



University Library

Author/Filing Title BARTZoudis, N.

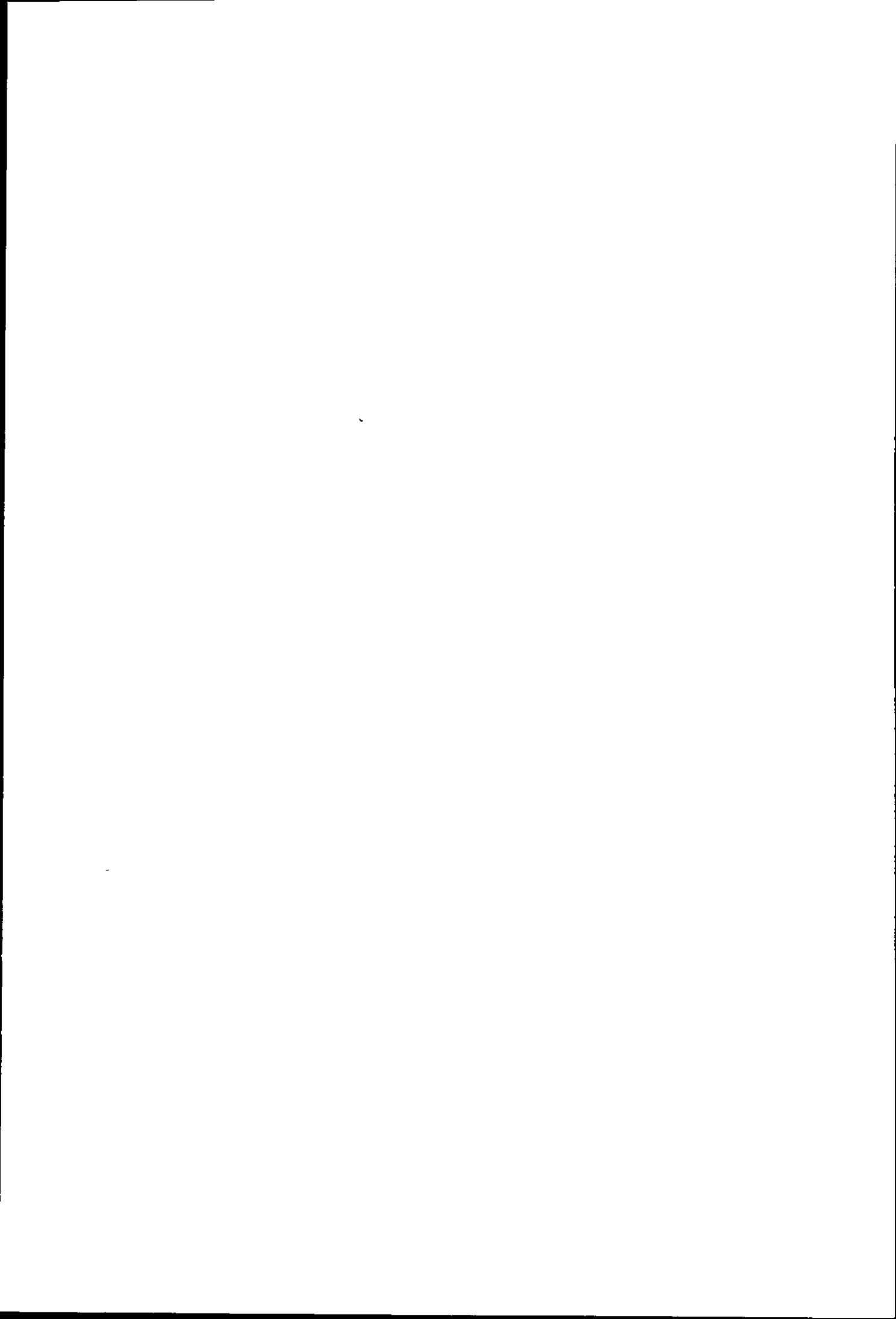
.....
Class Mark T

Please note that fines are charged on ALL
overdue items.

FOR REFERENCE ONLY

0403270960





LOUGHBOROUGH UNIVERSITY
DEPARTMENT OF ELECTRONIC AND ELECTRICAL
ENGINEERING

**Using embedded hardware monitor
cores in critical computer systems**


By

Nikolaos Bartzoudis

**A dissertation submitted for the degree of
Doctor of Philosophy**

Loughborough, March 2006

“... reconfigurable radiation-tolerant FPGAs are flying aboard the Australian scientific mission satellite FedSat, successfully launched on December 14, 2002 from Tanegashima, Japan ... the FedSat reconfigurable computer is the world's first use of this technology in space. As a critical component of the High Performance Computing (HPC) payload, the reconfigurable nature of the FPGAs enable satellites to be rewired without having to be retrieved, thus drastically reducing cost and development time...”[1]

 Loughborough University Pilkington Library
Date SEPT 2006
Class T
Acc No. 0403270960

Abstract

The integration of FPGA devices in many different architectures and services makes monitoring and real time detection of errors an important concern in FPGA system design. A monitor is a tool, or a set of tools, that facilitate analytic measurements in observing a given system. The goal of these observations is usually the performance analysis and optimisation, or the surveillance of the system. However, System-on-Chip (SoC) based designs leave few points to attach external tools such as logic analyzers. Thus, an embedded error detection core that allows observation of critical system nodes (such as processor cores and buses) should enforce the operation of the FPGA-based system, in order to prevent system failures. The core should not interfere with system performance and must ensure timely detection of errors.

This thesis is an investigation onto how a robust hardware-monitoring module can be efficiently integrated in a target PCI board (with FPGA-based application-processing features) which is part of a critical computing system. An error detection and recovery core was implemented that requires no manual intervention for illustrating the benefits of this approach. The Error Detection Monitor Unit (EDMU) can be considered as a monitor and control wrapper for PCI-compatible cores that monitors the application transactions between the PCI-based FPGA board and the host in a timely manner. The nearly concurrent functionality of EDMU allows the core to operate as a “firewall”, preventing the communication between the FPGA-configured application and the host computer, upon detection of errors. A predefined policy framework groups the application and PCI protocol errors according to a severity scale; thus suspicious or undesirable transactions are treated considering their severity rating. Therefore, the PC-based critical system uses EDMU in order to guarantee quality in the offered services, such as maintainability, availability and error tolerance.

The high portability and reusability of EDMU makes it ideal for remote network configuration PC systems and failure-resistant critical computing platforms similar to Active Networks. There are several other target application examples for using the error-detection features of our module, including PCI protocol checking, hardware testing and debugging of predefined conditions.

Acknowledgements

This dissertation is obviously not the result of my own individual efforts alone, but the fruit of intense collaboration. First, I would like to thank my supervisor Professor David Parish for his guidance, and support. David has inspired my academic and research visions and helped me to carry on with my career. I also owe special thanks to Dr. José Luis Núñez who has helped me throughout my research and development efforts as my day-to-day advisor. Finally, I would like also to thank all the people of the department of Electronic and Electrical Engineering, who encouraged and supported me during the undertaking of this research.

What I have experience as a Ph.D. student goes further than academic and research experience alone. My stay at Loughborough was rewarding, challenging and fun. Coming here was a choice I made years ago, a choice that took me months to make, and which I have never regretted for a second. The only reason why this is true is all the people who made it possible for me to be here, to live here, enjoy myself and find motivation and inspiration. I could think of countless friends to thank. Too many to write down all the names and give them the credit and thanks they deserve. I am sure you all know who you are, without having to see your name appear in some list. Rather than fill this page with simply a dull enumeration, I prefer to just say:

'Thanks'. Thanks for being there for me, thanks for all the good times. I won't forget.

Αυτή η διατριβή αφιερώνεται στους
Γιώργο Κρυσταλλία και Πασχάλη
Ευχαριστώ για όλα αυτά που κάνατε για μένα

Publications*

- [1] N.G. Bartzoudis, A.G. Fragkiadakis, D.J. Parish and J.L. Núñez, "A System for Fault Detection and Reconfiguration of Hardware Based Active Networks", in *Proceedings of the 10th IEEE International On-Line Testing Symposium (IOLTS 2004)*, Madeira, Portugal, July 2004, pp. 207-213.
- [2] N.G. Bartzoudis, A.G. Fragkiadakis, D.J. Parish, J.L. Núñez and M.J. Sandford, "Reconfigurable Computing and Active Networks", in *Proceedings of The 2003 International Multiconference in Computer Science & Engineering (ERSA'03)*, Las Vegas, U.S.A., June 2003, pp. 280-284.
- [3] N.G. Bartzoudis, A.G. Fragkiadakis, D.J. Parish and J.L. Núñez, "A Monitor Module for Active Networks with Hardware Support", in *Proceedings of IEE System-on-Chip Design, Test and Technology*, Cardiff, U.K., September 2003.
- [4] N.G. Bartzoudis, A.G. Fragkiadakis, D.J. Parish and J.L. Núñez, "An Embedded Monitor Module for Active Networks with Hardware Support", Loughborough University- Department of Electronic & Electrical Engineering, Electronics Systems and Control Research Division, in *Proceedings of the 1st Divisional Mini-conference*, Swithland, September 2003, pp. 28,29.
- [5] A.G. Fragkiadakis, N.G. Bartzoudis, D.J. Parish and M.J. Sandford, "Hardware Support for Active Networking", in *Proceedings of The 2003 International Multiconference in Computer Science & Engineering (SAM'03)*, Las Vegas, U.S.A., June 2003, pp. 27-33.
- [6] A.G. Fragkiadakis, N.G. Bartzoudis, D.J. Parish and M.J. Sandford, "Active Networking using Programmable Hardware", in *Proceedings of the 2003 Postgraduate Networking Conference (PGNet 2003)*, Liverpool, U.K., June 2003.
- [7] M. Sandford, D. Parish, A. Fragkiadakis, N. Bartzoudis, "An Internet-Friendly Architecture to support the Rapid Deployment of new network services", *IFAN whitepaper*, HSN Group, Department of Electronic Engineering, Loughborough University, 2003.

* Two more papers, submitted to journals, are under review.

Citation by other authors

- Noriuki Aibe and Moritoshi Yasunaga, "Reconfigurable Parallel Comparison Architecture and Its Application to IP Packet Filters", In *Proceedings of the IEEE International Conference in Field-Programmable Technology (ICFPT) 2003*, pp.363-366, Tokyo, December 2003.
- A.G. Fragkiadakis and D.J. Parish, "Performance Evaluation of a PC-based Active Router and Analysis of an Active Secure FTP Application", in *Proceedings of Fourth IEEE International Symposium on Network Computing and Applications*, pp. 283-286, Cambridge-USA, July 2005.
- P. H. Cheung, R. Kadgi, V. Saxena and M. Subhadra, "PCI Monitor Product Specification Version 1.0", Department of Electrical & Computer Engineering, Portland State University, ECE-585 Microprocessor System Design, 2005.

