GeneticFPGA: A Java-Based Tool for Evolving Stable Circuits

Delon Levi and Steven A. Guccione

Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124

ABSTRACT

GeneticFPGA is a Java-based tool for evolving digital circuits on Xilinx XC4000EX™ and XC4000XL™ devices. Unlike other FPGA architectures popular with Evolutionary Hardware researchers, the XC4000 series architectures cannot accept arbitrary configuration data. Only a small subset of configuration bit patterns will produce operational circuits; other configuration bit patterns produce circuits which are unreliable and may even permanently damage the FPGA device. *GeneticFPGA* uses novel software techniques to produce legal circuit configurations for these devices, permitting experimentation with evolvable hardware on the larger, faster, more mainstream devices. In addition, these techniques have led to methods for evolving circuits which are neither temperature, voltage, nor silicon dependent. An 8-bit counter and several digital frequency dividers have been successfully evolved using this approach. *GeneticFPGA* uses Xilinx's *JBits*™ interface to control the generation of bitstream configuration data and the *XHWIF* portable hardware interface to communicate with a variety of commercially available FPGA-based hardware. *GeneticFPGA*, *JBits*, and *XHWIF* are currently being ported to the Xilinx Virtex™ family of devices, which will provide greatly increased reconfiguration speed and circuit density.

Keywords: Genetic, evolvable hardware, FPGA, Java

1. INTRODUCTION

Genetic Algorithms (GA) are computational processes for finding the optimal or a near optimal solution for a problem. Borrowing ideas from genetics and Darwinian survival of the fittest principles, the algorithm mutates and recombines solutions to progressively construct better performing solutions. They have been applied to a wide range of problems, from optimizing the design of mechanical objects to evolving computer programs. Recently they have been applied to evolving fully synthesized, placed, and routed FPGA-based circuits. By custom tailoring the algorithm they can be used to create a wide range of circuits.

Thompson's[1] ground breaking work in Evolvable Hardware produced circuits that performed simple pattern recognition and robot control. However the circuits were highly asynchronous which caused them to be very sensitive to temperature and voltage changes. The circuits also only worked on the FPGAs they were evolved on, the same circuit would not operate when downloaded onto a different device. Furthermore, to avoid contention the XC6200 was used, for it was thought that only a device in which the hardware guaranteed non-contentious circuits could execute randomly generated circuits.

GeneticFPGA is a software toolkit for evolving FPGA-based circuits. It works with mainstream FPGAs, the Xilinx XC4000EX/XL family, on numerous commercial-off-the-shelf reconfigurable computing boards. It uses novel software techniques to avoid producing contentious circuits, and it also produces stable digital circuits that reliably operate on multiple devices.

2. THE ALGORITHM FLOW

At the start, a population of individuals is created, where each individual is represented by chromosome. The chromosomes are initiated with 1's and 0's using a random number generator. Like their biological counterparts, the codes on the chromosome act as genes that specify the configuration of a particular entity. For example, one gene may indicate the

Further Author Information
D.L E-Mail: Delon.Levi@xilinx.com
S.A.G. E-Mail: Steven.Guccione@xilinx.com

configuration of a look-up-table and another may indicate the configuration of a flip-flop. Using the JBits bitstream interface, codes on the chromosome are translated into circuits in a FPGA bitstream. Test circuits can also be inserted into the bitstream. Once all of the circuits have been inserted into a bitstream, it is downloaded to a FPGA on a board using the XHWIF interface. The circuit is executed on the FPGA for a number of clock cycles, and the output data is readback and scored. After all the individuals are scored, the best ones are genetically recombined and mutated to form a new generation of individuals which are then processed in the same way. This procedure continues until a suitable individual is discovered. By discarding the poor performers and recombining the characteristics of the strong performers, the individuals in each generation get better at solving the expected task.

