International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 3, September 2005, pp. 206-215

# Design of Evolvable Hardware based on Genetic Algorithm Processor(GAP)

**Kwee-Bo Sim* and Fumio Harashiam****

* School of Electrical and Electronic Engineering, Chung-Ang University
221, Heukseok-Dong, Dongjak-Gu, Seoul, 156-756, Korea
Tel : +82-2-820-5319, Fax : +82-2-817-0553, E-mail : kbsim@cau.ac.kr

** Department of Electrical Engineering, Tokyo Denki University, Japan
2-2, Kanda-nishikicho, Chiyoda-ku, Tokyo 101-8457, Japan
E-mail: f.harashima@cck.dendai.ac.jp, f.harashima@ieee.org

**Abstract**

In this paper, we propose a new design method of Genetic Algorithm Processor(GAP) and Evolvable Hardware(EHW). All sorts of creature evolve its structure or shape in order to adapt itself to environments. Evolutionary Computation based on the process of natural selection not only searches the quasi-optimal solution through the evolution process, but also changes the structure to get best results. On the other hand, Genetic Algorithm(GA) is good for finding solutions of complex optimization problems. However, it has a major drawback, which is its slow 'execution speed when is implemented in software of a conventional computer. Parallel processing has been one approach to overcome the speed problem of GA. In a point of view of GA, long bit string length caused the system of GA to spend much time that clear up the problem. Evolvable Hardware refers to the automation of electronic circuit design through artificial evolution, and is currently increased with the interested topic in a research domain and an engineering methodology. The studies of EHW generally use the XC6200 of Xilinx. The structure of XC6200 can configure with gate unit. Each unit has connected up, down, right and left cell. But the products can't use because had sterilized. So this paper uses Vertex-E (XCV2000E). The cell of FPGA is made up of Configuration Logic Block (CLB) and can't reconfigure with gate unit. This paper uses Vertex-E is composed of the component as cell of XC6200 cell in VertexE

**Key words :** Evolvable Hardware, Genetic Algorithm Processor, Field Programmable Gate Arrays, Hardware reconfiguration method

## 1. Introduction

The evolution is basically the process of self-copy with inherited variation. Therefore the software evolution get accomplished by executing the self-copy program made by machine language in the virtual world of computer which have the tool making mutant. How do hardware evolution proceed? It needs the hardware of structure that can mutate gene and do self-copy as software evolution.

Reconfigurable hardware is applied to the many fields because it can configure logic circuit of a variety of structure without semi-conductor manufacturing process. Also it can make a robust hardware system to adapt environment variation.

Evolvable Hardware (EHW) is a new concept toward the development of on-line adaptive machines. The most apparent distinction between conventional hardware (CHW) and EHW is as follows. The design of CHW cannot be started unless hardware specifications are given to the designer. In contrast, EHW can be used in situations where no one knows the desired hardware specification in advance. EHW can reconfigure its hardware structure by genetic learning [1]. EHW tries to attain the following two goals by implementing adaptive machines. First is the development of a new type of fault-tolerant systems where EHW changes its own hardware architecture to adapt to changes in the environment. The second is the development of an innovative machine learning system based on EHW. It is capable of storing the learned result directly in the hardware structure. This leads to a new learning paradigm totally different form artificial neural network and other rule-based systems [2].

EHW uses Genetic Algorithm (GA). Because the size of hardware is not change, the change that this paper says is the redundancy to make a digital logic. Recently there are the applications of embryology [3] and Genetic Programming [4] and so on.

Evolutionary Computation is model on the process of natural evolution. Not only it searches the right solution, but also changes the structure to get results. Genetic Algorithm (GA) is applied to various fields because it is simple to theory and easy to application. GA is good at finding solutions for complex optimization problems. GA has a major drawback [5], which is its slow execution speed when is implemented in software of a conventional computer. Parallel processing has been approached overcoming the speed problem of GA.

Manuscript received March 21, 2005; revised August 25, 2005
This research was supported by the project of *Developing SIC (Super Intelligent Chip) and Its Applications* under the program of Next generation technologies in 2000 of Ministry of Commerce, Industry, and Energy.

More recently, engineers have been allured by certain natural processes, giving birth to such domains as artificial neural networks and evolutionary computation. Living organisms are complex systems exhibiting a range of desirable characteristics, such as evolution, adaptation, and fault tolerance, that have proved difficult to realize using traditional engineering methodologies. Such systems are characterized by a genetic program, the genome, that guides their development, their functioning, and their death. If one considers life on Earth since its very beginning, then the following three levels of organization can be distinguished [6].

① Phylogeny: The first level concerns the temporal evolution of the genetic program, the hallmark of which is the evolution of species, or phylogeny. The multiplication of living organisms is based upon the reproduction of the program, subject to an extremely low error rate at the individual level, so as to ensure that the identity of the offspring remains practically unchanged. Mutation (asexual reproduction) or mutation along with recombination (sexual reproduction) give rise to the emergence of new organisms. The phylogenetic mechanisms are fundamentally nondeterministic, with the mutation and recombination rate providing a major source of diversity. This diversity is indispensable for the survival of living species, for their continuous adaptation to a changing environment, and for the appearance of new species.

② Ontogeny: Upon the appearance of multicellular organisms, a second level of biological organization manifests itself. The successive divisions of the mother cell, the zygote, with each newly formed cell possessing a copy of the original genome, is followed by a specialization of the daughter cells in accordance with their surroundings, i. e., their position within the ensemble. This latter phase is known as cellular differentiation. Ontogeny is thus the developmental process of a multicellular organism. This process is essentially deterministic: an error in a single base within the genome can provoke an ontogenetic sequence which results in notable, possibly lethal, malformations.