CONTENTS

Declaration.....	i
Abstract.....	ii
Acknowledgements.....	iii
Publications.....	v
Contents.....	vi
List of figures.....	x
List of tables.....	xii

Chapter 1: Introduction

1.1 The research topic.....	1
1.2 The research and development challenges.....	3
1.2.1 Application processing with reconfigurable programmable arrays.....	3
1.2.2 Active Networks – a dependable, reconfigurable-computing system	4
1.2.3 Security implications and counteract framework.....	6
1.2.4 Design and implementation of a hardware monitor component.....	7
1.3 Target architectures and contribution.....	8
1.4 Summary.....	9

Chapter 2: Critical real time systems and FPGAs

2.1 Field Programmable Gate Arrays.....	11
2.1.1 Single context FPGAs.....	14
2.1.2 Partially reconfigurable FPGAs.....	15
2.1.3 Reconfigurable computing.....	16
2.2 Integration of programmable logic in critical systems.....	17
2.3 Embedded real-time critical systems.....	20
2.4 Fault, errors and failures.....	22
2.4.1 Systematic and physical failures.....	22
2.4.2 Failure modes.....	23
2.5 Security issues.....	25

2.5.1 The security challenges.....	26
2.5.2 Embedded system design and security issues.....	27
2.6 Testing, debugging and monitoring.....	29
2.6.1 Testing and debugging.....	29
2.6.2 Monitoring.....	30
2.6.3 Types of monitoring systems.....	31
2.7 Active Networks.....	35
2.7.1 Active Network usage.....	36
2.7.2 Basic concepts.....	37
2.7.3 Active Networks and programming interfaces.....	38
2.7.4 Using existing networks and topologies.....	39
2.7.5 Active Networks based on PC routers.....	39
2.7.6 Active Networks processing requirements.....	41

Chapter 3: Background in monitoring and programmable networks

3.1 Monitoring and error detection.....	43
3.1.1 Hardware monitoring domains.....	43
3.1.2 Hardware monitoring using logic analyzers.....	44
3.2 Reconfigurable computing and programmable networking.....	46
3.2.1 The P4 architecture.....	46
3.2.2 The Field-programmable Port Extender (FPX).....	47
3.2.3 ANN - Active Network Node.....	48
3.2.4 AMnet.....	48
3.2.5 Further similar research.....	49
3.3 Remote FPGA (re)configuration.....	49
3.3.1 Xilinx – Internet Reconfigurable Logic (IRL).....	50
3.3.2 Altera – Remote system configuration.....	50
3.3.3 Triscent E5 – Secure remote updates.....	51
3.3.4 Hardware Objects Technology Manager.....	51
3.4 Relation to our work.....	52

Chapter 4: Motivation and Framework

4.1 Processing power versus flexibility.....	54
4.2 Active Networks - a critical computing system.....	59
4.2.1 The security context.....	62
4.2.1.1 Packet replication.....	64
4.2.1.2 Packet corruption.....	64
4.2.1.3 Packet damage.....	64
4.2.1.4 Packet theft.....	65
4.2.1.5 Packet transformation.....	65
4.2.1.6 Packet fission and fusion.....	65
4.2.2 Hardware “bugs” and malicious attacks.....	66
4.2.2.1 A generic hardware attack scenario.....	66
4.3 Counteract framework.....	67
4.3.1 Policy-based monitoring.....	69
4.3.2 Timely detection of system threats.....	70
4.3.3 Error tolerance.....	71
4.3.4 Erroneous application removal.....	72

Chapter 5: System Architecture & Design

5.1 Design issues.....	73
5.1.1 The hardware platform.....	73
5.1.2 The specific properties of the PCI board.....	75
5.2 The policy-based monitoring framework.....	78
5.2.1 PCI protocol errors.....	83
5.2.2 Common application errors.....	87
5.3 The Error Detection Monitoring Unit.....	92

Chapter 6: Results

6.1 Design validation.....	101
6.2 Simulation results.....	102
6.3 Real time verification of the Error Detection Monitor Unit.....	109
6.4 EDMU and Active Networks.....	120

Chapter 7: Conclusion

7.1 Synopsis of achievements.....	123
7.2 Future work.....	127
References.....	129
Appendix I.....	140
Appendix II.....	146
Appendix III.....	147
Appendix IV.....	149

List of figures

1.1	An Active Network with a reconfigurable processing engine.....	5
2.1	The world of Integrated Circuits.....	11
2.2	A general representation of a SRAM-based FPGA device.....	12
2.3	The CLB contains two slices. Each slice contains two 4-input look-up tables (LUT), carry & control logic and two registers. There are two 3-state buffers associated with each CLB, that can be accessed by all the outputs of a CLB.....	12
2.4	The sequence steps that result in system failure.....	19
2.5	Cost function of real time tasks.....	21
2.6	Cause consequence diagram of fault, error and failure.....	22
2.7	The relation between the failure modes.....	24
2.8	Critical systems and the inherent design requirements.....	28
2.9	Hardware monitoring steps.....	29
2.10	Temporal impact of monitor types.....	34
4.1	Bandwidth and computational complexity of some applications.....	56
4.2	Temporal versus spatial computation.....	57
4.3	Performance and flexibility issues related to development and unit cost.....	59
4.4	An Active Network with a reconfigurable processing engine.....	62
5.1	The required reconfigurable target components.....	77
5.2	The two state machine types used for the monitor logic.....	79
5.3	A general view of the PCI core signals.....	80
5.4	Generating policies.....	82
5.5	Another representation of the policy generation.....	83
5.6	A delayed transaction example.....	89
5.7	The EDMU is passively integrated with the PCI core	94
5.8	An “X-ray” of the EDMU.....	95
5.9	An indicative ASM chart of the EDMU functionality.....	96
6.1	The Altera PCI simulation testbench.....	103
6.2	The Xilinx simulation testbench.....	104
6.3	When the host sample BAR0 (i.e. signal bar0_rd) the output data of	

	EDMU are transferred to the my_s_data_in vector signal.....	105
6.4	The detection of a PCI protocol error in simulation (error tolerance).....	107
6.5	EDMU forces the configured application to issue a target abort.....	108
6.6	The ADM-XPL board topology.....	111
6.7	Providing error-tolerance in the system operation.....	113
6.8	Another case of error tolerance.....	114
6.9	The PCItree application “reads” the BAR0 memory space.....	115
6.10	A data capture from the ChipScope Analyzer.....	116
6.11	Multiple errors with different severity rating are detected by EDMU.....	117
6.12	The execution of “memtest” is interrupted due to error detection	118
6.13	The JTAG chain shows the 3 programmable devices of the ADM-XPL board: the application FPGA, the PCI core/EDMU FPGA and the EEPROM.....	119

List of tables

2.1	Revenue loss per hour, by business segment.....	18
2.2	Percentage Availability and Downtime	18
3.1	Overview of the related research.....	53
4.1	Implementation details for several FPGA-based application implemented.....	58
4.2	A synopsis of the Active Application functions and their effects.....	68
5.1	Auxiliary signals and their respective Boolean expression.....	81
5.2	PCI specification comparison.....	84
5.3	Monitoring events.....	87
6.1	Implementation results for the EDMU.....	110
7.1	A synopsis of the most important properties of the EDMU.....	126

CHAPTER 1

Introduction

1.1 The research topic

Today's computer-based products are complex and require extensive effort for their design and test. The complexity in such systems is increased because they comprise many components, advanced software and hardware functionality. This trend is clearly seen in the consumer electronics market, and in state-of-the-art industrial systems. The development of these products tends to be challenging as well as increasingly time-consuming, expensive, and error-prone. Therefore, the developers need to cut down the development time and improve quality, which in turn, demands better tools and development methodologies.

One important aspect in the development process of computer systems is observability, i.e. the ability to observe the system's behaviour at various abstraction levels in the design. The observations are required for many reasons, for instance, when looking for design errors, during debugging, during optimisation of algorithms, for extraction of design data, and a lot more. Observability is however not an issue restricted to development purposes only, it may also be necessary after the deployment of products (e.g., for error recovery, for surveillance issues, for collection of statistical measurements). The quality of observability could be characterised as high if the system allows for detailed and accurate analysis of all of its components, and low if the system is obstructive and hard to analyse confidently. The motivation for investigating this area is to provide better observability for complex computer systems based on state-of-the-art programmable logic architectures.

In more detail the aim of this research study is to offer a failure resistant PC platform with in-built hardware monitor and control features, for high performance processing of applications. The Field Programmable Gate Array (FPGA) devices satisfy the ever growing demand for flexible and dedicated processing power. Therefore,

various software-based applications are transformed into programmable hardware applications in order to be integrated in systems that use the FPGA technology. Moreover nowadays, the trend is to create a whole system within a FPGA device by "hardwiring" hardware cores (e.g. microprocessors, memory, transceivers etc) inside the programmable logic area of a FPGA. The new era belongs to this new form of FPGAs that can realise a programmable System-on-Chip (SoC).

The state-of-the-art FPGA devices are becoming increasingly popular; systems that demand re-programmability and high performance parallel processing adopt the solution of FPGAs. The more the popularity of the FPGAs increases, the more they play a key role in important systems since they are used as critical components within them (e.g. co-processors). Therefore, it is obvious that such systems need an integrated monitoring support in order to trace the activity of the applications and guarantee their stable operation. Run-time observability in embedded system architectures is also a requirement for testing, debugging, and for validating design assumptions made about the behaviour of the system and its environment. The classical approach to run-time observability is to apply monitoring, i.e. the process of detecting, collecting, and interpreting run-time information regarding the system's execution behaviour. A significant conclusion of the work presented in this thesis is the fact that on-chip policy-based monitoring support will be required in future development of similar systems, especially those based on SoC architectures. The timely/concurrent detection of errors that are produced from applications configured in FPGA devices will be a major issue in the near future.

One of the main deliverables of the research process is to investigate the impact that application errors can have in critical systems that are built around FPGA devices, using as a case study a PCI-based reconfigurable FPGA platform. Some indicative examples of such systems could be:

- Real time systems,
- Safety critical systems,
- Security sensitive systems,
- Systems that require maintainability, availability and operability features.

The target was to define the requirements of a similar system in terms of performance, security and fault tolerance; also, in which way the applications errors can critically influence vital functions and services of the system. Public network and communication systems are built with the inherent requirement to operate even under harsh conditions (i.e. system malfunctions, security attacks, lack of processing resources, reaching the limit of users that can be served etc). The functions and the services that they deliver are -in most of the cases- important and must be maintained at all cost, or at least up to an acceptable level (i.e. fault tolerance).

1.2 The research and development challenges

A synopsis of the research challenges that this work had to cope with are given in the list that follows:

1. Define a flexible and high performance application-processing platform; justify the selection of the individual hardware and software components with reference to the target topology.
2. Select and study the properties of a real world critical system, based on a computer architecture (i.e. PC).
3. Define the security framework of the system according to the specific properties and requirements of the critical system that is under investigation.
4. Design, implement and test a hardware, monitor and control core that does not introduce intrusiveness to the system application-processing. The core should be permanently configured in order to satisfy the pragmatic needs of a real world critical computer system.
5. Prevent the propagation of errors in the system offering by this way high availability, operability and dependability. Thus, the application-layer errors have to be detected in a timely manner.

1.2.1 Application processing with reconfigurable programmable arrays

Reconfigurable computing systems comprising microprocessor, memory and reconfigurable logic represent a promising technology, which is well suited for bit-intensive applications such as digital signal processing, error correcting codes, control

applications, bit manipulation, compression/decompression and encryption/decryption. The flexibility, capacity and performance of Field Programmable Gate Arrays (FPGAs) have opened up completely new avenues in high performance computation forming the basis of reconfigurable computing. FPGAs can be considered as the obvious candidate for achieving the performance requirements of the bit-intensive applications that are going to be used as part of reconfigurable computing applications. Moreover, the reconfigurability of FPGAs makes them ideal for remote field upgrades of systems. A new bitstream containing a new version of a circuit can be downloaded through a remote connection. Several different approaches address the remote reconfiguration of FPGAs; commercial schemes and academic research projects have proposed different architecture solutions for a remote configuration execution environment. More information on this topic is given in chapter three.

The execution environment of reconfigurable computing applications can be defined as the abstraction layer or the physical component where the applications are implemented or executed. It can be one or more of the following abstraction domains:

1. The microprocessor
2. The kernel space of an operating system
3. A reconfigurable array (FPGA, CPLD etc)
4. A topology comprising of Application Specific Integrated Circuits (ASICs), or other types of ICs
5. A hybrid solution of the above components (System-on-Chip)

1.2.2 Active Networks – a dependable, reconfigurable-computing system

The detection of errors that are initiated from FPGA-based applications in PC systems is a platform-independent concept; hence it is not necessary to integrate it in any of the critical systems that were described in the previous paragraph in order to prove –the obvious- necessity for using it. However, it is important to use an example platform for studying and identifying the requirements, the limits, the technical details, the development issues and the theoretical background of the system under investigation. Active Networks [2] are security problematic systems with real time and fault tolerance constraints that require significant processing resources; their operation

is critical and must be maintained because the services that they deliver are important for the network infrastructure and the users. Considering the available infrastructure, the experience and the know-how at Loughborough University, Active Networks was the obvious candidate for collecting the required structure-definitive specifications for the error detection system. For this reason, a detailed study of the properties and the specifications of an Active Network was delivered. It can be claimed that the performance and security-critical issues of Active Networks with programmable logic support, defined the fundamental design goals and the response of the system.

Architecture level solutions have already been offered to confront the side effects of Active Applications when the Execution Environment is realised either in the Active Host's microprocessor or in the kernel space of the operating system. However, in the scenario under investigation, the execution environment is an FPGA-based reconfigurable platform. Figure 1.1 is a representation of a PC-based Active Router with programmable logic support; the PC carries a PCI board with one FPGA device dedicated for applications that need a bit intensive processing environment.

