1-1-2005

FPGA based computing platform with temporal partitioning mechanism

Valeri Kirischian
Ryerson University

FPGA BASED COMPUTING PLATFORM WITH TEMPORAL PARTITIONING MECHANISM

By

Valeri Kirischian
B.Sc. University of Toronto,
Toronto, 2004

A Thesis
Presented to Ryerson University
In partial fulfillment of the requirement
Of the degree of Master of Applied Science
In the program of Electrical and Computer Engineering

Toronto, Ontario, Canada, 2005

Declaration of Authorship

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

Name	Signature	Address	Date

FPGA-based Computing Platform with Temporal Partitioning Mechanism

Valeri Kirischian,
Master of Applied Science, 2005,
Electrical and Computer Engineering,
Ryerson University

ABSTRACT

In the presented work the FPGA based run-time reconfigurable platform with temporal partitioning of hardware resources is proposed. This platform is based on the Field Programmable Gate Array (FPGA) device that can be reconfigured "on-fly" to provide the optimal adaptation of a processing architecture to the algorithm and data structure by utilization of developed mechanisms of temporal partitioning of computational / logic resources. It was shown that the proposed approach allows reaching very high cost-effectiveness of the computing platform oriented on processing of framed data-streams. On the other hand, the hardware programming and compilation processes could be simplified by utilization of library of precompiled Virtual Hardware Components stored in the on-board FLASH memory. Paper presents theoretical proof of the proposed approach by analytical comparison of the performance that could be reached on the conventional processors and FPGA platform with Temporal Partitioning Mechanism (TPM) of hardware resources. The implementation of the proposed TPM on the basis of Xilinx Spartan-3 and Xilinx Virtex II FPGA devices is described. Experimental results gained on the prototype of the FPGA based platform with TPM are discussed and analyzed.

Keywords: Reconfigurable computing, Data-stream processing, FPGA, Run-Time Reconfiguration, Temporal partitioning.

Acknowledgements

The author would like to thank his supervising professor, Dr. Vadim Geurkov, for providing his guidance, knowledge and support as well as the Department of Electrical and Computer Engineering for providing facilities and resources needed for the research.

The author would also like to thank the members of the review committee for their participation.

The author also would like to express his gratitude for Mr. Jim Koch and members of the Embedded Reconfigurable Systems Lab (ERSL): Mr. Peter (Pil Woo) Chun and Mr. Sergei Zhelnakov for their technical assistance during the implementation stage of the project.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ASIC - Application Specific Integrated Circuit
BRAM – Block Random Access Memory
CAD – Computer Aided Design
CLB - Configurable Logic Block
CPLD – Complex Programmable Logic Device
CPU - Central Processing Unit
CISC – Complex Instruction Set Computer
DSP – Digital Signal Processor
EEPROM - Electrically Erasable Programmable Read Only Memory
ERSL – Embedded Reconfigurable Systems Lab (at Ryerson University)
FPGA – Field Programmable Gate Array
FPSLIC – Field Programmable System Level Integrated Circuit
GUI – Graphic User Interface
HDL – Hardware Description Language
ICAP – Internal Configuration Access Port
IOB – Input /Output Block
ILP – Instruction Level Parallelism
JTAG – Joint Test Action Group (Test Access Port)
MIPS – Million Instructions per Second
OTS – Off-The-Shelf
PCB – Printed Circuit Board
PCI – Peripheral Component Interconnect
PIC - Peripheral Interface Controller
PLCC - Plastic Leaded Chip Carrier
RAM – Random Access Memory
RISC – Reduced Instruction Set Computer
RTR – Run-Time Reconfigured
RTHOS - Real-Time Hardware Operating System
SPI – Serial Peripheral Interface
SRAM – Static Random Access Memory
TPM – Temporal Partitioning Mechanism
UART – Universal Asynchronous Receiver – Transmitter
USART – Universal Serial Asynchronous Receiver – Transmitter
USB – Universal Serial Bus
VHC – Virtual Hardware Component
VHDL – Vhsic(Very high speed integrated circuits) Hardware Description Language
VLIW – Very Long Instruction Word

CHAPTER 1

INTRODUCTION

1.1 Motivation

In recent years demand for high speed processing systems that can perform manipulations with large volumes of data increased tremendously and several developments have been made in different technological fields. Most demanding fields that require such processing are video and image processing applications that are beginning to fill the market in forms of entertainment systems, portable players, cell phones etc. As well as Video/Audio recognition in fields of security and military applications, which became so demanding in resent years. On the other side of spectrum, we have to consider field of digital communication and broadcasting, and associated with it data integrity, encryption/decryption and security, and at last, but not least in industrial applications such as intelligent manufacturing and advanced robotics on manufacturing plants of many large companies. In most of those areas a solution was to try to accommodate such requirements with PC solutions or embedded control systems. However, in certain cases these solutions are becoming not suitable due to fact of not being able to be fast enough, or due to limitation of power consumption, which is becoming much more critical in modern mobile as well as space oriented environment. On behalf of embedded solution it is important to note that most of the household devices include one or more processing units which are usually RISC microprocessors or microcontrollers, and they serve quite an important role. Disadvantage of these systems is that their execution cycles are quite slow and do not give enough of processing power for higher demanding application. In the case of high speed processing application usual approach for the embedded solution was Application Specific Integrated Circuit (ASIC) and it is currently the leading solution. However, the downside to ASICs is long time-to-market, costly development which is only profitable in case of production of many thousands of units and in most of the cases it consumes same amount of power even if it is not really used. On the other hand, implementation of such tasks on the Field Programmable Gate Array devices (FPGA), doesn't only decrease the time-to-market but also has a broad range of several advantages. In latest studies it was found that current cost-effectiveness of development of product / platform on FPGA has increased to the point where manufacturing of particular chip has to be in quantities of more then 100,000 pieces in most of the cases. In addition to this, modification / upgrades of the FPGA operation core is very simple and inexpensive since it requires modification of code written on one of the Hardware Description Language (HDL), recompilation and in most cases remote update of the platform. In case of ASIC it is simply not possible and requires remanufacture of the whole chip and physical replacement on the platform.

By using FPGA type devices it is possible to achieve highest cost-effectiveness by varying cores with partial configuration that is by means of spatial and temporal partitioning on an FPGA chip. At a point of execution when system is sitting idle or real-time task requires very slow response it is possible to load core which will not use much resources of the chip and therefore consumes much less power. Therefore the proposed work is oriented on development of FPGA based platform with run-time (dynamic) reconfiguration of internal FPGA recourses while task is executing, which allows reusability of the same computational recourses for different sections of the task and does dramatic minimization of hardware recourses and associated complexity, cost, and power effectiveness. This required development of spatial and temporal partitioning mechanism which allows run-time partial reconfiguration of FPGA device.

1.2 Project Objectives

Objective of this project was to research and develop Temporal Partitioning Mechanism (TPM) for the FPGA computing platform based on partially configurable SRAM FPGAs.

The Temporal Partitioning Mechanism should allow automatic run-time reconfiguration of the SRAM based FPGA resources (Logic, routing and Input / Output resources) for different segments of application algorithm and thus minimization of FPGA resources for a task.

This would require solving several problems starting from theoretical analysis to actual TPM development and implementation of the system in hardware and software followed by analysis of performance and resources consumption.

1.3 Original Contributions

Main contributions for this work consist of several parts that subdivided into three main categories:

1. Analysis of existing computing platforms with fixed and reconfigurable architectures which allows software or hardware adaptation on task algorithm and / or data structure. The focus was made on reconfigurable (FPGA based) systems

with ability for partial reconfiguration of its computational and interface resources. In addition to that theoretical comparison of the proposed approach to the existing solutions that are available on the market and developed by different research groups, the development of algorithms and procedures for the TPM was done.

2. As per completion of the theoretical component of the project, development of three prototypes of the platform with Temporal Partitioning Mechanism was conducted. This part of the project include development of the following parts:
    a. Hardware: platforms that allowed run-time partial reconfiguration of SRAM based FPGA with partially reconfigurable architecture (Xilinx Virtex II XC2V1000 FPGA) and all associated controllers and embedded cores including simulation, hardware verification and integration with firmware part and associated peripherals.
    b. Firmware: that consist of first version of Real-Time Hardware Operating System (RTHOS) and its FPGA resident, interface drivers and communication utilities which would control the peripherals such as Microcontroller and CPLD for correct platform operation
    c. Software Agent: special GUI / interface component running on the host-PC to perform operations such as communication with the platform, user interface, and data/core management.
3. Upon completion of the TPM implementation it became possible to run tests and experiments for comparison of proposed idea to already existing platform solutions based on different processors with architecture organization: Superscalar

architecture (e.g. PC-platform) and RISC with Harvard architecture (e.g. Microchip RISC controller). This allowed obtaining practical proof of the proposed concept of the FPGA platform with TPM.

1.4 Thesis Organization

This thesis consists of six Chapters and their organization is presented in the following manner.

- First Chapter introduces the concept of the work done, as well as, why this topic presented research interest. It also states, what were the objectives of work, and contributions that were made. Chapter concludes with a Chapter breakdown.
- Second Chapter consists of the literature overview from several journal publications, as well as books, internet sources and product datasheets. Based on this information, classification was made of major classes of existing platforms with different types of architectures. In addition their advantages and disadvantages were presented, as well as, their real world implementation. This Chapter gives overall background information on run-time reconfigurable computing platforms.
- Chapter three gives the observation of different computational schemes, and also presents comparative analysis to prove the concept of FPGA based platform with temporal partitioning of its computational resources. As well as analysis on minimization of hardware resource utilization in the FPGA based platforms with TPM vs. non-TPM FPGA platforms.

- Chapter four describes in detail the implementation of the Temporal Partitioning Mechanism platform in particular it describes all the aspects of the hardware used including the reasoning for usage of particular parts and their brief explanation. In addition to that all of the implemented prototypes are presented including the protocols and interfaces. On the software side, operation of the windows software agent is presented, in particular its graphical user interface, communication and data/core management.
- Chapter five talks about the results acquired from the experiments that were performed on the embedded system, PC, and Temporal Partitioning platform. It also presents quantitative comparison between the platforms and conclusions regarding the project.
- At last Chapter six concludes this thesis with recap on all of the done work and achieved results.

At the beginning of the work Table of contents and list of figures gives reader an ability to quickly navigate thought the thesis, and at the end of the thesis bibliography of the used materials is presented as well.

CHAPTER 2

OVERVIEW OF COMPUTING SYSTEMS WITH ADAPTIVE ARCHITECTURE

Introduction

In today's world, all of the personal and most of the computers in general are based on statically fixed architecture processors[1]. As we know, processors remain unchanged throughout their operational life and their operation is controlled by the software algorithms. Due to the fact that it was possible to change software algorithms or in other words programming of the processor, it was possible to achieve execution of the wide range of different types of tasks. On the other hand, this kind of approach presents a problem which currently is only resolved by technological advances. As tasks become more and more complicated and require complex operations such as Fourier Transform[2] it takes substantially longer for an ordinary processor to process it. For some applications a solution was found in a form of additional dedicated functional units (e.g. Float-Point Multipliers in case of DSP[3]) in order to accelerate the execution of the task. However, these types of processors are not very flexible and are application oriented. Therefore, in case if completely different type of task is given to the DSP processor its cost-effectiveness might be even worse then a general purpose processor[2].

In the following sections overview of several topics will be discussed regarding computer system architectures, sequential processors, as well as reconfigurable hardware, In particular processors with reconfigurable architecture.

2.1 Correspondence between algorithm / data structure and computing architecture.

Relation between the types of algorithm/data to be executed on a system and systems hardware architecture determines the performance of the system[4]. Therefore, if system was designed to perform only certain specific task it will have the highest performance for that particular task. However, if slightly different task is needed to be processed it may be much less effective. All though, if system is well adapted for several different types of operations it may be possible to perform wider range of operations but with much lower effectiveness. As an example of it, we can consider performing processing the 64-bit data on an 8-bit microprocessor [5]. Instead of single operation on a specific 64-bit processor it would take 16 or more cycles of operations using means of software algorithms. So we can identify that *effectiveness* of the system is closely related to the *complexity* of the system's components and correlation between the task algorithm and hardware architecture that is designed for processing the task, as well as *software* algorithms, which supplement hardware deficiency.

Therefore, choosing appropriate combination of complexity of system components, their correlation with the tasks directly results in the effectiveness of the system.

Generally there are two possible approaches that can be used in designing of processing systems:

- Sequential processors with fixed hardware architecture (components and links) and procedural adaptation to the application algorithm and data structure by reprogrammable software based on Instructions.
- Reconfigurable processors with hardware that can reflect algorithm and data structure of the application in its architecture at point of execution.

2.2 Processors with fixed hardware architecture and procedural adaptation.

Processors with fixed hardware architecture with no strict real-time time requirements are widely available on the market and generally classified as micro-processors and based on micro-processors different types of industrial microcontrollers, which are used in most of industrial applications. During the last decade when performance requirements for computing platforms have increased two architectural approaches have become the most popular solutions: Superscalar [6] and VLIW (Very long Instruction Word) types of processors [7]. Each of them has its advantages and pitfalls. Both of the architectures exploit a parallel instruction processing scheme, which means they run small number of the instructions (2-4) in parallel to be able to process application(s) faster. The idea of Instruction Level Parallelism (ILP)[7] is based on execution of instructions simultaneously and / or on a short pipeline (3-5 stages), which ideally allows increasing performance of processor. However it still doesn't achieve the desired instruction per clock cycle operation. This is due to a fact that several clock cycles are still wasted on control and service operations that are required to coordinate the operation of the processor. In addition clock cycles are also wasted in case of different hazards, which are divided on data, control and structural hazards. As an example branches cause huge amount of hazards [8] since if a branch is required especially if it is conditional branch whole pipeline has to be cleared and reloaded with the instructions that follow after branch, due to that problem there is no essential acceleration with using pipeline. Several complex control schemes such as branch predictors have been developed to be able to predict branches and avoid loosing clock cycles which can increase the productivity. Nowadays, superscalar processors are quite common and superscalar architectures are used in high performance systems as well as personal computers and latest hand held devices. Their high processing capability comes from the multiple internal processing units that accelerate specific functions such as floating point processing, and it is possible to have several of the same type of processing units, which allows pipelining the instructions without stalls. Some of the examples of the superscalar processors are MIPS R10000, DEC21164 [6]. Control units of the superscalar operate independently and user doesn't require knowing the architecture of the processors. Very complex control units embedded to the processor have their own bus which can access instructions and data independently which gives big acceleration of these processors comparing with traditional microprocessors.

However, the drawback of complex control units in the superscalar processors requires more hardware, power, as well as much longer design, testing and verification periods. Therefore, this kind of approach generates much more costly design.

Very Long Instruction Word [7] processors have a slightly different approach to increase productivity. Even though, they are similar in some sense to the superscalar architecture by having separate processing units that can execute several instructions simultaneously, it is necessary to place instructions in the program in specific order to simplify parallel issuing of instructions for execution. This approach allows dramatic simplification of control unit in VLIW processor and thus, reduction of its cost. If the program is well scheduled it is possible to achieve high performance with the minimum cost. That is why, in order for VLIW processor to work effectively, programmer has to program it in a way which will reflect not only algorithm and data specifics but also the hardware architecture of processor. Thus, this presents a problem since programming becomes very complex. The only area where VLIW processors seem to be very popular is the area of DSP application (e.g. TI TMS3206x [9]).

2.3 Processors with reconfigurable architecture.

Over past 15 years there has been an increasing demand for reprogrammable and reconfigurable computing devices[10]. Makimoto's wave – a prediction done in early nineties indicated that most of the computing technologies nowadays will need to include field programmability[8]. Reconfigurable platforms are gradually filling the gap between the ASICs that are oriented for high performance of one dedicated application and microprocessors, which can be easily adapted (programmed) to any application but has physical restrictions for their performance (sequential processing nature).

Several different types of reconfigurable architectures were developed and some are still in development, which mostly concentrated of having hybrid architectures of processing and configuration of communicating busses. Main distinction between reconfigurable processors is static and dynamic reconfigurability.

Static configurable systems are generally used in cases of rapid prototyping of ASICs since it is possible to design test and then make modification without involving manufacturing steps as it is done in case of real ASIC design [11]. Also, it is advantageous in cases of small production volumes of computing platforms when manufacturing of ASIC may be too costly[12][13]. In addition, rapid prototyping with reconfigurable devices allows shorting time-to-market, which is critical in current competitive environment. Usually statically configurable processors had a fine grain organization and such as Splash-2 [14][15], Achillies[16]. Most of the statically configured adaptive processors require long loading time due to the fact that routing as well as combinational logic has to be initialized. On the other hand, course grain architecture processors did not need to be configured for extensive periods of time since they have fewer connections and therefore could be *dynamically reconfigured*[17].

Dynamically reconfigurable processors are much more promising for the future implementation of most high performance systems due to several key features. An ability to reconfigure the device while other parts of it continue to operate allows reusing the hardware of the device for different types of functions. This gives a possibility of having a notion of non-fixed hardware or in other words virtual hardware, where virtual hardware is described in the Hardware Description Language (HDL) [19]. Dynamic reconfigurable processors can be divided onto three subcategories:

1. Processor with reconfigurable co-processor(s) [18].
2. Reconfigurable Processor with spatial partitioning [20].
3. Reconfigurable Processor with temporal partitioning [21].

Processor with reconfigurable co-processor creates so-called hybrid architecture [17][22][23][24] and allows reconfiguring the connection or routing from the main processing unit to several different components (co-processors), which than can perform the functional processing. However, there are still bottlenecks of performance related to the actual co-processors that are linked to the main processor. Considering that co-processors can not be fully adapted for the task it will probably loose some clock cycles on that operation. Some of the examples of such processors were the ones with reconfigurable busses: KressArray Family of processors, as well as: RAW (Reconfigurable Architecture Workstation) which incorporated MIPS superscalar processors; DReAM Array which was used in communication for mobile systems [25], PipeRench [26] and RaPid [17][27].

Reconfigurable processors with spatial partitioning of resources are systems that allow flexible distribution of hardware components *in space* of the homogeneous field of configurable logic and routing resources [26]. Reconfigurable processors with spatial partitioning of computing resources gives a possibility to increase the flexibility of the system by being able to select components that are required for the particular task and loading them in a FPGA. If different type of task is needed to be processed system doesn't need to be halted, it can be simply reloaded with different type of components which will be exactly suited for that particular task. As well as all of the components could be checked for correct operation, this will increase the fault tolerance of the system [28].

Alternatively, a reconfigurable processor with temporal partitioning of computing resources exploits reusing of the same computing resources of the system in different periods of time for different tasks or segments of a task. This allows minimization of hardware resources per task while keeping high performance for certain types of applications such as: data-flow tasks or streamed data processing applications (video/audio processing[29][30], DSP and digital communication and broadcasting, etc.). In this approach processing of a task is divided onto several stages. After completion of each stage temporal results are stored in memory and new core reconfigures computing resources for the new segment of algorithm to be executed. Since it is possible to have a division of the cores inside an FPGA, some of the cores can be reloaded often to perform certain tasks, where as others might stay there longer to perform completely independent tasks. Also it is possible to dedicate more resources if sudden requirement for a particular task has appeared either in the data flow graph (DFG)[31] or as an external event. These systems are more dynamic and reflect behavior of the natural world. Some of the examples of such systems could have been found even in the previous years (PipeRench [30][32][26], KressArray) [17], however they had much more course architecture and did not allow fine tuning for the algorithm at hand, and therefore were not used in the manner of partial configuration. More detailed explanation of the implementation of temporal partitioning system as Run-Time Reconfigurable (RTR) systems is presented in the next section.

Another big advantage of the pre-compiled cores is that since they are completely "Of-The-Shelf" (OTS) virtual units, internal routing is already embedded in these units. This fact can dramatically simplify virtual assembling of soft-cores on-chip by simple loading in different moment of time. Alternatively, the software compilation of different soft-cores and their mapping in space of logic and routing resources is a must. As it turns out from the experimental results, routing of the FPGA takes the most time for creation of the completely assembled core and even though right now it does not present a big problem since sizes of chips are still relatively small. However, as sizes of the FPGAs will continue to increase in their volume and complexity, it will be more and more complex to perform core compilation. As an example, Table 2.1 below indicates the core generation times and chip size predictions based on a 3Ghz Dual core CPU machine (PC-workstation) with a 1 GB of RAM.

Table 2.1: Core compilation time depending on amount of available logic resources in the Field Programmable Gate Array (FPGA) devices

FPGA family and Part number	Number of logic gates	Time of core generation
Xilinx Virtex-E (XCV-50E)	50,000	30 seconds
Xilinx Virtex II Pro (XC2V1000)	1,000,000	3.5 minutes
Xilinx Virtex 4 (XC4VLX200)	10,000,000	~1.0 hour
Virtex 4 100M system gates (in near future)	100,000,000	~10 hours.

As it can be seen from the table, as the size of the FPGA device increases, it will be more and more difficult to get completely compiled bit stream for and System-on-Chip (SoC) design. However, it will be much more advantageous to use already configured cores since routing and configuration outputs are fixed and was done before. In case of reconfigurable processor generation of design is going to be much simpler and the rest of the process will be left up to creating well scheduled reconfiguration scheme. Even if generation of specialized cores will be required it will not take that much time either since usually processing cores do not need to use whole chip and by restricting the routing only to a small area it will be able to achieve result in reasonable amount of time as it occurs nowadays.

However, to implement reconfigurable processors with spatial and / or temporal partitioning of computing resources it is mandatory to have appropriate hardware FPGA, which allows partial reconfiguration of its logic and routing resources. Partial reconfiguration in this consideration means that FPGA architecture allows reconfiguration of a part of the FPGA resources without suspension or interruption of operations conducting on the rest of FPGA device.