③ Epigenesis: The ontogenetic program is limited in the amount of information that can be stored, thereby rendering the complete specification of the organism impossible. A well-known example is that of the human brain with some $10^{10}$ neurons and $10^{14}$ connections, far too large a number to be completely specified in the four-character genome of length approximately $3\times10^{9}$. Therefore, upon reaching a certain level of complexity, there must emerge a different process that permits the individual to integrate the vast quantity of interactions with the outside world. This process is known as epigenesis and primarily includes the nervous system, the immune system, and the endocrine system. These systems are characterized by the possession of a basic structure that is entirely defined by the genome (the innate part), which is then subjected to modification through lifetime interactions of the individual with the environment (the acquired part). The epigenetic processes can be loosely grouped under the heading of learning systems.

In analogy to nature, the space of bio-inspired hardware systems can be partitioned along these three axes: phylogeny, ontogeny, and epigenesis; we refer to this as the POE model [7] The distinction between the axes cannot be easily drawn where nature is concerned; indeed the definitions themselves may be subject to discussion. We therefore define each of the above axes within the framework of the POE model as follows: the phylogenetic axis involves evolution, the ontogenetic axis involves the development of a single individual from its own genetic material, essentially without environmental interactions and the epigenetic axis involves learning through environmental interactions that take place after formation of the individual. As an example, consider the following three paradigms, whose hardware implementations can be positioned along the POE axes: (P) evolutionary algorithms are the (simplified) artificial counterpart of phylogeny in nature, (O) multicellular automata are based on the concept of ontogeny, where a single mother cell gives rise, through multiple divisions, to a multicellular organism, and (E) artificial neural networks embody the epigenetic process, where the system's synaptic weights and perhaps topological structure change through interactions with the environment. Within the domains collectively referred to as soft computing [4], often involving the solution of ill-defined problems coupled with the need for continual adaptation or evolution, the above paradigms yield impressive results, frequently rivaling those of traditional methods.

## 2. Evolvable Hardware

In this section we explore the phylogenetic axis of bioinspired systems, also referred to as evolvable hardware. The main motivation is to attain adaptive systems that are able to accomplish difficult tasks, possibly involving real-time behavior in a complex, dynamical environment. We begin by briefly introducing two underlying themes, artificial evolution and large-scale programmable circuits.

### 2.1 Artificial Evolution

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than three decades ago, has seen impressive growth in the past few years. Usually grouped under the term evolutionary algorithms or evolutionary computation, we find the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming [8]. As a generic example of artificial evolution, we consider genetic algorithms [9]. A genetic algorithm is an iterative procedure that consists of a constant-size population of individuals, each one represented by a finite string of symbols, known as the genome, encoding a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at

hand. The algorithm sets out with an initial population of individuals that is generated at random or heuristically. At every evolutionary step, known as a generation, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the fitness, or fitness function. To form a new population, individuals are selected according to their fitness and then transformed via genetically inspired operators, of which the most well known are crossover and mutation. Iterating this procedure, the genetic algorithm may eventually find an acceptable solution, i. e., one with high fitness. Evolutionary algorithms are common nowadays, having been successfully applied to numerous problems from different domains, including optimization, automatic programming, machine learning, economics, immune systems, ecology, population genetics, studies of evolution and learning, and social systems.

### 2.2 Large-Scale Programmable Circuits

An integrated circuit is called programmable when the user can configure its function by programming. The circuit is delivered after manufacturing in a generic state and the user can adapt it by programming a particular function. In this paper we consider solely programmable logic circuits, where the programmable function is a logic one, ranging from simple Boolean functions to complex state machines. The programmed function is coded as a string of bits representing the configuration of the circuit. Note that there is a difference between programming a standard microprocessor chip and programming a programmable circuit-the former involves the specification of a sequence of actions, or instructions, while the latter involves a configuration of the machine itself, often at the gate level. The first programmable circuits allowed the implementation of logic circuits that were expressed as a logic sum of products. These are the PLD's (programmable logic devices), whose most popular version is the PAL (programmable array logic). More recently a novel technology has emerged, affording higher flexibility and more complex functionality: the field programmable gate array (FPGA) [10]. An FPGA is an array of logic cells placed in an infrastructure of interconnections, which can be programmed at three distinct levels:

(a) the function of the logic cells.
(b) the interconnections between cells.
(c) the inputs and outputs.

All three levels are configured via a string of bits that is loaded from an external source, either once or several times. In the latter case the FPGA is considered reconfigurable. FPGA's are highly versatile devices that offer the designer a wide range of design choices. This potential power, however, necessitates a suite of tools to design a system. Essentially, these generate the configuration bit string, given such inputs as a logic diagram or a high-level functional description.

### 2.3 Research of Evolvable Hardware

If one carefully examines the work carried out to date under the heading evolvable hardware, it becomes evident that this mostly involves the application of evolutionary algorithms to the synthesis of digital systems [10]. From this perspective, evolvable hardware is simply a subdomain of artificial evolution, where the final goal is the synthesis of an electronic circuit. The work of Koza [11], which includes the application of genetic programming to the evolution of a three-variable multiplexer and a two-bit adder, may be considered an early precursor along this line. It should be noted that at the time the main goal was that of demonstrating the capabilities of the genetic programming methodology, rather than designing actual circuits. We argue that the term evolutionary circuit design would be more descriptive of such work than that of evolvable hardware. For now, we shall remain with the latter term.

