Intrinsic Evolvable Hardware for Combinational Synthesis Based on SoC+FPGA and GPU Platforms

Artificial Life, Robotics, Evolvable
Hardware

ABSTRACT

Evolutionary algorithms are another option for combinational synthesis because they allow for the generation of hardware structures that cannot be obtained with other techniques. This paper shows a parallel genetic programming (PGP) boolean synthesis implementation on a low cost cluster of an embedded platform called SIE, based on a 32-bit processor and a Spartan-3 FPGA. Some tasks of the PGP have been accelerated in hardware and results have been compared with GPU and HPC implementations, resulting in speedup values up to approximately 2 and 180 respectively.

Keywords

Embedded systems, Evolutionary algorithms, boolean synthesis, cluster.

1. INTRODUCTION.

One of the main gools in combinational synthesis consists of finding compact boolean expressions in the sum of products (SOP) form with the smallest possible number of variables and terms. Boolean algebra offers a way to find compact expressions, but this process depends on the designer's experience, resulting in non-optimal or inadequate expressions. There are other techniques available for combinational synthesis such as the Karnaugh maps, the Quine-McCluskey algorithm, the Reed-Muller algorithm. In general terms, these algorithms have disadvantages such as exponential complexity, lack of restrictions management, and multiple solutions. As an alternative to the traditional design of combinational circuits, some authors have proposed bio-inspired techniques based on simple genetic algorithms (SGAs), variable-length genetic algorithms (VGAs), tree-based genetic programming (GP), simulated annealing, ant colony algorithms and others. NOTA DE NIÑO: REFERENCIAS !!!!!

NOTA DE CAIN: Se debe cambiar el enfoque del paper y hacer que la parte principal sea la evoución intrínseca, adicionar varios papers donde se muestre este tipo de evolución, y como resuleven el problema.

These strategies allow to create combinational blocks that cannot be obtained with traditional methods, and to add some restrictions to the design such as delay, area. These designs have a very low limited number of variables [3] and they are mainly oriented to obtain a few basic structures.

In order to use parallel genetic programming (PGP) an FPGA cluster-based architecture to solve the combinational synthesis problem on-chip it has been developed. The fitness coprocessor unit (FCU) on each FPGA helps to accelerate the convergence of the algorithm, as well as provide an appropriate support for 12-variable synthesis problems. The success of the system is mainly due to the capability of evaluating a chromosome in the FPGA through a virtual LUT-oriented architecture without using high-latency partial reconfiguration techniques, and determining the fitness value for an individual faster than other references.

Due to that the capability of GP to run on massive parallel architectures, the algorithm results on boolean synthesis were compared to a High Performance Cluster (HPC) and a graphics processing unit (GPU) implementations.

2. GENETIC PROGRAMMING IN COMBINATIONAL SYNTHESIS

This section describes some of the most important aspects of the proposed evolutionary algorithm for the combinational synthesis problem, such as chromosome representation, the fitness problem and the genetic operators.

2.1 Chromosome representation.

The codification is the way a logic circuit is represented using a bit array in order to be used in the evolution process [4]. This representation must be able to represent all the different solutions to the problem, also the crossover and mutation operators should not generate infeasuble individuals, and it must cover all the solution space. There are different ways of representing combinational hardware for a genetic algorithm: tree-based representation in 2-D, PLD-like structures and cartesian representation are some of them [6] [12] [7].

The 2-D tree representation is appropriate for implementing parallel systems because it allows splitting the chromosomes for balancing the computational load [9]. Figure 1 shows the selected cell-based structure representation. Each cell has 3 functions f and 4 input variables v coded in binary. This representation allows the addition of more cells to represent larger circuits. It must be mentioned that the chromosome length has to be variable because the length of the solution to the synthesis problem is unknown a priori.

2.2 Fitness function.

Finding the appropriate fitness function is important because it is responsible for quantifying the way a chromosome or individual meets or does not meet the requirements.

