Sumador evolutivo de 2 bits tolerante a fallos

Juan Lanchares, Oscar Garnica, J. Ignacio Hidalgo, and Jose L. Risco-Martín

Grupo de Arquitecturas Paralelas y Algoritmos Bioinspirados,
Universidad Complutense de Madrid, España
http://bioinspired.dacya.ucm.es

Resumen En este artículo se presenta un SoC (del inglés System on Chip) evolutivo que converge a un sumador de dos bits tolerante a fallos. El SoC se compone de un procesador Microblaze sobre el que se ejecuta un algoritmo genético que realiza la evolución, una FPGA (del inglés Field Programmable Gate Array) virtual implementada sobre la FPGA real cuya misión es realizar la evaluación de los circuitos durante la evolución e implementar el circuito final obtenido, y un módulo de evaluación encargado de generar los vectores de entrenamiento y los valores esperados para calcular la función de coste de cada circuito. Se han realizado tres pruebas diferentes. En la primera de ellas se permite que el circuito evolucione pudiendo seleccionar 8 funciones diferentes en cada Bloque Lógico Configurable o CLB (de sus siglas en inglés Configurable Logic Block) virtual. En la segunda, sólo se van a utilizar en los CLBs las funciones necesarias para implementar un sumador tradicional, lo que sería equivalente a permitir exclusivamente la evolución de las conexiones. Por último se han simulado fallos en la FPGA virtual, en concreto se ha provocado el fallo del 16 % de los CLB virtuales. Como resultados se ha obtenido que permitiendo únicamente la evolución de las conexiones se alcanza la convergencia mucho más rápidamente y que si se producen fallos el sistema es capaz de evitar las zonas dañadas de la placa alcanzando la convergencia al circuito esperado.

Keywords: Hardware Evolutivo, Tolerancia a Fallos, Estrategias Evolutivas

1. Introducción

El hardware evolutivo (EHW de sus siglas en inglés, Evolvable Hardware) es un hardware capaz de modificar su estructura interna autónomamente, es decir, sin intervención del diseñador, para alcanzar el circuito objetivo. La modificación de la estructura se realiza en tiempo de ejecución y puede deberse a dos factores: las condiciones de trabajo del entorno han cambiado (variaciones de temperatura, picos de potencia, radiaciones, cambios de localización, ajuste fino de parámetros de algoritmo, etc.) o se ha producido un fallo en alguno de los componentes del sistema. En ambos casos la modificación se produce o bien porque el hardware ha dejado de funcionar correctamente o bien porque la calidad de los resultados generados por el hardware ha disminuido. Es importante resaltar que las modificaciones se realizan sin que previamente el diseñador tenga

conocimiento de cómo van a evolucionar las condiciones internas y externas de trabajo [6] [3] [2].

El EHW se inspira en el principio darwiniano de evolución de forma que un conjunto de posibles soluciones (circuitos) compiten entre sí para poder llegar a la fase de reproducción en la que las características de dos de ellos se combinan para obtener un circuito probablemente mejor. Este nuevo circuito formará parte de una nueva población más apta en promedio que la anterior [8][4] [1].

Una de las principales ventajas del EHW se deriva del hecho de que sólo tiene en cuenta el valor de la función de coste para autodiseñarse, y esto le dota de una gran adaptabilidad en tiempo de ejecución. Esta característica puede ayudar a solucionar problemas de tolerancias a fallos y de adaptación a entornos cambiantes, problemas que aparecen a menudo en aplicaciones críticas como por ejemplo las misiones espaciales o problemas en medicina.

Entre los ejemplos de utilidad resaltan la tolerancia a fallos cuando el sistema está sometido a radiaciones extremas o a temperaturas que puedan dañar los dispositivos lógico-programables, y el diseño de estructuras in situ en cuyo caso el sistema se puede diseñar directamente en el entorno en el que va a trabajar. Esta tecnología se ha utilizado para diseñar una antena incorporada en la misión Marte Odyssey, lanzada en el 2001 y que en la actualidad continúa en funcionamiento y enviando imágenes [5].

