ISSN 1870-4069

Implementación en hardware de un algoritmo genético para resolver un problema combinatorio

Eliseo Díaz-Nacar, Katya Rodríguez-Vázquez

Universidad Nacional Autónoma de México,
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas,
México

{eliseo.diaz, katya.rodriguez}@iimas.unam.mx

Resumen. En el presente trabajo, se realiza la implementación en hardware mediante FPGA´s, de un Algoritmo Genético Simple (GA, por sus siglas en inglés *genetic algorithm*), para resolver problemas de tipo combinatorio, que en este caso, ilustra la solución del problema de ubicar 8 reinas en un tablero de ajedrez sin que haya ataque entre ellas. Así mismo se realiza una comparación de desempeño de la implementación en hardware, con respecto a la implementación del mismo problema en software mediante MATLAB. Los procesos que integran el GA se implementan mediante módulos de hardware, así mismo también se implementaron los módulos que se requieren para operar el GA, tales como Generación Aleatoria de la Población Inicial, Módulo de Despliegue de Resultados, Módulos de Memoria, Buffers para los Módulos de Memoria. Los resul,tados se presentan de manera gráfica a través de un monitor VGA. La implementación de todo el algoritmo se realiza empleando lenguaje VHDL y una tarjeta de desarrollo SPARTAN3 de DIGILENT, la cual contiene un FPGA SPARTAN 3 de XILINIX. También se presentan mediante tablas, los resultados de las pruebas de desempeño realizadas a la implementación en hardware y la implementación en software.

Palabras clave: FPGA, algoritmos genéticos, problemas combinatorios.

A Hardware Implementation of a Genetic Algorithm for Solving Combinatorial Problems

Abstract. In this work, the hardware implementation is carried out by means of FPGA's, of a Simple Genetic Algorithm (GA), to solve combinatorial problems, which, in this case, illustrates the solution of the problem of placing 8 queens on a chess board without any attack between them. Likewise, a comparison of performance of the implementation in hardware is made, with respect to the implementation of the same problem in software through MATLAB. The processes that integrate the GA are implemented through hardware modules. Likewise, the modules that are required to operate the algorithm were also implemented, such as random generation of the initial population, results display module, memory modules, buffers for memory modules. The results are presented graphically through a VGA monitor. The implementation of the entire

algorithm is carried out using VHDL language and a SPARTAN3 development card from DIGILENT, which contains a SPARTAN 3 FPGA from XILINIX. The results of the performance tests of the hardware implementation and the software implementation are also presented in tables.

Keywords: FPGA, genetic algorithm, combinatorial problems.

1. Introducción

Un GA es un proceso de optimización muy poderoso, inspirado en la evolución natural tal como lo establece Goldberg [1]. Los GA fueron creados por John Holland en la década de los 60's, basados en los procesos naturales de evolución y selección natural tales como selección, cruza y mutación. Estos algoritmos encuentran su principal aplicación en problemas de búsqueda y optimización de problemas paramétricos y combinatorios [2].

Los GA simulan el proceso de evolución natural. Tal simulación requiere de una codificación del dominio del problema por lo que se requiere determinar el espacio en el que se encuentran las posibles soluciones al problema que se pretende resolver u optimizar. Para obtener la solución del espacio de búsqueda se requiere de un proceso de evaluación, el cual se aplicará a una población inicial de soluciones. El GA inicia con la generación de la población inicial y el proceso de evaluación, el cual, se realiza a través de una función objetivo que modela la solución del problema. Después del proceso de evaluación, se obtiene el resultado de dicha evaluación para cada miembro de la población inicial.

Con la población inicial ya evaluada, se aplican los siguientes procesos de forma iterativa: *Selección*, de este proceso se obtienen los mejores individuos (soluciones) de la población, de acuerdo a los resultados de la evaluación, el proceso de selección puede emplear varios métodos como ruleta, torneo etc. En este trabajo se empleó torneo, el cual consiste en asignar parejas de forma aleatoria entre los miembros de la población, de cada pareja se selecciona el individuo mejor evaluado. A esta nueva población (padres), se aplica el proceso de *Cruza* por parejas obteniendo una nueva población (hijos). A esta población de hijos se aplica el proceso de *Mutación* obteniendo una población mutada (nueva generación), con características que se espera, se adapten mejor al problema [3]. Este proceso genera en cada iteración una nueva generación de nuevos individuos con características, que se espera, sean cada vez mejores, para la solución del problema, aunque no siempre sucede y puede ocurrir que el algoritmo ya no encuentre una mejor solución, en cuyo caso es conveniente iniciar el algoritmo nuevamente. Este proceso iterativo presume un consumo de tiempo considerable de procesamiento en un equipo de cómputo; dicho tiempo está en función, en gran medida, de la complejidad del problema que se pretende resolver y del software empleado.

La forma de implementación de un GA puede ser mediante un lenguaje de alto nivel o software especializado como MATLAB. Para dar solución al problema del tiempo de proceso, se han realizado implementaciones en hardware a través de diferentes plataformas tal como se describe en la sección 2 del presente trabajo. Para la implementación en hardware se emplea Lógica Programable la cual generalmente se realiza mediante FPGA's.

