An Evolvable Network Controller Model for Reconfigurable Network Devices

Ramtin Raji Kermani, Fariborz Sobhan manesh

***Abstract*— The concept of Evolvable Hardware has attracted increased attention because it offers adaptive and highly optimized systems. Also there have been some researches toward implementing reconfigurable network devices that can change their architecture while running as a network node. In combination with evolvable hardware concept we can build adaptive and self-optimizing network systems which can autonomously adapt themselves with dynamically changing network conditions by means of evolutionary algorithms.**

In this paper a new model for a high-performance and flexible implementation of evolvable network controllers is proposed. Modularity of network controllers is used to decrease the complexity of such a system and as a case study we have focused on designing an Evolvable CRC generator module.

***Index Terms*—Adaptive networks, Evolvable hardware, FPGA, Genetic algorithm, Reconfigurable network devices.**

I. Introduction

One of the most important and rapidly growing trends in the development of networking technology is the ever-increasing demand by users for more functionality from their network devices, including higher data rate, quality-of-service routing, better security, multicasting, etc. [1]. Run-time flexibility in network devices could increase the flexibility of those devices while making them suitable for next generation networks and the networks with nodal processing capabilities.

While designing a network controller, the designer cannot predict the situations in which the controller would start malfunctioning. Then not all scenarios are considered in design time. Evolvable hardware lets our device adapt itself autonomously with dynamically changing network conditions just like it is exactly tailored for that specific situation. In this paper a preliminary model for applying hardware evolution concept to network controller architectures is proposed and cons and pros of such a model are discussed.

This paper is organized as follows. In the next section, some background on different issues including the concept of hardware evolution and reconfigurable networks are presented and an overview about the related research is given. In section three, the main idea about Evolvable Network Controllers (ENC) and our proposed model is brought up. Finally section four concludes this paper and possible future work is discussed.

Manuscript received January 7, 2008.

Ramtin Raji Kermani is a senior student of computer engineering in Computer Science and Engineering department of Shiraz University. His main fields of research are Hardware Design, Reconfigurable Computing, Evolvable Hardware and Adaptive Data Communication (Phone: +98-913-3276533; e-mail: ramtinraji@cse.shirazu.ac.ir).

Fariborz Sobhan manesh is assistant professor in Computer Science and Engineering department of Shiraz University. His main areas of research are FPGA, Computer and Signal Processing Implementable Architecture design (e-mail: sobhan@shirazu.ac.ir).

II. Background

In this section the main ideas behind hardware evolution are presented and the benefits of such devices are described. Then we have reviewed current researches on reconfigurable network devices.

A. Evolvable Hardware

Evolvable Hardware (EHW) refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment [2]. The objective of evolvable hardware is the autonomous reconfiguration of hardware structure by means of evolutionary algorithms such as genetic algorithms [3].

The key concept behind hardware evolution is to regard the configuration bits of programmable hardware devices (mostly FPGAs) as the chromosomes of Genetic Algorithms (GA). The architecture programming bits of an FPGA refer to those bits that specify its logic function and interconnections.

By designing an appropriate fitness function and performing the evolution process (e.g. reproduction, crossover) on an initial population (i.e. initial circuits) and then evaluating next generation chromosomes, we may obtain an optimal solution for the circuit.

It is worth mentioning that evolvable hardware is fundamentally different from the hardware implementation of evolutionary algorithms in which the hardware architecture does not change and is used to implement GA functions [2]. Evolvable systems could be used for fault-tolerant systems, auto diagnosis and repair systems and adaptive systems.

In Fig. 1 a programmed PLA along with its architecture bits is shown. There are two output functions Y0 and Y1 declared by these bit strings. Different variations of the configuration bit string will result in different circuits and therefore different output functions. In this approach a limited number of designs based on some presumptions are defined in design time and depending on the network operation conditions, one of them is downloaded into the device. It should be noted that reconfiguration is done by a human operator while the device is not operating.