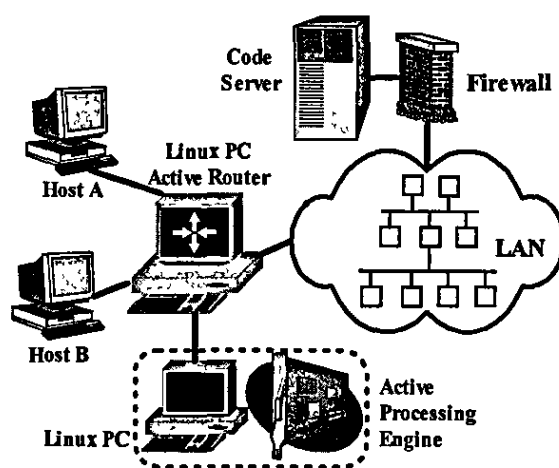


Figure 1.1: An Active Network with a reconfigurable processing engine.

Following the Active Network design approach, a configuration bitstream that targets such a platform could be carried in a series of Active Packets. These packets are identified as active when the Active Router processes them. The Active Router then

decides whether to download the new configuration bitstream to the FPGA device or not, depending on the current activity on the FPGA board.

1.2.3 Security implications and counteract framework

The dependable system described in the previous section has FPGA processing and reconfigurable features that are examined as a case study in order to detect vulnerabilities and system defects. Applications that configure the FPGA device of the PCI board, pose threats analogous to software program threats (i.e. resource consumption, application layer faults, design bugs). The new configuration can potentially extend and modify the network infrastructure. The routing and the forwarding of the packets are vital processes of such system and must not be interfered with, by any means since their maintenance is important for the viability of the network. Moreover, the operability of routers in general is critical to the proper and correct running of many other important systems and services. Therefore, a designer of reconfigurable systems of this type should consider new possible threats on top of the already existent network security issues [3], [4], [5], [6], [7], [8], and [9].

The security threat model has to be identified, considering all the platform-specific characteristics and the requirements that were set for a realistic implementation of the FPGA-based reconfigurable system. The potential threats and the “enemies” for the candidate architecture have to be identified. This important design consideration could allow the monitoring component to be integrated in a FPGA-based programmable architecture.

Designers of dependable computing systems should deal with several issues related with reliability, availability, testability, maintainability and system robustness. In public network systems, a part of which could be Active, security issues are raised and thus measures must be taken to overcome the effects of potential failures. Considering the above, the design of the FPGA-based reconfigurable platform realises a set of measures, which are divided into four levels of action:

- I. Policy-based monitoring,
- II. Timely detection of system threats,
- III. Error tolerance,

IV. Interrupting the erroneous application and applying reconfiguration.

The analysis of these levels of action is given in Chapter 4.

1.2.4 Design and implementation of a hardware monitor component

This thesis focuses on design and implementation issues for building an efficient hardware monitor component, which is passively configured in an embedded FPGA-based PC platform. One of the fundamental concerns of the monitoring system is the concurrent detection of errors (e.g. protocol and application) in order to prevent their propagation to other system areas and their interference with system performance and dependability. The errors were grouped in two categories; PCI protocol errors and application errors related with misuse of local resources, "bugs" and execution of malicious code.

The application, which is configured in the FPGA of the PCI board, could potentially act as a master PCI agent across the PCI bus; this means that it could access various system services and sensitive areas in almost a concurrent way. It is clear that this case is highly undesirable because it can result in system malfunctions, degradation of services and system-processes, application hijacking, breaching of security policies and data-integrity policies. In the worst case scenario "hard"-restart of the PC that carries the PCI-FPGA board might be needed (e.g. when an illegal memory write is attempted, PCI bus contention etc). Re-booting a PC with high availability and operability requirements, breaches the specifications and design requirements of most reconfigurable FPGA-based computers, used within dependable systems.

For this reason it is required a carefully designed monitor component capable of identifying and intercepting illegal or unauthorised activity in a concurrent way. The sequential nature of software monitors make them unsuitable for concurrent monitoring of FPGA-based applications. However, even by using hardware monitors, the technical and research challenge of concurrent (or nearly concurrent) application monitoring of the PCI-based FPGA board, still remains to be met. The monitoring component has to examine the application transactions as well as the interaction of the FPGA application with the host. Although the main operational and design issues are related with the activity of the FPGA application, the monitor component should also check the host's

transactions (e.g. when the FPGA is a target PCI agent); a bidirectional monitoring policy is applied by this way. A series of signals has to be monitored each clock cycle acquiring by this way several qualitative and quantitative data. Various problems related with contention (e.g. PCI-FPGA board local bus) or reuse of system-signals make the concurrent monitoring a difficult task to deliver. Details for these issues will be given in chapter six. Additionally, the fact that the FPGA-based processing system has to be monitored online increases the degree of complexity, as far as the research challenge is concerned. This means that the FPGA-based reconfigurable PC cannot afford to stay offline due to misuse of local resources from the configured application, since it will be part of a dependable system. Poor quality in the services and processes delivered by such systems is also an unwanted event.

1.3 Target architectures and contribution

The platform could be used in several different application examples and architectures. It can be adapted to fit the specific needs of various application-topologies, when applying simple modifications in the specifications:

- Configuration bitstreams can be stored at a known network location and then upgrade the PCI-based reconfigurable FPGA platform, once it is shipped and installed for instance in the PC of a final customer. The goal of the error detection core is to ensure that the operation of the PC will remain stable.
- Design Corrections – In the event that a flaw appears in a commercial IP after it is shipped and configured to the PCI-based FPGA board of a final customer, the error detection core provides operability without the need for recalls and field service.
- Performance Upgrades - a company or a service provider intends to upgrade the FPGA-based end station performance without putting at risk the availability and operability of the system after the upgrade.
- End users, system administrators or service providers are able to reconfigure remotely the FPGA devices of host-based Active Routers within an Active Network, offering new application services or reprogramming the whole network, changing for instance the IP routing tables (Active Node). The context of this

topology implies the need of a monitoring mechanism that secures the constant operation of the Active Router.

- The platform can be used for testing, evaluation and cross-technology integrations since it offers a reliable and failure resistant framework for upgrades of FPGA-based reconfigurable systems (PCI boards).
- In stand-alone mode the platform can also be used for online testing and as a PCI bus protocol checker.
- On-line verification of the PCI protocol for dependable applications which are configured in FPGA-based PC systems.

The research contribution of this thesis is related with a unique hardware monitor and control component (i.e. Error Detection Monitor Unit – EDMU), which is embedded in a FPGA-based reconfigurable PCI board. EDMU delivers concurrent monitoring of application transactions and acts as a “firewall” when predefined policy rules are violated, satisfying by this way the reliability and operability requirements of dependable FPGA-based computer systems.

1.4 Summary

The introductory chapter briefly covered the reasons that inspired this research and also included a short description of terms like monitoring, reconfigurable computing and programmable networks. The implementation limitations and the specific topology requirements were also discussed in this chapter, together with the specifications and the projected target topologies. The second chapter gives an outline of the terminology context related to the research domain. This general theoretical background intends to assist people who are outsiders to the field in order to identify and understand fundamental concepts. At the same time, the information quoted in Chapter 2 aims to transform this thesis to a self-contented research document. The next chapter includes a classification of the relevant literature in different areas: i) Monitoring and error detection, ii) Reconfigurable computing and programmable networking and iii) Remote FPGA reconfiguration.

Chapter four describes the policy framework which decisively shaped the research and development efforts of this work. The security issues and the security framework

that initiates countermeasures for confronting the effect of errors are also analysed. Chapter five introduces the policy framework for the error detection core with specific reference to the particular group of errors that were identified in chapter four. This chapter also includes information about the system architecture as well as design and implementation issues with specific code examples and design details. Chapter six presents the simulation and real time results of the embedded monitor hardware core. Finally, a discussion of the contributions and the potential future enhancements of this research are given in the last Chapter.

The background information of the first two chapters is reasonably extended because the intention throughout the research and development process was to present this work in a self-contained way. In most of the cases the potential target platforms are either networked or they follow strict real time constraints and thus this formulates a practical problem: On the one hand, it is quite difficult for the hardware designers (i.e. FPGA-ASIC design) to comprehend issues related with the requirements and operation of a critical system (i.e. the dependability problems in programmable networks, field upgrade of network reconfigurable systems, security and safety issues etc). On the other hand it is quite difficult for the people who work on software-based critical real time systems to understand hardware concepts related to FPGA design.

CHAPTER 2

Critical real time systems and FPGAs

The purpose of this chapter is to give an overview of the theoretical background aspects that are related to the research goals of this work. Therefore, the reader can familiarise himself/herself with terms that are very useful for the thorough understanding of the design and development concepts that are presented in the following chapters of this dissertation.

2.1 Field Programmable Gate Arrays

The consumer electronic market is offering nowadays a wide range of different Integrated Circuits (ICs), in order to satisfy the different requirements for applications and services (figure 2.1 includes some relevant information).

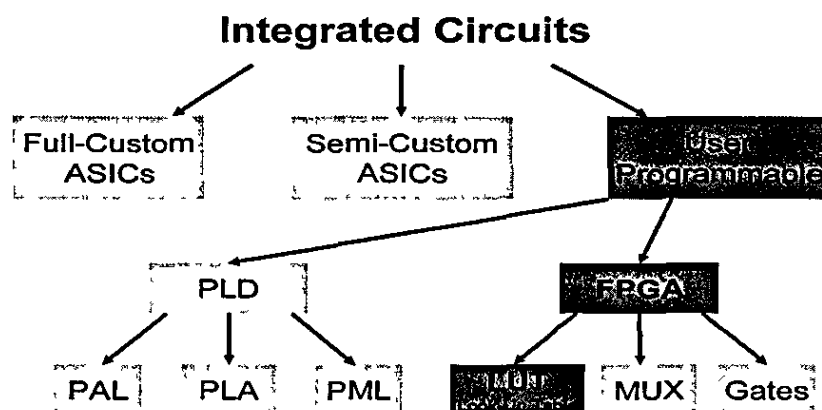


Figure 2.1: The world of Integrated Circuits.

A Field Programmable Gate Array (FPGA) can be configured in the field to implement a desired logic function. Many different architectures exist, but Figure 2.2 shows the basic structure of a typical FPGA: a matrix of configurable logic blocks (CLBs) and interconnection resources surrounded by I/O blocks.

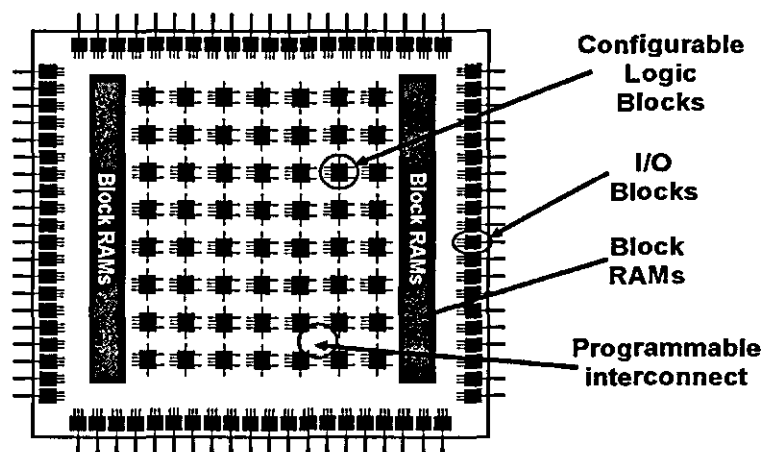


Figure 2.2: A general representation of a SRAM-based FPGA device.

The CLBs (figure 2.3) are often complex but are likely to contain one or more function generators followed by flip-flops:

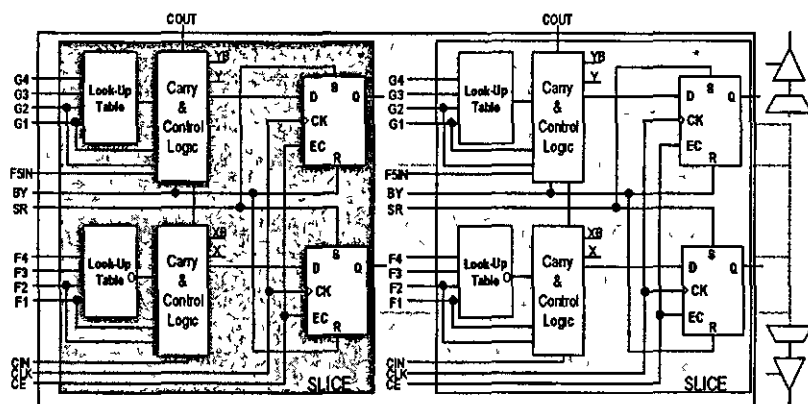


Figure 2.3: The CLB contains two slices. Each slice contains two 4-input look-up tables (LUT), carry & control logic and two registers. There are two 3-state buffers associated with each CLB, that can be accessed by all the outputs of a CLB [10].

