Re-Configurable Parallel Stream Processor with Self-Assembling and Self-Restorable Micro-Architecture

Lev Kirischian
Ryerson University

Irina Terterian
Ryerson University

Pil Woo Chun
Ryerson University

Vadim Geurkov
Ryerson University

Please Cite:
Kirischian, L., Terterian, I., Chun, P. W., & Geurkov, V. (2004). Re-configurable parallel stream processor with self-assembling and self-restorable micro-architecture. *Proceedings of the International Conference on Parallel Computing in Electrical Engineering 2004*, 165–170.
doi:10.1109/PCEE.2004.60

Re-Configurable Parallel Stream Processor with Self-Assembling and Self-Restorable Micro-architecture

Lev Kirischian, *Ryerson University* *Toronto, CANADA* lkirisch@ee.ryerson.ca
Irina Terterian, *Ryerson University* *Toronto, CANADA* iterteri@ee.ryerson.ca
Pil Woo Chun, *Ryerson University* *Toronto, CANADA* pchun@ee.ryerson.ca
Vadim Geurkov *Ryerson University* *Toronto, CANADA* vgeurkov@ee.ryerson.ca

Abstract

In this paper we present a concept of the self-assembling micro-architectures of Application Specific Virtual Processors for data-stream processing. The procedure for micro-architecture assembling is developed for Xilinx "Virtex" FPGA devices. It is shown that proposed approach allows a minimization of system resources for multi-task data-stream workload and gives ability for self-restoration of processing micro-architectures when hardware fault occurs. This Paper presents a description of system level architecture of run-time re-configurable multi-stream parallel processor for video applications and results gained on the prototype.

1. Introduction

In many industrial data-stream processing systems such as: video-surveillance, video-recognition, digital video-broadcasting and digital communication systems it is required to process multiple streams of data with a very high rate (Gb/s). The usual approach is an implementation of application specific integration circuits (ASICs), or a Field Programmable Gate Array (FPGA) devices were application specific processor core (ASPC) is loaded. This approach seems more cost-effective when ASIC or ASPC architecture is designed for one specific application (task) [1]. However, there are several disadvantages associated with this approach. One of the main problems associated with ASICs is the lack of hardware flexibility and the inability for a modification of its micro-architecture. Thus, if any change in the processing algorithm is required or hardware bug is found, the ASIC micro-architecture cannot be modified. 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. 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 their processing micro-architecture is designed for all tasks, and all task modes. Thus, this micro-architecture that reflects all possible processing algorithms should be stored in the ASIC or FPGA in the form of real hardware. Taking in account that: a) not all tasks are initiated at the same time in the multi-task workload and b) only one mode from the selected many can be requested for each task, we can predict a greater waste of logic resources and power to feed big portion of non-active hardware. To solve this problem the concept of run-time re-configurable (RTR) systems can be implemented. RTR approach assumes a utilization of FPGA devices with partially re-configurable micro-architecture [2]. 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 [3] each processing core has to be developed and pre-compiled using CAD system that is associated with utilized FPGA family (e.g. ISE Foundation for Xilinx FPGA devices). Instead, our approach assumes assembling a complete Application Specific Virtual Processor (ASVP) on-chip on the basis of uniformed "LEGO" blocks: sub-cores, which we call Virtual Hardware Components (VHC). Furthermore, the proposed approach assumes that the ASVP assembling procedure is fully automated because it should be performed during hundreds of microseconds without any influence of the 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 active in the FPGA. In this case any damaged Virtual Hardware Component can be restored by scrubbing procedure [4] or re-loaded to another available slot of the FPGA. Thus, our goal is to create 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 stream executions and data transmission processes.

The rest of the paper is organized as follows: Section 2 describes architectural organization of Virtual Hardware Components (VHC) in Xilinx Virtex-type FPGA devices. In Section 3 the ASVP assembling procedure is discussed, Section 4 gives brief overview of system architecture of the proposed Re-configurable Parallel

Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'04)
0-7695-2080-4/04 $ 20.00 IEEE

Stream Processor (RPSP). Following Section 5 is where we will discuss self-restoration aspect of a proposed approach. As well as, Section 6 presents analysis of gained results. Lastly, the summary is presented in Section 7 to finalize this paper's purpose.