En este artículo se presenta un SoC evolutivo que converge a un sumador de dos bits tolerante a fallos. Los resultados experimentales demuestran que permitiendo únicamente la evolución de las conexiones se alcanza la convergencia mucho más rápidamente y que si se producen fallos el sistema es capaz de evitar las zonas dañadas de la placa alcanzando la convergencia al circuito esperado.

El resto del trabajo está organizado de la siguiente manera. En la sección 2 se hace una breve introducción al Hardware Evolutivo. En la sección 3 se explica la propuesta y en la 4 se dan los detalles del sistema hardware, es decir del System On Chip que implementa el EHW. En la sección 5 se comentan los resultados experimentales y en la sección 6 se exponen las conclusiones obtenidas en este trabajo y las posibles líneas de trabajo futuro.

2. Hardware Evolutivo

Como ya hemos mencionado, el hardware evolutivo se basa en los principios darwinianos de la evolución. Por lo tanto, trabaja con un conjunto de posibles soluciones (representaciones de circuitos) o poblaciones de individuos, de manera que los circuitos compiten entre sí para poder llegar a la fase de reproducción en la que las características de dos de ellos se combinan para obtener un circuito probablemente mejor que formará parte de una nueva población más apta en promedio que la anterior.

Para conseguir esto no se trabaja directamente con circuitos sino con representaciones de los mismos. Un circuito se representa mediante la cadena de ceros y unos necesaria para implementarlo en una FPGA. A este código se le

llama cromosoma. Un algoritmo evolutivo se encarga de llevar a cabo la evolución de la población de cromosomas. La evolución se inicia evaluando cada uno de los cromosomas de la población. La evaluación se utiliza para saber lo que se aproxima un circuito al objetivo. Para realizarla el cromosoma se utiliza para programar la FPGA (ver Figura 2. Una vez programada se envía al circuito un vector de test. Las salidas proporcionadas por el circuito se comparan con las salidas esperadas. Contabilizando el número de coincidencias entre las salidas esperadas y las salidas generadas se tiene una medida de lo que se aproxima el circuito que está siendo evaluado al circuito objetivo [2].

Después de evaluados todos los cromosomas se genera la siguiente población utilizando operadores de cruce y mutación. El operador de cruce consiste en seleccionar dos cromosomas e intercambiar su información entre sí mientras que el operador de mutación consiste en seleccionar un cromosoma y aplicar cambios aleatorios a los bits que componen el cromosoma. Ambos operadores se aplican con una determinada probabilidad. Este proceso de evolución se repite hasta que se obtiene un circuito que cumple las especificaciones deseadas. Finalmente, el circuito obtenido se implementa sobre una FPGA. Este proceso puede verse modificado para implementar otro tipo de algoritmo evolutivo, como por ejemplo el que se utiliza en este artículo, una Estrategia Evolutiva (EE). La EE aplica solamente el operador de mutación para generar nuevos individuos.

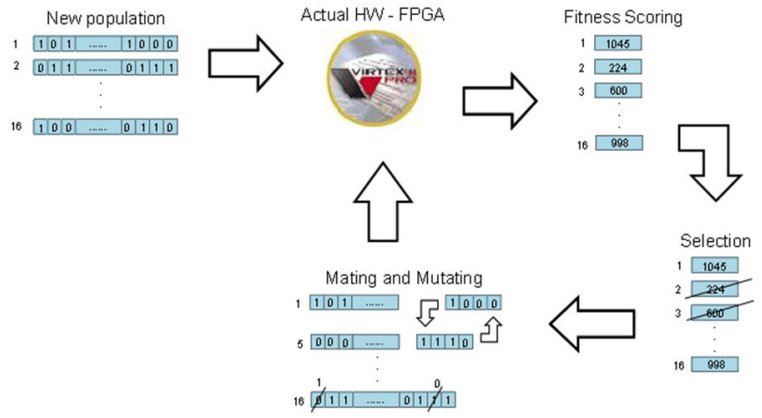

Figura 1. Sistema de Hardware Evolutivo.