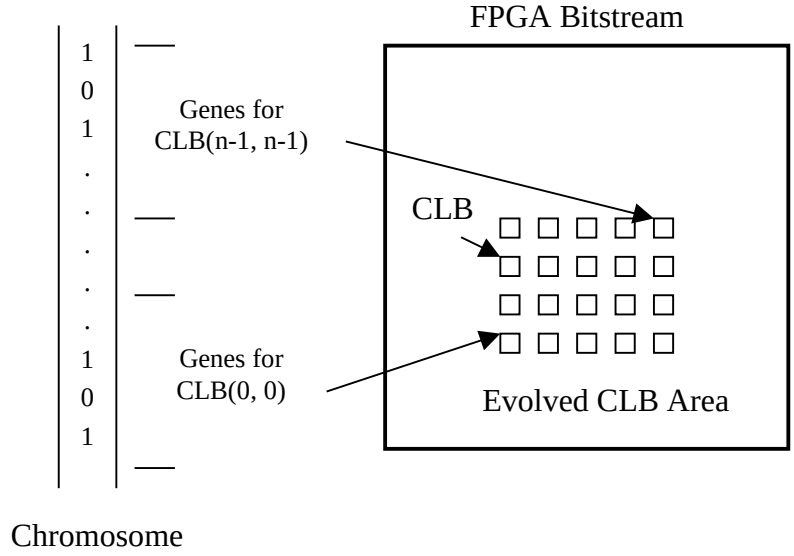

Figure 1 - CLB Gene Mapping

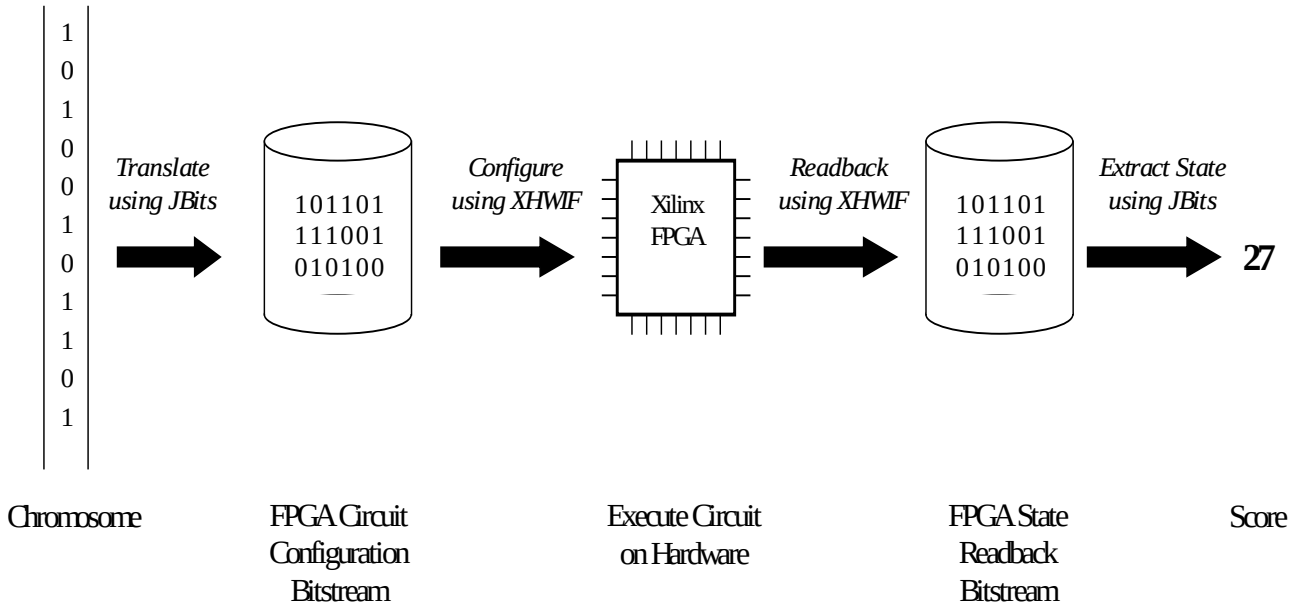

Figure 2 - Algorithm Flow

3. AVOIDING CONTENTION

Contention occurs when two or more outputs drive the same wire. If the wires drive opposing voltages, which inevitably happens during operation, then the device can be damaged or even destroyed from the resulting high currents. Although mainstream FPGAs have always allowed contention, the place and route software prevented it by generating only one or no drivers for a particular wire. Similarly with Evolvable Hardware, the software has to guarantee that one or no outputs drive a single wire. However, since the circuits are randomly mutated and recombined, special care has to be taken to avoid contention.

Two schemes can be used to avoid contention. In one scheme, a bitstream is check before download, and if contention is found the wires are rerouted until contention is eliminated. Although a valid method, this can be very compute intensive, which can be a significant detractor to the overall methodology that tries to process as many individuals as quickly as possible. Another scheme is to simply prevent contention by construction. *GeneticFPGA* uses the second method by only allowing genes to manipulate wires with single drivers. The wires and resources that can assume contentious configurations are turned off, and because they have no genetic representation they are unable to evolve connections to the circuit. By performing the mutations and recombinations on the chromosome rather than directly on the bitstream, and by guaranteeing a non-contentious mapping between chromosome and bitstream, the software guarantees generation of non-contentious circuits.

4. STABLE DIGITAL CIRCUITS

