ISSN 2007-9737

Comparación de algoritmos evolutivos multi-objetivo para síntesis de alto nivel en dispositivos FPGA

Darian Reyes Fernández de Bulnes, Yazmin Maldonado

Instituto Tecnológico de Tijuana, Tijuana,
México

{darian.reyes, yaz.maldonado}@tectijuana.edu.mx

Resumen. La planificación de operaciones es un problema fundamental al mapear un diseño en un dispositivo electrónico, tal como un arreglo de compuertas programables en campo (*FPGA*, por sus siglas en inglés: *Field Programmable Gate Array*). En este artículo, se presenta una propuesta de aplicar algoritmos evolutivos multi-objetivo, tales como, *NSGA-II*, *SPEA2* y *NSGA-III* para la optimización simultánea de área, retardo y potencia en la etapa de síntesis de alto nivel en los *FPGAs*. Se compararon los resultados obtenidos de cada algoritmo evolutivo multi-objetivo utilizando los indicadores de calidad, Épsilon, Hipervolumen y R, los cuales muestran cuales algoritmos son los más adecuados para este problema a resolver. Además, los resultados de los procesos evolutivos son analizados y se proponen mejoras.

Palabras clave. DFG, FPGA, NSGA-II, NSGA-III, SPEA2, optimización de circuitos.

Comparison of Multi-objective Evolutionary Algorithms for High Level Synthesis in FPGA Devices

Abstract. Operations scheduling is a fundamental problem of mapping a design into an electronic device, such as a Field Programmable Gate Array (FPGA). In this paper, a proposal for apply the multi-objective evolutionary algorithms, such as, NSGA-II, SPEA2 and NSGA-III for the simultaneous optimization of area, delay and power in the in high-level synthesis stage in the FPGAs is presented. Results obtained are compared using the quality indicators, Epsilon, Hypervolume and R, these results shown which of the algorithms are the most suitable for this problem to solve. Then, the results of the evolutionary processes are analyzed and improvements are proposed.

Keywords. DFG, FPGA, NSGA-II, NSGA-III, SPEA2, circuit optimization.

1. Introducción

El arreglo de compuertas programables en campo (*FPGA*, por sus siglas en inglés: *Field Programmable Gate Array*), es un dispositivo semiconductor que contiene bloques de lógica programable cuya interconexión y funcionalidad puede ser configurada *in situ* mediante un lenguaje de descripción de alto nivel (*HDL*, por sus siglas en inglés: *High Description Language*), tales como, VHDL o Verilog. Un *FPGA* se caracteriza por ser programable o reconfigurable, y es justamente esta propiedad la que lo distingue de otros dispositivos similares, como el circuito integrado de aplicación específica (*ASIC*, por sus siglas en inglés: *Application-Specific Integrated Circuit*). En la actualidad, los *FPGAs* son utilizados en varios escenarios, como en robótica, comunicaciones y en equipos electrodomésticos [5, 11, 26]. Dentro de un *FPGA* existen varios tipos de conexiones entre ellos, los bloques lógicos configurables (*CLBs*, por sus siglas en inglés: *Configurable Logic Blocks*) y los bloques de entrada-salida (*IOBs*, por sus siglas en inglés: *Input-Output Blocks*): las conexiones directas, las conexiones de propósito general y las líneas de largo recorrido. Estas conexiones se hacen de manera aleatoria, por lo que diseñar el algoritmo de manera óptima, reduciendo los recursos según la arquitectura del dispositivo *FPGA*, es un problema científico que debe ser tratado.

Para crear una implementación en un *FPGA*, el diseñador debe especificar la *HDL* del circuito, para luego sintetizar en el dispositivo una descripción estructural de transferencia de registros (*RTL*, por sus siglas en inglés: *Register-Transfer Level*), que cumple ese comportamiento. Esto último se realiza con una herramienta de diseño asistido por computadora (*CAD*, por sus siglas en inglés: *Computer-Aided Design*). *HDL* consiste en declaraciones algorítmicas que contienen operaciones computacionales (adiciones, multiplicaciones, comparaciones y operadores lógicos) y operaciones de control (sentencias condicionales, bucles y llamadas a procedimientos) [19].

La descripción *RTL* mapea las operaciones en Unidades Funcionales (UFs) y distribuye las transferencias de datos en el dispositivo con una unidad de control. La unidad de control coordina el flujo de datos entre varias UFs, como lo son las unidades de hardware (*ALUs*, multiplicadores y compuertas lógicas), unidades de almacenamiento (registros, registros de archivos, *RAM* y *ROM*) y unidades de interconexión (multiplexores). Estos últimos están conectados entre sí para lograr un comportamiento especificado.

El proceso de transformar una descripción algorítmica en una descripción *RTL* se le denomina síntesis de alto nivel (*HLS*, por sus siglas en inglés: *High-Level Synthesis*). En la mayoría de los sistemas de *HLS*, la descripción algorítmica se representa como un grafo acíclico dirigido (*DAG*, por sus siglas en inglés: *Directed Acyclic Graph*), llamado grafo de flujo de datos (*DFG*, por sus siglas en inglés: *Data Flow Graph*) [23]. Un *DFG* es planificado y optimizado con los tres procesos necesarios en *HLS* (asignación, mapeo y planificación) [28]. Luego es necesario convertir el *DFG* planificado a *RTL*. Para finalizar, el *RTL* es sintetizado en el dispositivo *FPGA* con una herramienta de automatización de diseño electrónico (*EDA*, por sus siglas en inglés: *Electronic Design Automation*), como Xilinx ISE Design Suite [31] o Vivado Design Suite [32]. De igual manera, utilizando el *DFG* se hace viable un proceso de optimización de recursos, tales como: retardo, área ocupada y consumo de potencia. Este tipo de optimización para la etapa de *HLS* es independiente de la arquitectura del dispositivo *FPGA*, lo cual hace que pueda funcionar en cualquier modelo, familia o fabricante.

La principal contribución de esta investigación es la aplicación y comparación de algoritmos evolutivos multi-objetivo (*MOEAs*, por sus siglas en inglés: *Multi-objective Evolutionary Algorithms*), para la optimización simultánea de retardo, área ocupada y consumo de potencia en *HLS*, para dispositivos *FPGAs*. Particularmente utilizando el algoritmo *NSGA-III*, el cual, según la literatura, es un algoritmo que tiene un mejor rendimiento cuando se tienen más de 3 funciones objetivo, consideramos pertinente su utilización para la solución de este problema debido a que en un futuro pretendemos aumentar la cantidad de funciones objetivo y con ello explotar al máximo su rendimiento ya que es un algoritmo preparado para enfrentar esto. Además, no ha sido reportado en la literatura para resolver este problema, así como la comparación de los resultados con otros algoritmos como el *NSGA-II* y *SPEA2*.