2. 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) [1]. In consideration of 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 [2]. The structure of "Virtex" FPGA consists of (Figure 1): Arrays of CLBs (*Configurable Logic Blocks*); Arrays of IOBs (*Input Output Blocks*), SRAM memory blocks (*Block RAM*); Clock logic resources (DLLs, etc.) and routing resources (Global and local routing).

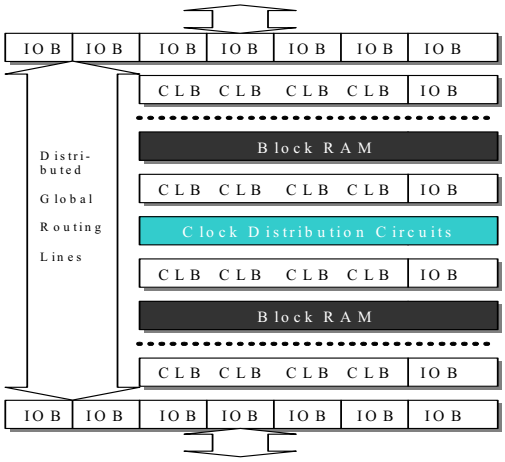

Figure 1. Structure of "Virtex" FPGA

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 in-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. Also, each small configuration data file can represent a Virtual Hardware Component (VHC) to be downloading into addressable FPGA slots (CLB-columns). The micro-architecture of VHC consists of two major components visualized in (Figure 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 it was shown in "Virtex" FPGA device data sheet [5] 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.

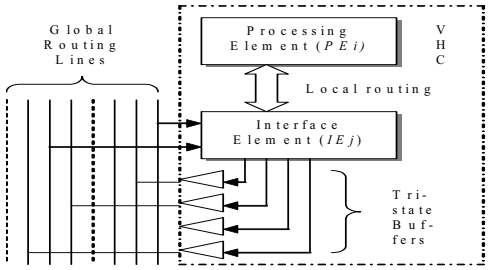

Figure 2. Micro-architecture of a VHC

3. Application Specific Virtual Processor assembling procedure

The Architectural synthesis of the application specific processor normally is based on application (task) algorithm analysis. That creation of the Data Flow Graph (DFG) and architectural is the optimization based on DFG and data-path synthesis [6]. 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. Although instead, we propose 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. This is similar to "Port Map" procedures in Hardware Descriptive Language but in an 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 shown in Figure 3. To simplify the case, in our example we will not consider scheduling and binding procedures together [6] and just assume that the DFG should be mapped in the hardware "as is". In reality, we

use special Architecture-to-Task Optimization System (ATOS) for finding an optimal task DFG mapping [7].

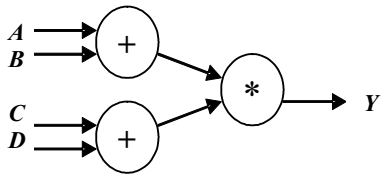

Figure 3. Data Flow Graph of stream processing task

To assemble any micro-architecture by VHCs – we have to have the library of available VHCs. Because Virtual Hardware Components (VHSc) are pre-compiled cores (configuration data files for certain FPGA device) and each VHC has to be located in VHC-memory at a certain location. The VHC-identifier consists of two parts: One being the Processing Element Type Identifier (PETID) and the other Interface Element Type Identifier (IETID). Based on this ID- information 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 assigned available FPGA slot for the VHC and loading the VHC to the FPGA has to be done by special hardware unit which we call Hardware Operating System (HOS). In our example, (Figure 3) we require to have 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 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. Similarly, Interface Element with IETID=01 will have two groups of 8-bit inputs connected to GRL #24 - #39 and on output of 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, VHSs, 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 as follow: a) External interface: Input / Output blocks (IOBs), b) Internal links routing and c) Clock routing scheme. All of these components are usually task-specific and should be combined to the fixed part of ASVP architecture. Thus, for task *Ti* we will have a fixed part of ASVP[*i*] architecture – *Afix[i]*. Now we can start to consider the complete process of ASVP creation in a partially re-configurable FPGA. This process consists of the following steps: 1) hardware Operating System (HOS) which receives a request for task activation and loads a fixed part of ASVP optimized for this task, 2) HOS retrieves from task code VHC-identifiers to be loaded into the FPGA, 3) using Core 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 (CLB-column) and 6) When all components are loaded, HOS initiates data processing by sending start-signal to all VHCs in ASVP.