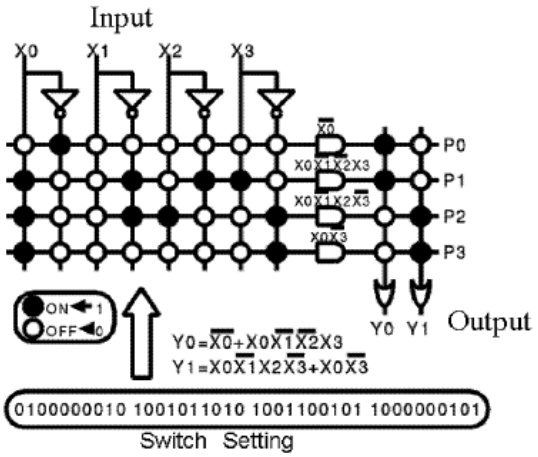

Fig. 1. Reconfiguration bits of a PLA [3]

Another approach is to consider this bit string as a genetic representation in the solution domain which in our case is the desired circuit. By applying genetic algorithms to configuration bits, we may evolve our circuit and get the best possible bit stream which results in an optimized circuit (Fig. 2).

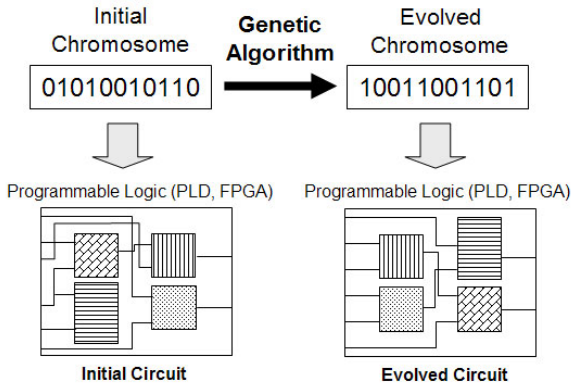

Fig. 2. Basic idea of Evolvable Hardware

To achieve evolvable hardware we need an evolutionary algorithm, a circuit problem to be optimized and a technology [4]. In our model, we are going to design a model for adaptive network controllers. As we will see, Xilinx Virtex FPGA families are a good choice for the technology and because all communicating end-systems need to cooperate on the evolution and evaluation processes, we could use co-evolutionary genetic algorithms.

There are three main methods for achieving evolvable hardware: Extrinsic, Intrinsic and complete hardware evolution (CHE) [5]. In our model CHE is chosen, in which the complete evolution process is located on the same chip as the evolving circuit. In CHE the evaluation of each individual circuit design is obtained by physical implementation of that circuit on the target device [7]. This method is used as an alternative to software simulation in Intrinsic method.

One major disadvantage of hardware evolution is to produce quite unconventional designs that are very difficult to be perceived and debugged by human designers. So we need to consider methods to take fault detection and auto repair properties of such systems into account. This matter is beyond the scope of this paper.

B. Reconfigurable Network Devices

The function of a network controller is to gather data packets that are addressed to it off a particular network medium and make them available to application software [6]. It might perform some kind of operation on data packets such as packetizing and depacketizing using deferent protocols. A network controller could be implemented as an instruction based processor, a dedicated hardware on ASICs or on programmable devices such as FPGAs. One of the most important benefits of the last one is that we may use the feature of partial reconfiguration that would be necessary for evolvable systems. Using FPGAs makes our device as flexible as processors with the same performance as ASICs.

In the following subsections two major researches toward implementations of network devices on FPGAs related to our model are presented. The first one is a partially reconfigurable Ethernet controller, while the other has tried to utilize the hardware evolution in its system.

1) Partially Reconfigurable Network Controller

There have been some researches and implementations of reconfigurable network devices that try to implement some lower layers of OSI model on ASICs and some higher layers in FPGAs. Most of them implemented PHY on ASICs which gives the system a high performance and Data Link layer and Transport layer on FPGAs to achieve a high degree of flexibility [9, 11].

Chaubal in [6] has proposed and implemented a partially reconfigurable network controller on IIM7010 Ethernet module platform which utilizes Xilinx Virtex FPGAs.

Partial bitstreams that can configure certain channels of the network controller without affecting the functioning of others have been created. Experiments have been performed that quantify the manner in which the performance of the controller can be changed by loading these partial bitstreams onto the FPGA. Fig. 3. Illustrates the design of the network controller and compares it to previous implementations.

In this design, he has implemented Transport layer of TCP/IP protocol suite into FPGAs and a small part into ASICs. He has achieved an acceptable increase in speed while keeping the system flexible.

2) Evolvable Reconfigurable Internet Platform

Research focused on the development of an evolvable Internet hardware platform is also being carried out at the Applied Research Lab at Washington University [10]. The Field Programmable Port Extender (FPX) system uses FPGAs to allow reconfigurable hardware modules to be dynamically installed into network devices. An FPX module contains two FPGA devices. One of these is a static Network Interface Device (NID) FPGA, which is a Xilinx Virtex XCV600E part. The other is a single Reconfigurable User Application Device (RAD), which is a Xilinx Virtex XCV1000E part.