As previously mentioned, Thompson's circuits were highly sensitive to temperature, voltage, and the particular piece of silicon the bitstream was evolved on. In addition, the circuits displayed analog behavior despite being implemented on digital devices. This implies that in order for the circuit to operate, the FPGA has to be maintained at narrow temperature ranges and supply voltages. It also implies that every circuit has to be custom evolved for each chip, which precludes the circuits from being sold to a mass market.

All these effects were caused because the circuits are highly asynchronous. Asynchronous circuits create very small high frequency pulses. If employed in a controlled manner by a design engineer, they can produce stable and predictable results. But when employed in a widespread and random manner, they produce unusually small high frequency signals that actually violate the timing requirements of the hardware. Figure 3 graphically shows the flip-flop states of two XC4028EX FPGAs loaded with the same asynchronous evolved circuits. The same circuit clearly produces different results. Even when downloaded onto the same device the results differ.

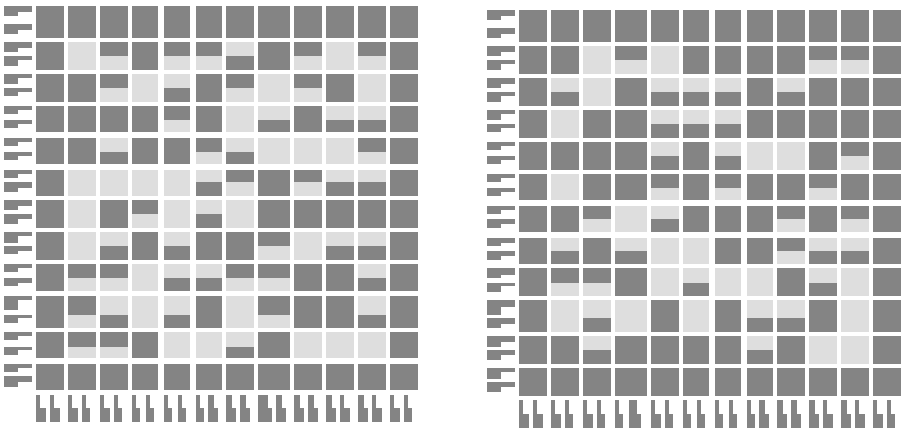

Figure 3 - Flip Flop States of Asynchronous Circuit on 2 FPGAs

The solution that *GeneticFPGA* uses is to only allow the evolutionary process to produce synchronous circuits. Synchronous circuits have the property that all pulses are larger in duration and smaller in frequency than the clock signal. So if the clock source runs at a controlled frequency, then the circuits behave in a reliable, predictable, and digital manner. To eliminate

asynchronous circuits, the genes prevent the flip flops from evolving into latch mode and asynchronous feedback loops from forming, and prevents the flip flop clock inputs from being driven by anything except the singular clock source. Figure 4 graphically show the flip-flop states of two XC4028EX FPGAs loaded with the same synchronous evolved circuit. The same circuit produces identical results, even when downloaded onto the same FPGA.

