SELF-RESTORATION MECHANISM FOR RUN-TIME RECONFIGURABLE DATA-STREAM PROCESSORS

by

Irina Terterian, B.Sc., B.Ed.,

State University of Georgia,

Tbilisi, 1978

A thesis

presented to Ryerson University

in partial fulfillment of the

requirement for the degree of

Master of Applied Science

in the program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2004

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institution or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Borrower's Page

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Name	Signature	Address	Date

Self-Restoration Mechanism for Run-Time Reconfigurable Data-Stream Processors

Irina Terterian,
Master of Applied Science, 2004
Electrical and Computer Engineering,
Ryerson University

ABSTRACT

The cost of a hardware failure in high-performance computing systems is usually extremely high because of the system stall where billions of operations can be lost within one second. Thus, implementation of self-restoration mechanisms is one of the most effective approaches to keep system performance on a required level.

The project presents a new approach, which allows retaining the performance of the Run-Time Reconfigurable stream processing system on its maximum level. This becomes possible by development of multi-level self-restoration mechanism that consists of: restoration by FPGA-scrubbing, restoration by FPGA-slot replacement and restoration with optimum performance degradation. All above levels of restoration procedure were developed and tested on reconfigurable computing platform based on XILINX Virtex FPGA. Analysis of achieved results of the developed mechanism shows a very fast restoration of functionality and dramatic increase of lifetime of FPGA based computing platforms.

Acknowledgements

The author would like to thank her supervising professor, Dr. Vadim Geurkov, for providing his guidance, knowledge and support. The author would also like to thank the members of the review committee for their participation. The author would like to thank the National Science and Engineering Research Council of Canada (NSERC) for providing funding support for this research project in the form of multiple grants to my supervising professor. The author would like to thank Ontario Graduate Scholarships (OGS) for funding this research through a scholarship.

Table of Contents

1 . INTRODUCTION.....1

1.1 . Motivation..... .1

1.2 . Major principles of the proposed approach.....3

1.3 . Original Contributions.....4

1.4 . Thesis Organization.....6

2. REVIEW OF RECONFIGURABLE COMPUTING PLATFORMS.....9

2.1 Instruction level reconfigurable architectures.....10

2.2 Task level reconfigurable architecture.....11

2.3 Segment level reconfigurable architecture.....12

3. RADIATION TOLERANCE OF THE RCP 14

3.1 Radiation Damage of the Semiconductor Devices.....14

3.2 Radiation Effects on Field Programmable Technologies 17

4. FAULT DETECTION AND RECOVERY IN FPGA DEVICES.....20

4.1 Fault detection and location.....21

4.2 Fault recovery.....25

5. ORGANIZATION OF MULTI-TASK RECONFIGURABLE STREAM PROCESSOR WITH SELF- ASSSEMBLING MUCRO-ARCHITECTURE.....34

5.1 Organization of Virtual Hardware Components (VHC) 36

5.2 Application Specific Virtual Processor assembling procedure.....38

5.3 Re-configurable Parallel Stream Processor Architecture Organization.....42

6. MECHANISM FOR ASVP SELF-RESTORATION.....47

6.1 VHC scrubbing..... 49

6.2 CLB-column (slot) replacement.....50

6.3. ASVP functional restoration with performance degradation.....51

6.3.1. ASVP restoration with minimum restoration time.....52

6.3.2. ASVP restoration with minimum performance degradation.....53

7. IMPLEMENTATION OF THE ELEMENTS OF SELF_RESTORATION MECHANISM AND ANALYSIS OF THE RESULTS.....63

7.1 Test Platform.....63

7.2 Implementation of procedures for restoration without performance degradation 67

7.3. Implementation of procedures restoration with performance degradation ...69

7.3.1 Specifications for ASVP: "Hamming Distance Calculator" 70

7.3.2. Hamming Distance Calculator Architectural Representation..... 72

7.3.3 Performance Estimation Model.....74

7.3.4.Architecture-to-Task Optimization for Hamming Distance Calculator.....76

8. CONCLUSIONS AND FUTURE WORK..... 82

REFERENCES.....89

LIST OF ACRONYMS.....99

List of Tables

Table 5.1: Code of associated ASVP

Table 5.2: VHC-core address conversion table

Table 7.1: Experimental results of functional restoration time and acceleration of functional restoration comparing with entire FPGA re-programming

List of Figures

Figure 4.1 A simplified architecture of Adaptive Re-Configurable Platform..... 21

Figure 4.2 Architecture of a Field Programmable Gate Array.....22

Figure 4.3 A covering Graph for a fault tolerant FPGA.....26

Figure 4.4 Testing Programmable Logic Blocks.....33

Figure 5.1 Structure of partially reconfigurable Xilinx "Virtex" FPGA device. .37

Figure 5.2 Micro-architecture of a Virtual Hardware Component 38

Figure 5.3 Data Flow Graph of stream processing task.....39

Figure 5.4 ASVP architecture assembled in the FPGA from pre-compiled VHCs reflecting task DFG 41

Figure 5.5 Architecture of Re-configurable Parallel Stream Processor (RPSP) for multi-task & multi-mod workload.....43.

Figure 5.6 Code of a task = Code of associated ASVP.....45

Figure 5.7 VHC# to VHC-core address conversion table.....45

Figure 6.1 VHC replacement using spare CLB-slots..... ..51

Figure 6.2 Graphical presentation of a Virtual Hardware Component - *R i, j*....54

Figure 6.3 Architecture Configurations Graph..... 54

Figure 6.4 ACG hierarchic arrangement.....56

Figure 6.5 Selection of the optimal variant of ASVP processing architecture on the partially arranged ACG.....58

Figure 6.6 Architecture Configurations Graph and performance diagram associated to ASVP variants.....60

Figure 7.1: Block diagram of the Reconfigurable Parallel Stream Processor

Figure 7.2: Reconfigurable Parallel Stream Processing platform

Figure 7.3: Reconfigurable PCI-interface Module

Figure 7.4: VHC with corrupted interconnection on its micro-architecture

Figure 7.5: Spectral scanning on the Earth surface by the array of spectral sensors

Figure 7.6: Data Flow Graph for Hamming Distance Calculation

Figure 7.7: Schematic diagram of resource ***R1,*** Variant ***R1,1***: 8- XOR channels & 8-bit

Figure 7.8: Architecture Configurations Graph (ACG) for HD-calculator variants Adder

Figure 7.9: Hamming distances pipeline computational process of spectral and code vectors on the variant of calculator presented in Figure 5.2

Figure 7.10: Arrangement of architecture variants of Hamming Distance Calculator in order to values of data processing time.

CHAPTER 1

INTRODUCTION

1.1 Motivation

In many areas of application: video and image processing systems, high-speed communication, digital TV and audio broadcasting, data encryption and compression, multimedia and DSP the major requirement is processing of high volume of streamed data with very high performance. Recently, these performance requirements are in a range of 10-100 Giga bits per second. Usually, for all of the above areas of application the Application Specific Processors (ASP) are used. Most of these computing systems based on Application Specific Integrated Circuits (ASIC) are logic devices with a *fixed architecture* optimized for tasks algorithm and data structure. Within recent decade, the complexity of ASICs has increased in orders. That is why the probability of hardware fault in ASIC based computing platforms also has increased. On the other hand, in many applications such as: aerospace, nuclear power stations, military and some others the data-stream processing systems must reliably perform their functions in radiation intensive environment. The usual approaches of increasing system reliability based on modular or information redundancy as well as shielding technique is very expensive and power consumable. Simultaneously in most of the mission critical applications such as: satellite platforms, process control equipment in nuclear power stations, planetary exploration systems, earth orbital stations and many military applications hardware failure of even one system gate (out of millions) can cause incredible damage or even lost of very expensive systems. Even in most of commercial stream

processing systems such as Digital Video Broadcasting (DVB) and Digital Audio Broadcasting (DAB) every minute of broadcasting interruption costs many thousands of dollars.

The promising approach to increase system reliability for complex and mission critical computing systems is utilization of reconfigurable computing systems based on Field Programmable Array (FPGA) devices. This approach assumes ability to configure the application optimized processing micro-architecture by programming logic functions and on-chip interconnections into special Static Random Access Memory (SRAM) incorporated to the FPGA as a circuit configuration memory unit. This ability gives this system a unique feature for restorating its functionality simply by re-programming a configuration file into the configuration SRAM. In the case when one or more logic gates are damaged, micro-architecture can be redesigned to avoid faulty logic gates and loaded then into the FPGA. Thus, this approach allows restoration or even modification of system functionality by remote reprogramming of the FPGA's configuration SRAM. However, several problems still exist for the FPGA based computing platforms. Firstly, configuration SRAM itself can be affected by radiation and / or power instability. Secondly, it can take a lot of time for re-designing on-chip micro-architecture avoiding faulty logic cells. In many practical applications long restoration time means complete mission fault.

Instead, in this paper the novel approach for restoration of the FPGA based reconfigurable computing system is presented. This approach allows *self-restoration* of on-chip processing micro-architectures of stream processing logic circuits within very short period of time (hundreds of microseconds). A multi-level self-restoration mechanism developed in the capacity of this project can itself make decision what kind of restoration

procedure has to be implemented to: a) minimize restoration time, b) minimize logic resources, c) minimize performance degradation if, there is no other way to restore functionality of a system.

1.2 Major principles of the proposed approach

The proposed approach is based on the following major principles:

1. Utilization of partially reconfigurable FPGA devices as a component base of proposed self-restoration mechanism. This type of FPGA devices allows reconfiguration of small portions of on-chip logic circuitry without any interruption of data processing in the rest of FPGA device.
2. Utilization of Run-Time Reconfigurable computing platform as a system base of proposed self-restoration mechanism. This type of a computing platforms allows utilization of one part of system logic resources to restore other part of (damaged) logic without suspending of all computational processes running on a system.
3. Implementation of proposed method of self-assembling of processing micro-architectures on a base of the pre-compiled Virtual Hardware Components (VHC). This approach allows dynamic on-chip assembling of Application Specific Virtual Processors (ASVP). ASVP is a logic circuit with micro-architecture optimized for algorithm and data structure of a task to be

executed. On the other hand, if any hardware fault occurs restoration can be performed by virtual re-assembling of ASVP micro-architecture.

4. Organization of self-restoration mechanism as multi-level procedure. This approach reflects nature of possible hardware faults occurring in the SRAM based FPGA devices. Multi-level organization allows to react on each type of possible reasons with respective restoration procedure. Multi-level organization of self-restoration also reflects internal conditions in the FPGA. There can only be two possible situations: there are spare logic resources or all off them are in use. In case when there is no spare logic resources in the FPGA self-restoration procedure reaches the highest possible level of restoration – restoration with performance degradation.

5. Incorporation of Architecture-to-Task Optimization System (ATOS) - automated high-level architectural synthesis mechanism for optimization of the restored micro-architecture to the task algorithm and data structure reflecting limitations of available logic resources.

All of the above principles that are embedded into the proposed self-restoration mechanism allowed reaching very high cost-reliability, cost-performance parameters and increase system lifetime of high-performance parallel stream processing systems working in harsh and radioactive intensive environment.

1.3 Original Contributions

The thesis presents a new approach for self-restoration mechanism and methods for its implementation in Run-Time Reconfigurable data-stream processing systems based on

partially reconfigurable Field Programmable Gate Array devices. This approach allows fast and effective restoration of system functionality after temporary and permanent hardware faults. Temporary hardware faults in the SRAM based FPGA devices usually are caused by radiation effects (e.g. SEU - Single Event Upset). However, permanent hardware faults can be caused by: hidden manufacturing defects, corrosion, time-dependant dielectric breakdown, etc. The proposed self-restoration mechanism being multi-level is adaptive to the type of a hardware failure because it can apply strategies of different levels of restoration. For example, it tries first to recover from the fault assuming that fault is temporal. However if, after certain attempts fault still exists mechanism jumps to the next level of restoration using logic replacement procedures. It can happen that all spare logic resources are already in use when permanent hardware fault is detected. In this case, mechanism would jump to the level of restoration with performance degradation. On this level mechanism selects the best configuration of stream processing micro-architecture with minimum performance degradation.

In this regard the major contributions are as follows:

- Development of logic replacement technique based on re-addressing of Virtual Hardware Components to the spare slots in the partially reconfigurable FPGA devices (Xilinx Virtex-E, Virtex-2 and Virtex-2Pro families of FPGAs). This technique was presented in [1].
- Development of method for optimization performance degradation in case of a restoration of functionality with limited logic / routing resources. This method was presented in [2] .

- Analysis of radiation effects which can cause hardware failure in SRAM based FPGA devices and possible mitigation techniques for this effects. On a base of this analysis the complete multi-level hardware restoration mechanism was developed and presented in [3].
- Modeling and simulation of major elements of the proposed self-restoration mechanism were performed using ALTERA MAX Plus II and XILINX ISE 6.2i CAD systems. Results were presented in [3] and [4].
- Implementation and testing of major elements of the proposed self-restoration mechanism in the RTR Multi-Stream Processor AGORA-2 (Adaptive Group Organized Reconfigurable Architecture). Results were presented in [2], [3] and [4].
- Modeling of prototype of Hamming Distance Calculator with ability of restoration with optimum performance degradation. Analysis of results was utilized for preliminary study for development process of the new generation of on-board computing platform for Canadian satellite RADARSAT-2.

1.4 Thesis Organization

The thesis consists of eight Chapters. The rest of the thesis paper is organized as follows:

- Chapter two consists of the literature overview and classification of major types of existing reconfigurable computing systems. This chapter gives background on run-time reconfigurable computing platforms.
- The third chapter presents analysis of radiation factors that can cause hardware fault in the semiconductor devices. Also, a special section in this chapter gives literature survey on radiation effects in the SRAM based FPGA devices.

- Chapter four encompasses the analysis of existing methods of hardware fault detection, location and restoration based on extensive literature survey. This study was very important because it allows utilization of some existing methods of hardware fault detection and location. This component of fault tolerance for FPGA based computing platforms was not a target for this project. However, without understanding fault detection and location methods and procedures it would be difficult to develop fault restoration procedures.
- Chapter five describes architecture organization of multi-task reconfigurable stream processor with self-assembling micro-architectures of application specific computing circuits. This chapter gives background information for computing platform where the proposed self-restoration mechanism can be implemented. Self-assembling procedure and its application for self-restoration of the Application Specific Virtual Processor is also described in details in this chapter.
- Chapter six gives the overview of the proposed method. It consists of a full description of all components of the proposed self-restoration mechanism. Interaction between different levels of the self-restoration mechanism is also described. In this chapter the method of restoration with performance degradation also is presented. This chapter gives also background on architectural synthesis method used for the developed method of restoration.
- The implementation and analysis of test results are presented in Chapter seven. Brief description of test platform is given, as well as comparative analysis of test results of FPGA scrubbing and logic replacement procedures comparing with the existing approaches. The detailed explanation of Hamming Distance Calculator

(HDC) prototype implementation with ability for restoration with performance degradation is also presented in deep details. Analysis of results of modeling of prototype behavior also is given.

- Finally, conclusions and description of future work is given in the Chapter eight.

CHAPTER 2

REVIEW ON RECONFIGURABLE COMPUTING PLATFORMS

Recently more and more research and development groups and organizations working in the area of high-performance computing systems start to use Field Programmable Gate Array (FPGA) devices as a component base of their systems. Since 1995, number of types of FPGA-based computing platforms (R&D stage) had doubled [5]. The reason is that in many applications where the performance of a software implementation is not sufficient and the high expense and long design time of Application Specific Integrated Circuits (ASICs) cannot be afforded, FPGA devices has provided a very cost-effective solution. These new types of microprocessor-free architectures based only on reconfigurable devices are becoming more and more popular solution for high-performance computing platforms. Main areas of application for these platforms are digital signal processing, image processing and pattern recognition, digital video/ audio broadcasting and communication, high-performance data acquisition and control systems, compression / decompression and encryption / decoding devices.

Broadly all FPGA-based computing systems can be divided on two big categories:

1. Statically configurable systems and
2. Run-time reconfigurable systems.

The first approach is normally used for rapid prototyping and small production, where development of ASICs is too expensive and time-to-market is very short.

Run-time reconfigurable systems allow reuse of FPGA hardware during the lifetime of the task in the system. This gives a rise to a powerful computing paradigm where one or

more reconfigurable hardware components can serve as a virtual hardware space, similar in principle to virtual memory. Several architectural approaches were proposed and implemented in different reconfigurable computing platforms within the recent decade.

From system architecture viewpoint, run-time reconfigurable computing systems can broadly be classified into three categories associated with the level of computational process where re-configuration takes place:

1) Instruction level;

2) Task level and

3) Task segment level.

2.1 Instruction level reconfigurable architectures

These systems are based mostly on Dynamically Programmable Gate Array (DPGA) devices [6] and allow reuse DPGA parts (often called frames) for different instructions. This reconfiguration could be done in a very short time and allow implementation of a custom instruction set. The reconfigurable device in these systems usually tightly coupled with the processor to be used as the accelerator for computationally intensive and repetitive segments of a program. There are many examples of such systems [7], [8] and [9]. The Hybrid RISC reconfigurable architectures, where reconfigurable component is a part of CPU [9] and [10] also can be associated with this class of architectures. An advantage of this type of reconfigurable accelerators is that being implemented into common general purpose computing architectures they can dramatically increase CPU performance. However this performance still is limited by performance of a microprocessor coupled with FPGA. This microprocessor now becomes responsible for management of the reconfigurable component

and synchronization of two parallel processes in the microprocessor itself and in FPGA-based accelerator. It increases amount of extra control hardware or decrease performance of the microprocessor. Another disadvantage of this type of systems appears in a multiprocessing environment. Cost-performance parameters are very poor because of limitations for reconfigurable resource sharing among the processors due to the tight coupling.

2.2 Task level reconfigurable architectures

This type of systems usually contains a collection of configurable processing elements (functional units) communicating via a network of configurable switches.

The architecture of these systems has to be configured for the task. There is a big number of systems that fall in this category: RaPid [11], NAPA [12], Splash-2 [13] and many others. This type of architecture allows achieving very high performance because the data-flow graph of the task can be completely "covered" by hardware specifically tuned for task operations. However, the time for compiling a system can be very long and consequently it is very difficult or even impossible to employ these systems in applications where fast dynamic recompilation in real-time needed.

Another performance limitation comes from the communication network. Three main concepts were used in this type of systems: a) direct (point-to-point) connections between FPGA-based functional units (usually pipeline connections [11], [13]). This type of connectivity gives minimum propagation delay for the data transfer; b) crossbar switches (switching networks) [14], [15]. This type of connectivity gives higher propagation delay than direct connections, because signals have to pass a multi-stage multiplexing units; c) system buses usually used as a message passing networks for multi-processing

reconfigurable systems [16], [17]. Information transfer rate is the lowest comparing with previous types of interconnections but cost-effectiveness is usually much better.

2.3 Segment level reconfigurable architectures

These types of reconfigurable computing systems are based on the idea that a task can be divided on several segments when the processing time of each segment is equal or higher than FPGA reconfiguration time. In this type of systems the first temporal partition receives input data, performs computation and stores the intermediate result into on-board memory. FPGA-based functional unit is then reconfigured for the next segment, which computes results based on intermediate data from previous partition. Such temporally partitioned systems are called Run- Time Reconfigured (RTR) systems [18], [19]. RTR-systems allow to "program" cost-performance parameters. If it is necessary to reach maximum performance, all segments of the task can be mapped in different FPGA-based functional units. If performance requirements are low, task segments can be performed one after another on one functional unit. Thus, the concept of RTR systems allows achieving very high level of cost-efficiency. Another big advantage of this type of systems is that reliability of the system can be very high. If system architecture contains many uniform RTR modules and fault occurs in one module within task processing, system can eliminate fault module by reconfiguration. In worst case it can cause reduction of performance if no spare modules are available. Time delay in this case is equal to the segment processing time plus module configuration time and is in matter of tens of milliseconds. Thus, there is a direct relationship between performance and reliability for such type of systems when the module fault causes only performance reduction but not a fault of all system.

Proposed research was concentrated on the Run -Time Reconfigurable-computing systems because of high cost-performance and performance-reliability characteristics.

CHAPTER 3

RADIATION TOLERANCE OF THE RCP

One of the major problems for FPGA based computing systems as well as any other semiconductor devices is radiation tolerance. To mitigate this problem usually a special radiation hardened devices have to be utilized. However, high cost of such devices increases system price dramatically. Furthermore, radiation hardened semiconductor devices still have limited lifetime. Instead, as an alternative the recovery mechanisms can be implemented. This approach is conceptually impossible to implement in regular logic devices due to the lack of hardware flexibility. However, it is possible to implement it in reconfigurable logic, especially in SRAM based FPGA devices. To investigate this possibility of radiation effects in semiconductor devices including FPGA, the literature research was conducted.