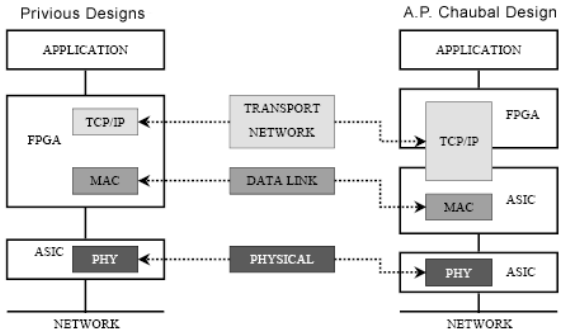

Fig. 3. Two different approaches on Network controller design

The FPX modules are inserted into an existing network stream to provide user-specified traffic processing including routing, buffering, and packet content modification. Recent work on this system has included the development of an interface that allows FPX modules to be programmed remotely via a TCP/IP network.
Although this system is capable of implementing various design strategies and configurations, it doesn't fully utilize the basic idea of hardware evolution.

III. Evolvable Network Controller (ENC)

In this section the advantages of hardware evolution in network devices which are known as complex systems is introduced. Then regarding the benefits of hardware evolution and design constraint, a model for incorporating run-time hardware evolution is proposed.

A. The use of run-time evolution in network devices

A network device or simply a network controller functioning inside a network may be dedicated to perform some functions on data packets or bit streams. A network interface card (NIC) for example, is devoted to get data from upper network layers of a host computer and after doing some manipulation, such as encryption, would pass the processed data appropriately to the network it belongs to via a standard network interface. The internal architecture of most NICs is usually modular.

In real world applications, digital systems are always functioning in a noisy, unreliable, dynamically changing and somehow randomly varying condition. We may use the reconfigurable approach to change the functionality of our system if and only if we are aware of the future possible changes in network condition and system parameters. But what would be the solution if we cannot precisely predict changes? We need to incorporate some kind of intelligence into our system. In this way, our network controller would have the ability to recognize all changes and adapt itself to them.

One approach is to take advantages of Evolvable Hardware concept in our network device. The following shows the evolution process of an adaptive ENC (using CHE method) [3, 13, 14].

1. If there is a major change in network conditions (Network traffic, flow, high congestion, etc.), go to step 2, otherwise go to step 1.
2. Start with a randomly generated of N L-bit chromosomes (representing candidate circuits)
3. Calculate the fitness measure f(x) of each chromosome x in the population in the following manner:
    a. Physically implement each individual circuit on the FPGA.
    b. Apply a test (training) input and check out the corresponding output. (Test input is achieved)
    c. Depending on how close is the real output to the desired one, assign each chromosome (i.e. circuit design) a deserving fitness value.
4. Select individuals with higher fitness values to reproduce the next generation.
5. Breed new generation through crossover and mutation (genetic operations) and give birth to offspring.
6. Evaluate the individual fitnesses of the offspring
7. Replace worst ranked part of population with offspring.
8. While the terminating condition is not satisfied, go to step 3.
9. If terminating condition is true, we have reached the best solution. Download the configuration bits into the FPGA and put the system back to work.

As you can see, the main problem in designing an EHW is to find an appropriate fitness function to evaluate each individual circuit designs to find out how "Good" is that design. Unfortunately as the complexity of our system grows, the length of FPGA configuration bits would increase and the evolution process will take much longer processing time. Also finding an appropriate fitness function for a complex system would be a very difficult task.

To overcome this problem in complex EHWs we have two choices:

- Using function level evolution, instead of bit-level evolution [15]
- Take advantage of the complex systems modularity.

In the first approach, we are not dealing with zeros and ones. Instead, we try alternative functions of hardware as chromosomes. For example we would try the following alternatives as input/outputs of a mathematical calculation [15]:

- Input: (a,b) Function: Sin (a) + Sin (b)
- Input: (a,b) Function: Cos (a) + Sin (b)
- Input: (a,b) Function: Tan (b) + Sin (b)
- …
- Input: (a,b) Function: Sin(a) – Cos (~b)

In the second approach, we would try to utilize divide and conquer strategy to design evolvable modules, instead of a complete evolvable system. Since the computation of the CRC is one of the most computational extensive operations performed by a network controller, in our case, we are going to show how to make the CRC (Cyclic Redundancy Check) generator module evolvable. However because of the standard feature of communication protocols, many

constraints should be set. To overcome this issue, we would use function level evolution. Furthermore we will have a limitation on chromosomes length while designing the GA processing unit. A small circuit like CRC generator could be presented by a relatively short configuration bit string.