Made using either look-up tables (LUTs) or multiplexers, function generators are capable of producing any k-input Boolean function where k is usually four. LUTs are 1-

bit wide memories and essentially store the truth table of the Boolean function they generate. They often can be used for general storage when not acting as a function generator. The output of a function generator can serve as part of combinational logic or can be directed to a flip-flop to create a latched signal. The CLBs provide functional elements for combinatorial and synchronous logic, including basic storage elements.

Interconnection resources are composed of horizontal and vertical wires that can form connections with each other through programmable switches. There are also programmable switches that connect wires to CLBs. I/O blocks can be programmed to allow their associated pin to operate as either an input or an output. Current FPGAs often include additional components such as clock managers, RAM, and dedicated circuitry for common arithmetic operations.

A user can specify a logic function and then use a specific software to map the logic function to a network of CLBs, which are then placed and routed. The end result is a particular configuration for the FPGA that implements the logic function.

While industrial work with reconfigurable hardware has primarily utilized the technology for verification of and sometimes a replacement for ASICs, researchers have explored the use of FPGAs as a means of improving computational performance in scientific and multimedia applications. The appeal of this technology for researchers is that an FPGA can be configured on-the-fly with customized computational circuits that provide hardware acceleration for end applications. Unlike single-purpose ASICs, FPGAs can be reconfigured and re-used for a number of diverse computational tasks. The hardware in such systems can be reconfigured multiple times at run-time in order to adapt to different computing requirements for different applications. Reconfigurable systems offer higher computational density and higher throughput for many applications compared with conventional fixed hardware systems. Reconfigurable computing is an active area of research.

Emerging commercial FPGA architectures are supplementing traditional programmable logic with multiple hardware units for increased design flexibility and computational power. The most attractive feature in these new architectures is the inclusion of high-speed transceivers. The Xilinx Virtex II Pro architecture [11] contains multiple 3.125 Gbps transceivers for use with networks such as 10-Gbps Ethernet and

OC-192c. A second feature of emerging FPGA architectures is the inclusion of powerful embedded processor cores. Altera's Stratix includes the NIOS "soft" embedded microprocessor [12], while Excalibur FPGA devices include a dedicated 32-bit 200 MHz ARM core. Likewise Xilinx includes in the Virtex II pro architectures multiple PowerPC cores. Finally, modern FPGA architectures contain dedicated hardware such as internal SRAM memory and high-speed multiplier array cores to accelerate FPGA computational performance.

The trend to implement entire systems on an FPGA is creating a market for intellectual property 'cores'. These are designs created by third parties, which are sold to FPGA users to incorporate into larger systems. Examples of cores include bus interfaces such as PCI bus, signal processing functions such as Reed Solomon Decoders and communications interface functions such as Serialiser / Deserialiser (SERDES). Leading FPGA manufacturers offer access to a catalogue of cores and customers expect to be able to create a large part of the functionality of their system using cores.

2.1.1 Single Context FPGAs

A single context FPGA is programmed using a serial stream of configuration information. Because only sequential access is supported, any change to a configuration on this type of FPGA requires a complete reprogramming of the entire chip. Although this does simplify the reconfiguration hardware, it does incur a high overhead when only a small part of the configuration memory needs to be changed. This type of FPGA is therefore more suited for applications that can benefit from reconfigurable computing without run-time reconfiguration. Most current commercial FPGAs are of this style. In order to implement run-time reconfiguration using a single context FPGA, the configurations must be grouped into contexts, and each full context is swapped in and out of the FPGA as needed. Because each of these swap operations involve reconfiguring the entire FPGA, a good partitioning of the configurations between contexts is essential in order to minimize the total reconfiguration delay. If all the configurations used within a certain time period are present in the same context, no reconfiguration will be necessary. However, if a number of successive configurations

are each partitioned into different contexts, several reconfigurations will be needed, slowing the operation of the run-time reconfigurable system.

2.1.2 Partially Reconfigurable FPGAs

In some cases, configurations do not occupy the full reconfigurable hardware, or only a part of a configuration requires modification. In both of these situations a partial reconfiguration of the array is required, rather than the full reconfiguration supported by a single context device. In a partially reconfigurable FPGA, the underlying programming bit layer operates like a RAM device. Using addresses to specify the target location of the configuration data allows for selective reconfiguration of the array. Frequently, the undisturbed portions of the array may continue execution, allowing the overlap of computation with reconfiguration. This has the benefit of potentially hiding some of the reconfiguration latency. When configurations do not require the entire area available within the array, a number of different configurations may be loaded into unused areas of the hardware at different times. Since only part of the array is changed at a given point in time, the entire array does not require reprogramming for each incoming configuration. Additionally, some applications require the updating of only a portion of a mapped circuit, while the rest should remain intact. For example, in a filtering operation in signal processing, a set of constant values that change slowly over time may be re-initialized to a new value. But the overall computation in the circuit remains static. Using this selective reconfiguration can greatly reduce the amount of configuration data that must be transferred to the FPGA. Unfortunately, since address information must be supplied with configuration data, the total amount of information transferred to the reconfigurable hardware may be greater than what is required with a single context design. A full reconfiguration of the entire array is therefore slower than with the single context version. However, a partially reconfigurable design is intended for applications in which the size of the configurations is small enough that more than one can fit on the available hardware simultaneously. Plus, the fast configurations methods presented in a previous section can help reduce the configuration data traffic requirements.

2.1.3 Reconfigurable Computing

One common use of a FPGA is for prototyping the design of an ASIC. In this scenario, the FPGA is present only on the prototype hardware and is replaced by the corresponding ASIC in the final production system. However, many system designers are choosing to leave the FPGAs as part of the production hardware. Such systems retain the execution speed of dedicated hardware but also have a great deal of functional flexibility. The logic within the FPGA can be changed if or when it is necessary, which has many advantages. For example, hardware bug fixes and upgrades can be administered as easily as their software counterparts. In order to support a new version of a network protocol, the internal logic of the FPGA can be redesigned sending later the enhancement to the affected customers by email. Once they have downloaded the new logic design to the system and restarted it, they will be able to use the new version of the protocol. This is configurable computing; reconfigurable computing goes one step further.

Reconfigurable computing involves manipulation of the logic within the FPGA at run-time. In other words, the design of the hardware may change in response to the demands placed upon the system while it is running. Here, the FPGA acts as an execution engine for a variety of different hardware functions — some executing in parallel, others in serial — much as a CPU acts as an execution engine for a variety of software threads. FPGA could be considered as a reconfigurable processing unit. One theoretical application is a smart cellular phone that supports multiple communication and data protocols, though just one a time. When the phone passes from a geographic region that is served by one protocol into a region that is served by another, the hardware is automatically reconfigured. Reconfigurable computing has the following advantages:

- **Performance:** Configurable computing can be highly optimised for a specific data set or specific applications.
- **Cost Effectiveness:** Configurable computing can be used to reduce system costs through *hardware reuse*. Any new features or can be easily updated to the system and due to the field programmability, any problems with the design can be corrected without any changes to the on board resources.

- **System prototyping:** Large systems (either ASICs or boards) can be built on single or multiple FPGAs that makes system testing and debugging more easier than testing the real system and even cheaper.
- **Custom I/O:** FPGAs provides a flexible set of programmable I/O signals. That gives the designer the opportunity to reuse existing hardware and to add new changes to it easily. New FPGAs supports Different types of IOs levels and standards. One important use of this feature is matching different families.
- **System density:** Configurable computing “specially Run-time configurable logic” increases the system density and can deliver functionality of the device many times more than its size by dynamically reconfiguring the system and increase resource utilization through loading and unloading different system modules.
- **Fault-Tolerance:** System reliability can be increased through redundancy by duplication or building some testing circuits. Dynamic reconfiguration technique can destroy old circuits and build new ones after detecting any error.

2.2 Integration of programmable logic devices in critical systems

The spread of programmable reconfigurable devices in various different technology applications domains, targeting either consumer electronics or electronics used from service providers, scientific organisations and state bodies have made their use highly important. The constant increase in the integration of reprogrammable chips in systems has enforced the inter-dependence relation for the services and the applications that these systems deliver. There is an intensive research in a world wide scale that tries to address the challenges that FPGA (or similar reprogrammable devices) pose because of their high penetration to critical systems. There are several conceptual requirements for such systems related with terms like reliability, availability, dependability, testability, maintainability and operability. These terms could be referred as *critical system requirements*. The different critical computing systems have different levels of requirements for the above mentioned features. Critical computing systems are found in air transport systems, in the rail industry, in space engineering, in medical equipment, in information systems (i.e. servers, routers, switches), in power and nuclear energy systems, in military systems etc.

All computer systems are subject to failures due to different types of reasons. An indicative example is the fact that typically more than half of the errors in a system are due to ambiguous or incomplete requirement specifications [13], [14], [15]. A major issue also for the industry is the revenue loss per hour of system downtime in various business segments. Table 2.1 shows that an hour of downtime per year is extremely expensive, regardless of the industry segment. Clearly reliability is a huge asset but at the same time it is a concern that needs to be addressed. Another indicator that translates system downtime is given in Table 2.2. Current systems are being designed for minimal downtime of 5 minutes or less per year. Critical computing systems are considered vital to be maintained and must satisfy some or all of the previously mentioned *critical system requirements*.

Industry Sector	Loss Revenue per Hour
Energy	\$2.8 million
Telecommunications	\$2.0 million
Manufacturing	\$1.6 million
Financial Institutions	\$1.4 million
Information technology	\$1.3 million
Insurance	\$1.2 million
Retail	\$1.1 million
Pharmaceuticals	\$1.0 million
Banking	\$996,000

Table 2.1: Revenue loss per hour, by business segment [16].

System Availability (%)	Yearly Downtime
95	438 hours
99.0	88 hours
99.5	44 hours
99.9	8.8 hours
99.95	4 hours
99.99	53 minutes
99.9995	5.3 minutes
99.9999	32 seconds
99.99999	3.2 seconds

Table 2.2 Percentage Availability and Downtime [16].

To avoid confusion between “reliability” as a precisely defined statistical measure [17] and “reliability” as a qualitative attribute of systems and computations, use of

“dependability” was proposed to convey the second meaning [18], resulting in the name “dependable computing”. Thus, dependable computing deals with impairments to dependability (defects, faults, errors, malfunctions, degradations, failures, and crashes), means for coping with them (fault avoidance, fault/error tolerance, design validation, failure confinement, monitoring etc.), and measures of success in designing dependable computer systems (reliability, availability, performability, safety, security etc.).

A computer system can have three different types of assessment depending on the interest of the external viewer. The maintainer’s external view consists of a set of interacting subsystems that must be monitored for detecting possible malfunctions in order to reconfigure the system or, alternatively, to guard against hazardous consequences (such as total system loss or crash). The operator’s external view consists of a black box capable of providing certain services and is more abstract than the maintainer’s system-level view. Finally, the end user’s external view is shaped by the system’s reaction to particular situations or requests.

A deliberate or unintended error (e.g. hardware bug, misuse of local services etc) results in information errors within a system. Consequently, the system may malfunction resulting in service degradation, which finally can result in system failure. A five-level view of the impairments to dependability could be deducted according to this approach (figure 2.4).

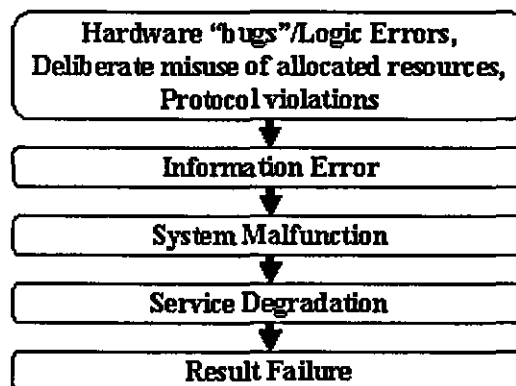


Figure 2.4: The sequence steps that result in system failure.