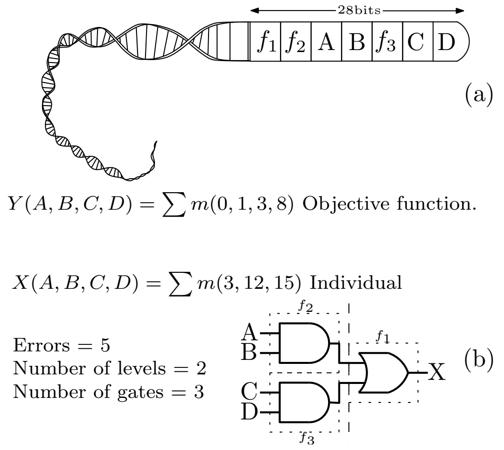
28bits
f_1 f_2 A B f_3 C D
(a)
$Y(A,B,C,D) = \sum m(0,1,3,8)$ Objective function.
$X(A,B,C,D) = \sum m(3,12,15)$ Individual
Errors = 5
Number of levels = 2
Number of gates = 3
f_2
f_1
f_3
A
B
C
D
X
(b)

Figure 1: Cell-based structure representation.

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

In equation 1 the fitness function for the proposed GA is shown. Constants ω_1, ω_2 and ω_3 are used for establishing the weights of each of the parameters that will determine the fitness function. The double-summation term calculates the number of matches of the individual X for all the possible combinations at the output with the target functions Y; the $P(X)$ function is used for calculating the number of logic gates of a chromosome taking into account some of the *introns* or segments of the genotype string that will not have any associated function and that do not contribute to the result of the logic circuit that they represent. The function $L(X)$ is used for determining the number of levels the circuit has, or in other words the number of gates that the critical path crosses. The m constant is the number of outputs in the circuit and n the number of possible combinations of inputs in the circuit.

2.3 Genetic operators.

The selection operator is responsible for identifying the best individuals of the population taking into account the exploitation and the exploration [9]. The first one allows the individuals with better fitness to survive and reproduce more often, and the second one means searching in more areas and making it possible to find better results. In the other hand, the mutation operator modifies the chromosome randomly in order to increase the search space. It can change: 1) an operator or variable and 2) a segment in the chromosome. Both are executed randomly with a certain probability. A variable mutating probability during the execution of the algorithm (evolvable mutation) [10] is more effective for evolvable systems. Finally, the crossover operator combines two selected individuals for obtaining two additional individuals to add to the population. A crossover system with one or two crossover points randomly selected has been implemented because it is more efficient for evolvable systems [8].

3. COPYLEFT SIE PLATFORM.

The hardware project copyleft SIE [1] created by our team work, is composed of a custom embeddded platform and a software development kit based on Linux operating system; allowing the generation of commercial applications under the Creative Commons BY-SA license [2], which allows the distribution and modificacion of the design (including commercial applications), with only the requirements that the derived products be under the same license and that the proper credit be given to the author of the original work. This is what defines the base of the *hardware copyleft* products.

3.1 SIE Platform.

The platform is composed by (Figure 2) a System-On-a-Chip processor and a FPGA. The SoC is a MIPS processor (Ingenic JZ4725) running at 400MHz with peripherals that allows to control: a 2GB NAND memory for file storage, a 32MB SDRAM, a serial communication channel (UART), micro-SD memories, I2C port, 3-inch color LCD display, 2 input and 1 output audio stereo interfaces and 2 analog inputs. The XC3S500E Xilinx FPGA gives 25 GPIOs and controls an 8 channel analog-to-digital converter. There are two communication channels between the FPGA and the processor, the first one is a JTAG that allows the FPGA configuration and the execution of tests to the implemented circuits. The data, address and control buses are part of the second channel, which is used to exchange information between the processor and the peripherals implemented inside the FPGA.

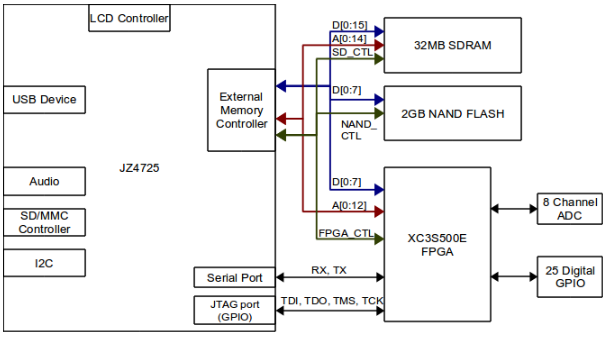

Figure 2: SIE development platform structure

SIE provides power connection through a USB port, which is also configured as a network interface. This allows the file transfer and the execution of a remote console using the *ssh* protocol. This communication channel can also be used to configure the non-volatile NAND memory, which simplifies the whole configuration of the platform through the USB port.