3.1 Radiation Damage of the Semiconductor Devices

The operation under radiation of any semiconductor devices depends on the near perfection of the crystalline lattice to prevent defects that can trap charge carriers and lead to incomplete charge collection. Any extensive exposure of semiconductor devices, however, ensures that some damage to the lattice will take place because of the disruptive effects of the radiation passes through the crystal. While the energy that goes into the creation of electron-hole pairs leads to fully reversible processes that leave no damage, non-ionizing energy transfers to the atoms of the crystal lead to irreversible changes. The effects tend to be relatively minor for lightly ionizing radiation (beta particles or gamma rays) but can become quite significant

under typical conditions of use for heavy charged particles. For example, prolonged exposure of silicon surface p-n junction to heavy ions or fission fragments will lead to measurable increase in the leakage current [20].

The radiation-induced damage can be classified into the two categories of bulk and surface effects. The most fundamental type of bulk radiation damage is *the Frenkel defect*, produced by the displacement of an atom of the semiconductor material from its normal lattice site. The vacancy left behind, together with the original atom now at an interstitial position, constitutes a trapping site for normal charge carriers. These are sometimes called *point defects* to distinguish them from more complex "clusters" of crystalline damage that are formed along the track of a primary "knock-on" atom if sufficient energy is transferred. Gamma rays and electrons with energy of a few MeV or less create only point defects, whereas heavy charged particles of equivalent energy are generally more damaging because they also form clusters. The number of Frenkel defects produced by a fission fragment is estimated to be about 100-1000 times greater than that produced by an alpha particle. At the other extreme, an incident electron or beta particle requires a minimum of about 145 keV to produce a defect, and very little damage is observed for electrons whose energy is much below 250 keV. The severity of damage to be anticipated is therefore a strong function of the nature of the radiation involved. Some minor annealing of the radiation damage can occur over long periods of time, but for all intents and purposes, the damage is permanent [21].

The increase in leakage current appears to be more directly related to surface effects and also contributes to increase in leakage current fluctuation. In devices such as silicon MOS-FET that include oxide passivity layers, the surface effects are closely related to the ionization created within the oxide and its trapping at interfaces. For penetrating radiation including

gamma rays or neutrons, the damage is generally distributed throughout the device and the direction of incidence of the radiation has little effect. For electrons or charged particles, however, the orientation with respect to the silicon wafer is important. Irradiation of the front (or gates) surface of totally depleted MOS-FET requires exposures that are several orders of magnitude less than those needed to produce the same effects by irradiation of the back (or aluminum) contact.

For silicon surface p-n junction various data have been published on the integrated flux of charged particles required to produce a significant deterioration in p-n junction performance. Although subject to a great deal of variability, depending on the specifics of each experiment, serious changes appear to take place for irradiations of about 10^{14} fast electrons/cm^2, 10^{12} to 10^{13} protons/cm^2, 10^{11} alpha particles/cm^2, and about 3×10^{8} fission fragments/cm^2. Exposure to fast neutron fluxes of about 3×10^{11} neutrons/cm^2 and gamma rays doses of about 10^{6} R are also sufficient to lead to significant performance degradation

An effect known as *type inversion* has been observed to occur in high resistively *n*-type silicon after prolonged exposure to fast neutrons or high-energy particles with integrated fluency of about 10^{13}/cm^2. The effective concentration of donors gradually decreases with exposure, until a transition to *p*-type behavior is observed. Some models for this change postulate the radiation-induced formation of deep acceptor levels, close to the center of the band gap, that become electrically active when voltage is applied to p-n junction. The device may continue to function after the inversion with the same polarity of applied voltage, but eventually the growing concentration of acceptors raises the voltage level required for full depletion to levels beyond those causing breakdown.

3.2 Radiation Effects on Field Programmable Technologies

Field programmable gate array (FPGA) devices, used in spacecraft electronics, have grown in size (number of logic gates) over the past few years. Manufacturers of FPGA take different technological and architectural approaches that directly affect radiation performance. After analyzing current technologies and architectures and their radiation effects implications we can consider as following:

1. There are basic radiation effects on semiconductor devices as was described in the paragraph 3.1, radiation damage of the semiconductor devices. The radiation start to directly affect the Single Event Upset (SEU) performance with threshold LET far beyond 60 MeV-cm^2/mg and total dose more than 400 krad(Si).
2. There is limitation for MOS-FET – devices with the dielectric layers in the active area of device. This layer accumulates the electrical charge with the radiation dose. During exposure to ionizing radiation, electron-hole pairs are formed in the insulating silicon oxide layer immediately below the gate. If a positive bias voltage is applied to the gate during the exposure period, there is a tendency for these charges to separate. The electrons will move toward the gate, and the holes toward the SiO_2-Si interface where they tend to be trapped and form a fixed positive charge. The effect of this charge is to cause a shift to more negative values in the threshold gate voltage, and the amount of this voltage shift is a reasonably linear measure of the integrated dose. The higher the bias voltage, the greater will be the fraction of the charges collected, resulting in higher no stability. If the MOSFET structures have an effective area of less than 0.1 mm^2 and oxide thickness of 100nm it shows sensitivities in the range of 10-1000 mV/Gy. With a

readout capable of measuring changes of the order of 0.5 mV, the corresponding minimum measurable dose would be 5×10^{-2} Gy (5 rad) to 5×10^{-4} Gy (50 mrad). MOSFET designs with very thin oxide layers and low threshold voltage can achieve radiation stability. It can reach SEU performance for total dose more than 300 krad and threshold LET 65MeV-cm^2/mg.

3. There is technological and architectural "noise". Manufacturers update their designs and continue to shrink the fabrication process. As the result it is difficult to understand the causes of changing radiation characteristics. Sometimes change of technology can lower SEU performance for total dose less than 3 krad and threshold LET 5MeV- cm^2/mg. The racing for the Commercial-Off-The-Shelf (COTS) devices to be used in a radiation environment leads to very complicate situation. Several studies performed to determine the affect of design and process on total dose capability of commercial parts and gain additional insight into the devices' total dose limitation [24, 25] had little success. When quick radiation test can not be performed during the manufacturing, the very nature of radiation stability requires carefully selected architecture of the device and very stable technological process. We have reasons to believe that radiation stable FPGA will be fabricated on special foundry in the very near future. The research group from NASA and Actel Corporation has the same vision and shifted the research efforts to the development of new software to improve the SEU performance of a flip-flop, exploiting the configurable nature of FPGA technology, untill radiation – proof FPGA will be available.
4. There are radiation–proof technologies MNOS (Metal-Nitride-Oxide-Semiconductor) - EPROM devices and radiation stable MNOS-FET devices on an insulating silicon nitride

substrate. It has been found [22] that under an irradiation of 1000 krad in these devices was not produced inversion layer in the silicon and no leakage current was observed.

5. Recently XILINX Inc. has developed the Vertex QPRO family of radiation hardened FPGA devices for space and satellite applications with up to 1,124,022 system gates (96 frames x 64 CLBs per frame) and guaranteed total ionizing doze to 100 KRad(si) with latch-up immune to LET = 125 MeV cm2 / mg [23].

The Appendix C "Radiation in Space Environment" describes some particles and their properties important for radiation effects for semiconductor devices. It also gives definitions and conversion units in irradiation physics used above.

CHAPTER 4

FAULT DETECTION AND RECOVERY IN FPGA DEVICES

Another aspect to be considered was fault tolerance of RCP-based on-board computing systems. Conceptually, the uniform organization of all levels of system hierarchy of RCP hardware such as:

a) logic gates in Common Logic Blocks (CLBs),

b) CLBs in FPGAs

c) memory cells in ROM and SRAM-devices

allows simple replacement of faulty elements (system gates, memory cells, etc.). This conceptual possibility needs to be researched more deeply for the development of reliable run-time diagnostic and recovery mechanisms.

The objective of the research is development of methods and hardware/software means, which being incorporated into the RCP would increase its fault tolerance. These methods should provide high reliability and small time for tolerating faults, and the corresponding means should have small hardware/software overhead. We will break this task into three subtasks:

1. Fault detection,
2. Fault location and
3. Fault recovery

Each of these subtasks will contribute to the above-mentioned characteristics of fault tolerance.

The simplified architecture of the On-Board RCP is presented in Figure 4.1. The Configuration Controller (CC) supplies configuration sequence to the RCP.

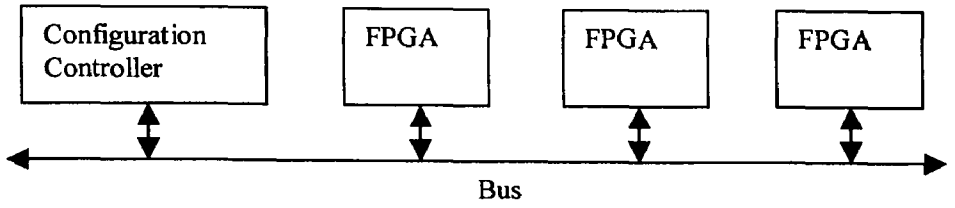

Figure 4.1: A simplified architecture of the Adaptive Re-Configurable Platform

4.1 Fault detection and location

Fault detection in FPGA devices directly depends on internal organization of FPGA micro-architecture. This micro-architecture consists of a number of Configurable Logic Blocks (CLBs) interconnected by wires and switch boxes with each other and to the Input/Output Blocks (IOBs). The simplified architecture is shown on Figure 4.2. The CLB contains logic function generators, flip-flops (latches), and a number of multiplexers and associated circuitry. Wires within the FPGA are connected through pass transistors. The pass transistor is turned on or off by loading a binary value into the configuration SRAM cell connected to its gate. The CLBs contain function generators, which are also made up of SRAMs. A function generator has a Look-Up Table (LUT) made out of SRAM cells, together with some addressing circuitry. The LUT can be used for implementing combinational logic or as a RAM.

For an FPGA-based Reconfigurable system, faults can occur in the logic blocks (LUTs, CLBs, etc) and in switches that provide connection between the different logic blocks. A typical fault in a logic block may be a stuck-at fault in any LUT location. A typical

interconnect fault may be a stuck-at fault in the configuration SRAM cell whose content turns a pass transistor on or off, providing interconnection between two leads. Both of these types of faults are equally important. From the test point of view, it is now widely admitted that FPGAs cannot be considered as classical digital ASICs [35].

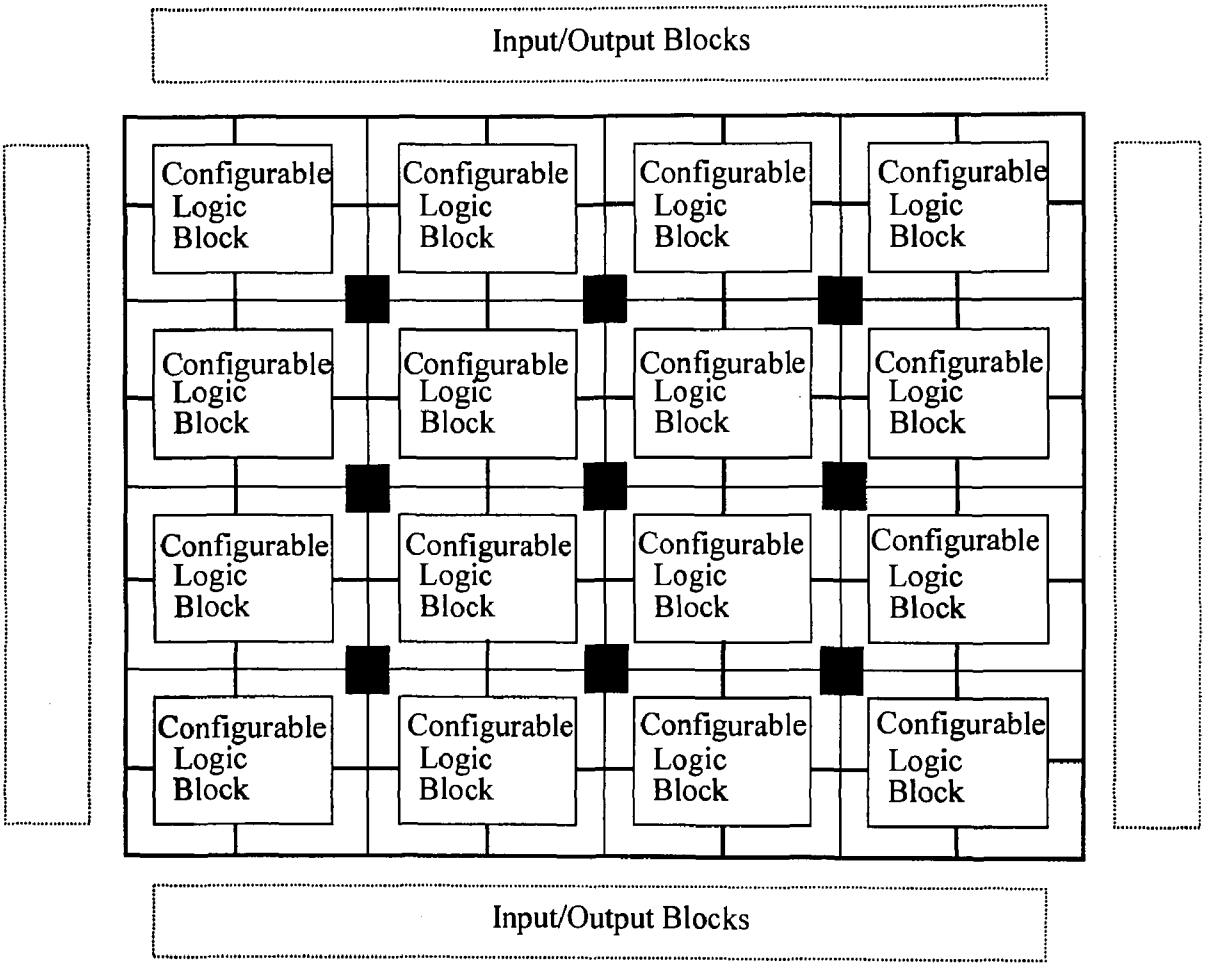

Figure 4.2: Architecture of a Field Programmable Gate Array (FPGA)

Classical test approaches fail when applied on FPGA. In the recent published works, different test aspects are considered: Boolean testing of FPGA [25, 26, 28], BIST for FPGA

[25, 26], I_{DDQ}[1] testing of FPGA [27], diagnosis of FPGA [29]. Because of the complexity of FPGA testing, usually each paper targets a specific FPGA part: the interconnect in [26, 28,30], the logic cells in [25, 27], the memory cells in [32, 33, 34],

the logic interconnect interface. Indeed, FPGAs appear as very complex circuits and these works use a classical divide and conquer approach.

In [25 -34] authors try to generate a *manufacturing oriented* test procedures. In [35], the *application oriented* test procedure is proposed, which can be used by an FPGA user. In this procedure, the user does not need to test the complete FPGA. He is only interested in testing the part of the FPGA used for his specific application. Test pattern generation in such case can be significantly accelerated. The paper targets only logic cells of an FPGA.

In [24] authors address the fault location problem for the FPGA logic blocks. Testing the logic blocks of Reconfigurable FPGA devices has been studied by many researchers [36-38]. Testing interconnect switches was discussed in [28, 39]. Techniques to locate faulty FPGA interconnects are described in [29, 40]. The technique, discussed in [38] requires a fixed number of reconfiguration sessions. It reconfigures some of the logic blocks as pattern generators or response analyzers, while testing the other blocks and vice-versa. The technique does not use any knowledge of the application that was implemented in the FPGA. Hence, it requires a set of configurations that cover all the faults under consideration for all possible configurations. This technique is extended in [41] for diagnosis to locate the faulty logic blocks in an FPGA.

[1] I_{DDQ} is the quiescent current consumed by an FPGA. If two gate's outputs (or interconnects) are connected (bridging fault) then placing opposite logic values on those outputs will lead to the increase in I_{DDQ}. This current can be measured for every possible bridging fault, which allows testing the FPGA [**Error! Reference source not found.**].

The method discussed in [42] can be used to locate multiple faults in an FPGA. The basic idea is similar to that of [41]. Typically, a part of the FPGA is reconfigured to test another part and vice-versa. For example, the FPGA is divided into three sets of CLBs and each set tests another set according to a diagnostic graph. The test time in this method depends on the number of faults and is independent of the array size. The techniques in [41] are based on BIST.

Another application independent diagnostic technique is presented in [43]. The technique consists of two steps: horizontal diagnostic and vertical diagnostic. These two steps identify the row and column respectively that contain a faulty CLB. The C-testability concept [44] is used to improve this technique such that the test time is independent of the array size.

All the techniques mentioned in [28, 29, 36-44] are application independent and can be used for both production test and field test. However, when diagnostic is needed for an FPGA that implements a fixed application, the diagnostic procedure can be accelerated by using the design information rather than testing for all possible configurations [24]. In [24], the authors focus on combinational logic. A novel aspect of the technique is the way the concept of pseudo-exhaustive BIST has been applied for fault-location purposes. Another novelty of the approach lies in the fact, that the routing structure of a CLB is not altered. This is important feature, because typically designs mapped to FPGAs are routing-limited. Also, modifying the existing routing configuration during each fault location step is time consuming. Another advantage of this technique is that, with partially Reconfigurable FPGAs, while the circuit under consideration is being diagnosed, the remaining part of the system whose inputs are not connected to the outputs of this circuit can still be operational.

In [45] distinction is made between configuration-independent testing and configuration-dependent testing. In configuration-independent testing, no assumptions are made about the way in which the user will configure the FPGA. The goal is to maximize the fault coverage for all possible configurations.

Configuration-independent testing is done when the FPGA is manufactured. Configuration-dependent testing, on the other hand, involves testing that a *particular* FPGA configuration is fault free. Higher fault coverage (for the particular configuration) can be achieved with less test time. The approach described in [45] is a configuration-dependent test technique for interconnects. Configuration-independent test techniques for interconnects [27-29, 37, 46] has the following drawbacks:

1. Time it takes to develop the diagnostic test necessary to locate the faults
2. Time to develop test configurations that provide a high coverage
3. Time to run the tests

The configuration-dependent test technique test method of [45] addresses all of the above problems. The method detects and locates all stuck-at and bridging faults in the interconnect switches for particular FPGA configuration. This is done by modifying the original configuration by only changing the logic function of the CLBs to form test configurations that can be used to quickly test the interconnect (the "walking-1" approach). A systematic procedure is applied to the original configuration to generate a small set of test configurations. The logic functions of the CLBs in the test configurations are chosen in such a way that the "walking-1" approach can be used to detect and locate all stuck-at and bridging faults in the interconnect with a small set of test vectors. Since only the logic

functions of the CLBs are changed, time-consuming placement and routing is avoided. The process of generating the test configurations and test vectors is fully automated and very fast.

4.2 Fault recovery

If the fault has been detected and located, then it can be tolerated in a way described in [47]. Each primary cell u in the FPGA is assigned a cover cell, which can be reconfigured to replace it in the event that cell u becomes faulty. Primary cells are assigned to cover other primary cells in a chain-like manner, with a spare cell covering the last primary cell in the chain (Figure 4.3). These chains are defined to be along either rows or columns of the array, with a spare cell at the end of each row or column. In order for a cell to cover another cell, first, the cover cell must be able to duplicate the functionality of the dependant cell. This is easy in an FPGA, since all cells are identical.

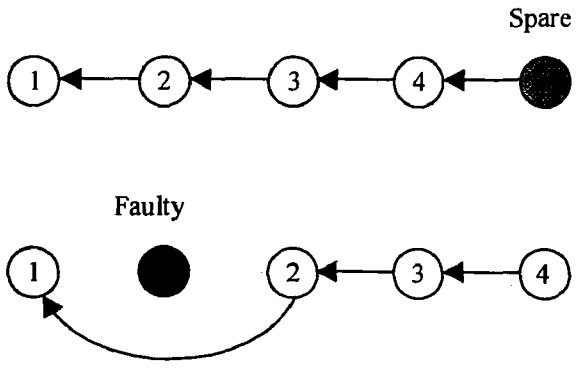

Figure 4.3: A covering graph for a fault tolerant FPGA with four primary cells and one spare cell. Reconfiguration in the covering graph after cell # 2 becomes faulty.

Second, the cover cell must be able to duplicate the connectivity of the dependant cell with respect to the rest of the array. The method of ensuring connectivity is described in [47].

Each net connected to a cell must also include a cover (reserved) segment bordering the cover cell. These cover segments allow the logical connectivity to be maintained. The reconfiguration procedure assumes that cover segments necessary for reconfiguration around a faulty cell are provided by the routing tools and included in the initial configuration data that is loaded serially. Channel segments reserved for use in reconfiguration do not add extra parasitic delay to nets in a non-reconfigured array, since they are connected only when needed. No re-routing is necessary in the event of a fault.

In their fault tolerant technique, Kumar at al. used adaptive customization to avoid defective sections and artificially enhance a programmable gate array's yield [48]. Their technique was strictly off-line and fault tolerance was achieved by bypassing faulty programmable logic cells at the time the circuit was initially mapped to the FPGA. They described a two-step process to achieve defect tolerance: test of an un-programmed device to locate defective components and program the device to avoid the defective portions. They presented heuristics for adaptively programming an FPGA in the presence of faults.