For our example ASVP architecture is shown in Figure 4:

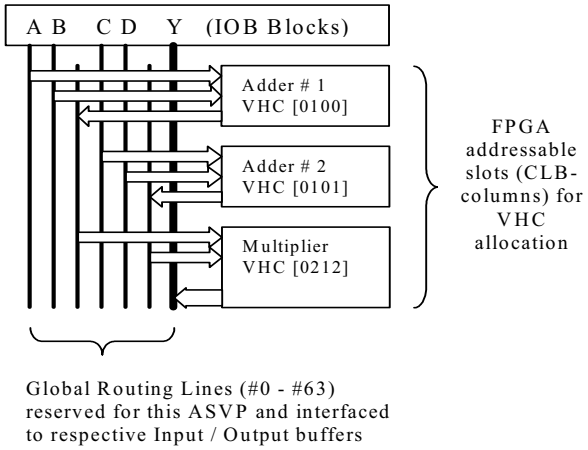

Figure 4. ASVP architecture assembled in the FPGA from pre-compiled VHCs reflecting task DFG (Figure 3)

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)* In our case only Multiplier should be replaced by Divider with the same Interface Element. Therefore, 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 280 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. We would like to mention that the process of VHC replacement in existing ASVP architecture looks like an automated plug-in operation of virtual component in Virtual Bus. Thus, the mode switching in a proposed system can be done not by switching between existing hardware modules (Multiplier and Divider) but by re-configuring same logic resources (CLB) in very short period (hundreds of microseconds). This in result will allow for a dramatic minimization of

hardware resources and associated cost, dimensions, weight and power consumption. Comparison with regular FPGA-based systems will be discussed in Section 6.

4. Re-configurable Parallel Stream Processor Architecture organization

Re-configurable Parallel Stream Processor (RPSP) architecture, adaptable for the multi-task and multi-mode workload, implements the micro-architecture assembling mechanism described in Section 3. Architecture of RPSP includes the following components shown in Figure 5:

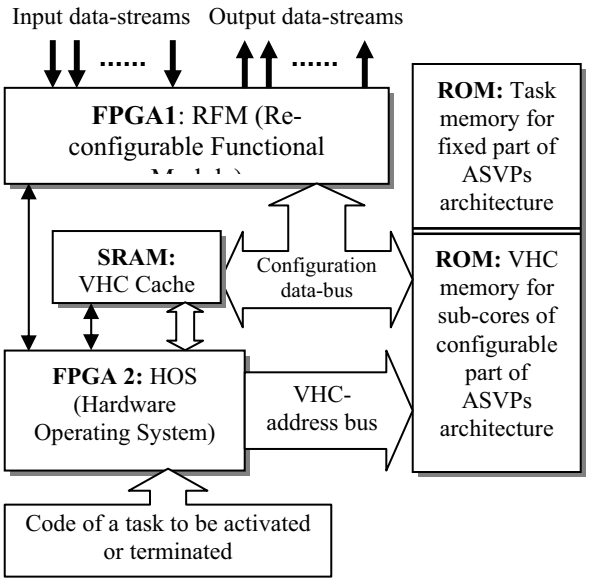

Figure 5. Architecture of Re-configurable Parallel Stream Processor (RPSP) for multi-task & multi-mode workload.

a) Re-configurable Functional Module (RFM) is based on partially re-configurable FPGA and 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 VHCs,
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 workload.
d) VHC cash 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 the FPGA, performs the following major functions: i) Task initiation and termination; ii) Mode switching by loading a 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 a great minimization within 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 a data processing structure 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 of data files for an entire Application Specific Processor is assembled from "LEGO"-type component cores (VHC in our terminology). It is possible for the reason of utilization of same component cores in a different ASVP. Thus, the architecture of RPSP contains hierarchy of memory units reflecting on previously explained 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 of 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). The task code being the ASVP code contains the following: i) code of a fixed part of ASVP architecture and ii) list of all VHCs associated with a requested version of ASVP. For instance, in the case of an example discussed earlier in Section 3, that application code of ASVP will look as shown in Figure 6.