Ensuring the reliability of critical computing and electronic systems has always been a challenge. As the complexity of systems increases the inclusion of reliability

measures becomes progressively more complex and often a necessity for VLSI circuits where a single error could potentially render an entire system useless.

2.3 Embedded Real-time critical systems

An embedded system is typically a product which includes a computing system. The product is said to "embed" the computing system inside. Embedded systems do not necessarily look like computers, however it is typical that embedded systems interact with their environment. For instance, a mobile phone is regarded an embedded system: it reacts on incoming calls, user input, cell roaming, etc.

A real-time system is a system that interacts with its environment in a time constrained manner. The real-time system must produce results within specified time limits. A computation result (or actuation) must be delivered neither too late, nor too early. The criticality of violated timing constraints, or missed execution deadlines, classifies real-time systems into hard or soft real-time systems [19]. Timing failures in a hard real-time system are considered hazardous and very critical and should never be allowed. Examples on hard real time requirements can be found in automotive and avionic systems, medical equipment, military systems, energy and nuclear plant control systems. On the contrary, the requirements in soft real-time systems are not so critical and may tolerate timing constraint violations, either by discarding the produced results or by allowing a degraded quality. Soft real-time requirements can be found in telecommunication systems, audio and video applications, streaming media, airline reservation systems, etc.

A typical real-time system consists of a controlling subsystem (the computer), and the controlled subsystem (the physical environment). The interactions between the two subsystems can be described by three main operations:

- Sampling
- Processing
- Responding

The computer subsystem samples data from the physical environment. Sampled data is then immediately processed by the computer subsystem, and a proper response is sent to the physical environment. All three operations must be performed within the required

timing constraints. For example, it is imperative that an air bag control system in an automobile responds within set timing constraints in the event of a crash. The response must neither be too late (being non-effective), or too early (risking hazardous manoeuvring of the car).

This section includes further evidence for the importance of critical systems by examining the timing-related restrictions as an indicative property of reliability, dependability and availability. Because of the central role of time constraints in critical real-time systems, these systems are classified according to the type of time constraints. The first class of critical real-time systems is the soft real-time systems. This class covers those systems where a miss of a deadline results in degraded performance, and/or the increasing of costs. The second class is the firm real-time systems. In these systems, the miss of a deadline leads to the result becoming worthless. An example of such a system is a weather forecast system, where the result becomes void no later than when the forecasting horizon has passed. The hard real-time systems build the third and last category. A miss of a deadline here is followed by more or less catastrophic consequences. The consequence is usually loss of life or, at the very least a serious amount of money. This usually regards X-by-wire systems in vehicles and planes, or simple things like the destruction of a machine and the following halt of a complete production line. Line 3 in Figure 2.5 shows the enormous jump in costs while line 1 and 2 show a more controllable loss of cost. A deadline violation is considered fatal and cannot be tolerated especially in dependable systems.

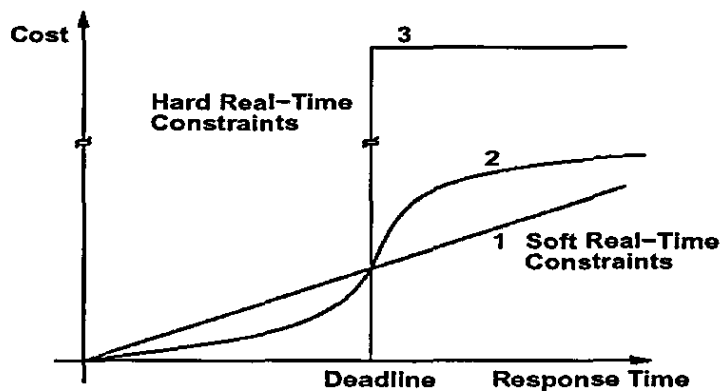


Figure 2.5: Cost function of real time tasks [20].

2.4 Faults, errors and failures

A very important design requirement in dependable systems is to define the sequence that creates system failures. A typical sequence of this kind is shown in figure 2.6.

Fault → Error → Failure → Fault ...→

Figure 2.6: Cause consequence diagram of fault, error and failure.

Definition: A failure is the non-performance or inability of the system or component to perform its intended function for a specified time under specified conditions [21]. That is, an input, X, to the component, O, yields an output, O(X), non-compliant with the specification.

Definition: An error is a design imperfection, or a deviation from a desired or intended state [21]. A program can be viewed as a state machine, whereas an error (bug) is an unwanted state. An error can be also thought as a corrupted data state, caused by the execution of an error (bug) but also due to e.g., physical electromagnetic radiation.

Definition: A fault is the adjudged (hypothesized) cause for an error [22]. Generally a failure is a fault, but not vice versa, since a fault does not necessarily lead to a failure.

2.4.1 Systematic and physical failures

Failures are usually divided into two categories:

- I. *Systematic failures* which are caused by specification or design flaws, i.e., behaviours that do not comply with the goals of the intended, designed and constructed system. Examples of contributing causes are erroneous, ambiguous, or incomplete specifications, as well as incorrect assumptions about the target environment. Other examples are failures caused by design and implementation faults. Wear, degradation, corrosion, etc. do not cause these types of failures, all errors are built in from the beginning, and no new errors will be added after deployment.

A systematic failure occurs if and only if:

- 1) the location of an error is executed in the program,

- 2) the execution of the error leads to an erroneous state, and
- 3) the erroneous state is propagated to the output.

This means, that if an error is not executed it will not cause a failure. If the effect of the execution of the error (infection) is indistinguishable from a correct system state it will not cause a failure. If the system's state is infected but not propagated to the output there will be no failure.

II. *Physical failures* which are the result of a violation upon the original design. Environmental disturbances, wear or degradation over time may cause such failures. Examples, are electromagnetic interference, alpha and beta radiation, etc.

A physical failure occurs if and only if:

- 1) the system state is corrupted or infected, and
- 2) the erroneous state is propagated to the output.

Fault-tolerance mechanisms usually try to prevent (1) by applying robust designs, and (2) by applying redundancy, etc.

2.4.2 Failure modes

Depending on the architecture of the system, different degrees and classes, of failure behaviour could be assumed. Components can fail in different ways and the manner in which they fail can be categorized into failure modes. The failure modes are defined through the effects, as perceived by the component user. Several categories will be presented i.e. failure modes, (1 to 6) ranging from failure behaviour that sequential programs, or single tasks in solitude, can experience, to failure behaviour that is only significant in multitasking, distributed systems and real-time systems, where more than one task is competing for the same resources (e.g. processing power, memory, computer network, etc).

Failure modes:

1. *Sequential failure* behaviour:

- Control failures, e.g., selecting the wrong branch in an if-then-else statement.
- Value failures, e.g., assigning an incorrect value to a correct (intended) variable.
- Addressing failures, e.g., assigning a correct (intended) value to an incorrect variable.

- Termination failures, e.g., a loop statement failing to complete because the termination condition is never satisfied.
 - Input failures, e.g., receiving an (undetected) erroneous value from a sensor.
2. *Ordering failures*, e.g., violations of precedence relations or mutual exclusion relations.
 3. *Synchronization failures*, i.e., ordering failures but also deadlocks.
 4. *Interleaving failures*, e.g., unwanted side effects caused by non-re-entrant code, and shared data, in pre-emptively scheduled systems.
 5. *Timing failures*. This failure mode yields a correct result (value), although the procurement of the result is time-wise incorrect. For example, deadline violations, too early start of task, incorrect period time, too much jitter, too many interrupts (too short inter-arrival time between consecutive interrupt occurrences), etc.
 6. *Byzantine and arbitrary failures*. This failure mode is characterized by a non assumption, meaning that there is no restriction what so ever with respect to which effects the component user may perceive. Therefore, the failure mode has been called malicious or fail-uncontrolled. This failure mode includes two-faced behaviour, i e. a component can output "X is true" to one component user, and "X is false" to another component user.

The above listed failure modes build up a hierarchy where Byzantine failures are based on the weakest assumption [23] (a non-assumption) on the behaviour of the components and the infrastructure, and sequential failures are based on the strongest assumptions. Hence, Byzantine failures are the most severe and sequential failures the least severe failure mode. The Byzantine failure mode covers all failures classified as timing failures, which in turn covers synchronization failures, and so on (Figure 2.7).

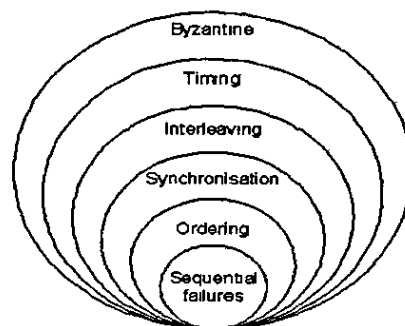


Figure 2 7: The relation between the failure modes.

The component user can also characterize the failure modes according to the viewpoints domain. A distinction can be made between primary failures, secondary failures and command failures [21]:

Primary failures: A primary failure is caused by an error in the software of the component so that its output does not meet the specification. This class includes sequential and Byzantine failure modes, excluding sequential input failures.

Secondary failures: A secondary failure occurs when the input to a component does not comply with the specification. This can happen when the component is used in an environment for which it is not designed, or when the output of a preceding task does not comply with the specifications of a succeeding task's input. This class includes interleaving failures, sequential input failure modes, as well as changed failure mode assumptions.

Command failures: Command failures occur when a component delivers the correct result but at the wrong time or in the wrong order. This class covers timing failures, synchronization failures, ordering failures, as well as sequential termination failures.

2.5 Security issues

The philosophy of opening the network to be programmable raises many administrative issues. In particular, safety and security are points of increasing concern. Programmable networks allow system administrators and router vendors to deploy well-tested router software that provides the required service functionality. Such a model might later migrate to a fully open programming platform as proposed by the active networking community.

The word security suggests protection against malicious attack by outsiders. Security also involves controlling the effects of errors and equipment failures. Anything that can protect against a deliberate, intelligent, calculated attack will probably prevent random misfortune as well. Before going into specifics (Chapter 4), it will be very helpful to understand the some basic concepts that are essential to any security system.

2.5.1 The security challenges

Know your “enemy”

Architecting platforms based on reconfigurable programmable logic devices demands a careful consideration of who might want to circumvent the security measures and identify also the motivations that are hidden behind such actions. Determine what the attackers or intruders might want to do and the damage that they could cause to the network as well as to the network components and services. Security measures can never make it impossible for a user to perform unauthorized tasks with a computer system. They can only make it harder. The goal is to make sure the network security controls are beyond the attacker's ability or motivation.

Security cost

Security measures almost always reduce convenience, especially for sophisticated users. Security can delay work and create expensive administrative and educational overheads. It can use significant computing resources and require dedicated hardware. Designing security measures involves understanding of the costs and weigh of the costs against the potential benefits. In order to achieve that, there must be a thorough understanding of the costs and the measures themselves and the costs and likelihoods of security breaches. Incurring security costs out of proportion of the actual dangers, results in a disservice.

Identify the assumptions

Every security system has underlying assumptions. For example, one might assume that the network is not tapped, or that attackers know less than you do, that they are using standard software. A good practice is to examine and justify all the assumptions. Any hidden assumption is a potential security hole. Another important consideration is to understand the areas that need to be protected. Security systems should be designed so that only a limited number of secrets need to be kept. Many security procedures fail because their designers do not consider how users will react to them. If the security

measures interfere with essential use of the system, those measures will be resisted and perhaps circumvented.

Security weaknesses

Every security system has vulnerabilities. A programmable logic system designer should be aware of the platform's weak points and also know how these could be exploited. The areas that present the largest danger should be secured in a more robust way in order to prevent access to them. Understanding the weak points is the first step toward turning them into secure areas. Therefore, appropriate barriers should be created inside the system so that if intruders access one part of the system, they do not automatically have access to the rest of the system. The security of a system is only as good as the weakest security level of any single host in the system. Understanding how a system normally functions, knowing what is expected and what is unexpected, and being familiar with how devices are usually used, helps to detect security problems. Unusual events can help to identify intruders before they can damage the system.

Security is pervasive

Almost any change that is being made in a system may have security effects. This is especially true when new services are created. Administrators, programmers, and users should consider the security implications of every change they make. Understanding the security implications of a change is something that takes practice. It requires lateral thinking and a willingness to explore every way in which a service could potentially be manipulated.