Hatori at al. introduced redundancy as a fault tolerant method for FPGAs [49]. Their method used specialized selector circuitry to reconfigure FPGA circuits in the presence of faults. Similar to methods used for fault tolerance in SRAMs, an additional column of programmable logic blocks is added to the FPGA. In the event that a logic block was faulty, all functions mapped to the column where the faulty block was located were shifted toward the spare column. All subsequent columns of functions between the column with the faulty block and the spare column were also shifted toward the spare column. They did not detect faults, only reconfigure around them. Since they eliminated an entire column for one fault,

their fault tolerance was low, and they estimated a performance degradation of 5%. This was an off-line technique.

Durand and Piquet proposed a fault tolerant FPGA architecture with the ability to repair itself in the presence of faults [50]. They used a special multiplexer with repair and self-diagnosis capabilities. Their special circuitry was capable of detecting logic failures during runtime, and the architecture had a limited capability to reconfigure itself dynamically. Their fault tolerant technique allocated two extra columns of cells, and in the event that a fault was detected, they bypassed the faulty column by shifting toward the spare column. Since the author's reported that more than complete logic redundancy was required for their method, the area overhead for fault tolerance exceeded 50%.

Narasimhan at al. developed an of-line fault tolerant technique for FPGAs or Wafer Scale Integrated Arrays [51,52]. They use a pebble shift algorithm to reconfigure around faulty blocks. Their method is flexible, since it is not limited to one fault per row, column or tile.

Howard and Tyrrell described ideas for increasing yield on FPGAs [53]. Their idea was to insert extra bypassing columns or rows into the FPGA. They looked at both removing an entire row (column), if it contained a fault and removing only the faulty block from the row. Their methods were strictly for yield enhancement and the faults were bypassed at the time the configuration for the FPGA was downloaded. They also introduced global long lines to reduce the performance hit associated with reconfiguration.

Kelly and Ivey used redundancy to tolerate faults in applications mapped to FPGAs [54]. Their technique used a shift method to reconfigure in the presence of faults. They incorporated a reconfiguration switch and used normal place and route tools for mapping

circuits to FPGAs. A step-based switch configuration algorithm was used to configure the special switches around faulty elements. This technique was also off-line.

Cuddapa and Corba used Xilinx SRAM based FPGAs to demonstrate the fault tolerant capabilities of FPGAs [55]. In their study, they randomly picked logic blocks to be faulty. Then they reconfigured the circuit around these faults using commercially available tools. The main contributions of their work were an algorithm to determine fault coverage of a design and a definition of the fault recovery rate for any given design implemented in SRAM-based FPGAs. Additionally, they demonstrated that fault recovery was possible on FPGAs by other than modular redundant methods. This technique was off-line and did not include fault detection.

Hanchek and Dutt developed a method to increase FPGA yield [47,56]. Their method uses node covering and reserved routing resources to replace the functionality of faulty cells. One row (column) of cells is reserved for spares. If a cell in any given column (row) goes bad, the functionality of all cells in the column (row) from the faulty cell to the spare is shifted toward the spare cell. Spare routing resources are used to eliminate overhead of re-routing the updated circuit placement. Relative to covering interconnect faults, they use a similar idea where spare interconnect resources are allocated as a grid, and when a fault occurs, it is replaced by an extra segment or grid of segments. The main advantage of this method is that it is very fast. Since the spare resources have already been allocated to cover a limited number of faults, the reconfiguration time is linear with respect to the number of faults. The main problem of this method is limited fault tolerance, namely only one cell per column (row). If two faults were found in the same row, this method breaks down.

Mahapatra and Dutt proposed a method that dynamically allocates interconnect resources to bypass faulty cells after faults have been located [57]. If the new required segments conflict with the current usage of routing tracks, the layout is incrementally modified to make room for the new segments. This method is dynamic only with respect to the interconnect resources, as the spare cells are statically allocated.

Emmert and Bhatia developed a fault tolerant technique for incrementally reconfiguring FPGA mapped circuits around faulty programmable resources [58, 59]. They used minimax grid matching, to match faulty PLB locations to unused spare PLB resources. Then they incorporated a shift methodology to shift PLBs between the faulty PLB and its matched spare location toward the spare location. For interconnect faults they introduced the idea of an embedded incremental router to route around faulty interconnect resources. The router uses the read/write capability of the FPGA configuration memory to rip-up and reroute without a netlist. While their method for reconfiguring around faulty PLBs was applied to the Xilinx FPGA, their fault tolerant technique for interconnects was applied to an artificial FPGA architecture, not to a commercial FPGA.

Lach et al. developed a low-overhead on-line fault tolerant system [60]. Their basic approach was to partition a design into a number of tiles. Each tile was allocated one spare PLBs. In the event a fault in a tile was found, the spare PLB was used to replace the faulty PLB by using a precompiled replacement configuration. Several replacement configurations were stored for each tile. In the event that a fault was detected, the configuration associated with that fault was downloaded. The main drawbacks of this method are static spare allocation and low tolerance, typically limited to one fault per tile. If multiple faults occur in close

proximity (within a tile), this method breaks down. There is no ability to draw spares from other tiles to cover several faults in a single tile.

Fault tolerance in a system relies on spare resources to replace the faulty ones. Spare resources lead to area overhead and degradation in the system performance. In all of the above-mentioned works, spare resources are statically allocated prior to fault occurrence. But static allocation may not provide sufficient resources in an area affected by multiple faults, while resources in areas not affected by faults will be wasted. Efficient spare usage is critical in long-life missions, which must be terminated when one more fault occurs in an area where all spares have been used. The regular structure of FPGAs and their inherent redundancy enable the implementation of low-cost fault-tolerant techniques. Since a typical design uses only a portion of the logic and interconnect resources of the FPGA, the unused resources can provide spares to be used to replace the faulty ones.

On-line testing that occurs concurrently with the normal system operation is essential in high-availability systems that may not be taken off-line for testing, and whose operation should be interrupted as little as possible. In such systems, fault diagnosis and reconfiguration must be accomplished very fast. On-line testing typically relies on modular redundancy and on information redundancy used in coding techniques.

One problem with conventional on-line testing is that it detects only faults that affect the current operation performed in the system. New faults, however, are equally likely to occur in spare resources or in the currently unused portion of the operational part of the system. Thus to guarantee a reliable operation, the on-line test must completely check all the system resources, including spares.

A novel on-line fault-tolerant approach for CLBs of an FPGA is presented in [61]. The approach employs the Self-Testing AReas (STARs) of the FPGA that are off-line and under test, while the rest of the device is on-line, continuing normal system operation. Partial run-time reconfigurations via the boundary scan interface of the FPGA allow the test configuration used by STARs to be downloaded without any impact on the system operation. After testing of a STAR has been completed, the STAR moves to a new location, which is implemented by a sequence of pre-computed partial reconfigurations and assures that the entire FPGA will be eventually tested. If faulty CLB is detected, spare one from a STAR will replace it. Every working CLB has a spare one. The configuration that replaces the working CLB with the spare one is pre-computed.

Most previous methods described in literature are for off-line fault tolerance and used for manufacturing yield improvement. In contrast with them, the proposed approach removes these limitations.

Reconfiguration process in a system is controlled by a module external to the FPGAs (typically this is embedded microprocessor having some storage for the various FPGA configurations). The tasks of this processor are also extended to controlling the test, diagnosis, and fault-tolerance functions, including their associated reconfigurations.

STAR is configured as several disjoint regions with independent BIST structures. Every such structure (Figure 4.3) is composed of a Test Pattern Generator that applies pseudo-exhaustive test patterns to two identically configured programmable logic Blocks Under Test (BUTs), whose outputs are compared by an Output Response Analyzer. The BUTs are repeatedly configured for testing in all their programmable modes of operation.

Once the BUT has been completely tested, the next pair of BUTs is tested in all of their modes of operation.

The cost of the approach is significantly lower than replicated modular redundancy.

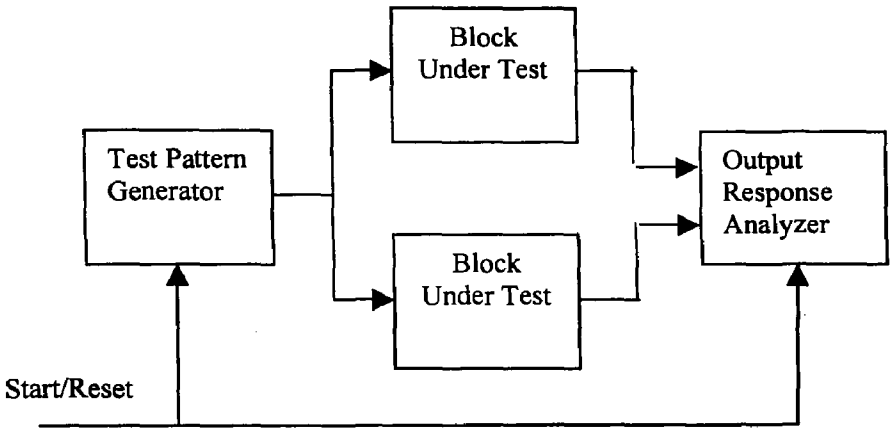

Figure 4.4: Testing Programmable Logic Blocks.

CHAPTER 5

ORGANIZATION OF MULTI-TASK RE-CONFIGURABLE STREAM PROCESSOR WITH SELF-ASSEMBLING MICRO-ARCHITECTURE

In most of industrial data-stream processing systems (e.g. video-surveillance, video-recognition, digital video-broadcasting and digital communication systems, etc.) it is required to process multiple streams of data with very high rate (GB/s). The usual approach is implementation of application specific integration circuits (ASICs) or Field Programmable Gate Array (FPGA) devices were application specific processor core (ASPC) is loaded [62]. This approach seems cost-effective when ASIC or ASPC architecture is designed for one specific application (task).

However, there are several disadvantages associated with this approach. One of the main problems associated with ASICs is lack of hardware flexibility and inability for modification of its micro-architecture if any change of processing algorithm is required or hardware bug is found. Generally, FPGA utilization can mitigate this problem. However, FPGA by its nature requires much more logic resources (configuration SRAM, routing switches, look-up table logic etc.) than ASIC for the same application [63]. On the other hand, in most of real applications multiple streams should be processed in parallel and processing algorithms can vary in different processing modes. ASIC and ASPC approach assumes that all processing architectures for all tasks and their modes should be stored in the ASIC or FPGA in the form of real hardware. However, in most of real applications:

a) Not all tasks are initiated at the same time in the multi-task workload and

b) Only one mode from many can be requested for each task,

Thus, if all application processing architectures are presented in a form of real hardware a lot of logic resources and power will be wasted because of non-active hardware. To solve this problem the concept of run-time re-configurable (RTR) systems can be implemented. RTR approach discussed in Chapter 2, assumes utilization of FPGA devices with partially reconfigurable micro-architecture [63], [64]. This approach allows loading into the FPGA only that processing core, which is needed for the task and task mode going to be activated. However, in the existing RTR computing platforms each processing core has to be developed and pre-compiled using CAD system associated with utilized FPGA family (e.g. ISE Foundation for Xilinx FPGA devices). Instead, in [66] was proposed an approach where Application Specific Virtual Processor (ASVP) can be assembled on-chip by uniformed "LEGO" blocks. These LEGO blocks are cores, which were called Virtual Hardware Components (VHC). Furthermore, this approach assumes that the ASVP assembling procedure is fully automated because it should be performed during hundreds of microseconds without any influence of designer / system operator.

Thus, the process of creation of ASVP micro-architecture is organized as *self-assembling procedure*. Same procedure can be activated when any hardware fault is detected in any of ASVP running in the FPGA. In this case any damaged Virtual Hardware Component can be restored by scrubbing procedure [65] or re-loaded to another available slot of the FPGA. That is why RTR computing platform based on above concept allows creation of a universal computing platform with self-assembling micro-architecture for parallel acquisition, processing and transmitting (via high-bandwidth network) multiple data-streams where each data-stream task can be initiated, terminated and re-loaded without interruption of other data-stream execution and / or data transmission processes.

5.1 Organization of Virtual Hardware Components (VHC)

Most of the data-stream processing architectures can be represented as a pipeline reflecting the structure of Data-Flow Graph (DFG) [62]. Considering the structural organization of different FPGA families we found that the best candidate for these requirements was Xilinx "Virtex" family of partially re-configurable FPGA devices [64]. The structure of "Virtex" FPGA consists of (Figure 5.1):

1) Arrays of CLBs (*Configurable Logic Blocks*),

2) Arrays of IOBs (*Input Output Blocks*),

3) SRAM memory blocks (*Block RAM*),

4) Clock logic resources (DLLs, etc.) and

5) Routing resources.

These resources can be configured into one or more data-paths for one or more pipelined data-stream processors. The configuration can be done by loading configuration data file into the Configuration SRAM, which programs logic functions of Look-Up-Table (LUT) of each CLB and interconnections between logic, I/O, clock and memory resources. The configuration data file for entire FPGA device can be divided into smaller configuration data files for partial FPGA reconfiguration. Each small configuration data file can represent a Virtual Hardware Component (VHC) to be downloading into addressable FPGA slots (CLB-columns).

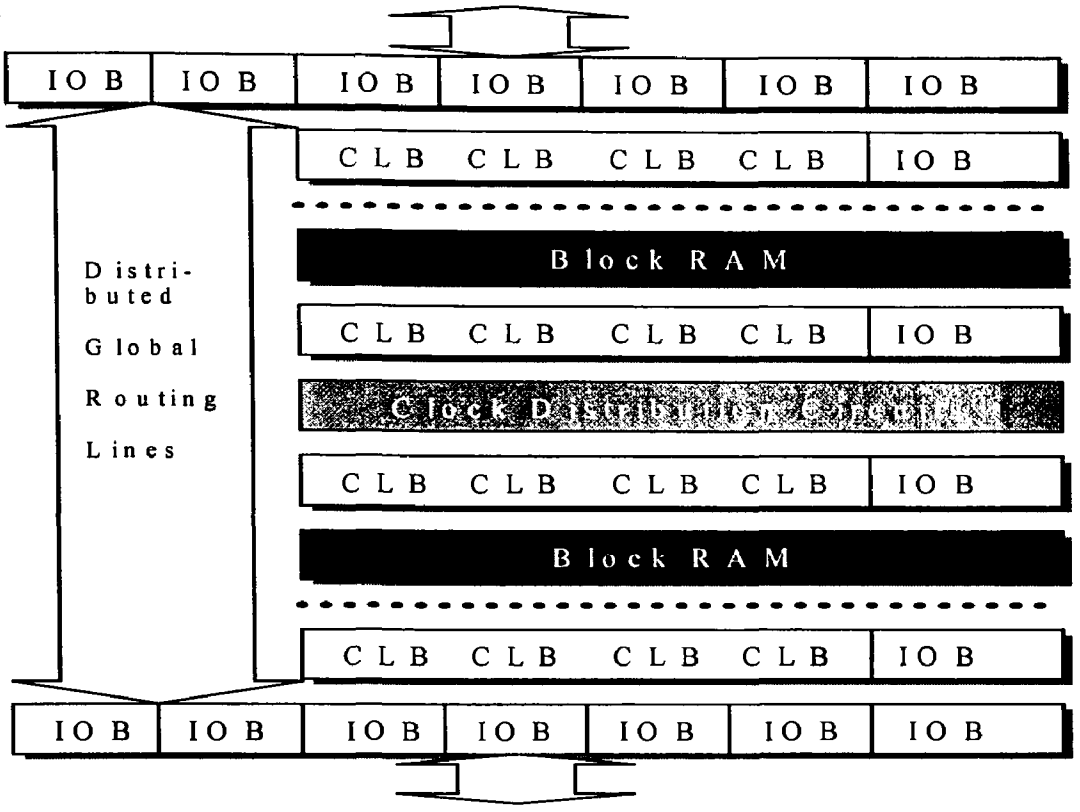

Figure 5.1: Structure of partially reconfigurable Xilinx "Virtex" FPGA device

The micro-architecture of VHC consists of two major parts (Figure 5.2):

a) Processing Element (PE): Adder, Multiplier, FFT, etc.

b) Interface Element (IE): 8-bit, 16-bit, 32-bit, etc.

Xilinx "Virtex" FPGA structure allows loading of VHC partially, because partial reconfiguration for this family of FPGAs allows addressable configuration of each frame (part of a CLB-column). As was shown in "Virtex" FPGA device data sheet [63] the special tri-state buffers (T-buffers) can be implemented to connect or disconnect Virtual Hardware Components (VHCs) to the Global Routing Lines. Those T-buffers can be dedicated to specific global routing lines. Thus, each VHC, which contains the Interface Element with T-

buffers associated with specific global routing lines, will be connected to those lines but initially tri-stated at the initial architecture loading state.

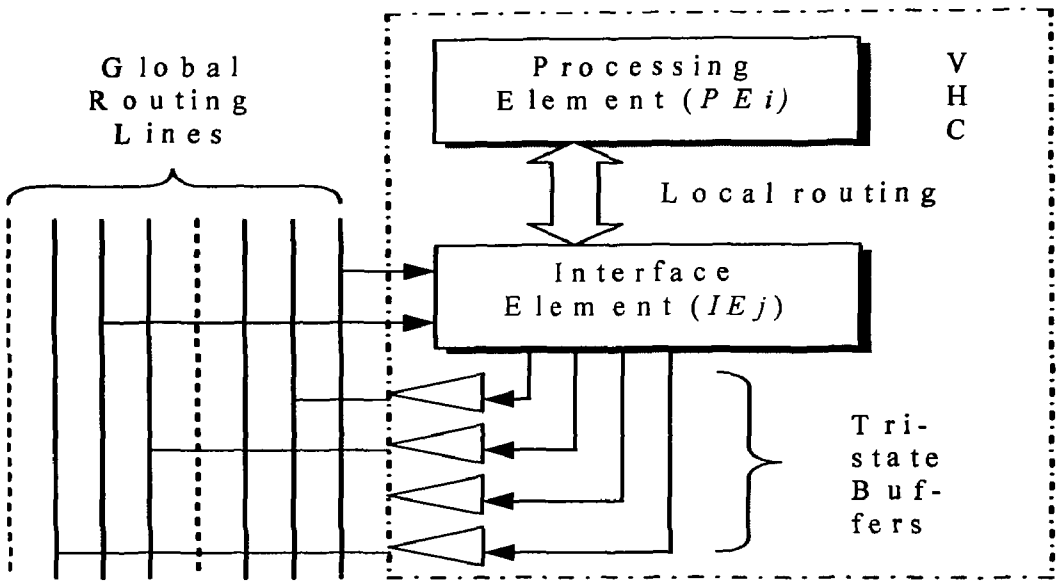

Figure 5.2: Micro-architecture of a Virtual Hardware Component (VHC)

5.2. Application Specific Virtual Processor assembling procedure

Architectural synthesis of application specific processor normally is based on application (task) algorithm analysis, creation of the Data Flow Graph (DFG), architectural optimization based on DFG and data-path synthesis [66]. As a result, a complete processing architecture described in one of Hardware Description Languages (VHDL, AHDL or Verilog) is usually compiled to be implemented in the ASIC or FPGA. Instead, the approach of *self-assembling* of task optimized Application Specific Virtual Processor (ASVP) inside the partially re-configurable FPGA using pre-compiled sub-cores – VHCs was implemented.

This is similar to "Port Map" procedures in Hardware Descriptive Language but in on-chip level. To illustrate this concept let us consider the following example. Let us assume that the task requires to process four streams of data A, B, C and D. These streams should be processed as follows:

$$Y= (A+B)* (C+D)$$

In this case task algorithm can be represented by the Data Flow Graph (DFG) shown in Figure 5.3. To simplify the case, in our example we will not consider scheduling and binding procedures and assume that the DFG should be mapped in hardware "as is".

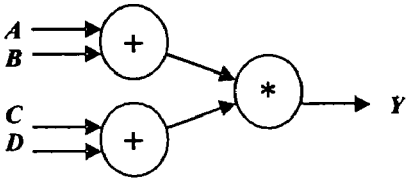

Figure 5.3: Data Flow Graph of stream processing task

To assemble any micro-architecture by VHCs – we have to have the library of available VHCs. Because Virtual Hardware Components (VHCc) are pre-compiled cores (configuration data files for certain FPGA device), each VHC has to be located in VHC-memory in certain location. The VHC-identifier consists of two parts: Processing Element Type Identifier (PETID) and Interface Element Type Identifier (IETID). Based on this ID-information, the requested VHC can be retrieved from VHC-library and loaded into the FPGA to the selected slot (addressed CLB-column). All above operations such as: getting VHC from the library assigning available FPGA slot for the VHC and loading VHC to the FPGA has to be done by special hardware unit - Hardware Operating System (HOS). HOS structure will be described in the Section 5.3. In our example (Figure 5.3) we need two

adders and one multiplier. Let us assume that 8-bit adder has PETID = 01 and 8-bit multiplier PETID = 02. Let us also assume that Interface Element with two groups of input lines connected to Global Routing Lines (Figure 5.2) with numbers from 0 to 7 and from 8 to 15 and with output tri-state buffers (T-buff) connected to Global Routing Lines (GRL) with numbers from 16 to 23 has IETID= 00. Similar, Interface Element with IETID=01 will have two groups of 8-bit inputs connected to GRL #24 - #39 and output T-buffs connected to GRL #40 - #47. Interface Element with IETID =12 has two groups of input lines connected to GRL #16 - #23 and #40 – 47 and 16 output T-buffs connected to GRL #48 - #63. Thus, VHCs, which should be requested to create Application Specific Virtual Processor for our task will have the following ID:

a) Adder #1: [0100] for *(A+B)* - operation,

b) Adder #2: [0101] for *(B+C)* -operation,

c) Multiplier: [0212] for *Y* calculation.