SIE provides an economical alternative (70 USD per node) for implementing evolutionary algorithms; Vasicek et al. [11], Higuchi et al. [5] and Sekanina[13] use architectures based on Xilinx Virtex 2 Pro family for implementing similar applications, closed to 1000 USD y 3000 USD per node. By separating the FPGA and the processor and providing a communication channel between them, SIE can use cheaper devices (4 USD for the processor, 10 USD for the FPGA) reducing significantly the cluster's costs. On the other hand, selected GPU for the present work is about 200 USD.

4. GA IMPLEMENTATION

The GA was implemented in two stages: the first one is about software development and its parallelization on HPC and GPUs, and the second one refers to a hardware implementation to speed it up on the SIE plataform.

4.1 Parallel software design.

4.1.1 HPC

Natural evolution works with a whole population not with a single individual (except for selection and reproduction) some operations can be done separately, therefore almost all operations in a GP are implicitly parallel. Using the island approach, the population is divided into sub-populations that will evolve in each processor of the cluster or parallel architecture. Figure 3 shows the GP communications scheme in the parallel system. When the system starts, each processor receives the objective function (i.e, a true table), creates its sub-population and starts the evolution process, consisting of fitness evaluation, selection, crossover, mutation and reproduction. Once the system reaches a number of generations, some individuals are selected to be transfered from one processor to another one. A master processor is in charge of collecting the in-transfer individuals and moving them to the rest of the nodes (slaves),

increasing the probability of convergence of the algorithm. The ratio of data exchange (the number of the best individuals to exchange increases the probability of finding a better solution) and migration frequency are important parameters to improve the performance of the algorithm.

NOTA DE NIÑO: NO es claro por q afirma eso To implement communications in SIE, a custom message passing library was created to program the distributed system. NOTA DE NIÑO: so ????

4.1.2 GPU

Two main parts can be identified to implement the system on GPUs. The first part involves the random number generation and the second one the GP as such. The GP requires random numbers for generating an initial population and then to mutate and cross their individuals. Due that a GPU can not generate random numbers by using C classical libraries, it is necessary generate them in a different way. Generate these numbers on CPU and then transport them to the GPU is not viable because it takes a lot of system clocks. To solve this problem a Mersenne-twister algorithm is executed on the GPU before the $kernel - GP$ to make a buffer of random numbers on its own global memory.

Kernel structure..

Figure 4 shows the way the GP has been implemented on the graphics device. A thread t executes a $kernel - GP$, and generates a μ-population, performs the operators of selection, mutation and crossover a number of generations required. After P generations, M individuals will be transferred to the global memory and then to the host device (CPU system). P is known as the frequency of migration and the M number is called the migration factor.

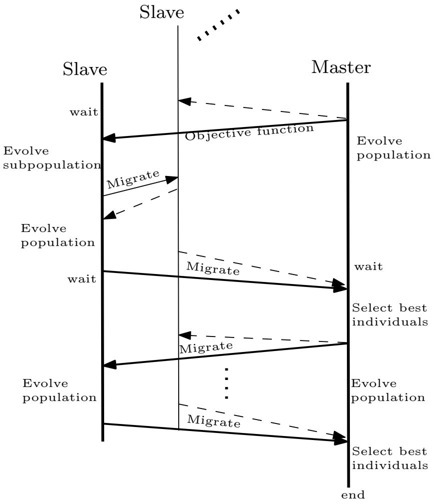

Figure 3: GP communications scheme in parallel architecture.

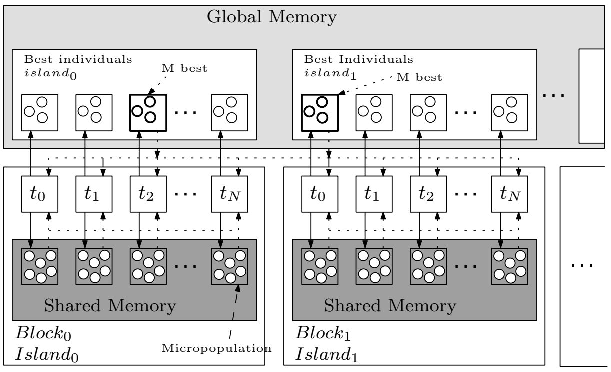

Figure 4: GPU implementation of the evolvable algorithm.