El resto del artículo está organizado de la siguiente manera. En la *sección 2*, son descritos algunos trabajos anteriores relacionados con esta investigación. En la *sección 3*, se presenta la definición de *DFG* y el banco de pruebas *Mediabench*. El proceso de optimización llevado a cabo con tres *MOEAs* se detalla en la *sección 4*. En la *sección 5*, se presentan los resultados de los indicadores de calidad y las gráficas con las trazas de datos del proceso evolutivo. Para finalizar, se concluye el artículo y se exponen algunas notas de trabajo futuro en la *sección 6*.

2. Trabajos relacionados

El uso de *MOEAs* para mejorar el proceso de *HLS* ha sido comúnmente reportado en la literatura. La aplicación de un *MOEA* para la optimización de retardo, área, y potencia en *HLS* es presentado en [14]. Un resumen de metodologías basadas en *MOEAs* para *HLS* es presentado en [19]. Una demostración de que las heurísticas basadas en población de algoritmos evolutivos (*EAs*, por sus siglas en inglés: *Evolutionary Algorithms*), son muy efectivas en la exploración del espacio de diseño se

expone en [19]. Los autores proponen un esquema de representación en dos componentes para la planificación de los *DFGs*. Una mejora al consumo de potencia de los dispositivos *FPGAs* basado en la unificación de rutas desde un *DFG* es presentado en [15]. Después de varios años de investigación por los autores, varias aplicaciones de *MOEAs* en *HLS* fueron presentadas en [2]. En una investigación previa [24], realizamos un análisis de seis MOEAs para HLS, demostrando que los algoritmos *NSGA-II* [9, 10] y *SPEA2* [36] tienen un desempeño mejor que el resto de los *MOEAs* analizados.

3. Preliminares

3.1. Grafo de flujo de datos

Un *DFG* representa un diseño en *HLS*. Entendiendo como diseño al conjunto de instrucciones que secuencialmente o en paralelo, deben ejecutarse para obtener los resultados requeridos [27]. Cada nodo del grafo representa una región contigua de código sin ramas, y las aristas entre los nodos indican saltos en el flujo de los datos. La función de un nodo es generar un nuevo valor a su salida, dependiendo de sus entradas. Formalmente, un *DFG* está definido de la siguiente manera [17]:

$$G = (V, E), \tag{1}$$

donde:

1. $V = \{v_1, ..., v_n\}$, es un conjunto finito cuyos elementos son nodos.
2. $E \subset V \times V$ es una relación asimétrica de flujo de datos, cuyos elementos son aristas dirigidas que representan datos.

3.2. Banco de pruebas *Mediabench*

El banco de pruebas *Mediabench* [21, 22], fue utilizado para realizar y validar varios experimentos para la optimización de *DFGs*. Elegimos este banco de pruebas dado que es un punto de referencia comúnmente utilizado en la literatura [2, 14, 15, 19, 30].

Mediabench cuenta con una base de 20 circuitos accesibles en formato **.dot* con una estructura en *DAG*. Por lo tanto para utilizarlo, diseñamos un algoritmo implementado en lenguaje *C* para realizar la conversión de los *DFGs* de formato **.dot* a formato **.graphml* (formato de grafos estándar escritos en *xml*). Ya teniendo los *DFGs* en **.graphml*, estos pueden ser graficados y estudiados con la herramienta yEd [34]. Además, estando los *DFGs* en formato estándar *xml*, es posible leerlos desde lenguaje *C* con la librería *pugixml* [18].

Las características de cada uno de los *DFGs* del *Mediabench* (cantidad de nodos, cantidad de aristas y cantidad de operaciones) se detallan en la tabla 1.

Tabla 1. Características de los *DFGs* del banco de pruebas *Mediabench* para 20 circuitos

DFG	Nodos	Aristas	Operaciones
HAL	11	8	4
MESA - Horner Bezier (MESA-HB)	18	16	4
Auto Regression Filter (ARF)	28	30	2
MPEG - Motion Vectors (MPEG-MV)	32	29	4
Elliptic Wave Filter (EWF)	34	47	2
Finite Input Response Filter 2 (FIR2)	40	39	4
Finite Input Response Filter 1 (FIR1)	44	43	4
JPEG - Smooth (JPEG-SD)	51	52	5
MESA - Feedback (MESA-FP)	53	50	6
EPIC - Collapse_pyr (EPIC)	56	73	7
Cosine 1 (C1)	66	76	5
Cosine 2 (C2)	82	91	5
JPEG - Write BMP (JPEG-BMP)	106	88	8
MESA - Interpolate Aux (MESA-IA)	108	104	5
MESA - Multiplication (MESA-MM)	109	116	4
MPEG - Inverse (MPEG-IDCT)	114	164	7
JPEG - Inverse (JPEG-IDCT)	122	162	6
JPEG - Forward (JPEG-FDCT)	134	169	6
MESA - Smooth Triangle (MESA-ST)	197	196	4
MESA - Invert Matrix (MESA-IM)	333	354	7

4. Algoritmos evolutivos multi-objetivo

4.1. Representación del problema

Para planificar un *DFG* es necesario contar con una representación que la defina. La representación se realiza con dos componentes [14, 15, 19]:

1. Prioridad de los nodos: representa el orden en que son ejecutados los nodos.

ISSN 2007-9737

2. Cantidad de UFs: representa la cantidad de UFs disponibles para ejecutar las operaciones de los nodos.

En la figura 1, se observa un ejemplo de un *DFG* con una planificación descrita en los dos componentes de la representación. La prioridad de los nodos define primero ejecutar el nodo 3, luego el nodo 1 y así sucesivamente hasta el nodo 7. Y la cantidad de UFs establece que se tienen tres UFs (dos multiplicadores y un sumador).

En la figura 2, se muestra como queda el *DFG* planificado por ciclos de reloj según la disponibilidad definida de UFs, el orden de nodos a ejecutar y la dependencia de nodos en el grafo. En el primer ciclo de reloj habría que ejecutar las operaciones de multiplicación 3 y 1.

La operación 2 debe ejecutarse en el segundo ciclo de reloj porque las operaciones 3 y 1 ocuparon las dos UFs de multiplicación que habían disponibles en las restricciones de la cantidad de UFs.