Other components, which have to be provided for ASVP configuration, are the following:

a) External interface: Input / Output buffers,

b) Internal link routing and

c) Clock routing scheme.

All this components are usually task-specific and should be combined to the fixed part of ASVP architecture. Thus, for task *Ti* we will have fixed part of ASVP[*i*] architecture – *Afix[i]*. Now we can consider complete process of ASVP creation in the partially re-configurable FPGA. This process consists of the following steps:

1. Hardware Operating System (HOS) receives a request for task activation and loads fixed part of ASVP optimized for this task,

2. HOS receives the mode of data processing and retrieves from special table list of VHC-identifiers to be loaded into the FPGA,
3. Using ID-Address Conversion Table, HOS generates one after another, addresses of each VHC-configuration data files,
4. Each VHC-core HOS stores in the VHC-loading buffer. Then, HOS concatenates the FPGA-slot address to the VHC-core and creates configuration bit-stream for this Virtual Hardware Component,
5. Bit-stream of the selected component, HOS loads to the FPGA into selected slot,
6. When all components are loaded, HOS initiates data processing. For our example ASVP architecture is shown in Figure 5.4: Let us assume that mode of task has to be switched from Mode 1: $Y = (A+B)*(C+D)$ to Mode 2: $Y = (A+B) / (C+D)$

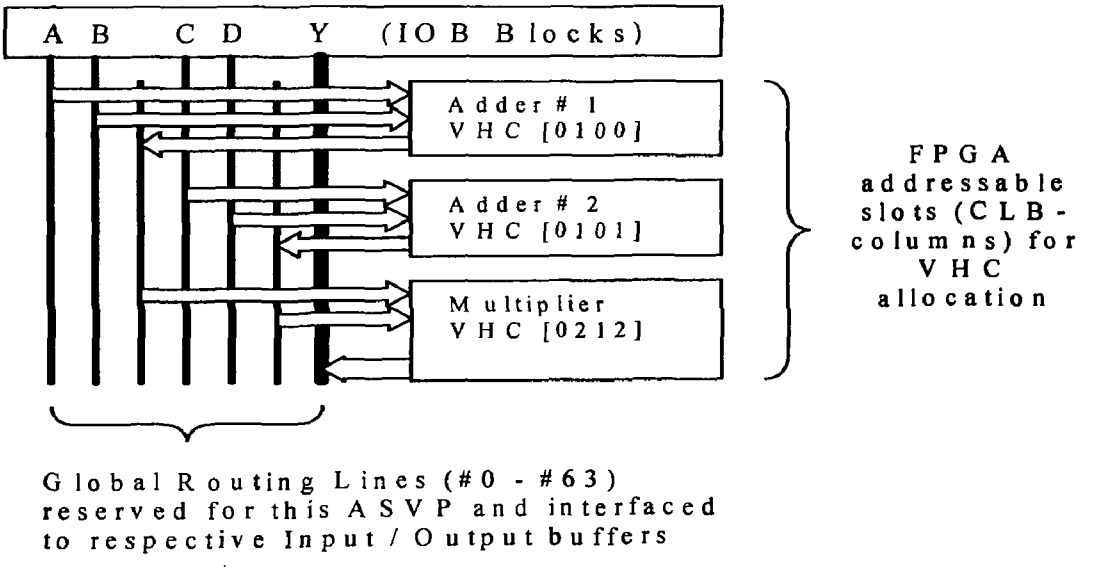

Figure 5.4: ASVP architecture assembled in the FPGA from pre-compiled VHCs reflecting task DFG (Figure 5.3)

In our case only Multiplier should be replaced by Divider with the same Interface Element. So, if Divider's PETID = 04 the VHC ID=[0412]. This replacement can be done very fast. In our experiments we measured time for replacement of one CLB-column VHC equal to 140 uS (for Xilinx XCV-400E, 50MHz parallel load). This was 142 times faster than if complete ASVP is replaced by re-configuration of entire FPGA. More detailed these results are discussed in the Chapter 7. We would like to mention that the process of VHC replacement in existing ASVP architecture looks like automated plug-in operation of virtual component in Virtual Bus. Thus, the mode switching in the proposed approach can be done not by switching between existing hardware modules (Multiplier and Divider) but by re-configuring same logic resources (CLB) in short period of time (hundreds of microseconds). This reconfiguration time for many applications is close to switching time. However, reconfiguration allows dramatic minimization of hardware resources and associated cost, dimensions, weight and power consumption. Comparison with regular FPGA-based systems will be discussed in Chapter 7.

5.3 Re-configurable Parallel Stream Processor Architecture organization

Re-configurable Parallel Stream Processor (RPSP) architecture, adaptable for the multi-task and multi-mode workload includes the following major components shown in Figure 5.5

a) Re-configurable Functional Module (RFM) based on partially re-configurable FPGA contains all necessary circuits: power regulators, line buffers, etc. This module is a field of uniform logic resources that can be configured to a number of task-optimized data-stream processors.

b) VHC memory based on ROM (Read Only Memory) contains configuration data files for all available Virtual Hardware Components to be utilized in all modes of all tasks in workload.

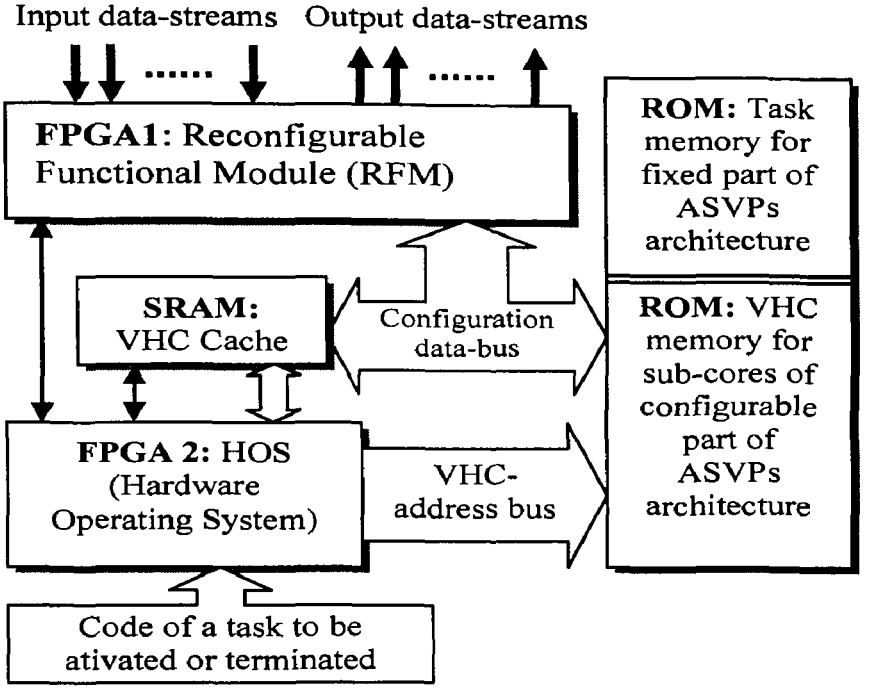

Figure 5.5: Architecture of Re-configurable Parallel Stream Processor (RPSP) for multi-task and multi-mode workload

c) Task memory based on ROM (Read Only Memory) stores the configuration data files of the fixed part of architecture *{Afix[i]}* for all tasks in the workload.

d) VHC cache based on SRAM (Static Random Access Memory) stores all VHC cores that can be used for active tasks (loaded in RFM).

f) Hardware Operating System (HOS) based on FPGA performs the following major functions: i) Task initiation and termination; ii) Mode switching by loading respective set

of VHCs to certain slots of RFM; iii) Data-streams switching and interface control; vi) RFM diagnostic and restoration functions.

The proposed architecture allows significant minimization of hardware resources for processing multi-task workload when each task can work in multiple modes of operation. Our approach is based on the fact that "density" of data processing structure (e.g. application specific processing circuits) is much higher in a form of configuration data files rather than in a form of real hardware logic. It can be "squeezed" even more when configuration data file for an entire Application Specific Processor is assembled from "LEGO"-type component cores (VHC in our terminology) It is possible because of utilization of the same component cores in different ASVP. Thus, the architecture of RPSP contains hierarchy of memory units reflecting above concept:

i) Non-volatile memory for fixed and re-configurable parts of ASVP's architectures,

ii) Cache for VHCs to be used for active ASVPs and

iii) Configuration SRAM in the Functional Module itself.

Let us consider the process of task activation under the HOS control. Firstly, HOS receives the code of the application (task) to be activated (Figure 5.5). Task code = ASVP code which contains of:

i) Code of fixed part of ASVP architecture,

ii) List of all VHCs associated with requested version of ASVP. In case of example discussed in Section 5.2, that application code of ASVP will look as shown in Table 5.1

Table 5.1: Code of associated ASVP

Fixed part #	**VHC1**	**VHC2**	**VHC3**	**VHCn**
0001	0100	0101	0212	0000

Secondly, HOS converts VHC # to the start-address of VHC configuration file in VHC Memory. The conversion mechanism is based on special table which is located in the VHC Cache and looks as shown in Table 5.2

Table 5.2: VHC# to VHC-core address conversion table

IETID# / PETID#	**00**	**01**	.....	**12**
01	0x01100	0x01200	.....	0x01C00
02	0x02000	0x02200	.....	0x03800
.....				

After that HOS retrieves address of the first VHC, sends it to VHC Memory and starts VHC1 loading process to the RFM FPGA to the dedicated slot(s). In our example for VHC1: Adder 1 [0100], the start-address of configuration data file is equal to 0x01100. This procedure HOS repeats for all VHCs in the ASVP code till the end (0000). At this moment of

time a complete processing architecture is assembled in the RFM FPGA. Now, HOS can initiate data-stream processing. We have described this process to demonstrate major steps of interaction between HOS, Cache memory and RFM to show the high-level of ASVP self-assembling mechanism implemented in the RPSP.

CHAPTER 6

MECHANISM FOR ASVP SELF-RESTORATION

There are a few main reasons for hardware faults that can occur in FPGA-based devices: i) physical defects in wafer; ii) hidden manufacturing defects; and iii) radiation effects. In our project, we considered faults occurring in small amount of CLBs in the SRAM based FPGA devices. Mostly, this type of faults occurs as a result of radiation effects (SEU – single event upset, SEL – single event lutch-up, etc.) usual for aerospace applications or applications for radiation intensive environment (nuclear power stations, etc). We have discussed major rediation factors and their influence on SRAM based FPGA devices in the Chapter 3. The methods for off-line and on-line diagnostic and fault location were analysed in the Chapter 4. The focus of the proposed research was on development of the mechanism for self-restoration of parallel reconfigurable stream processors based on the concept of Application Specific Virtual Processors described in the Chapter 5. As an alternative approach based on protection of materials from radiation intensive environment, we proposed self-restoration mechnism based on re-assembling of damaged ASVP processing structures.

To restore ASVP a three-level procedure was developed:

First level of self-restoration is based on VHC-scrubbing: re-loading of same VHC configuration data-file to the same address area (CLB-slot) of configuration SRAM, of the FPGA where this VHC was located. This can eliminate wrong state of Flip-Flops in the configuration SRAM and correct damaged VHC structure. This prosedure can be performed using the same ASVP assembling mechanism discussed in Chapter 5. This approach allows

dramatic reduction of restoration time, logic resources and power consumption in comparison with the common approach [71], [72]. That is when scrubbing procedure should be performed cyclically for the entire FPGA device. Experimental results and comarison analysis in this regard will be discussed in Chapter 7. Another benefit is that there is no additional hardware expenses for this level of restoration.

Necessity of second level of self-restoration appears when radiation effects or other reasons (e.g. hidden manufacturing deffects) make one of logic gates damaged. This case can be considered only when VHC scrubbing does not restore ASVP functions. In this case, the damaged gate should be avoided. However, because any VHC is based on CLB-column (slot) organization we developed a mechanism for re-location of VHC from the damaged slot, to the spare slot. In this approach extra hardware cost is involved to increase reliability of the system. However, it is possible that after some period of time (may be very long) all reserved slots will be used.

Third level of restoration with performance degradation appears when all spare CLB-columns (slots) are used. In this case it is possible to load the existing CLB-columns with another variant of VHC, with a reduced area and functional parameters. It allows the computing system to overcome the damaged CLB in the column being corrupted. In this case we "pay" extra processing time to increase the reliability.

6.1 VHC scrubbing

Firstly, the self-restoration mechanism based on scrubbing procdure was developed for platform based on partially reconfigurable Xilinx Virtex family of FPGA devices [68], [70]. This level of self-restoration of ASVP consists of the following steps:

i) Determination of a VHC which doesn't process data properly using Built-in-Self Test sub-system [73].

   Note: The BIST was not a target for this project and thus, it was assumed that we already know the fault location.

ii) Pausing the data-stream computation in the damaged pipeline

iii) Re-programming (scrubbing) the faulty VHC using the same VHC configuration bitstream.

iv) Continuing the data-stream computation in the restored ASVP.

v) Doblecheck ASVP performance.

vi) If fault still exist in the same CLB-column, increment the number of counted restoration cycles in this CLB-column and return to the step ii).

vii) If the number of counted restoration cycles exceed the preset limit initiate the second level of VHC restoration – CLB-column replacement.

Thus, if VHC scrubbing procedure does not help it means that hardware fault is permanent. In the proposed restoration procedure we do not analyse the reason which causes this fault. In this case an entire CLB-column where VHC was located is considered to be damaged and has to be excluded from further utilization. To restore the functionality of ASVP, VHC that is where the hardware fault was detected has to be re-located to another

CLB-slot(s). This is guiding us into the second level of self-restoration mechanism – CLB-column replacement.

6.2 CLB-column (slot) replacement

When the Hardware Operating System (see Figure 5.5) detects the case when the first level of self-restoration mechanism does not restore ASVP functions (when the counter of scrubbing cycles exceeds the limit) the second level of self-restoration mechanism: CLB-slot replacement is activated by the HOS.

The following can be considered to be the process of replacement of damaged slot by reserved ones. This process consists of the following steps:

i) Pausing the damaged ASVP pipeline.

ii) Disabling damaged Virtual Hardware Component by loading "dummy" VHC (marked as * in Figure 6.1) to tri-state all outputs and thus, to prevent data contention on the bus;

iii) Selecting the available CLB-column(s) for loading configuration bitstream of VHC to be restored;

iv) Composing the configuration bit-stream by inserting the selected slot frames addresses and associated information into the VHC-configuration data file.

v) Loading the VHC configuration file to the selected column(s) while continuing data-stream processing.

In the case of a lack of spare CLB-columns, HOS detects this fact and initiates the third level of self-restoration procedure. In this case different variant of VHC with reduced area

and performance should be loaded to the non damaged part of the same CLB-column. For example, a 16-bit VHC architecture can be replaced by the 8-bit VHC, which can restore a ASVP, but with a reduced performance. This will be discused in more ditail in Section 6.3.

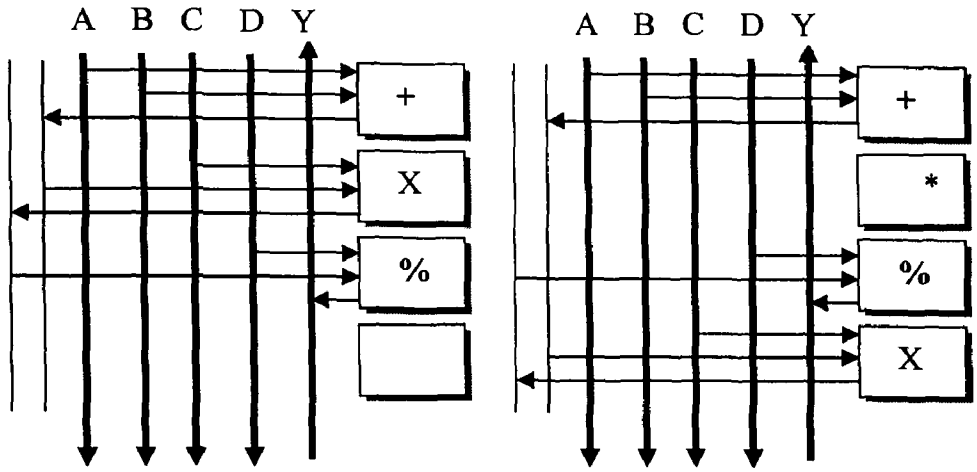

Figure 6.1: VHC replacement using spare CLB-slots.

In this example a fault is detected in the CLB-column where Multiplier (X) VHC was located. The "dummy" VHC (indicated as *) is loaded to the faulty CLB-slot and then, Multiplier VHC is loaded to one of spare CLB-slots.

6.3 ASVP functional restoration with performance degradation

In this situation let us consider that all spare CLB-columns were already used or were not reserved at all. We also should put in account the fact of hardware fault in some CLB-slots, which means that one or few transistors in one or few logic gates are damaged. A CLB-column consists of 20,000 up to 50,000 transistors (depending on FPGA device specifics). Therefore, damaged logic gates can be avoided by reloading other variants of Virtual

Hardware Component - *Ri*, which requires less logic gates. In result we have considered two possible solutions to restore ASVP with performance degradation:

a) First approach – "Minimization of restoration time" – implies reloading another variant of the same VHC (e.g. Multiplier), which requires less area, to the same CLB-slot. This approach gives the shortest restoration period and a simplest restoration control scheme. However, this solution may not provide minimum of performance degradation. For example, if a fault appears in a CLB-slot where Multiplier is located it may be more efficient to relocate Multiplier as is to another slot and load the damaged slot by a smaller variant of another VHC (e.g. Adder).

b) Second approach – "Minimization of performance degradation" – implies that before functional restoration an optimisation of ASVP architecture to the task algorithm & data structure should be done. This procedure should select the variant of ASVP architecture which satisfies new (after fault) resource limitations and allows minimization of the performance degradation (e.g. latency, data processing rate, etc.)

6.3.1 ASVP restoration with minimum restoration time

Firstly the approach which allows minimization of the restoration time was considered. The restoration procedure consists of the following steps:

i) Suspension of damaged ASVP pipeline.

ii) Selection of one of VHC variants with reduced logic area and performance (usually in this VHC some part is "dummy" and thus different VHC variants have different "location" of the "dummy" component).

iii) Loading configuration bit-stream of selected variant of VHC to the damaged CLB-slot.

iv) Continuation the data-stream processing on restored ASVP.

v) If hardware failure still affects data-stream processing the next VHC variant with reduced area and performance should be selected from the VHC table.

vi) Return to the point iii) of presented procedure.

This is a very general description of the restoration procedure. The detailed description of this procedure is specific for Xilinx Virtex FPGA devices and requires detailed explanation of organization of this family of FPGA devices. More detailed this procedure will be discussed in Chapter 7.

6.3.2 ASVP restoration with minimum performance degradation

Generally, if the goal is to minimize performance degradation when amount of logic resources had decreased, the entire ASVP architecture should be reconsidered in this regard. This means that ASVP architecture must be once again optimized to the task DFG and data structure but with more strict requirements for hardware resources. In the case of self-restoration this architecture-to-task optimization procedure should be performed: a) automatically without operator and b) as fast as possible.

To solve this problem the method of data-flow architecture to task adaptation [67] was implemented and modified for ASVP self-restoration specifics. As it was mentioned in Chapter 5, ASVP is assembled by Virtual Hardware Components (VHC). Each VHC can be presented in the system architecture in few different variants – Ri,j, where i – indicates the type of resource (adder, multiplier, shifter, FFT, etc.) and j- indicates the variant of resource

presentation in the architecture (for example: 8-bit adder, 16-bit adder, etc.). Graphically *Ri,j*, can be presented as it is shown in Figure 6.2.

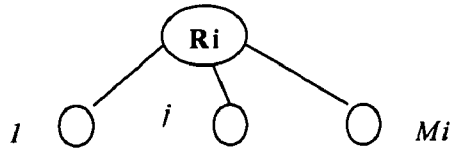

Figure 6.2: Graphical presentation of a Virtual Hardware Component - *R i, j*

Assuming that ASVP architecture contains *n* types of different VHCs – *R1* .. *Rn*, (reflecting operators in the task DFG) all set of ASVP architecture configurations based on *R1* . . . *Rn* VHCs can be presented as a Tree, shown in Figure 6.3.