Como se puede deducir de lo expuesto, este tipo de hardware se diferencia del clásico en dos aspectos fundamentales: la adaptación autónoma en tiempo de ejecución y el modo en que se implementa el circuito. Centrándonos en la primera diferencia, el hardware clásico no puede modificar su estructura sin una intervención -directa o indirecta- del diseñador, y sin haber tenido conocimien-

to y haber previsto determinados cambios en el entorno. En el caso del EHW esta adaptación se realiza en tiempo de ejecución sin intervención externa y sin necesidad de evaluar previamente los posibles cambios que se produzcan en el entorno ya que es precisamente el proceso evolutivo descrito quien lo realiza. En cuanto al modo de implementación del circuito, en el hardware clásico hay que fijar qué se quiere hacer mediante un conjunto de especificaciones y el cómo se hace viene prefijado por un flujo de diseño conocido. No es así en el caso del EHW, donde sólo hay que indicar qué se quiere hacer mediante la función de coste, pero no es necesario aportar ningún conocimiento de cómo se va a realizar la implementación puesto que ésta la realiza el algoritmo evolutivo [7].

El mayor problema del EHW es la escalabilidad. Hasta el momento se ha conseguido implementar circuitos de extrema sencillez si se les compara con el grado de complejidad existente en la actualidad. Esto se debe a varios factores que se explican a continuación. El primero de ellos es la propia escalabilidad de la representación. Los circuitos complejos necesitan muchas puertas y conexiones para implementarse. Para poder representar estos circuitos se necesitan cromosomas extremadamente grandes que a su vez dan lugar a espacios de soluciones enormes. Por ejemplo, para implementar un circuito de unas cien puertas lógicas se podría necesitar, dependiendo de la representación, un cromosoma de hasta dos mil bits.

Otro problema asociado a la escalabilidad es la forma en la que se debe representar la función de coste. Aparentemente la forma más sencilla de fijar el objetivo de convergencia es usar las tablas de verdad que describen el comportamiento del sistema, pero esto método queda invalidado en el momento que el circuito alcanza cierta complejidad. En estos casos se deben utilizar métodos alternativos siendo uno de los más extendidos utilizar unos vectores de entrenamiento más unos vectores de test que nos permiten en cierta medida cubrir todos los posibles comportamientos del circuito.

En este sentido es en el que nosotros hacemos nuestra propuesta. La idea no es evolucionar todo un circuito desde cero en el que la obtención de la función de coste es costosísima, y el espacio de soluciones enorme lo que hace la convergencia prácticamente imposible. Nuestra propuesta aplicada a la tolerancia a fallos es partir de un circuito que se sabe que funciona, cuyos módulos funcionales son conocidos de antemano , de manera que se realiza un proceso de ubicación y rutado evolutivo, lo que disminuiría enormemente la complejidad del cromosoma, de los operadores genéticos y por lo tanto se reduciría enormemente el espacio de soluciones , consiguiendo una convergencia en mejores tiempos. En definitiva la idea es incluir información del problema a la evolución para facilitarla.