Evolvable hardware, taken as a design methodology, offers a major advantage over classical methods. The designer's job is reduced to constructing the evolutionary setup, which involves specifying the circuit requirements, the basic elements, and the testing scheme used to assign fitness. If these have been well designed, evolution may then generate the desired circuit. Currently, most evolved digital designs are suboptimal with respect to traditional methodologies; however, improved results are regularly demonstrated. When examining work carried out to date, one can derive a rough classification of current evolvable hardware, in accordance with the genome encoding, and the calculation of a circuit's fitness.

**(1) Genome Encoding**: High-level languages. Using a high-level functional language to encode the circuits in question means that the final solution must be transformed to obtain an actual circuit. Thus, such a representation is far removed from the structural description. The evolved solution is a program describing the multiplexer or adder rather than an interconnection diagram of logic elements.

Low-level languages. The idea of directly incorporating the bit string representing the configuration of a programmable circuit within the genome was expressed early on by Atmar [12] and more recently by de Garis [13] and Higuchi et al. [14]. As a first step one must choose the basic logic gates (e.g., AND, OR, and NOT) and suitably codify them, along with the interconnections between gates, to produce the genome encoding. An example of this approach is offered in [15]. Higuchi et al. [13] used a low-level bit string representation of the system's logic diagram to describe small-scale PAL's, where the circuit is restricted to a logic sum of products. The limitations of the PAL circuits have been overcome to a large extent by the introduction of FPGA's, as used, e.g., by Thompson [15]. The use of a low-level circuit description that requires no further transformation is an important step forward since this potentially enables placing the genome directly in the actual circuit, thus paving the way toward truly evolvable hardware. Until recently, however, FPGA's had presented two major problems:

(1) the genome's length was on the order of tens of thousands of bits, rendering evolution practically impossible using current technology.

(2) within the circuit space, consisting of all representable circuits, a large number were invalid (e.g., containing short circuits). With the introduction of the new family of FPGA's, the Xilinx 6200 [16]

These problems have been reduced. As with previous FPGA families, there is a direct correspondence between the bit string of a cell and the actual logic circuit; however, this now always leads to a viable system. Moreover, as opposed to previous FPGA's where one had to configure the entire system, the new family permits the separate configuration of each cell, a markedly faster and more flexible process.

Thompson has employed this latter characteristic to reduce the genome's size, without, however, introducing real-time, partial system reconfigurations.

**(2) Fitness Calculation**: Off-line evolvable hardware. The use of a high-level language for the genome representation means that one has to transform the encoded system to evaluate its fitness. This is carried out by simulation, with only the final solution found by evolution actually implemented in hardware. This form of simulated evolution is known as off-line evolvable hardware.

On-line evolvable hardware. As noted above, the low level genome representation enables a direct configuration of the circuit, thus entailing the possibility of using real hardware during the evolutionary process. D. Common Features of Current Phylogenetic Hardware Examining work carried out to date we find many common characteristics that span most current systems, both on-line and off-line, often differing from biological evolution.

Evolution pursues a predefined goal: the design of an electronic circuit, subject to precise specifications. Upon finding the desired circuit, the evolutionary process terminates. The population has no material existence. At best, in what has been called on-line evolution, there is one circuit available, onto which individuals from the (offline) population are loaded one at a time, to evaluate their fitness. The absence of a real population in which individuals coexist simultaneously entails notable difficulties in the realization of interactions between "organisms." This usually results in a completely independent fitness calculation, contrary to nature which exhibits a coevolutionary scenario.

If one attempts to resolve a well-defined problem, involving the search for a specific combinatorial or sequential logic system, there are no intermediate approximations. Fitness calculation is carried out by consulting a lookup table which is a complete description of the circuit in question, that must be stored somewhere. This casts some doubts as to the utility of applying an evolutionary process, since one can directly implement the lookup table in a memory device, a solution which may often be faster and cheaper. The evolutionary mechanisms are executed outside the resulting circuit. This includes the operators as well as fitness calculation. As for the latter, while what has been advanced as on-line evolution uses a real circuit for fitness evaluation, the fitness values themselves are stored elsewhere. The different phases of evolution are carried out sequentially, controlled by a central software unit.

The evolutionary mechanisms are executed outside the resulting circuit. This includes the operators as well as fitness calculation. As for the latter, while what has been advanced as on-line evolution uses a real circuit for fitness evaluation, the fitness values themselves are stored elsewhere. The different phases of evolution are carried out sequentially, controlled by a central software unit.

### 2.4 FPGA device

The studies of EHW generally use the XC6200 or GAL [17][18]. FPGA are the preferred device for many groups because they can be re-configured virtually instantaneously to produce a physical circuit, which can be evaluated in real time [19]. Fig. 1 is a simplified diagram showing the architecture of the Xilinx XC6216 FPGA. The magnified view at the bottom of the diagram is of the basic configurable element or cell. Each cell can be connected to any of its adjacent neighbors with further hierarchical interconnections possible. This particular device used 64 X 64 cells totaling 4096. Despite this large number of configurable elements and interconnections, FPGA are far from ideal for hardware evolution for a number of reasons. There is very little choice over the type of element employed in commercial devices - the XC6200 series uses simple Boolean function, while other FPGA employ higher-level configurable logic blocks (CLB) such as adders and multipliers [20]. It can configure by gate unit. Differently other device, XC6200 is known to data format and can be changed in one area of the device without affecting another area. The cell of XC6200 put out the each output up, down, left, right. The logic of each output is different: AND, OR, XOR and so on. Because of this property, XC6200 series could be used in EHW. But we use Vertex of Xilinx because it can't make general a logic circuit.

