Síntesis Booleana Mediante Algoritmos Evolutivos Implementados en una Plataforma HW/SW Abierta

Carlos Iván Camargo
Facultad de Ingeniería
Universidad Nacional de Colombia - Bogotá.
cicamargoba@unal.edu.co

César Pedraza Bonilla
Facultad de Ingeniería de Telecomunicaciones.
Universidad Santo Tomás - Bogotá.
cesarpedraza@usantotomas.edu.co

***Abstract*—Los algoritmos evolutivos presentan una opción a la síntesis combinacional ya que permiten crear estructuras que no pueden obtenerse con las técnicas tradicionales. Este artículo muestra una programación genética paralela (PGP por sus siglas en inglés) para síntesis booleana basado en un cluster de plataformas embebidas abiertas de bajo costo llamadas SIE, basadas en un procesador de 32-bits y una FPGA spartan-3E. Algunas tareas de el PGP han sido aceleradas en hardware y su desempeño ha sido comparado con una implementación HPC, obteniendo una aceleración de hasta 180.**

***Index Terms*—Embedded systems, Evolutionary algorithms, boolean synthesis, cluster.**

I. Introduction.

Uno de los objetivos principales de la síntesis combinatoria consiste en encontrar expresiones booleanas compactas en forma de suma de productos (SOP) con el menor número de variables y términos. El álgebra booleana ofrece un camino para encontrar expresiones compactas, pero este proceso depende de la experiencia del diseñador, por lo que se pueden obtener expresiones no óptimas o inadecuadas. Existen otras técnicas para realizar la síntesis combinatoria tales como mapas de Karnaugh, el algoritmo Quine-McCluskey, el algoritmo Reed-Muller, etc. En términos generales estos algoritmos tienen desventajas como la complejidad exponencial, pérdida del control de restricciones, y múltiples soluciones. Como una alternativa al diseño tradicional de circuitos combinatorios, algunos autores han propuesto técnicas bio-inspiradas basadas en algorítmos genéticos simples (SGAs), algorítmos genéticos de longitud variable (VGAs), programación genético (GP), simulated annealing, algorítmos de colonias de hormigas etc. Estas estrategias permiten crear bloques combinacionales que no pueden ser obtenidos con los métodos tradicionales, y permiten adicionar restricciones tales como retardso, área, consumo de potencia, etc. En estos trabajos se realizaron implementaciones con pocas variables [1] y se obtuvieron estructuras muy básicas.

Con el fín de utilizar programación genética paralela (PGP) se implementó un custer basado en una arquitectura SoC-FPGA para resolver el problema de la síntesis lógica combinatoria. La Unidad de co-procesamiento Fitness (FCU) de cada FPGA ayuda a acelerar la convergencia del algoritmo, al tiempo que proporciona un soporte adecuado para manejar problemas de síntesis de hasta 12 variables. El éxito del sistema se debe principalmente a la capacidad de evaluación de cromosomas utiizando una arquitectura que implementa una LUT virtual en la FPGA, sin utilizar técnicas de reconfiguración parcial las que introducen grandes latencias, obteniendo el valor del fitness para un individuo de forma más rápida.

El software y hardware utilizado para dar solución al problema de la síntesis combinatoria es completamente abierto; se utilizó la plataforma embebida SIE, la cual ha sido diseñada siguiendo la filosofía de la iniciativa *hardware copyleft*, la cual busca crear un movimiento con características similares a la comunidad software libre.

II. De la Programación Genética a la Síntesis Combinatoria

Esta sección describe algunos de los aspectos más importantes del algorítmo evolutivo para el problema de síntesis combinatoria, tales como la representación del cromosoma, la evaluación del fitness y los operadores genéticos: Representación de los Cromosomas. La codificación es la forma de representar un circuito lógico utilizando un arreglo de bits para ser utilizado en el proceso evolutivo [2]. Esta representación debe ser capaz de manejar todas las diferentes soluciones al problema, adicionalmente, los operadores de cruce y mutación deben generar individuos adecuados, y deben cubrir todo el espacio de solución de tal forma que la búsqueda sea realmente aleatoria. Existen diferentes formas de representar circuitos combinatorios en un algorítmo genético: *tree-based 2-D* , Estructuras PLD y cartesiana son algunas de ellas [3] [4] [5]

La representación *2-D tree* es adecuada para la implementación de sistemas paralelos ya que permite la división de los cromosomas para balancear la carga computacional [6]. La figura 1 muestra la estructura seleccionada, cada celda tiene 3 funciones f y 4 variables de entrada v codificadas en binario. Esta representación permite agregar más celdas para representar circuitos más complejos. Debe mencionarse que la longitud del cromosoma debe ser variable debido a que la longitud de la solución al problema de síntesis se desconoce.