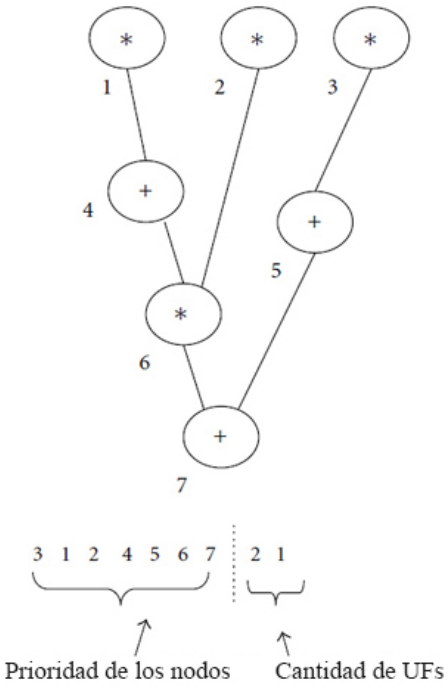

Fig. 1. Ejemplo de planificación de un *DFG*

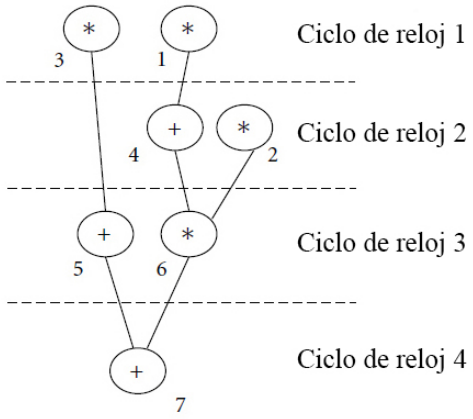

Fig. 2. Planificación de ciclos de reloj de un *DFG*

En [14, 19, 24], se detallan todos los aspectos de la implementación de la representación del problema.

4.2. Estimadores de calidad

Cada planificación es sometida a 3 estimaciones para medir su calidad (retardo, área y potencia) y de esta manera poder hacer comparaciones. Estas estimaciones son propuestas en [14, 19, 20].

1. Retardo: Es el número de pasos de tiempo o ciclos de reloj necesarios para ejecutar el *DFG*. En el ejemplo de la figura 2 el retardo es 4 al ser calculado el *DFG* en 4 ciclos de reloj.
2. Área: Es el total de UFs más los registros necesarios para ejecutar a todos los nodos del *DFG*. La cantidad de UFs es conocida a partir de la planificación, sumando todos los valores de la cantidad de UFs. La cantidad de registros se obtiene con el algoritmo *Left Edge*, descrito en [12].
3. Potencia: Para estimar la potencia primero se crea un grafo de compatibilidad, donde un nodo *i* se relaciona con otro nodo *j*, si la operación del nodo *j* puede ejecutarse en la misma UF del nodo *i*. Para esto, tienen que tener el mismo tipo de operación y sus variables no pueden solaparse en el tiempo.

Luego, la potencia (P) es calculada por la siguiente fórmula:

$$P = (\frac{n}{m_1}) + m_2 + m_3, \quad (2)$$

donde m_1 es el número total de aristas en el grafo de compatibilidad. La variable n hace a m_1 inversamente proporcional a P. Si m_1 es grande significa que existe flexibilidad de movilidad, por lo que P debe disminuir. La variable m_2 es el promedio de los pesos de las aristas del grafo de compatibilidad, que estén por debajo de $K\,\%$ del rango de valores de pesos de aristas para cada nodo. La variable m_3 es una generalización de m_2 con $K = 100\,\%$. La variable K es definida en 60 % en todos nuestros experimentos, pero puede ser modificada de acuerdo al *DFG*.

En [14, 24], se detallan todos los aspectos de la implementación de los estimadores de calidad.

4.3. Algoritmos evolutivos multi-objetivo e indicadores de calidad

En este artículo, se propone optimizar los *DFGs* con *MOEAs*, y realizar comparaciones de los dos que mejor resultados alcanzaron en [24] junto con un *MOEA* que aún no ha sido utilizado en la literatura para este tipo de problema (*NSGA-III*). Se utiliza la plataforma de interfaz independiente para algoritmos de búsqueda (*PISA*, por sus siglas en inglés: *Platform Independent Interface for Search Algorithms*) [4, 3]. Los *MOEAs* propuestos son:

1. *Nondominated Sorting Genetic Algorithm II (NSGA-II)* [9, 10],
2. *Strength Pareto Evolutionary Algorithm 2 (SPEA2)* [36],
3. *Nondominated Sorting Genetic Algorithm III (NSGA-III)* [8].

Con el objetivo de comparar la calidad de los Frentes de Pareto (FPs) alcanzados por cada *MOEA*, se utilizan 3 indicadores de calidad:

1. Épsilon [37],
2. Hipervolumen [1],
3. R [13].

NSGA-II y *SPEA2* fueron descritos en trabajo previo [24], a continuación se presenta el funcionamiento del algoritmo *NSGA-III*, dada la novedad de su utilización ante este problema de optimización.

4.4. *NSGA-III*

NSGA-III fue presentado en [8], como un nuevo algoritmo de ordenamiento no dominado enfocado en problemas de tres o más funciones objetivo. El algoritmo consiste en mantener el esquema general de *NSGA-II*, pero incorporándole una técnica de puntos de referencia [7]. Esta técnica sustituye al segundo criterio de selección -distancia de apilamiento- entre soluciones no dominadas. Estos puntos de referencia representan las regiones del FP que deben ser descubiertas, con el objetivo de mantener la diversidad en la población resultante de cada generación. Para discriminar entre las soluciones no dominadas se utiliza una función de utilidad, cuyo valor indica la relevancia de una solución para aproximar un punto de referencia. En la figura 3 [25], se muestra un ejemplo de la diferencia entre el funcionamiento de la distancia de apilamiento que emplea *NSGA-II* y la técnica empleada por *NSGA-III*. Todos los círculos representan los individuos del primer frente no dominado de la población, y los círculos negros representan los individuos seleccionados para la siguiente generación. La efectividad de este algoritmo fue analizada en varios problemas de hasta 15 objetivos en [8]. Los resultados experimentales mostraron que posee un desempeño competitivo respecto a dos versiones de MOEA/D [35] frente a problemas continuos.

La cantidad de puntos de referencia que se crean en el hiperplano está definido por la siguiente ecuación:

$$H = \binom{M + p - 1}{p} = \frac{(M + p - 1)!}{(M - 1)! \times p!}, \quad (3)$$

donde H es la cantidad de puntos de referencia definidos en el hiperplano, M es la cantidad de

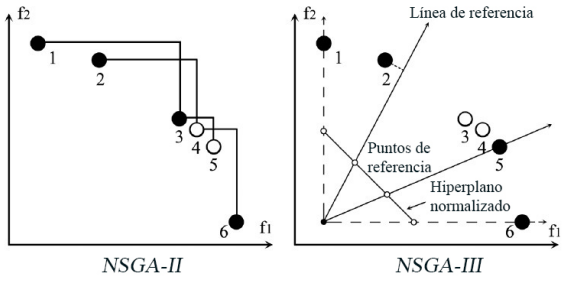

Fig. 3. Ejemplo de la diferencia entre el funcionamiento de *NSGA-II* y *NSGA-III*

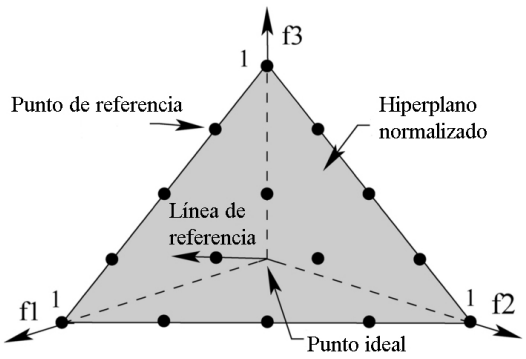

Fig. 4. Distribución de 15 puntos de referencia en el hiperplano para un problema de 3 funciones objetivo

funciones objetivo y p es la cantidad de divisiones en las abscisas.

