3	12.791	14.523	3	14.794	16.085	3	15.112	15.533
4	12.952	13.548	4	12.942	14.575	4	14.612	14.978	4	15.302	17.269
5	13.255	14.005	5	13.041	13.809	5	14.251	14.798	5	14.676	15.979
6	12.495	12.64	6	13.146	14.209	6	14.388	15.039	6	15.926	17.85
7	12.567	14.408	7	13.65	15.504	7	13.705	15.487	7	15.446	17.266
8	12.306	13.567	8	13.89	15.275	8	14.256	15.796	8	15.686	17.529
9	12.259	13.482	9	12.567	14.165	9	13.165	14.097	9	15.927	15.977
10	13.287	14.947	10	12.91	14.422	10	14.331	15.39	10	15.997	17.658
11	13.072	13.385	11	13.636	15.614	11	15.084	16.514	11	15.278	15.331
12	13.074	13.24	12	12.698	13.934	12	14.348	16.258	12	15.833	16.206
13	12.631	13.163	13	12.546	14.389	13	14.622	16.407	13	15.12	16.72
14	13.485	15.002	14	12.67	13.778	14	13.331	14.248	14	14.409	14.876
15	12.54	14.496	15	13.538	14.879	15	14.784	15.647	15	15.413	15.843
16	12.422	13.511	16	12.611	13.617	16	14.396	15.128	16	15.683	17.498
17	12.651	14.448	17	13.294	13.952	17	13.249	14.172	17	14.695	16.056
18	13.023	14.198	18	12.523	13.222	18	14.359	14.417	18	14.084	16.029
19	13.469	14.448	19	12.576	13.79	19	13.324	13.92	19	15.228	16.531
20	13.03	14.378	20	13.147	13.844	20	14.589	15.947	20	15.89	16.007
21	12.153	13.764	21	12.838	13.263	21	13.909	14.774	21	15.204	15.494
22	13.4	14.831	22	13.596	15.388	22	14.358	14.695	22	14.034	16.03
23	12.069	12.234	23	13.142	14.101	23	14.106	15.51	23	15.713	17.285
24	13.308	14.02	24	13.85	14.623	24	14.139	15.132	24	14.038	15.798
25	12.397	13.539	25	13.09	13.528	25	14.324	14.715	25	14.673	15.766
26	12.22	12.326	26	13.105	14.675	26	14.16	15.109	26	15.943	17.535
27	12.835	13.237	27	12.906	13.147	27	15.071	16.216	27	15.245	15.801
28	12.961	14.11	28	13.08	13.645	28	14.597	16.27	28	14.751	15.863
29	13.006	13.28	29	12.989	14.019	29	13.787	14.872	29	15.79	16.483
30	12.286	12.987	30	13.848	14.414	30	13.356	15.223	30	15.126	15.679
mean	12.8543425	13.6812	31	13.16775862	13.85387	31	14.2117954	15.281069	31	15.1458989	16.1653103

Tabla A.11: Problema Tower resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	164.4375	145.875	1	178.1875	163.125	1	212.5	200.0625	1	232.3125	212.3125
2	166	150.1875	2	198.875	188.75	2	213.5	176.0625	2	211.1875	171.75
3	194.4375	182.5	3	208.3125	197.375	3	219.25	200.3125	3	218.375	199.3125
4	165.875	137.8125	4	188.125	162.875	4	221.6875	204.5625	4	247.3125	214.0625
5	191.4375	174.3125	5	177.75	153.5	5	194.8125	179.375	5	209.875	165.375
6	192.6875	178.375	6	172.5	157.0625	6	195.3125	165.6875	6	234.0625	206.5625
7	195.125	165.9375	7	172.375	143.3125	7	213.75	200.875	7	244.875	207.25
8	189.0625	167.9375	8	170.1875	154.6875	8	202.1875	177	8	205.3125	179
9	150	136.5625	9	176.6875	164.25	9	191.8125	160.3125	9	237	203.3125
10	191.0625	171.8125	10	202.5625	191	10	218	184.375	10	200.4375	178.6875
11	160.3125	132.8125	11	188.5	166	11	200.5	170.875	11	218.6875	172.5625
12	184.5625	162.5625	12	218.3125	202.8125	12	198.5625	184.875	12	223.9375	202.8125