A. Función de fitness.

Es muy importante encontrar la función de fitness adecuada ya que esta es la responsable de cuantificar si los individuos cumplen con los requerimientos.

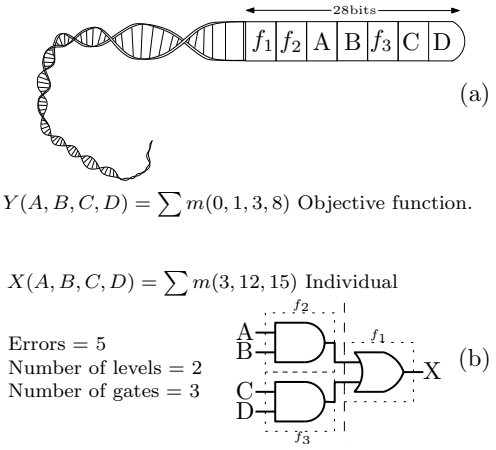

Fig. 1. Representación basada en una estructura de celda.

$$fitness = \omega_1.[\sum_{j=1}^{m}\sum_{i=1}^{n} Y(j,i) - X(j,i)] + \omega_2.P(x) + \omega_3.L(x) \quad (1)$$

En la ecuación 1 se muestra la función de fitness para nuestro GA. Las constantes ω_1, ω_2 y ω_3 se utilizan para establecer los pesos de cada uno de los parámetros que determinarán la función de fitness. La doble sumatoria calcula el número de coincidencias del individuo X para todas las posibles combinaciones con la función objetivo Y. La función $P(X)$ se utiliza para calcular el número de compuertas lógicas de un cromosoma teniendo en cuenta los $introns$ o segmentos de la cadena del genotipo que podrían no tener una función asociada y que no contribuyen al resultado del circuito lógico que el representa. La función $L(X)$ se utiliza para determinar el número de niveles que tiene el circuito, en otras palabras, el número de compuertas que cruza el camino crítico. La constante m representa el número de salidas del circuito y n el número de posibles combinaciones de las entradas.

B. Operadores Genéticos.

El operador *selección* es responsable de identificar los mejores individuos de la población teniendo en cuenta la *explotación* y la *exploración* [6]. La primera permite que sobrevivan los individuos con mejor fitness y se reproduzcan con más frecuencia, y la segunda permite "expadir"el espacio de búsqueda haciendo posible que se encuentren mejores resultados. Por otro lado, el operador *mutación* modifica de forma aleatoria al cromosoma con el fin de aumentar el espacio de búsqueda. Este operador puede modificar: 1) un operador o variable y 2) un segmento en el cromosoma. Ambos son ejecutados de forma aleatoria y con cierta probabilidad. Una probabilidad de mutación variable durante el tiempo de ejecución del algoritmo (evolvable mutation) [7] es más eficiente para sistemas evolutivos. Finalmente, el operador de *cruce* combina dos individuos (con un valor de fitness alto) para obtener otros 2 individuos miembros de la población. En nuestro caso se implementó un sistema con uno o dos puntos de cruce ya que este es más eficiente para sistemas evolutivos [8].

III. Plataforma Hardware copyleft SIE

El proyecto hardware copyleft SIE [9] permite la creación de aplicaciones comerciales bajo la licencia Creative Commons BY - SA [10] permitiendo la distribución y modificación del diseño (incluso para aplicaciones comerciales), con el único requisito de que los productos derivados deben tener la misma licencia y deben dar crédito al autor del trabajo original. Lo que constituye la base de los productos *hardware copyleft*.

A. Hardware copyleft

Al ser inspirado en el movimiento FOSS, los dispositivos *hardware copyleft* comparten la misma filosofía [11], y son su complemento perfecto. Los requisitos para que un dispositivo HW sea reproducible y modificable son: Disponibilidad de los esquemáticos y los archivos de la placa de circuito impreso en un formato que permita el uso de herramientas abiertas; la cadena de herramientas de compilación y depuración para desarrollo de aplicaciones; el código fuente de: el programa que inicializa la plataforma, la herramienta que carga dicho programa en la memoria no volátil, el sistema de archivos y aplicaciones; documentación completa que indique como fué diseñada, construida, como utilizarla, como desarrollar aplicaciones y tutoriales que expliquen el funcionamiento de los diferentes componentes. Adicionalmente, se debe contar con la posibilidad de fabricación y montaje, lo que constituye la principal diferencia entre el software y el hardware libre. Esto contrasta fuertemente con el movimiento de software libre, en donde no se requiere inversión de capital para modificar un proyecto existente. Por esta razón, pueden existir varios niveles de libertad, un proyecto que utilice componentes costosos y de difícil consecución limitará su alcance a un sector determinado.