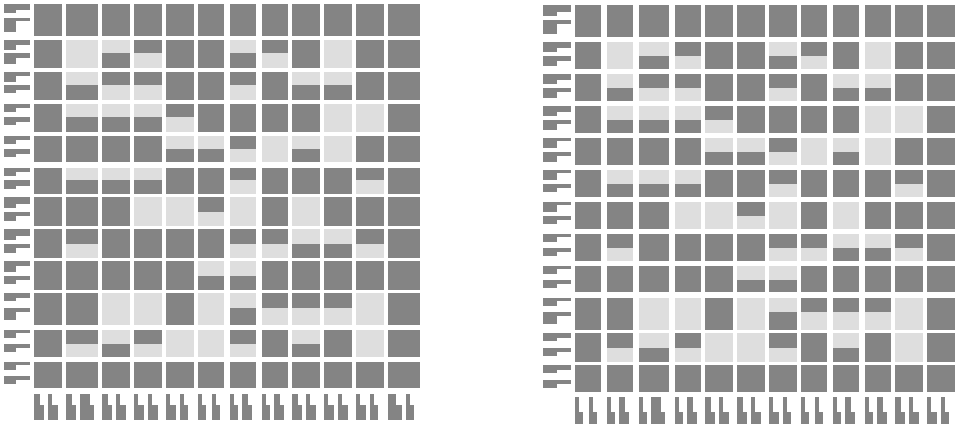

Figure 4 - Flip Flop States of Synchronous Circuit on 2 FPGAs

5. INSERTING TEST CIRCUITS

In order to score a circuit's performance, it has to be driven with test data and its outputs have to be stored. The circuit can then be measured by how well it produces expected outputs given certain inputs. The test data can come from other devices, like an ASIC, CPU, or even another FPGA designed traditionally with VHDL. SRAMs can be used for storage. However, this approach requires additional and dedicated hardware. Also, it ties the software to a particular board. Instead, *GeneticFPGA* allows the user to insert test and storage circuits directly into the bitstream so that it is embedded with the evolved circuit inside the FPGA. With the test circuits embedded in the FPGA, the software can be used on any XC4000EX/XL based reconfigurable computing board.

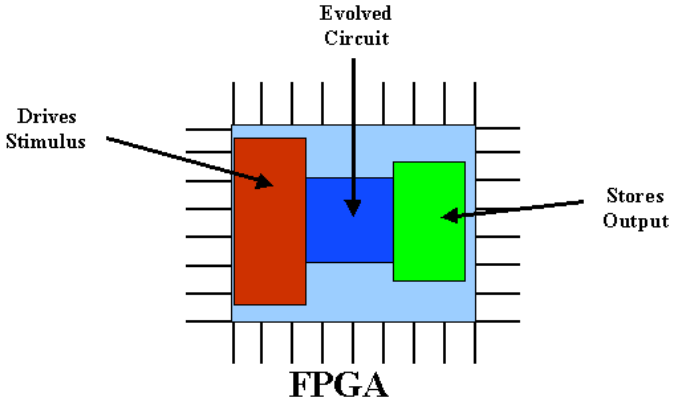

Figure 5 - Embedding Test Circuits with Evolved Circuit

The test circuits are constructed in the bitstream using JBits and JBits-based cores. The user can construct the test circuits with the test data pre-defined in the circuits, perhaps inside registers or block RAM. The output storage can also be constructed using registers or block RAM. The test circuits can also be constructed with an instruction and data interface so that they can receive test instructions and test data from software run within *GeneticFPGA*. This allows the test data to be dynamically constructed while the circuits are executing, and even for intermediate output to be read back to influence the

construction of the test vectors. Furthermore, since the test circuits are constructed during run-time, every individual can have embedded with it different test and storage circuits. Considering the relatively slow readback speeds of XC4000EX/XL FPGAs, if intermediate data is read to generate test vectors, it is probably best to write the data through the IOBs.