There are two methods of EHW [21]: first method known as Extrinsic EHW and is the evolution of electronic circuits through simulation; at the end of each generation the best individual is downloaded to the electronic device. The second method, Intrinsic EHW, is when each genotype is assessed on the device by downloading the new configuration and testing the device directly. Intrinsic EHW, therefore, requires that the hardware be reconfigured for each individual in every generation. This means that the hardware requires the ability to be reconfigured very quickly. We use GAP to reconfigure EHW. This method is faster than intrinsic method.

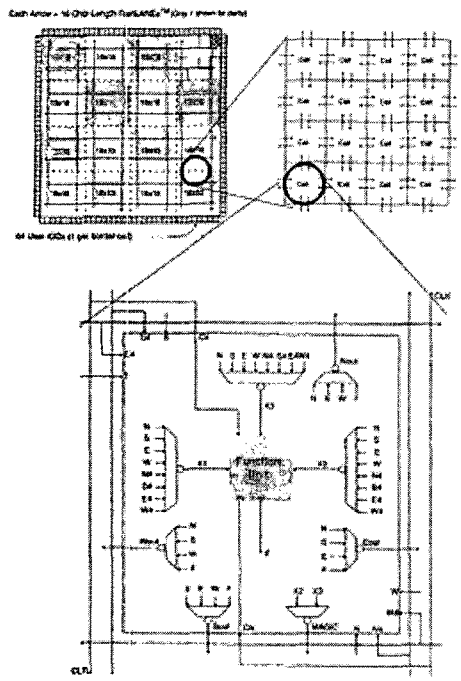

Fig. 1. The structure of XC6200

### 2.5 The structure of EHW at the vertex

We use the vertex to make GAP at FPGA. Loading application-specific configuration data into internal memory configures Vertex devices. Configuration is carried out using a subset of the device pins, some of which are dedicated, while others can be reused as general-purpose inputs and outputs after configuration is complete.

The structure of Vertex is different with XC6200. Vertex is composed of Configuration Logic Block (CLB) and can't reconfigure by gate unit. It is composed of Lookup Table (LUT) and Flip Flop. LUT can make some sequential logic that has 4 inputs and 1 output. Each LUT output can connect to three other LUTs in the same CLB. It can act fast local feedback routing. CLBs are connected between adjacent they increases the speed of designs.

For applying Vertex to the structure of XC6200, we have made a component to use Verilog HDL. It makes to act as the cell of XC6200. Each direction of function unit has two configuration bits which make roles of AND, OR, NOT, BUFFER. EHW of this paper is made up 36 cells. Each cell outputs different signals because it is composed of different function logic from inputs of 4 directions. Because of this architecture, it can be composed of complex logic circuits. EHW module is made of 36 cells. It needs 288 bits because each cell needs 8 bits. It needs the repeated sequences of 9 times to configure 288 bit on EHW. We make EHW module by this method. The cell is connected each other as the components. The outer edge wires of EHW are inputs and outputs. It is 24 bit because EHW is made of 36 cells.

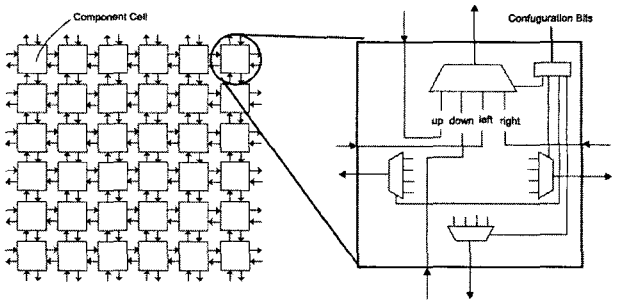


Fig. 2. The structure of EHW module.

There are another modules to evaluate fitness: IO unit, memory of evaluation vectors, counter and comparator. It uses the evaluation vectors to make a logic circuit. The comparator evaluates fitness to compare input-output pare with evaluation vector. The controller of EHW is synchronized with GAP.

### 2.6 Design of state machine

State machine is a model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state[22]. Computation begins in the start state with an input string. It changes to new states depending on the transition function. There are many variants, for instance, machines having outputs associated with transitions or states, multiple start states, transitions conditioned on no input symbol or more than one transition for a given symbol and state, one or more states designated as accepting states, etc [23].

To design state machine in the logic circuit, we must make state diagram as Fig. 3. Then we make state table and state transition table. Using it, we can compose a karnough map, and then design a logic circuit. It is important fact that state-machine is a feedback circuit because must know the current state. Output of this state machine is the current state.

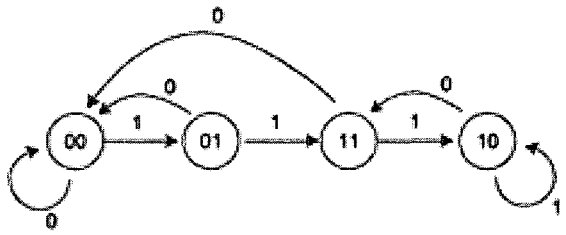


Fig. 3. State diagram. It changes by input data and current state.

To design state machine using EHW module, we make test vector. Table 2 shows test vector that is used to make state machine in the EHW module. The vector is serial data loaded sequentially from memory. As show Table 2, output is different as current state though input is same. Using test vector, we can measure fitness. Fig. 4 shows a process to evaluate fitness. We can know to compare memory data with EHW outputs. EHW return the fitness to GAP. Then we can proceed to Genetic Algorithm.