B. Arquitectura de la plataforma SIE

La plataforma SIE está compuesta por (Figura 2) un System on Chip MIPS (Ingenic JZ4725) de 400 MHz con periféricos que permiten controlar: una memoria NAND de 2GB para almacenamiento de datos y programas, una memoria SDRAM de 32 MB un canal de comunicación serial (UART), memorias micro-SD, un puerto I2C, un LCD a color de 3 pulgadas, 2 entradas y salidas de audio stereo, 2 entradas análogas; una FPGA XC3S500E de Xilinx que proporciona 25 señales de entrada/salida digitales de propósito general (GPIOs) y controla un conversor análogo digital de 8 canales. Existen dos canales de comunicación entre la FPGA y el procesador: uno para controlar el puerto JTAG, lo que permite la configuración de la FPGA y ejecución de pruebas a los circuitos implementados en ella; y otro que proporciona el bus de datos, dirección y control para comunicarse con las tareas HW o periféricos implementados en la FPGA.

SIE proporciona un canal de comunicación y alimentación a través del puerto USB, y es configurado para ser utilizado como una interfaz de red, permitiendo la transferencia de

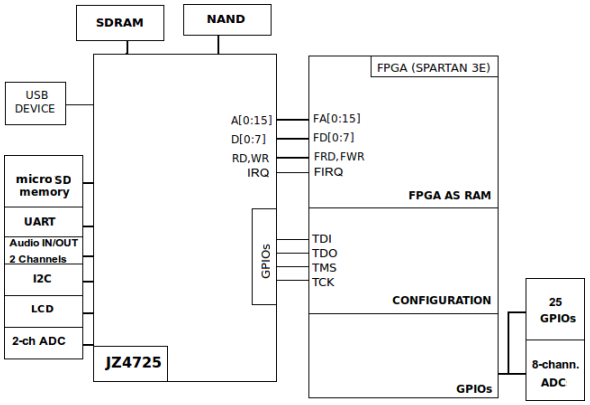

Fig. 2. Estructura de la plataforma de desarrollo SIE

archivos y ejecución de una consola remota utilizando el protocolo *ssh*; este canal de comunicación también se utiliza para programar la memoria NAND no volátil, por lo que para realizar la programación completa de los componentes de la plataforma solo es necesario un cable USB (ver figura 3. Los archivos necesarios para reproducir y modificar esta plataforma pueden ser descargados de la página del proyecto [12].

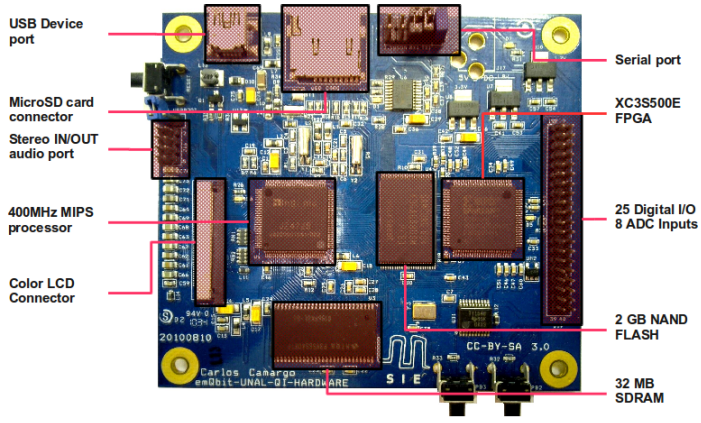

Fig. 3. Plataforma de desarrollo SIE

IV. EA IMPLEMENTATION.

El algoritmo ha sido implementado en dos etapas: En la primera se realizó el desarrollo software y la segunda hace referencia a la implementación hardware para acelerar la convergencia en la plataforma SIE.