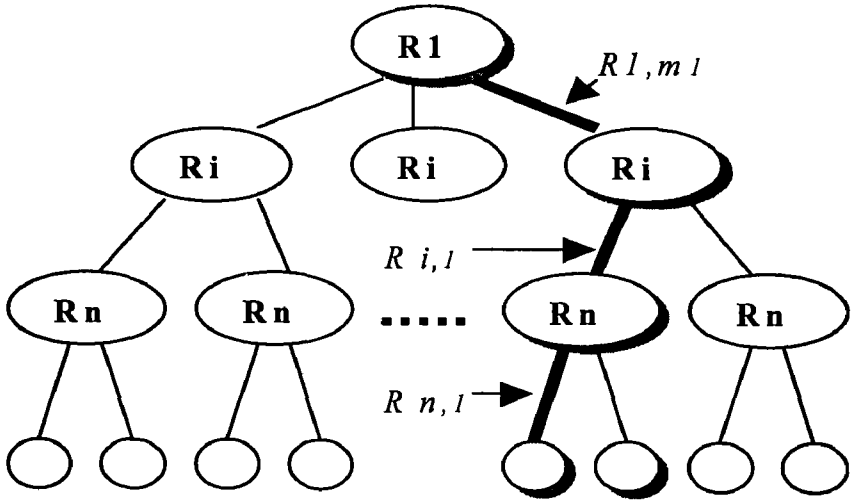

Figure 6.3: Architecture Configurations Graph - ACG

This graph represents the ASVP design space and thus, was called the Architecture Configurations Graph or ACG. On the ACG the vertices are associated with types of Virtual

Hardware Components - *R1..Rn* and the edges are associated with possible variants of VHC presentation in the architecture. Thus, each path from the root to any terminal vertex on ACG presents one of possible variant of architecture configuration. For example, in Figure 6.3 is the bold path: *{R 1, m1 → R i,1 →... → R n,1 }* can be interpreted as ASVP that consists of:

{32 bit-multiplier; 16 bit-adder and 16-bit logic module}

To select the variant of architecture configuration that satisfies performance requirements and needs a minimum of resources, it is necessary to estimate performance on all possible ASVP architecture variants and select the best one. In the majority of real applications it is very difficult or even impossible to do this because the number of all possible architecture variants is very high. However, in [69] was proposed an approach for a strong decrease of number of estimated variants of ASVP architecture configurations using partial arrangement of the ACG in order of increasing of value of performance parameter *Ps* (e.g. cycle time, latency or complete computing time for a block of data, etc.). It was shown, that appropriate partial arrangement of the ACG can be done by performing two procedures:

1. Local arrangement of the VHC-variants in order of decreasing the performance parameter (e.g. complete processing time for a data block) - *T(Ri,j):*

$$T(Ri1) > \ldots > T(Rij) > \ldots > T(Ri,mi) \text{ for all } i = 1 \ldots n$$

2. Hierarchic arrangement on the ACG of all VHC *R1..Rn*, associated with respective operators on the task DFG.

Local arrangement for each type of resource can be done in most practical cases without calculations or any other type of estimation. For example, if in architecture *Ai* is used an

adder with higher bit rate then in *Aj* architecture, it is clear that performance of *Ai* will be at least not less then performance of architecture *Aj*.

For hierarchic arrangement of ACG it is necessary to perform following steps (see Figure 6.4):

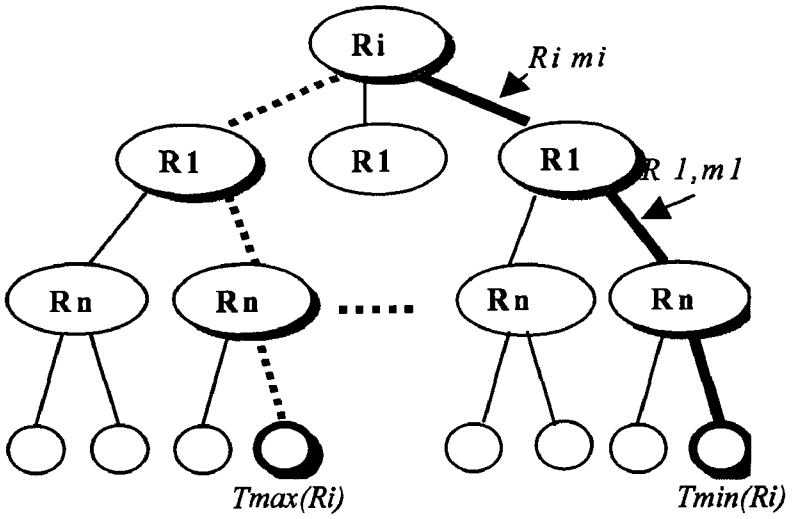

Figure 6.4: ACG hierarchic arrangement

1. Associate each VHC - *Ri* with ACG root;
2. Measure the minimum level of performance parameter (data-block processing time, latency, etc.) – *Tmin(Ri)* for the ASVP architecture variant with maximum used resources (see bold path form the top to bottom vertex in Figure 6.4):

$$Tmin(Ri) => \{ Ri, mi; R1,m1; \ldots Rn,mn \}$$

3. Measure maximum level of performance parameter - *Tmax(Ri)* for the ASVP architecture variant with minimum used logic resources of the VHC *Ri* (marked on Figure 6.4 by dotted lines): $Tmax(Ri) => \{ Ri,1; R1,m1; \ldots Rn,mn \}$

4. *Calculate the value of arrangement criterion – K(Ri):*

   $K(Ri) = \frac{T\max(Ri) - T\min(Ri)}{mi - 1}$, where mi – is number of possible variants of the VHC Ri presentation.

5. Repeat steps from # 1 to # 4 for the all types of VHCs used in ASVP
6. Arrange VHCs hierarchically on the ACG levels according to the rule:

   "*Ri* should be put on higher level than *Rj* in ACG if $K(Ri) > K(Rj)$"

When ACG is arranged, the value of the performance parameter (e.g. processing time) on each possible variant of ASVP architecture associated with respective terminal vertex will decrease from the left to the right terminal vertex.

If required performance – *T lim* is given in the specification, there are a few well known methods and algorithms [De Michelly] to find the closest path (ASVP variant) on the partially arranged ACG associated with respective architecture configuration – *Aopt* so that $T(Aopt) \leq T\,lim$.

Thus, we can select an architecture configuration *Aopt*, (bold on Figure 6.5), which satisfies performance requirements ($T(Aopt) \leq Tlim$) and on the other hand needs minimum hardware resources.

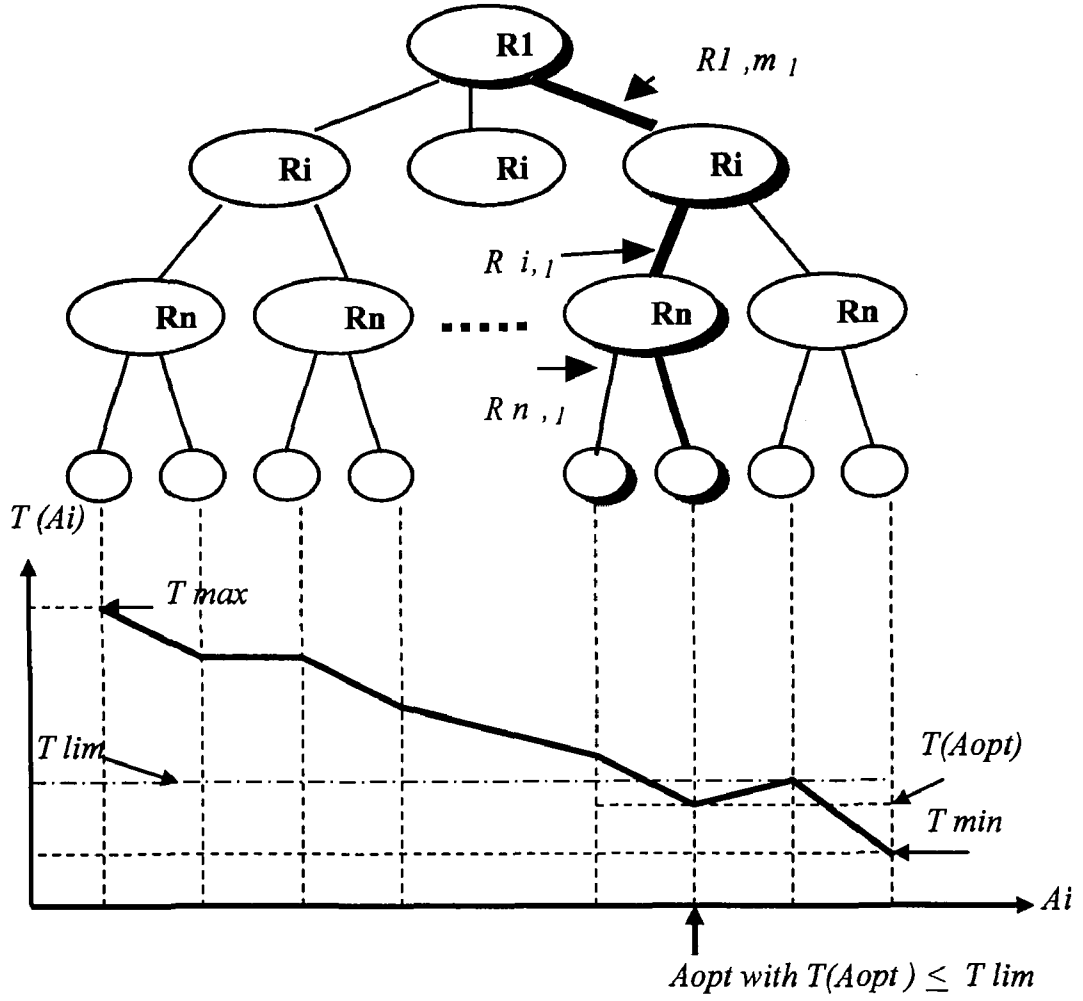

Figure 6.5: Selection of the optimal variant of ASVP processing architecture on the partially arranged ACG

For fast selection of the optimal variant of ASVP architecture the number of architectural variants to be estimated on the ACG should be minimal. The total number of ASVP configuration variants, which are necessary to estimate for selection of the *Aopt* can be found by the following equation:

$$N(Aopt) = (n+1) + \log_2(\prod_{i=1}^{n} mi)$$

Where: $(n + 1)$ is number of variants to be estimated for ACG partial arrangement and

$\log_2(\prod_{i=1}^{n} mi)$ is number of variants to be estimated for selection the *Aopt* on the partially arranged ACG. Therefore, for *Aopt* selection it is necessary to estimate a very small number of variants on ACG. For example, assuming that the total number of types of VHC utilized for ASVP architecture is 32 and each VHC can be represented in 32 different variants (32 different configuration files for each virtual component). Thus, to select the *Aopt* for this case we need to estimate:

$N\ (Aopt) = (32+1) + \log_2(32^{32}) = 193$ variants of ASVP architecture configurations when the total number of architecture configuration variants on ACG is equal to:

$N\ total = 32^{32} = \sim 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000$

Therefore, the above method allows selection of ASVP architecture configuration on the enormous set of possible configurations within very short time and thus, can perform this selection procedure in real-time. In the [69] was shown that the implementation of this method for ASVP selection can be done within hundreds of milliseconds.

Now let us consider the extension of the above method on application in self-restoration with minimum performance degradation. As was assumed earlier (in Chapter 4) the hardware fault in our consideration can occur only in one or very few transistors in local area of the FPGA device. Being an element of pipelined data processing structure of ASVP, it results fault of a complete VHC associated with the CLB-column where the damaged transistor(s). If no one previous levels of self-restoration does not restore ASVP functionality, the third and last level of restoration has to be initiated. However, at this moment of time the ACG of damaged ASVP is already arranged and thus hierarchical arrangement of VHCs included in

this ASVP is known. Therefore, using the hierarchical arrangement of the ACG it is possible to minimize performance degradation by a replacement of that VHC which is located on the lowest level of the ACG because it can cause the smallest degradation of performance parameter (see Figure 6.6).

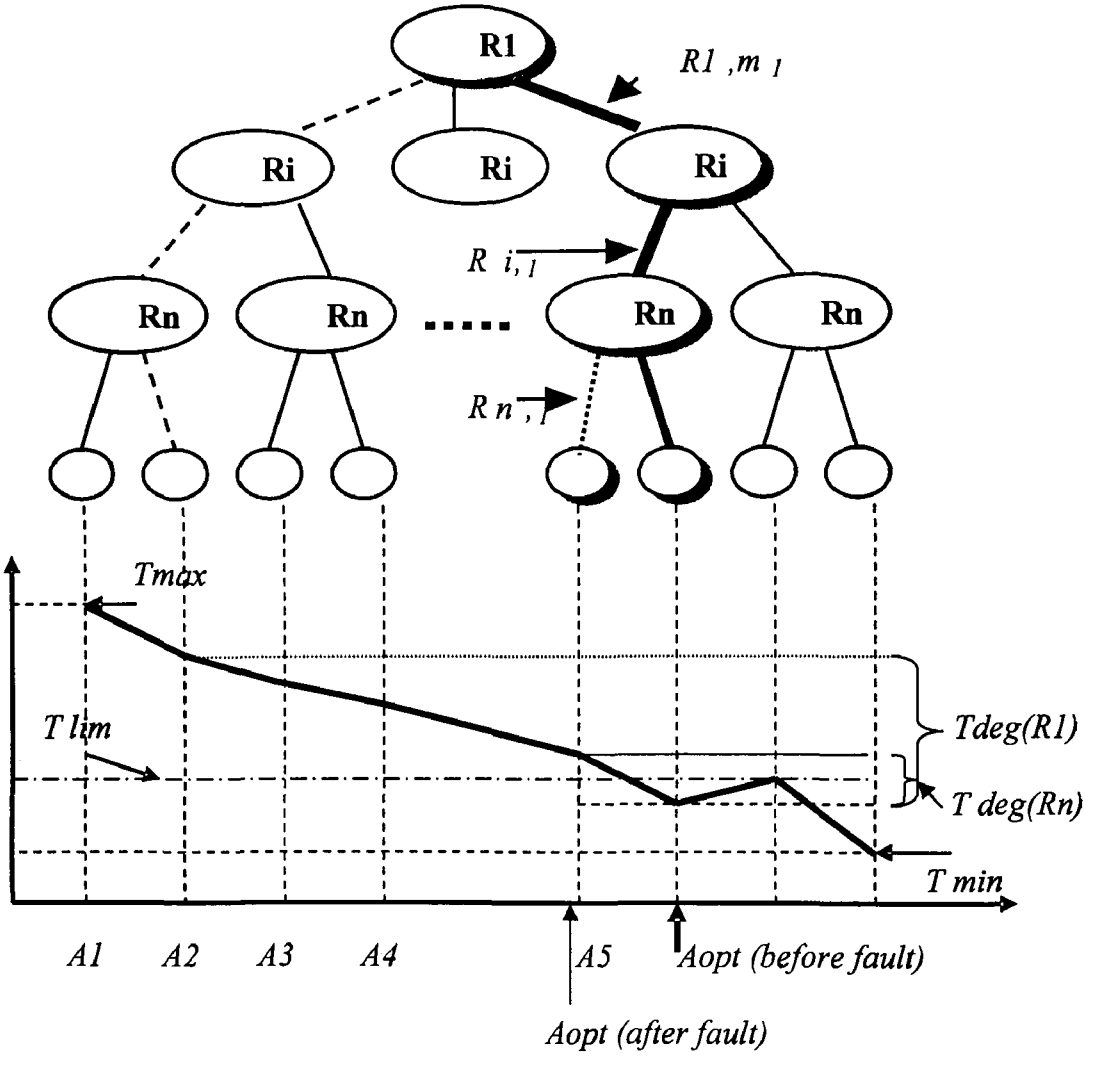

Figure 6.6: Architecture Configurations Graph (partially arranged Design Space) for an ASVP (Top tree) and performance diagram associated with ASVP variants.

For example, let us assume that ASVP architecture: *{R1,m1 → R i,1 →... → Rn,mn}* (marked bold on the ACG presented in Figure 6.6) was determined as optimal for a task. Let's also

assume that a hardware fault has occurred in CLB-slot where VHC -*R1* is located. In this case VHC - *R1,m1* (for example 32-bit adder) has to be replaced by smaller variant of VHC - *R1* (for example VHC- $R_{1,1}$: 8-bit adder) the architecture will change from *Aopt* to variant *A2* This will result performance degradation *Tdeg(R1)*.

Let us assume for simplicity that VHC - *R1,m1* and VHC – *Rn,mn* require equal amount of logic resources (for example – 2 CLB-columns) as well as amount of logic resources for VHC - *R1* variant $R_{1,1}$ is equal to amount of logic resources for VHC – *Rn* in variant $R_{n,1}$ (for example 1 CLB-column). However, if we change architecture *Aopt* to variant *A5* changing VHC from variant R_{n,m_n} to $R_{n,1}$ it will cause performance degradation equal to *T deg(Rn)*. This level of performance degradation is much smaller than *Tdeg(R1)* because VHC – *Rn* stands in lower level of the arranged ACG when *R1* stands in the top level of ACG. Now, let us re-load the configuration file of VHC - *R1,m1* to the location (CLB-slots) where VHC - R_{n,m_n} was mapped. It is possible to do because logic area for both VHC variants is equal. After that we can re-load smaller version of *Rn*

the $R_{n,1}$ configuration file to non-damaged CLB-column where previously VHC - *R1,m1* was allocated. In this case the architecture *A5* is created $\{R1,m1 \rightarrow R\ i,1 \rightarrow \ldots \rightarrow Rn,1\}$ and the performance degradation *T deg(Rn)* is minimal comparing with any other variant of possible architectures. It is because the *A5* is the "closest" terminal on the arranged ACG to the *Aopt* - optimal architecture variant before the fault.

This example shows that replacement of VHC allocated on the lower level of ACG than damaged one can restore data processing with minimum performance degradation. Thus, for restoration with minimum performance degradation we have to check if there are VHCs

other than damaged one located in lower levels of ACG, which require same or bigger area (logic resources). If answer is "Yes", than the following procedure can be applied assuming that data-stream processing was already suspended on previous stages of ASVP restoration:

i) Create list of VHCs and their variants included in ASVP architecture need to be restored.

ii) Select the one VHC variant from the above list, which is located on the lowest level of ACG associated with damaged ASVP.

iii) Re-load the configuration file of the damaged VHC to the area where VHC selected in ii) was allocated.

iv) Select on partially arranged ACG the "next left" variant of VHC which was replaced by damaged VHC in step # iii) of this procedure.

v) Load the selected in step # iv) "next left" VHC variant to the non-damaged area of CLB-columns where the damaged VHC was originally located.

vi) Activate data processing on the restored ASVP.

This is a general procedure, which does not include many technical details such as: automated mapping procedure, data-stream suspension procedure, etc. These details are chip specific and thus, are reflected in associated circuitry of the Hardware Operating System. This procedure at this moment of time is not fully automated. However, all of the above mentioned steps were implemented on the prototype platform and tested manually.

The step-by-step analysis of the implementation of the above procedure is presented in Chapter 7 of this thesis.

CHAPTER 7

IMPLEMENTATION OF THE ELEMENTS OF SELF_RESTORATION MECHANISM AND ANALYSIS OF THE RESULTS

7.1 Test Platform

All levels of self-restoration procedures were implemented and tested on the base of multi-task and multi-mode Reconfigurable Parallel Stream Processor (RPSP). This data-stream processing platform (see Figure 7.1) was developed with utilization of the concept of AGORA: Adaptive Group Organized Reconfigurable Architecture [74].

Reconfigurable Parallel Stream Processor (RPSP) architecture, adaptable for the multi-task and multi-mode workload, implements the ASVP architecture-to-task adaptation mechanism described in Chapter 6. Architecture of RPSP includes the following components described in Chapter 5:

i) Re-configurable Functional Module (RFM) is based on partially re-configurable Xilinx Virtex-E FPGA device XCV-400E and contains all of the necessary circuits: power regulators, line buffers, etc. This module is a field of uniform logic resources that can be configured to a number of task-optimized Application Specific Virtual Processors.

ii) Task memory and VHC memory is based on 20 GB Hard Drive in this implementation incorporated in PC Pentium III. This memory contains the library of configuration data files (configuration bit-streams) of all initial architectures of ASVPs and associated Virtual Hardware Components to be utilized in all modes of all tasks of the workload.

iii) VHC Cache is based on SRAM (Static Random Access Memory) IDT71V416S (4 SRAM chips x 4Mb). This Cache module is able to store all VHC cores that can be used for up to 4 active tasks running in parallel at RFM,

iv) Hardware Operating System (HOS) is based on Xilinx Virtex-E FPGA XCV-50E and performs the following functions:

a) Task initiation & termination,

b) Mode switching by loading set of VHCs to certain CLB-slots in RFM,

c) Switching data-streams,

d) Performing interface control,

e) RFM diagnostic and restoration functions,

v) Dynamically reconfigurable LVDS Input / Output subsystem with the bandwidth equal to 7.2 Gb/sec,

vi) Statically configurable LVTTL bus interface with aggregate bandwidth equal to 8.5 Gb/sec.

Host-PC is based on Pentium III with re-configurable high bandwidth (~1Gb/s) interface module (see Figure 7.2). It plays role of human-machine interface, high-level (Software) Operating System for RPSP and secondary storage for all ASVPs and associated VHC. It also contains all instrumentation CAD tools.