Currently on the market there is very limited availability of FPGA devices that allow partial reconfiguration. Atmel Inc. is producing FPSLIC (Field Programmable System Level Integrated Circuit) [33], which also combines a microcontroller, and FPGA and EEPROM. Downside of that particular device is that size and complexity of the FPGA which is included is quite small, only up to 40K system gates and not suited for high speed or complex designs. The only company that produces such family of FPGA devices is Xilinx Corp. Starting from Xilinx Virtex family and moving to Virtex II[34] and recently to Virtex 4, Xilinx fully captured the market for partially reconfigurable devices. That is why for further utilization of the unique features of these FPGA devices it is necessary to give some background regarding architecture organization of the above mentioned families of Xilinx FPGAs, and it is done in section 2.5. However, it is also important to reflect on already existing works and some implementations of the temporal partitioning for FPGA type systems, which is done in the next section.

2.4 Existing reconfigurable platforms with temporal partitioning of resources

In past couple of years several developments have been made in the FPGA temporal partitioning field and this section will talk about some of them giving a general overview and their approaches and implementations. Most of the works that have been found to be concentrated mostly on the actual approaches of division of tasks for the platforms allowing temporal partitioning of its resources, and had a very brief explanations of the actual hardware systems that were implemented or used for the temporal partitioning experiments.

One of the research groups from Cergy Pontoise University in France[35], proposed a very similar approach to the one presented in this thesis where actual algorithms were divided in the different cores. Based on the schedule of execution of the task, those algorithms were loaded one after another for execution on the ARDOISE FPGA platform, which they developed for the experiments. However, there are several disadvantages associated with this approach, which can cause major processing delays. Their implementation uses a host PC for scheduling of reconfiguration and a hard drive to store the precompiled cores. In addition to that they are using a technique of preloading the scheduled cores into a SRAM "core cache", and during the reconfiguration system's operation is halted. As it was mentioned previously a disadvantage of being tied to the PC for core storage doesn't give ability of being able to have the platform which is completely embedded and used in areas where PC can not be placed. In addition storage of cores on a hard drive can introduce significant stalls since the initial seek time of the hard drive is measured in milliseconds, where if it was implemented on board of the platform in FLASH memory avoiding all the communications channels access time of the required core is measured in max of 25 microseconds[36]. Also, it would not depend on the location of the core in FLASH memory, as well as, seek time would be exactly the same at any time of operation. In this thesis it is proposed to first develop the temporal partitioning mechanism and implement it in a first stage, which will give it capability of being completely embedded and only then move to the implementation of platform with embedded FPGA.

Another main disadvantage of the ARDOISE platform is that it was implemented on Atmel FPGA with 40K system gates[38][31] which allows only implementations of the very limited algorithms and this can cause a problem of requirement to divide task in much finer segments. This fact in turn would cause longer down-times of the FPGA platform during the reconfiguration cycles. If the implementation was done on the Xilinx Virtex II FPGA it would be possible to implement much more complicated algorithms since the sizes of Xilinx FPGAs are much bigger (eg. 8,000,000 system gates) [37]and can operate on much higher clock frequencies. In addition reconfiguration of the Atmel FPGAs is preformed at maximum frequency of 33MHz[31], where Xilinx FPGA reconfiguration can be performed at 50MHz[39], and this is very crucial in case of minimization of down-time for the processing platform.

Some of the works were more concentrated on development of tools [21] which would be able to perform the task partitioning for DSP type applications and not on the design of the actual hardware platforms. Similarly, there were other proposed approaches and applications[40] [35], [29],[21] which explored the theories behind divisions of the tasks as well as their use in areas of multi-media processing. As it was stated in the L.

Kessal paper [38] and this thesis proposed approach the temporal reconfiguration was suggested to be using an external data storage memory, as well as, "Stored Circuit Configuration Memory", however the emphasis was placed more on the software division of the tasks in to groups for later scheduling and processing. Most of the implementations were done on the simulators, and the actual implementations of the temporal partitioning platform were not discussed in detail. Some of the works related to the capability of implementation of the temporal partitioning platforms on an Atmel 4000 FPGAs for non-complex implementations, as well as Virtex family FPGA for much larger designs, as it was proposed for the final stage of implementation in this thesis. Therefore, it is important to briefly explain the organization of the Virtex Family FPGAs, which is done in the next section.

2.5 Virtex Family Architecture

Main components of the Virtex FPGAs are Configurable Logic Blocks (CLBs) [41], Input Output Blocks (IOBs), memory blocks (BRAMs), clock and configurable routing. Configuration of the components is based on configuration SRAM memory therefore it can be easily reconfigured by loading of configuration bit-stream to the configuration SRAM. In Figure 2.1 the general organization of the Virtex Family FPGA devices is presented. As it can be seen, CLBs placement is structured in rows and columns as well as intermixed with the BRAM (Block RAM) columns. IOBs are located on the perimeter of the chip and can be configured to different part of chip over a very complex routing network.

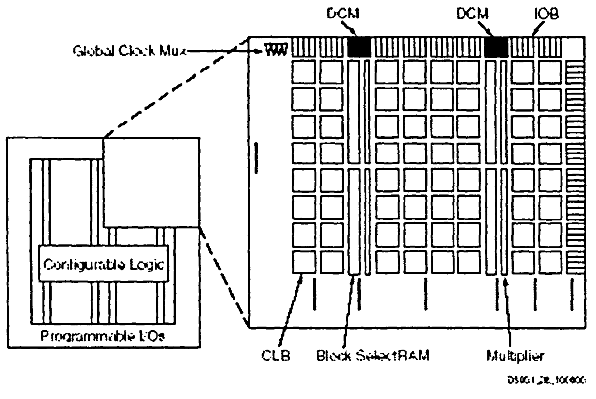


Figure 2.1: Diagram of Virtex II FPGA organization (courtesy of Xilinx Corp.)

As it was stated before Virtex FPGA devices have a capability of reconfiguration of its parts while other parts of FPGA are still in operation, however there are some restrictions associated with that feature. For example Virtex II devices can be partially reconfigured since *frames* (columns of CLBs) can be configured as a group. However, it is not possible to configure single CLB individually since the frames belonging to a given CLB are common to all other CLBs in the column. On the other hand, Virtex 4 devices have a capability of addressing and reconfiguration of individual CLBs.

2.6 General approach to designing partially reconfigurable processing system

An approach that was used in number of cases [42],[12] and also recommended by the Xilinx Corp.[43] suggests developers to divide the application of the design in a few parts. First, a controller has to be designed which will be able to perform the reconfiguration of the FPGA device. It can be an external controller such as Microcontroller or CPLD, as well as an internal controller. Xilinx Virtex II / II Pro FPGAs actually include an internal configuration access port (ICAP) [41] which can reconfigure frames of the FPGA during its operation using a embedded SelectMAP

(Byte-parallel) protocol [44]. ICAP transfers data over an 8-bit parallel configuration data bus, which allows configuration at a speed of 400 M bits per second. Organization of the FPGA has to be done in a specific way, similarly to how it was suggested by the group from Montpellier University (France) [42] in order to allow the partially configured units to communicate with each other. Figure 2.2 illustrates a suggested setup where several processing units represent the possible positions for activated cores and main controller resides at the bottom side of the chip.

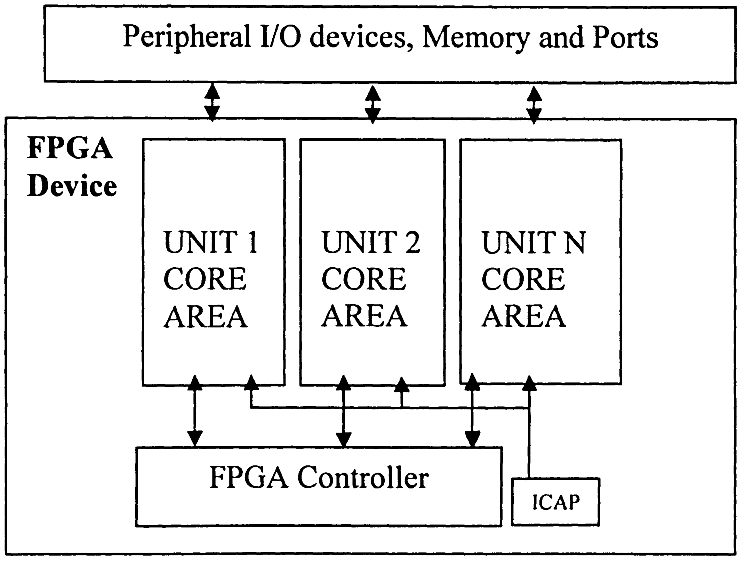


Figure 2.2: FPGA organization for partial reconfiguration

Communication between the units is done using the tri-stated interconnection with the main controller. Controller itself contains three cores: communication bus, which connects the unit cores; arbiter, which allows data transmission between units on a given data-line and master core, which is responsible for the communication with an external world. Due to the limitation of the development kit it is necessary to include an empty unit cores in the original design including all the tri-state buffers and routing, since it is not possible to restrain the wire routing positions.

Another essential part of the partial reconfigurable system that University of Montpellier group developed is a CAD tool which is called *core unifier* which generates loading bit-streams by combining different partial cores, which user has selected. The CAD tool basically automates the process that usually has to be done manually by the developer since Xilinx ISE CAD system doesn't directly support this feature. Essentially *Core Unifier* opens a default template with the dummy empty units and allows developer to insert the desired core into a specific spot instead of one of the dummy units. Following that procedure CAD tool creates the partial bitstream which can be loaded into the FPGA device. However in order to make sure that all the cores will operate correctly they have to be initially generated with the restriction of the area and also generated with all the tri-state buffers fixed, otherwise partial design that would be inserted into the operating device will cause incorrect operation and may even require complete re-initialization. But there still exists a possibility that the design will not operate correctly according to the paper that was submitted by Montpellier [42] group. The problem lays in actual Xilinx ISE CAD, which can not fully constrain core logic to reside in the bounding box defined by floor planning, as well as it can not fully constrain the routing of the chip within the floor planer, and at last but not least it is not possible to define exactly the same routing between the tri-state buffers.

2.7 Cores library and database

In order to implement truly dynamic partial reconfigurable system it is necessary to have a set of generated cores that can be loaded as a partial bitstream into the FPGA to perform certain function[45], based on the research paper by National Institute of Astrophysics Optics and Electronics in Puabla Mexico. A database of cores is classified by the type of core such as: communication, processing, testing, etc. Also, since cores are device dependant it will be needed to have classification based on the type of the device as well. In some cases different versions of the core can be made depending on their speed and power usage which can be classified in this database. This kind of organization opens up a lot of possibilities for development groups, since a platform can be used by several users and doesn't even have to be located at the developer's station. There is additional benefit which relates to the safety of the intellectual property, which is a big issue due to the problems of reverse engineering. With the capability of remote upgrade or remote partial configuration [46], product that was sold to a customer can contain only hardware and its base core could be kept completely empty. At the point of installation of the hardware by the user a secure link is established by the system with the request for a specific core which client has license to. This way it becomes practically impossible to perform reverse engineering on the device. As an additional benefit, a new update to the system can be loaded without user's knowledge and system interruption. The other possible scenario how remote partial configuration can be beneficial is that the hardware can be sold before the firmware part is completed and loaded to the target hardware only after its shipment and installation, which sometimes takes several weeks.

2.8 Configuration bit-stream transfer mechanism

The other issue that has to be addressed is a transfer mechanism, which will perform the actual loading of cores from the source server to the target platform. In order to do this task a communication link has to be established between the server and the platform, however that doesn't cause a big problem since target hardware usually has some means of communication with the outside world and can use the same channel to do so. In order to guarantee security of the transfer it is possible to use already existing encrypting algorithms for protecting the so-called IP (Intellectual Property) from being copied or tampered with. As a suggestion of implementation from article from Xilinx [43] it is beneficial to have an EEPROM/FLASH memory on the designed platform due to the fact that transmitted core has to check errors and compare the computed CRC to the CRC of the transmitted bit-stream configuration file. Another benefit of on-board EEPROM / FLASH memory appears when the system has to be powered down but the initial configuration still resides in the EEPROM memory. When system is powered up again it will be loaded locally from the EEPROM using microcontroller or CPLD over a parallel SelectMAP port [41]. On the other hand, in the project described in this paper, it was decided to extend the Xilinx recommendation to a system that will allow the user to load all the required cores and instead of retransmitting the cores from the source to the target FPGA, to be able to transmit the identification of the core that is needed to be loaded. Updates or new cores could be loaded to the EEPROM in non-critical time which would not cause delay in the operation. Currently sizes of EEPROM or FLASH memory increased to the amount where thousands of configuration bit-streams of processing soft-cores could be stored in a single FLASH chip. Considering that cores for partial configuration require much less configuration bits than a bit-stream for the complete FPGA device, it could be assumed that it is possible to hold any combination of cores for the FPGA operation. With the capability of using partial reconfiguration and all necessary functional cores in FLASH memory, adaptation of the reconfigurable processor to any data-flow application requires only proper scheduling of core loading to the FPGA. Further data execution could be done automatically under control of either a controller or a source server which could control the sequence of core transfer process. Detailed this concept will be discussed in the next Chapter 3.

Summary

In conclusions it is important to note that fixed architecture systems are widely used and still under development and improvement as we can see from a PC CPUs, and probably stay for a long time, as well as small embedded microcontroller systems that are everywhere from household appliances to space probes. We also should mention that processor architectures such as Superscalar and VLIW allowed to increase the productivity of the systems by employing Instruction Level Parallelism (ILP), however, their productivity closely depends on a software algorithm programming, and therefore do not perform at their peak performance. Although fixed architectures processors are beneficial in certain application, they lack flexibility of reconfiguration for completely different types of tasks. Reconfigurable processors have an advantage that they can be well adapted for different types of tasks/algorithms by supplying them with one or other configuration core and there are several distinctions between the reconfigurable processors. Statically configurable processors are used in cases of rapid-prototyping and are oriented for small scale production. Dynamic reconfiguration processors present an attractive opportunity for most of the applications since its functionality can be changed real-time without processing interruption of the other parts of the processor. Especially it is useful in cases of temporal partitioning where the execution of the algorithm is based not on generation of specific core, but on actually creating a data flow graph from the selection of existing cores; as well as new cores, which can be quickly generated for performing of specific task.

CHAPTER 3

ARCHITECTURE OF FPGA BASED PLATFORM WITH RUN-TIME TEMPORAL PARTITIONING MECHANISM

Introduction

In this Chapter the examination of different computational schemes is presented. The general analytical models are created for performance estimation of different data processing platforms based on conventional microprocessors and processors with instruction processing pipelines. The comparative analysis was done to prove the concept of FPGA based platforms with temporal partitioning of hardware resources. Based on the above, the architecture organization of the FPGA platforms with embedded Temporal Partitioning Mechanism (TPM) is described. Finally, the comparative analysis is done regarding minimization of hardware resources utilization in the FPGA based platforms with TPM comparing with regular FPGA platforms.

3.1 The Concept of Run-Time Temporal Partitioning of Hardware Resources

In most of industrial applications FPGAs are dedicated to one particular task, and do not change configuration of digital circuits while working. However, SRAM based FPGAs by its nature can perform run-time reconfiguration (RTR) and thus, reuse of internal hardware resources for different tasks or parts of a task. This approach allows fast optimization of the cost-performance characteristics of FPGA-based platform.

There are two possible mechanisms for reusing hardware resources of the SRAM based FPGA:

a) Spatial partitioning of resources – sharing the area of uniformed logic and routing resources between tasks inside the FPGA and

b) Temporal partitioning of resources – sharing the same resources between tasks or their segments in different periods of time.

In this project the research was focused on the research, development, and implementation of the temporal partitioning mechanisms for data-stream processing, implemented in partially reconfigurable FPGA devices.

Scheduling of an application begins from the analysis of the Data Flow Graph (DFG) of a task and performance parameters (response time, data rate, latency, etc.). For example in case of video-processing application, there is given time period when video-frame(s) should be processed. This period depends on frame rate, GUI (Graphical User Interface) requirements, etc. In case of network communication this period depends on network bandwidth, packet size, etc. Let us consider an example of video-processing task where stereo video-frame (2 x 640 x 480 pixels) has to be captured and processed with 30 frames / s (within 33 ms / frame). Let us also consider that maximum resources are given to the task and internal clock frequency is equal to 200 MHz (clock rate = 5 ns). Assuming that one pixel is processed within 1 clock cycle and all pixels are progressively processed, the video-frame processing time will be equal to: 2 x 640 x 480 x 5 ns = 3.072 ms.

The difference between video-frame capturing period (33 ms) and video-frame processing time (~3 ms) is about 11 times. Obviously, this difference will increase when clock frequency increases. Therefore, having data-acquisition time or real-time control timing requirements much longer than processing time of the associated data block, the idea of temporal partitioning of hardware resources would look very attractive. In the Figure 3.1 it is shown the example of task DFG and simple data-frame processing schedule. This schedule consists of data processing periods (according to DFG statements -*Si*) and reconfiguration periods for the next DFG statement (*RSi+1*). This example shows that utilization of FPGA resources can be improved by pipelining of the reconfiguration and computation processes while next data-frame is capturing.

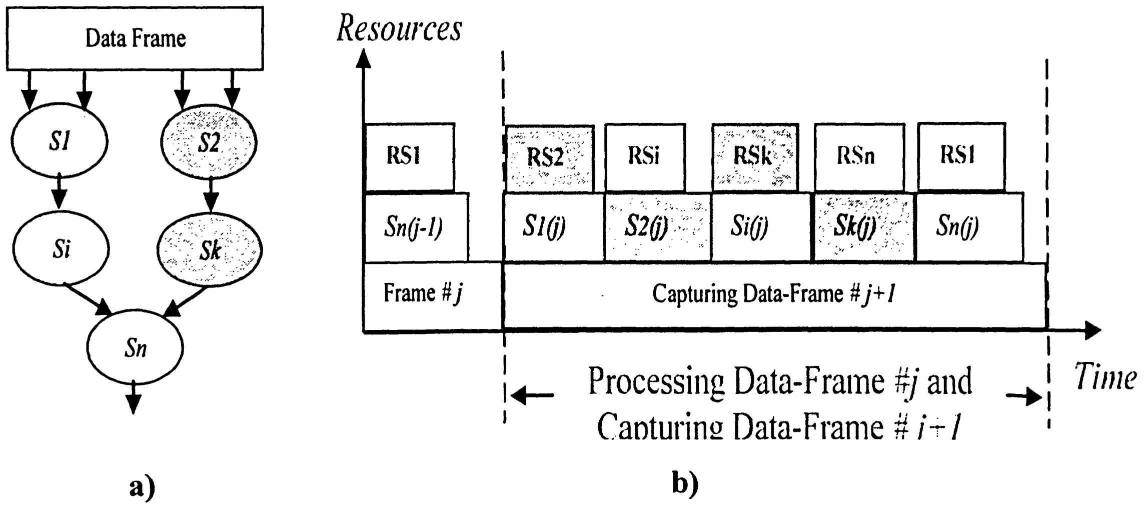


Figure 3.1: Data-Flow Graph (DFG) of a task and associated schedule of data-frame capturing and processing combined with reconfiguration of FPGA resources

Let us compare performance of data-paths in the following computing architectures:

i) Conventional microprocessor [8],

ii) Processor with instruction processing pipeline [5],

iii) Proposed FPGA platform with temporal partitioning of resources.

In all of the above cases we assume that the same task is executed. In other words same data-frame is processed by the same procedure presented on the Data-Flow Graph – DFG (Figure 3.1 a).

3.1.1 Performance of the conventional microprocessor

In case of conventional microprocessor the data-frame processing time will be equal to the sum of processing times of each element of data-frame (e.g. pixel) on each DFG statements – *Si,* when each statement in general case is implemented as a linear segment or subroutine in the program. Thus, we can consider each statement – *Si, i =1,2, ...n* as a list of instructions (machine code). In this case processing time of a statement $-T_{Si}(uP)$ is equal to the sum of execution times of all instruction in the procedure associated with statement - *Si, i =1,2, ...n* and can be calculated by the following formula:

$$T_{Si(uP)} = \sum_{j=1}^{Ki} CCj \times \tau_{cc}\ , \qquad (3.1)$$

where $j = 1,2,...Ki$ is instruction number in the procedure of a statement – *Si,*

CCj – is number of clock cycles needed for execution of the instruction #***j*** and

τ_{cc} - is the clock cycle period

Thus, a data-frame computation time - T_{DF} *(uP)*can be calculated by formula assuming that each data element (e.g. pixel or packet element) should pass (in worst case) all statements in associated DFG:

$$T_{DF}(uP) = [\sum_{i=1}^{n} (\sum_{j=1}^{Ki} CCj \times \tau_{cc})] \times Nd \quad , \quad (3.2)$$

Where ***Nd*** – is number of data elements in a data-frame (e.g. number of pixels in the video-frame equal to resolution of camera).

3.1.2 Performance of the processor with instruction processing pipeline

In case of the processor with architecture that exploits Instruction Level Parallelism (ILP) the instruction processing pipeline should be considered [5]. In general case there are several stages of instruction pipeline: IF- Instruction fetch, ID – Instruction decode, RD – Read data or "Read Operands", EXE- Execution and WR- Write result [8].

Definitely, there are different variants of pipeline architectures that are utilizing different of instruction processing schemes and hazards elimination circuits (e.g. Harvard architecture, forwarding, etc.). In this consideration we can simplify this case to the system that sequentially process instructions (one after another) when each instruction execution process consists of the above 5 stages. This process is illustrated in Figure 3.2.