2.5.2 Embedded systems design and security issues

In the past, embedded systems tended to perform one or a few fixed functions. The trend is for embedded systems to perform multiple functions and also to provide the ability to configure new hardware and software in order to implement new or updated applications in the field, rather than only in the more controlled environment of the factory. While this certainly increases the flexibility and useful lifetime of an embedded system, it poses new challenges; embedded systems often provide critical functions that

could be sabotaged by malicious entities. An embedded system should ideally provide required security functions, implement them efficiently and also defend against attacks by malicious parties. It is also important to note that there are many entities involved in a typical embedded system manufacturing, supply, and usage chain. Security requirements vary depending on the different perspectives that are considered (i.e. users, service providers, administrators, companies, government services).

Embedded systems, which will be ubiquitously used to capture, store, manipulate, and access data of a sensitive nature, pose several unique and interesting security challenges. Security is often misconstrued by embedded system designers as the addition of features, such as specific cryptographic algorithms and security protocols, to the system. In reality, it is an entirely new metric that designers should consider throughout the design process, along with other metrics such as cost, performance, and power. Several security processing architectures targeting embedded systems, (presented in Chapter three) implement basic security functions like confidentiality, integrity and authentication, but do not provide protection from software or physical attacks by malicious entities. Furthermore, architectural features that accelerate cryptography and security protocols do not protect against Denial-of-Service (DoS) attacks. A secure embedded system should also incorporate appropriate attack-resistant features. Figure 2.8 is a graphic synopsis of the issues stated in the previous sections:

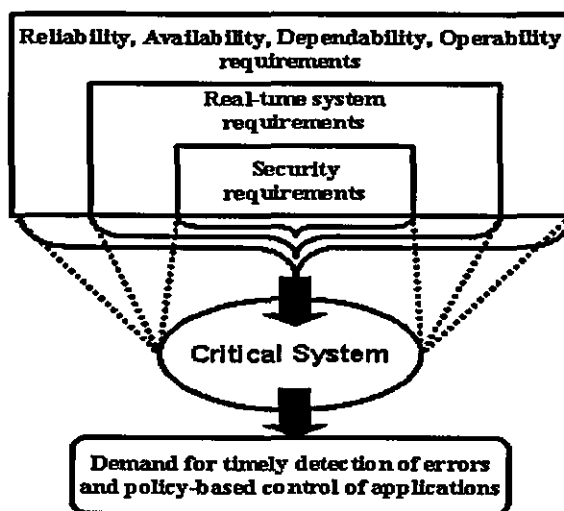


Figure 2.8: Critical systems and the inherent design requirements.

The requirements quoted in each rectangle shape can be either separate design demands of a critical system or it can include a combination of the nested rectangles (i.e. reliability may or may not require security).

2.6 Testing debugging and monitoring

This section includes useful background information for the usefulness of testing, debugging and monitoring in real time systems. Special interest exists for the monitoring function of critical real-time computer systems.

2.6.1 Testing and debugging

Debugging is defined as “the process of locating, analysing, and correcting suspected faults” [24]. A fault is defined to be the direct cause of some error. Since the occurrence of errors can have different reasons, they are usually not predictable, and therefore they must be located them using debuggers. A debugger is a tool which helps the designer to examine suspected errors in a program, and eventually also remedy the errors. Cyclical debugging is commonly used to describe debugging as an iterative process, in which the debugger is repeatedly used to find and correct errors, until no more errors can be found.

Testing and debugging are similar activities with respect to finding errors. However, testing is more of an automated process of exposing different input to the system under test, and evaluating its results (output). The objective is to find input data, or patterns of data, that cause erroneous results [25]. The faults that are found during the testing process are then put under observation in a debugger.

In real-time systems, errors may also occur in the time-domain. Real-time systems are therefore harder to debug than non-real-time systems. The ability to track down timing-related errors was largely an unexplored area until the early 1980's. Today, various debugging systems and methods have been developed in order to address timing-related issues [26], [27].

2.6.2 Monitoring

Monitoring is the process of gathering information about a system [28], [29]. This information cannot normally be obtained by studying the program code only. The collected information may be used for program testing, debugging, task scheduling analysis, resource dimensioning, performance analysis, fine-tuning and optimisation of algorithms. Monitoring is also important in state-of-the-art critical computing systems in order to control the results of errors and prevent system failures. The applicability of monitoring is wide, and so is the spectrum of available monitoring techniques. This section includes a generic presentation of a monitor and describes different monitoring systems, the type of information collected by monitors, and the problem-related issues with monitoring.

In essence, a monitor works in two steps: detection (or triggering) and recording. The first operation refers to the process of detecting the object of interest. This is usually performed by a trigger object that is inserted in the system; the object indicates an event of interest for recording when it is executed or activated. Recording, is the process of collecting events and saving them in buffer memory, or transfer them to external computer systems for the purpose of further analysis or debugging. An event is a record of information which usually constitutes the object of interest together with some additional meta data regarding that object (e.g. the time when the object was recorded, the object's source address, task/process ID, etc.). The types of monitored objects depend on the level of abstraction which the user is interested in. The trigger object can be inserted in the target hardware platform as an instruction, a function, or a module that is formed by several signal assertions. It may also be a physical sensor, or probe, connected with physical wires in the hardware, such as CPU address, data, and control busses.

An important issue regarding the monitoring process is the amount of execution interference that may be introduced in the observed system due to the involved operations of a monitor. This execution interference, or perturbation, is unwanted because it may alter the true behaviour of the observed system, in particular such systems that are inherently timing-sensitive such as real-time and distributed systems.

2.6.3 Types of monitoring systems

Monitoring systems for system-level analysis are typically classified into three types: 1) software monitoring systems, 2) hardware monitoring systems, and 3) hybrid monitoring systems. A short description of each type of monitoring system follows.

Software Monitoring Systems

In this category of monitoring systems, only software is used to instrument, record, and collect information about software execution. Software monitoring systems offer the cheapest and most flexible solution where a common technique is to insert instrumentation code at interesting points in the target software. When the instrumentation code is executed the monitoring process is triggered and information of interest is captured into trace buffers in target system memory. The drawback of instrumentation is the utilisation of target resources such as memory space and processor execution time.

Hardware Monitoring Systems

In this category of monitoring systems, only hardware (custom or general) is used to perform detection, recording and collection of information regarding the system. The primary objective of hardware monitoring is to avoid, or at least minimize, interference with the execution of the target system. A hardware monitoring system is typically separated from the target system, and thus, it does not use any of the target system's resources. The target system is monitored using passive hardware (or probes) connected to the system busses and signals. Hardware monitoring is especially useful for monitoring real-time and distributed systems since changes in the program execution time are avoided.

In general, the operation of monitoring hardware can be described by the three steps (see Figure 2.9): event detection, event matching, and event collection.

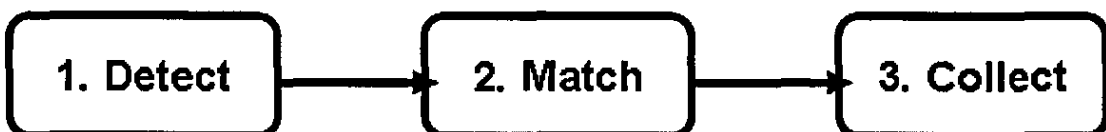


Figure 2.9: Hardware monitoring steps.

In the first step the hardware monitor listens continuously to the signals. In the second step, the signal samples are compared with identified patterns that define the system events. When a sample matches an event-pattern, the process triggers the final step, collection, where the sampled data is collected and saved. The saved samples may be stored locally in the monitoring hardware, or be transferred to a host computer system where usually more storage capacity can be obtained.

This type of hardware monitor avoids target interference and has the advantage of precision and accuracy. Since the sole duty of a hardware monitor is to perform monitoring activities (usually at equal or higher system speed than the target's) the risks of losing samples are minimized. A disadvantage of hardware monitors is their dependency on the target's architecture. The hardware interfaces, and the interpretation of the monitored data must be tailored for each target architecture. Thus, monitoring solutions using hardware are more expensive than software alternatives. Moreover, a hardware monitor may not be available for a particular target, or takes time to customize, which may increase the costs further in terms of delayed development time. Another problem with hardware is the integration and miniaturisation of components and signals in today's chips which renders difficulties in reaching information of interest, e.g. cache-memory, internal registers and busses, and other on-chip logic. To route all internal signals out from a chip may be impossible because of limited pin counts.

In general, hardware monitoring is used to monitor either hardware devices or software modules. Monitoring hardware devices can be useful in performance analysis and finding bottlenecks in e.g. caches (accesses/misses), memory latency, CPU execution time, I/O requests and responses, interrupt latency, etc. Software is generally monitored for debugging purposes or to examine bottlenecks, load-balancing (degree of parallelism in concurrent and multiprocessor systems), and deadlocks. The current trend of making application specific hardware using FPGAs and VHDL (an early example of this type is given in [30]) gives an opportunity to conveniently integrate non-intrusive monitoring mechanisms in the hardware for single node systems.

Hybrid Monitoring Systems

Hybrid monitoring uses a combination of software and hardware monitoring and is typically used to reduce the impact of software instrumentation alone. A hardware monitor device is usually attached to the system in some way, e.g. to a processor's address/data bus, or on a network, and is made accessible for instrumentation code that is inserted in the software. The instrumentation is typically realised as code that extracts the information of interest, e.g. variable data, function parameters, etc., which is then sent to the monitor hardware. For instance, if the monitor hardware has memory-mapped registers in the system, the instrumentation would perform data store operations on the monitor's memory-addresses. The hardware then proceeds with event processing, filtering, time-stamping, etc., and then communicates the collected events to an external computer system. This latter part typically resembles the operation of a pure hardware monitor. The insertion of instrumentation code also resembles the technique used in a software monitoring system; i.e. it can either be done manually by the programmer, automated by a monitoring control application or by compiler directives.

The complexity of the capabilities of a monitor varies vastly from detection of a simple combination of a number of input signals, the combination of input signals to numerical values and tracking of the, for example, less, greater, between limits type, to the congregation of rather complex signal patterns over time. In older systems the hardware monitors were basically plug boards, or a combination of probes and bus analyzers. The idea of using embedded programmable logic devices like FPGAs for monitoring and controlling critical real time systems is introduced in this research work. Hardware monitors are often built on the basis of personal computers. This frees the developer of a hardware monitor from having to design the basic system which usually consists, besides other less important things, of a user interface, storage facilities and memory space (i.e. series of registers).

Figure 2.10 is a representative way to show the impact of the different monitoring schemes during the time that the execution of the application is suspended. The hardware monitor does not introduce such an impact, whereas hybrid monitors generate a rather small suspension time on the application software and software monitors suffer

considerably from this effect. It should be noted that the actual suspension time is strongly correlated to the amount of data gathered in the software part of the monitor. The arguments so far manifest the need of a concurrent hardware monitor embedded in the target architecture for meeting the reliability, dependability and maintainability requirements of a real time system.

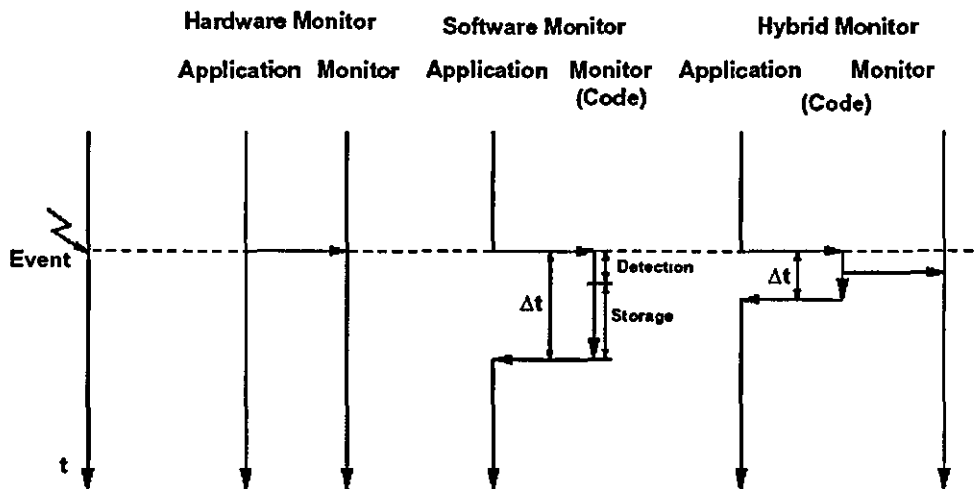


Figure 2.10: Temporal impact of monitor types [31].