Un FPGA (FIELD PROGRAM GATE ARRAY) es un dispositivo que contiene miles o incluso millones de transistores cuya interconexión se puede configurar para implementar funciones lógicas, sumas, restas, hasta complejos filtros digitales y sistemas de detección y corrección de errores. Aeronaves, automóviles, radares, misiles, son algunos de los sistemas que emplean FPGA´s [4]. Los principales fabricantes de FPGA's proveen plataformas de desarrollo y herramientas de software para efectuar la implementación física dentro del FPGA. Uno de los lenguajes empleados para el desarrollo de sistemas en FPGA'S es VHDL que es un lenguaje de descripción de hardware.

En la sección 2 del presente trabajo se hace un resumen de los trabajos previos relacionados. En la sección 3 se presenta el diagrama esquemático de la implementación del GA, se detallan las señales de entrada y salida de cada módulo que lo integran, así mismo se presentan los módulos necesarios para la operación del algoritmo tales como el módulo de control y el de despliegue de resultados. En la sección 4 se hace una descripción del funcionamiento del sistema. En la sección 5 se realizan las pruebas correspondientes obteniendo el tiempo requerido para llegar a la solución del problema, así mismo se hace una comparación de desempeño de la implementación del mismo problema empleando MATLAB. En la sección 6 se presenta el trabajo futuro para mejorar el desempeño de la implementación en hardware y poderlo aplicar a problemas más complejos. Finalmente, se presentan las conclusiones del desarrollo del presente trabajo.

2. Trabajos relacionados

Los trabajos que se mencionan a continuación son solo una muestra de la cantidad de trabajos que se han realizado sobre la implementación de un GA en Hardware. Scott *et al.* [5] presentan en 1995 uno de los primeros trabajos relacionados con la implementación en hardware de un GA. Presentan resultados del desempeño de la implementación versus una versión en software. Proporcionan la arquitectura empleada mediante diagramas de bloques, cabe aclarar que emplean pipeline y paralelismo para mejorar el desempeño de la implementación en hardware.

En 2000 Tufte y Haddow [6] presentan la implementación de un filtro digital dinámico cuyos coeficientes de operación se obtienen mediante un GA implementado en un FPGA, en la implementación del GA emplean un arreglo en PipeLine. En el trabajo reportan diagramas de bloques y los resultados los presentan mediante gráficas de respuesta del filtro de amplitud frecuencia y amplitud tiempo aplicando diversas señales de entrada.

En 2001, Aporntewan y Chongsttvatana desarrollaron la implementación de un GA compacto [7]. En este trabajo realizaron la implementación mediante FPGA empleando VHDL, presentando diagramas de bloques y pseudocódigos de la implementación, así mismo presentaron una versión de software realizada en lenguaje "C" con la finalidad de efectuar una comparación de desempeño reportando un *speedup* de 1000.

Barry Shackleford *et al* [8] en 2001 presentan una arquitectura en pipe-line de un GA en FPGA's con la finalidad de acelerar la operación del GA. Presentan dos prototipos de implementación usando diferentes plataformas y frecuencias de reloj.

Para probar la implementación resolvieron algunos problemas como el problema del plegamiento de proteínas. Los resultados reportados presentan una mayor rapidez empleando pipe-line logrando evaluar un cromosoma por ciclo de reloj.

En 2004, Tang y Yip [9] presentan la implementación de un GA usando 2 FPGA's, describen los módulos que integran el algoritmo, proporcionan simulaciones y diagramas de bloques de la implementación. Plantean la implementación en paralelo y pipeline del algoritmo. Para las pruebas emplean una función simple de optimización modificada de la ecuación de línea recta de Witte y Holst.

Mostafa *et al.* en 2004 [10] realizan la implementación de un GA en hardware empleando FPGA's y VHDL, dicha implementación posee una arquitectura paralela y pipeline. Los módulos del algoritmo los diseñaron de forma independiente e implementaron un protocolo de *handshake* para su comunicación. Las pruebas del algoritmo las realizan aplicándolo a 3 problemas diferentes: interpolación lineal, procesamiento de datos de un termistor y la aceleración de un vehículo. Los resultados de las pruebas los presentan mediante gráficas.

Nonel Thirer [11] en 2012 presenta el artículo *About the FPGA Implementation of a Genetic Algorithm for solving Sudoku Puzzles*, en el cual presenta la posibilidad de la implementación para resolver un problema combinatorio como es el Sudoku y de igual forma presenta algunas adaptaciones de los módulos del GA para su implementación en un FPGA.

Tuncer y Yildirim [12] presentaron en 2016 el artículo *Design and Implementation of a Genetic Algorithm IP core on an FPGA for path planning of mobile robots*, en el cual describen la implementación de cada módulo del GA y su implementación física en un robot que ejecuta un recorrido a través de un ambiente de prueba.

En 2016, Alansi *et al.* [13] publicaron un artículo donde plantean el desarrollo de una herramienta adaptativa basada en un GA para sistemas SDMA-OFDM (GASOS, nombré asignado a la herramienta) para mejorar el rendimiento y la complejidad computacional en casos de escenarios multiusuario totalmente cargados y sobrecargados. Presentan los detalles de la implementación con todos los módulos requeridos mediante diagramas de bloques.