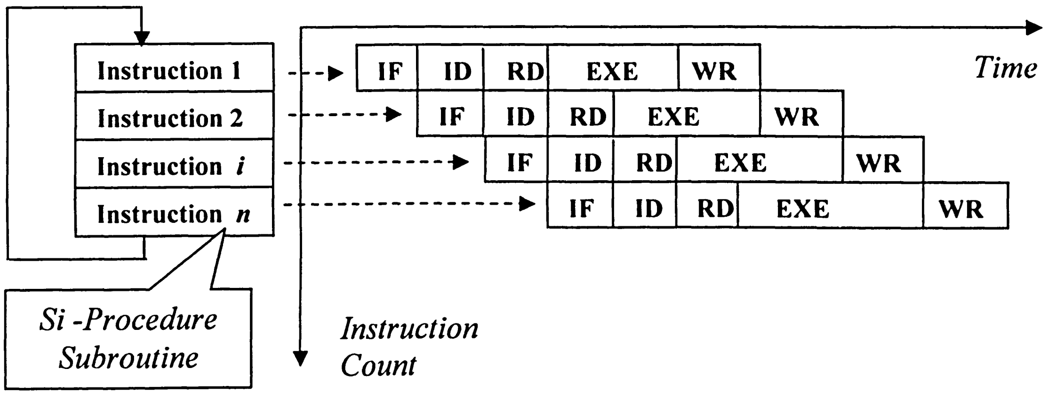


Figure 3.2: Pipeline processing of Program Instructions without hazards

Let us assume that instruction execution process is close to ideal situation when no data dependency between data elements in the data-frame and no structural hazards occurs. Let us also assume that each statement – *Si* procedure is organized as a loop and each stage of instruction execution requires same time for any instruction in the *Si* procedure. In this case we can estimate data frame processing time –***T****Si (pipeline)* on each statement – *Si,* by the following formula (3.3):

$$\boldsymbol{T}_{Si}\ (pipeline) = [\sum_{p=1}^{S} CCp + CC_{BRANCH} + Nd \times (\sum_{j=1}^{Kj} CCstall\ (j) + CCoutput)] \times \tau_{cc},$$

where: ***CCp*** – is a number of clock cycles for *p* – stage of the pipeline (e.g. IF, ID, RD, etc.).

In case shown in Figure 3.2 - $\sum_{p=1}^{S} CCp = CC_{IF} + CC_{ID} + CC_{DF} + CC_{EXE} + CC_{WR}$ - is simply a sum of clock cycles of all stages of the instruction execution process, which is equal to initial time delay (latency) caused by filling of the pipeline.

CC*BRANCH* - is a number of extra clock cycles spent for a branch instruction assuming that branch predictor is ideal and control hazard appears only once in the *Si* procedure.

CCstall(j) - is a number of stall clock cycles caused by the instruction #***j,*** where

j =1,2,...Ki is instruction number in the procedure loop of a statement – *Si,*

CCoutput – is a number of clock cycles for the output stage of the instruction cycle (e.g. WR-stage in Figure 3.2).

Nd – is number of data elements in data-frame.

τ_{cc} - is the clock cycle period

Considering the *Si* execution procedure as a subroutine (function) we can define the formula of complete data-frame execution time, which is the sum of all data frame processing time –$T_{Si\,(pipeline)}$ of each statement – *Si*, $i=1,2, \ldots n$

(3.4)

$$T_{DF} = \sum_{i=1}^{n} \left[\sum_{p=1}^{S} CCp + CC_{BRANCH} + (Nd - 1) \times \left(\sum_{j=1}^{Kj} CCstall\,(j) + CCoutput \right) \right] \times \tau_{cc}$$

As it was shown in [8] it is very difficult to create general analytical model for instruction level pipeline because it strongly depends on many architectural factors as well as organization of the instruction set itself. However, the above model may be useful tool for comparison the performance of existing computing platforms with the proposed FPGA based platform with temporal partitioning of hardware resources.

3.1.3 Performance of the FPGA platform with temporal partitioning of resources

In case of the FPGA based computing platform with temporal configured resources each statement *Si* could be represented by respective configuration file of the Virtual Hardware Component (VHC) – soft core of application specific circuit with architecture optimized on data processing by the algorithm of the DFG statement – *Si*. Thus, instead of pipelines oriented to processing the instruction word the data processing pipeline is used [47]. Long pipelines of application specific data-paths in each virtual hardware component - VHC (e.g. FFT, Edge Detector, Matrix Multiplier, etc.) associated with each statement *Si* of the DFG, allows processing data-frames with the speed comparable to

ASIC performance. Thus, loading the virtual hardware components from the configuration memory according to the processing schedule (Figure 3.1 b) will allow the FPGA based platform reaching required high performance with minimum computational resources. At the same time "programming" of a task (application) can be simplified by development of only the processing schedule. This becomes possible because Virtual Hardware Components are already precompiled and stored in the library of VHCs (similar to Instruction Set in regular microprocessor).

The performance of the FPGA platform with Temporal Partitioning Mechanism (TPM) strongly depends on organization of reconfiguration scheme and VHC architecture. Considering VHC architecture as a single data path with single (global) clock synchronization and dual-port data-memory read / write interface (Figure 3.3) we can find out the time of data-frame processing on VHC associated with respective DFG statement – *Si*.

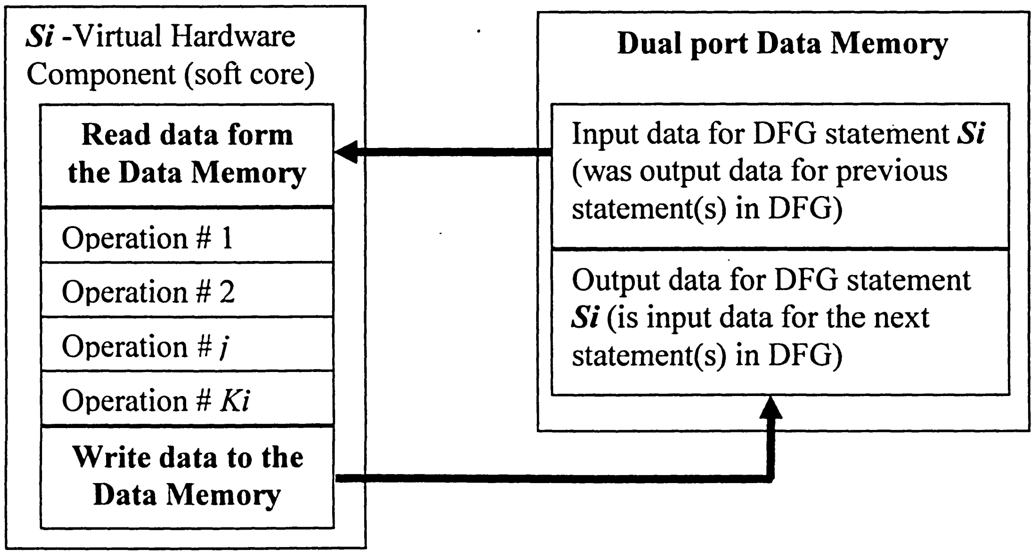

Si -Virtual Hardware Component (soft core)
Read data form the Data Memory
Operation # 1
Operation # 2
Operation # *j*
Operation # *Ki*
Write data to the Data Memory
Dual port Data Memory
Input data for DFG statement *Si* (was output data for previous statement(s) in DFG)
Output data for DFG statement *Si* (is input data for the next statement(s) in DFG)

Figure 3.3: Organization data processing path in the FPGA platform with TPM

In this case data-frame processing time (in worst case when all elements of data frame should be processed) can be calculated using the following formula:

$$T_{Si\,(TPM)} = \left[\sum_{j=1}^{Ki} CCj + (Nd - 1) \times max\{CCj\}\right] \times \tau_{cc}, \qquad (3.5)$$

Where:

CCj – is a number of clock cycles for Operation #*j*, *J=1,2, ...Ki* of the VHC associated with DFG statement – *Si*. (Figure 3.3).

Nd – is number of data elements in data-frame.

τ_{cc} - is the clock cycle period

Ki – number of operations to be executed on the soft-core of DFG statement ***Si.*** *I=1,2,...n*

Assuming that TPM provides effective pipelining of the reconfiguration and data computation processes while next data-frame is capturing, we can determine the complete data-frame computation time as a sum of $T_{Si\,(TPM)}, i=1,2,\ldots n$

$$T_{DF\,(TPM)} = \sum_{i=1}^{n} Tsi(TPM) = \left(\sum_{i=1}^{n}\left[\sum_{j=1}^{Ki} CCj + (Nd-1) \times max\{CCj\}\right]\right) \times \tau_{cc} \qquad (3.6)$$

The above formula may be used for performance estimation of data-frame processing on the FPGA based computing platforms with Temporal Partitioning Mechanism. This mechanism should provide pipelining of the reconfiguration and data computation processes and the Data Flow Graph (DFG) of a task should be acyclic (or can be converted to acyclic form).

3.2. Performance comparison between the FPGA platform with TPM and platforms based on processors with fixed architecture

To compare performance of FPGA platform with TPM and performance of conventional microprocessor formulas (3.6) and (3.2) should be analyzed:

$$T_{DF\,(TPM)} = (\sum_{i=1}^{n} [\sum_{j=1}^{Ki} CCj + (Nd-1) \times max\{CCj\}]) \times \tau_{cc} \qquad (3.6)$$

And $$T_{DF\,(uP)} = [\sum_{i=1}^{n} (\sum_{j=1}^{Ki} CCj \times \tau_{cc})] \times Nd \;, \qquad (3.2)$$

The above formulas shows that pipelining of data path in the FPGA based platform with TPM gives very high performance acceleration comparing to microprocessor with fixed achitecture. Obviously, data-frame processing time in case of FPGA platform linearly depends on number of data elements multiplied on longest operation cycle – $(max\{CCj\}) \times \tau_{cc}$. That is true in most of cases because number of stages in the data-path for each statement ***Si.*** *I=1,2,...n,* is negligibly small comparing with number of data elements in a data frame. In other words: $\sum_{j=1}^{Ki} CCj << (Nd-1) \times max\{CCj\}$. Thus,

$$T_{DF(TPM)} = (\sum_{i=1}^{n} [\sum_{j=1}^{Ki} CCj + (Nd-1) \times max\{CCj\}]) \times \tau_{cc} \cong (\sum_{i=1}^{n} [Nd \times max\{CCj\}]) \times \tau_{cc}$$

Therefore, assuming that clock period $-\tau_{cc}$ is the same for both platforms, the FPGA based platform with TPM gives performance acceleration comparing to conventional microprocessor equal to:

$$A_{(TMP\,/\,uP)} = [\sum_{i=1}^{n} (\sum_{j=1}^{Ki} CCj \times \tau_{cc})] \times Nd \;/ (\sum_{i=1}^{n} [Nd \times max\{CCj\}]) \times \tau_{cc}$$

$$= \sum_{j=1}^{Ki} CCj_{(\mathrm{uP})} \;/\; max\{CCj_{(TPM)}\}$$

Thus, acceleration does not depend on number of elements in data frame as well as number of statements in DFG but is proportional to the number of instructions in each DFG statement ***Si*** and number of clock cycles spent for each instruction in the microprocessor and longest operation cycle in the FPGA based platform.

For example if average number of instructions in each statement execution procedure is equal to 100 and average number of clock cycles per instruction is equal to 10 c.c. when the longest operation cycle in the FPGA platform usually does not exceed 1 c.c. than the acceleration will be equal to:

$A_{(TMP\,/\,uP)}$ = 100 instructions x 10 c.c. / 1 c.c. = 1000.

Thus for most of real applications FPGA based platform with TPM can give performance *acceleration from 2 to 3 orders of magnitude.*

Let us now compare the proposed FPGA based TPM platform with the processor exploiting Instruction Level Parallelism (architecture with instruction processing pipeline). Using formulas 3.4 and 3.6 the acceleration of FPGA based platform with TPM can be determined:

$$A_{(TMP/Pipeline)} = \sum_{i=1}^{n} [\sum_{p=1}^{S} CCp + CC_{BRANCH} + (Nd-1) \text{ x }$$

$$(\sum_{j=1}^{Kj} CCstall\ (j) + CCoutput)] \text{x } \tau_{cc} \ / \ (\sum_{i=1}^{n} [\sum_{j=1}^{Ki} CCj + (Nd-1) \times max\{CCj\}]) \times \tau_{cc}$$

In data frame processing applications initial latency of instruction pipeline - $\sum_{p=1}^{S} CCp$,as well as, number of clock cycles required for branch execution is negligibly small comparing to number of clock cycles needed for data processing in pipelined processing procedure for any of DFG statements. Therefore, the above formula for $A_{(TMP/Pipeline)}$ can be simplified as follows:

$$A_{(TMP/Pipeline)} = Nd \text{ x} \sum_{i=1}^{n} [\sum_{j=1}^{Kj} CCstall\ (j) + \text{CCoutput}] \ / \ \sum_{i=1}^{n} [Nd \times max\{CCj\}]$$

$$\cong (\sum_{j=1}^{Kj} CCstall\ (j) + \text{CCoutput}) \ / \ max\{CCj\}$$

Thus, acceleration value depends on number of clock cycles in the output stage of the pipeline plus additional clock cycles when the pipeline has been stalled.

Let us consider the ideal situation when no stall cycles appears in no one statement processing procedure (subroutine) and number of clock cycles for each instruction processing pipeline stage is equal to the number of clock cycles for longest operation in the FPGA based platform with TPM. In this case acceleration of the FPGA platform will be: $A_{(TMP/Pipeline)}$ = (0 + CC) / CC = 1. In other words, FPGA base platform with TPM always has higher performance than the platform based on processor instruction processing pipeline. If we consider more realistic case there are 20% - 30% of stall cycles in every loop of data element processing procedure. On the other hand the number of output clock cycles associated with Write Result stage (writing to memory or cache) is about 4 – 20 clock cycles. Thus, if consider the example with 100 instructions in the statement processing loop and 20% of stall cycles in average loop as well as 4 clock cycles per WR stage of the pipeline the acceleration of the FPGA based platform will be:

$$A_{(TMP/Pipeline)} = (20 \text{ c.c.} + 4 \text{ c.c.}) / 1 \text{ c.c.} = 24.$$

Therefore, for real applications FPGA based platform with TPM can give performance *acceleration in a range of one order of magnitude* comparing with most popular processing architectures that utilizes Instruction Level Parallelism in their architecture.

In the Chapter 5 the results gained on the prototype of the FPGA platform with TPM are discussed and performance acceleration comparing with RISC embedded microcontroller and AMD 3000+ based platform is estimated.

3.3 Architecture Organization of an FPGA based Platform with Run-Time Temporal Partitioning Mechanism

The concept of run-time temporal partitioning of hardware resources in the SRAM based FPGA devices described in the Section 3.1 and organization data processing path in the FPGA platform with temporal partitioning mechanism (Figure 3.3), dictates the architecture organization of the FPGA based platform with TPM.

Major components to be included into this architecture should be as follows:

[1] Reconfigurable Field of Operating Resources based on the FPGA device with partially reconfigurable architecture. FPGA micro-architecture consists of blocks of uniformed Configurable Logic Blocks (CLBs), reconfigurable routing resources, dual-port Blocks of RAM (BRAM) and reconfigurable Input / Output Blocks. Additionally modern FPGA devices include different types of embedded hardware cores such as: conventional processor(s), multipliers, Gigabit serializers, etc. [34]. Partial reconfiguration of internal hardware resources allows reconfiguration of any part of the FPGA hardware resources without suspension or interrupt of the rest circuits in the FPGA device [39]. This unique feature of the recent families of XILINX Virtex FPGA devices allows allocation of all Reconfigurable Field of Operating Resources inside the FPGA. This component should contain the following major elements:

i) Reconfigurable data-processing area (Data Processing Slots - DPS) for Virtual Hardware Components (VHCs) - soft-cores associated with DFG statements of each task (application);

ii) Configuration controller for run-time reconfiguration of Data Processing Slots (arrays of Configurable Logic Blocks with associated local routing and BRAM) and proper interfacing to the internal and external hardware components;

iii) Resident of the Real-Time Hardware Operating System (RTHOS) – soft-core of the RTHOS that allows keeps track of current situation in the Reconfigurable space and provides control information regarding next cycle(s) of reconfiguration in accordance to the temporal partitioning schedule.

[2] Reconfigurable Field of Memory Resources based on static random access memory (SRAM) devices interfaced to FPGA. This part of architecture in conjunction with Block RAM modules embedded in the FPGA is divided on the following components: i) Memory for raw data collected from input data-frame sources (e.g. video-cameras), ii) Buffers for processed data frames between DFG statement processing procedures, iii) Memory for output data-frames to be sent via output ports or communication links, iv) Cache for configuration bit-streams for Virtual Hardware Components (VHC), requested by the active task (application);

[3] Soft-Core Memory based on flash memory devices and contains the Library of soft-core configuration bit-streams of all Task Initial Architectures with associated Virtual Hardware Components;

[4] Real-Time Hardware Operating System (RTHOS) is a soft-core loadable to the FPGA. This component performs the following functions:

i) Real-time scheduling and monitoring of data-frame processing operations,

ii) Real-time performing FPGA partial reconfiguration operations (configuration bit-stream transfer from flash to memory and from memory to FPGA,

iii) Synchronization of data transfer and computation processes;

[5] Reprogrammable Controller-Loader is responsible for the following operations:

i) Loading initial soft-core (resident) of the Real-Time Hardware Operating System,

ii) Loading task (application) schedule – Task Schedule File (TSF) to the RTHOS Resident - TSF-buffer allocated in the FPGA Block RAM,

iii) Reprogramming of Soft-Core Memory,

iv) Communication with Design center for updating task schedules and / or Virtual Hardware Components Library if requested.

[6] Reconfigurable Input / Output Interface allows flexible configuration of input / output operations with custom data transfer protocols and programmable bit-rates.

3.3.1. Temporal Partitioning Mechanism: Components and Principle of operation

Temporal Partitioning Mechanism (TPM) consists of the components located inside the FPGA device and external devices. The Block diagram of the Platform with Temporal Partitioning Mechanism is presented in Figure 3.4. In this diagram the hardware partitioning of the FPGA based Re-configurable Field of Operating Resources is shown. Initial architecture consist of the following soft-cores: i) Data Frame Acquisition Module - DFAM, ii) Data Processing Slot 0 – DPS0, iii) Data Processing Slot 1 – DPS1, iv) Configuration Controller for Data Processing Slots, v) RTHOS Resident with Task Schedule File Buffer (TSF - Buffer).

The initial architecture is loaded to the FPGA by the Reprogrammable Controller-Loader through the standard configuration port (JTAG). The initial architecture is task specific and is stored in the Soft-Core Memory together with associated Virtual Hardware Components associated with task DFG. When initial architecture is loaded into the FPGA, the input data-frames, starts to come to the data-frame acquisition module - DFAM. This module is application specific soft-core that includes hardware (electrical) interface to the data-stream source and associated with this interface data transfer protocol (i.e. I2C, UART, Firewire, Centronics, etc.). Data Frame Acquisition Module is responsible for receiving, pre-processing input data (e.g. decompression, decryption, etc.) and storage data in required order in one of memory bank (Raw Data Bank).

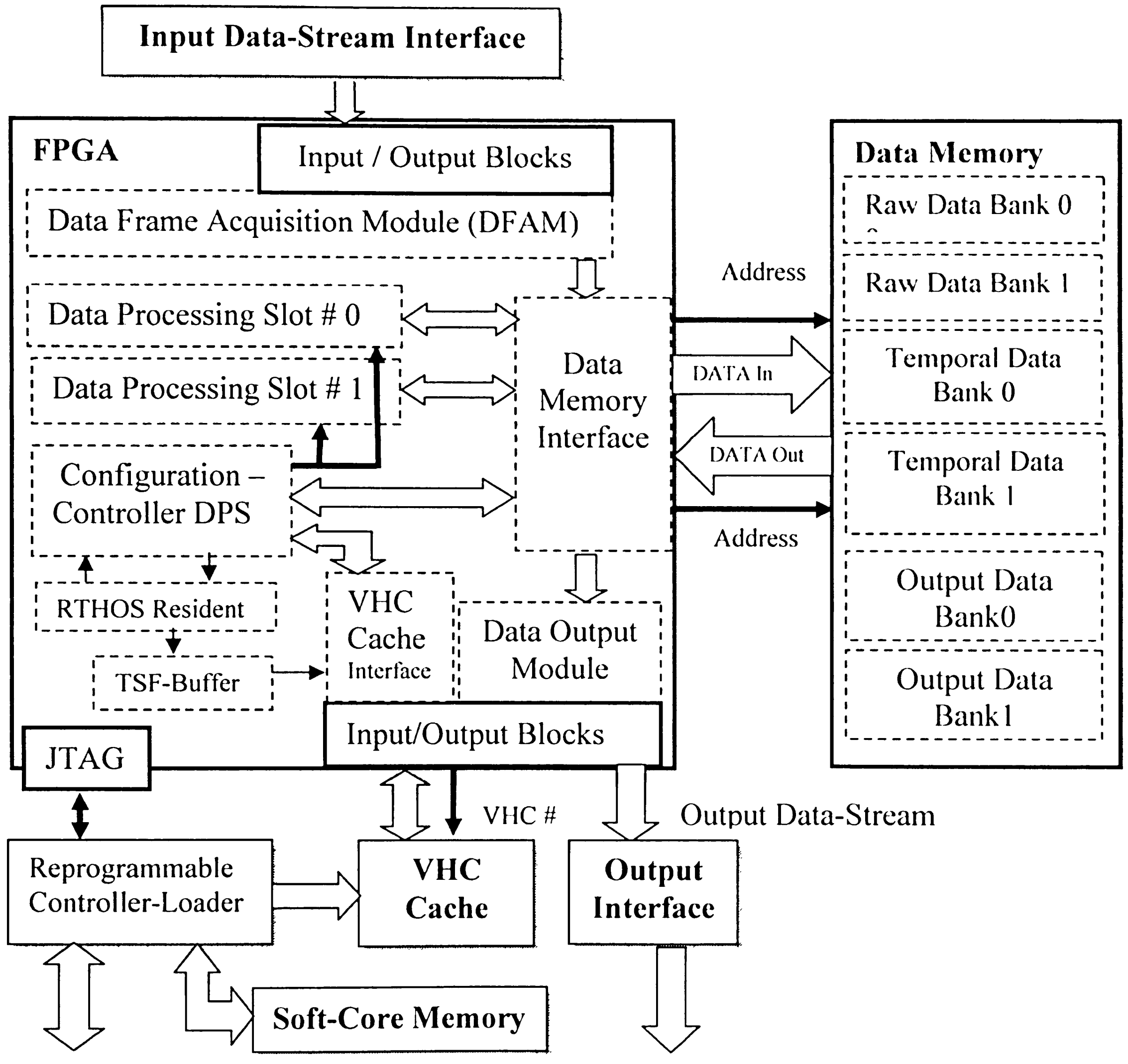


Figure 3.4: Block Diagram of the FPGA based platform with Temporal Partitioning Mechanism. Soft-core (loaded into the FPGA) modules are shown as dotted boxes.