3. La Propuesta

Cuando surgió la idea del hardware evolutivo el principal objetivo de la evolución no sólo era converger a un circuito que cumpliera los funcionalidad fijada, sino que de alguna manera este circuito mejorara la implementación que se pudiera hacer mediante métodos clásicos de diseño, bien fuera porque en la adapta-

ción era capaz de hacerse tolerante a fallos, bien porque el diseño final mejoraba en área, consumo o rendimiento al clásico. Para conseguir este objetivo la idea general era dar la máxima flexibilidad posible al diseño y esto se conseguía en parte fijando un amplio repertorio de funciones que se podían implementar en los CLBs de la FPGA. Esto da lugar a cromosomas de una elevada complejidad. Por ejemplo si un CLB puede implementar 16 funciones diferentes, el cromosoma necesita 4 bits para seleccionar cada una de las posibles funciones del CLB. Por lo tanto, si el circuito se compone de 1000 CLBs, el cromosoma necesita 4000 genes sólo para representar las funciones. A esto habría que añadirle los genes necesarios para representar las conexiones entre los CLBs. como resultado el tamaño final del cromosoma hace prácticamente inviable la convergencia del circuito en unos tiempos aceptables, sobre todo si estamos hablando de sistemas autónomos on-line.

Para solucionar el problema, algunas propuestas usan bloques lógicos más complejos, es decir, en lugar de utilizar puertas lógicas básicas, se utilizan bloques funcionales más complejos, de manera que se reduce la complejidad del cromosomas y por lo tanto del espacio de búsqueda. El problema de esta aproximación es que conceptualmente sigue la misma línea que la anterior. La diversidad de bloques que se pueden seleccionar es muy amplia con lo que la complejidad y amplitud del espacio de búsqueda sigue siendo demasiado grande. El problema de ambas aproximaciones es que pretenden realizar la evolución desde cero, sin incluir ningún tipo de información al sistema evolutivo del circuito final a obtener a parte de la proporcionada por la función de coste. Esta aproximación puede ser correcta cuando la evolución se aplica al diseño evolutivo, cuyo objetivo es encontrar un circuito que mejore en algún aspecto los obtenidos mediante un diseño clásico y en el que se tiene a disposición del diseñador toda la infraestructura de un laboratorio. Pero no es una aproximación válida cuando lo que se pretende es implementar un sistema evolutivo autónomo en el que ni se dispone de las infraestructuras de un laboratorio, ni el tiempo de evolución puede ser indeterminado puesto que la adaptación a estímulos externos o externos se tiene que llevar a cabo en tiempos aceptables.

En los sistemas evolutivos autónomos la idea es que el circuito evolucione o bien para adaptarse a cambios en las condiciones de trabajo externas, o bien a condiciones de trabajo internas (fallos hardware). En estos casos se parte de un diseño en concreto cuya funcionalidad se conoce de antemano, y que está diseñada y probada, por lo tanto no es necesario que el circuito evolucione desde cero, es suficiente que el circuito evolucione para poder adaptarse a las necesidades.

Esto lleva a una importante conclusión: en el caso de los sistemas evolutivos autónomos, el sistema debería incluir información sobre el circuito a evolucionar, lo que se puede traducir en que las funciones a utilizar deberían ser exclusivamente aquellas que de antemano sabemos va a utilizar el circuito. De esta manera se reduce enormemente el espacio de soluciones dando lugar a una convergencia más rápida. Esta propuesta debería estudiarse en más profundidad para el caso de la adaptación a variaciones externas puesto que en estos casos la adaptación

puede no afectar tanto a la funcionalidad del circuito como a otros factores, como ralentización del circuito por cambios bruscos de temperatura, parece totalmente adecuada cuando se utiliza con el objetivo de hacer los circuitos tolerantes a fallos. Suponiendo que un conjunto de CLBs y de conexiones de una FPGA falla el objetivo sería recolocar los bloques en otros CLBs activos y reconexionar el circuito hasta alcanzar la funcionalidad deseada evitando las zonas dañadas. En definitiva ya no se buscan nuevas funciones sino que se recolocan las funciones que sabemos implementan la funcionalidad del circuito.