Guo *et al.* en 2016 [14] proponen en este artículo un sistema de algoritmos genéticos paralelos en hardware, como Field-Programmable-Gate-Arrays (FPGA's). El enfoque apunta a múltiples FPGA mediante la exploración de diferentes áreas de búsqueda del mismo espacio de solución con diferentes comportamientos. Cada FPGA contiene un GA optimizado y personalizable que puede configurarse utilizando parámetros de tiempo de ejecución, eliminando la necesidad de una compilación costosa. Los experimentos en tres problemas, muestran el alto rendimiento de la propuesta, con una aceleración 30 veces mayor en comparación con una implementación basada en CPU multinúcleo.

En 2017, Thorbole *et al.,* presentaron una implementación en hardware de un GA para la detección y predicción de ataques de epilepsia, lo cual se realiza mediante el uso de métodos analíticos aplicados a las características que se extraen de las señales electroencefalográficas (EEG) [15]. En el trabajo, la detección y predicción de ataques se realiza mediante dos pasos: la extracción de características y la clasificación de características. La extracción de características se realiza mediante el uso de filtrado promedio de pico a pico móvil.

La clasificación de las características de las convulsiones se realiza combinando un enfoque de redundancia mínima de máxima relevancia (mRMR) para la selección de características de las convulsiones y el GA para la clasificación de las convulsiones. El preprocesamiento se realiza en MATLAB y la implementación en hardware del GA se realiza en FPGA. En los resultados se informa sobre el conjunto de datos EEG estándar y la simulación de hardware del GA en FPGA.

Praveena *et al*. en 2017 desarrollan un nuevo algoritmo de segmentación de imágenes médicas tales como imágenes de retina e imágenes cardiacas, con diferentes umbrales el cual se basa en un GA implementado en un FPGA [16]. La arquitectura FPGA propuesta para la segmentación de imágenes es un algoritmo eficiente tanto en tiempo como en energía. Presentan diagramas de bloques de la implementación así como tablas donde presentan el área extraída y el tiempo requerido.

Peker en 2018 propone un IP core de (GA) de propósito general, totalmente personalizable implementado en un FPGA usando una arquitectura de pipeline y arquitecturas paralelas para acelerar el proceso del GA [17]. Los operadores del GA y las funciones de desempeño están diseñados en una estructura modular para permitir el uso de estos módulos de forma asincrónica. El código VHDL de la implementación está escrito con parámetros genéricos para permitir la personalización de casi todos los parámetros del IP Core FPGA propuesto según el problema. El sistema propuesto se utilizó para resolver el problema del agente viajero (TSP). En los experimentos, el autor concluye que el núcleo propuesto mejoró tanto los ciclos de reloj necesarios para iterar una generación como la velocidad de convergencia de las implementaciones de GA existentes.

Torquato y Fernandes proponen una implementación totalmente paralela de un GA implementado en FPGA's [18]. La optimización del tiempo de procesamiento del sistema es el objetivo principal de este trabajo. Se analizan los resultados asociados con el tiempo de procesamiento y la ocupación del área (en FPGA) para varios tamaños de población. Las pruebas se realizaron en la optimización de funciones de dos variables. Los resultados mostraron que la implementación paralela completa de GA logró un rendimiento de aproximadamente 16 millones de generaciones por segundo. También en el trabajo presentan los diagramas de bloques de todos los módulos que integran la arquitectura.

Sunitha y Pradeep presentan la implementación de los módulos que integran el GA basados en una estructura flexible que dinámicamente puede realizar tres tipos de codificación binaria, valor real y enteros [19]. Las estructuras las diseñan y sintetizan usando lenguaje VHDL, simulan con ModelSim y posteriormente lo implementan en FPGA's. Presentan los diagramas de la implementación de cada módulo, así como la simulación de estos mediante la herramienta de implementación en los FPGA´s.

Los trabajos anteriores son solo una muestra de la gran cantidad de aplicaciones en las que se puede usar un GA implementado en hardware, en los cuales, por lo general se emplean FPGA's. Los resultados obtenidos en la mayoría de los trabajos descritos poseen una mayor rapidez en la ejecución del algoritmo, comparando sus respectivas implementaciones en software. La presentación de los resultados es a través de simulaciones, tablas y gráficas, resolviendo generalmente problemas de tipo paramétrico y algunos combinatorios.

3. Implementación en hardware

La implementación del GA se realizó empleando una tarjeta de desarrollo STARTER KIT SPARTAN 3 de DIGILENT [20], la cual contiene un FPGA Spartan 3 de XILINIX así como diversos bloques de memoria, puertos de comunicación paralela y serial, un display de LCD de 2 líneas y 16 caracteres, interruptores diversos y una barra de 8 leds. El software empleado para configuración y comunicación con la tarjeta es proporcionado por XILINIX.