At the same time <u>Reprogrammable Controller-Loader</u> transfers the Task Schedule File (TSF) to the RTHOS Resident. The RTHOS Resident stores TSF in the TSF-buffer (BRAM-based) and initiates loading DPS0 and DPS1 with the requested VHC. The number of this VHC is encoded in the first TSF word. The TSF-word contains the physical address of the VHC, which determines the place of the requested VHC in the Soft-Core Memory (Library of Virtual Hardware Components). The proposed approach assumes that VHC location (initial address of the VHC in the Library) is known for the programmer or compiler. Using the information in the first TSF-word retrieved from the TSF-Buffer, the RTHOS Resident gives command to the DPS Controller-Loader and it starts to loading the requested VHC0 (associated with the first task DFG-statement) from the VHC-Cache to the DPS0 slot. After finishing this operation the RTHOS initiates loading the second VHC1 from the VHC-Cache to the DPS1 slot. When the first and second VHCs are configured in the FPGA the "Data In" and "Data Out" multiplexers in the Data Memory Interface core are switched to the DPS0 slot. After capturing of the first data-frame in the Raw Data Bank0, RTHOS switches multiplexer connected to the Data Frame Acquisition Module (DFAM) to the Raw Data Bank1. Starting from this moment of time the processing cycle starts. While the DFAM captures the next data-frame in the Raw Data Bank1, VHC0 starts processing data from the Raw Data Bank0 and stores the temporal results in the Temporal Data Bank0. After processing of all elements of the data-frame RTHOS switches the multiplexer of Data Memory Interface from the DPS0 to DPS1 slot which means from the VHC0 to VHC1 and "Data In" switches to Temporal Data Bank0 when "Data Out" is connected to Temporal Data Bank1 of the Data Memory. At the same time RTHOS initiates loading the next VHC2 from the VHC-Cache to the DPS0 instead of VHC0. Therefore, the data processing and VHC reconfiguration processes are pipelined. This process continues while all VHCs listed in Task Schedule File (TSF) are completed. In the end of the process the last VHC stores output data in one of available Output Data Banks in the Data Memory and the Data Output Module start transmission of the output data via the Output Interface. Two Output Data Banks are necessary because output data-stream frequency or data rate could be different than input data-stream frequency or data rate. Thus, three parallel processes simultaneously are running on the Platform while TPM works (Figure 3.5): i) Capturing the data-frame # *(i+1),* ii) Processing data-frame # *i* and iii) Transmission data-frame # *(i-1).*

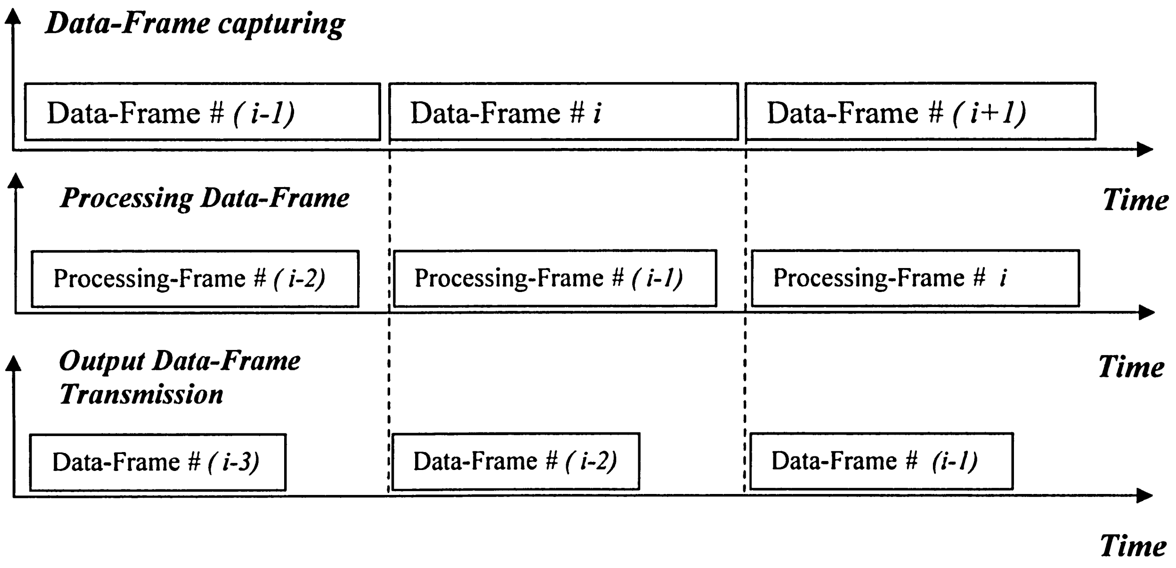


Figure 3.5: Pipelining of three processes: Data-frame capturing, processing using TPM and output data-frame transmission.

Thus, Temporal Partitioning Mechanism allows reaching high performance on the data-frame processing applications and at the same time dramatically reduces FPGA resources needed for the task, which means smaller size and lower cost of the FPGA device and platform itself. The architecture organization of the FPGA based platform with TPM and its cost-effective application for intelligent manufacturing was presented at AMT 2005 (Advanced Manufacturing Technologies) and published in [48]

3.3.2 Temporal Partitioning Mechanism: Reduction of hardware resources

In this section the utilization of hardware resources in the FPGA platform equipped with TPM will be analyzed. As per description of the FPGA platform with TPM the architecture could be broadly divided on three parts: i) Input / Output and Memory Interfacing part, ii) RTHOS and reconfiguration control part and iii) Data processing part.

Let us compare the hardware resources utilization in the proposed approach and regular FPGA platform where complete design is loaded to the FPGA from the beginning.

Obviously, the Input / Output interfaces as well as Memory interface will be the same in both cases because the amount of hardware used for the interface part depends on the type of data-frame sources and receivers as well as type of memory modules. Thus, in general case, there is no reduction of hardware resources for the first part of architecture.

The second part of platform architecture, associated with dynamic control of run-time reconfiguration (RTHOS) does not exist in the regular FPGA platform. That is why this part brings hardware overhead to the platform with TPM comparing with regular FPGA platforms.

On the other hand the Data processing part of platform architecture could give reduction of hardware resources per task if TPM is used. Let us assume that each ***VHCi*** associated with each DFG statement - ***Si*** requires the same amount of hardware resources (logic gates and routing) – ***Ri***, *i= 1,2 ...n*, when it is loaded to the FPGA one after another (TPM-case) or loaded as the complete design (regular FPGA platform). In this case the maximum amount of hardware resources for the data-path - $\boldsymbol{R_{DP}}$ is equal to:

- In case of TPM system needs to have two DPS, where each slot must accommodate VHC with the maximum hardware resources. Thus, $\boldsymbol{R_{DP(TPM)}} = 2 \times max\{\boldsymbol{Ri}\}$
- In case of regular FPGA platform the data-path requires hardware resources close to the sum of hardware resources of each ***VHCi.*** Thus, $\boldsymbol{R_{DP(FPGA)}} = \sum_{i=1}^{n} Ri$

Therefore, the complete amount of hardware resources in case of FPGA platform with TPM will be equal to: $\boldsymbol{R_{TPM}} = \boldsymbol{R_{I/O}} + \boldsymbol{R_{RTHOS}} + 2 \times max\{\boldsymbol{Ri}\}$, when the complete amount of hardware to be used for application processing in the regular FPGA platform is:

$\boldsymbol{R_{FPGA}} = \boldsymbol{R_{I/O}} + \sum_{i=1}^{n} Ri$, where $\boldsymbol{R_{I/O}}$ - is amount of the hardware required for Input / Output and Memory interfacing part of architecture and $\boldsymbol{R_{RTHOS}}$ – is amount of extra hardware needed for the RTHOS and reconfiguration control part.

Let us analyze the reduction of the hardware resources in accordance to the number of Virtual Hardware Components. In case of regular FPGA platform $n = 1$. Thus, in case of FPGA platform with TPM, the number of VHCs – $n > 1$. In case of $n = 2$, there is no reduction of hardware because even if designer can bind the design in two hardware components with pretty equal amount of hardware resources – $\boldsymbol{R_1} = \boldsymbol{R_2} = \boldsymbol{R}$, total amount of resources required for a task will be: $\boldsymbol{R_{TPM}} = \boldsymbol{R_{I/O}} + \boldsymbol{R_{RTHOS}} + 2 \times \boldsymbol{R}$. However, the amount of resources needed for regular FPGA platform with combined parts running simultaneously is equal to:

$$\boldsymbol{R}_{FPGA} = \boldsymbol{R}_{I/O} + \sum_{i=1}^{n} Ri = \boldsymbol{R}_{I/O} + 2 \times \boldsymbol{R} < \boldsymbol{R}_{I/O} + \boldsymbol{R}_{RTHOS} + 2 \times \boldsymbol{R} = \boldsymbol{R}_{TPM}$$

Therefore, TPM can be effective in reduction of hardware resources only in cases when design can be divided on 3 or more components to be loaded sequentially one after another. Furthermore, the utilization of TPM makes system more cost-effective when number of VHCs – n increases. For example, if $n = 4$ and all VHCs require same amount of hardware resources - $\boldsymbol{R}$, regular FPGA platform needs twice more hardware resources for the data processing part:

$$\boldsymbol{R}_{FPGA} = \boldsymbol{R}_{I/O} + \sum_{i=1}^{n} Ri = \boldsymbol{R}_{I/O} + 4 \times \boldsymbol{R} > \boldsymbol{R}_{I/O} + \boldsymbol{R}_{RTHOS} + 2 \times \boldsymbol{R} = \boldsymbol{R}_{TPM}$$

However, the extra hardware needed for RTHOS implementation could mitigate the reduction of total hardware resources. Also, if specification (DFG) allows division of a task algorithm on 10 statements, the reduction in hardware needed for data processing will be 5 times smaller than in regular FPGA platform. The detailed analysis of experimental results regarding reduction of hardware resources when TPM is used will be presented in Chapter 5.

Summary

The proposed FPGA platform with run-time Temporal Partitioning Mechanism (TPM) allows reaching performance in data processing of framed data-streams (e.g. video-frames, communication packages, etc.) in orders of magnitude higher than platforms based on regular microprocessors and processors with instruction processing pipelines. The proposed approach allows dramatic reduction of hardware resources comparing to regular FPGA platforms in most of real cases.

CHAPTER 4

IMPLEMENTATION OF THE FPGA BASED PLATFORM WITH TEMPORAL PARTITIONING MECHANISM

Introduction

In this Chapter the implementation of the FPGA platform with embedded run-time Temporal Partitioning Mechanism (TPM) is described including platform FPGA device selection, platform prototype design and test. Based on the developed and manufactured prototype of the FPGA platform for run-time TPM, the major architectural components implemented in a form of real hardware and on-chip soft-cores were emulated, loaded to the FPGA and tested. Results of the above tests has been analyzed and discussed.

4.1 Selection of the platform FPGA device for the real-time TPM

In order to implement system that would allow running experiments for temporal partitioning of hardware resources it was required to have a specific platform that would have particular characteristics. As it was described in Chapter 3, platform has to be able to have processing capabilities on a FPGA type device, as well as different types of inputs and outputs. However, in order to be able prove the effectiveness of the proposed approach it is necessary to select the appropriate type of the FPGA device that would ideally work for the platform with TPM.

As it is known that there are several manufacturers of FPGA devices such as Xilinx [37], Altera [49], Actel [50], and Lattice [51], each having a wide range of choices on the market. As per description of system architecture provided in the Section 3.3, the concept of embedded TPM dictates necessity of run-time reconfiguration of the part of FPGA micro-architecture reserved for Data processing slots (DPS) without suspension the rest of running soft-cores (e.g. RTHOS, I/O Interfaces, etc.). The only FPGA device family which allows the requested features is Xilinx Virtex families of FPGA devices [34]. However, Xilinx Virtex devices are complex high-end and thus, relatively expensive. The recent Virtex II and Virtex II Pro families of these FPGAs are available only in ball-grid (BGA) packages. This makes very difficult to manufacture and rework these devices in case of damage or prototype board defects. Thus, it seemed more suitable to divide implementation process in four separate parts (stages):

i) Development and test the hardware components of the proposed architecture using inexpensive FPGA device without partial reconfiguration but similar in its architecture to Xilinx Virtex II FPGA. The best candidate for Virtex FPGA replacement was Xilinx Spartan-3 FPGA. Thus, the first prototype of the test-platform was done on this FPGA device.

ii) Development of soft-cores of on-chip architectural components (e.g. RTHOS resident, I/O and Memory interfaces, etc.) and further simulation and test each of them on the first prototype of the test-platform. The soft-cores have to be developed on VHDL using Xilinx ISE Foundation CAD. This will allow easy migration of these cores from Spartan-3 to Virtex II / II Pro FPGAs.

iii) Development of Windows Agent for communication with the platform to perform several tasks such as: loading cores from a PC local drive to platform's FLASH, core readback and verification, scheduling core loading into the FPGA, maintenance of existing cores, and general system check.

iv) Integration of the above into complete prototype based on Xilinx Virtex II FPGA, emulation of complete system and test. After this stage all necessary experiments and real-time task execution could be done and performance parameters could be collected for further analysis.

4.2. Development and test the hardware architectural components

As per FPGA device selection in the Section 4.1, Xilinx Spartan 3 XC3S400 FPGA device [52] with 400,000 system gates was selected for the first platform prototype. Xilinx Spartan-3 FPGA is closely compatible with the high level FPGA devices such as Virtex II / II Pro and further Xilinx Virtex-4 families, and if needed all soft-cores developed for Spartan-3 could be easily implemented in these high end FPGAs.

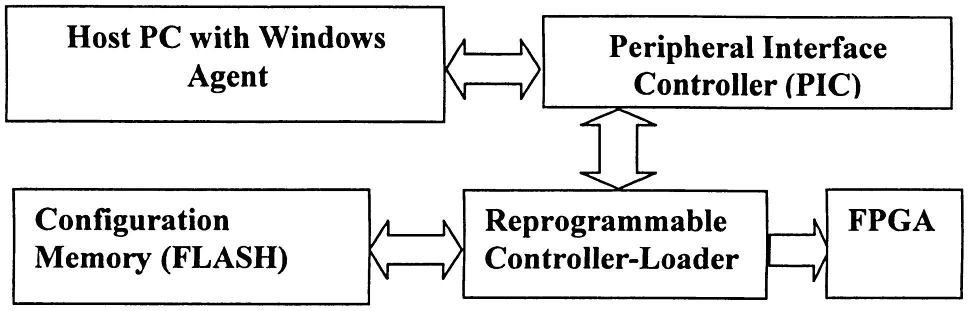
On the other hand, an essential function of the system was to be able to provide temporal partitioning capability and that would encompass reconfiguration of the FPGA depending on the particular processing core required at that instant. For achieving such a result there are several development platforms that are available on the market [53], [54], [55], however the main drawback of these platforms is that they either have a capability of reconfiguring FPGA over the communication interface such as JTAG from a PC, or reconfiguration of FPGA of only one core from the on onboard EEPROM over the 8-bit configuration bus. The main drawback to all of those systems is that it is impossible to provide constant dynamic reconfiguration of the FPGA devices, since a new configuration file or *core* has to be loaded from a PC, which is relatively slow process.

This particular requirement forced to be able to store all of the required VHC soft-cores on board and be able to load them as fast as possible into FPGA. As per proposed in the Section 3.3 architectural solution, the Soft-Core Memory has to be implemented in an on-board FLASH memory device. Nowadays the volume of FLASH memory devices increased up to 8 Gigabits per chip and would be able to store up to 4818 complete FPGA cores of Spartan-3 with 400,000 system gates. Programming of such FLASH memory module can be done off-line (before the operation of the platform) to store all necessary VHCs and all possible their variants. As well as, it can be loaded during the operation of the FPGA, at the time when FLASH memory is not being accessed for FPGA reconfiguration.

At the same time loading of the VHC-cores onto a FLASH memory, as well as from flash to FPGA has to be conducted under control of Reprogrammable Controller-Loader (Section 3.3). There are several possibilities of implementation of this kind of controller, some of which include software and hardware solutions. In case of software solutions we are again required to be linked to the PC and this will slow down operation of the embedded system. In addition, it will not allow having a stand alone embedded platform, which is necessary in lots of applications (i.e. remote and space applications) and also is necessary for real-time performance analysis. Therefore, it was required to have an on-board controller-loader. This device would manage number of functions. It would be responsible for communication between the PC and FLASH memory, as well as it would act as a scheduler which will load certain cores in sequence if it is needed to be done, and it would have the knowledge about the types of cores and their locations on

FLASH memory. Thus, it should perform functions of on-chip Configuration – Controller of DPS (Section 3.3.1)

Therefore after analyzing initial requirements of the platform the general setup of the platform looked like it is shown in Figure 4.1.



Host PC with Windows Agent
Peripheral Interface Controller (PIC)
Configuration Memory (FLASH)
Reprogrammable Controller-Loader
FPGA

Figure 4.1: Platform organization – reconfiguration component

For the choice of the controller there are several possibilities that could have been selected. However, due to the requirements it was needed to have a controller with non-volatile program memory, and at the same time fast enough to be able to manage communication and file exchange processes in real-time with range of 20-50 ns. These requirements narrowed down the possibility to Reduced Instruction Set Controllers (RISC-controllers) or Complex Programmable Logic Devices (CPLD). Each having its own disadvantages caused to reconsider structure of the system. Problem with EEPROM based CPLDs was that it is needed to write all the communication drivers for communicating with PC which would require a lot of logic resources. Problem with the RISC controllers is that they usually are operating at about 1-5 MIPS (Mega Instructions per Second) and will not be able to provide maximum transfer rate from FLASH memory to the FPGA device during loading VHC soft-cores. Therefore, RISC controller may slow down reprogramming at least by 10 times, since programming speed for Spartan 3 is about 50 MB/s . Another problem is that a typical microcontroller has about 30 I/O pins[56], which clearly would not be enough for communication with PC, FLASH, and FPGA in parallel mode. In order to solve this problem it would be good to combine two components RISC-controller and CPLD to take advantages of each of them. CPLD can provide a lot of I/O lines (up to 130 configurable I/O channels) and will be able to communicate in real time with FLASH, RISC-controller and the FPGA. CPLD also could provide custom interface to other components and operate at relatively high frequencies (up to 250 MHz). Therefore, CPLD can reconfigure the FPGA at the maximum allowable rate. At the same time controller will be able to communicate with PC using embedded communication interfaces (e.g. UART) and also be able to coordinate and schedule loading process of the appropriate VHC-cores, since it has enough internal memory to store the index table as well. Development of the firmware for both CPLD and RISC controller was slightly complicated due to the fact that development had to be done in a manner that will allow both communications with each other and at the same time not interfere with their operation. For that purpose the Microchip Peripheral Interface Controller (PIC) was selected because in addition to RISC architecture and 10 MIPS performance it is equipped with many embedded hardware cores including most of communication subsystems (UART, I2C, SPI, Byte-parallel and PWM). Then, a special PIC-CPLD protocol was designed and communication was done in SPI (Serial Peripheral Interface) type protocol. Since the platform embeds many different components and design needs to be tested before complete integration, the development cycle was divided on number of stages:

First stage was to develop and build an embedded system which would perform storage, management and loading of FPGA cores. This system was required to have:

i) Communication interface with PC over RS232 / USB using USART module in the RISC controller (PIC)

ii) Communication interface between RISC controller and CPLD as well as between CPLD and FLASH memory to be able to access and save transmitted cores and load them onto the Spartan 3 FPGA device.

Also, it was needed to develop an Agent application for managing cores on the PC side and communication protocol for data exchange with the developed embedded system.

The block diagram of the first prototype of the platform with Temporal Partitioning Mechanism is shown in Figure 4.2. In Figure 4.3 the image of the actual board of this prototype is displayed.