En la figura 4, se muestra un ejemplo de como se distribuyen 15 puntos de referencia en el hiperplano para un problema de 3 funciones objetivo. En la figura 5, se presenta un ejemplo de como se realiza la asociación de los individuos con los puntos de referencia [8].

El problema de optimización en *HLS* de asignación, mapeo y planificación puede tener más de 3 funciones objetivo [6]. Y en este caso, la utilización del algoritmo *NSGA-III* adquiere más importancia, ya que en [8] se demuestra su buen desempeño con muchas funciones objetivo (entre 4 y 15).

Todos los experimentos los realizamos en *PISA*, adaptando el artículo de [29] a esta plataforma, en donde se presenta una implementación del algoritmo en lenguaje C++.

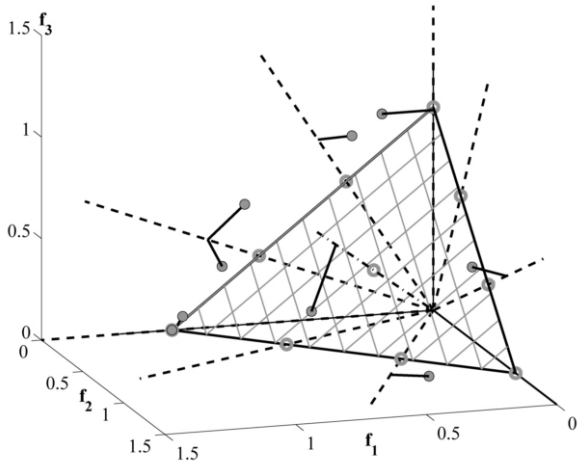

Fig. 5. Asociación de los individuos con los puntos de referencia en el hiperplano

En la figura 6, presentamos los puntos de referencia y el FP que alcanza *NSGA-III* contra el problema continuo DTLZ2 en 3 dimensiones [8], en donde se puede apreciar una buena distribución de individuos.

Otro ejemplo es mostrado en la figura 7, donde se presentan los puntos de referencia y el FP que alcanza *NSGA-III* contra el problema contínuo DTLZ4 en 3 dimensiones [8].

Los ejemplos anteriores son problemas continuos, sin embargo, nosotros pretendemos demostrar que *NSGA-III* puede tener el mismo rendimiento (mejorar la diversidad del FP) con un problema discreto como el de la optimización en *HLS*.

5. Resultados

En la tabla 2, se muestran los parámetros utilizados en todos los experimentos para los 3 *MOEAs*, posteriormente los resultados son analizados utilizando los 3 indicadores de calidad en la sección 4.3.

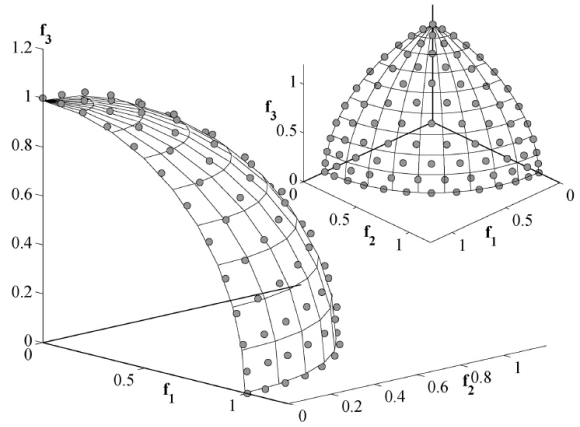

Fig. 6. *NSGA-III* contra el problema continuo DTLZ2

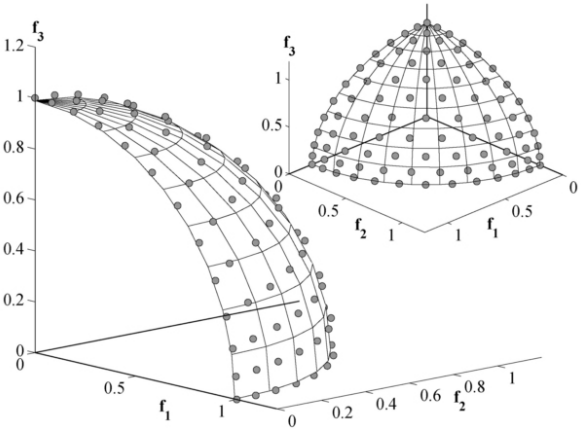

Fig. 7. *NSGA-III* contra el problema continuo DTLZ4

5.1. Resultados de los indicadores de calidad

Se realizaron comparaciones entre *NSGA-III* y los dos *MOEAs*: *NSGA-II* y *SPEA2*. La tabla 3 presenta las medias del indicador de calidad Épsilon, la tabla 4 presenta las medias del indicador de calidad Hipervolumen y la tabla 5 presenta las medias del indicador de calidad R. Se señala en negrita el mejor resultado para cada *DFG*.

La tabla 6 presenta los porcentajes de mejoras para cada *DFG*. Las mejoras son de *NSGA-III* respecto a *NSGA-II* y *SPEA2*. Se señalan en negrita los resultados positivos.