El lenguaje empleado para programar y configurar las funciones requeridas por el algoritmo en el FPGA es VHDL. VHDL es un lenguaje para describir sistemas electrónicos digitales. Se desarrolló bajo los auspicios del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y se adoptó en forma de Manual de Referencia de Lenguaje VHDL Estándar, IEEE estándar 1076 1987. Como todos los estándares IEEE, están sujetos a revisión cada cinco años. Los comentarios y sugerencias de los usuarios del estándar de 1987 fueron analizados por el grupo de trabajo IEEE responsable de VHDL y en 1992 se propuso una versión revisada del estándar y esta fue finalmente adoptada en 1993 [21].

Los módulos que integran la implementación se enlistan a continuación:

- módulo de control,
- generador de población inicial,
- módulo de lectura de memoria de población,
- módulo de memoria de población,
- módulo de memoria de evaluación,
- módulo de memoria de selección,
- buffer de memoria de población,
- buffer de memoria de evaluación,
- buffer de memoria de selección,
- módulo de evaluación,
- módulo de selección,
- módulo de cruza,
- módulo de mutación,
- módulo de control de monitor VGA.

Se muestra el diagrama esquemático de la implementación (Fig. 1) del GA, en el cual se indica la interconexión de todos los módulos que la integran. Los módulos de memoria a los que se hace referencia fueron implementados con los bloques de memoria RAM interna que posee el FPGA, no se usaron los módulos de memoria externos con los que cuenta la tarjeta, únicamente los recursos propios del FPGA. En el esquema de la Fig. 1 se muestran también líneas de entrada y salida a la implementación las cuales se describen a continuación.

- ***CKop*** Señal de reloj de operación que proviene de un oscilador externo a la tarjeta que es la base de operación de la implementación.
- ***CK*** Señal de reloj de 50MHz que provee la tarjeta de desarrollo. Se emplea para generar números aleatorios requeridos en el funcionamiento del algoritmo.

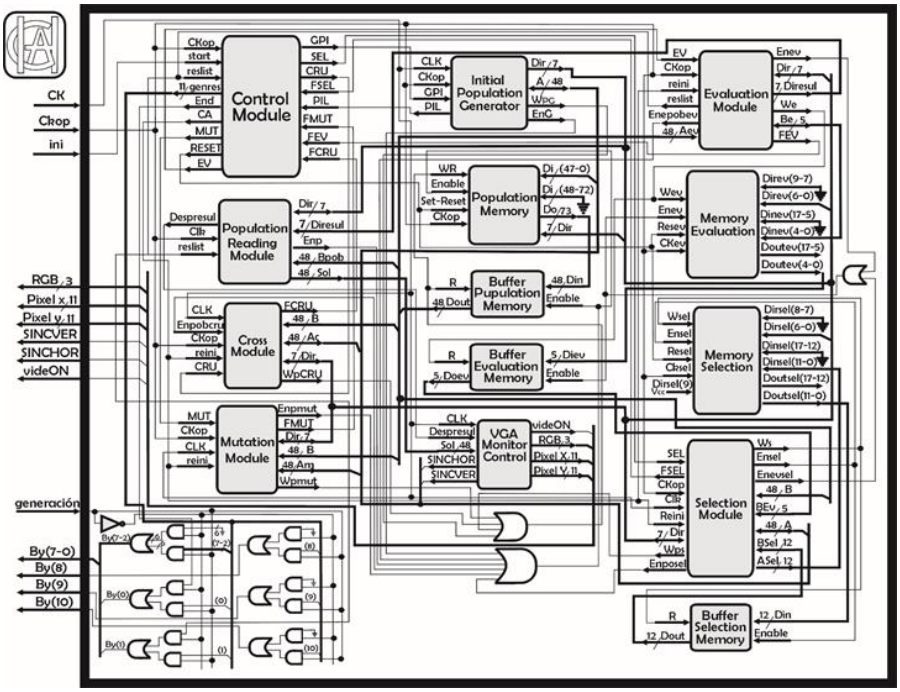

Fig. 1. Diagrama esquemático de la implementación en FPGA.

— ***ini*** Señal externa proveniente de un interruptor que provee la misma tarjeta para dar inicio de forma manual al inicio de la operación del algoritmo.

— ***generación*** Línea de entrada proveniente de un de los interruptores de la tarjeta de desarrollo, que le indica a la implementación la información a desplegar en la barra de 8 leds de la tarjeta de desarrollo más 3 leds adicionales externos para un total de 11 bits. Cuando esta línea se encuentra en cero, los leds correspondientes a los bits "0" y "1" indicarán *"Fin de Algoritmo"* y *"Resultado Listo"* respectivamente, y cuando la línea se encuentra en uno, en los 11 leds se desplegará en binario el número de generación en la cual se encontró la solución.

— ***By(0-10)*** Líneas de salida a la barra de 11 Leds (8 de la tarjeta y 3 externos).

— ***RGB, PIXEL X, PIXEL Y, SINCVER, SINCHOR y VIDEON*** son señales de salida para el control del monitor VGA, cuya conexión se encuentra habilitada en la misma tarjeta de desarrollo.

4. Operación del algoritmo genético en hardware

La operación del algoritmo inicia cuando la señal de entrada *"ini"* adquiere un nivel alto de voltaje, la cual activa la operación del módulo de control. A continuación, se describe la operación de todos y cada uno de los módulos.