The block diagram of the Reconfigurable Parallel Stream Processor (RPSP) is presented in Figure 7.1. RPSP consists of two major parts: RPSP-platform and Host PC.

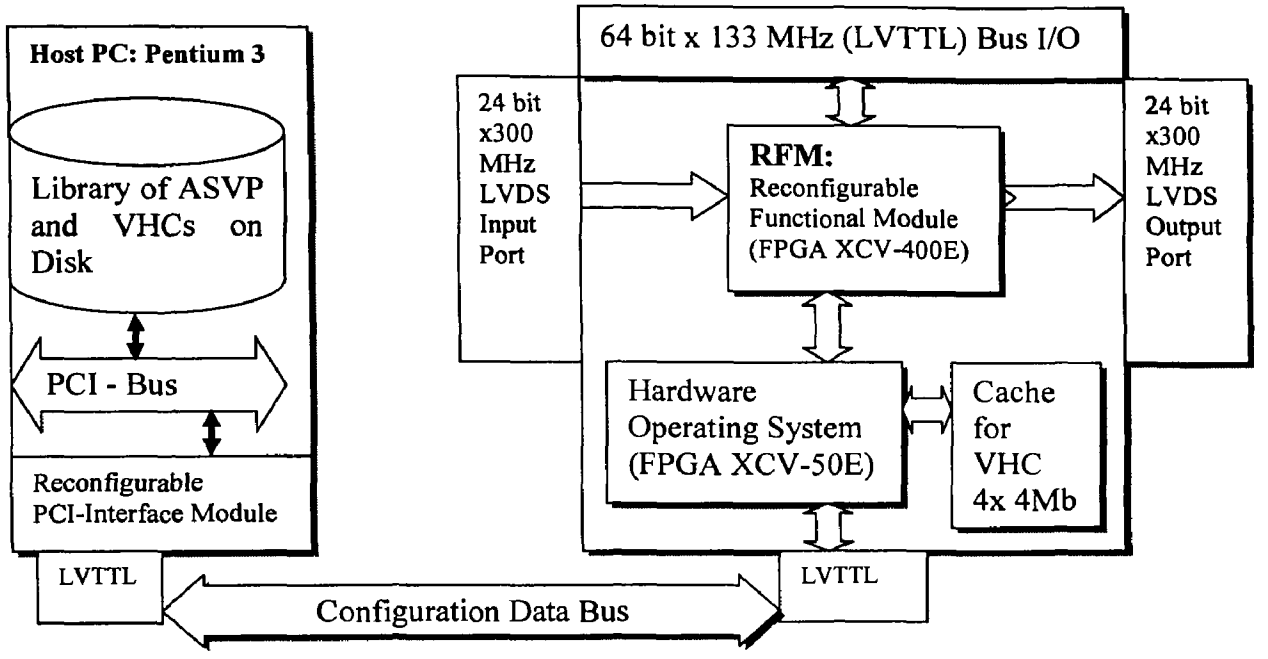

Figure 7.1: Block diagram of the Reconfigurable Parallel Stream Processor (RPSP)

The RPSP-platform is presented in Figure 7.2. In the centre of the platform the RFM is located (mezzanine board). The HOS FPGA and Cache SRAM are located under RFM board when configuration data bus is connected to the bottom DB-15 and DB-25 connectors. Reconfigurable PCI-Interface module is presented in Figure 7.3 with attached 4 M bit SRAM-buffer (mezzanine board).

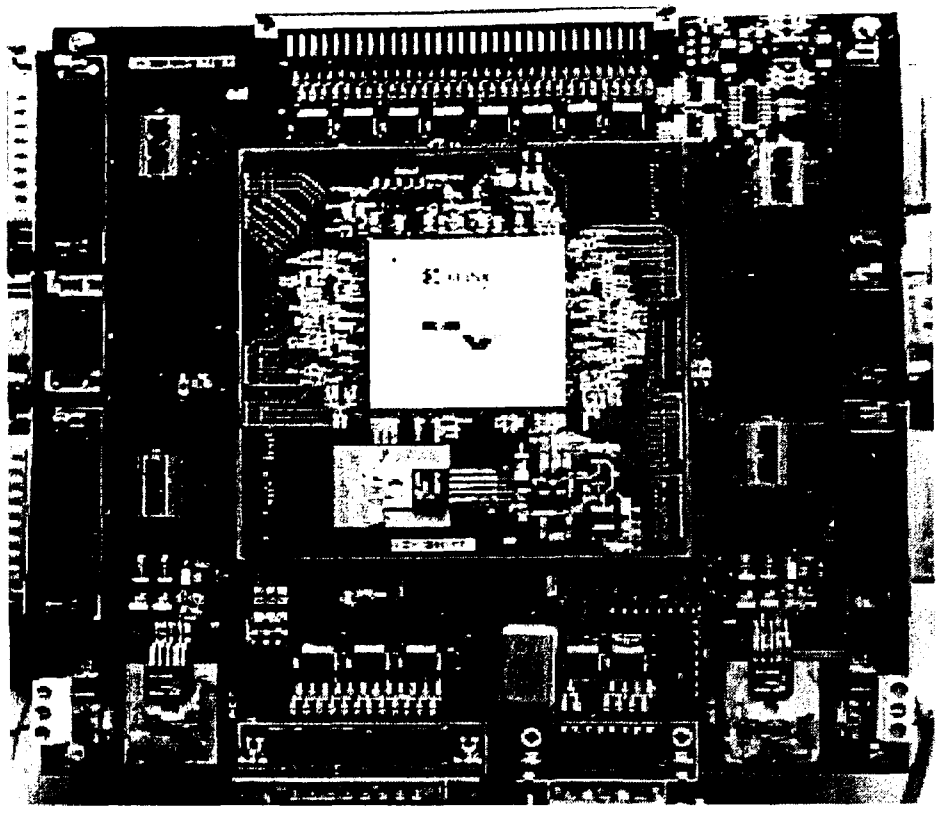

Figure 7.2: Reconfigurable Parallel Stream Processing platform *

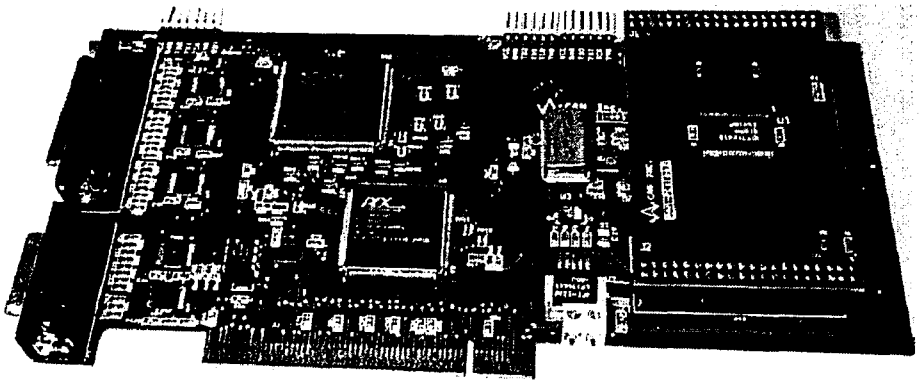

Figure 7.3: Reconfigurable PCI-interface Module *

* Pictures 7.2 and 7.3 courtesy of Materials & Manufacturing Ontario (MMO) and CAN Inc.

7.2 Implementation of procedures for restoration without performance degradation

To test and adjust ASVP restoration procedures a set of "Faulty" Virtual Hardware Components (FVHCs) was developed for parallel video-stream processing tasks such as: Video Edge Detection (ASVP1: "VED"), Brightness-to-Colour Conversion (ASVP2: "B2C") and Run-time Image Combination (ASVP3: "RIC").

Each FVHC was developed on a base of normal VHC by corrupting some interconnection on its micro-architecture. The example of one of VHC (for ASVP2 B2C) is presented in Figure 7.4.

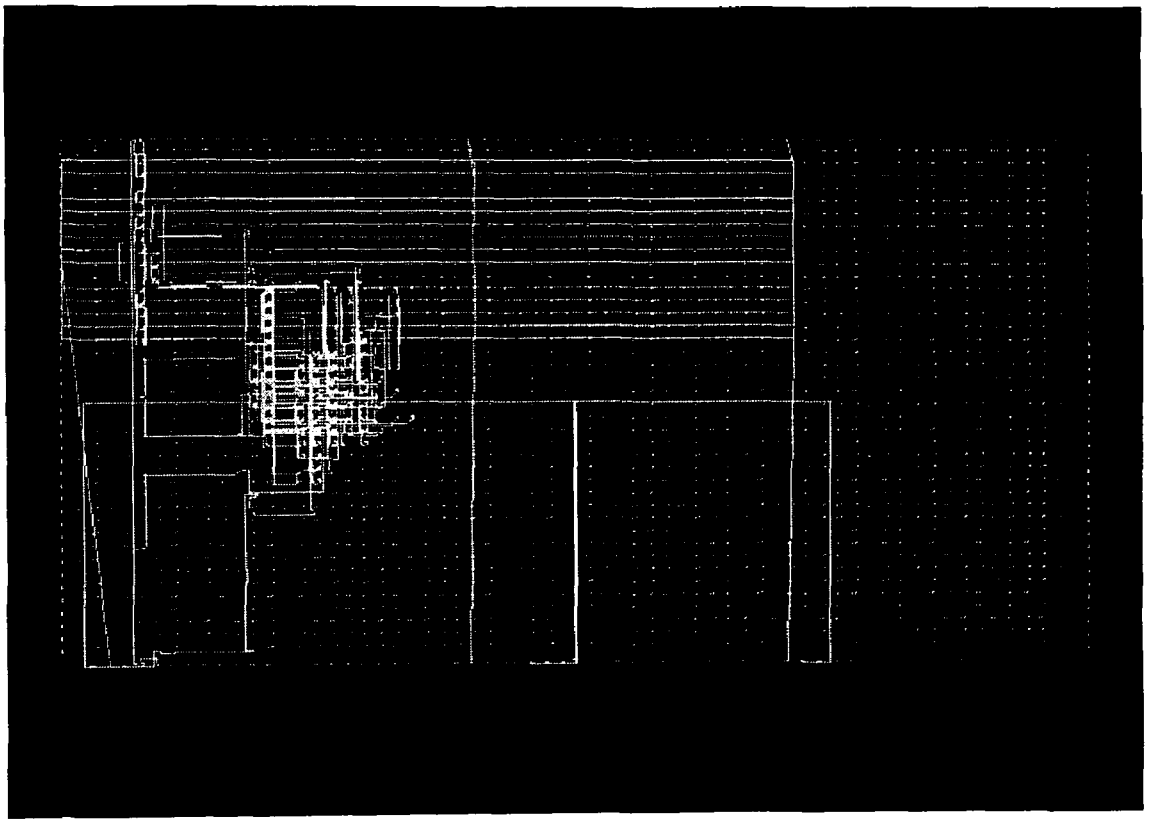

Figure 7.4: VHC with corrupted interconnection on its micro-architecture.

The experiment included the following steps:

i) Initiation of ASVP and processing test-data stream.

ii) Loading FVHC in selected CLB-slot at random moments of time.

iii) Initiation of first level of self-restoration procedure (VHC scrubbing).

iv) Measurement of scrubbing time till ASVP restoration.

v) Repetition of steps ii) – iv) to initiate second level restoration.

vi) Initiation of CLB-slot replacement to the spare CLB-slot.

vii) Measurement of CLB-slot replacement time.

Results of the above experiments are summarized in Table 7.1. We have measured restoration periods for Level 1 (VHC-scrubbing) and Level 2 (CLB-slot replacement). Based on this information we calculated the acceleration of the functional restoration comparing with common approach when an entire FPGA device has to be re-programmed (See Table 1).

These results show that restoration time directly depends on size of VHC.

Restoration time is minimal if VHC can be allocated in one CLB-column and increases nearly linear when VHC requires 2, 3 or more CLB-columns.

Another result is that CLB-slot (column) scrubbing (1st level of restoration) works twice faster than CLB-slot replacement. This effect appears because of the extra time needed for a "dummy" VHC loading. Thus, for maximum acceleration (comparing with the scrubbing of entire FPGA) designers should develop a Virtual Hardware Components close to 1 CLB-column. This can dramatically increase the restoration time (up to 70 – 100 times depending on FPGA volume and organization), for the reason that it was mentioned in [8] most of hardware faults in LEO and GEO are caused by the SEU which can be restored by scrubbing procedure.

Table 7.1: Experimental results of functional restoration time and acceleration of functional restoration comparing with entire FPGA re-programming (all results collected for Xilinx Virtex XCV-400E FPGA)

Number of CLB-Slots in the ASVP	**1**	**2**	**3**	**4**
Restoration time if VHC-scrubbing	0.09 ms	0.17 ms	0.25 ms	0.33 ms
Restoration time if CLB-slot replacement	0.18 ms	0.34 ms	0.50 ms	0.66 ms
Acceleration (in times) of restoration for scrubbing	73.3	38.8	26.4	20
Acceleration (in times) of restoration for CLB-slot replacement	36.6	19.4	13.2	10

7.3. Implementation of procedures for restoration with performance degradation

This implementation was performed in capacity of feasibility study for real-time image recognition of the hyper-spectral sub-system based on the new generation of on-board computing platform for perspective Canadian satellite RADARSAT-2.

7.3.1 Specifications for ASVP: "Hamming Distance Calculator"

The ASVP "Hamming Distance Calculation" (HDC) should perform processing of data streams coming from the array spectral sensors and compare it with special spectral code-vectors from the on-board codebook. The process of spectral scanning on the Earth surface is shown in Figure 7.5

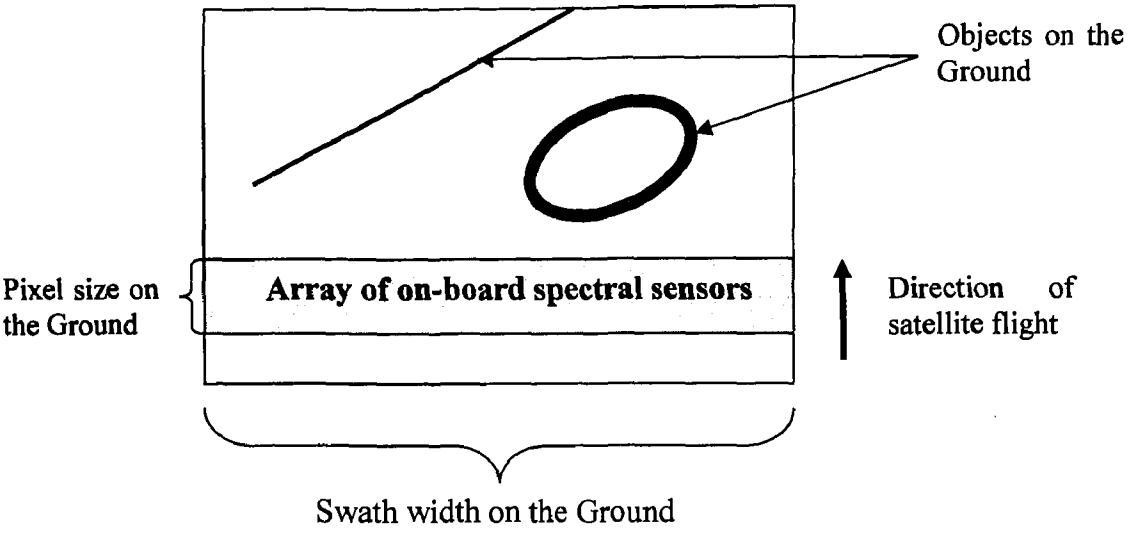

Figure 7.5: Spectral scanning on the Earth surface by the array of spectral sensors

The data stream processing consists of the following steps:

1. Parallel reading of spectral data from array of sensors for each of spectral bands
2. Analogue-to-Digital Conversion (ADC) of spectral levels for each sensor and comparison with preset threshold. If level is higher than associated threshold than "1" is stored to the respective register cell, otherwise – "0". Thus at this step the spectral vector from appears for each of spectral bands.
3. Hamming Distance Calculation (HDC) between spectral vector from sensor's array and spectral code-vector (from spectral code-book). Results of comparison with each

code-vector for each spectral band are collected in special memory unit for further processing.

In our implementation only the Step #3 was considered. This HD-calculation should be performed by the formula: $Dh\ (Uj, Vi) = \sum_{l=1}^{N_b} uj\ (l)\ XOR\ vi\ (l)$, where

$Uj = [uj\ (1) \ldots\ uj(Nb)]$ – is a spectral vector from the sensor's array

$Vj = [\ vj\ (1) \ldots\ vj(Nb)]$ – is a spectral code-vector

Technical specifications consist of the following data:

1. Earth observation accuracy – 1.0 m * 1.0 m on Ground = 1pixel size.
2. Swath width on the Ground – 2048 m (2048 pixels / track). Thus, $Nb = 2048$ in this case
3. Orbital period = 100 min. equal to 150 uS per 1 pixel row (on Ground).
4. Number of spectral bands to be analyzed - 240

The Data Flow Graph for Hamming Distance Calculation reflecting the above analytical equation is presented in Figure 7.6.

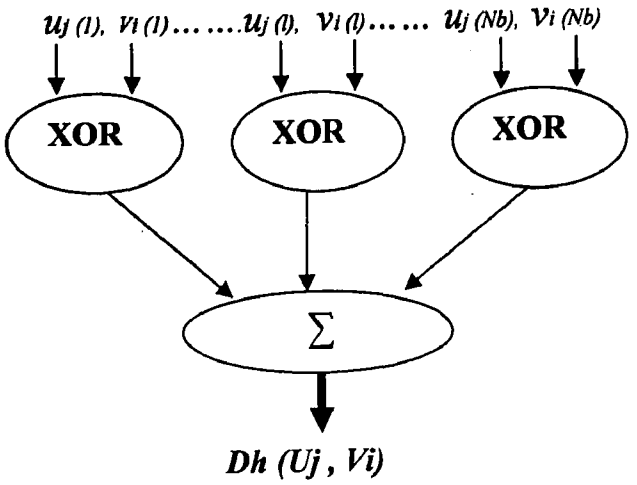

Figure 7.6: Data Flow Graph for Hamming Distance Calculation (HDC)

7.3.2. Hamming Distance Calculator Architectural Representation

Hamming distance calculator should contain two types of resources associated with respective DFG macro-operators: XOR and Adder and functional model of FPGA-based systems. Because FPGA-based processing circuits must be developed as synchronous digital circuits another component of HDC processor has to be internal clock generator. Thus, two Virtual Hardware Components (VHCs) were developed for this implementation in several variants:

1. **VHC1 (*R1*):** N-channel HDC => N-channel Adder &N-channel XOR, where:

   ***R1.1*:** 8 channel HDC

   ***R1.2*:** 16 channel HDC;

   ...