13	168.9375	140.5625	13	191.75	172.5625	13	203.75	190.1875	13	202.6875	159.375
14	163.25	139.875	14	196.625	181.1875	14	217.5625	199.6875	14	249.875	201.75
15	186.8125	160.375	15	194.4375	172.0625	15	216.5	185.5625	15	232.0625	197
16	198.25	178.875	16	207.125	180.5	16	212.625	194	16	233.875	205.0625
17	184.375	161.9375	17	199.4375	172.5625	17	190.25	157.75	17	201.0625	188.5625
18	169.25	139.875	18	184.9375	161.6875	18	214.4375	204.3125	18	215.8125	168.875
19	192.3125	177.75	19	189.1875	161.625	19	203.6875	164.375	19	217.25	203.4375
20	151.0625	134.5625	20	208.25	189.0625	20	211.625	199.875	20	218.8125	175.375
21	169.5625	159.3125	21	176.8125	147.8125	21	220.25	184.375	21	244.375	226.5
22	180.25	169.75	22	171.25	147.9375	22	224.0625	207.75	22	246.375	203.6875
23	151.0625	127.125	23	178.375	159.8125	23	207.1875	177.1875	23	221.625	201.0625
24	191.8125	172.4375	24	193.0625	170.625	24	217.4375	187	24	232.4375	194.3125
25	172.625	154.5	25	172.1875	154.3125	25	203.8125	176	25	208.8125	192.5
26	154.5	139.6875	26	213.3125	193.1875	26	191.3125	152.75	26	223.9375	186.3125
27	185.6875	157.125	27	196.75	186.5	27	224.4375	208.75	27	227.75	203.0625
28	180.3125	159.25	28	219.375	206	28	213.1875	176.25	28	210	162.3125
29	159.625	149.4375	29	211.6875	187.375	29	211.1875	172	29	206.75	174.375
30	187.75	159.9375	30	182.625	164.5625	30	224	204.6875	30	204.6875	173.8125
mean	176.576389	155.944444	mean	190.8958333	170.55093	mean	209.034722	184.090278	mean	222.951389	190.62963
clk	11.735 us	10.305 us	clk	10.505 us	8.225 us	clk	8.605 us	7.515 us	clk	7.675 us	6.465 us

Tabla A.12: Problema Tower resuelto con GSGP-Software, para datos de entrenamiento y prueba con 2 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	216.465	110.707	1	244.754	104.848	1	221.538	110.36	1	293.242	128.291
2	190.15	61.761	2	240.614	86.591	2	255.893	124.126	2	266.194	156.448
3	215.874	79.624	3	220.83	65.063	3	258.58	148.152	3	241.052	174.048
4	197.984	85.637	4	216.529	95.951	4	259.475	142.092	4	271.506	155.899
5	228.326	60.014	5	216.273	81.552	5	249.883	106.189	5	282.978	152.566
6	219.967	91.474	6	215.231	133.006	6	241.543	140.919	6	285.716	177.479
7	225.166	116.691	7	218.654	85.532	7	224.784	149.761	7	295.324	163.632
8	225.895	65.121	8	241.524	110.097	8	252.094	142.537	8	282.683	148.19
9	208.567	87.243	9	249.326	120.192	9	235.596	144.893	9	296.795	173.5
10	228.97	95.116	10	208.502	82.136	10	227.49	103.764	10	247.988	171.774
11	215.438	64.546	11	240.715	92.502	11	221.109	122.908	11	270.019	146.591
12	201.892	88.116	12	202.865	103.075	12	234.926	117.581	12	265.49	171.853
13	225.669	100.956	13	230.922	81.157	13	254.393	139.795	13	276.122	164.661
14	212.88	99.05	14	224.389	121.002	14	254.814	105.454	14	271.01	166.857
15	218.326	106.426	15	214.299	72.901	15	229.352	121.835	15	255.647	166.031
16	215.823	63.231	16	204.206	102.799	16	250.997	108.516	16	275.984	164.17
17	225.574	88.992	17	235.072	139.137	17	248.762	145.166	17	263.382	128.544