Tabla 1. Descripción del Módulo de Control.

MODULO DE CONTROL			
Paso	**Señales de Entrada Activas**	**Señales de Salida Activas**	**Acción**
1	Ini	GPI	Activa la operación del Generador de Población inicial
2	PIL	EV	Termina la generación de la población inicial e inicia el proceso de evaluación
3	FEV	SEL	Después del proceso de evaluación inicia la selección
4	FSEL	CRU	Una vez realizada la selección, procede a realizar la cruza.
5	FCRU	MUT	Terminado el proceso de cruza, inicia la mutación
6	FMUT	EV	Terminada la mutación de la población inicial, ejecuta nuevamente el proceso de evaluación, si encuentra la solución al problema (cero ataques) detiene el proceso y activa el despliegue de resultados, de lo contrario regresa al paso 3 y repite el ciclo para una nueva generación.

Módulo de Control. Este módulo realiza el control de la operación sincronizada de todos los módulos de la implementación. La primera función que realiza es generar una señal de *reset* a las memorias para que se configuren en sus respectivos módulos de acuerdo a los requerimientos de la implementación. Al recibir la señal *"ini"* inicia la operación del algoritmo, para lo cual éste va generando las señales de forma ordenada para que se vayan ejecutando las funciones de cada módulo, así mismo realiza el control de las generaciones a las que se van aplicando el algoritmo, finalmente se encarga de activar el proceso para desplegar los resultados mediante el módulo de control VGA y la barra de leds. En la Tabla 1 se muestra la operación de este módulo.

Módulo Generador de Población Inicial. En este módulo se genera de forma aleatoria la población inicial de soluciones, la cual está constituida por 128 individuos de 48 bits cada uno que se almacenan en la memoria de población cuya capacidad es de 128x48. Cada individuo está codificado como se indica en la Fig. 2. Se propone emplear una población de 128 individuos debido a que, para poder manipular los 100 individuos se requiere de un contador de 7 Bits, y para aprovechar el total de las cuentas se decidió que la población inicial fuese de 128 individuos.

Como se indica en la Fig. 2, cada reina posee un total de 6 bits para su codificación, 3 son para la columna (C) y 3 son para el renglón (R), que indican la posición de cada Reina en un tablero de 8 Columnas y 8 Renglones. La generación de la población inicial se realiza mediante un proceso aleatorio, para lo cual se emplea una base de tiempo diferente a la de operación de la implementación; es decir, provienen de fuentes físicamente diferentes. El dato de 48 bits se forma por 6 contadores de 8 bits en *free runing*, los cuales inician en valores diferentes.

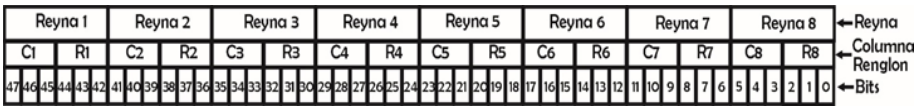

Fig. 2 Descripción de la codificación de una solución en la memoria de población, que contempla la codificación de las columnas y renglones de las 8 Reinas.

Estos contadores se detienen cada que el generador solicita un dato para almacenarlo en la memoria de población hasta completar 128 datos (soluciones o individuos).

La frecuencia de reloj empleada para este proceso aleatorio y todos los procesos aleatorios requeridos en la ejecución del algoritmo es de 50MHz que se obtiene de un oscilador que se encuentra en la misma tarjeta de desarrollo.

Módulo de Evaluación. En este módulo se encuentra implementada la función objetivo la cual determina el número de ataques que se presenta en cada solución (individuo) de la población. El cálculo del número de ataques se realiza por partes, primero se obtienen los ataques por comparación directa de las columnas y renglones de cada reina, enseguida se obtiene el número de ataques en diagonal con pendiente positiva sumando columna más renglón de cada reina, se compara directamente la suma de cada reina y si son iguales se contabiliza un ataque. Finalmente se obtienen los ataques en diagonal con pendiente negativa restando la columna del renglón de cada reina, se compara directamente la resta de cada reina y si son iguales se contabiliza un ataque. Los resultados de la evaluación de cada individuo son almacenados en la memoria de evaluación de forma tal que la dirección del individuo que se evalúa corresponde a la misma dirección en la memoria de evaluación, donde se almacena el resultado. De esta forma, cada solución en la memoria de población posee su evaluación en la misma dirección, pero en la memoria de evaluación. Cuando el número de ataques es cero, se detiene el proceso y este módulo lo indica a través de la señal *"reslist"* y proporciona la dirección de la memoria de población donde se encuentra la solución con ataques cero a través de las líneas *"Dirresul"*.