B. *ENC architecture*

Any well designed complex system including a network interface card, built upon Ethernet, comprises different functional modules that interact to accomplish a specific job. These modules have to do some processes on incoming and outgoing data units of MAC and LLC sub layers of Data Link Layer such as:

- Segmentation, fragmentation, and reassembly
- Error detection and correction
- Queuing
- Packetizing and depacketizing
- Frame and protocol demultiplexing, etc.

Defining a complete fitness function for the complete system would be a complex task. Instead, we divide our system into some sub systems and try to make each one Evolvable and then combining them as a whole. In the following subsections we have concentrated to show how to make a CRC generator Evolvable.

1) *Cyclic Redundancy Check Generator*

A typical Ethernet frame has six fields and the last field is the Frame Check Sequence (FCS). An FCS refers to the extra checksum characters added to a frame in a communication protocol for error detection and correction [16].

One popular solution for error detection and generating FCS field in Ethernet is Cyclic Redundancy Check (CRC). In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission.

Preamble	Dest. Address	Src. Address	Type	Data	FCS
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46 - 1500 Bytes	8,16, 32 Bytes

Fig. 4. A typical Ethernet frame

The CRC is calculated based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial by another. In CRC, original data is treated as a polynomial. This procedure can be described by (1) [17].

$$CRC = rem[M(x) \times \frac{x^n}{G(x)}] \quad (1)$$

Where M(x) is the original data, G(x) is the generator polynomial and *n* is the degree of G(x). Depending on different strategies, G(x) could be a polynomial of various orders. CRC hardware implementation is based on a loop-back shift register with several XORed taps (Fig. 5). After the related CRC is generated using software or hardware implementations, it is appended to the original data, forming the Codeword to be transmitted.

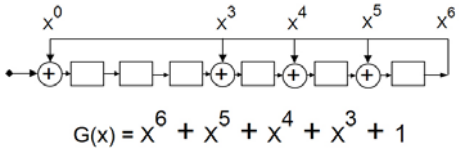

Fig. 5. Typical CRC hardware with G(x) as generator polynomial

CRC generation has been implemented in the following schemes [17]:

1. Software solution
2. Serial and parallel ASIC solution
3. LUT based solution

The LUT implementation enables some configurability since it is possible to change the polynomial by changing the content of the LUT memory. Nordqvist et al. in [17] have implemented and manufactured "Radix-32 Configurable CRC Unit", a reconfigurable CRC generator. By noticing that any polynomial of a fixed length can be represented by implementing the CRC using a LSR with switches on the loop back as illustrated by fig. 6, a configurable hardware can be implemented using NAND-gates to implement the switches.

This allows us to change the polynomial G(x) of a given length L by storing a bit description of the polynomial in a register. However some protocols use CRC with different constraint length L [17].

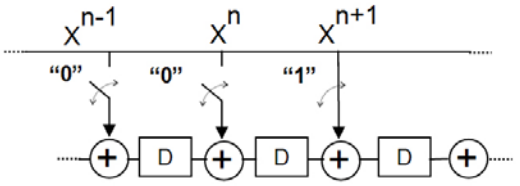

Fig. 6. The architecture of Radix-32 configurable CRC generator [17]

Different configurations that are stored in the configuration register, can build a specific CRC module with different generator polynomials. We regard this register as our target chromosome and let it evolve until the best possible generating polynomial for that specific network condition is obtained. Fig. 7 shows the overall architecture of Evolvable CRC generator. In this scheme we've utilized a GA processing unit (GPU) based on the system proposed in [18, 19].

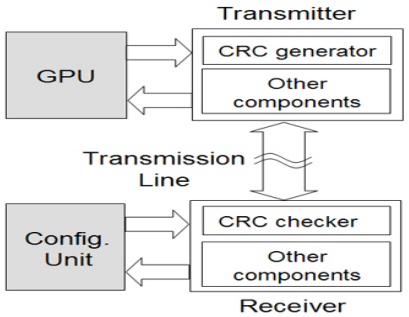

Fig. 7. The overall architecture of two ENC communicating

The scenario is as follows: Each system is configured with a primary configuration. If there is a detected major change in critical parameters or network conditions, a signal is sent to the GPU. After fetching the previous configuration from the memory and comparing previous and current parameters, an evolution cycle is started by the GPU. After reaching a limit, the new optimized and adapted configuration is downloaded into the reconfigurable device.