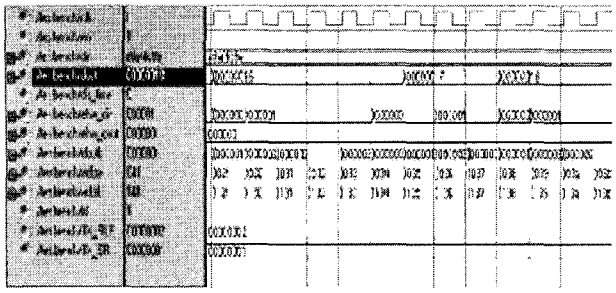

Fig. 4. The waveform of EHW action.

## 3. Genetic Algorithm Processor(GAP)

### 3.1 Genetic Algorithm

John Holland, finding their inspiration in the evolutionary process occurring in nature, invented Genetic Algorithm. The main idea is that in order for a population of individuals to collectively adapt to some environment, it should behave like natural system: survival, and therefore reproduction, is promoted by the elimination of useless of harmful traits and by rewarding useful behavior. Holland's insight was in abstraction the fundamental biological mechanisms that permit system adaptation into a mathematically well-specified algorithm [24].

Genetic Algorithm (GA) has been used essentially for searching and optimization problems and for machine learning. However, it is still an unresolved question whether the natural evolutionary process is really an optimization process. Evolution is essentially one-shot experiment, although many alternatives were tried along the way and discarded through the selection process. GA is the iterative procedure that consists of a constant-size population of individuals, each one represented by a finite string of symbols encoding a possible solution in some problem space. This space, also known as the search space, comprises all possible solutions to the problem at hand.

However, GA has one major draw back, which is their slow execution speed when implemented in software of a conventional computer. Parallel processing [25] has been one approach to overcome the speed problem of GA. There is the case used the multiple GAP [8]. In a point of view of GA, long bit string length caused the system of GA to spend much time that clear up the problem. It makes to slow the execution speed of GA.

### 3.2 The structure of Genetic Algorithm Processor

This paper propose pipeline GAP to manage long bit sting and to use efficiently the hardware resource. A genetic algorithm processor can be constructed to directly execute the operation of a genetic algorithm. Fig 5 shows the structure of GAP. Such a processor can be used in situations where high throughput is required and where the logic of the genetic algorithm is expressible in simple units that can be synthesized in hardware. This is generally the case as genetic algorithms are inherently simple and contain only a few logic operations. The whole GAP section was written in Verilog HDL(Hardware Descriptive Language) and simulated and synthesized by Xilinx software

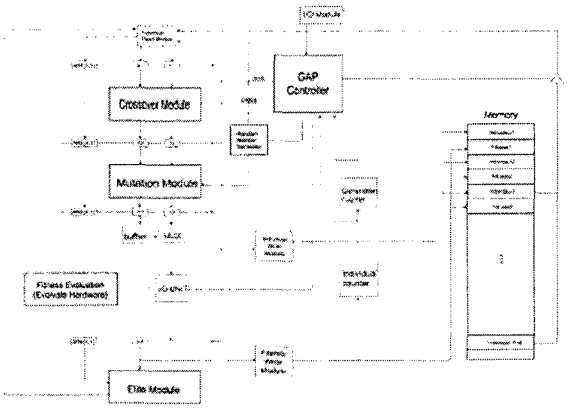


Fig. 5. The structure of GAP

It is composed of five essential parts. First part is a crossover module. It acts crossover operation. The mutation module acts the one of 32 bits to compare random number with mutation rate. So mutation rate should be lower than general Genetic Algorithm. The elite module saves the fitness and the address for individual of high fitness to reproduce and GA converges much faster. Because GAP wastes many memories when it save long bits, so it saves the only fitness and address for efficiency. Because the nature is the complex system, we need the random number for its imitation. The random number is caused by follow.

$$R_i+1 = (A^{\times} R_i + B)\%M \quad (1)$$

In upper equation, A, B and M are any numbers with no rule. $R_i$ is a seed number, and initial $R_i$ is number counted clock.

In this Fig. 6, each data bus is 32 bits. But it manages much long bit string because this process is the structure of pipeline. The process is as follows. It loads the two of 32bits data from the dual port memory. And it passes the data to crossover module then loads again the 32 bits data. It loads long data by this method. Because the datum has passed the mutation module aretwo individuals, one individual is waiting during the evaluation of another individual.

The operational sequence of the GAP controller is as follows.

Table 1 is the control sequences that need to act GAP. For generation and reproduction, the sequence is divided in two. When the bit string is also more than 32 bits, some of the sequence repeats to manage a serial of data pipeline GAP. Specially time 0~4 among reproduction sequences are repeated at different timing each time. We design the counter control unit for these sequences. As show Fig. 6, this module is composed of comparator and counter. A time signal is counter signal of main clock number. But the repeated signals have nothing to do the main clock. It related t[3:0] signals. The comparator outputs a less signal until to satisfy the repeated times. It repeats until to make bit string that Individual need.

The down registers of Fig. 6 delay one clock for repeated signal. The sequences are repeated by seach delayed signal. Time 6 also repeated same as time 0~4. Time 7 is different from repetition of time 0~4. It repeats to evaluate fitness in evaluation module. The fitness evaluation module is different as what it solves the problem. It means the time to evaluates fitness. So fitness evaluation module outputs time-stop signals during to measure fitness.

Memory address is divided in two. Main address and sub address. Main address is address for individual and sub address is address for long bit string. When bit string is long, main address is fixed and sub address only increased. So address of each individual is uniform, its fitness also saves a fixed position.

GAP is efficient to long bit string because it has this structure that can manage a variable bit string.