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

Figure 6. Code of a task = Code of associated ASVP

Secondly, HOS converts VHC# to the start-address of VHC configuration file in VHC Memory. The conversion mechanism is based on a table which is located in the VHC-Cache and appearances as shown in Figure 7.

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

Figure 7. VHC# to VHC-core address conversion table

Afterwards, HOS retrieves an address of the first VHC and sends it to VHC Memory, then 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, and HOS can initiate the 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.

5. Self-restoration mechanism for ASVP

There are a few main reasons for hardware faults to occur in SRAM-based FPGA devices: i) Physical defects in wafer; ii) Hidden manufacturing defects; iii) Radiation effects. In our project, we considered faults occurring however, in small amounts of CLBs or in configuration SRAM of the FPGA. Mostly this type of faults occurs as a result of radiation effects (SEU – single event upset and SEL– Single Event Lutch-up) usual for space applications [4] or applications for radiation intensive environments (nuclear power stations, etc). Instead of usual approach based on protection of materials from radiation intensive environment, we proposed a self-restoration mechnism [8] based on re-assembling of damaged processing structures. This mechanism utilize so-called scrubbing procedure [4] but allows functional re-storation much faster. In this paper we will give only a brief description of the proposed method. Thus, to restore ASVP we propose a two-level procedure. The First level of self-restoration is based on VHC-scrubbing: re-loading of same VHC configuration data-file to the same address area of configuration SRAM. Which is of the FPGA where this VHC was located. This can correct wrong state of Flip-Flops in the configuration SRAM and thus, correct damaged VHC-structure. This prosedure can be performed using the same ASVP assembling mechanism discussed in Sections 3 and 4.

The necessity of second level of self-restoration appears when other reasons than just SEU make one of logic gates damaged. This case can be considered when VHC scrubbing does not help. In this case the damaged gate should be avoided. In our implementation based on Xilinx Virtex family of FPGA devices the smallest addressable unit of FPGA is a frame. Thus, if any gate appears to be damaged by any reason except SEU, we have no other choise than to avoid a complete frame where this gate is located. However, because any VHC is based on CLB-column organization we developed a mechanism for re-location of VHC from the damaged slot into a spare slot. In this approach extra hardware costs are involved to increase reliability of the system. However, it is possible that after some period of time all reserved slots will be used. In the case of a hardware fault the third level of restoration with performance of degradation will be initiated. In this case it is possible to load the existing CLB-columns with another variant of VHC with 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. Let us consider the process of replacement of damaged slot by reserved ones. This process consists of the following steps:

a) Pausing the data-stream processing pipeline.
b) Disabling damaged Virtual Hardware Component by loading a "dummy" VHC (with code [0000]) to tri-state all outputs and to prevent data contention on the bus.
c) Selecting the available CLB-column(s) for loading configuration data files of VHC to be restored.
d) Composing the configuration bit-stream by inserting the selected frame address and associated information into the VHC- configuration data file.
e) Loading the VHC configuration file into the selected column(s) while continuing data-stream processing.

In case if there is a lack of spare CLB-columns, the same procedure can be performed but with different variant of VHC. For example, a 16-bit VHC architecture can be replaced by 8-bit VHC, which can restore functionality of ASVP, but with reduced performance. Experiments in this regard were performed on the prototype of RPSP with restoration of different ASVPs for executing different video-stream processing algorithms.

6. RPSP Implementation and analysis of experimental results

Performance results were gained on the first prototype of RPSP based on Xilinx XCV-400E (RFM) and XCV-50E (HOS) devices. Aggregate bandwidth of I/O interface was equal to 7.2 Gb/s (LVDS) and 8.5 Gb/s (LVTTL). Cache volume = 2MB. Configuration bus bandwidth = 528 Mb/s.