In our case, evaluating the CRC generator architecture is performed by applying a training input string to the CRC generator and sending the resulting codeword to receiving device. The receiver then checks the codeword such that the closer the sent FCS and the calculated FCS are, the higher fitness value is assigned to the current configuration of the CRC generator.

As Kajitani in [18] shows, the best Genetic Algorithm suitable for such systems is Linking Elitist Recombination (LER). In LER two parents are selected and after cross over and mutation operations, two children are made. Finally two best fit chromosomes among two parents and two children are selected as next generation parents [20].

2) Reconfigurable GA processing unit (GPU)

Kajitani et al. in [18] have proposed a model for implementing GA operations on a single LSI chip. In their model the complete hardware evolution is performed by a reconfigurable hardware (GPU) (Fig. 8).

Torressen and Glette in [19] also have introduced a method for implementing an on-chip evolution system on Xilinx Virtex II chips. Their design allows for a rapid processing of the time consuming parts in hardware and leaving other parts to more easily modifiable software. The GPU used in our model is closest to the first model proposed by Kajitani.

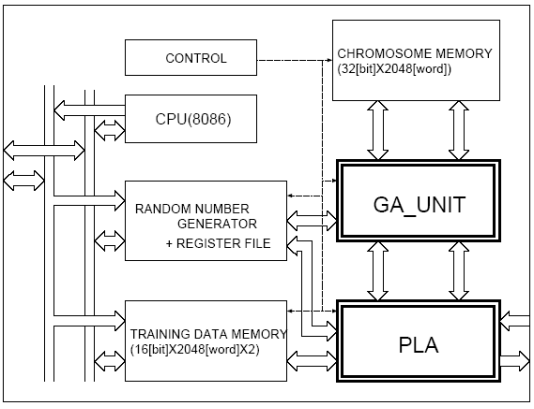

Fig. 8. GPU proposed by Kajitani et al.

3) The proposed Evolvable CRC generator architecture

Unfortunately EHW are not scalable while implementing complex and large systems. The more complex the circuit is, the more time it takes to evolve it. To overcome this problem, Torresen in [8] has proposed some solutions. For network controllers, the best possible approach is to use Multi-FPGA systems (MFS). MFS systems utilize multiple FPGAs, connected in a fixed pattern, to implement complex logic structures. In our model we could use a MFS with two Xilinx Virtex FPGAs, one for implementing the GPA and the other as the target for the main circuits of ENC. Any EHW needs run-time and partial reconfiguration which Virtex family of Xilinx FPGAs fully supports these features [12]. Fig. 9 shows the complete design of our proposed system.

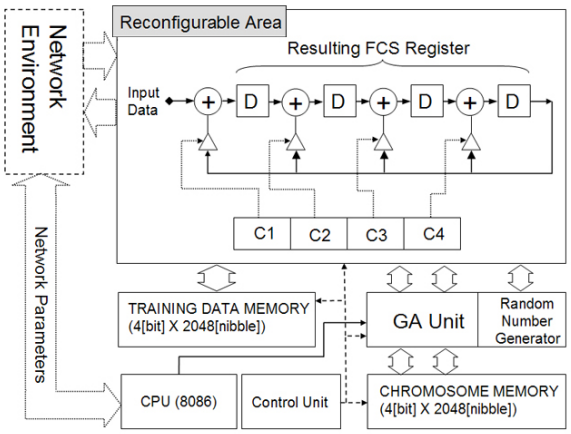

Fig. 9. The proposed Evolvable CRC generator architecture

The proposed architecture is composed of six functional blocks: Reconfigurable area, GA unit, Training data memory, Chromosome memory, CPU and control unit. The detail of each functional block is as follows:

Reconfigurable area: While the other blocks have a fixed architecture, this block is reconfigured in run-time, based on partial reconfiguration capability of some FPGAs. It reads a 4bit chromosome from the GA unit and implements them. Also the final optimized circuit is to be implemented in this area. A configurable CRC generator circuit is implemented using tri-state buffers. By applying different nibbles to C register, we may change the generating polynomial.