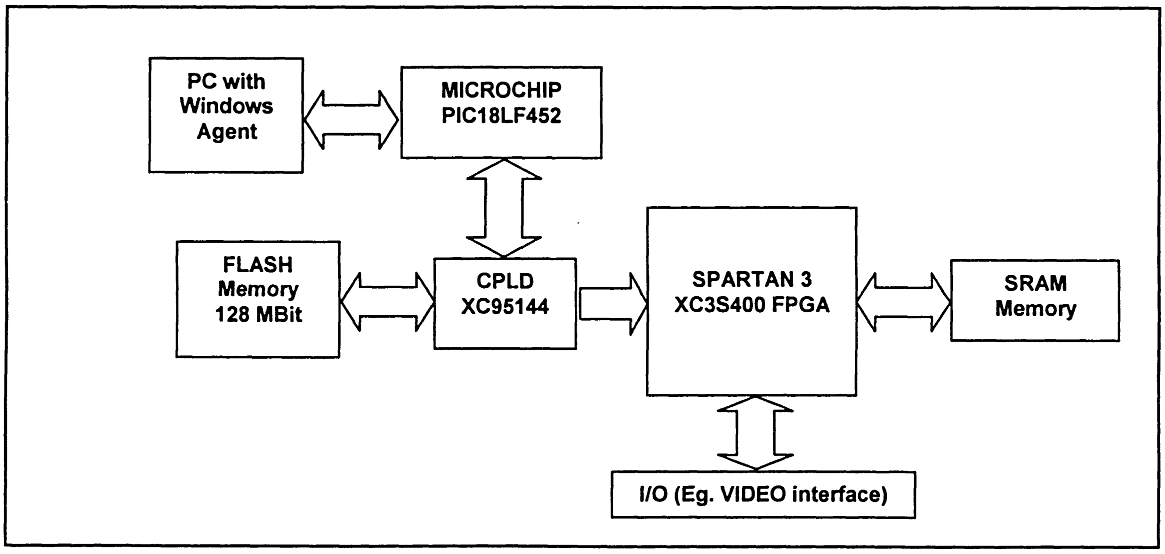


Figure 4.2: Block diagram of the first prototype of the platform with TPM

Even thought the first stage of the development of this platform was successful, after running several tests and experiments a problem was identified, which required for slight redesign of the loading mechanism, which was included in the second stage of development. It was found that even though the FLASH memory was outputting data at a rate of 8 bits per every 25ns, but it would have to wait after every 527 bytes for 50us in order to address the correct block of memory and copy it to its local SRAM register. This is 2000 times longer then regular period and creates a substantial delay of the system reconfiguration. Due to that fact maximum loading time for Spartan 3 XC3S400 FPGA device was limited to:

207 KB (size of configuration bit-stream) / 527 B * (25 ns * 527 + 50 us) = 25.41 ms per core.

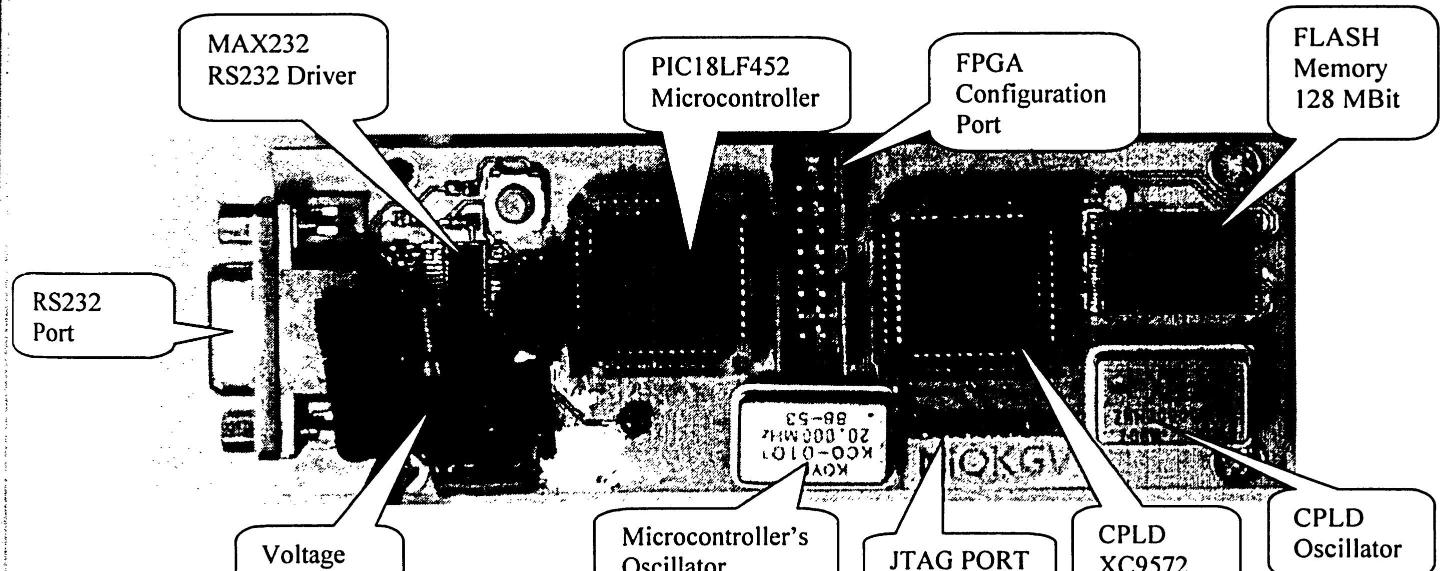


Figure 4.3: Evaluation board of the first prototype of the FPGA platform with TPM.

However, the maximum speed at which Spartan can be configured is: 207 KB (size of configuration bit-stream for XC3S400) / 50 MB /s (speed of configuration) = 4.2 ms per core. This is more than 6 times faster, and that is considering that the FLASH was able to access the right block in 50 us and did not cause more delay. To solve this problem I have decided to create a buffer SRAM memory, which would be preloaded with the next required core in 25.41ms or longer. At the time when new core has to be loaded into the FPGA device it will be able to output data at a rate of up to 12 ns, therefore, it would be able to surpass the maximum speed of FPGA configuration, and will not cause stalls in configuration due to the addressing and timing issues.

As a second stage of implementation was to integrate the first stage with the additional SRAM module (buffer), as described previously and also add FPGA. At this stage requirement was to be able to fill the FLASH with library of cores, and set a reprogramming sequence on the RISC-controller side to direct particular core to SRAM block and eventually to the FPGA.

4.3 Hardware design of the platform prototype

This section will present the description of hardware design of the first prototype of FPGA platform with TPM. As it was mentioned in the Section 4.2, scheduling and indexing of FPGA cores had to be done using a RISC-controller – Microchip PIC families. The Microchip PIC18F452 was selected in a 44-pin PLCC package. It is a 16-bit (instruction) pipelined RISC-processor with Harvard architecture, running at maximum clock speed of 40MHz, and has 32K or internal instructions EEPROM (the biggest in family), which is enough to store all of the cores indexes [56]. It also has

UART interface for possible serial interface to the PC and number of built-in protocols for communication with other peripherals, such as I^2C and SPI to communicate with CPLD. This would greatly decrease the development time since it has almost all required protocols and embedded device drivers in it.

Initially selected CPLD was Xilinx XC9572XL in a 44-pin PLCC package since it had to be a bridge between RISC-controller, FLASH memory, and Xilinx FPGA device. The amount of logic gates and Flip-Flops was quite sufficient for development of communication between all of above devices. However, the number of I/Os was enough only for the first stage. At a point when second stage was designed and additional SRAM module had to be added. At the initial schematic it was clear that 44 I/O pins are not sufficient since only one SRAM module required 16 data pins, 17 address pins, and 5 control pins; therefore a larger CPLD package was chosen to be XC95144XL with 117 I/O pins and twice more logic gates.

Regarding the FLASH memory which would contain FPGA cores, a Toshiba 128Mbit TC58DVM72A1FT1 was selected since its communication protocol and package was compatible with larger memory sizes.

As it was stated before, for the choice of FPGA, the SPARTAN-3 XC3S400 in a 208 PQFP (Plastic Quad Flat Pack) IC package was selected, since it was the largest one before the BGA (Ball Grid Array) packages. Besides the main components there were peripheral components such as oscillators, voltage regulators and indication LEDs, etc. For this particular setup it was needed to have several well stabilized voltages since just FPGA alone required 3.3 V, 2.5 V and 1.2 V supplies. When all of the components were chosen general design of schematic had to be done according to the block diagram presented above. All of the schematics of all the boards that were made through this project was done on Cadence OrCAD Schematic 9.2. Printed circuit board (PCB) layout was made using Cadence OrCAD Layout Plus 9.2 design tools which was advantageous since number of components were provided as schematic and footprint symbols. As the schematic was done verified (See Appendix A) with datasheets of all the components it was possible to begin routing process. However, there were several factors that had to be carefully considered during routing, which otherwise could become a huge obstacle in further work such as laying thick power lines, as well as routing out extra test points for easier debugging. Especially important was the routing of the FPGA which would not operate correctly in case of incorrect placements or complete omittance of decoupling capacitors. The hardware setup of the first platform prototype is shown in Figure 4.4.

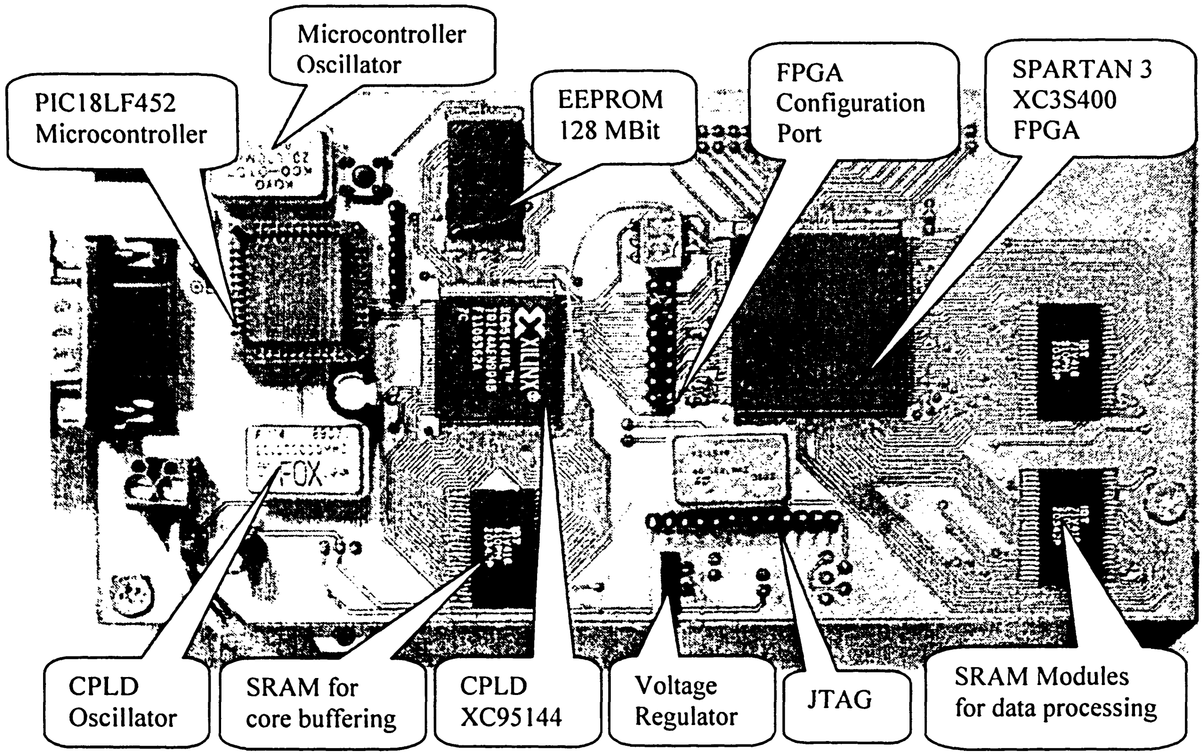


Figure 4.4: First prototype of FPGA platform with Temporal Partitioning Mechanism

Due to the fact that this platform is an experimental one, it was important to choose types o parts that are easiest for removal and replacement and also pin compatible to other parts with similar functionality. Example of this is PIC-controller which is pin compatible not only to the ones in its family but also to PIC16F8XX family (dew to it's widely use of 44- PLCC package). At the same token CPLD can be increased in its logic cells from 144 to 288 since the package and pins are compatible as well [57].

4.4 Development of soft-core interfaces.

In this section there will be a detailed explanation about the development of firmware and soft-cores for FPGA platform with Temporal Partitioning Mechanism. Development had to be done in several stages since every component of the system required separate testing and integration. In case that component was not suited for the system it could be replaced without need of whole system redesign. As it was described before for the first prototype of the system it was needed to establish communication between the PC and the Peripheral Interface Controller (PIC). This would allow performing the loading of FLASH memory with soft-cores configuration bit-streams, control of the CPLD and therefore execution of FPGA reconfiguration. Serial communication interface was the most attractive solution since it was supported by the PIC microcontroller and also serial port function exists in the MS Visual Studio CAD environment which allowed access to Windows operating system. In this case high bandwidth of the communication channel was not an issue because FLASH memory loading could be done before real-time data processing. However, in case of higher speed requirements a USB interface would be more preferable and in order to account for this possibility a test development board was designed using and FTDI 245 USB interface chip [58]. After manufacturing of the prototype board, tests has shown that upload speed is about 8 Mb / s, however this could be needed in case of use of large number of cores or much bigger FLASH memory. A set of commands was developed for identifying what kind of data after an identification flag byte was designated to perform. List of commands is shown in Table 4.1.

Table 4.1: List of Commands

Command	**Action**	**Additional parameter(s)**
01110111	Write a core into a specific FLASH memory location	Location 0-63
01110010	Read a core from a specific FLASH memory location	Location 0-63
01010111	Write an array of 528 bytes into a specific FLASH address	3 address bytes
01010010	Read an array of 528 bytes from a specific FLASH address	3 address bytes
01100101	Erase a core from a specified location in FLASH	Location 0-63
01000101	Erase a specific page of FLASH	2 address bytes
01000110	Full FLASH erase	N/A
01110000	Program core from the FLASH location into the FPGA	Location 0-63
Additional Commands after merging with Virtex II Pro platform		
01101100	Load core from specific FLASH location to SRAM module	Location 0-63
01110011	Read back SRAM module contents to PC for core verification	
01011000	Program FPGA from the SRAM module	

After the reception and decoding of the command the PIC-controller performs the associated operation either by communicating with the FPGA or with the CPLD.

In order to communicate with a CPLD a protocol had to be written and there were several options to choose from. However, due to a fact that serial communication with a PC creates a transfer bottleneck it was decided that SPI (Serial Peripheral Interface) protocol will be sufficient solution. That would save another 7 I/O pins without loosing any bandwidth, since with a clock speed of 20 MHZ it was possible to transmit information at a rate of 20 Mb/s where maximum possible speed of RS232 communication was 115 Kb/s. For the SPI protocol three lines were used: Data, Transmission enable, and Clock. Timing diagram for PIC-CPLD communication is shown in Figure 4.5.

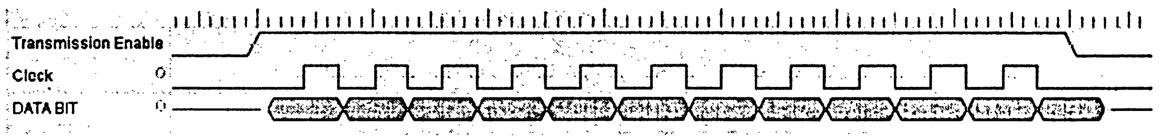


Figure 4.5: Timing diagram of the PIC-CPLD SPI communication protocol

In case of communication with the FLASH memory it was required to be closely compliant with the datasheet [36]. Therefore a controlling mechanism had to be written for the CPLD to make sure it can perform writes and reads with maximum allowable speed. As it was stated before for this particular platform a 128Mbit FLASH was used and it was decided to divide it into 64 units for storage of soft-cores of VHCs. Since all of the FLASH chips are compatible by communication interfaces it is possible to expand the size of memory simply by replacing the FLASH chip. The reason for division on equal core units was due to the fact that they easily can be accessed from the microcontroller since their positions are just the increments of top 6 bits of the address and therefore can be directly accessed without keeping the extensive table of core location addresses. Since the microcontroller itself has 1024 bytes of EEPROM memory it is possible to store ID numbers of the cores in first 64 locations so maintenance of cores could be left onto microcontroller and just an ID number has to be supplied from PC to erase, read back, or write FLASH, as well as reconfiguring the FPGA. In addition to the communication with the FLASH it was needed to make an interface to the SRAM IDT71v416 chip [59], which on its own has 39 pins connected to the CPLD and was required to be an FPGA core buffer. As it was mentioned before EEPROM had one disadvantage, which was the 50us delay between every 528 bytes which would significantly decrease the speed of reconfiguration. Therefore another addition had to be made in CPLD firmware to be able to integrate the communication protocol with SRAM. Closely following the timing from SRAM datasheet [59] it was possible to achieve the maximum speed of load from FLASH to SRAM, and from SRAM further to the FPGA and replicated port. However, there were several issues related to the SRAM load. Due to the fact that the memory was operating at a high speed of around 50MHz during the transmission, some random errors were introduced which did not resemble any pattern. As it was found out this was occurring mostly during the transition of all the lines from all ones to zeros. Control lines even though being separated with the "Ground" were triggered to glitch and thus either register an incorrect data, or even restart the whole programming process. In order to overcome this problem it was decided to switch data on data line not all at once but only half of them at a time first doing odd and then even bits.

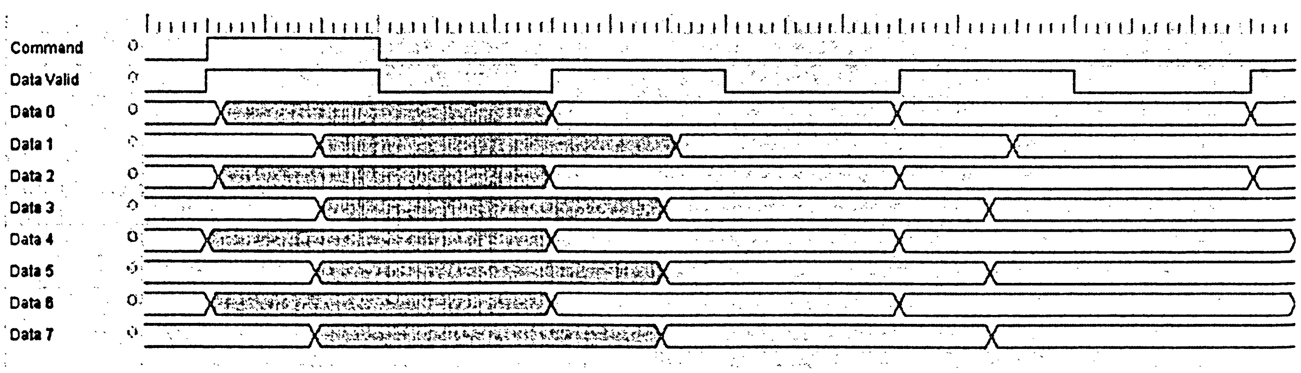


Figure 4.6: Timing diagram scheme for avoiding coupling glitches on SRAM

Since, SRAM memory is rated to have a response time of 7 ns it was enough time to have data smoothly placed on the data lines without causing any glitches on the control lines, since requirement for the FPGA reconfiguration was 20ns per byte. Modified data writing protocol and SRAM access timing diagram is displayed in the Figure 4.6.

As this problem was overcome, only FPGA reconfiguration procedure had to be implemented, which is described in the flow chart shown in Figure 4.7.

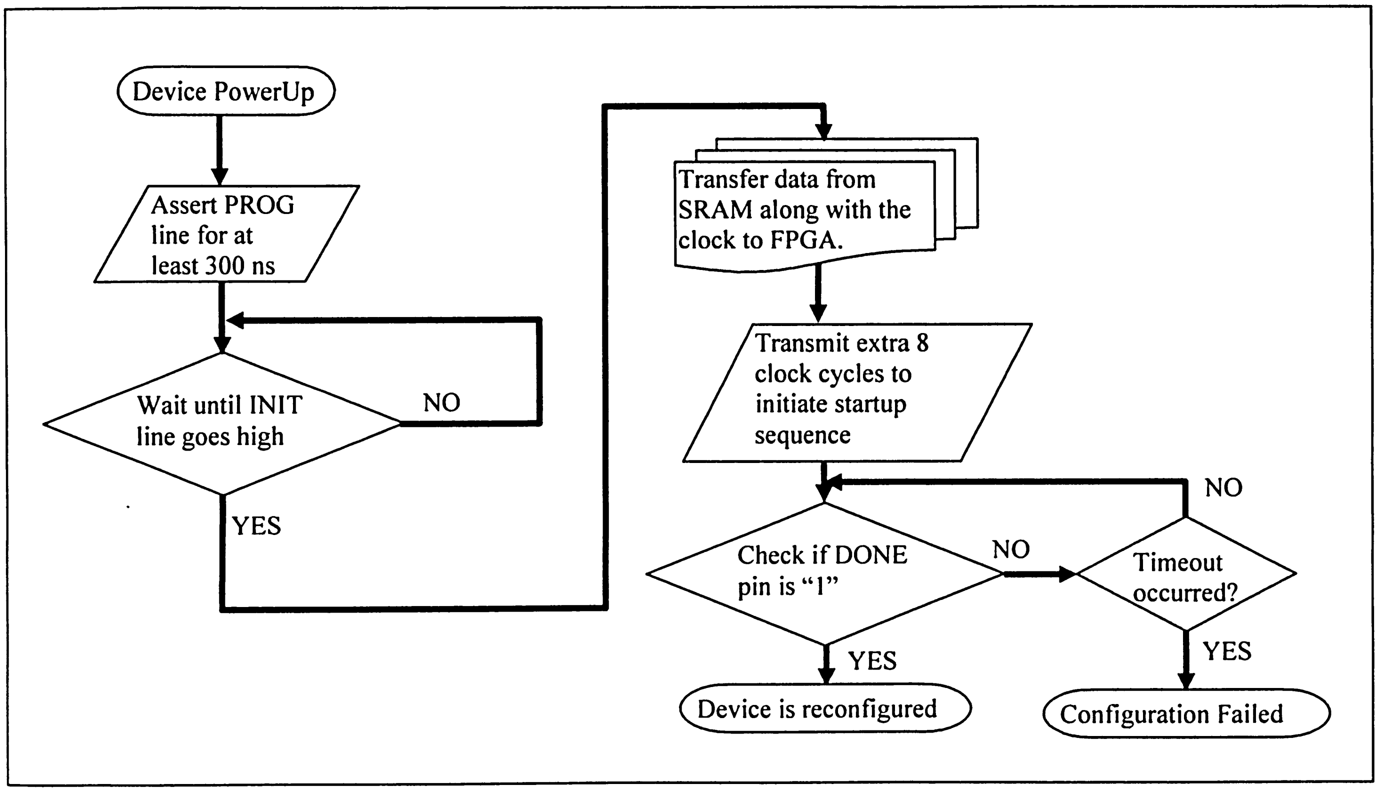

Device PowerUp
Assert PROG line for at least 300 ns
Wait until INIT line goes high
NO
YES
Transfer data from SRAM along with the clock to FPGA.
Transmit extra 8 clock cycles to initiate startup sequence
NO
Check if DONE pin is "1"
NO
Timeout occurred?
YES
YES
Device is reconfigured
Configuration Failed

Figure 4.7: Flow chart of the FPGA configuration process

4.5 Windows Agent development

One of the essential parts of this project was the ability to manage and upload cores to the platform from the PC and the rest of the operation is left up to the microcontroller. However, PC side would be capable to collect information, upload new cores, and change the core reload schedule. For this purpose an application had to be build since simple simulation through hyper terminal is not sufficient for this purpose. There were several choices of either to make it based on MATLAB [60] GUI generator and use it's built in serial port communication protocol, or choose more traditional approach of Visual Basic or Visual C++/C#. MATLAB GUI generator was an attractive option due to its ease of programming, however, after implementing one of the first prototypes it was found that even though graphics application operates quite stably, when using communication port such as serial it causes halts and application "freezes". One of possible explanation is that it is based on Java virtual machine, and a serial driver included with it is not stable for large uploads. As a second option it was decided to develop an application in Microsoft Visual Studio, which proved to be a success since it was very stable in working with the serial driver that was included with it. An application in itself has two parts to it: GUI interface, and core manager [61]. Core manager keeps track which cores were loaded into which locations of the FLASH memory as well as assigning the ID numbers, which it exchanges with the microcontroller as it was mentioned previously in the Section 4.5. As stated before, UART protocol was used for communication between PC and the platform. That is done because MS Visual Studio embeds UART drivers for COM ports and also applicable for the USB interface. Since, USB serial driver is included in the package of the FTDI 245 USB chip it will not require any modifications in programming.. Actual GUI interface is displayed in the Figure 4.8 and has several features.