Table 1. The control sequences of GAP.

| | | |
|---|---|---|
| Generation | | |
| Time 0 | EHW_en, sel_chrom(r_num), **mem_in**, inc_sub_add | Repeated signal |
| Time 1 | EHW_en, EHW_inst, ld_sub_ad | |
| Time 2 | mem_in, clr_sub_addr, clk_rst | |
| Reproduction | | |
| Time 0 | mem_addr_ain, mem_addr_bin, inc_sub_ad | Repeated signal |
| Time 1 | cross_in, cross, clr_sub_ad | Repeated signal |
| Time 2 | mutation_in | Repeated signal |
| Time 3 | EHW_en, sel_chrom(a), fifo_ | Repeated signal |
| Time 4 | EHW_en, EHW_eval, ld_sub_ad | |
| Time 5 | mem_in, clr_sub_addr, sel_mem, clr_sub_ad | |
| Time 6 | EHW_en, fifo_aout, fifo_bout, inc_sub_addr, sel_chrom(b) | Repeated signal |
| Time 7 | EHW_en, EHW_eval, ld_sub_addr | Repeated signal |
| Time 8 | mem_in, clr_sub_addr, sel_mem, clr_sub_addr | |

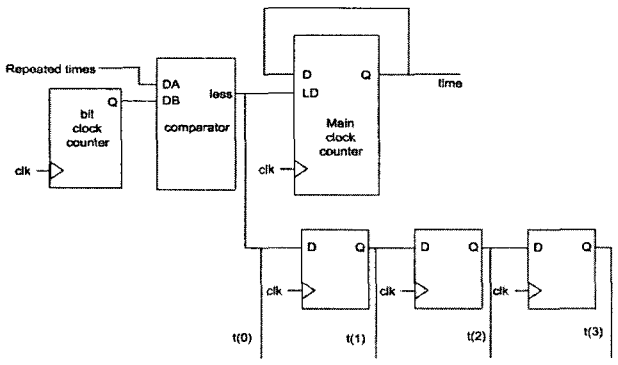


Fig. 6. Counter control unit. It needs to control iterative sequences.

## 4. Simulation Results

We have used as following table in all experiments. It is numerical value that is used general. FPGA is acted by 100 Mhz. It is the time that a one-generation act is 22s. It is faster than conventional computer that use Pentium4 1.4GHz. There is no system that stores the process of hardware evolution. We show sampling of the special part among evolution because we see only short interval(0.5μs).

Table 2. Genetic operator

| Operator | |
|---|---|
| Crossover rate | 0.8 |
| Mutation rate | 0.026 |
| Population size | 100 |

### 4.1 Test of Genetic Algorithm Processor

We tested to confirm the action and the performance of GAP. In the first place, The solution of Linear function is found by GAP. Fig. 7 shows the results of test. It shows through elite individuals of test how the solution evolves. Fitness normalized to 100. As shows, because of the simple problem, it has high fitness initially, converges about 20 generation.

Fig. 8 is displaying the result of one max problem. One max problem is how many one the bit string has. Fitness advances slowly because it changes by mutation rather than crossover.

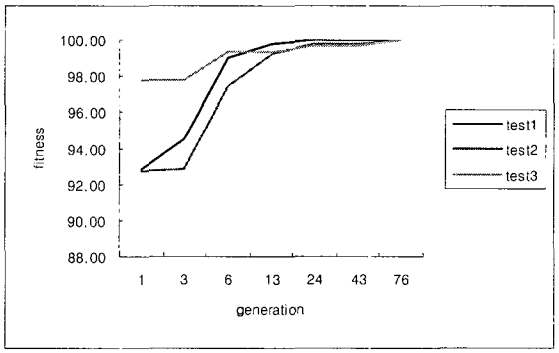


Fig. 7. The solution of linear equation

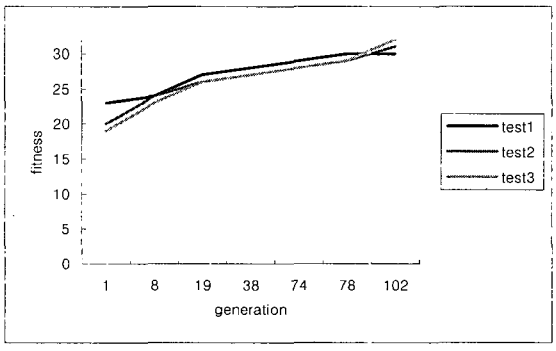


Fig. 8. The solution of one max problem (32bits)

In order to verify the performance of pipeline structure, We tested 64 bits one max problem. It can check up on pipeline GAP because chromosome is acted during two clocks. The result of test is Fig. 9. We can know for GAP to act normally because it converges on 64. It takes longer than 32 bits one max problem because of 2 times length.

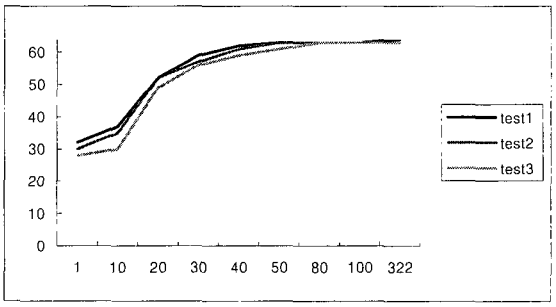


Fig. 9. The solution of one max problem (64bits)

For next, we will consider the NP-hard set covering problem [26]. The set covering problem is an optimization problem that models many resource selection problems and is important for logic circuit minimization [27].

The set covering problem can be defined as follows: given a collection C of finite sets, each with non-negative cost, find a minimum-cost sub-collection C' such that every element within the sets in C belongs to at least one set in C'.

Fig. 10 show the result of test. It also converges initially in the same result as former experiments.