GA unit: This block reads two chromosomes from the Chromosome Memory and initiates crossover and mutation operations on them to make children.

Chromosome memory: A memory for the chromosomes of all individuals. The maximum length for each chromosome is 4 bits, and the maximum population is 2048.

CPU: This 16 bit processing unit is the interface between outside and inside the chip. It also gathers critical parameters (e.g. traffic, data flow, etc.) from the network and determines when an evolution is necessary to adapt to the current network conditions.

Control Unit: Sets the control signals for all functional blocks.

Training data memory: Memory used for storing test data. For evaluating each generation, after implementing an individual, an appropriate test input is applied to the circuit and the results determine how good the circuit is functioning.

Because we have summarized our system into a CRC generator, the system is not complex and only one FPGA would suffice. But in case we are designing a complete system, we need to use Multi-FPGA systems to let our design preserve its scalability.

IV. Conclusions and Future Work

In this paper a model for utilizing hardware evolution for network devices were proposed. Benefits of a network

controller to be evolvable were mentioned and it was tried to take advantages of complex systems modularity and design an Evolvable CRC generator module. It was also described how a particular module could be designed to be evolvable and adaptive. As future works, one may try to design an overall fitness function for a complete complex system or all internal modules separately.

ACKNOWLEDGMENT

Hereby authors would like to acknowledge professor Kajitani (University of Tsukuba, Japan), Professor Torresen and Kyrre Glette (University of Oslo, Norway) for their kind cooperation, useful guides and providing us with their precious papers.

REFERENCES

[1] David C. lee, et.al, "Reconfigurable Routers: A New Paradigm for Switching Device Architecture"
[2] X. Yao, T. Higuchi, "Promises and challenges of evolvable hardware", Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware, 1996
[3] Tetsuya Higuchi, YongLiu, Masaya Iwata, Xin Yao, "Introduction to Evolvable Hardware", Book Chapter: "Evolvable Hardware", Springer, 2006
[4] P. Haddow, G. Tufte, P. Van Remortel, "Evolvable hardware: pumping life into dead silicon", On Growth, Form and Computers, 2003
[5] H De garis, "LSL evolvable hardware workshop report", ATR, Japan, Tech. Rep., Oct, 1995
[6] Aditya P. Chaubal, "Design and Implementation of an FPGA-based Partially Reconfigurable Network Controller", M.Sc. thesis, Virginia Polytechnic Institute and State University, 2004
[7] P Haddow, G Tufte, "Prototyping a GA Pipeline for Complete Hardware Evolution", Proc. The First NASA/DOD Workshop on Evolvable Systems, 1999
[8] J. Torresen, "A Scalable Approach to Evolvable Hardware", Genetic Programming and Evolvable Machines, 2002 – Springer
[9] R. Jaganathan, K. Underwood, and R. Sass, "A Configurable Network Protocol for Cluster Based Communications using Modular Hardware Primitives on an Intelligent NIC," 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM 2003), 2003
[10] J. Lockwood, "Evolvable Internet Hardware Platforms," Third NASA/DoD workshop on Evolvable Hardware, pp. 271–279, July 2001
[11] P. Bellows, J. Flidr, T. Lehman, B. Schott, and K. Underwood, "GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing," 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2002), 2002
[12] Xilinx Inc., ISE software manuals, 2005
[13] Melanie Mitchell, "An introduction to genetic Algorithms", The MIT press, ISBN 0−262−13316−4, 1999
[14] Wikipedia the free encyclopedia, "Genetic Algorithm", http://en.wikipedia.org/wiki/Genetic_algorithm, 2007
[15] Tetsuya Higuchi et al., "Evolvable hardware at function level", IEEE International Conference on Evolutionary Computation, 1997
[16] Wikipedia, "Frame Check Sequence (FCS)", http://en.wikipedia.org/wiki/Frame_Check_Sequence
[17] Ulf Nordqvist, Tomas Henrikson and Dake Liu, "Configurable CRC generator", Proceedings DDECS, Brno, Tjech Republic, pp. 192-199, Apr 2002
[18] I. Kajitani et al., "A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI", Proc. 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware, 1998
[19] K. Glette, J. Torresen, "A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device", Proc. 6th Int. Conf. on Evolvable Systems: From Biology to Hardware, 2005
[20] Dirk Thierens, D. E. Goldberg, "Elitist recombination: an integrated selection recombination GA", In proceedings of First conference on Evolutionary Computation, page 508-512, 1994