El trabajo que presentamos en este artículo es una primera aproximación a la propuesta, cuyo objetivo es observar las diferencia en la convergencia entre la aproximación sin conocimiento del problema y con conocimiento del problema. El circuito que vamos a utilizar es un sumador de dos bits, circuito sencillo y en el que evidentemente no vamos a poder comprobar la diferencia de escalabilidad entre las dos aproximaciones. Pero, como decimos, este trabajo es sólo una primera aproximación y esperamos en un futuro poder enfrentarnos a problemas de más envergadura.

4. El sistema

La plataforma general de EHW se puede ver como un sistema empotrado formado por los siguientes componentes: un microprocesador sobre el que se ejecutará el algoritmo evolutivo, la FPGA virtual encargada de implementar los circuitos cuyo código se envía desde el microprocesador, y el IP evaluador encargado de evaluar la validez de los circuitos cargados en la FPGAV, el sistema de memoria, buses de sistema y periféricos. A continuación se explican en detalle cada uno de los módulos. La Figura 2 representa un esquema general del sistema de hardware evolutivo.

4.1. El microprocesador

El objetivo del procesador es ejecutar el algoritmo genético que lleva a cabo la evolución. Desde el procesador se envía a la FPGA virtual el cromosoma que configura el circuito a evaluar. Una vez evaluado, el circuito se envía al procesador el valor de la función de coste obtenida para que el algoritmo pueda seguir su evolución.

4.2. La FPGA virtual o FPGAv

Este módulo es el encargado de implementar los circuitos cuyos cromosomas se envían desde el procesador. Esta FPGAV es una matriz de nodos programables conectados mediante una red de interconexión que se implementa sobre la FPGA real. Esta FPGAV debe estar conectada con el IP evaluador de manera que pueda recibir el vector de test (función de coste) y enviar las salidas generadas, y con el procesador para poder recibir el cromosoma que describe el circuito que se quiere implementar.

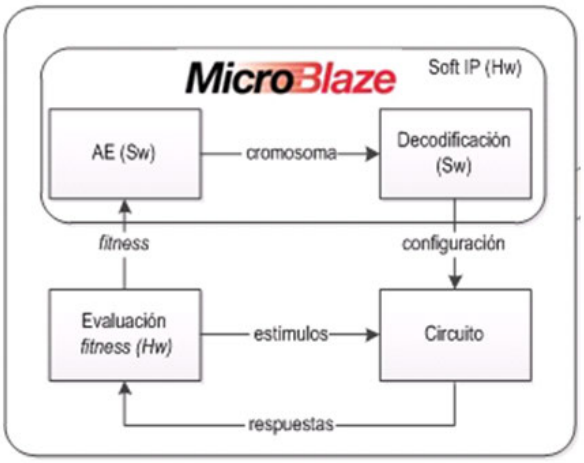

Figura 2. Sistema de Hardware Evolutivo.

4.3. IP evaluador

El IP evaluador es el encargado de realizar la evaluación de los circuitos cuyo cromosoma se envía desde el procesador del sistema. Tendrá que ser capaz de generar un vector de test y un vector de salidas esperadas. Para realizar la evaluación se envía el vector de test a la FPGAv y se comparan las salidas generadas con las salidas esperadas contabilizando los aciertos obtenidos.. Este módulo también debe incluir toda la lógica necesaria para intercambiar información con el procesador. El IP evaluador es de vital importancia porque a la postre es el que va a guiar la evolución de los circuitos.

5. Resultados experimentales

Con el objetivo de validar la propuesta se ha estudiado la convergencia de un sumador de 2 bit en tres casos diferentes. En el primero, el algoritmo evolutivo puede seleccionar 8 funciones diferentes para cada CLB del circuito programable, en el segundo sólo se pueden seleccionar las 3 funciones lógicas necesarias para implementar el sumador, y en el tercer experimento se estudia la convergencia de este último circuito simulando fallos en el circuito programable. En todos los casos se ha utilizado una Estrategia Evolutiva (λ+1) para diferentes valores de λ. Este tipo de algoritmo evolutivo genera λ soluciones nuevas mediante mutación, a partir de la mejor solución (+1) de la generación anterior. El padre no se conserva de una generación a otra