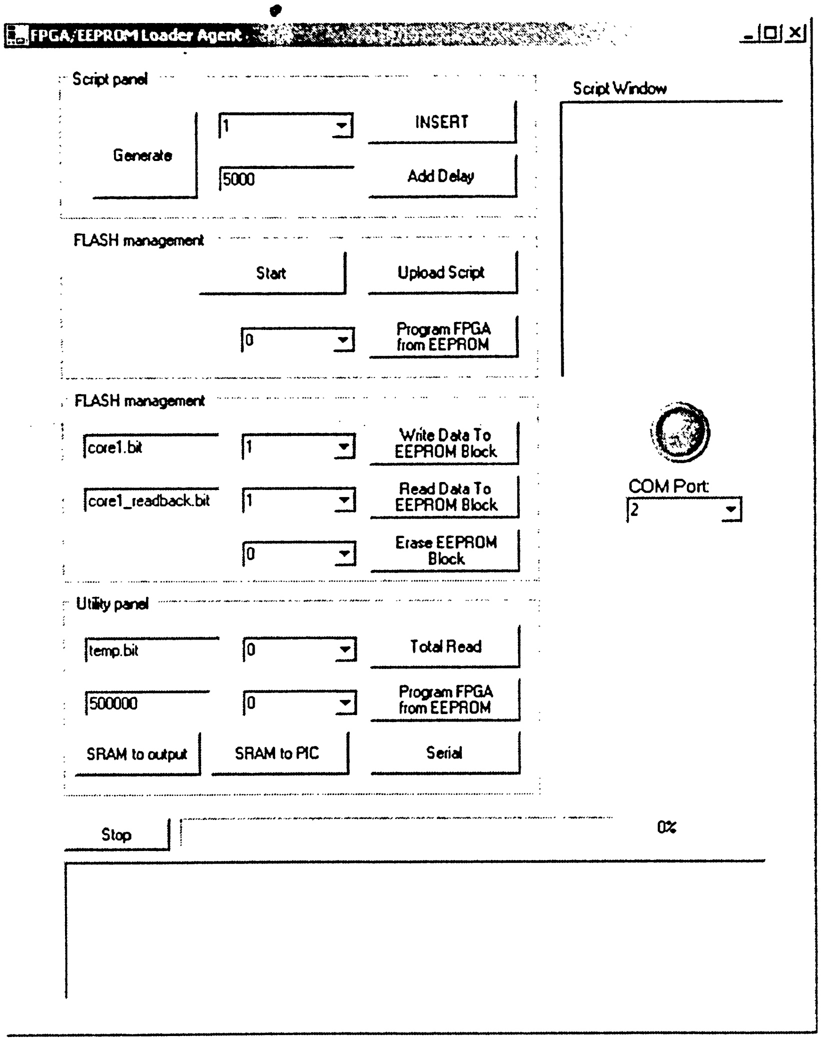


Figure 4.8: Graphical User Interface for PC Windows Agent

Control section of the application allows the user selection of the COM port and baud rate at which system should be operated. On a short distance (~2-3 m), with well shielded cable it is possible to get up to 115200 baud and operate without problems, however, if distance is longer, baud rate has to be much lower otherwise it misses some bytes. Second set of controls relates to actual core management. First option that exists is to either browse file to be sent to the specified location in the FLASH or directly specify the file in the text window. Second control is responsible for downloading the core to the file on the hard drive, similarly, it can be either browsed or a filename can be specified. Similarly as it was described in the Table 4.1, it has a capability of erasing specific UNIT in FLASH. The rest of controls are related to the programming of the FPGA. There are essentially two options: it can either be configured using the immediate reconfiguration by sending command that points from which FLASH location it has to be reconfigured, or a script can be written by selecting the desired core and inserting with an *INSERT* button (on the PC keyboard) into the list. A delay in microseconds is specified is inserted in the same manner, and will indicate how long microcontroller will wait until reloading the next core. Essentially list with core locations and delays is uploaded to the microcontroller and at the instant as the list is uploaded microcontroller begins to execute the list. There is also option of the loop and operation can be interrupted any moment, since, RS232 communication will interrupt the operation of reconfiguration and based on the request could either terminate the process, or reload a new sequence. As an addition it is also possible to get current status of the platform by requesting it from the microcontroller and displaying it in the status window.

In case of "freeze" up for example, agent will specify actions to be performed to restart it or to recover it. Also in case of power failure it is possible to record the log of what happened and when, since microcontroller can scene that voltage has fallen bellow 3.3V and have enough time to record it. This is possible since PIC18LF452 only stops its operation at 1.5-1.7 V, which was proven by several experiments.

4.6 Integration of the platform on to Xilinx Virtex II FPGA platform.

As a last stage of prototyping it became possible to integrate the all above developed and tested components onto a FPGA platform based on partially reconfigurable Xilinx Virtex II FPGA that was developed in the Embedded Reconfigurable Systems Lab as the uniform evaluation system with ability for partial reconfiguration. The platform hardware consists of: Xilinx Virtex II XC2V1000 FPGA with one million system gates, as well as four SRAM modules with 7 nanosecond access time and total capacity of 4 MB. The Controller-Loader is built on the Xilinx XC95144XL CPLD. The platform is displayed in the Figure 4.9.

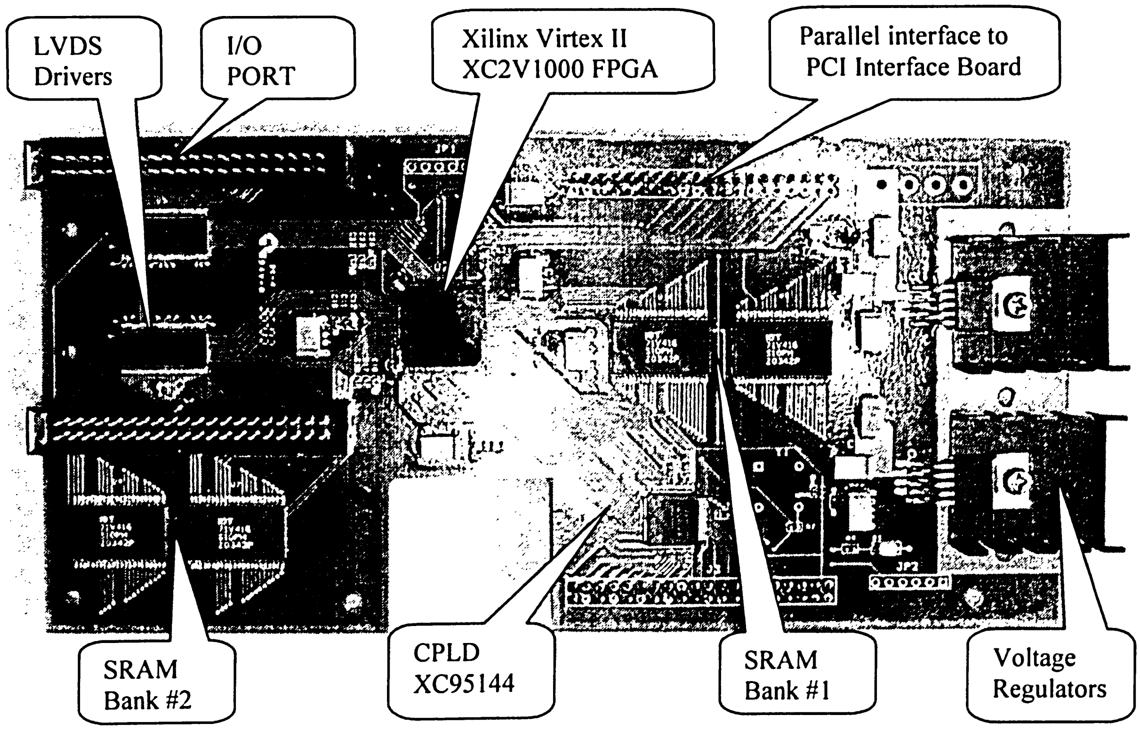


Figure 4.9: Virtex II FPGA based platform with ability for partial reconfiguration

The advantage of this system is ability for spatial and temporal partitioning of the application in the on-board FPGA using partial reconfiguration of the FPGA device [62]. In addition to that it is possible to interface this platform to a PCI-bus since one of the implemented tasks of that platform was connectivity to a PCI passed platform with a PLX 5030 PCI BUS controller [63]. In order to be able to interface the existing platform developed in the stage 2 it was required to have a communication channel from the CPLD which would be able to transmit new core to the FPGA for the its partial reconfiguration. For that purpose the top I/O communication port was used and communication was organized in 8-bit channel with a transmission enable and strobe signal. On the CPLD side a configuration data bus was used to communicate with the Virtex II platform since it was not used for Spartan 3 configuration and had a special extension header reserved on the Spartan 3 platform. Data was routed directly to FPGA and then could be used for various tasks. First of all a communication protocol with the command structure had to be established closely related to the one designed for the CPLD, which required designing a monitor on the Virtex II FPGA. FPGA conducts tasks monitoring, data-frame reception and storage onto a SRAM module, read back of data from the SRAM onto the microcontroller directly, and the most important one is reconfiguration of the other part of FPGA. As it was mentioned in Chapter 2 it is possible to reconfigure the Virtex FPGA using an internal built-in module called ICAP (Internal Configuration Access Port)[62].It resides in the lower right hand side of the FPGA and can be easily accessed using modular design in Xilinx ISE CAD system. Therefore, for the design of FPGA organization it was decided that whole right side of the FPGA including the ICAP module will be dedicated for the controller and will not be changed except from the time of configuration at the beginning of operation. In addition to that, for verification purposes it was needed to have direct access to the data that was uploaded to SRAM prior to partial reconfiguration of FPGA, and it was decided to use some of the remaining I/O pins of the FPGA to upload data back to the microcontroller and in turn follow it by the upload to a PC Windows Agent. Read back from SRAM ensured that the transmission of the data to the FPGA will be correct and configuration will not cause FPGA to malfunction. Since read back of the data from SRAM was not time critical it was possible to use microcontroller to perform this operation and significantly decrease the coding complexity for CPLD. Flowchart of sample operation of the platform is displayed bellow in Figure 4.10.

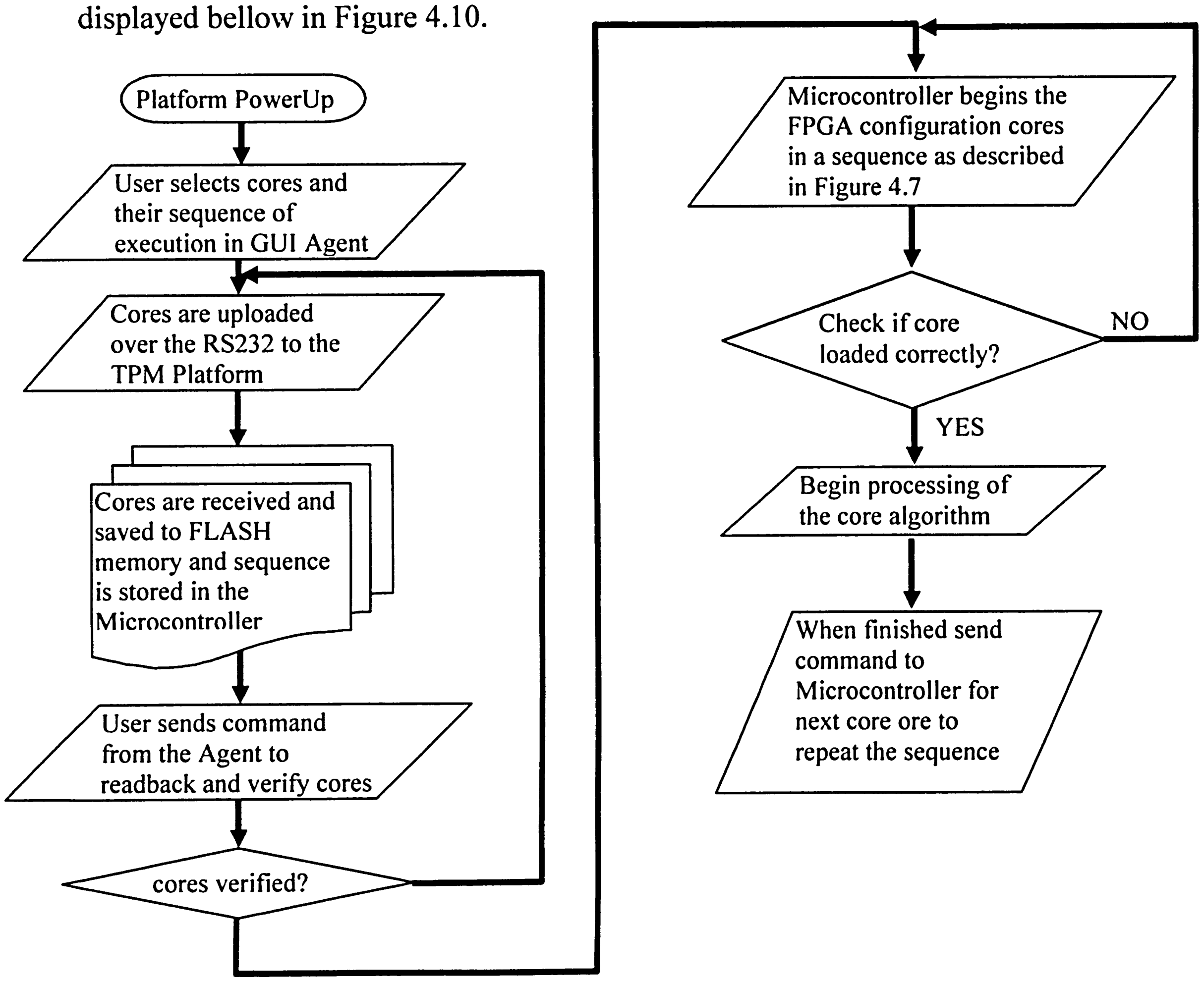

Platform PowerUp
User selects cores and their sequence of execution in GUI Agent
Cores are uploaded over the RS232 to the TPM Platform
Cores are received and saved to FLASH memory and sequence is stored in the Microcontroller
User sends command from the Agent to readback and verify cores
cores verified?
Microcontroller begins the FPGA configuration cores in a sequence as described in Figure 4.7
Check if core loaded correctly?
NO
YES
Begin processing of the core algorithm
When finished send command to Microcontroller for next core ore to repeat the sequence

Figure 4.10: Flow chart for sample platform operation

Summary

In conclusion it can be said that the development of the project went through several important stages that consisted of design of the hardware components such as PCBs schematics and layout. It also required development of first stage platform which allowed to test and design interfaces to such peripherals as PIC microcontroller, FLASH memory module, and CPLD which served as a bridge between the PIC microcontroller, Flash, and target FPGA. In addition, soft-cores had to be developed for the CPLD in order for communication with FLASH and other peripherals, along with Firmware which was developed for PIC18LF452 Microcontroller to communicate with the PC over the RS232 interface and the CPLD. On the PC side a Windows Agent had to be developed in order to give user capability of communicating with the platform and manage the created cores on the FLASH memory, as well as acquire the status of the platform. It was done using the Microsoft Visual Studio and had a GUI interface for simplification of user interaction. At last, integration of the second stage platform onto the Virtex II platform, as well as, implementation of mechanism for partial reconfiguration on the Virtex II FPGA.

CHAPTER 5

EXPERIMENTS AND RESULTS

Introduction

This Chapter will describe the experiments that were run on the actual FPGA platform with TPM with comparison to conventional platforms based on RISC microcontroller and PC with Superscalar architecture. All of the obtained results were scaled down to the same frequency of operation in order to have a reasonable comparison of the platforms. In case of FPGA platform with TPM, the XC2V1000 FPGA (1 Million system gates) was used. The microcontroller platform was based on RISC controller similar to the chip which was used on the stage 2 platform (see Chapter 4). For the experiments on the PC an AMD 3000+ based system was used with 1 GB of SDRAM. All the testing algorithms were implemented on a Visual C# .NET development studio. For testing the data-frame processing procedures, the image enhancement using the Laplacian algorithm was implemented as well as Sobel Edge Detection algorithm which was implemented using matrix manipulation over nine neighboring data points. At last, a histogram process was implemented to collect the statistical data about the video image. Most of those algorithms are used in the application of video navigation as well as surveillance and unmanned autonomous robot operations.

Experiments on each of the platforms with the analysis of results are described in the following sections.

5.1 Implemented Algorithms

In the following subsections each of the implemented and tested algorithms is briefly described.

5.1.1 Laplacian Image enhancement algorithm

Laplacian image enhancing algorithm is commonly used filter for sharpening the image at question. Generally Laplacian of an image $f(x,y)$, is denoted as second derivative of an image $\nabla^2 f(x,y)$

Where $\nabla^2 f(x,y) = \dfrac{\partial^2 f(x,y)}{\partial x^2} + \dfrac{\partial^2 f(x,y)}{\partial y^2}$ {1}

This equation can be approximated to

$$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y) \qquad \{2\}$$

This expression is directly translated into a 3x3 matrix with coefficients:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

If diagonal elements will be considered as well

$$\nabla^2 f = \begin{bmatrix} f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) + f(x+1,y+1) + \\ f(x-1,y+1) + f(x+1,y-1) + f(x-1,y-1) \end{bmatrix} - 8f(x,y) \quad \{3\}$$

which is translated into the following matrix:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In order to use Laplacian enhancement algorithm the original image has to be added to the filtered, which would generate the enhanced image: $g(x,y) = f(x,y) + c[\nabla^2 f(x,y)]$, Where $f(x,y)$ is original image, $\nabla^2 f(x,y)$ is filtered image, and *c* is coefficient $+1$ or -1, depending if mask is positive or negative.

5.1.2 Sobel Edge detection algorithm

The Sobel operator performs a 2-D spatial gradient measurement on an image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel algorithm operator uses a pair of 3x3 convolution masks, one which performs it in vertical and second in horizontal direction. Matrices are shown bellow:

$$\text{Horizontal}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \text{Vertical}\begin{bmatrix} +1 & 2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

To get edges in both direction and get a resulting image it is required to add the results using following formula: $g(x,y) = \sqrt{h^2(x,y) + v^2(x,y)}$

5.1.3 Image Histogram Statistic Algorithm

In order to get a histogram of the original image it is required to count number of points that fall under each of the grayscale levels and than represent it in the graphical form similar to the one in Figure 5.5 d). In some cases histogram is divided on much course levels such as only 10 and it is done by summation of all the points that fall under $\frac{256}{10} i$ levels.

5.2 Experimentation on the FPGA platform with TPM

In order to obtain proper measurement of the TPM performance it was decided to use at least three different processing procedures that can not correlate to each other and therefore would require separate implementation in silicon (e.g. ASIC). On top of that, a data loading / output cores had to be implemented and used as one of the separate tasks. It should perform the loading of the video-image data into the memory from the video capturing board connected to one set of I/Os according to TPM Block-diagram presented in Figure 3.4 in the Section 3.3. Similarly, when video-data processing is completed, a core would take care of data output onto the display board. Therefore, five separate non-correlating cores had to be developed for the FPGA platform with TPM and loaded one after another using the TPM. Diagram of the setup of the FPGA platform with TPM and all links to the peripheral boards is presented in Figure 5.1.