A. *Diseño software.*

La evolución natural trabaja con una población no con individuos aislados (excepto para selección y reproducción) algunas operaciones pueden ejecutarse de forma separada, po Natural evolution works with a whole population not with a single individual (except for selection and reproduction) some operations can be done separately, por lo tanto la mayoría de las operaciones del GP son paralelas. La población es dividida en sub-poblaciones que evolucionaran en cada procesador del cluster o arquitetcura paralela. Cuando los sistemas inician, cada procesador crea su sub-población y comienza el proceso de evolución, compuesto por evaluación del fitness, selección, cruce, mutación y reproducción. Una vez se ha alcanzado un determinado número de generaciones, algunos individuos son seleccionados para ser transferidos a otras sub-pobaciones. Un procesador maestro se encarga de recolectar la información de los individuos en transferenica y moverlos a los demás nodos (esclavos), esto aumenta la probabilidad de convergencia del algoritmo. La tasa de intercambio de datos (el número de los mejores individuos para intercambiar aumenta la probabilidad de encontrar una mejor solución) y la frecuencia de migración son parámetros importantes para aumentar el desempeño del algoritmo. Se creó una librería de paso de mensajes propia para implementar las comunicaciones en SIE.

B. *Diseño hardware.*

Una implementación a alto nivel del algorítmo genético determinó que las funciones calculo de fitness y generación de nuevos individuos requieren el mayor esfuerzo computacional (25% y 35% del tiempo de ejecución respectivamente). por lo tanto, estos dos pasos han sido acelerados con el coprocesador implementado en la FPGA. La unidad de cálculo de fitness (FCU ver figura 4) es el elemento hardware diseñado para acelerar este proceso. Este coprocesador se conecta al procesador JZ4725 utilizando una interfaz dedicada.

El cromosoma es enviado desde la memoria SDRAM del procesador a la FCU a través de una interfaz dedicada, cada una de sus células son convertidas a una Look Up Table equivalente. and each of its cells are converted to a equivalent Look Up Table in ROM based translation. A continuación, se calcula el número de minitérminos que no cumplen con la función objetivo. Finalmente, la FCU calcula el valor del fitness incluyendo el número de compuertas y el camino crítico, y los retorna al procesador JZ4725. Con el fín de acelerar la generación de números aleatorios, se implementó un módulo Mersenne-Twister que se comunica con el procesador utilizando la misma interfaz dedicada.

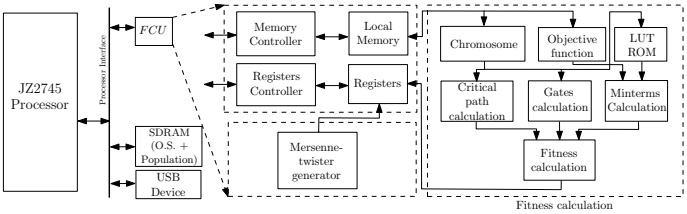

Fig. 4. Estructura de la FCU.

V. EVALUACIÓN.

Para obtener el desempeño y determinar la escalabilidad del algoritmo en SIE, se implementó el mismo algoritmo en un cluster de alto desempeño llamado ALTAMIRA, el cual esta compuesto por 18 servidores BladeCenter, con 256 nodos JS20 (512 procesadores) en una red Myrinet con 1Gbps de

ancho de banda. Por otro lado, el cluster basado en SIE esta conformado por 6 nodos JZ4725-FPGA con la arquitectura descrita anteriormente. La prueba mide el tiempo de respuesta para la version paralelizada del GP en ALTAMIRA y SIE. Se consideraron varios escenarios con diferentes parámetros de configuración: 1) Número de variables de entrada (4, 8 o 12, correspondiente a un comparador de 2, 4, y 6 bits); 2) tamaño de la población (512, 1024 o 2048) y 3) Número de nodos, desde 1 hasta 6 en SIE, y de 2 a 16 o 64 en ALTAMIRA. El primer y segundo parámetro determina el tamaño del problema. El último proporciona información sobre la escalabilidad.

A. Tiempo de Respuesta.

La figura 5 muestra el tiempo de respuesta de ambas plataformas con diferente número de nodos y variables con 1024 individuos durante 100 generaciones. Este experimento demuestra que el tiempo de respuesta de SIE no depende del tamaño del problema; contrario al tiempo de respuesta de ALTAMIRA que tiene una fuerte dependencia con el tamaño del problema, esto debido a que los individuos son evaluados en software.

La figura 6 muestra la respuesta de ambas arquitecturas cuando se varía el número de individuos de la población. Se observa que ambas arquitecturas tienen una fuerte dependencia con el número de individuos y el número de entradas se fija en 12. Esto es debido al aumento de la carga computacional para los 2 clusters. Aún en este escenario SIE mmuestra un desempeño excelente frente a ALTAMITRA.