Also, it can be observed that each thread can cooperate with other threads inside the same block through shared memory, sharing the best individuals and improving the efficiency of the GP.

4.2 Hardware design.

In the second stage a profiling of the software implementation determined that the most costs were the fitness function calculation and the new individual generation (25% and 35% of the execution time, respectively). NOTA DE NINO: Como se calculó esto

Therefore, these two steps have been accelerated with a coprocessor in the FPGA. The Fitness Calculation Unit (FCU, see Figure 5) is the hardware element designed in order to accelerate this processes. This coprocessor is connected to the JZ4725 processor through its custom interface (using data, address, control buses).

NOTA DE NINO: No es claro, no se define que es una

LUT,

The chromosome passes from the DRAM memory to the FCU through the custom interface, and each of its cells are converted to an equivalent Look Up Table in ROM based translation. Then the number of wrong minterms is calculated using the information from the objective function and a counter as input. Finally, the FCU calculates the final fitness value including the number of gates and the critical path, and will be sent back to the JZ4725 processor. In order to speed up pseudo random number generation, a Mersenne-Twister-based coprocessor was inserted through the same custom interface.

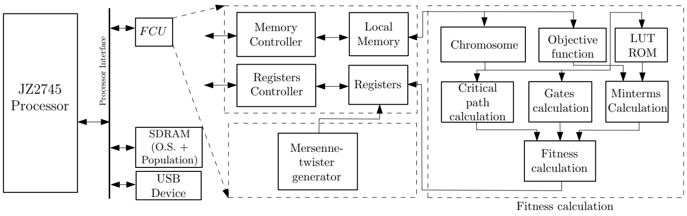

Figure 5: FCU structure.

5. EVALUATION.

To obtain the performance and prove the scalability of the algorithm in SIE, it was compared to a High Performance Computer (HPC) called ALTAMIRA. The cluster setup is made up of 18 eServer BladeCenters with 256 JS20 nodes (512 processors), using a Myrinet network with 1 Gbps bandwidth. Also, in order to test the performance of the GP in a GPU architecture a NVIDIA GTS450 was used, made by 192 CUDA cores, each one working at 1,56 GHz and a DDR5-1GB global memory.

On the other hand, the SIE configuration is made up of 6 JZ4725-FPGA nodes with the architecture described obove. The response times are measured for the parallelized versions of the GP in both ALTAMIRA and SIE. Several scenarios have been tested with different input parameter configurations: 1) number of input variables (4, 8 or 12, corresponding to a comparator problem of 2, 4 and 6 bits); 2) population size (512, 1024 or 2048) and 3) number of nodes running the experiment, ranged from 1 to 6 in SIE, and 2 to 16 or 64 in ALTAMIRA. The first and second parameters determine the size of the problem. The last one gives an idea about the scalability of the system. NOTA DE NINO: El título no es claro, se debe describir el experimento, y explicar mejor cuales son las plataformas que se están comparando

5.1 Response Time.

Figure 6 shows the response time for both platforms with different numbers of nodes and variables with 1024 individuals during 100 generations. These results show the high performance of SIE cluster for the algorithm. This experiment demonstrates that the response time for SIE does not depend on the size of the problem. In contrast, response time in ALTAMIRA has a high dependency of the size of the problem, because individuals had to be evaluated by software.

Figure 7 shows the response time in both architectures when varying the number of individuals of the population. It is observed that both architectures have a strong dependency of the number of individuals in the algorithm. This is because increasing this number causes an increment of the software computational load for both clusters. Even in this scenario, SIE shows an excellent performance compared to ALTAMIRA.

NOTA DE NIÑO: COMO LO OBSERVO??