18	212.15	128.241	18	219.077	97.379	18	244.412	111.956	18	292.607	178.751
19	221.133	120.662	19	244.903	139.498	19	220.058	132.454	19	299.911	140.169
20	202.636	104.658	20	230.524	129.125	20	223.657	118.247	20	266.091	152.009
21	204.368	122.697	21	210.983	79.201	21	238.78	121.603	21	273.834	156.392
22	205.431	106.931	22	208.792	127.58	22	248.722	131.535	22	275.353	167.493
23	207.192	63.599	23	216.968	114.893	23	245.606	123.105	23	253.766	136.04
24	222.428	79.858	24	223.749	129.405	24	240.821	98.001	24	287.741	151.18
25	224.599	102.427	25	239.112	137.292	25	220.043	134.87	25	288.569	177.366
26	200.272	113.792	26	236.073	133.249	26	220.044	108.488	26	247.988	122.738
27	197.152	103.739	27	249.722	68.164	27	252.418	132.187	27	262.347	124.011
28	210.169	82.291	28	245.707	115.344	28	254.325	130.043	28	258.103	125.097
29	224.843	102.477	29	227.887	94.141	29	228.836	147.259	29	263.861	130.57
30	214.579	102.513	30	212.109	118.253	30	244.834	126.893	30	270.062	163.418
mean	214.719593	94.3147407	mean	225.3375185	107.57641	mean	239.547185	126.224111	mean	273.365815	154.703

A.2. Pruebas con 12 variables

Las tablas y gráficas mostradas en esta sección se realizaron con los parámetros descritos en la Tabla 5.2, para esta prueba se probó el algoritmo con 3, 6, 8, 10 generaciones y 12 variables.

Tabla A.13: Problema Tower resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 12 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	242.8125	261.5	1	249.25	273.625	1	308.125	320.75	1	300.6875	313.125
2	213.5	227.9375	2	205.375	215.1875	2	262.125	271.4375	2	258.25	264.3125
3	202.625	213.9375	3	262.8125	284.3125	3	272.875	275.9375	3	306.8125	325.5625
4	269.0625	272.1875	4	293.3125	296.5	4	243.5	246.125	4	270.6875	270.9375
5	229.6875	244.5	5	243.5	251.875	5	240.5	256.6875	5	263.0625	272.9375
6	184.125	188.875	6	202.75	217.9375	6	276.875	291.4375	6	317.625	339
7	256.9375	261.8125	7	235.875	251.125	7	287.5625	293.125	7	248.4375	263.5
8	220.4375	229.75	8	205	223.6875	8	249	271.8125	8	319.25	327.5625
9	205.5	212.5625	9	266.6875	286.75	9	217.625	240	9	282.4375	291.8125
10	248.625	250.1875	10	299.625	318.9375	10	252.625	267.5	10	252.5	253.9375
11	192.5	201.6875	11	259.875	270.6875	11	281.0625	300.625	11	315.625	316.5625
12	254.3125	257.375	12	205.75	210.5	12	235.5625	241.75	12	257.375	273.6875
13	183.875	192.1875	13	288.1875	294	13	275.6875	286.5625	13	282.1875	289.0625
14	228.625	239.3125	14	235.3125	254.5	14	223.5625	247.9375	14	282.25	284.5
15	218.9375	228.625	15	298.9375	314.5625	15	300.6875	304.75	15	258.9375	279.1875
16	207	214.875	16	247.5	251.9375	16	265.875	284.375	16	238.8125	249.75
17	253.3125	267.625	17	231.9375	244.75	17	294.875	302.8125	17	293.9375	302.375
18	214.3125	220	18	272.8125	295.625	18	238.75	260.8125	18	247.1875	261.4375
19	195.5	206.8125	19	216.125	228.6875	19	286.6875	288.75	19	238.25	258
20	217.6875	230.625	20	231.25	254.9375	20	212.9375	215	20	328.6875	334.0625