Tabla 2. Parametrización de los experimentos

Parámetro	Valor
Número de corridas	20
Tamaño de población	28
Generaciones	100
Cantidad de evaluaciones	2800
Funciones objetivo	3

Tabla 3. Resultados de las medias del indicador de calidad Épsilon para los algoritmos *NSGA-II*, *SPEA2* y *NSGA-III*

DFG	NSGA-II	SPEA2	NSGA-III
MESA-FP	1.81E-01	**1.54E-01**	1.60E-01
EPIC	2.30E-01	**2.24E-01**	**2.24E-01**
C1	1.60E-01	1.52E-01	**1.46E-01**
C2	1.47E-01	1.37E-01	**1.32E-01**
JPEG-BMP	1.43E-01	1.51E-01	**1.42E-01**

Tabla 4. Resultados de las medias del indicador de calidad Hipervolumen para los algoritmos *NSGA-II*, *SPEA2* y *NSGA-III*

DFG	NSGA-II	SPEA2	NSGA-III
MESA-FP	2.42E-01	**2.16E-01**	2.31E-01
EPIC	**3.28E-01**	3.32E-01	3.50E-01
C1	2.54E-01	**2.28E-01**	2.29E-01
C2	2.29E-01	**2.15E-01**	2.50E-01
JPEG-BMP	**2.21E-01**	2.31E-01	2.42E-01

Tabla 5. Resultados de las medias del indicador de calidad R para los algoritmos *NSGA-II*, *SPEA2* y *NSGA-III*

DFG	NSGA-II	SPEA2	NSGA-III
MESA-FP	3.77E-02	**3.17E-02**	3.39E-02
EPIC	5.27E-02	**5.18E-02**	5.70E-02
C1	3.53E-02	3.00E-02	**2.87E-02**
C2	3.29E-02	**2.96E-02**	3.27E-02
JPEG-BMP	**3.15E-02**	3.35E-02	3.30E-02

Tabla 6. Porcentajes de mejoras para cada caso de *NSGA-III* respecto a los algoritmos *NSGA-II* y *SPEA2*

Mejora respecto al **MOEA**	Indicador de calidad	**MESA-FP**	**EPIC**	**C1**	**C2**	**JPEG-BMP**
NSGA-II	Épsilon	**11.60 %**	**2.61 %**	**8.75 %**	**10.20 %**	**0.70 %**
	Hipervolumen	**4.55 %**	-6.71 %	**9.84 %**	-9.17 %	-9.50 %
	R	**10.08 %**	-8.16 %	**18.70 %**	**0.61 %**	-4.76 %
SPEA2	Épsilon	-3.90 %	0.00 %	**3.95 %**	**3.65 %**	**5.96 %**
	Hipervolumen	-6.94 %	-5.42 %	-0.44 %	-16.28 %	-4.76 %
	R	-6.94 %	-10.04 %	**4.33 %**	-10.47 %	**1.49 %**

La tabla 7 presenta la cantidad de casos y porcentaje que fue mejor el algoritmo *NSGA-III* respecto a los algoritmos *NSGA-II* y *SPEA2*.

En la mayoría de los casos, *NSGA-III* es mejor respecto a *NSGA-II*, con un 2.62 % de mejora promedio. Sin embargo, en casi todas las comparaciones de *NSGA-III* respecto a *SPEA2*, los resultados son negativos. También se observa que según el indicador de calidad Épsilon, *NSGA-III* respecto a *NSGA-II* tiene un promedio de mejora de 6.77 % con los 5 *DFGs* analizados.

Al analizar los FPs de *NSGA-III*, no existe una mejora significativa en la diversidad de los individuos, por lo que la hipótesis planteada no se cumplió. Notamos que la diversidad es muy parecida a la del *NSGA-II* y *SPEA2*. Sin embargo, *NSGA-III* en algunos casos logra ligeramente una mayor convergencia. Esto lo apreciamos cuando analizamos los FPs. En la figura 12 se presenta un ejemplo, con una gráfica del espacio objetivo y los FPs obtenidos por los *MOEAs NSGA-II*, *SPEA2* y *NSGA-III* para el *DFG C1*.

5.2. Gráficas de datos de los procesos evolutivos

Crear gráficas de las trazas de datos del proceso evolutivo nos permite detallar el comportamiento del mismo. De esta manera se pueden detectar debilidades y luego, con más precisión, se pueden proponer mejoras. Conociendo los datos, es posible crear gráficos de tendencia para analizar el comportamiento de la búsqueda.

Donde por cada generación, se observa la cantidad de individuos mutados que dominan a su padre, la cantidad de individuos cruzados que dominan al menos a uno de sus padres, la cantidad de individuos no-dominados en la población hija y la cantidad de individuos diferentes en la población hija.

En la figura 9, se presenta una gráfica de tendencias de los datos correspondientes a *NSGA-II* para el *DFG C1*. En la figura 10 se presenta una gráfica de tendencias de los datos correspondientes a *SPEA2* para el *DFG C1*. En la figura 11 se presenta una gráfica de tendencias de los datos correspondientes a *NSGA-III* para el *DFG C1*.

En las figuras anteriores se observan varios comportamientos interesantes. Existe mucha redundancia de individuos en las generaciones, debido a que los operadores de búsqueda en muchas ocasiones generan individuos iguales a los padres. El cruzamiento no está explorando bien el espacio de búsqueda, ya que es muy baja la cantidad de hijos que dominan a sus padres. Incluso se puede notar, que para los tres *MOEAs* disminuye la cantidad de hijos que dominan a sus padres obtenidos del cruzamiento, a medida que avanzan las generaciones.

Con el algoritmo *NSGA-III* se demora más en tener en todas las corridas una población completa no dominada, con respecto a los otros dos *MOEAs*. Estos temas serán estudiados a más detalle en trabajo futuro.

Además, en [16], ampliamos la información del espacio objetivo, graficando el movimiento generacional de los individuos para los *MOEAs NSGA-II* y *NSGA-III* utilizando 5 *DFGs* (una de las 20 corridas con 100 generaciones).

Notamos que con *NSGA-III* los individuos convergen más aceleradamente con respecto a *NSGA-II*. Esta característica se acentúa proporcional a la cantidad de nodos del *DFG*. Sin embargo, notamos que con *NSGA-III* hay mayor redundancia de individuos en las poblaciones.

Tabla 7. Cantidad de casos y porcentaje que fue mejor el *NSGA-III* respecto a los algoritmos *NSGA-II* y *SPEA2*

MOEA	**Indicador de calidad**	**C**asos que fue mejor	Porcentaje respecto a casos totales
NSGA-II	Épsilon	5	100 %
	Hipervolumen	2	40 %
	R	3	60 %
SPEA2	Épsilon	3	60 %
	Hipervolumen	0	0 %
	R	2	40 %