With today's highly integrated hardware, encapsulating complete systems on a chip (SoC), the traditional hardware monitors (i.e. hardware probes) are facing severe difficulties. Processor cores, I/O components, cache memories, and even standard memory, are all integrated on the same reprogrammable chip (i.e. FPGA). Given also that chip packages can be obstructive (as in Ball-Grid Array packages) and have limited pins, it has become almost impossible for external hardware to probe internal signals. For real-time systems built on these premises there is a need to access execution information residing on-chip, as well as to avoid interference with the system's execution behaviour.

Observability into SoC designs is today mostly supported for RTL verification. As SoC designs tend to increase in size and complexity, the verification process need also to take place at the system-level. Support for system-level verification already exists for the software part of a SoC, where a common technique is the use of software monitors. These monitors are typically found in RTOS (Real-Time Operating System)

development tool environments, and provide the developer with process-level information such as task-scheduling events start/stop, block/resume, taskswitch, etc.), inter-process communication events (e.g. send and receive messages), synchronisation and resource utilization (CPUs, semaphores, I/O), external interrupts, etc. On the other hand, the available commercial or academic on-chip monitoring solutions commonly operate with a user interface and a software API, giving the opportunity to the developer to debug the design and achieve a satisfactory degree of robustness. However, the observability of systems is not limited to debugging, testing and validation purposes.

The outcome of the research regarding embedded monitoring support of dependable systems is evident of the fact that the available time for reacting to potential threats (i.e. hardware application bugs, misuse of vital system resources and protocol violations) exercised within such systems is considerably limited. Software monitoring and control solution are therefore excluded in this context because of the significant delay that they introduce in their response (see also figure 2.10). The system/application monitoring function, the error detection and the applied control have to operate in a timely, if not concurrent, manner. This attributes can be satisfied with an embedded, passive hardware monitor that is built with the help of an HDL language and it is inserted/integrated on a permanent basis in the target critical system topology.

2.7 Active Networks

Active Networks [32] are a relatively new concept, where a network is not just a passive carrier of bits but a more general computation model. Active Network may be simplistically viewed as a set of "Active Nodes" that perform customized operations on the data flowing through them. Traditional data networks provide a transport mechanism to transfer bits from one end system to another, with a minimal amount of computation (e.g., header processing and signalling). In contrast to that, a programmable network consists of programmable routers that have general-purpose processing units in their data path. These processing units can be programmed to perform various protocol operations as well as complex payload processing. Unlike traditional routers, deployment of new protocols can be achieved by reprogramming the

system rather than exchanging expensive hardware. This programmability of the data plane extends the traditional store-and-forward paradigm of routers to store process-and-forward. The processing step is where interesting new services and protocols can be integrated into the network.

Active networks not only allow the network nodes to perform computations on the data but also allow their users to inject customized programs into the nodes of the network, that may modify, store or redirect the user data flowing through the network. These programmable networks open many new doors for possible applications that were unimaginable with traditional data networks. For example, there may be a video multicast session where at every node the video compression scheme is modified, based on the computation done by that node and depending on the network bandwidth available. The research community has realized the potential of Active Networking and a lot of work is underway at different research sites.

Various approaches on solving current networking problems use a part of the Active Networking concept. Applications including packet filtering [33] in firewalls (also routers) where the filters in the firewall decide which packet should go through and which should be blocked. Other examples would be congestion [34], web proxies [35], multicast routers [36] and video gateways [37] etc. that perform user driven computations "within" the network. For example web proxies provide a user transparent service to serve and cache web pages. Nomadic Routers determines the means by which a host is connected to a network (e.g., a modem or a high speed network connection) and adapt to the conditions. For example it might perform link compression and perform more file caching when connected through a modem. Also, it would enable for example encryption when user is remotely connected. The list goes on and on. A lot of recent trends and developments in networking are a subset of the Active Network Architecture.

2.7.1 Active Network usage

From the point of view of a network service provider, active networks have the potential to reduce the time required to develop and deploy new network services. The shared infrastructure of the network presently evolves at a much slower rate than other

computing technology. One consequence of being able to change the behaviour of network nodes on the fly is that service providers would be able to deploy new services quickly, without going through a lengthy standardization process.

At a finer level of granularity, active networks might enable users or third parties to create and tailor services to their particular applications and even to current network conditions. Though it seems likely that most end users would not write programs for the network it is easy to imagine individuals customizing services by choosing options in code provided by third parties. Indeed, this prospect should appeal to network providers as well, because it enables them to charge more for such value added services.

Networks are expensive to deploy and administer. For researchers, a dynamically programmable network offers a platform for experimenting with new network services and features on a realistic scale without disrupting regular network service.

2.7.2 Basic concepts

An active network is a kind of store-and-forward network. A store-and-forward network consists of a set of nodes interconnected by transmission links. The purpose of the network is to support the sharing of these transmission facilities. The basic unit of multiplexing of transmission facilities is the packet. Nodes receive packets from users and other nodes, perform a computation based on their internal state and the control information (header) carried in the packet, and as a result of that computation may forward one or more packets toward other nodes or to users. The nature of the network service is defined by the behaviour of the individual nodes of the network, and how users can control that behaviour through coded information placed in their packets. In today's Internet, for example, routers examine the destination address field of the Internet Protocol header along with internal routing tables to determine to which neighbour they should forward the IP packet. The extent of user control over the network's behaviour is limited to the range of values that can be placed in that field (and a few others) in the IP header; other services can, however, be envisioned. One example, which has been proposed for the Internet is a "premium" service in which certain nodes classify packets based on information contained in all of their headers

(TCP or UDP port numbers as well as source and destination IP addresses) and then route and schedule them for transmission on the basis of that classification. In this case, it has to be identified the term user (or end user) with the originators and recipients of the packets actually carried by the network. Users are, in general, different from node/network administrations, which control the configuration and interconnection of network nodes. Often the relationship between user and network administration is that of service provider and subscriber.

2.7.3 Active Networks and Programming Interfaces

One way to think about active networks is that they provide a programmable network API. The IP header in the traditional network could be considered as the input data to a virtual machine, whereas packets in the active network containing programs could be considered as input data. In the context of this model, a variety of active networking approaches can be characterized by the following attributes:

Language Expressive Power: The degree of programmability of the network API may range from a simple list of fixed size parameters that select from predefined sets of choices, to a Turing-complete language capable of describing any effective computation. The advantage of a less powerful language is that it constrains the possible node behaviours and so simplifies correctness analysis. It is also more likely to admit fast-path optimization, e.g. through special purpose hardware. Many active networking projects, however, have opted for more powerful languages that use typing and other mechanisms to help ensure correctness. Most of these languages feature some form of restriction on their expressive power, order to guarantee that the effect of any packet sent into the network is bounded. For example, a language may admit only straight-line programs, without loops or branches.

Statefulness: Another important characteristic of the network API is the ability to install state in the interior nodes of the network, and to refer to state installed by other packets. Some active network APIs provide this capability, while others do not. Where it is present, the API must include control mechanisms to protect users' state from unauthorized access.

Granularity of Control: This mainly refers to the scope of node behaviour that can be modified by a received packet. One possibility is that a single packet can modify the node behaviour seen by all packets arriving at the node, and this change persists until it is overridden. At the other extreme, a single packet modifies the behaviour seen only by that one packet. Between these extremes, modifications might apply to a flow, which could be defined to be a set of packets sharing some common characteristic, such as temporal locality and/or a particular source and destination address in the headers. In general, the active network API must include security mechanisms that ensure that packets affecting the node behaviour have localized effect and/or come from authorized users.

2.7.4 Using existing network protocols and technologies

The fundamental idea of Active Networks is to utilise the current infrastructure of IP-based networks (e.g. the Internet), without having the need to change a part of the network, nor interrupt its operation nor evolve its current technology. Active Networks do not require any changes in the standard communications protocols used in the IP-based networks. Active Networks offer added services and programmability either to the network nodes or to the Active end-stations of the currently available IP-based networks. This is accomplished through a transparent operation on top of the current IP networks infrastructure, utilising the current features of the communication protocols. Considering the above concept, the reconfigurable hardware platform can be smoothly integrated in a host-based router and serve the Active Applications.

2.7.5 Active Networks based on PC routers

The ability to not only forward packets through a network, but also process them on a router, is the key to implementing new services and protocols without changing the underlying hardware infrastructure. Such processing can range from simple routing and queuing decisions to complex payload modifications. Performing such processing in software rather than custom logic opens the possibility to adapt and deploy new services by simple changes in the software. The crucial challenge in such a system is not only to be able to provide the functionality to dynamically change the forwarding

loop for selected data streams. It is equally important that the performance of the system be comparable to custom logic solutions despite the fact that software.

General-purpose workstation processors that perform packet routing and forwarding in software were common router configurations in the 1980's. Typical link speeds of a few kilobits per second did not exceed the processing power of such a system. As performance demands for communication changed in the 1990's towards link speeds of several megabits per second, software-based routers were not able to keep up with this trend. As a result, ASIC-based routers were developed to provide basic protocol processing functionality at high speeds. The majority of current Internet routers are still ASIC-based. Their limitations in supporting new protocols led to efforts to re-introduce the flexibility and extensibility of software-based systems.

General-purpose processing as part of the data path and deployment of processing code via the packet itself was initially proposed by Tennenhouse and Wetherall in 1996 [38]. From this idea, numerous research projects have spawned to develop infrastructure that can process packets in the data path and dynamically deploy the processing code. The majority of these projects were and are aimed at investigating and implementing software-based "active routers". Key questions are how to dynamically install protocol-processing code in the data path, how to deploy code modules, and how to safely and securely execute arbitrary code on a router.

High-end routers are designed around special-purpose hardware: they use a switched interconnect (e.g., a crossbar) rather than a shared bus, and they implement the performance-critical forwarding function in hardware (e.g., an ASIC) rather than software. As a consequence of this hardware-intensive design, high-end switches are able to forward packets at a very high rate (on the order of 1–10Mpps) and scale to fairly large sizes (on the order of 128 100Mbps ports), but they provide virtually no programmability. In addition to the highly optimised forwarding path, high-end routers also have a network processor—usually connected to the one of the switch's I/O ports—that handles exceptional cases like IP options and routing updates. Low-end, routers can be built using a standard workstation running a conventional operating system. The host-based forwarding [39] is implemented with the use of the general-purpose computers with two or more network interfaces to act as a router. Because they

are implemented in software, such routers could be programmed to support nearly every imaginable function, but they offer very limited performance; they can forward 10-100Kpps and support only a handful of 100Mbps ports. These routers offer certain advantages over the use of a high end router, allowing open, public source code access to the forwarding, queuing, and routing algorithms, and the use of more flexible, commodity host interfaces and host CPUs.

2.7.6 Active Networks processing requirements

As already stated, the Active Networks are packet-switched networks in which the packets can contain code fragments that are either executed on the intermediary nodes of the network (in the switches, routers, flexible servers) or in the users end station. The execution of programs can be requested and delivered either by the Active Network users or the network operator:

- Routers within the network act on user data flowing through them.
- Users can configure the network by supplying their own programs that perform computations.

This demands that the Active Host (e.g. a router) has sufficient processing resources. The Active Host has to be built having the capability to execute customisable code; the extra processing and security requirements increase the complexity and raise concerns that are far more difficult to be addressed compared with the common header processing and signalling requirements of a standard PC-based router. The processing of bit intensive applications, which are carried in the payload of the packets, underlines the need for using a dedicated programmable logic device.

In real world application of active networks, an important aspect of usability is the degree of flexibility offered by an active network node. The degree of flexibility on the one hand depends on the manageability of a node but on the other hands it depends on the execution environments since these provide the run-time environments where the code can be installed and run. The installation and execution of code must be achievable neither resulting in a compromised node nor in an interruption of service of an active node. The execution environment of the Active Applications can be a

combination of hardware components and software interfaces. It is a host platform for applying computational processes and implementing algorithms.

FPGAs can be the optimal candidate either for implementing efficiently the routing and forwarding of the Active Packets or for the processing of the Applications that are carried in the payloads of the Active Packets. More details for the flexibility, the performance and the suitability of the FPGAs for the purposes of this research are given in the section 4.1.

Summary

This Chapter was a detailed description of background information and definitions regarding aspects of programmable devices, real time systems, critical embedded systems and Active Networks. The analysis was extended in order to prove the connection and interdependence of this research work with the above mentioned topics.