21	200.1875	212.3125	21	262.3125	275.75	21	224.8125	234.0625	21	261.5625	273.1875
22	188.5	202.5	22	219	243.6875	22	277.625	281.25	22	302.75	323.625
23	185.5	192.75	23	259.875	284.6875	23	243.6875	258.25	23	322.0625	346.5
24	246.5625	259.875	24	278.4375	280.5	24	285.5625	292.75	24	292.6875	317
25	183.4375	184.875	25	291.6875	292	25	215.0625	216	25	244.5625	258.75
26	218.6875	218.8125	26	238	255.4375	26	234	251.1875	26	296.1875	300.8125
27	259.9375	278.6875	27	222.9375	226.6875	27	265.5625	274.6875	27	267.1875	270.5
28	202.5625	210.1875	28	267.4375	276.3125	28	268.5	292.5625	28	260.3125	276.4375
29	200.9375	215.6875	29	297.8125	308.5625	29	295.75	316.375	29	312.375	333.6875
30	197.875	211.625	30	241.875	242.75	30	220.0625	233.6875	30	324.9375	333.625
mean	217.208333	226.159722	mean	252.3634259	264.9398 1	mean	256.074074	268.550926	mean	280.810185	292.68287

Tabla A.14: Problema Tower resuelto con GSGP-Software, para datos de entrenamiento y prueba con 12 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	199.153	200.67	1	299.836	328.422	1	309.183	333.861	1	327.658	362.593
2	222.576	233.701	2	289.137	295.224	2	299.424	300.997	2	266.893	278.549
3	220.595	229.531	3	279.098	292.85	3	278.697	279.705	3	290.572	314.292
4	199.446	222.799	4	269.741	289.637	4	291.613	324.45	4	282.552	303.339
5	185.347	198.672	5	247.144	252.918	5	254.407	280.744	5	316.195	354.739
6	239.037	241.024	6	293.519	317.287	6	296.01	299.252	6	321.824	341.114
7	247.805	268.536	7	207.831	219.016	7	253.275	277.809	7	324.987	328.129
8	255.807	265.44	8	219.229	231.391	8	296.674	324.834	8	266.785	295.78
9	269.501	290.541	9	266.035	274.26	9	301.952	322.483	9	272.908	278.942
10	225.322	227.894	10	267.53	280.038	10	265.798	293.359	10	338.611	370.529
11	194.712	214.925	11	292.088	311.628	11	304.08	315.177	11	357.987	369.482
12	220.413	229.102	12	200.898	209.757	12	257.63	289.524	12	284.528	314.906
13	221.871	243.271	13	285.493	298.396	13	273.803	293.987	13	269.96	283.582
14	191.25	197.398	14	212.261	231.738	14	272.278	286.607	14	288.385	319.253
15	176.489	188.788	15	236.649	239.753	15	312.599	346.044	15	327.467	365.029
16	260.248	284.104	16	273.081	278.455	16	272.448	291.786	16	354.277	375.954
17	257.467	286.883	17	271.461	288.369	17	287.253	291.073	17	344.702	348.309
18	256.421	274.865	18	278.538	301.393	18	252.078	258.218	18	340.61	365.532
19	212.116	228.54	19	210.572	239.554	19	324.936	364.317	19	320.563	350.544
20	176.875	189.749	20	299.779	308.807	20	250.432	262.541	20	292.95	309.003
21	219.404	245.26	21	288.243	305.284	21	257.282	280.445	21	292.458	328.137
22	228.162	252.039	22	272.74	299.745	22	315.48	320.08	22	296.242	322.619
23	197.038	201.263	23	271.236	282.081	23	240.978	266.379	23	306.497	311.608
24	173.76	176.036	24	207.384	229.553	24	258.072	263.008	24	346.023	385.442
25	254.347	257.654	25	274.991	288.758	25	329.694	351.645	25	281.392	308.545
26	232.388	239.401	26	298.244	311.892	26	248.691	279.252	26	319.612	347.604