RPSP was interfaced with CMOS digital camera (388 x 280 pixels) as high speed video-input (86.9 Mb/s) and 3 SVGA CRT monitors as video-outputs (3 x 1.258 Gb/s). To evaluate the performance characteristics for multi-task and multi-mode workload, three ASVP were developed. All three ASVP performed different types of video-stream processing algorithms. It was then proved that all three independent tasks could run on the RPSP simultaneously. Initial (fixed part) architecture loading period was equal to 6.6 ms. Mode switching time was as low as 280 us for one CLB-column VHC replacement. Mode switching time was 420 us for 2 column-wide VHC. Furthermore, mode switching time for 3 CLB-column wide VHC was 560 us. Based on these results the comparison analysis was made with a common approach. That is when a complete FPGA device has to be re-configured to switch from one task to another or from one task mode to another. The acceleration of mode switching - Ams was calculated for Xilinx Virtex FPGAs by the following formula: $Ams = Tfpga / Tasvp,$ where $Tfpga$ is re-configuration time for complete FPGA and $Tasvp$ is re-configuration of 2 CLB-column wide VHC in a ASVP.

We considered that in a bit-stream loading time in SelectMAP programming mode to be at a 50MB/s loading rate. Results of this comparison for Xilinx Virtex II family are presented in Table 1.

Table 1. Acceleration of task / mode switching

Device XC2V	250	500	1000	4000	8000
Ams: 1 CLB-column VHC	13.9	22.9	33.6	127.1	234
Ams: 2 CLB-column VHC	9.3	15.2	22.4	87.8	156
Ams: 3 CLB-column VHC	6.9	11.4	16.8	63.6	117

Thus, the proposed approach gives very high acceleration in task or task mode switching, which increases when FPGA with more logic gates is used and decreases when VHC with more CLB-columns has to be replaced.

We also estimated minimization of hardware resources (in times) comparing RPSP approach and common FPGA-based systems. Comparison was conducted for the workload, which consists of *N* different tasks. Each task can work in *M* different modes. For simplification of this analysis we assumed that each ASVP for each task in the workload requires 10 CLB-columns and only 4 of them can be modified in mode switching. We also assumed that at any period of time only 50% of possible tasks can work simultaneously. Obviously, each task could work only in one of possible modes at a time. Results of this comparison are presented in Table 2.

Table 2. Minimization of hardware resources

N \ *M*	2	4	8	16
4	2.8	4.4	7.6	14
8	5.6	8.8	15.2	28
16	11.2	17.6	30.4	56

Digits in the cells of this table shows a minimization (in times) of hardware resources (number of logic gates) if RPSP architecture is used. It is compared with the usual approach when one multi-stream processing core is implemented in the same family of FPGA devices. This table illustrates that for the workload that consists of small number of tasks and task modes, minimization of hardware resources is not so high. When number of different tasks and modes increases in a workload, respectively it increases the effectiveness of proposed RPSP approach. This hardware resources minimization will also decrease cost, power consumption and power dissipation as well as mass and dimensions of the system. Not only that, but it will also increase the reliability and radiation tolerance, because the smaller area of logic gates has less probability for radiation effects.

7. Summary

The proposed concept of RPSP: Re-configurable Parallel Stream Processor can be a very effective solution to the multi-task and multi-mode data-stream processing workload for many real-time embedded computing platforms. It provides architectural adaptability and ASIC comparable system performance which keeps most of the hardware in "virtual" form in the non-volatile memory rather than in a form of acting hardware. As was investigated on the prototype of RPSP running video-processing tasks, the proposed approach allows large reduction of hardware resources and power consumption. Switching from mode to mode can be done much faster than in regular FPGA-based systems without interruption of other tasks running in parallel on the RPSP. Lastly, the ability of self-restoration of stream-processing pipelines was also investigated and tested on the RPSP prototype.

References

[1] Scott Rixner, "Stream Processor Architecture", *Kluwer Academic Publishers,* 2002, 120 p.

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

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

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

[5] XILINX. Virtex II Platform FPGA Handbook, UG002 (v1.0) December 6, 2000.

[6] Giovani De Micheli, "Synthesis and Optimization of Digital Circuits", *McGraw-Hill, Inc*. 1994, 580 p.

[7] L. Kirischian, L. Szajek, F. 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.

[8] L. Kirischian, V. Geurkov, I. Terterian and J. Kleiman, "Self-Restoration as SEU Protection Mechanism for Re-configurable On-board Computing Platforms", *to appear in Proc. of 7-th International Conference on Protection of Materials and Structures from Space Environment – ICPMSE-7,* May10-13, 2004, Toronto, Canada