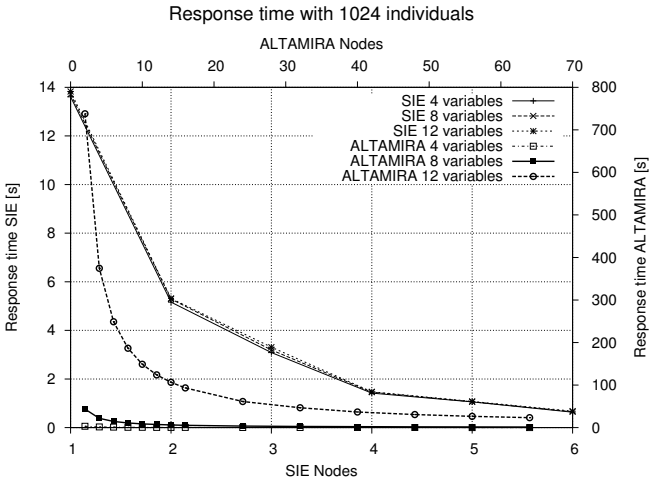

Fig. 5. Tiempo de respuesta en SIE y ALTAMIRA para diferente número de variables.

B. Aceleración.

En la figura 7 se muestra la aceleración de SIE vs ALTAMIRA para diferente número de variables. El excelente desempeño de SIE puede ser explicado porque los individuos han sido probados en un periférico dedicado implementado en hardware (FPGA), obteniendo una combinación de sus tablas de verdad en cada ciclo del reloj del sistema. Por otro lado, los individuos evaluados en software por ALTAMIRA requieren una gran cantidad de ciclos de reloj, lo que hace que su tiempo de respuesta sea cientos de veces mayor que en SIE.

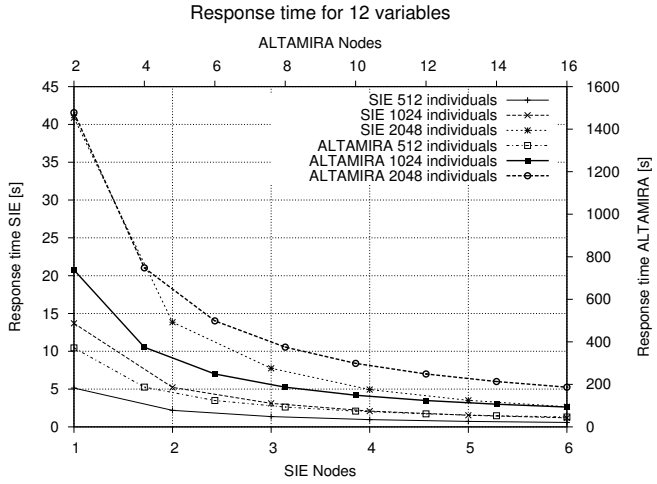

Fig. 6. Tiempo de respuesta en SIE y ALTAMIRA para diferente número de individuos.

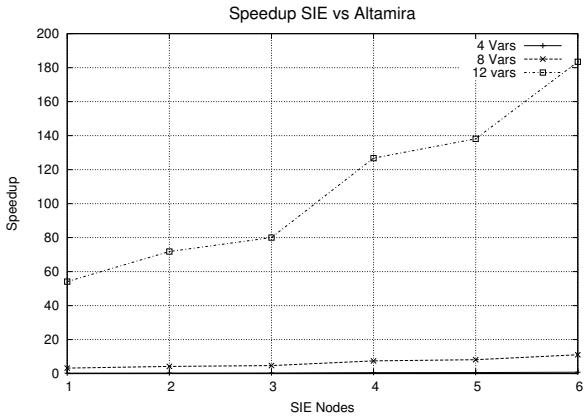

Fig. 7. Aceleración de SIE vs ALTAMIRA comparing 1 SIE node = 2 Altamira nodes.

C. Resulting circuits.

La Figura 8 muestra un ejemplo del circuito resultante para el problema de un comparador de 2 bits. esta estructura es novedosa y diferente a las que originan los algoritmos utilizadoas en la actualidad para realizar síntesis combinatoria.