6. CONCURRENT EVOLUTION

GeneticFPGA has the capability to evolve multiple populations at the same time. Each population is threaded, so the microprocessor can execute their genetic operations concurrently, and each population uses a different FPGA, so the circuit executions can also occur concurrently. Many reconfigurable computing boards have multiple FPGAs to enable this capability. Although, the populations are not limited to a single board, each population can use FPGAs on separate boards. In fact, because of the remote interface on *XHWIF*, the FPGAs can even be on boards installed in remote computers. So multiple populations can evolve at the same time using FPGAs on multiple locally installed boards and on multiple remote boards.

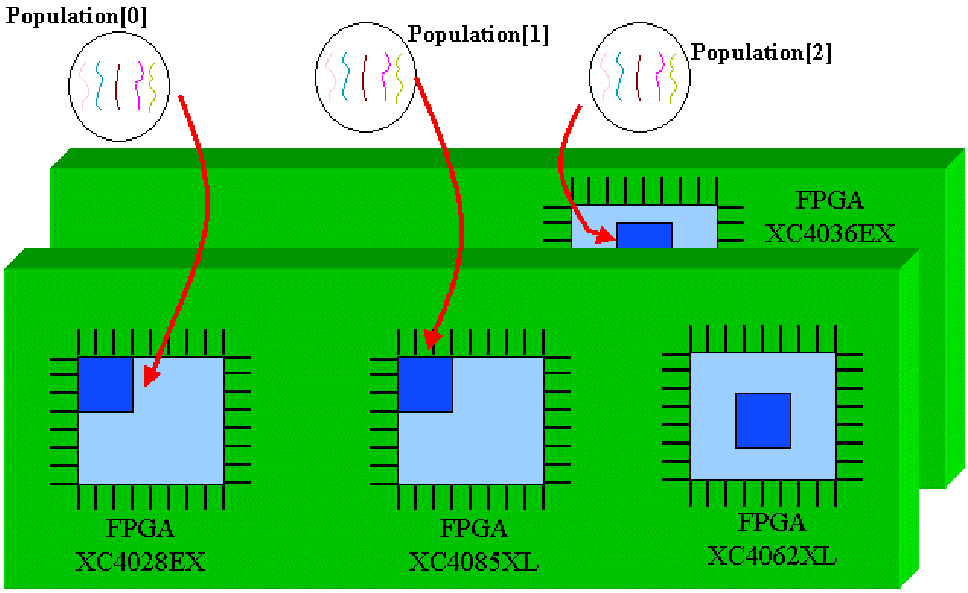

Figure 6 - Concurrent Evolution on Multiple FPGAs

Each population evolves independently, there aren't yet any cross-breading capabilities. The populations can try to converge on the same goal, or any of the populations can converge on its own goal. For example, one population can try to evolve a counter while another tries to evolve a pattern recognizer. Because each population follows a different path through the search space, pointing multiple populations at the same goal increases the odds of finding a good solution. This can be useful for problems with difficult-to-find solutions.

This feature is useful if the software is executed on a multi-processor computer, or if algorithm is such that most of the computation time is spent executing circuits on the FPGAs. If the microprocessor is bogged down performing the genetic operations, executing them concurrently essentially gives the same performance as executing them sequentially.

7. RESULTS

To prove that a toolkit could evolve functional circuits from completely random starting points, several experiments were attempted. In one set of experiments, 4-bit and 8-bit one-hot counters were successfully evolved. Several frequency dividers were also successfully evolved. Each success required many runs to correctly set population and genetic recombination parameters and to allow the system to hit on the solution.