Módulo de Selección. En este módulo se realiza el proceso de selección mediante el método de torneo, el cual consiste en seleccionar de forma aleatoria parejas y determinar cuál de ellas posee la mejor evaluación (menor número de ataques). Para seleccionar la pareja de la primera solución de la memoria de población, se genera una dirección aleatoria entre 0 y 127. Con esta dirección se obtiene la pareja de la misma memoria de población, con esta misma dirección se obtiene el número de ataques de la pareja en la memoria de evaluación. La dirección de la pareja y el número de ataques se almacena en la memoria de selección en la primera localidad, el proceso lo repite con la segunda dirección de la memoria de población hasta recorrer las 128 direcciones. La codificación del dato que se almacena en la memoria de selección es de 12 bits, los primeros 5 bits corresponden al número de ataques de la pareja y los 7 bits restantes corresponden a su dirección. Para la selección, recorre la memoria de evaluación y la de selección, compara el número de ataques, si el número de ataques en la memoria de evaluación es mayor, se procede a reemplazar la solución de la memoria de población con su pareja, y el proceso se concluye al terminar de recorrer las 128 localidades de las memorias de evaluación y selección. Cabe aclarar que la nueva población resultante, fungirán como padres en el proceso de cruza.

Módulo de Cruza. Este módulo realiza la cruza de cada una de las soluciones (padres) resultado del proceso de selección que se ha almacenado en la memoria de población, el resultado de la cruza son los hijos, los cuales se almacenan nuevamente en la memoria de población obteniéndose ahora una población de hijos. En este proceso se generan 2 números aleatorios que corresponden a la probabilidad de cruza y al punto de cruza. La probabilidad se obtiene de un contador que opera libremente de 0 a 100, el punto de cruza se obtiene de otro contador que opera libremente de 0 a 47 ya que la longitud de cada solución es de 48 bits. La probabilidad de cruza seleccionada es del 80%, siendo el operador de cruza el operador principal y en la práctica, dicha probabilidad generalmente se elige en un rango del 60% al 100%. Con el punto de cruza se genera una palabra de control de cruza de 48 bits, que está formada por ceros del bit más significativo (47) hasta el punto de cruza y después del punto con unos hasta el bit menos significativo (0); de esta palabra también se obtiene su negado. El proceso de cruza se efectúa de la siguiente forma: se leen dos padres contiguos de la memoria de población, de cada padre se obtienen 2 registros; registro a, aplicando una función AND con la palabra de control y registro b, aplicando una función AND con el negado de la palabra de control. Posteriormente se obtiene el hijo uno realizando una función OR del registro 1b con el registro 2a, el segundo hijo se obtiene efectuando una función OR del registro 2b con el registro 1a. De este proceso se generan dos hijos que reemplazan a sus padres y se almacenan nuevamente en sus respectivas direcciones de la memoria de población. El proceso se repite para cada 2 padres hasta recorrer la totalidad de la memoria de población.

Módulo de Mutación. El proceso de mutación se aplica a cada hijo resultante del proceso de cruza, que se encuentran almacenados en la memoria de población. Se lee cada hijo y se recorre bit por bit desde el 0 hasta el 47. En cada bit que se va recorriendo, se obtiene la probabilidad aleatoria de cruza y si es menor o igual al 2% se procede a la mutación del bit en cuestión que consiste en negar dicho bit; si la probabilidad es mayor no se hace la negación y se pasa al siguiente bit. Este proceso se realiza en la totalidad de los hijos de la memoria de población. La probabilidad aleatoria de mutación se obtiene de un contador de 0 a 1000 que opera libremente, cabe aclarar que la base de tiempo de este contador es diferente físicamente a la base de tiempo con la que opera la implementación. La probabilidad de mutación se estableció de acuerdo a la longitud del individuo, definiéndola como *1/Lind*, donde *Lind* corresponde a 48 y es aproximadamente 0.02, lo que corresponde al 2% de mutación, mutando así en promedio un bit de cada individuo en promedio.

Módulo de lectura Memoria de Población. La función de este módulo inicia cuando el Módulo de Evaluación activa la señal "*reslist"* que significa que encontró una solución que genera cero ataques entre las reinas, así mismo le envía la dirección de dicha solución mediante bus "*Dirresul"*, con la dirección se lee la solución de la memoria de población, que es la solución al problema, ésta la envía al Módulo de control del Monitor VGA mediante el bus *"Sol"*. La señal *"reslist"* también la recibe el Módulo de Control con la finalidad de que proporcione el número de generación donde se encontró la solución. Dicho número de generación será desplegado en la barra de leds de 11bits.

Módulo de Control Monitor VGA. Este módulo se encarga de desplegar el resultado del algoritmo mediante un monitor VGA. Envía las señales de control al monitor, el bus RGB (3 Bits), las señales de Pixel "x" y Pixel "y" y las señales de sincronización.

Tabla 2. Resultados de las pruebas de desempeño de la implementación.

N° Pruebas	Generaciones	Pulsos	Tiempo [ms]
42	240.69	$6.1322x10^6$	1985.88
	[4, 1581]	[$6.6560x10^4$, $9.9008x10^7$]	[33.28, 13153.92]

Tabla 3. Resultados de las pruebas de desempeño de la implementación en software.

N° de Pruebas	Generaciones	Tiempo [seg]
42	206.17	507.14
	[69, 1650]	[136, 4548]

El monitor empleado es un VGA HP LV1911 con una resolución de 1366X768. El despliegue de los resultados se realiza de forma gráfica; es decir, el módulo genera un tablero de ajedrez de 8 columnas por 8 renglones, así mismo despliega la figura de las reinas en cada una de las posiciones según lo indique la solución recibida. El diseño de la figura de la reina se realizó pixel por pixel, de forma que el diseño de la figura es propio del presente trabajo.