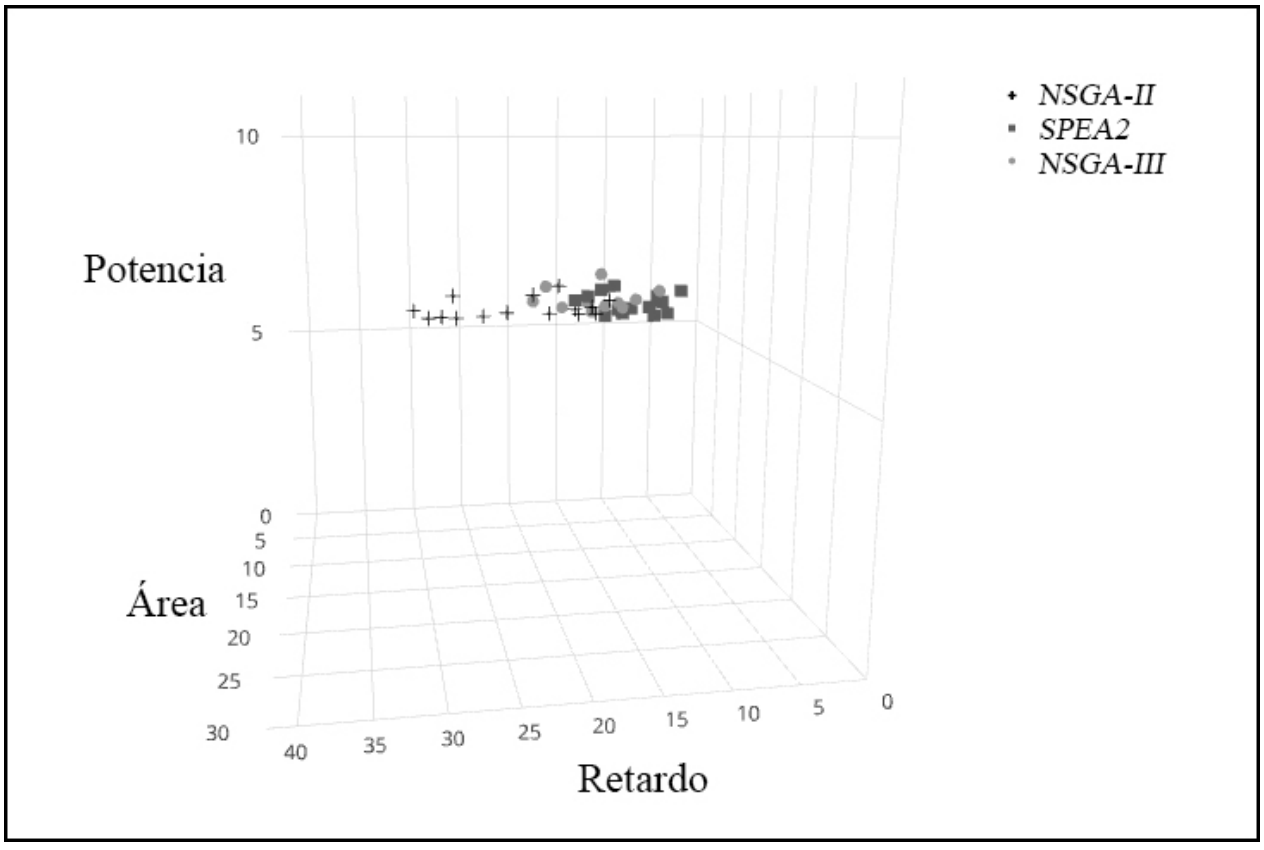

Fig. 8. Espacio objetivo con los FPs logrados por los *MOEAs NSGA-II*, *SPEA2* y *NSGA-III* para el *DFG C1*

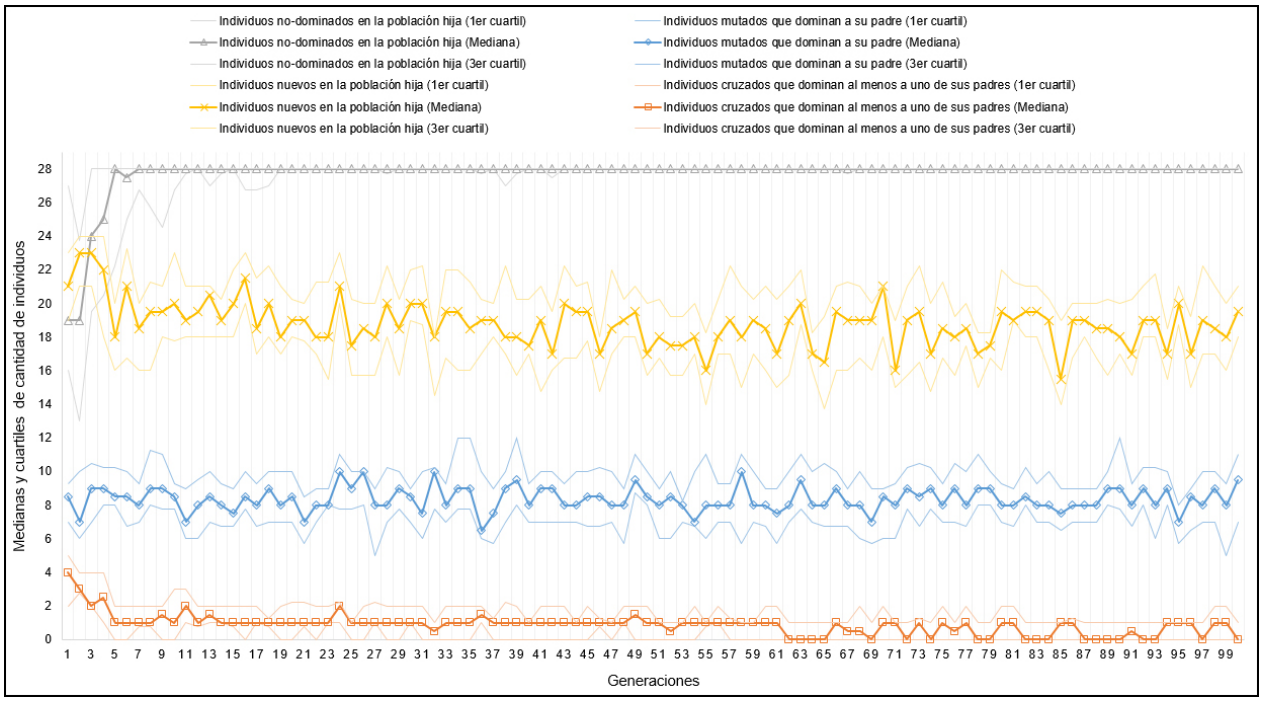

Fig. 9. Gráfica de tendencias de los datos correspondiente al *NSGA-II* para el *DFG C1*