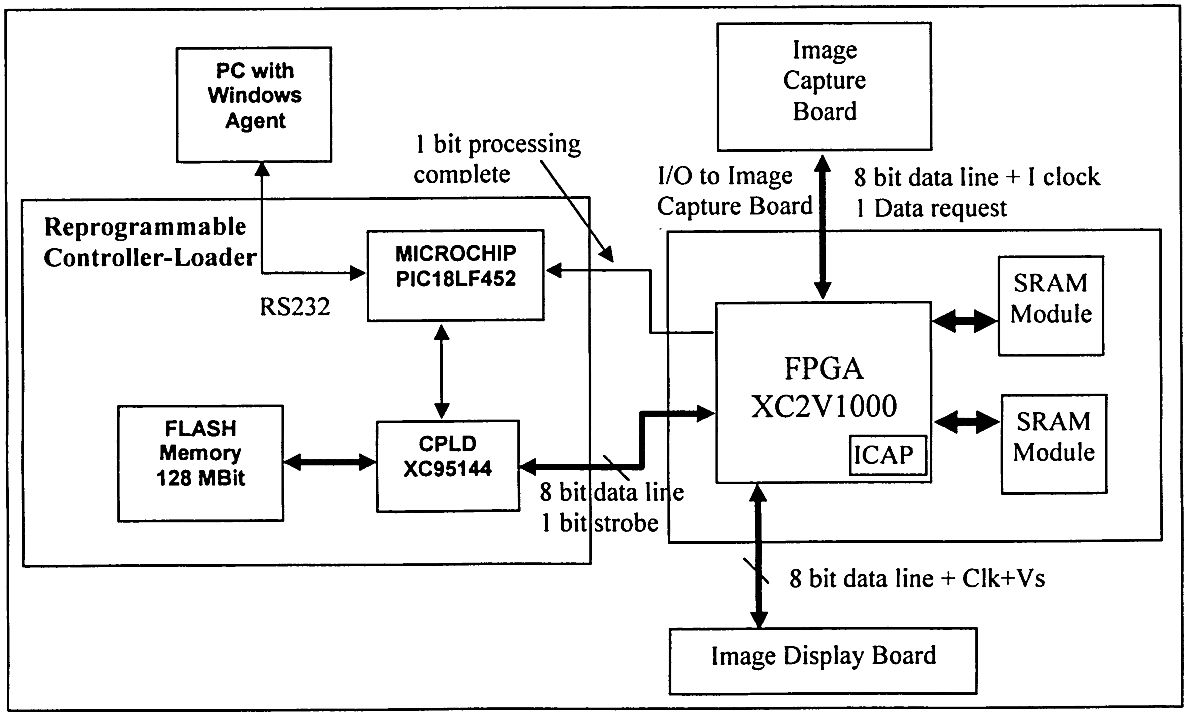


Figure 5.1: Setup of the FPGA platform with TPM and links to the peripheral boards

As it can be seen from the diagram, the platform consisted of microcontroller based sub-system that performs core management and transmission to the Virtex II FPGA device and also communicates with the PC to get the commands for reconfiguration. General setup of the system including the interface between the Virtex II FPGA and Reprogrammable Controller-Loader (Stage 2) is the same as it was described in the section 4. However, in addition to the Stage 2 platform it was required to have a video capturing board which would input an image data-frame over an eight bit bus accompanied with the clock signal and data request. Upon the request of the Virtex II FPGA by assertion of the request line video capturing board would transmit a frame with a size of 640 x 480 using 8 bit resolution, which is converted into 640x480=307200 bytes and it would be saved in to one out of four available SRAM modules on the Virtex II FPGA board (Figure 4.9). The resulting data would be first saved in to the SRAM modules and eventually transmitted to a display output board over similarly designed I/O interface. However, the amount of data to be output is much larger (as it was described in the Section 5.1) and equal to 640 x 480 x 2 + 256 x 3 = 615168 bytes.

After completion of each stage of processing, FPGA would assert the done signal to the Microcontroller, which in turn would initiate the reconfiguration of the FPGA using the next core and this loop would repeat until it wouldn't be interrupted from the PC side. Detailed operational diagram of the TPM platform with reconfiguration sequences is presented in Figure 5.2.

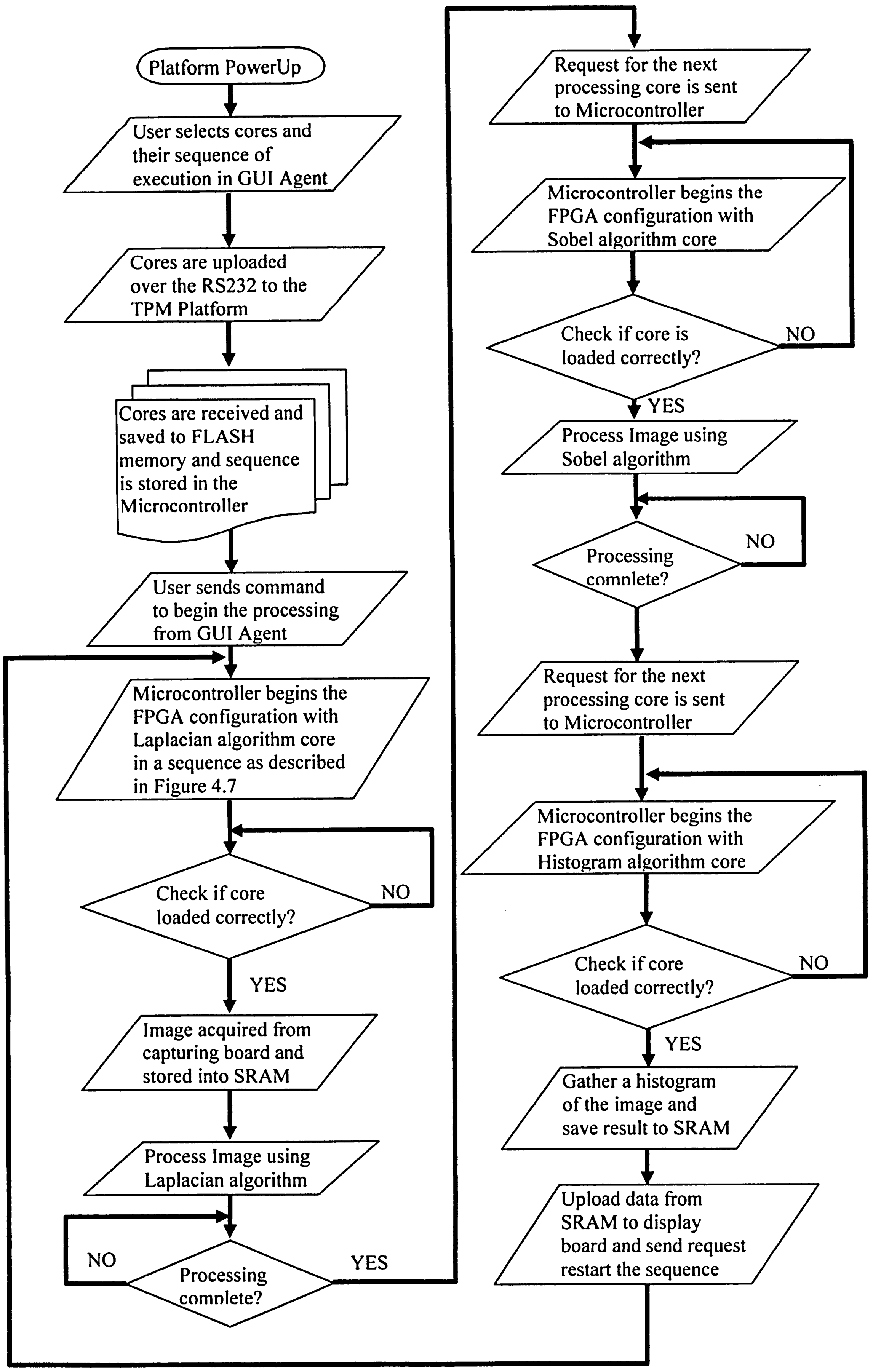

Platform PowerUp
User selects cores and their sequence of execution in GUI Agent
Cores are uploaded over the RS232 to the TPM Platform
Cores are received and saved to FLASH memory and sequence is stored in the Microcontroller
User sends command to begin the processing from GUI Agent
Microcontroller begins the FPGA configuration with Laplacian algorithm core in a sequence as described in Figure 4.7
Check if core loaded correctly?
NO
YES
Image acquired from capturing board and stored into SRAM
Process Image using Laplacian algorithm
Processing complete?
NO
YES
Request for the next processing core is sent to Microcontroller
Microcontroller begins the FPGA configuration with Sobel algorithm core
Check if core is loaded correctly?
NO
YES
Process Image using Sobel algorithm
Processing complete?
NO
Request for the next processing core is sent to Microcontroller
Microcontroller begins the FPGA configuration with Histogram algorithm core
Check if core loaded correctly?
NO
YES
Gather a histogram of the image and save result to SRAM
Upload data from SRAM to display board and send request restart the sequence

Figure 5.2: TPM platform operation flowchart

After running several tries of experiments it was determined that to process 307200 pixels by performing consecutively parallel 3x3 matrix manipulations each per clock cycle it would take approximately 3 ms running at 105 MHz system clock. Similarly, downloading the image from the image capturing board to the first SRAM module required transmission of 307200 pixels and at a speed of 105 MHz it took about 3.4 ms as it can be seen from the Figure 5.3 bellow. Timing of this process was photographed from the screen of Logic Analyzer that was attached to the TPM platform.

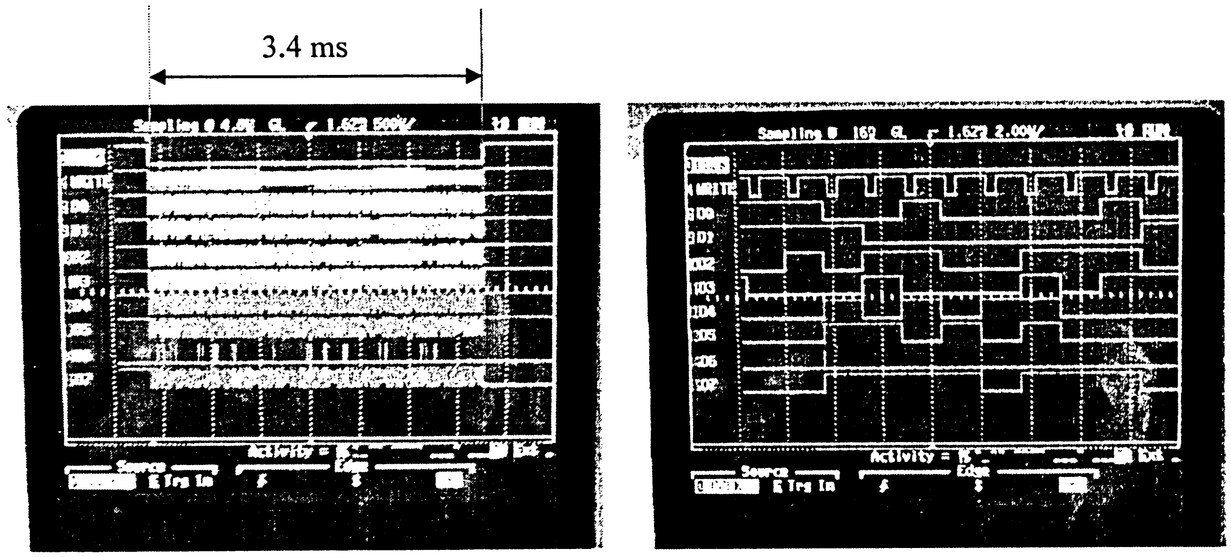


Figure 5.3: Timing capture of the Logic Analyzer attached to the TPM platform.
a) Shows whole data transmission process and
b) Shows the zoomed data transmission in details.

In case of processing the Laplacian image enhancement it was not possible to process all of the pixels at once since an image had to be processed in steps as it was described in section 5.1.1 and this required to perform first the parallel 3x3 matrix multiplication using the Laplacian matrix, which took ~3 ms, and temporary result was saved in the third block of the SRAM. This operation took another ~3 ms cycles since 3x3 multiplication was done in one clock cycle *(i)* and on the next clock cycle *(i+1)* resulting data from that multiplication was stored into the memory. The last operation was to write the last resulting pixel into the SRAM. Following this operation, a pixel by pixel subtraction of temporary result from the original image and saving the final result was performed which gave the enhanced result of the original image which required another ~3 ms. For the Sobel algorithm implementation it was required to perform two 3x3 multiplications and storage of the sum of the two multiplication into an SRAM location. Since multiplications were pipelined it is possible to perform the processing in ~6ms. At last, for the histogram of the image, it was important to consider that there is a possibility that all of the pixels will have the same grayscale value and due to that fact enough space had to be allocated per grayscale value to save number up to 307200. Since two bytes would only give the maximum value of 65536 it was decided to use 3 bytes per grayscale value, and since camera had an 8 bit grayscale resolution only 256 such locations were required. To get a histogram it was required to go through the whole image and using the pixel value as an index to the 256 location increment the location. Similarly such procedure was preformed in approximately ~3ms for the case of going through the whole image and another 768/105000000=7 us. As a final step was a transmission of the original image, Laplacian enhanced image, edge detected image, along with the histogram on to the displaying platform, which would perform the image output to the user. Final transfer from the SRAM to the display board took about 9 ms to perform and potentially could be increased if the data bus is increased to 16 bits of parallel transfer.

In conclusion it is important to recap that all of those image processing operations were made on a FPGA with a maximum size of the biggest core, and potentially could be increased to as many cores as we want, the only limitation would be the size of the external FLASH memory.

5.3 Implementation of the algorithms on the RISC Microcontroller platform

To compare execution of the FPGA platform with TPM and the platform based on RISC controller, it was decided to develop and build a special microcontroller prototype test platform. This platform is based on the PIC18F442 Microchip RISC controller in conjunction with Xilinx Complex Programmable Logic Device (CPLD) XC95144XL – TQ100, which was chosen for the purposes of custom interfacing to different peripherals similar to peripherals interfaced to the FPGA platform with TPM. For example, due to the fact that SRAM module needed 5 control lines along with 16 data lines, and 17 address lines, microcontroller with a total of 33 lines would not be able to handle even one SRAM module, On the other hand CPLD with 77 I/O lines would be able to accommodate that need. Figure 5.4 depicts the experiment setup of the Microcontroller platform.

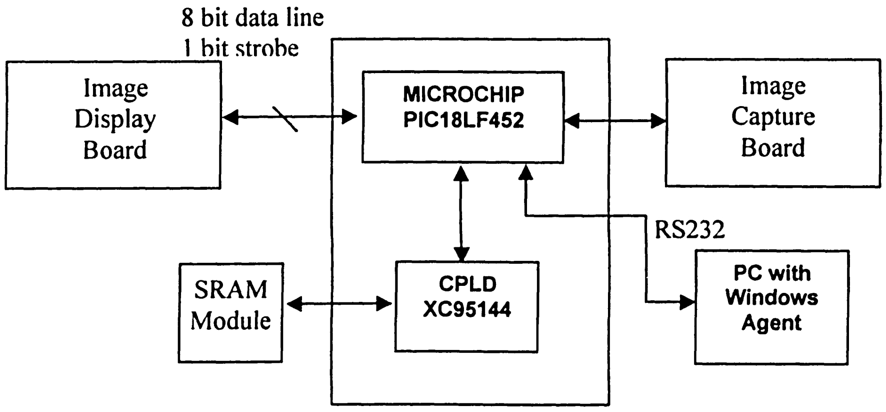


Figure 5.4: Microcontroller setup for the experiments

Similarly to the FPGA platform with TPM, microcontroller would request the information from the video capture board and following it by downloading the data into the SRAM memory block through the CPLD. In case of the microcontroller it was possible to implement all of the algorithms and execute them one after another, however running at a maximum 40 MHz clock, execution of multiplication of 3x3 matrix could not be done in parallel and on top of that RISC architecture only allowed to perform the multiplication in 20 instructions. In addition, execution of pipelined instructions on a Microchip microcontroller with RISC Harvard architecture takes 4 clock cycles per instruction cycle. Therefore, the actual execution speed is decreased to 10 MIPS (Million Instructions per Second). Whole implementation of the program that would first receive the data, transfer it onto the CPLD, perform the execution of Laplacian image enhancement algorithm, Sobel edge detection, get the histogram of the image, and at last transfer it onto a display board required 199,752,997 instruction cycles and execution took 19.9752ms. There is a limit of functions that could be stored on the microcontroller platform because of limited size of the program memory of the microcontroller. The flowchart is presented in the Figure 5.5.

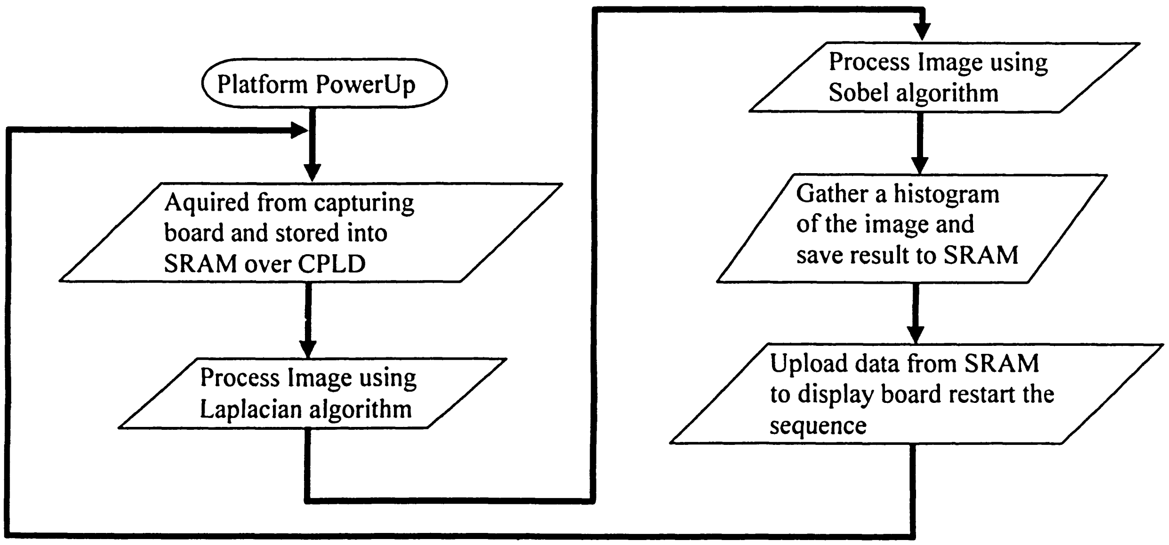

Platform PowerUp
Aquired from capturing board and stored into SRAM over CPLD
Process Image using Laplacian algorithm
Process Image using Sobel algorithm
Gather a histogram of the image and save result to SRAM
Upload data from SRAM to display board restart the sequence

Figure 5.5: Flowchart for the microcontroller's platform operation.

At last, for getting the perspective on algorithm execution performance it was needed to implement the algorithms on the PC.

5.4 Implementation of processing algorithms on a PC platform

For the case of a PC implementation a system with an AMD 3000+ processor was selected running at a 2.1 GHz clock speed. In addition to that a 1 GB of SDRAM was present on the system and a Windows XP operating system was running on it. Image processing algorithms were implemented on the MATLAB tool at first and in addition they were also implemented in Visual C# .NET development studio in order to get realistic results that can be obtained in execution on a conventional PC. For the purpose of measuring exact timing a time stamp was implemented which recorded the beginning of the execution of the algorithm and its completion. All of the algorithms that were implemented in previous two sections were implemented in MATLAB and Visual C# as well and were run one after another in a one monolith program. Parts of reception of the image from the image capturing board and its transmission of the processed images to the display board was omitted since it was not critical for the experiment and in addition to that MATLAB has only RS232 serial interface which is much slower and would not give a reasonable comparison result, and in case of the Visual C# it required a USB interface which was not present on a capturing board. A pre-captured and saved image was used for the original image, and the resulting images were plotted to the screen and can be seen in Figure 5.6 bellow.

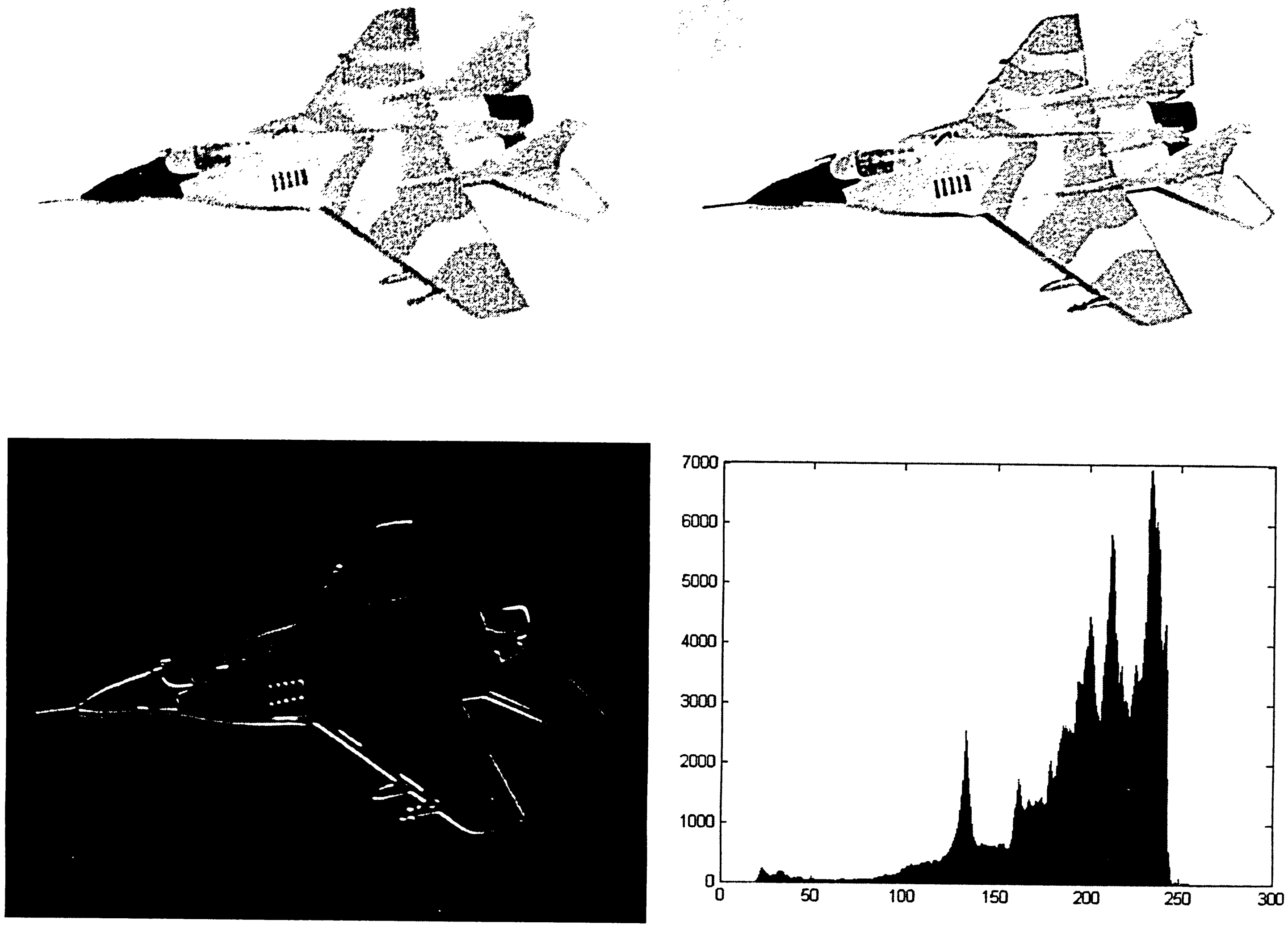


Figure 5.6: a) Original Image b) Laplacian enhancement image c) Sobel Edge detection image d) Histogram of the image

After running the experiments it was found that execution of Laplacian enhancement algorithm on a 640x480 grayscale, Sobel Edge detection algorithm, and histogram calculation took 550 ms.