27	204.392	221.653	27	288.843	307.812	27	321.485	324.698	27	303.213	338.276
28	176.009	197.837	28	277.771	305.705	28	298.142	337.335	28	301.633	302.392
29	214.134	238.467	29	253.094	274.857	29	295.773	306.835	29	294.959	326.079
30	248.878	265.495	30	203.027	211.144	30	326.071	364.612	30	343.155	344.968
mean	219.949593	235.09763	mean	258.0526667	273.67511	mean	283.664222	304.314593	mean	310.758333	332.956889

Tabla A.15: Problema Housing resuelto con GSGP-VHDL, para datos de entrenamiento y prueba con 12 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	7.9375	8.25	1	8.4375	7.875	1	15.1875	16.9375	1	20.0625	19.6875
2	5	5.3125	2	11.9375	13.0625	2	12	12.4375	2	14.375	15.9375
3	4.3125	4.5	3	12.125	11.375	3	14.1875	14.4375	3	22.5	23.25
4	4.125	4.25	4	8.0625	8.25	4	13.5	13.875	4	16.5625	18.375
5	5.5625	4.9375	5	14.8125	14.9375	5	12.625	13.25	5	19.8125	20.4375
6	7.3125	6.625	6	13	12.0625	6	19.75	19.625	6	18.1875	19.9375
7	4.8125	4.6875	7	10.6875	11.9375	7	19.5625	20.1875	7	15	16
8	8.5625	7.875	8	9.125	10.375	8	14.5	16.375	8	19.9375	19.75
9	4.4375	4.3125	9	13.625	12.6875	9	13.125	14.375	9	14.9375	16.4375
10	5.875	6.0625	10	12.8125	11.9375	10	17.625	18.375	10	20.0625	19.5625
11	4	4.1875	11	12.125	12.6875	11	17.3125	17.625	11	22.9375	23
12	3.0625	2.25	12	10.125	9.625	12	12.3125	13.6875	12	15.25	16.0625
13	6.5	6.5	13	9.4375	10.375	13	17.6875	17.4375	13	21.6875	22.4375
14	7.8125	8.125	14	10.0625	9.875	14	15.25	14.8125	14	22.6875	22.8125
15	5.25	5.1875	15	14.125	15	15	16.6875	17.75	15	17.9375	19
16	8.4375	7.8125	16	9	9.8125	16	14.8125	14.0625	16	22.8125	22.1875
17	9.125	9.3125	17	10.625	10.125	17	11.625	11.375	17	18.625	19.625
18	5.4375	5.0625	18	8.125	7.625	18	19.625	19.75	18	22	21.3125
19	8.3125	8.4375	19	14.8125	15.0625	19	13	14.4375	19	17.125	18.5625
20	9.125	8.75	20	14.6875	14.8125	20	13	14.625	20	13.75	14.5
21	7.625	8	21	11.375	10.875	21	13.1875	12.3125	21	13.0625	12.875
22	9.625	9.375	22	13	12.0625	22	11	10.6875	22	22.875	23.1875
23	5.125	4.75	23	14.5	15.4375	23	18.625	18.3125	23	16.9375	18.25
24	6.8125	6.75	24	9.4375	10.9375	24	18.3125	19.0625	24	14.125	16
25	9.5625	10	25	10.8125	10.5625	25	19.9375	21.9375	25	19.8125	21.8125
26	4.75	4.5	26	8.25	8.5	26	16.6875	18.3125	26	17.3125	17.125
27	6	6	27	14.625	14.5625	27	13.0625	14.4375	27	21.875	21.3125
28	4.125	3.625	28	10.6875	11.125	28	18.4375	18.9375	28	15.75	17.625
29	7	7.375	29	10.0625	10.75	29	14.5	16.4375	29	15.4375	17.1875
30	7.3125	7.125	30	13.8125	12.9375	30	19.5625	19.1875	30	20.375	20.5
mean	6.50694444	6.36574074	mean	11.5486111	11.6643519	mean	15.7523148	16.3425926	mean	18.4027778	19.1064815
clk	11995 ns	10145 ns	clk	9995 ns	8515 ns	clk	8125 ns	7745 ns	clk	7295 ns	7095 ns

Tabla A.16: Problema Housing resuelto con GSGP-Software, para datos de entrenamiento y prueba con 12 variables, 30 corridas 16 individuos, 10, 8, 6 y 3 generaciones. Se muestra la media y la cantidad de ciclos de reloj.