5. Análisis de resultados

Las pruebas realizadas a la implementación se ilustran en la Tabla 2, se presenta el número de pruebas, el número de generaciones requeridas para llegar a la solución (cero ataques entre las reinas), el total de pulsos empleados, y el tiempo requerido. La cantidad de pruebas se consideraron 42 ya que se observó que los resultados en un número mayor a 42 era muy similar. El tiempo se calculó de acuerdo a la frecuencia del reloj empleado para la operación de la implementación, que fue de 2MHz. Esta señal de reloj se implementó mediante un oscilador integrado de la marca SARONIX NTC040C de 2Mhz, así mismo se calculó el tiempo que requiere la implementación para ejecutar el algoritmo en cada generación, contabilizando el número de pulsos que requiere cada módulo del algoritmo sumando un total de 16640 pulsos por generación.

De acuerdo con las pruebas realizadas, el tiempo promedio que requiere la implementación para llegar a la solución es de 1985.88 ms o 1.98588 seg en 42 pruebas realizadas. La mejor ejecución requirió de únicamente 4 generaciones y un tiempo de 33.28 ms; mientras que la ejecución que requirió más generaciones ocupó 1581 con un tiempo de 13.153 seg, representados en la Tabla 2 con los valores mínimos y máximos entre corchetes.

El desempeño de la implementación en FPGA es comparada con la implementación del mismo GA en software, para resolver el problema de ubicar 8 reinas en un tablero de ajedrez de 8x8 sin que se ataquen entre ellas. Dicha implementación se realizó usando MATLAB, considerando el toolbox de Algoritmos Genético del intérprete MATLAB, para mayor rapidez en la comparación. Sin embargo, el tiempo requerido de ejecución en software mejora si se implementa usando un lenguaje de alto nivel como “C”, dado que es un lenguaje de programación y no un intérprete. Las pruebas de la implementación en software se realizaron empleando una computadora DELL XPS con un sistema operativo WINDOWS 7 de 64 bits, con un procesador INTEL CORE i7-3770 y una frecuencia de reloj de 3.4 GHz. En la Tabla 3 se muestran los resultados

Tabla 4. Comparación de desempeño de las implementaciones en FPGA's y en MATLAB.

Implementación	N° de Pruebas	Promedio Generaciones	Promedio Tiempo [seg]
Hardware (FPGA's)	42	240	1.985
Software (MATLAB)	42	206	507.14

Tabla 5. Proyección del tiempo de la implementación en FPGA si se empleara una base de tiempo de 3.00 GHz.

N° de Pruebas	Generaciones	Pulsos	Tiempo [μs]
42	240.69 [4, 1581]	$6.1322x10^6$ [$6.6560x10^4$, $9.9008x10^7$]	1333.3 [22.18, 8770]

de la prueba, en dicha tabla se muestra el número de pruebas, el número de generaciones en la que encontró el resultado (cero ataques entre las reinas) y el tiempo requerido para llegar al resultado.

De las pruebas realizadas, en la Tabla 3 se muestra el tiempo promedio requerido por la implementación en MATLAB que es de 507.14 segundos. Al igual que en la Tabla 2, las generaciones y tiempos mínimos y máximos de las 42 ejecuciones se presentan en corchetes. En la Tabla 4 se muestra una comparación de los promedios obtenidos de las dos implementaciones.

De los resultados mostrados en las tablas anteriores se puede observar que la implementación en Hardware a través de FPGA's es 500 veces más rápida que la implementación en software empleando MATLAB. Si la implementación en software se realizará utilizando algún lenguaje de alto nivel como "C" o "JAVA", el tiempo requerido se reduciría ya que MATLAB es un software de aplicación y no un lenguaje de programación.

Es importante considerar que la base de tiempo de la implementación en FPGA es de 2MHz y la base de tiempo que emplea la computadora donde se probó la implementación en MATLAB es de 3.4 GHz. Ahora bien, si proyectamos los resultados de desempeño de la implementación en FPGA empleando de forma hipotética una base de tiempo de por ejemplo 3GHz, los resultados de la Tabla 2 serían los que se muestran en la Tabla 5.

De los datos mostrados en la Tabla 5, el tiempo que se emplearía para llegar a la solución es mucho menor, en promedio 1.333 ms, los valores mostrados entre corchetes representan los valores mínimos y máximos para llegar a la solución. Para poder realizarlo se tendría que investigar si las plataformas de FPGA existentes soportan la frecuencia propuesta de 3GHz.

6. Conclusiones y trabajo a futuro

En el presente trabajo se desarrolló la implementación de un GA Simple en hardware mediante FPGA's para resolver un problema ejemplo de tipo combinatorio que consiste en ubicar en un tablero de ajedrez de 8 x 8 casillas 8 reinas sin que se ataquen entre ellas. El algoritmo resuelve el problema ubicando las 8 reinas sin ataques, la disposición

de las reinas no siempre es la misma, siendo este problema combinatorio un problema multimodal dado que se tiene más de una configuración del tablero con cero ataques (solución óptima).