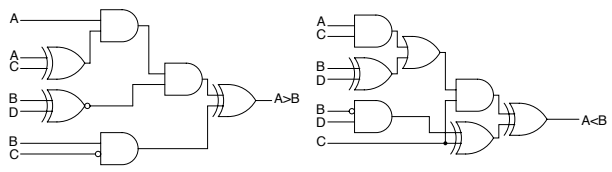

Fig. 8. Ejemplo del circuito resultante para el problema de un comparador de 2 bits (A=MSB number1, C=MSB number2).

D. Consideraciones económicas

SIE proporciona una alternativa económica (70 USD) para la implementación de algoritmos evolutivos; existen en la actualidad un gran número de estudios [13] [14] [15] que utilizan arquitecturas similares basadas en la familia Virtex 2 Pro de Xilinx, para realizar aplicaciones similares, el costo de estas plataformas oscila entre 1000 USD y 3000 USD, debido en gran parte a que la familia Virtex 2 incorpora un procesador Power PC y un circuito lógico programable en el mismo sustrato, por lo que el costo de este dispositivo varía entre 200 y 900 USD. Al separar la FPGA del procesador y proporcionar un canal de comunicación entre ellos SIE puede utilizar dispositivos más económicos (procesador 4 USD, FPGA 10 USD) reduciendo de forma considerable el costo del cluster.

VI. Conclusiones y trabajo futuro.

Este artículo mostró una forma novedosa de evaluar individuos en un algorítmo evolutivo utilizando la plataforma *hardware copyleft* SIE, el desempeño de esta implementación fué comparado con el HPC ALTAMIRA. Para acelerar el proceso de evolución, se implementó un co-procesador para evaluar la función de fitness y generar números aleatorios, mejorando el desempeño para problemas con más de 6 variables de entrada. Las pruebas demostraron que el algorítmo es más efectivo para problemas con entradas de 4-bits y 8-bits. Problemas con 12 señales de entrada tienen un desempeño excelente en la plataforma SIE, pero, debido a que el espacio de búsqueda es muy extenso, el algoritmo presenta problemas de convergencia. Este problema puede resolverse en un trabajo futuro implementando varias FCUs en la FPGA, utilizar más nodos yutilizar otros operadores genéticos acelerados en hardware.

References

[1] D Goldberg and J Holland. Genetic algorithms and machine learning. *Machine Learning*, January 1998.

[2] F. Rothlauf. *Representations for genetic and evolutionay algorithms.* Springer, January 2006.

[3] J Koza, F Bennett, D Andre, and M Keane. Genetic programming iii: darwinian invention and problem solving. *Evolutionary Computation*, January 1999.

[4] T Higuchi, T Niwa, T Tanaka, H Iba, and H de Garis. Evolving hardware with genetic learning: a rst step towards building a darwin machine. *Proc. of the second Int. Conf. on from animals to animats*, January 1992.

[5] J Miller and P Thomson. Aspects of digital evolution: Evolvability and architecture. *Lecture Notes in Computer Science*, January 1998.

[6] Q. Yu, C. Chen, and C. Pan. Parallel genetic algoritms on programmable graphics hardware. *Lecture notes in computer scienc*, January 2006.

[7] R Krohling, Y Zhou, and A Tyrrell. Evolving fpga-based robot controllers using an evolutionary algorithm. *Proc. I Int. Conf. on Articial Immune Syst*, January 2002.

[8] J Miller and P Thomson. Aspects of digital evolution: Evolvability and architecture. *Lecture Notes in Computer Science*, January 1998.

[9] C. Camargo, W. Spraul, and A. Wang. Proyecto SAKC. *URL:http://en.qi-hardware.com/wiki/SAKC*.

[10] Creative Commons. Licencias Creative Commons. URL: http://creativecommons.org/licenses., 2004.

[11] R. Stallman. Philosophy of the GNU project. URL: http://www.gnu.org/philosophy/, 2007.

[12] C. Camargo. Proyecto SIE. URL:http://en.qi-hardware.com/wiki/Sie.

[13] R. Oreifej, R. Al-haddad, H. Tan, and R. Demara. LAYERED APPROACH TO INTRINSIC EVOLVABLE HARDWARE USING DIRECT BITSTREAM MANIPULATION OF VIRTEX II PRO DEVICES. *International Conference on Field Programmable Logic and Applications*, 2007.

[14] J. Jeon and P. Rhee. An Evolvable Hardware System Under Varying Illumination Environment. *Advances in Natural Computation*.

[15] Z. Vasicek and L. Sekanina. An evolvable hardware system in Xilinx Virtex II Pro FPGA. *Int. J. Innovative Computing and Applications*, 2007.