   R1,10 : 4098 channel HDC

   Example of ***R 1.1*** and hardware organization of this variant of VHC1 is presented in Figure 7.7. VHDL-code of ***R 1.1*** and ***R 1.2*** are attached in Appendix A

2. **VHC 2 (*R 2*)**: Clock generator for HDC where:

R2,1 : 80MHz clock generator

R2,2: 100MHz clock generator

R2,3: 120MHz clock generator

***R2.4*:** 140MHz clock generator

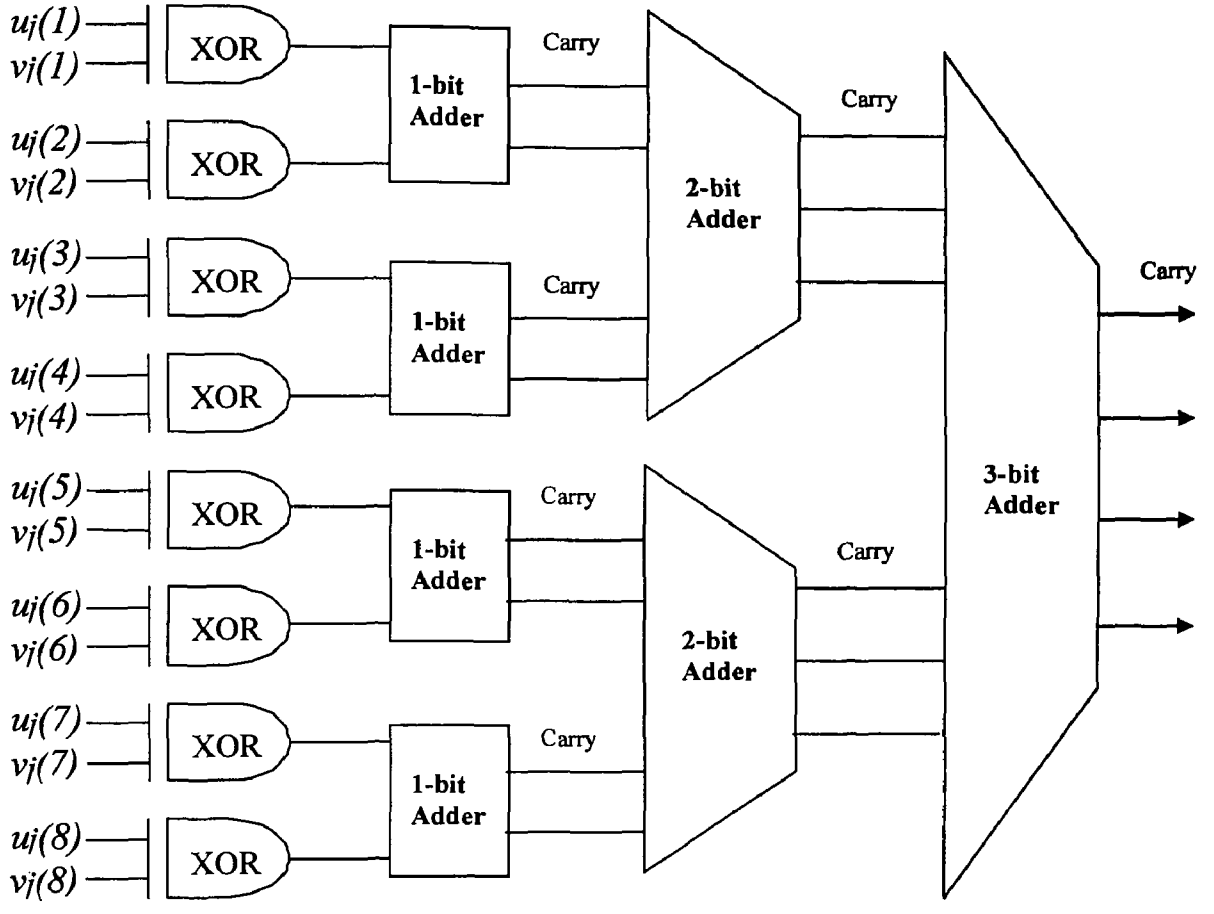

Figure 7.7: Schematic diagram of resource ***R1,*** Variant ***R1,1***: 8- XOR channels & 8-bit Adder

Now, according to Section 6.3, all possible variants of architecture configurations of the Hamming Distance Calculator can be presented graphically as an Architecture

Configurations Graph (ACG) presented in Figure 7.8. On this ACG each path from the root to a terminal vertex represents one of possible variants of Application Specific Virtual Processor (ASVP) for Hamming Distance Calculation (HDC). For example, the bold path: *{R 1,1& R 2,4 }* represents 8-channel HDC module coupled with 140 MHz Clock generator.

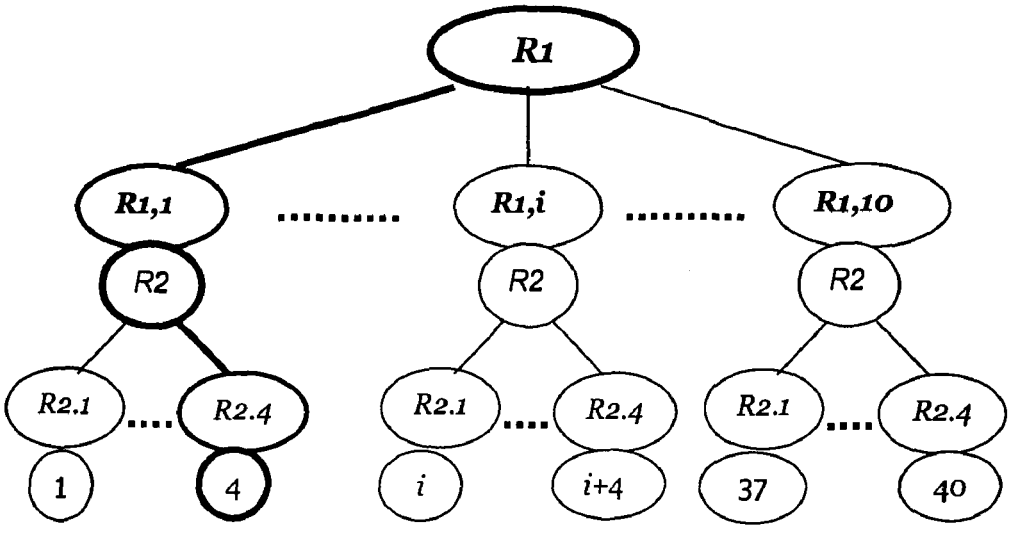

Figure 7.8: Architecture Configurations Graph (ACG) for HD-calculator variants

7.3.3 Performance Estimation Model

To create the performance estimation model the data computation process analysis was performed. As a result of this analysis the following analytical model was determined:

$$T\,process = (T\,latency + T\,pipeline * [(N\,pixels / N\,channels) - 1]) * N\,bands$$

This analytical model allows calculate the period of time while one row of pixels – *N pixels* will be compared with the spectral code-vectors in all spectral bands – *N bands* taking in account a pipeline principle of data processing.

In this analytical model:

T latency = T xor + T Adder; where: T xor – propagation delay of data in XOR- elements and T *Adder* – propagation delay of data in all stages of n-bit adder.

T xor = Number of clock cycles for XOR / Clock rate

T Adder = Number of clock cycles for Adder / Clock rate

T pipeline = Number of clock cycles for pipeline stage / Clock rate

The example of processing of data array for *N pixels = 32* and *N bands = 1* on the 8-channel XOR coupled with 3-stage Adder (VHC variant - ***R1.1***) is shown in Figure 7.9.

0	**1cc**	**2cc**	**3cc**	**4cc**	**5cc**	**6cc**	**7cc**	**8cc**
u(1-8), v(1-8)	XOR	1Stage Adder	2Stage Adder	3Stage Adder	Output			
u(9-16), v(9-16)		XOR	1Stage Adder	2Stage Adder	3Stage Adder	Output		
u(17-24), v(17-24)			XOR	1Stage Adder	2Stage Adder	3Stage Adder	Output	
u(25-32), v(25-32)				XOR	1Stage Adder	2Stage Adder	3Stage Adder	Output

Figure 7.9: Hamming distances pipeline computational process of spectral and code vectors on the variant of calculator presented in Figure 7.8

This figure (Figure 7.9) shows that deepness of pipeline is equal to 4 and 32 component vectors will be processed within:

T *process* = (T *latency* + T *pipeline* * *[(N pixels / N channels) – 1])* * *N bands* =

T process = (1 c.c. + 3 c.c. +1 c.c. * [32 / 8 – 1]) * 1 = 7 c.c.

To check the accuracy of this analytical model the VHC variants ***R1.1 and R 1.2*** where implemented in a form of virtual hardware (VHDL-cores) compiled, simulated using Xilinx ISE 4.2 CAD software and emulated on the special prototype platform (see Appendix "A").

The emulation results didn't exceed a 5% instrumental error of real T *process* measurement. Thus, the above analytical model for data processing time estimation was accepted as satisfactory and all further calculation was performed on the base of this model.

7.3.4. Architecture-to-Task Optimization for Hamming Distance Calculator

To find the optimal ASVP architecture of HDC for the above data-flow task let us consider all steps of the method described in Section 6.3:

- *Hierarchical arrangement of the ACG*

1. Associate VHC1 - ***R1*** with ACG root (See Figure 7.8)
2. Measure the minimum of processing time – *min*{T *process(**R1**)*}for the ASVP architecture variant (Terminal #40 on ACG) with maximum used logic resources. This variant consists of: ***{R1,10 & R2,4}*** => 4098-channels HDC coupled with 140 MHz Clock generator.

   Using the Performance estimation model described in Section 7.3.3

   T *process **(R1.10 & R2.4)*** = 22.3 uS (microseconds)
3. Measure maximum of processing time - max{T *process(**R1**)*} for the ASVP architecture variant with minimum used *R1* resources (Terminal #4 on ACG). This variant consists of: ***{ R1,1; R2,4}*** => 8-channels HDC coupled with 140 MHz Clock generator:

T *process(R1.1 & R2.4)* = 443.8 uS (microseconds)

4. Calculate the value of arrangement criterion for VHC - *R1*: *K(R1)*, using formula presented in Section 6.3

$$K(Ri) = \frac{T\max(Ri) - T\min(Ri)}{mi - 1},$$ where m_i – is number of possible variants of *Ri* resource presentation.

Thus, $K(R1) = (443.8 - 22.3) / (10 - 1) = 46.83$

Repeating same steps of the algorithm for the VHC2 - ***R2*** the following results will appear:

5. *Tmin(R2)* for the ASVP architecture variant with maximum used resources is the same for all ASVP (Terminal #40 on the ACG).

   Thus, $Tmin(R2) = Tmin(R1) = 22.3$ uS.

6. To measure maximum of processing time for VHC2 - *Tmax(R2)* for the ASVP architecture variant with minimum used *R2* resources (Terminal #37):

   This variant of ASVP consist of: ***{R2.1 & R1.10}***

   $Tmax(R2) = \{ R2,1; R1,10\} = 39$ uS.

7. Calculation of the value of arrangement criterion for VHC2 - ***R2***: *K(R2)*,

   Thus, $K(R2) = (39 - 22.3) / (4 - 1) = 5.57$

In accordance to Step # 6 of the method described in Section 6.3 to arrange ACG it is necessary to associate resource *Ri* with higher level of ACG than *Rj* ($1 \leq i, j \leq n$) if $K(Ri) > K(Rj)$. In our case *R1* should be associated with the higher level of ACG than *R2* because $K(R1) = 46.83 > 5.57 = K(R2)$.

Arranged ACG means that the value of the data processing time on architectures associated with respective terminal vertices will decrease from the left to the right terminal vertex. To check the quality of ACG arrangement we can calculate processing time of hamming distance computation for each of architecture variants of ACG. In our case it is possible because of little number of possible variants (10 x 4 = 40 variants).

Figure 7.10 shows the arrangement of ACG terminals in order of data processing time.

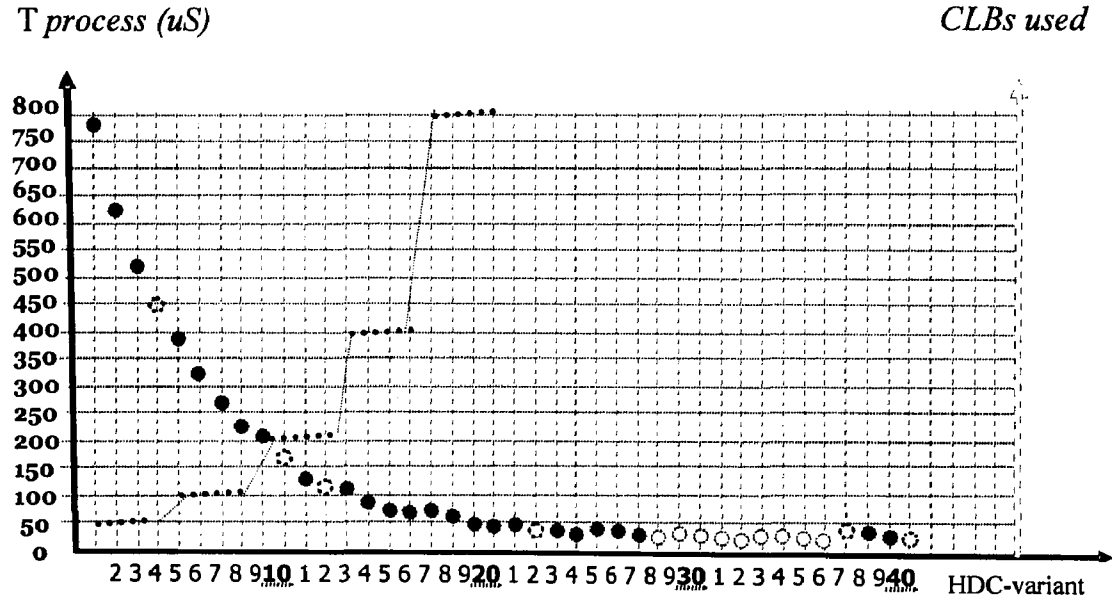

Figure 7.10: Arrangement of architecture variants of Hamming Distance Calculator in order to values of data processing time.

The graph presented in Figure 7.10 shows that values of data processing time monotonously decrease from the variant # 1 to variant # 40. It means that after hierarchical arrangement, which needed estimation of only three ASVP variants (#40, #4 and #37), the 40 ACG terminal vertices are pretty well arranged by the data processing time value.

This result experimentally proves ACG arrangement method, which allowed arranging architecture selection area performing very little number of experiments.

- *Architecture Optimization*

After ACG arrangement in order of data processing time it is possible to find an optimal architecture variant for this task.

There are many possible criteria of optimization. In this paper we consider two of them:

a) Maximum performance with minimum logic resources needed for HDC ASVP;
b) Performance, which satisfies real-time requirements with minimum logic resources needed for ASVP HDC.

For both of above criteria the binary search method or dichotomy can be applied. The difference is only how one can find the data processing time limit.

For the first criteria the role of time limit plays value reached on the architecture variant with maximum logic resources. This variant obviously gives absolute minimum of data processing time. In our case: Variant #40 with T *process* = 22.3 uS. If we take this minimum {T *process*} = limit {T *process*} and apply it as a time constrain for binary search we will find **Variant #28** with the same value of data processing time: T *process* =

22.3 uS (See Figure 7.10) but with much less logic resources (~3200 CLBs instead of ~25,600 CLBs).

The explanation of this fact is that task Data Flow Graph has limited parallelism and cannot be mapped to all possible resources. That is why there is certain amount of logic resources on which task macro-operators can be allocated. Furthermore, for all ASVP variants with numbers higher than #12 doubling of logic resources does not increase much their performance (See Figure 7.10 "CLB used").

For the second criteria when performance has to satisfy real-time requirements with minimum logic resources needed for ASVP the time limit is equal to 150 uS (See specifications in Section 7.3.1). In this case using binary search procedure the ASVP **Variant # 11** with T *process* = 137.5 uS was found. This is the closest to time limit value of T *process* = 150 uS. ASVP variant #11 consist of:

{R1,3 & R2,3} => 32-channels HDC coupled with 120 MHz Clock

- *Minimization of performance degradation*

If some permanent hardware failure occurs and there are no spare logic resources for replacement, the procedure for restoration with performance degradation will be initiated. Assuming that hardware fault has happened in the XOR or Adder circuits the following steps has to be performed according to algorithm described in the Section 6.3.2:

i) List of VHCs and their variants included in ASVP architecture will be created. This list will contain: ***R1,3*** =32 channel HDC and ***R2,3*** = Clock 120 MHz

ii) There is no (in our consideration) any VHC variants from the above list, which is located on the lower level on ACG and requires equal or higher logic resources. That is because clock generator circuit (VHC2) requires much less logic than VHC1 and this logic is specifically adjusted for clock generation and distribution.

iii)Thus, the "next left" variant of VHC1 on the arrange ACG should be selected. This is ***R1,2*** the next left after ***R1,3***. However, to minimize performance degradation the closest left variant to optimal ASVP architecture (Variant #11) should be selected. This closest left variant, which contains ***R1,2*** is the variant # 8:

{R1,2 & R2,4} => 16 channel HDC with clock 140 MHz

In this case the time performance will decrease down to 226.2 uS instead of 137.5 uS. The difference however is minimal comparing with any other solutions (See Figure 7.10). It also has to be mentioned that some spare logic resources appeared after HDC ASVP restoration. As we may see in Figure 7.10, the original (before fault) ASVP, Variant#11 required almost twice more CLBs than ASVP, variant #8 (after restoration).

To compensate reduction of performance the resolution can be reduced from 2048 to 1024 pixels ([1meter * 2 meters] resolution instead of [1 meter * 1 meter] on the Ground) with the original 240 spectral bands or number of spectral bands can be reduced to 130 spectral bands. At the same time current consumption of HDC will also slightly decrease.

CHAPTER 8

CONCLUSION AND FUTURE WORK

The major contribution of this work is the creation of mechanism that allows run-time self-restoration of FPGA-based computing platforms and thus, mitigation of different types of hardware faults. Ideally, this restoration should not cause performance degradation. However, it was assumed that if there is no other choice, it is better to have ability to decrease system performance and restore it rather than allow complete turndown of a system. That is why the proposed mechanism also allows restoration with performance degradation.

Thus, a multi-level self-restoration mechanism was developed based on the idea of self-assembled and task optimized Application Specific Virtual Processors (ASVP). These processors are stored in the memory in a form of configuration files for particular FPGA devices can be assembled on-chip from the LEGO-type Virtual Hardware Components (VHC). These VHCs are small configuration data files for partially reconfigurable FPGAs are representing different processing elements (Adders, Multipliers, FFT or IR-cores, etc.). Because ASVP is VHC-based devices they can be re-assembled inside the FPGA when hardware fault occurs in one of VHCs. Established on this idea special procedures were developed for temporal faults caused by radiation (e.g. SEU: Single Event Upsets) or permanent faults (e.g. physical defects in the wafer of the chip).

To develop this self-restoration mechanism a deep literature research was conducted in the following areas:

1. Reconfigurable computing platforms with the focus on Run-Time Reconfigurable (RTR) computing platforms.
2. Factors that can cause hardware faults in the digital systems with focus on environmental factors (first of all radiation) influencing on SRAM based FPGA devices
3. Methods for hardware fault detection, location and restoration with a focus on fault recovery of FPGA based digital circuits.

After analysis of available literature sources it was found to be that most of the fault restoration methods in the FPGA based computing systems were:

a) Oriented for specific type of hardware faults caused by a specific factor (e.g. scrubbing procedures for SRAM based FPGA devices which mitigates radiation effect called Single Event Upset – SEU)
b) Require spare hardware resources in case of temporal hardware faults. The amount of these resources can be very large (e.g. very popular TMR: Triple Module Redundancy approach)
c) Usually requires operator's control or even redesigning of processing micro-architecture. This prevents any kind of run-time restoration and causes lost of enormous amount of corrupted or non-processed data.

Instead, in this research was proposed approach, which allows avoiding the above limitations ,which were found in the existing methods of restoration of reconfigurable platforms.

Major issues of the proposed approach were the following:

1. Restoration mechanism should be as universal as possible to mitigate multiple (if not all) expected environmental and internal factors causing hardware faults. Although it is very difficult to perform run-time analysis of the reasons caused by hardware fault it was proposed to make restoration mechanism as multi-level procedure, where each level is responsible for specific group of factors that have predictable probabilities. This approach allows avoiding physical analysis of hardware fault, by starting restoration from procedures oriented to mitigate the most probable factors. If this procedure does not help, self-restoration mechanism switches to the next level of restoration oriented on another class of factors with lower probability of hardware faults.
2. Restoration has to be done without any control from the human being and thus, can be performed in real-time to minimize the data loss. Because degradation of logic resources can reach the level when it will be impossible to keep performance of the system at required level, restoration mechanism should have built-in architecture-to-fault adaptation sub-system. This sub-system should find very quickly the best possible variant of processing architecture with minimal degradation of performance parameters.
3. Implementation of restoration mechanism should require minimum logic resources. Furthermore, restoration control procedures should be presented in form of virtual hardware (configuration files) but not in a form of software or an application specific integrated circuits (ASICs). Software implementation is not effective because restoration period will be dramatically delayed. On the other hand, in case of ASIC implementation of self-restoration procedures many other problems can occur. Firstly, ASIC can be damaged itself by radiation or other factors that are influencing on semiconductor devices and thus, no further restoration of processing system will be possible. Secondly, no

further modification of restoration procedures can be done because of fixed routing and logic configuration of any ASIC. However, if self-restoration procedures are implemented in a form of virtual hardware, even restoration controller can be self-restored and also can be easily modified or repaired remotely.

It is because of this reason this research was focused on solving the above listed of issues and thus the major contributions of this work can be divided on two groups:

a) theoretical contributions and b) implementation and analysis of test results.

The theoretical contributions are as follows:

1. Development of logic replacement technique based on re-addressing of Virtual Hardware Components to the spare slots in the partially reconfigurable FPGA devices (Xilinx Virtex-E, Virtex-2 and Virtex-2Pro families of FPGAs). In the best of our knowledge that is based on the conducted literature research this is a novel technique that can dramatically reduce restoration time up to hundreds of microseconds. This method was presented in [1]
2. Development of method for minimization of performance degradation in a case of restoration of processing functionality with limited logic / routing resources. After analysis of literature sources the novel approach was proposed. This approach assumes including an architectural synthesis sub-system into the self-restoration mechanism to allow run-time selection of new architectural configuration avoiding the damaged circuit. This method is based on special mathematical algorithm, which allows rapid arrangement of design space and selection of new processing architecture with reduced hardware resources. This method was presented in [2]

3. Incorporation of different hardware fault mitigation techniques into one multi-level self-restoration mechanism. In the best of our knowledge this is also novel approach where each level or restoration is associated with respective factors, that can cause hardware faults. Furthermore, because these levels are hierarchically arranged by probability of factors causing hardware faults, restoration can be performed in most optimal way and thus reduce restoration period.