Así mismo se realizó una comparación del desempeño de la implementación en FPGA's, con la implementación del mismo algoritmo en software empleando MATLAB resolviendo el mismo problema, comprobando que la implementación en hardware es más rápida. En este caso particular resultó ser 500 veces más rápida aproximadamente tal como se indica en la Tabla 4. También se presenta una proyección de tiempo si se empleara hipotéticamente una base de tiempo para la implementación con FPGA´s de 3GHz y evidentemente el tiempo es mucho menor, pudiendo existir la posibilidad de poder aplicar el algoritmo en problemas donde el tiempo es un factor muy importante.

Como trabajo futuro se plantea implementar el AG en un lenguaje de alto nivel para comprobar que el tiempo mejora con respecto a la implementación con MATLAB, así mismo agregar al algoritmo un módulo de elitismo, para mejorar aún más el desempeño, usar una frecuencia de reloj externa mayor a 2 MHz. Se plantea la posibilidad de utilizar FPGA's de última generación que permitan un ancho de banda mayor para poder utilizar frecuencias de reloj más altas para tener un desempeño mucho mejor.

Referencias

1. Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning. Addison Wesley (1989)
2. Sivanandam, S.N., Deepa, S.N.: Introduction to genetic algorithms. Springer, pp. 15–30 (2008)
3. Kuri, A., Galaviz, J.: Algoritmos genéticos. Fondo de cultura Económica, 3(3), pp. 13–30 (2000)
4. Smith, G.R.: FPGA's 101 Everything you need to know to get stated. Elsevier Inc., pp 43–55 (2010)
5. Scott, S.D., Samal, A., Seth, S.: HGA: A hardware-based genetic algorithm. In: Proceedings ACM International Symposium on Field Programable Gate Arrays, pp. 53–59 (1995)
6. Tufte, G., Haddow, P.: Evolving and adaptive digital filter. In: NASA/DoD Workshop on Evolvable Hardware. pp. 143–150 (2000)
7. Aporntewan, C., Chongstitvatana, P.: A hardware implementation of the compact genetic algorithm. In: Proceedings IEEE Congress on Evolutionary Computation Seoul Korea, pp. 624–629 (2001)
8. Shackleford, B., Snider, G., Carter, R.J., Okushi, E., Yasuda, M., Seo, K., Yasuura, H.: A high-performance pipelined, FPGA-based genetic algorithm machine. Genetic Programming and Evolvable Machines, Springer, 2, pp. 33–60 (2001)
9. Tang, W., Yip, L.: Hardware implementation of genetic algorithms using FPGA. In: IEEE International Midwest Symposium on Circuits and Systems, pp. 549–552 (2004)
10. Mostafa, H.E., Khadragi, A.I., Hanafi, Y.Y.: Hardware implementation of genetic algorithm on FPGA. In: National Radio Science Conference (2004)
11. Thirer, N.: About the FPGA implementation of a genetic algorithm for solving sudoku puzzles. In: IEEE 27th Convention of Electrical and Electronics Engineers in Israel (IEEEI) (2012)

12. Tuncer, A., Yildirim, M.: Design and implementation of a genetic algorithm IP core on an FPGA for path planning of mobile robots. Turkish Journal of Electrical Engineering and Computer Sciences, 24(6), pp. 5055–5067 (2016)
13. Alansi, M., Elshafiey, I., Al-Sanie, A., Mabrouk, A.: FPGA implementation of multi-user detection genetic algorithm tool for sdma-ofdm systems. Springer, 86, pp. 1241–1263 (2016)
14. Guo, L., Thomas, D.B., Cross, A.I., Fu, H.: Parallel genetic algorithms on multiple FPGA's. In: ACM SIGARCH Computer Architecture News, 43(4), pp. 86–93 (2017)
15. Thorbole, P.S., Kalbhor, S.D., Harpale, V., Bairagi, V.K.: Hardware implementation of genetic algorithm for epileptic seizure detection and prediction. In: International Conference on Computing, Communication, Control and Automation (ICCUBEA) (2017)
16. Praveena, M., N.B., Naidu, C.D.: FPGA implementation of high-speed medical image segmentation using genetic algorithm. Journal of Theoretical and Applied Information Technology, 95(13), pp. 2981–2988 (2017)
17. Peker, M.: A fully customizable hardware implementation for general purpose genetic algorithms. Applied Soft Computing, 62, pp. 1066–1076 (2018)
18. Torquato, M., Costa-Fernandes, M.A.: High-performance parallel implementation of genetic algorithm on FPGA. Circuits, systems, and signal processing, 38(7), pp. 4014–4039 (2019)
19. Gupta, V., Jain, A., Chourasia, B.: FPGA Based Implementation of Genetic Algorithm Using VHDL. Implementation of generic algorithm using VHDL on FPGA. International Journal for Recent Developments in Science and Technology (2011)
20. Xilinx: http://xilinx.com/products/boards-and-kits/HW-SPAR3E-SK-US-G.htm (2019)
21. Ashenden, P.: The Designer's Guide to VHDL. 3 (1995)