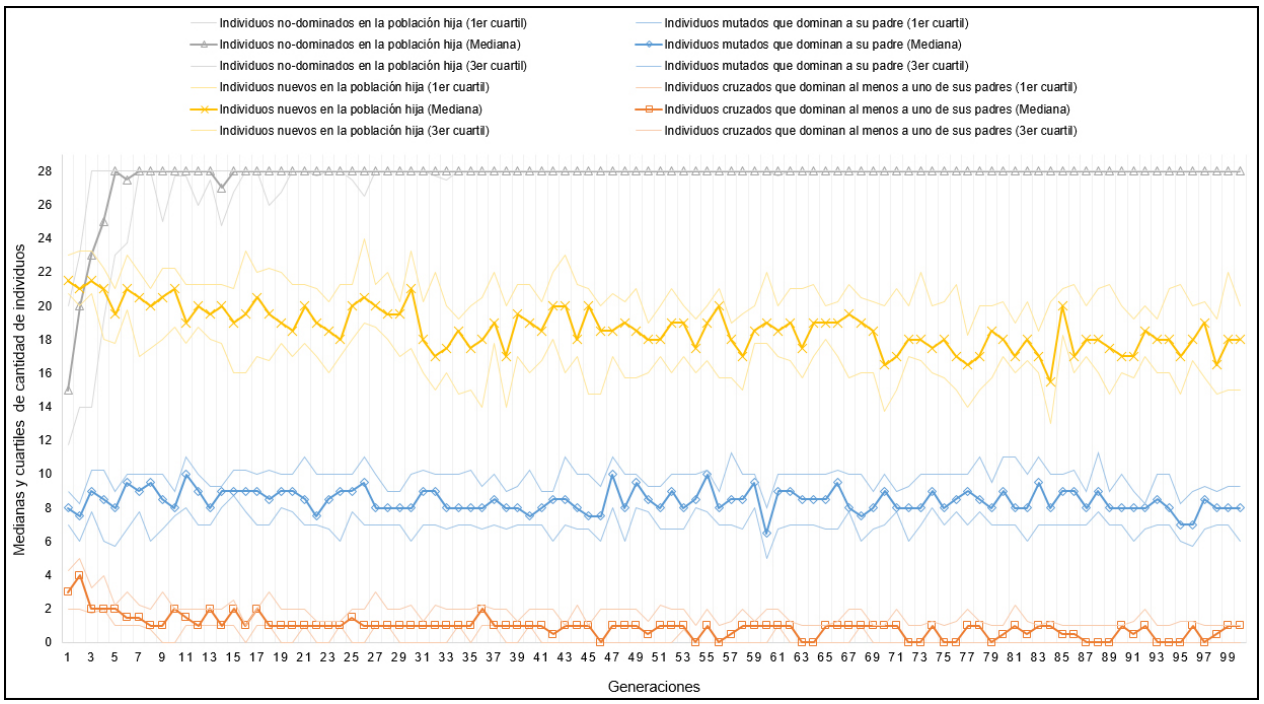

Fig. 10. Gráfica de tendencias de los datos correspondiente al *SPEA2* para el *DFG C1*

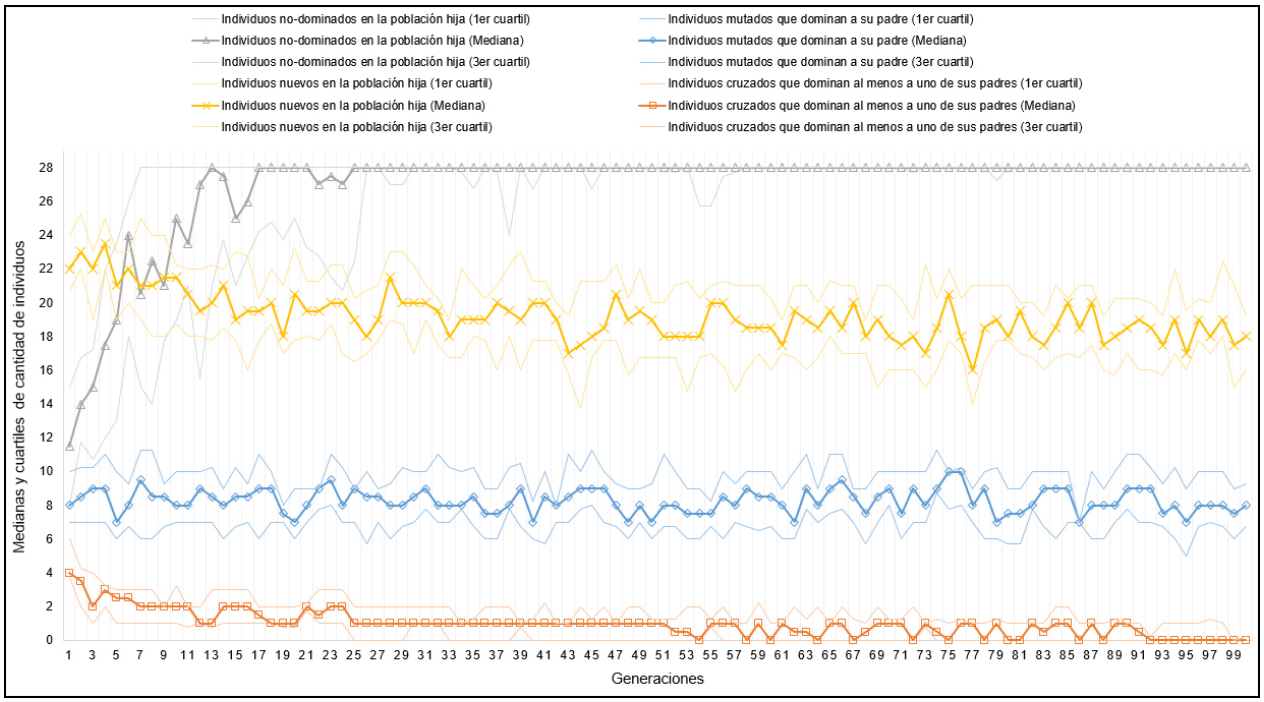

Fig. 11. Gráfica de tendencias de los datos correspondiente al *NSGA-III* para el *DFG C1*

6. Conclusiones y trabajo futuro

Fue demostrada la factibilidad de usar *MOEAs* en la etapa *HLS*. Al realizar la comparación entre los tres *MOEAs*, *NSGA-II*, *SPEA2* y *NSGA-III*, se pudo apreciar que las soluciones obtenidas por *NSGA-III* mejoran a las soluciones obtenidas por *NSGA-II*, pero no a las obtenidas por *SPEA2*.

Sin embargo, según el indicador de calidad Épsilon, *NSGA-III* tiene un mejor desempeño que los otros dos algoritmos. Además, las gráficas con las trazas de datos del proceso evolutivo, ayudaron a descubrir debilidades que pueden ser mejoradas.

Como trabajo futuro, se plantea implementar una técnica que elimine la redundancia en las poblaciones de individuos en el proceso evolutivo, esta redundancia fue detectada en las gráficas de tendencias de los datos que se realizaron. Además, antes de evaluar los individuos en las funciones objetivo, se quiere verificar si ya se había validado un individuo igual, para posteriormente realizar mas generaciones. Pretendemos analizar el artículo *NSGA-III* en [33] y validar una posible adaptación a ese procedimiento. Finalmente, investigar si añadir más funciones objetivo al problema de optimización de HLS es adecuado, y si es así, entonces seguir empleando *NSGA-III*.