5.5 Results and analysis.

In the Table 5.1 all the acquired results from three platforms were summarized and also the scaled results were presented as well to be able to have an equal comparison between the execution times of FPGA platform with TPM and Microcontroller platform.

	TPM platform	Microcontroller platform	AMD 3000+ PC
Algorithms executions at particular Frequency	21ms running at 105MHz	19975ms running at 40MHz	550 ms running at 2100MHz
Scaled execution of algorithms to 40Mhz	55.125 ms	19975ms	28875 ms
Acceleration in comparison to PC	523.80 times	1.45 times	1

Table 5.1: Results acquired from the algorithms experimentation

As it can be seen from the above table initially performance of the TPM platform running only at 105 MHz performs about 40 times faster than a PC running at 2100MHz clock. However, in order to bring the values in the perspective results were scaled to the frequency of Microcontroller which was 40MHz. As it is indicated on the Figure 5.7 which represents the speedup of the FPGA platform with TPM, it performs at least 523.80 times faster than a PC, and similarly 361.6 times faster then Microcontroller platform.

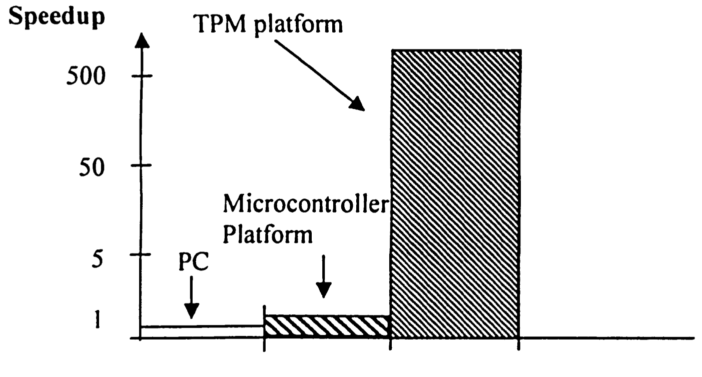


Figure 5.7: Logarithmic representation of platform's speedups

Considering that if more algorithms are required to be processed by the platform it doesn't require major modifications such as it would in case of a Microcontroller based system and PC, the only change would be scheduling an additional core. In some cases execution will be simply impossible since microcontroller based systems have limited program space and can't be dynamically reconfigured, to accommodate more cores, where in case of FPGA platform with TPM it doesn't cause a problem since cores are resided in an external FLASH memory and is only a matter of adding another memory chip. In addition as it can be seen from the Table 5.1 time considered for the processing of the algorithms did not include reloading of the cores since for the Virtex II platform core divisions can be organized in three section for example, where first is a monitor which is responsible for reconfiguration and reception of data, and the other two are the working cores. This enables to reload one core while the other one is performing the algorithm computation. Considering that whole configuration file of Virtex II 1000 FPGA takes 511268 bytes, therefore one third will need about 170kB, and at the speed of 50MHz [39] it will need about 3ms, which is falls under the algorithm processing time. Obviously it is possible to vary core sizes and therefore to decrease the time of reconfiguration. At last, it is important to mention that in this scenario in order to implement X algorithms on single platform with static configuration it would require to have a X/3 times size of the FPGA than it was used for TPM platform, where in the TPM scheme it's a matter of adding another core into a schedule list. This gives huge advantages for systems that have many steps of computation or different computing algorithm requirements.

The results of the above experiments as well as the FPGA based platform for processing video data-streams utilizing different algorithms (e.g. edge detection, stereo data-frame combination, etc.) were presented on SVAR2005 (Space Vision and Advanced Robotics) [65]. The platform was also used for experiments to prove the concept of self-restoration mechanism for SRAM FPGA devices working in the radiation intensive environment[66].

5.6 System Advantages and Limitations

The proposed platform architecture is oriented for the data-stream(s) processing where large and framed volumes of data should be processed within specified period of time. Therefore, as was shown in the Chapter 3, proposed approach is effective when required period of complete processing of the data-frame is longer than reconfiguration time of all involved processing cores. In this case the proposed architecture may provide high cost-efficiency because of reuse of the same hardware resources for different parts (cores) of processing pipeline. This is the biggest advantage of the proposed approach because it allows minimization of hardware and power consumption and thus, minimization of dimensions and weight of the system based on the above approach.

However, it is important to mention some limitations of this type of platform and general applications that this particular system is best suited for. As it was stated before in the Chapter 3, effective operation of the platform is possible only in the case when the configuration (loading) time of the next processing core is shorter than data processing period of the previous core. Therefore, if the data-frame associated with current core requires less processing time than the loading time of the next core, it would cause a delay in processing. This is the biggest limitation of this platform, and in turn does not give a possibility of data-frames to be very small (e.g. > 30000 data words for Xilinx Virtex II FPGAs).

Another limitation of proposed approach comes from the sequential nature of data-frame processing on the different processing cores (sub-processors). Obviously, if cores are loaded one after another the peak performance of the system is lower or much lower than in case when the complete processing core (for the entire FPGA device) is utilized. In fact these algorithms were implemented on a platform with XCV2000e by a group from University of Calabria in Italy [67] and based on their results the implementation of the Laplace and Sobel algorithm on a 1024x1024 image with 3x3 matrix convolution, similarly as it was done for the TPM platform, took 4.6ms. Therefore implementation of both algorithms and their execution simultaneously is about faster than the implementation using temporal partitioning mechanism.

However, this limitation can be avoided in cases when specification of the system requires data-frame processing period (real-time requirements) longer than sum of processing times of the data-frame on each of sub-processing cores (see Chapter 3). This and only this case allows reuse in different periods of time the same hardware and thus, minimization of hardware resources, physical dimensions of a system and power consumptions. Furthermore, recently it became possible for many applications because processing frequencies in the FPGA has increased dramatically when the real-time specifications did not change much. Thus, it made possible hardware sharing in time (temporal partitioning) of computing and communication resources between different segments of a task and associated processing cores. Definitely, TPM has its drawbacks but the gain of the TPM approach is in flexibility of having non-static amount of processing cores, which could be advantageous in wide range of applications where framed data-streams (e.g. video-frames, communication data-packages, etc.) has to be processed within certain periods of time.

Summary

In conclusion to this Chapter it can be said that in order to prove the benefits and efficiency of the TPM mechanism it was decided to perform processing of the video image processing algorithms on a FPGA platform with TPM, as well as AMD 3000+ PC, and PIC18F452 based microcontroller platform. Three different algorithms were selected: Laplacian image enhancement algorithms, Sobel Edge detection algorithms, and at last histogram calculation algorithm. For the FPGA platform with TPM algorithms were implemented as loading cores in VHDL language, and in Embedded C language for the Microcontroller system. In case of PC, algorithms were run on a pre-captured image and implemented on a Visual C# which in turn after compilation gives a realistic performance of how it would operate in case of its use in an industrial application. Several results were obtained that were presented in Table 5.1 and Figure 5.7 that indicated that FPGA platform with TPM performed by far faster than Microcontroller based system or PC. This proved that approach of TPM is much more efficient than RISC based embedded systems as well as a conventional PC with superscalar architecture for the class of tasks with big volumes of framed data (e.g. video data-frames).

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Major Contributions

In conclusion of this thesis it important to mention that motivation of this project was to develop a mechanism and implement it in a form of operational platform, which would allow fulfilling the growing demand for high speed processing systems that can perform manipulations with large volumes of data. It is also important to note that this type of solution became only possible in recent years due to technological advances in reconfigurable logic and especially the capability of Active reconfiguration. Due to the Xilinx FPGAs feature of Active reconfiguration it was possible to design a mechanism which would allow reconfigurations of the part of the device, while it is still in operation. The reasoning behind using FPGAs instead of Microprocessors, DSP processors, or Superscalars was that the FPGA's would be capable of having non fixed architectures unlike the ones mention previously and it would in turn decrease the size of the platform and make it suitable to be low cost a give a capability for future upgrades.

As an objective of this project was to develop Temporal Partitioning Mechanism (TPM) and implement it in a form of a platform using latest SRAM based FPGAs, which would require careful theoretical analysis, review of the existing system and of course development of actual hardware, firmware, and software.

Major contributions of this work were divided into three parts: theoretical analysis, development & design, and experimentation on the developed platform. Each of the parts was described in thesis Chapters. Where, Chapter two was concentrated on the literature overview from different types of sources such as books, journal and conference publications, as well as datasheets, and internet sources regarding the existing platforms with classification of different types of architectures. Primarily it gave a comparison between the advantages and disadvantages of fixed architectures of microprocessor, Superscalars, VLIWs and reconfigurable architectures such as the ones on FPGAs and earlier developments of RaPiD and Chess Array. In particular, the advances of employing Instruction Level Parallelism which increased the productivity of the currently used systems and flexibility that FPGAs offer with their reconfigurable processors. In addition Chapter discussed statically and dynamically reconfigurable processors and the capabilities they offer to the market. Following, the review of the works in current market Chapter 3 concentrated mostly on the theoretical proof of the advantage of TPM system and comparison of it to the other previously mentioned fixed architecture systems. Chapter also gave theoretical comparison of actual case of video processing and its comparison to the system implemented on microprocessor and Superscalar processor. In addition it described a general operation of the RTHOS and how it could be interacting with the Virtual Hardware Components (VHC), which at the implementation stage are called *cores*.

Following the theoretical part design and implementation was described in Chapter 4, which described several stages of the development beginning from the design the schematics and layout of the PCB, which was later manufactured. Chapter goes in to details of why particular parts such as Microcontroller, CPLD, Flash, and FPGA were chosen and how they were interfaced between each other. In addition to development of hardware, software component had to be added as well since interaction between the user and the platform had to be implemented on the PC in a form of Windows Agent application. This required implementation of such an application in the graphical interface which was done using Microsoft Visual Studio and allowed simplification of the interaction between the user and the TPM platform. In addition to software, soft-cores had to be developed for the CPLD and firmware had to be designed for the Microcontroller which involved use of Xilinx ISE and Microchip MPLAB IDE software packages. As the second stage of the platform development required several modification due to the problems found in the testing of the first one allowed adding the Spartan 3 FPGA and also having additional SRAM memory modules which increased core loading capability. At last, Chapter four concludes on an integration of stage 2 platform with the existing Virtex II platform which was developed in previous years in ERSL. As a continuation of the work Chapter five covers the experimental results that were obtained on the TPM platform and several its components as well as the comparison of the test and experiments that were run on the embedded microcontroller system and on a AMD 3000+ PC system implemented in Visual C#. Tests concentrated on gathering data of core reloading using different means such as the one used in the platform as well as the means of Microcontroller and PC interface. In addition results were also gathered from the execution of the particular task of video edge detection using the Sobel [68] algorithm running on the developed platform with use of TPM, as well as PC, and RISC Microcontroller with special firmware program developed specifically for the experimental purpose.

This Chapter concluded the work and recapped the Chapters included in this thesis and presented future work that would follow upon the completion of this project.

6.2 Future Work

As a future work of this project it is important to note that Temporal Partitioning Mechanism gave a great possibility in use of dynamically reconfigurable processor. However, as the work progressed at the last stage of the development it was clearly seen that combination of Spatial Partitioning Mechanism and Temporal Partitioning Mechanism will clearly give a huge advantage and already possible to be implemented on Xilinx Virtex II and Virtex 4 Families of FPGAs. This became obvious after implementation of the monitor and the reconfigurable core, and possibility of having more than one core if larger device is chosen. Also, for the future work it would be important to continue work on Real-Time Hardware Operating System (RTHOS) that incorporates the RTR capability. As a potential component base for future system Xilinx Virtex 4 family will be considered and also incorporation of the stage 2 components into one platform PCB for even higher performance and more compact design.

Bibliography

[1] Gerrit A. Blaauw, Frederick P. Brooks, "Computer Architecture: Concepts and Evolution", Addison-Wesley Professional, 1997

[2] John G. Proakis, Dimitris Manolakis, "Digital Signal Processing: Principles, Algorithms and Applications", Prentice Hall, 1995

[3] TigerSHARC Processor, Analog devices, Processor specifications http://www.analog.com/processors/processors/tigersharc/, 1998

[4] Giovanni De Micheli, "Synthesis and optimization of digital circuits", McGraw-Hill. 1994

[5] Alan Clements, "Microprocessor Systems Design: 68000 Family Hardware, Software, and Interfacing", 3rd edition, Thomson-Engineering, 1997

[6] J.E. Smith G.S. Sohi, "The Micro-architecture of Superscalar Processors", *Proceedings of IEEE Transactions on Parallel and Distributed Systems*, Vol 83, pp.1609-1624, Apr 1995

[7] Binu K. Mathew, "VLIW Processors and Trace Scheduling",CRC Press, 2001

[8] J.Hennessy, D. Patterson,. "Computer Organization and Design", Morgan Kaufman Publishers, California, 1994

[9] Texas Instruments TMS3206x DSP processor http://focus.ti.com/pdfs/univ/3-Wireless.pdf

[10] Stephen Brown, Zvonko Vranesic, "Fundamentals of Digital Logic with VHDL design", McGraw Hill, 2000

[11] Jan M. Rabaey, Anantha Chandrakasan, Borivoje Nikolic, "Digital Integrated Circuits: A design Perspective", Second Edition, Prentice Hall, 2003

[12] Richard C. Jaeger, "Introduction to Microelectronic Fabrication", Second Edition, Prentice Hall, 2002

[13] Gary S. May, Simon M. Sze, "Fundamentals of Semiconductor Fabrication", Wiley & Sons Inc., 2001

[14] Duncan A. Buell, Jeffrey M. Arnold, Walter J. Kleinfelder, "Splash 2: FPGAs in a Custom Computing Machine", Wiley – IEEE Press. , 1996

[15] 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, June 1992

[16] John Morris, Gary A. Bundell, Sonny Tham. "A Scalable Re-Configurable Processor," *Proceedings of 5th Australasian Computer Architecture Conference*, p. 64, 5th 2000.

[17] Reiner Hartenstein, "A decade of Reconfigurable Computing: a visionary Retrospective", *Proceedings of the conference on Design, automation and test in Europe*, Munich, Germany, pp. 642 – 649, 2001

[18] J.R. Hauser, 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

[19] Franco Fummi, Franco Fummi, Mirko Loghi, Stefano Martini, Marco Monguzzi, Giovanni Perbellini, Massimo Poncino,"Virtual Hardware Prototyping through Timed Hardware-Software Co-Simulation", *Proceedings of Design, Automation and Test in Europe 05,* Volume 2, pp. 798-803, 2005.

[20] "Xilinx MicroBlaze Soft Processor Core", http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze, Xilinx Press, 2002

[21] Meenakshi Kaul, Ranga Vemuri, Sriram Govindarajan, Iyad Ouaiss, "An Automated Temporal Partitioning Tool for a class DSP application*", Proceeding of International Conference on Parallel Architectures and Compilation Techniques*, PACT98, pp. 22-28, 1998

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

[23] R.D. Wittig, 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

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

[25] Ahmad Alsolaim, Janusz Starzyk, Jurgen Becker, Manfred Glesner. "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems", *Proceedings of the 2000 IEEE Symposium on Field-*

Programmable Custom Computing Machines, p. 205, 2000

[26] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, R. Reed Taylor. "PipeRench: A Reconfigurable Architecture and Compiler," *Computer*, vol. 33, no. 4, pp. 70-77, April 2000.

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

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

[29] Aravind Dasu, Sethuraman Panchanathan, "Reconfigurable media processing", *Parallel Computing*, Volume 28, Issues 7-8, pp 1111-1139 , August 2002

[30] 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

[31] C. Tanougast, Y. Berviller, P. Brunet, S. Weber, H. Rabah, "Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded real-time system", *Microprocessors and Microsystems*, Volume 27, Issue 3, pp115-130, April 2003

[32] 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

[33] Atmel FPSLIC (AVR with FPGA), http://www.atmel.com/products/FPSLIC/

[34] Xilinx Virtex II Platform FPGA Handbook VG002, December 6, 2000

[35] Farhad Mehdipour, Morteza Saheb Zamani, Mehdi Sedighi, "An integrated temporal partitioning and physical design framework for static compilation of reconfigurable computing systems", *Microprocessors and Microsystems*, In Press, May 2005,

[36] Toshiba 128 Mbit FLASH memory TC58DVM72A1FT Datasheet http://www.toshiba.com/taec/cgi-bin/display.cgi?table=ProductDetail&ProductID=10610

[37] Xilinx Inc. The Programmable Logic Company www.xilinx.com

[38] L. Kessal, N. Abel, D. Demigny, "Real-time image processing with dynamically reconfigurable architecture", *Real-Time Imaging*, Volume 9, Issue 5, pp 297-313, October 2003

[39] Xilinx Inc, "Xilinx application note XAPP290", www.direct.xilinx.com/bvdocs/appnotes/xapp290.pdf, Xilinx Press, 2002

[40] Xuejie Zhang and Kam W. Ng, "A review of high-level synthesis for dynamically reconfigurable FPGAs", *Microprocessors and Microsystems*, Volume 24, Issue 4, 1, pp199-211, August 2000

[41] Xilinx, Inc.:"Xilinx Virtex XCV400E Field Programmable Gate Array Specification," San Jose, CA, 2000.

[42] D. Mesquita, F. Moraes, J. Palma, L. Moller, and N. Calazans. "Remote and partial reconfiguration of FPGAs: tools and trends", *Parallel and Distributed Processing Symposium 2003. Proceedings. International*, 22-26, April 2003.

[43] Carmichael, Carl., XAPP137, "Configuring Virtex FPGAs from Parallel EPROMs with a CPLD", Xilinx Press, March 1999

[44] Kim Goldblatt, Xilinx application note XAPP178 "Parallel configuration protocol", Xilinx Press, 2001

[45] Mois´es P´erez-Guti´errez, Miguel Arias-Estrada,"Library of Hardware/Software Components for Remote Secure Configuration", *Proceedings of International Conference on Reconfigurable Computing and FPGAs*, Santa Maria Tonantzintla, Puebla, M´exico, 2004

[46] Thomas Branca, Brant Soudan, Chris Stinson "Remote Field Upgrades Using FPGAs", Xilinx Press, 2000

[47] Peter Chun, Valeri Kirischian, Sergei Zhelnakov, Lev Kirischian, "Reconfigurable Multiprocessor with Self-optimizing Self-assembling and Self-restoring Micro-architecture", *Presentation on Workshop on Architecture Research using FPGA Platforms,* pp. 52, San Francisco, February, 2005

[48] Valeri Kirischian, Sergei Zhelnakov, Peter Chun, Lev Kirischian, Vadim Geurkov", Uniform Reconfigurable Processing Module for Design and Manufacturing Integration", *Proceedings of Advanced Manufacturing Technologies 2005*, London, Canada, pp. 77-82, May 2005

[49] Altera Inc. FPGA, CPLD & Structured ASIC Devices www.altera.com

[50] Actel Inc. FPGA Programmable Logic Devices www.actel.com

[51] Lattice Semiconductor Inc. FPGA & CPLD Programmable Logic Devices www.latticesemi.com

[52] Xilinx Spartan 3 XC3S400 Datasheet http://direct.xilinx.com/bvdocs/publications/ds099.pdf

[53] Virtex II Pro Development http://www.digilentinc.com/info/XUPV2P.cfm

[54] AMIRIX Virtex-II PCI Platform FPGA Development Board http://www.nuhorizons.com/services/development/amirix/

[55] Silica Xilinx Virtex-E Evaluation Kit http://www.silica.com/en/products/evaluationkits/evaluationkit6.html

[56] Microchip PIC18FXXX Datasheet, www.microchip.com, Microchip Press., 2003

[57] "XC95288 High Performance CPLD Reference manual", Xilinx Press, 2004

[58] FTDI Chips Inc. www.ftdichip.com

[59] IDT Semiconductor SRAM 4 Mbit 256x16 IDT71V416 Datasheet http://www.idt.com/?app=search2&criteria=idt71v416

[60] Andrew Knight ,"Basics of MATLAB and Beyond", Chapman & Hall/CRC, 1999

[61] www.msdn.com

[62] Carl Carmichael, Michael Caffrey, Anthony Salazar, XAPP216 "Correcting Single Event Upsets Through Virtex Partial Configuration", Los Alamos National Laboratories, Xilinx Press, 2000

[63] "PLX CHIP Reference manual", www.plxtech.com/products/io_accelerators/ PCI9030/, PLX Technologies Press., 2000

[64] Kate Gregory, "Special Edition Using Visual C# .NET", QUE press, 2003

[65] Sergei Zhelnakov, Valeri Kirischian, Peter Chun, Lev Kirischian, Vadim Geurkov, "FPGA-based Computing Platform for Stereo-Panoramic Vision", Presentation at Space Vision and Advanced Robotics Workshop, MDA Space Missions, SVAR 2005, May 19 2005

[66] Lev Kirischian, Vadim Geurkov, Irina Terterian, Valeri Kirischian, "Multi-level Radiation Protection Mechanism Based on Self-Restoration of Partially Reconfigurable

FPGA Devices", *Journal of Spacecrafts and Rockets (JSR)*, Revision 2, August 2005

[67] Stefania Perri, Marco Lanuzza, Pasquale Corsonello, Giuseppe Cocorullo, "A high-performance fully reconfigurable FPGA-based 2D convolution processor", *Microprocessors and Microsystems*, Volume 29, Issues 8-9, pp 381-391, November 2005

[68] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, " Digital Image Processing using MATLAB", First Edition, Pearson Prentice Hall, 2004