Para implementar el hardware evolutivo se ha utilizado el sistema de desarrollo XUP Virtex-II Pro. Debido a que es un sistema anticuado han surgido ciertos problemas que ha habido que solucionar. Por ejemplo la FPGA virtual no entraba en tiempos por lo que se tuvo que segmentar en tres etapas. Por

Parámetro 1	Valor
Elitismo	1
Tamaño de la población	λ
Longitud del cromosoma	216
Probabilidad de cruce	0 %
Probabilidad de mutación de funciones	100 %
Probabilidad de mutación de conexiones	100 %
Límite de generaciones	20.000

Cuadro 1. Parámetros de las Estrategias Evolutivas utilizadas

otro lado la cantidad de memoria BRAM del sistema no es muy elevada lo que ha obligado a utilizar poblaciones de individuos muy pequeñas comparadas con el espacio total de soluciones lo que en cierta medida ralentiza la convergencia del circuito. Los circuitos obtenidos se han simulado con *Modelsim* para asegurarnos que efectivamente el circuito obtenido implementa el circuito buscado. A continuación se explica con más detalle cada uno de los experimentos

5.1. Convergencia sin información del circuito

En esta primera prueba el sistema sólo va a recibir información sobre QUÉ es lo que queremos diseñar y en ningún caso se le da información de CÓMO queremos diseñarlo. Es decir, la convergencia se va a guiar exclusivamente por la función de coste del sumador plasmada mediante las tablas de verdad del circuito. como no se da ninguna información sobre como implementar el circuito vamos a permitir que el número de funciones que puede seleccionar el algoritmo genético sea lo suficientemente amplia y flexible. con este objetivo se han fijado las 8 funciones lógicas que aparecen en la tabla 2. como se puede observar en estas funciones aparecen 3 conjuntos completos lo que nos asegura que se va a poder implementar cualquier circuito lógico. También se añade una función "dejar pasar" cuyo objetivo es utilizar el CLB como una conexión dado que el número de conexiones de la FPGA virtual es muy reducido.

función	código cromosoma		
AND	000	NOR	100
OR	001	XNOR	101
XOR	010	NOT	110
NAND	011	DEJAR-PASAR	111

Cuadro 2. Funciones lógicas disponibles en un CLB virtual

5.2. Convergencia con información del circuito

En este experimento añadimos conocimiento del problema al sistema. Sabiendo que un sumador se puede implementar usando exclusivamente puertas

lógicas XOR, AND y, NOT sólo vamos a permitir seleccionar esas puertas en la estrategia evolutiva. Al hacer esto estamos limitando la complejidad y el tamaño del espacio de búsqueda y por lo tanto facilitando la convergencia. La limitación de funciones a usar en el diseño no se realiza modificando la FPGA virtual sino que se realiza modificando las funciones del algoritmo genético que generan la población inicial aleatoria y la mutación de funciones. En ambos casos sólo se permiten las funciones xor, and, not y dejar pasar.

5.3. Estudio de la tolerancia a fallos

En el tercer estudio que realizamos comprobamos que el circuito puede evolucionar esquivando zonas dañadas de la fgpa para alcanzar un circuito final que cumple los requisitos. Los fallos los vamos a simular mediante software, forzando al algoritmoevolutivo a comportarse de una determinada manera. En concreto vamos a simular los fallos de 4 CLBs de la FPGA virtual. Para ello, suponemos que esas celdas no pueden generar ninguna función útil y por lo tanto ningún CLB de la FPGA puede conectar su entrada a la salida de los CLBs dañados. Esto traducido al algoritmo significa que determinadas conexiones de la FPGA virtual no van a poder seleccionarse en las diferentes fases del algoritmo. Para estudiar en más en profundidad el comportamiento bajo fallos hemos realizado tres pruebas diferentes:

- Desde el inicio de la evolución se suponen los CLBs dañados
- Los CLBs se dañan en mitad de la evolución
- Los CLBs se dañan una vez que se ha alcanzado un circuito válido

5.4. Discusión

La tabla 3 expone los resultados experimentales para el caso de convergencia con información del circuito, pero los tiempos de convergencia y los datos relativos a individuos son similares en todos los casos. No creemos de mayor utilidad un análisis más exhaustivo de los datos, puesto que nuestro objetivo es probar que la metodología funciona. La tabla recoge los valores de ejecución de la estrategia evolutiva para distintos valores de λ. Los resultados recomiendan un valor de entre 15 y 20 para λ. Los resultados experimentales demuestran que el sistema evoluciona y consigue converger a pesar de estar trabajando sobre un hardware defectuoso, en el cual hay CLBs dañados, por los que el sistema no puede acceder en su camino hacia la evolución. El sistema detecta si hay fallo al principio, y comienza la evolución disponiendo de menos CLBs. Obviamente, el número de generaciones necesarias para obtener la convergencia, es superior (en torno al 20 %) con respecto al caso de un hardware libre de fallos, ya que dispone de menos CLBs teniendo así mayores restricciones.

6. Conclusiones

En este artículo se ha presentado un sumador evolutivo de 2 bits tolerante a fallos. El sistema haciendo uso del hardware evolutivo, es capaz de obtener

λ	25 ind.	20 ind.	15 ind.	12 ind.
Tiempo por individuo (s)	1,55	1,30	1,29	1,28
Tiempo en 500 generaciones (s)	38,66	26,09	19,33	15,33
Generaciones (avg)	4416	4939	6779	13106
Tiempo medio (s)	341	257	262	401

Cuadro 3. Resultados experimentales

un circuito sumador implementado sobre una FPGA. Para realizar la evolución se ha implementado una Estrategia Evolutiva $(\lambda + 1)$, que converge en tiempos aceptables. Para comprobar la tolerancia a fallos, se han simulado errores bajo tres escenarios diferentes, en todos ellos el sistema es capaz de encontrar una nueva implementación que evite las zonas dañadas. Se trata de un trabajo preeliminar por lo que las posibles lineas de trabajo y sus aplicaciones son muy amplias, desde el diseño de nuevos circuitos de mayor complejidad a la mejora del algoritmo evolutivo y ajuste de sus parámetros.

Referencias

1. H. Garis. Evolvable hardware genetic programming of a darwin machine. In R. Albrecht, C. Reeves, and N. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 441–449. Springer Vienna, 1993.
2. G. W. Greenwood and A. M. Tyrrell. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley, John and Sons, 2006.
3. T. Higuchi, Y. Liu, and X. Yao, editors. *Evolvable Hardware*. Genetic and Evolutionary Computation. Springer US, 2006.
4. T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de fs, and T. Furuya. Evolving hardware with genetic learning: a first step towards building a darwin machine. In *Proceedings of the second international conference on simulation of adaptive behavior*, pages 417–424. MIT Press, 1993.
5. J. Lohn, G. Hornby, and D. Linden. An evolved antenna for deployment on nasa,Äôs space technology 5 mission. In U.-M. O,ÄôReilly, T. Yu, R. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, volume 8 of *Genetic Programming*, pages 301–315. Springer US, 2005.
6. L. Sekanina. Evolutionary design space exploration for median circuits. In G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, C. R. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, and G. Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, pages 240–249, Coimbra, Portugal, 5-7 Apr. 2004. Springer Verlag.
7. L. Sekanina, T. Martinek, and Z. Gajda. Extrinsic and intrinsic evolution of multifunctional combinational. In G. G. Yen, L. Wang, P. Bonissone, and S. M. Lucas, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9676–9683, Vancouver, 6-21 July 2006. IEEE Press.
8. A. Thompson. *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer US, 1998.