Agradecimientos

Los autores agradecen el apoyo del Consejo Nacional de Ciencia y Tecnología y al Tecnológico Nacional de México con el proyecto titulado ”Planificación inteligente de recursos en sistemas reconfigurables”, con No. de registro 6350.17-P.

Referencias

1. **Bader, J. (2010).** *Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods*. Ph.D. thesis, ETH Zurich, Switzerland.
2. **Bhuvaneswari, M. C. (2015).** *Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embedded Systems*. Springer India, 1th edition.

3. **Bleuler, S., Laumanns, M., Thiele, L., & Zitzler, E. (2003).** PISA - A platform and programming language independent interface for search algorithms. *Evolutionary Multi-Criterion Optimization (EMO 2003)*, Lecture Notes in Computer Science, Springer, Berlin, pp. 494–508.
4. **Bleuler, S., Laumanns, M., Thiele, L., & Zitzler, E. (2016).** PISA. Available at `http://www.tik.ee.ethz.ch/sop/pisa/`. Consulted on 05-31-2016.
5. **Ceballos, D., Juan, C., & Ferney, C. (2014).** Prototipo de ascensor en VHDL.
6. **Chen, D., Cong, J., & Fan, Y. (2003).** Low-power high-level synthesis for FPGA architectures. *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pp. 134–139.
7. **Das, I. & Dennis, J. E. (1998).** Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM J. Optimization*, Vol. 8, No. 3, pp. 631–657.
8. **Deb, K. & Jain, H. (2014).** An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, Vol. 18, pp. 577–601.
9. **Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002).** A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 182–197.
10. **Deb, K. & Srinivas, N. (1994).** Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, Vol. 2, pp. 221–248.
11. **Esteban Tlelo-Cuautle, L. G. d. l. F., Antonio de Jesus Quintas-Valles (2016).** Vhdl descriptions for the FPGA implementation of pwl-function-based multi-scroll chaotic oscillators. *PLoS*, Vol. 11, No. 12, pp. 1–32.
12. **Gerez, S. (2000).** *Algorithms for VLSI design automation*. Wiley, USA.
13. **Hansen, M. P. & Jaszkiewicz, A. (1999).** Evaluating the quality of approximations to the non-dominated set. Technical report, Technical University of Denmark.
14. **Harish Ram, D. S., Bhuvaneswari, M. C., & Prabhu, S. S. (2012).** A novel framework for applying multiobjective GA and PSO based approaches for simultaneous area, delay, and power optimization in high level synthesis of datapaths. *VLSI Design journal*, pp. 12.
15. **Harish Ram, D. S., Bhuvaneswari, M. C., & Umadevi, S. (2013).** Improved low power FPGA binding of datapaths from data flow graphs with NSGA II-based schedule selection. *Advances in Electrical and Computer Engineering*, Vol. 13, pp. 85–92.
16. **ITT Tijuana (2017).** MOEAs with differents circuits. *Available at http://http://plots.gruponival.com/*.
17. **Jerraya, A., Kission, P., Ding, H., & Rahmouni, M. (1997).** *Behavioral Synthesis and Component Reuse with VHDL*. Kluwer Academic Publishers.
18. **Kapoulkine, A. (2016).** Pugixml 1.7. Consulted on 05-31-2016.
19. **Krishnan, V. & Katkoori, S. (2006).** A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 3, pp. 213–229.
20. **Kurdahi, F. J. & Parker, A. C. (1987).** REAL: A program for register allocation. *Design Automation, 24th Conference on 1987*, pp. 210–215.
21. **Lee, C., Potkonjak, M., & Mangione-Smith, W. (1997).** Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. *International Symposium on Microarchitecture*, pp. 330–335.
22. **Lee, C., Potkonjak, M., & Mangione-Smith, W. (2016).** Benchmark Mediabench. Consulted on 05-31-2016.
23. **Quan, G. (2016).** Introduction to data flow graphs and their scheduling sources. Available at `http://web.cecs.pdx.edu/~mperkows/temp/JULY/data-flow-graph.pdf`. Consulted on 05-31-2016.
24. **Reyes, D., Dibene, J. C., Maldonado, Y., & Trujillo, L. (2017).** High-level synthesis through metaheuristics and LUTs optimization in FPGA devices. *AI Communications*, Vol. 30, No. 2, pp. 151–168.
25. **Seada, H. & Deb, K. (2014).** U-NSGA-III : A unified evolutionary algorithm for single, multiple, and many-objective optimization. *Lecture Notes in Computer Science*, Vol. 9019, pp. 34–49.
26. **Shahir, F. & Zakaria, B. (2012).** Design and implementation of elevator controller on a FPGA. Technical report, University Technical Malaysia Melaka, Malaysia.

27. **Tapia, J. H. (2013).** *Herramienta de síntesis de alto nivel para sistemas digitales.* Master's thesis, Centro de Investigación y Estudios Avanzados del I.P.N.
28. **Thomas, D. E., Lagnese, E. D., Nestor, J. A., Rajan, J. V., Blackburn, R. L., & Walker, R. A. (1989).** *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publishers, Norwell, MA, USA.
29. **Tsung-Che, C. (2016).** nsga3cpp: A C++ implementation of NSGA-III. Available at `http://web.ntnu.edu.tw/~tcchiang/publications/nsga3cpp/nsga3cpp.htm`. Consulted on 06-19-2016.
30. **Wang, G., Gong, W., DeRenzi, B., & Kastner, R. (2007).** Ant colony optimizations for resource- and timing-constrained operation scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, pp. 1010–1029.
31. **Xilinx (2016).** Xilinx ISE Design Suite. Available at `http://www.xilinx.com/products/design-tools/ise-design-suite.html`. Consulted on 05-31-2016.
32. **Xilinx (2016).** Xilinx Vivado Design Suite. Available at `http://www.xilinx.com/products/design-tools/vivado.html`. Consulted on 05-31-2016.
33. **Yuan, Y., Xu, H., & Wang, B. (2014).** An improved NSGA-III procedure for evolutionary many-objective optimization. *Genetic and Evolutionary Computation Conference (GECCO 2014)*, pp. 661–668.
34. **YWorks (2016).** yEd graph editor. Consulted on 05-31-2016.
35. **Zhang, Q. & Li, H. (2007).** MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 6, pp. 712–731.
36. **Zitzler, E., Laumanns, M., & Thiele, L. (2001).** SPEA2: Improving the strength pareto evolutionary algorithm. Technical report, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
37. **Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & da Fonseca, V. G. (2003).** Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, Vol. 7, pp. 117–132.

Article received on 01/02/2017; accepted on 09/07/2017.
Corresponding author is Darian Reyes Fernández de Bulnes.