Another set of experiments was attempted to see if the system could evolve a pattern recognizer. Afterall, one of the significant promises for this technology is that it may one day allow a machine to on-the-fly learn to recognize patterns in real-time data. In the experiments, an odd number of bits were driven to the evolved circuit using shift registers. The objective was for the evolved circuit to register a 1 at the output if the input contained more 1's than 0's, or a 0 otherwise. The outputs were stored in another shift register. Many attempts were made with different input sizes and evolved circuit sizes, but none produced successful results. Only 50% of the outputs were correctly matched, with the correlation never changing significantly. This indicates that the evolved circuits randomly generated 1s and 0s on the outputs.

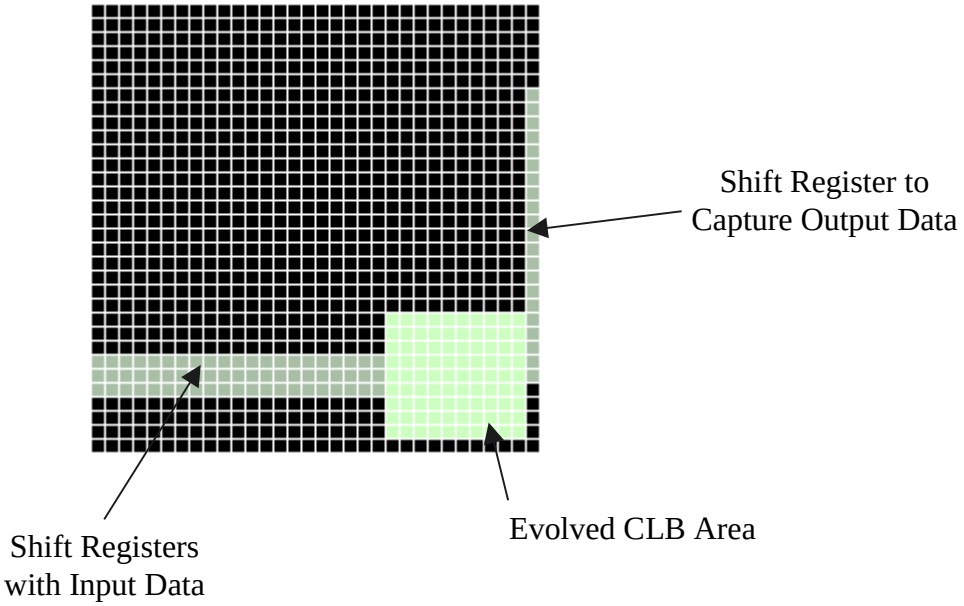

Figure 7 - Core Display for Pattern Recognizer

There are two problems with the pattern recognizing circuits. With a digital synchronous pattern matcher it is unclear how many clock cycles are required between submitting the input and sampling the output. Several values were tried, but it would seem to be that given the various evolved structures, the system would have to determine the correct value for each circuit. The other problem is connectivity. In avoiding contention, only a small percentage of the available routing was available for use by the evolved circuits. The limited routing made it very difficult for the evolutionary process to establish a computational path between the inputs and the outputs. Clearly, more routing needs to be made available to perform complex computations.

8. CONCLUSIONS AND FUTURE WORK

The XC6200's capability to guarantee non-contention served as a useful starting point for Evolvable Hardware research. But since it is no longer manufactured and is in limited supply, the field needs to move to more mainstream FPGA architectures. Rather than relying on the hardware for non-contention, *GeneticFPGA* uses software to avoid contention. Similarly, by using software to impose synchronous constraints, *GeneticFPGA* produces digital circuits that behave reliably on multiple FPGAs without stringent voltage and temperature requirements.