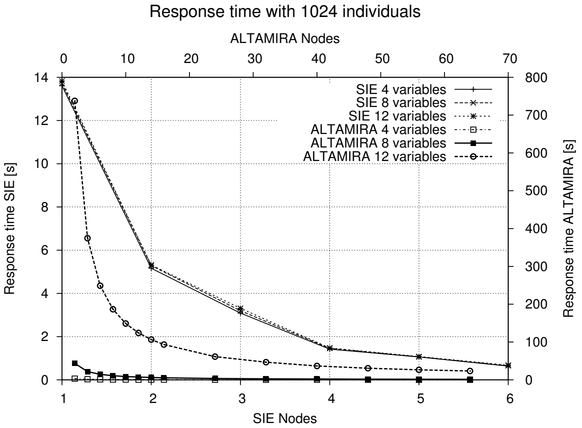

Figure 6: Response time in SIE and ALTAMIRA for different number of variables.

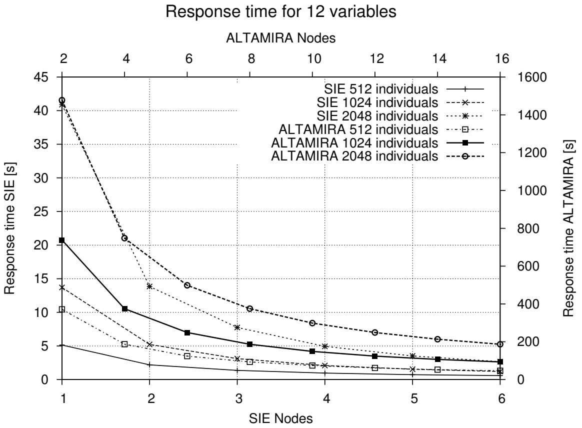

Figure 7: Response time in SIE and ALTAMIRA for different number of individuals.

Figures 8 shows the response time of executing the GP in the graphics hardware with problems of 4, 8 and 12 variables are fixed. Varying the number of islands and threads, can be observed that the best scenario is obtained when the number of threads is increased independently the number of islands.

This is because in this configuration more parallelism is performed in the system. NOTA De NIÑO: No se entiende porque de esa afirmación

NOTA DE NIÑO: NO SE ENTIENDE PORQUE FALTA DEFINIR EL MARCO DE TRABAJO, COMO SE REALIZÓ LA PRUEBA

5.2 Speedup.

The speedup of the SIE vs ALTAMIRA for different number of variables is presented in figure 10 with the number of

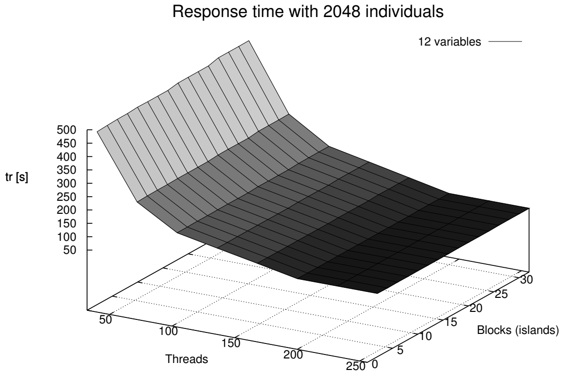

Figure 8: Response time of the algorithm in NVIDIA GTS450 GPU.

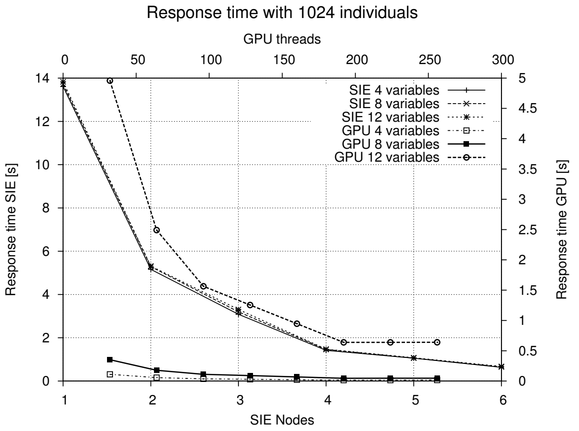

Figure 9: Response time in SIE and GPU for different number of individuals.

variables fixed to 12. The excellent performance of SIE can be explained because individuals have been directly tested in hardware (FPGA), obtaining a combination of their true table on each cycle of the system clock. On the other hand, individuals evaluated in software by ALTAMIRA require a lot of system clocks, causing response times hundreds of times higher than SIE.