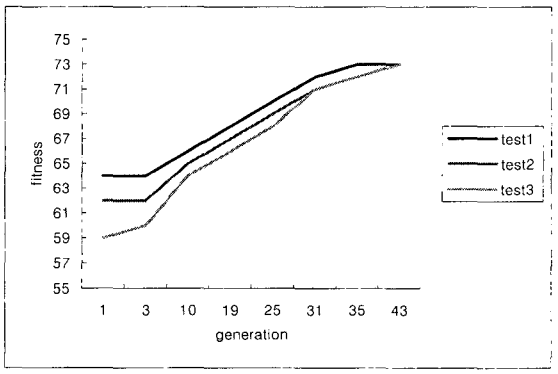


Fig. 10. Set Covering problem.

### 4.2 Test of Evolvable Hardware

We tested to design the 3 bits adder at Evolvable Hardware. It has fitness-64 When hardware evolves finally because all cases of 3 bits adder is $2^6$. It doesn't make the 3 bits adder. Fig. 11 shows the fitness of evolution process. It has converged fitness-20 at 20000 generation. It causes effect to have the EHW cell of small number. So we increase the number of EHW cell from 6×6 to 6×12. Fig. 12 shows the result of it. It increases fitness-47 but converges on that point. We can increase the EHW cell because it is small the memory of VertexE.

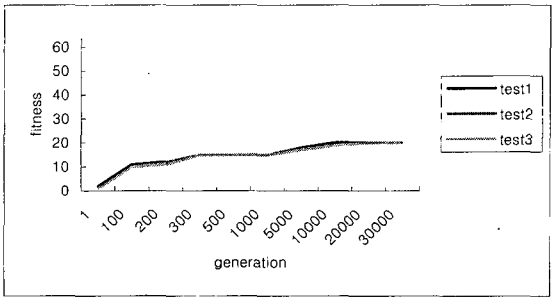


Fig. 11. 3bits adder of Evolvable Hardware (6×6)

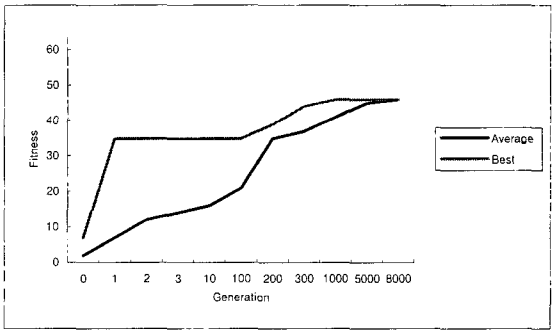


Fig. 12. 3bits adder of Evolvable Hardware (6×12)

## 5. Conclusion

This paper proposes the design method of logic circuit using GAP and EHW. We have addressed the problem of slow execution speed of software implementation of the genetic algorithm by design a pipelined genetic algorithm processor that can managed chromosome per machine cycle. It solves the problem without special knowledge as the machine evolves by it self when appears the problem. GA in the hardware shows faster then software. GAP supplements drawback of GA, which has many computation quantity. GAP can be used to general processor to find optimal solution because can handle variant bit. Also it is composed of pipeline, saves resource of FPGA when it manages long bit string. The total modules of this paper can be applied to EHW of other structure because GAP can manage variant bit string.

EHW download a configuration bits using the intrinsic method, but this paper shows one chip EHW It can evolve much faster than intrinsic method because reconfigure hardware. Also it can make complex logic circuit by connected the cell to 4 directions. It can't makes the digital circuit completely but shows the possibility of Evolvable Hardware. It need more memory than VertexE to make the complex circuit.

## References

[1] E. Sanchez, "Field Programmable Gate Array(FPGA) Circuits," *Lecture Notes in Computer Science*, pp1-18, Springer, 1991.

[2] Gordon Hollinworth, Steve Smith, Andy Tyrrell, "The Intrinsic Evolution of Vertex Devices," *Evolvable Systems: From biology to Hardware, ICES2000,* pp.72-79,2000

[3] Hector Fabio Restrepo, Daniel Mange, "An Embryonic Implementation of a Self-Replicating Universal Turing Machine", *Evolvable Systems: From biology to Hardware, ICES2001*, pp. 74-87, Springer, 2001

[4] Seok, Ho-Sik, "A Study on Autonomous Robot Controllers Using Genetic Programming and Evolvable Hardware", *School of Computer Sci. and Eng., Seoul National University*, 2001.

[5] Barry Shackleford, Greg Snider, Richard J. Carter, Etsuko Okushi, Mitsuhiro Yasuda, Katsuhiko Seo, Hiroto Yasuura, "A High-performance, Pipelined, FPGA-Based Genetic Algorithm Machine", *Genetic Programming and Evolvable Machines*, pp33-60, 2001.

[5] A. Danchin, "A selective theory for the epienetic specification of the monospecific antibody production in single cell lines," *Ann. Immunol.(Institut Pasteur)*, vol 127C, pp. 787-804, 1976.

[6] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "Phylogeny, ontogeny and epigenesis: Three sources fo biological inspiration for softening hardware, " *in Proc. Evolvable System: From Biology to Hardware(ICES96)*, Springer-Verlag, 1997

[7] D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. Piscataway*, NJ: IEEE Press, 1995

[8] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[9] E. Sanchez, "Field-Programmable Gate Array(FPGA) circuits," *in Toward Evolvable Hardware*, Springer-Verlag, 1996, pp.1-18.

[10] J. R. Koza, *Genetic Programming,* Cambridge, MTI Press, 1992.

[11] J. W. Atmar, "Speculation on the evolution of intelligence and its possible realization in machine form," *Ph.D. dissertation, New Mexico State University*, Las Cruces, 1976.