In the future, *GeneticFPGA* will be ported to Virtex FPGAs. Virtex FPGAs are larger, faster, and like the XC4000EX/XL FPGAs they are mainstream and widely available. In addition, Virtex has significantly faster configuration and readback speeds, which should allow the software to use larger populations to help it span the search space for successful solutions. The Virtex based solution will also attempt to solve the timing and routing problems.

9. REFERENCES

1. A. Thompson, "Silicon Evolution", in Proceedings of Genetic Programming 1996 (GP96), J.R. Koza et al. (Eds.), pp. 444-452, MIT Press, Cambridge, Ma, 1996.

2. A. Thompson, "On the Automatic Design of Robust Electronics Through Artificial Evolution", in Proc. 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), M. Sipper, D. Mange & A. Péres-Uribe (Eds.), pp13-24, Springer-Verlag,1998.

3. T.C. Fogarty, J.F. Miller, and P. Thomson., "Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs", in Soft Computing in Engineering Design and Manufacturing, P.K. Chawdhry, R. Roy, and E.K. Pant (Eds.), pp. 299-305. Springer-Verlag, 1998.

4. M. Korkin, H. de Garis, F. Gers, and H. Hemmi, "CMB (CAM-Brain Machine): A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time", in Proceedings of Genetic Programming 1997 (GP97), J.R. Koza et al. (Eds)., pp. 498-503, Morgan Kaufman Publishers, San Francisco, Ca, 1997.

5. J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, Ma, 1992.

6. L. Davis, Handbook of Genetic Algorithms, International Thomson Computer Press, London, 1996.

7. D. E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley Publishing Company, Inc., Reading, Ma, 1989.

8. S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware", in Confiurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.

9. D. Levi and S. A. Guccione, "BoardScope: A Debug Tool for Reconfigurable Systems", in Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.

10. Xilinx, Inc., The Programmable Logic Data Book, 1998.

11. N. Sitkoff, M. Wazlowski, A. Smith, and H. Silverman, "Implementing a Genetic Algorithm on a Parallel Custom Computing Machine", in IEEE Symposium on FPGAs for Custom Computing Machines, Peter Athanas and Kenneth L. Pocek (Ed.), pp. 180-187, IEEE Computer Society Press, Los Alamitos, CA, April 1995.

12. P. Graham and B. Nelson, "Genetic Algorithms in Software and in Hardware - A Performance Analysis of Workstations and Custom Computing Machine Implementations", in IEEE Symposium on FPGAs for Custom Computing Machines, Kenneth L. Pocek and Jeffrey Arnold (Ed.), pp. 216-225, IEEE Computer Society Press, Los Alamitos, CA, April 1996.

13. I. M. Bland and G. M. Megson, "The Systolic Array Genetic Algorithm, an Example of Systolic Arrays as a Reconfigurable Design Methodology", in IEEE Symposium on FPGAs for Custom Computing Machines, Kenneth L. Pocek and Jeffrey Arnold (Eds.), pp. 260-261, IEEE Computer Society Press, Los Alamitos, CA, April 1998.

14. P. Graham and B. Nelson, "A Hardware Genetic Algorithm for the Travelling Salesman Problem on SPLASH 2" in Field-Programmable Logic and Applications, Will Moore and Wayne Luk (Eds.), pp. 352-361, Springer-Verlag, Berlin, August/September 1995. Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications, FPL 1995. Lecture Notes in Computer Science 975.

15. M. Mitchell, J. P. Crutchfield, and R. Das, "Evolving Cellular Automata to Perform Computations", in Handbook of Evolutionary Computation, T. Back, D. Fogel, and Z. Michelwicz (Eds.), Oxford University Press, Oxford, 1997.

16. T. Toffoli and N. Margolus, Cellular Automata Machines: A New Environment for Modeling, MIT Press, Cambridge, Ma, 1987.

17. R. Dawkins, The Blind Watchmaker, W. W. Norton & Company, New York , NY, 1996.

18. J. H. Holland, Hidden Order, Addison-Wesley Publishing Company, Reading, Ma, 1995.

Patents Pending