Corrida	16 pob 10 gen		Corrida	16 pob 8 gen		Corrida	16 pob 6 gen		Corrida	16 pob 3 gen	
	Train	Test		Train	Test		Train	Test		Train	Test
1	8.987	9.527	1	14.315	15.244	1	19.318	18.484	1	17.518	16.793
2	8.607	8.956	2	14.626	14.03	2	17.36	15.684	2	19.713	17.628
3	9.61	6.851	3	10.534	10.119	3	15.801	14.616	3	23.369	20.776
4	7.206	5.227	4	13.498	13.481	4	18.297	17.374	4	13.45	12.46
5	6.326	6.191	5	11.676	11.475	5	16.427	17.997	5	13.338	13.178
6	6.036	5.768	6	14.06	12.632	6	17.847	14.966	6	21.008	17.721
7	8.7	10.264	7	14.851	14.415	7	15.835	16.228	7	12.665	9.271
8	5.51	4.445	8	14.456	12.986	8	18.402	16.72	8	18.537	16.247
9	5.706	3.463	9	12.406	12.338	9	15.62	15.579	9	23.899	20.687
10	9.076	8.764	10	12.638	13.495	10	16.517	16.479	10	27.799	26.881
11	8.952	10.867	11	10.324	8.675	11	18.285	20.16	11	21.535	18.187
12	8.963	8.108	12	12.029	12.161	12	16.87	15.194	12	27.162	23.555
13	6.781	4.99	13	13.477	14.178	13	18.382	18.772	13	13.564	14.986
14	6.33	6.796	14	13.523	11.983	14	17.067	18.432	14	20.898	21.581
15	9.526	11.18	15	13.111	13.208	15	18.693	20.497	15	24.224	25.256
16	6.728	5.795	16	12.116	11.576	16	19.829	16.889	16	11.705	12.223
17	5.061	2.676	17	14.089	13.445	17	16.257	16.862	17	21.282	18.085
18	9.089	10.785	18	11.597	11.349	18	15.704	14.749	18	19.6	16.541
19	7.596	5.217	19	10.411	8.462	19	19.374	19.24	19	22.75	20.588
20	7.066	6.279	20	13.571	14.075	20	15.788	16.891	20	26.115	27.904
21	7.812	5.773	21	11.543	10.741	21	17.537	15.146	21	22.521	20.334
22	7.083	6.83	22	14.217	12.532	22	15.171	13.22	22	16.946	15.994
23	9.197	9.174	23	10.558	10.471	23	16.465	14.138	23	28.241	24.96
24	7.126	8.186	24	13.922	14.534	24	19.988	17.368	24	14.213	10.49
25	5.794	2.947	25	11.34	9.729	25	18.09	18.421	25	24.495	22.564
26	6.405	3.853	26	12.126	10.161	26	15.981	17.134	26	25.492	25.328
27	6.646	8.557	27	11.345	9.516	27	17.252	17.31	27	20.64	19.327
28	9.973	11.431	28	10.873	9.74	28	19.681	21.076	28	28.21	24.996
29	7.093	4.255	29	11.539	11.291	29	19.73	18.677	29	20.38	19.387
30	7.569	5.89	30	10.629	11.396	30	15.158	14.178	30	10.187	7.853
mean	#########	6.80411111	mean	12.4416667	11.8535185	mean	17.4165556	17.0258148	mean	20.4020741	18.7623704