Contribution in implementation and analysis of test results are the follows:

1. Implementation and test of major elements of the proposed self-restoration mechanism (scrubbing and logic replacement) were performed on the base of RTR Multi-Stream Processor AGORA-2 (Adaptive Group Organized Reconfigurable Architecture). It was proven that scrubbing and logic replacing procedures can be performed much faster (in times) than using the existing methods. That is because the advantages of partially reconfigurable FPGA devices were utilized together with proposed in this research re-assembling procedures of Virtual Hardware Components. Results were presented in [2], [3] and [4].

2. Modeling and simulation of major elements of the proposed self-restoration mechanism were performed using ALTERA MAX Plus II and XILINX ISE 6.2i CAD systems. The results were presented in [3] and [4]
3. Modeling of self-restoration with minimum performance degradation was performed step-by-step for real task: Hamming Distance Calculator. It was proven that dramatic minimization of number variants to be estimated for selection of the optimum using

arranged design space presented in a form of Architecture Configurations Graph (ACG). Analysis of the results was utilized for preliminary study for development process of the new generation of on-board computing platform for Canadian satellite RADARSAT-2.

Summarizing all of the above contributions it is possible to say that the proposed novel self-restoration mechanism can dramatically improve lifetime, and reliability of the embedded computing platforms based on reconfigurable logic devices (SRAM based FPGAs). However, because this work was the "first step" in this feasibility study, the implementation component of the work included testing only major elements of this mechanism. Thus, the further research and development in this direction is a must.
In the future work the following issues have to be considered:

1. Research the restoration level's switching control mechanism, which will include development of scheduling and synchronization of processes. This means that for case when self-restoration system has to switch from one level to another (e.g. from scrubbing to logic replacement) some transition steps has to be performed. These steps have to be properly scheduled and synchronized with existing data processing tasks.
2. Restoration with performance degradation also needs more deep development. At this work we did not consider any mapping procedure for replacement the VHC with smaller logic requirements to the damaged CLB-slot. However, there are many questions, which do not have clear answers today. At this stage of work we also did not touch the problem of scheduling loading sequence for restorable VHCs. We also did not develop a complete control program for this level of self-restoration.

3. Built-In-Self-Test (BIST) system has to be incorporated with proposed self-restoration mechanism to create Built-In-Self-Recovery (BISR) system embedded to the Hardware Operating System (HOS), the process control core of RTR parallel computing platform. In capacity of this work we did not consider this incorporation. However, this research and development work is a must for any practical implementation of the proposed self-restoration mechanism.

Finally, we would like to mention once again that this project is the preliminary study and very first step to incorporate Run-Time Reconfigurable Parallel computing platform, based on partially reconfigurable FPGAs with self-restoration mechanism. Without this restoration mechanism a lot of applications of embedded high-performance computing platform are very difficult or even impossible. Thus, this work is very important and results are in big demand for modern aerospace, multimedia, digital signal processing and many other areas of application.

REFERENCES

[1]. L. Kirischian, P. Chun, I. Terterian, and V. Geurkov, "A Re-Configurable Data-Stream and Network Processor for Networked Manufacturing Systems", *in the Proceedings of the 14-th International Conference on Flexible Automation & Intelligent Manufacturing,* pp.1120-1127, Toronto, Canada, July 12-14, 2004.

[2]. L. Kirischian, I. Terterian, P. Chun, and V. Geurkov, "A Re-Configurable Parallel Stream Processor with Self-Assembling and Self-Restorable Micro-Architecture", to appear *in the Proceedings of the International Conference PARELEC-2004,* Dresden, Germany, September 7-10, 2004.

[3]. L. Kirischian, V. Geurkov, I. Terterian, and J. Kleiman, "Self-Restoration as SEU Protection Mechanism for Re-Configurable On-Board Computing Platforms". *Proceedings of the 7-th International Conference on Protection of Materials and Structures from Space Environment,* Toronto, Canada, May 10-13, 2004.

[4]. Pill Woo Chun, Irina Terterian and Lev Kirischian, "Multi-task Parallel Video-stream Processor with Self-Assembling Micro-architecture", *SVAR 2004 (Space Vision and Advanced Robotics)*, MD Robotics (former SPAR Aerospace), Brampton, ON, Canada, May 2004.

[5] List of FPGA-based Computing Machines, http://www.io.com/~guccione/HW_list.html

[6] XC6200 Field Programmable Gate Arrays, XILINX Inc.,1997

[7] P..M. Athanas and H.F. Silverman, "Processor Reconfiguration through Instruction-Set Metamorphosis", *Computer, 26 (3)*, pp. 11-18, March 1993

[8] J.R.Hauser and Wawrzynek, "GARP: A MIPS Processor with a Reconfigurable Coprocessor", *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines,* pp. 16-18, April 1997

[9] R.D. Wittig and P.Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", *Proc. of the IEEE Symposium on FPGAs for Custom Computing Machines,* pp. 126-135, Los Alamitos, California, April 1996

[10] B.Kastrup, "Automatic Hardware Synthesis for a Hybrid Reconfigurable CPU Featuring Philips CPLDs", *Proc. PACT'98 International Conference on Parallel Architectures and Compilation Techniquies, Workshop on Reconfigurable Computing,* pp. 5-10, Paris, France, October 1998.

[11] C.Ebeling, D.C. Cronquist and P. Franklin, "RaPid – Reconfigurable Pipelined Datapath", *in Proc. of Field Programmable Logic,* pp. 126-135, Springer-Verlag, Heidelberg, 1996

[12] NAPA1000 Adaptive Processor. *National Semiconductor,* 1998

[13] J.M. Arnold, D.A. Buell and E.G. Davis, "Splash-2", *in Proc.SPAA1992, 4-th Annual Symposium on Parallel Algorithms and Architectures,* pp. 316-324, San Diego, CA, June 1992

[14] F. Mayer-Lindenberg, "Crossbar Design for a Super FPGA Architecture", *in Proc. PACT'98 International Conf. on Parallel Architectures and Compilation Techniquies, Workshop on Reconfigurable Computing,* pp. 29-33, Paris, France, October 1998.

[15] M.D. May, P.W. Thompson and P.H. Welch, "Networks, routers & transporters: Function, performance and application", *IOS Press, ISBN 90 5199 129 0,* 1993

[16] M. Tudruj, "Connection by Communication – Paradigm for Dynamically Reconfigurable Multi-Processor Systems", *in Proc. PARELEC 2000 International Conference on Parallel Computing in Electrical Engineering (IEEE #PR00759),* pp.74-78, Trois-Rivieres, Quebec, Canada, August 2000.

[17] H. Singh, M-H. Lee, G. Lu, F.J. Kurdahi and N. Bagherzadeh, "MorphoSys: A Reconfigurable Architecture for Multimedia Applications", *in Proc. PACT'98 International Conf. on Parallel Architectures and Compilation Techniquies, Workshop on Reconfigurable Computing,* pp. 34-39, Paris, France, October 1998.

[18] R.D. Hudson, D.I. Lehn and P.M. Athanas, "A Run-Time Reconfigurable Engine for Image Interpolation", *IEEE Symposium on FPGAs for Custom Computing Machines, FCCM 98,* April 1998

[19] M. Kaul, R. Vemuri, S. Govindarajan and I. Ouaiss, "An Automated Temporal Partitioning Tool for a class of DSP applications", *in Proc.PACT'98 International Conf. on Parallel Architectures and Compilation Techniquies, Workshop on Reconfigurable Computing,* pp. 22-27, Paris, France, October 1998.

[20]. Claus Grupen, "Particle Detectors", *Cambridge monographs on particle physics,* Vol.5, 1985.

[21]. A. Draganic, "Radiation and Radioactivity on Earth and Beyond", Second edition, 1988.

[22]. V.I. Belyi, L.L Vasilyeva, A.S. Ginovker, V.A. Gritsenko, S.M. Repinsky, S.P.Sinitsa, T.P. Smirnova, F.L. Edelman, "Silicon Nitride in Electronics", *Materials Science Monographs*, Vol. 34, pp.110-115, Elsevier, 1988.

[23]. "Radiation Test Results on the Vertex FPGA for Space Based Reconfigurable Computing", XILINX Inc. at http://www.xilinx.com/products/hirel_qml.htm

[24]. Mitra S., Shirvani P., McCluskey E., "Fault Location in FPGA-Based Reconfigurable Systems.", *IEEE International High Level Design Validation and Test Workshop,* Monterey, CA, USA, 1998

[25]. Stroud C., Chen P., Konala S., Abramovici M., "Evaluation of FPGA resources for Built-in Self Test of Programmable Logic Blocks", *Proc. of.4th ACM/SIGDA Int. Symp. on FPGAs*, pp. 107-113, February 1996, Monterey, CA, USA

[26]. Stroud C., Wijesuraya S., Hamilton C, Abramovici M., "Built-in Self Test of FPGA" *Interconnect. Int. Test Conf.*, pp. 404-411, Oct. 1998, Washington, DC.

[27]. Zhao L., Walker D., Lombardi F., "Detection of bridging faults in Logic Resources of Configurable FPGAs Using I_{DDQ}," *Int. Test Conf.*, pp. 1037-1046, Oct. 1998, Washington, DC.

[28]. Renovell M., Figueras J., Zorian Y., "Test of RAM-Based FPGA: Methodology and Application to the Interconnect," *15th IEEE VLSI Test Symp.*, pp. 230-237, May 1997, Monterey, CA, USA.

[29]. Lombardi F., Ashen D., Chen X., Huang W., "Diagnosing Programmable Interconnect Systems for FPGAs", *FPGA '96,* pp. 100-106, February 1996, Monterey, CA,USA.

[30]. Michinishi H., Yokohira T., Okamoto T., Inoue T., Fujiwara H., "A test Methodology for Interconnect Structures of LUT-Based FPGAs", *IEEE 5th Asian Test Symp.*, pp. 68-74, Hsinchu, Taiwan, Nov. 1996

[31]. Renovell M., Portal J., Figueras J., Zorian Y., " Minimizing the number of Test Configurations for Different FPGA Families", *IEEE 8th Asian Test Symp.*, Nov. 1999, Shanghai, China.

[32]. Renovell M., Portal J., Figueras J., Zorian Y., "SRAM-Based FPGA: Testing the LUT/RAM Modules" *IEEE Int. Test Conf.*, pp. 1102-1111, Oct. 1998, Washington, DC.

[33]. Michinishi H., Yokohira T., Okamoto T., Inoue T., Fujiwara H., "Testing for the Programming Circuits of LUT-Based FPGAs", *IEEE 6th Asian Test Symp.*, pp. 242-247, Nov. 1997, Akita, Japan.

[34]. Huang W., Meyer F., Park N., Lombardi F., "Testing Memory Modules in SRAM-Based Configurable FPGAs", *IEEE Int. Workshop on Memory Technology, Design and Test,* Aug. 1997, San-Jose, CA, USA.

[35]. Renovell M., Portal J., Faure P., Figueras J., Zorian Y., "Analyzing the Test Generation Problem for an Application-Oriented Test of FPGAs" *IEEE International Test Symp.*pp.110 – 115, Oct.1998, Washington, DC.

[36]. Jordan C., Marnane W.," Incoming Inspection of FPGAs", *IEEE Euro. Test Conf.*, pp. 371-377, Rotterdam, Netherlands, April 1993.

[37]. Liu T., Lombardi F., Salinas J., "Diagnosis of Interconnects and FPICs Using Structured Walking-1 Approach", *IEEE VLSI Test Symp.*, pp. 256-261., Princeton, NJ, USA, April 1995.

[38]. Stroud C., Chen P., Konala S., Abramovici M.," Built-in Self Test of Logic Blocks in FPGAs ", *IEEE VLSI Test Symp.*, , pp. 387-392, April 1996, Princeton, NJ, USA

[39]. Renovell M., Portal J., Figueras J., Zorian Y., "Testing the Interconnect of RAM-Based FPGAs" *IEEE Design and Test of Computers*, Vol. 15, N1, pp. 45-50, Jan. 1998.

[40]. Huang W., Chen X., Lombardi F., "On the Diagnosis of Programmable Interconnect Systems: Theory and Application", *IEEE VLSI Test Symp.*, pp. 204-209, April 1996, Princeton, NJ, USA.

[41]. Stroud C., Lee E., Abramovici M.," BIST Beased Diagnostics of FPGA Logic Blocks", *Proc. Of Int. Test Conf.*, pp. 539-547, Washington, DC, USA, Nov.1997.

[42]. Wang S., et al., "Test and Diagnosis of Faulty Logic Blocks in FPGAs", *Proc. ICCAD*, Vol.15, N1, pp. 722-727, San-Jose, CA, USA, Nov.1997..

[43]. Inoue T., Miyazaki S., Fujiwara H., "Universal Fault Diagnosis for Look-Up Table FPGAs," *IEEE Design and Test of Computers*, Vol. 15, N1, pp. 39-44, Jan. 1998.

[44]. Friedman A., "Easily Testable Iterative Systems", *IEEE Trans. Comp.*, Vol. C22, No. 12, pp. 1061-1064, Dec. 1973.

[45]. Das D., Touba N., "A Low Cost Approach for Detecting, Locating, and Avoiding Interconnect Faults in FPGA-Based Reconfigurable Systems" *Proc. IEEE Symp. On FPGA for Custom Computing Machines*, pp.140-147, Napa Valley, CA, USA, April 1997.

[46]. Culberstone W., Amerson R., Carter R., Kuekes P., Snider G.," Defect Tolerance on the Teramac Custom Computer," *Proc. IEEE Symp. On FPGAs for Custom Computing Machines*, pp. 140-147, Napa Valley, April 1997..

[47]. Hanchek F., Dutt S., "Methodologies for Tolerating Cell and Interconnect Faults in FPGAs", *IEEE Trans. on Computers*, Vol5. pp. 15-33, 1998.

[48]. Kumar V., Dahbura F., Fischer F., Juola P., "An Approach for the Yield Enhancement of Programmable Gate Arrays", *Proc. IEEE Int. Conf. On CAD*, pp. 226-229, Santa Clara, CA, USA, November 1989.

[49]. Hatori F., Sakurai T., Sawada K., Takahashi M., Ichida M., Yoshii I., Kawahara Y., Hibi T., Sacki Y., Muraga H., Kanzaki K., "Introducing Redundancy in FPGAs," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 7.1.1-7.1.4, San- Diego, CA, USA, April 1993.

[50]. Durand S., Piguet C., "FPGAs with Self-Repair Capabilities", *Proc ACM Int. Workshop on FPGAs*, pp. 1-6, Berkley, CA,USA , March 1994.

[51]. Narasimhan J., Nakajima K., Rim C., Dahbura A., "Yield Enhancement of Programmable ASIC Arrays by Reconfigurations of Circuit Placements", *IEEE Trans. On CAD*, Vol. 13, No. 8, pp. 976-986, 1994.

[52]. Narasimhan J., Nakajima K., Rim C., Dahbura A., "Yield Enhancement of Wafer Scale Intehrated," *Proc. IEEE Conf. On Wafer Scale Integration*, pp. 178-184, San-Francisco, January 1991.

[53]. Howard N., Tyrrell A., Allinson N., "The Yield Enhancement of Field Programmable Gate Arrays", *IEEE Trans. On VLSI Syst.*, Vol. 2, pp. 115-123, 1994.

[54]. Kelly J., Ivey P., "A Novel Approach to Defect Tolerant Design for SRAM Based FPGAs," *Proc. ACM Int. Workshop on FPGAs*, pp. 7-11, Berkley, CA, USA, March 1994.

[55]. Cuddapah R., Corba M.," Reconfigurable Logic for Fault Tolerance", *Springer Verlag, 1995*

[56]. Hanchek F., Dutt S., "REMOD: A New Methodologies for Designing Fault Tolerant Arithmetic Circuits", *IEEE Trans. on VLSI Systems*, Vol. 5, pp. 34-56, 1997.

[57]. Mahapatra N., Dutt S., " Efficent Network Flow Based Technique for Dynamic Fault Reconfiguration in FPGAs", *Proc. Fault Tolerant Computing Symp.*, pp. 122-129, Madison, Wisconsin, USA, June 1999.

[58]. Emmert J., Bhatia D.," Reconfiguring FPGA Mapped Designs with Applications to Fault Tolerance and Reconfigurable Computing," *Lecture notes on Comp. Sci.*, Vol. 1304, pp. 141-150, 1997.

[59]. Emmert J., Bhatia D.," Incremental Routing in FPGAs", *Proc. IEEE Int. ASIC Conf.*, pp. 302-305, Rochester, NY, USA, Sept. 1998.

[60]. Lach W., Manione-Smith W., Potkonjak M., "Efficiently Supporting Fault Tolerance in FPGAs," *Proc. ACM Int. Symp. On FPGAs*, Monterey, CA, USA , February 1998.

[61]. Emmert,J., Stroud, C., Skaggs, B., Abramovici M., " Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration, Field- Programmable Custom Computing Machines", *IEEE Intl.Symposium on FPGA,* pp.165-174, Napa Valley, CA,USA, April 2000.

[62]. Scott Rixner, "Stream Processor Architecture", *Kluwer Academic Publishers,* 2002.

[63]. XAPP151 v1.6: "Virtex Series Configuration Architecture User Guide", *Xilinx Inc.*, March 2003

[64]. R. Hartenstein, "A Decade of Reconfigurable Computing: a Visionary Retrospective", *In Design, Automation and Test in Europe,* pp. 642-649, 2001

[65]. XAPP216 v1.0: "Correcting Single-Event Upsets Through Virtex Partial Configuration", *Xilinx Inc.,* June 1, 2000

[66]. Lev Kirischian, Pil Woo Chun, Irina Terterian and Vadim Geurkov, "A Re-Configurable Data-Stream and Network Processor for Networked Manufacturing Systems",- *in Proceedings of 14-th International Conference on Flexible Automation & Intelligent Manufacturing (FAIM2004),* vol. 2, pp.1120-1127, Toronto, Canada, July 12-14, 2004

[67]. Lev Kirischian, "Optimization of Parallel Task Execution on the Adaptive Re-configurable Group Organized Computing System", *in Proc. PARELEC 2000 International Conference on Parallel Computing in Electrical Engineering (IEEE #PR00759),* pp. 100-105, Trois-Rivieres, Quebec, Canada, August 2000.

[68]. XILINX Virtex II Platform FPGA Handbook VG002 v.1.0, December 6, 2000

[69]. Lev Kirischian, Lucas Szajek and Fayes Chayab, "Architecture-to-Task Optimization System (ATOS) for Parallel Multi-Mode Data-Flow Architectures on a Base of a Partially Re-configurable Computing Platform", *in Proc. PARELEC 2002 International Conference on Parallel Computing in Electrical Engineering,* Warsaw, September 2002.

[70]. XAPP 216 v.1.0"Correcting SEU through Virtex Partial Configuration", *Xilinx Inc.,* June 1, 2000.

[71] E. Fuller, M. Caffey, A. Salazar, C. Carmichael, J Fabula, "Radiation Testing Update,

SEU Mitigation and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing", *in Proc. of MAPLD International Conference*, Sept. 2000

[72]. Carl Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs", *Xilinx Application Note XAPP197 (v 1.0),* Nov. 2001

[73]. M. Abramovici, M. Breuer and A. Friedman, "Digital Systems Testing and Testable Design ", *Wiley-IEEE Computer Society Press,* 1994.

[74] . Lev Kirischian, Fayez Chayab and Kristopher Bates, "Architecture Organization of Partially Re-configurable Computing Platform for Multi-task and Multi-mode Applications", *in Proc. WASP-2002 International Symposium of Micro-architecture (MICRO-35),* pp. 128-137, Istanbul, Turkey, November 2002.

List of Acronyms

ACG – Architecture Configuration Graph
ASIC – Application Specific Integrated Circuit
ASPC – Application Specific Processor Core
ASVP – Application Specific Virtual Processor
BIST – Built-in-Self-Test
BUT – Block Under Test
CLB – Configurable Logic Block
COTS – Commercial-Off-the-Shelf
CPLD – Complex Programmable Logic Device
CPU – Central Processor Unit
DFG – Data- Flow Graph
DPGA – Dynamically Programmable Gate Array
EPROM – Electrically Programmable Read Only Memory
FFT – Fast Fourier Transform
FPGA – Field Programmable Gate Array
GPR – General Purpose Routing
GRL – Global Routing Lines
HDC – Hamming Distance Calculation
HOS – Hardware Operating System
IETID – Interface Element Type Identifier
IOB – Input/Output Block
LET - Linear Energy Transfer
LUT – Look Up Table
MO – Macro Operation
MOSFET – Metal Oxide Semiconductor Field Effect transistor
MNOS – Metal-Nitride-Oxide-Semiconductor
PETID – Processing Element Type Identifier
RCS – Reconfigurable Computing System
RFM – Reconfigurable Functional Module
RIM – Reconfigurable Interface Modules
RISC – Reduced Instruction Set Computer
RPSP – reconfigurable Parallel Sream Processor
RTR - Run-Time Reconfigured
SEU – Single Event Upset
SRAM – Static Random Access Memory
STAR – Self-Testing Area
TBG – Test Pattern Generator
VCB – Virtual Communication Bus
VHC – Virtual Hardware Component
VHDL – Very High Speed Logic Hardware Language
VHO- Virtual Hardware Object