[12] H. de Garis, " Evolvable hardware: Genetic programming of a Darwin machine," *in Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, 1996.

[13] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya, " Evolving hardware with genetic learning," *Proc. 2nd Int. Conf. Simulation of Adaptive Behavior*, MIT Press, pp. 417-424, 1993.

[14] A. Thompson, I. Harvey, and P. Husbands, "Unconstrained evolution and hard consequences," *Toward Evolvable Hardware*, Springer-Verlag, pp.136-165, 1996.

[15] Xilinx, *The Programmable Logic Data Book*, San Jose, CA, 1996.

[16] Tetsuya Higuchi, Hitoshi Iba, Bernard Manderick, *Evolvable Hardware: Massively Parallel Artifical Intelligence*, pp.398-421, MIT Press, 1994.

[17] Dong-Wook Lee; Chang-Bong Ban; Kwee-Bo Sim; Ho-Sik Seok; Kwang-Ju Lee; Byoung-Tak Zhang, "Behavior evolution of autonomous mobile robot using genetic programming based on evolvable hardware," *Systems, Man, and Cybernetics, 2000 IEEE International Conference on, vol.5* , pp: 3835 -3840, 2000.

[18] Paul Layzell, "Reducing hardware evolution's dependency on FPGAs," *Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, MicroNeuro '99, Proceedings of the Seventh International Conference on*, pp. 171 -178, 1999.

[19] Sanchez, E. "Field Programmable Gate Array(FPGA) Circuits, " *LNCS-Towards Evolvable Hardware*, Vol. 1062, Springer-Verlag, pp1-18. , 1966.

[20] H. De Garis. "Cam-brain the evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolves at electronic speeds inside a cellular automata machine(cam),", *Towards Evolvable hardware: The evolutionary engineering approach, volume 1062 of Lecture notes in computer science*, pp. 76-98.Springer-Verlag, 1966.

[21] Ziv Kohavi, *Switching and Finete Automata Theory*, New York, McGraw-Hill Inc., 1978.

[22] Natarajan, B.K., *Machine Learning: A Theoretical Approach*, Morgan Kaufmann, 1991.

[23] H. Mhenbein, "Parallel genetic algorithms, population genetics, and combinatorial optimization", *in Proc. Third Int. Conf. Genetic Algorithms: Morgan Kaufmann*, pp416-421, SanFransisco, 1989.

[24] Mehrdad Salami, Greg Cain, "Multiple genetic algorithm processor for the economic power dispatch problem," *Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp188 193, GALESIA, 1995

[25] B. C. H. Turton and T. Arslan, "A parallel genetic VLSI architecture for combinatorial real-time application-disc scheduling," *Proc, IEE Colloquium on Genetic Algorithms n Image Processing and Vision*, pp 1-6, 1994.

[26] O. Coudert, "On solving covering problems," *Proc. 33rd Design Automation Conf.* P

**Kwee-Bo Sim**

Kwee-Bo Sim received his B.S. and M.S. degrees in the Department of Electronic Engineering from Chung-Ang University in 1984 and 1986 respectively, and Ph.D. degree in the Department of Electrical Engineering from The University of Tokyo, Japan, in 1990. Since 1991, he has been a faculty member of the School of Electrical and Electronic Engineering at Chung-Ang University, where he is currently a Professor. His areas of interest include artificial life, neuro-fuzzy and soft computing, evolutionary computation, learning and adaptation algorithms, autonomous decentralized systems, intelligent control and robot systems, artificial immune systems, evolvable hardware, and artificial brain etc. He is a member of IEEE, SICE, RSJ, KITE, KIEE, KFIS, and ICASE Fellow.

Phone : +82-2-820-5319
Fax : +82-2-817-0553
E-mail : kbsim@cau.ac.kr
http://alife.cau.ac.kr / http://alife.cau.ac.kr/kbsim/kbsim.html

**Fumio Harashima**

Fumio Harashima received B.S., and M.S. and Ph.D. degrees all in Electrical Engineering from University of Tokyo in 1962, 1964 and 1967, respectively. He was employed as Associate Professor at Institute of Industrial Science, University of Tokyo in 1967, and had been Professor from 1980 through 1998. He was Director of the Institute from 1992 to 1995. He was President of Tokyo Metropolitan Institute of Technology since April, 1998 through March 2002. He is currently Professor at Tokyo Denki University. He is also Visiting Professor at KAIST(Korea) from 2002 to 2003. He was elected President of Tokyo Denki University with 4 years term starting June 15, 2004. He is also Professor Emeritus of University of Tokyo since April, 2000.

His research interests are in power electronics, mechatronics and robotics. He is a co-author of four books and has published over 1,000 technical papers in these areas. He has been active in various academic societies such as Institute of Electrical Engineers of Japan, Instrument and Control Engineers of Japan (SICE), Robotics Society of Japan and IEEE. He has served as President of IEEE Industrial Electronics Society in 1986-1987, and 1990 IEEE Secretary. He was also a member of IEEE N&A Committee in 1991-1992, and IEEE Fellow Committee in 1991-1993. He served as Founding Editor-in-Chief of IEEE/ASME Transactions on Mechatronics in 1995. He is currently Editor-in-Chief of the IEEE Transactions on Industrial Electronics. He served as President of IEE of Japan in the year 2001-2002.

He has received a number of awards including 1978 SICE Best Paper Award, 1983 IEE of Japan Best Paper Award, 1984 IEEE/IES Anthony J. Hornfeck Award, 1988 IEEE/IES Eugene Mittelmann Award and IEEE Third Millennium Medal. He is a Fellow in IEEE and SICE.

http://homepage3.nifty.com/Fumio-H/