NOTA DE CAIN: Acá se debe hablar de la ventaja de utilizar la arquitectura propuesta basada en la LUT virtual y de como es conveniente implementarla en HW, ya que se demuestra que en SW va mas lento.

On the other hand, figure 11 shows the speedup number when SIE and GPU are compared. It can be observed that SIE performance is higher than GPU only when 12-variables problems are evolved. This can be explained because the whole population have been tested in hardware, obtaining a combination of their output on each cycle of the system clock. But, when an individual is tested in software, each combination requires a set of instructions, that requires a lot of cycles of the system clock.

5.3 Resulting circuits.

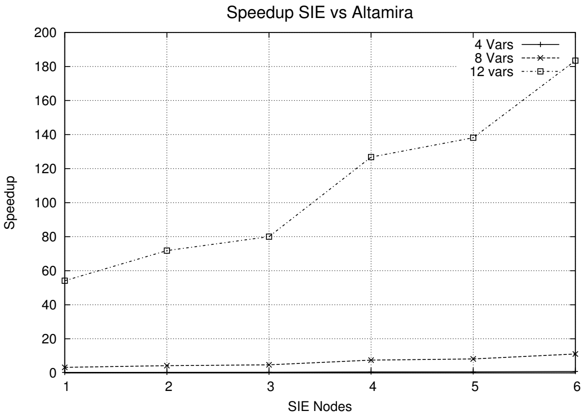

Figure 10: Speedup of SIE vs Altamira comparing 1 SIE node = 2 Altamira nodes.

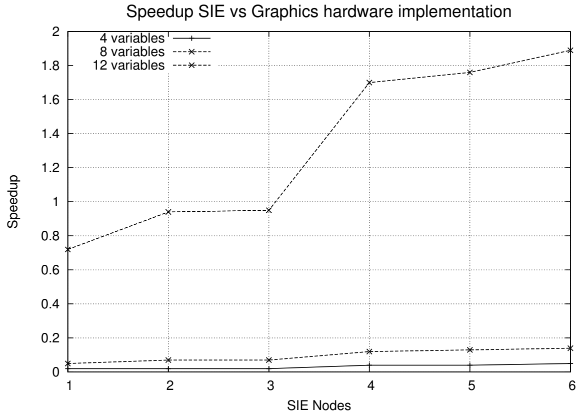

Figure 11: Speedup of SIE vs GPU comparing 1 SIE node = 32 threads.

Figure 12 shows an example of the resulting circuits for the 2-bit comparator problem. These resulting structures are novel compared to the results obtained with other techniques, such as Karnaugh maps and QuineMcCluskey.

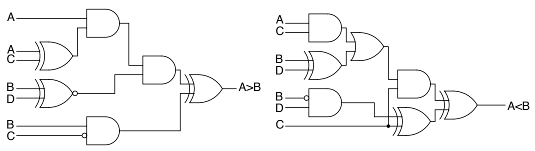

Figure 12: Example of resulting circuits for the 2-bit comparator problem (A=MSB number1, C=MSB number2).

6. CONCLUSIONS AND FUTURE WORK.

This paper showed a novel way to evaluate individuals in an evolvable algorithm on an open embedded platform called SIE, and results were compared to an HPC called ALTAMIRA and a high performance NVIDIA GPU. To accelerate the evolution process, a coprocessor was implemented to calculate the fitness function and to generate random numbers, improving the performance for problems with more than 6 bits. Tests proved that the algorithm is more effective for 4-bit and 8-bit problems. 12-bit problems in SIE had excellent performance, but because the search space is too long, converging to a suitable solution was difficult for the algorithm. This problem could be solved as future work with some improvements in terms of multiple FCUs inside an FPGA, more nodes, and other hardware-accelerated genetic operators.

NOTA DE CAIN: EN LAS CONCLUSIONES SE DEBE decir que se proporciona una arquitectura para realizar la evolución intrínseca

7. REFERENCES

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

[2] Creative Commons. Creative Commons Licenses. URL: http://creativecommons.org/licenses., 2004.

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

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

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

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

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