CHAPTER 3

Background in monitoring and programmable networks

This chapter is a synopsis of the related literature background, which is necessary - together with chapter 2- for reading and understanding the discussion, analysis, system architecture and results presented in the remaining chapters. It is assumed that the reader has a general knowledge of digital systems, computer hardware and networking. Several concepts are explained (hardware monitoring, reconfigurable computing, programmable logic architectures, network reconfigurable programmable devices, programmable networks) and at the same time a summary of the related work in similar fields is provided.

3.1 Monitoring and error detection

As it has already been described in the previous chapter, there are traditionally three methods for applying monitoring to systems: non-intrusive/passive hardware, intrusive software instrumentation, and hybrid where the software instrumentation is minimized. However, this research is mainly focusing on hardware monitors for the reasons that have already been quoted in the previous chapter. A transparent non-intrusive approach towards monitoring is the application of special hardware, e.g. hardware that allows bus sniffing, or non-intrusive access to memory via dual-port memories, etc., but also through the use of hardware CPU emulators.

3.1.1 Hardware monitoring domains

Hardware monitoring has been applied for performance measurements [40], [41], [42], execution monitoring of multiprocessor systems [43], [44], and real-time systems [45], [46], [47]. Since the monitoring hardware is interfaced to the target system's

hardware via the CPU socket (emulator) or via the data and address busses, it can observe the target system without interfering with its execution, and thus not introduce any probe-effects. Information processing provided by monitoring can be categorized into three functional components according to [48]: an event trigger, an event handler and an accumulation or sampling storage facility.

Computation observability received a lot of attention as well from the major silicon vendors. Attention was given to embedded monitor cores for microprocessors. Philips' real-time observability solution, SPY [49], allows the non-intrusive output of internal signals on 12 chip pins. The internal signals are grouped in sets of 12 signals and are hierarchically multiplexed on the 12 pins. The observed signals are a design-time choice. ARM's embedded trace macrocell (ETM) [50] offers real-time information about core internal operations. It is capable of non-intrusively tracing instructions and data accesses at core speed. The trace is made available off-chip by means of a trace port of up to 16 bits. The NEXUS standard [51] proposes a debug interface for real-time observability of standard defined features. It proposes a standard port while leaving the implementation details to the users.

Discussion

The work accomplished in the above academic and commercial solutions is mainly focused on performance and execution monitoring of microprocessor systems. Their contribution concerns testability, verification and debugging issues. Although it can be claimed that the context of these monitoring mechanisms shares common grounds the research goals of this thesis, none of them covers issues like concurrent error detection, application control and stable reconfiguration.

3.1.2 Hardware monitoring using logic analyzers

Xilinx ChipScope Pro [52] inserts logic analyzer, bus analyzer, and Virtual I/O low-profile software cores into the design. These cores allow the user to view all the internal signals and nodes within the FPGA. In more detail, that after place and route the user is able to access the cores and capture signal data on-chip, in real-time. Captured data is sent off-chip for analysis, via either a high-speed parallel or USB programming cable.

The ChipScope Pro tool consists of three components: the Core Inserter, the Core Generator, and the ChipScope Pro Analyzer.

SignalTap II [53] from Altera is a system tool that captures and displays real-time signals in a system-on-a-programmable-chip (SOPC) design. By using a SignalTap II Embedded Logic Analyzer (ELA) in systems generated by SOPC Builder, designers can observe the behaviour of hardware (such as peripheral registers, memory busses, and other on-chip components) in response to software execution.

Another major player in this area is Agilent who offers a big variety of hardware and hybrid monitoring solutions. The Agilent FPGA dynamic probe [54], used in conjunction with an Agilent logic analyzer, provides access to internal signal of the FPGA. It is possible to measure up to 128 internal signals for each external pin and measure a different set of internal signals without design changes. FPGA timing stays constant when you select new sets of internal signals for probing.

The TA-700 [55] PCI bus analyzer/exerciser from Catalyst is another logic analyser that applies performance, workload and timing analysis of the PCI bus. The PCI850 System Analyser/Exerciser [56] bus-analysis diagnostic tool from Silicon Control and the PI-PCI32E PCI Bus Analyzer [57] from Corelis are similar type of logic analysers with small differentiations in their GUIs. The DiaLite Platform [58] (Temento Systems) allows the designer to embed system assertions and conditions written along with the system specifications before synthesis and check them in real time. While the system design is running, the assertion checker verifies the properties in real-time. A property failure will trigger a process and open the corresponding debug interface with all properties verification details.

MAMon [59] is a system that integrates a small hardware component into a SoC topology. It works like a probe, either by listening to logic- or system-level events in a passive manner, or by being activated by software that writes to a specific register. Detected events are time-stamped and sent via a link to a host-based tool environment where the events are stored in a database. The tool environment includes a set of facilities to view, search, and analyse the events in the database.

Discussion

All these tools are specifically designed for verifying, testing and debugging system behaviour as well as for detecting and fixing bugs throughout the design process, rather than for run-time system monitoring. The work presented in this thesis is related to the implementation of a hardware on-chip module, embedded in a PC architecture for monitoring the transactions activity and detect potential erroneous conditions. The module does not interfere with system performance and ensures timely detection of the errors without requiring manual intervention. The goal is to prevent the propagation of an error in the system and thus to prevent possible system failures, by using passive (and occasionally active) control and reconfiguration of the execution environment.

3.2 Reconfigurable computing and programmable networking

As it is obvious from the discussion which was made in the previous chapter, FPGAs can deliver various tasks as parts of embedded systems in telecommunication platforms; they are able to adapt hardware in response to changes in processing environment and to develop processing elements for network protocols that change frequently. One very wide spread idea is to use FPGAs for implementing flexible and performance competitive routers or switches for computer networks.

3.2.1 The P4 architecture

The Programmable Protocol Processing Pipeline (P4) [60], [61] exploits the dynamic re-configurability of RAM-based Field Programmable Gate Arrays (FPGAs) to provide both hardware performance and dynamic functionality to network components. Forward error correction (FEC) was used as an example of a protocol processing function. The measurements showed that the P4 improved TCP performance in a noisy environment. Another import aspect of the P4 research concerns the –first ever- definition of the possible security threats of a networked FPGA-based system [62]. The security analysis focuses on attacks at the electrical signals level and sets the foundations for the discussion about the hardware equivalents of viruses (FPGA viruses).

Discussion

This early work in the P4 project is the initial step in the “new world” of the networked reconfigurable computing. This project envisions the future potential security threats for the FPGA-based critical systems, as far as the low-level signal attacks are concerned. Hardware bugs, application-level errors and resources misuse are not included in this study. The mechanism that will counteract against the signal-level security threats includes only standard computer network security precautions. P4 project does not propose a permanent embedded monitoring solution to prevent system failures.

3.2.2 The Field-programmable Port Extender (FPX)

Significant work in the area of reconfigurable computing applied to Active Networks and other similar network topologies has been accomplished in the Applied Research Laboratory at Washington University (St. Louis). Numerous publications were produced using the FPX as the test platform. The Field-programmable Port Extender (FPX) [63], [64] is a general-purpose, reprogrammable platform that performs data processing in Field Programmable Gate Array (FPGAs) hardware. Data packets can be actively processed by user-defined, reprogrammable modules as they pass through the device [65], [66]. The hardware-based processing allows the FPX to achieve multi-Gigabit per second throughput, even when performing deep processing of the packet payload. Results have also been delivered [67], [68], [69] using the partial reconfiguration features of Xilinx [70]. The FPX project has also offered recently a number of publications concerning FPGA-based TCP-IP packet processing for security purposes (TCP flow processing, header processing, payload scanning) [71], [72]. In more detail the work involves standard network security countermeasures implemented in FPGAs including encryption, data integrity checking and sequence number verification, integrity and authentication of control messages sent over the public Internet etc [73], [74].

Discussion

The thriving Applied Research Lab has a considerable contribution in the reconfigurable computing field with extensive applications in the programmable and

active networks. Their work is built around the Field Programmable Port Extender aiming to accelerate the network functions of a switch/router with the help of FPGA devices. The results therefore concern this specific custom-built platform. The security framework is limited to standard computer network security methods and algorithms that are implemented in hardware. Thus, it enforces the intrusion prevention mechanisms but without any reference to concurrent error detection techniques.

3.2.3 ANN - Active Network Node

ANN is a hardware-based reconfigurable router designed to support high performance active applications [75]. The platform uses a general purpose microprocessor and a Field Programmable Gate Array (FPGA) on every port of a switch backplane (ATM gigabit switch). The microprocessor and FPGA architecture is called the processing engine and it can be programmed on-the-fly. The microprocessor task is to manage the packet functions, while the FPGA implements performance critical functions in hardware. Functional details concerning the execution environment (DAN) and the operating system kernel are given in [76] [77]. The system is tested with an active multicast video application.

Discussion

This project demonstrates practical evidence for the performance acceleration of an FPGA-based reconfigurable platform. The system is used as part of an Active Network and therefore it is a useful source of information that helps us understand the performance and security requirements of such a critical computer network

3.2.4 AMnet

The goal of AMnet [78], [79] is to provide customized multicast services on demand, thus to provide a framework for a flexible open communication platform. AMnet is based on active and programmable networking technologies and uses active nodes (AMnodes) within the network. These AMnodes execute on-demand loadable service modules to enhance the functionality of intermediate systems without the need of long global standardization processes. An error control policy is applied for preventing network overload conditions.

Discussion

This project is another example of how FPGAs are used in a networked reconfigurable system. The error handling features of this system provide QoS and error control in a communication link of an Active Network.

3.2.5 Further similar research

This paragraph is a synopsis of the remaining research concerning FPGA-based configurable computing targeting various network topologies. PLATO configurable architecture [80] has been designed and built in order to serve as a platform for experimentation with active networks. This architecture provides 4 physical bi-directional connections for ATM networks with large reconfigurable resources. Detection of Denial-of-Service (DoS) attacks and real-time load balancing for ecommerce servers were the most interesting applications tested in PLATO. Another interesting reconfigurable architecture is presented in [81]. Active networks are used as the host platform and performance evaluation quantify the overheads of the system throughput. The researchers in [82] introduce a fully-automated fault recovery system for networked systems which contain FPGAs. If a fault is detected that can not be addressed locally, fault information is transferred to a reconfiguration server. Following design recompilation to avoid the fault, a new FPGA configuration is returned to the remote system and computation is reinitiated. Although this research has many abstract similarities with the work undertaken in this thesis, the main target is to locate specific permanent interconnect and logic faults in an FPGA devices.

3.3 Remote FPGA (re)configuration

The ability and the implications of implementing a system for network-based FPGA reconfiguration has been attempted in a number of projects. Early work in programming reconfigurable hardware through the Internet is presented by Gómez et al [83]. In [84], an FPGA in a remote system takes the place of a standard network interface. The FPGA is capable of downloading new services and upgrades from the network. In [85], a centralized job management system for reconfigurable computing systems is described. These systems are reconfigured over a network using job

scheduling. A series of standard job management systems and monitor of the resources are analyzed. In [86], an FPGA-based system is directly connected to the Internet. Both FPGA configuration and application data are transferred to the FPGA via the network. In [87] the research effort focuses on the implementation of an Active Network with programmable logic support in order to accelerate the applications. The framework includes software modules scheduling and real-time resource monitoring. Also, the IP stack has been modified in order to comply with the Active Networks concept. This project received considerable support from the writer of this thesis and thus more information concerning its specific details are given in *Appendix I*.

3.3.1 Xilinx - Internet Reconfigurable Logic (IRL)

The major programmable logic manufacturers have also introduced system solutions for the remote reconfiguration of FPGAs. Internet Reconfigurable Logic (IRL) [88] is a system design methodology that enables modification and upgrading of hardware and software in a target system across a network without the need for a service technician or user to directly perform the change. This methodology, when applied to the design process, creates products that are IRL-enabled. IRL can enable upgrades of multiple systems simultaneously, and the ability to go back to a previous configuration if necessary. This interesting rollback¹ property of IRL provides reconfiguration to a stable state. The system includes a fully customisable API called PAVE [88] which is part of the VxWorks real-time operating system.

3.3.2 Altera - Remote system configuration

Altera has developed a similar concept for the Stratix FPGA family devices. Using remote system configuration [89], a Stratix or Stratix GX device can receive new configuration data from a remote source, update the flash memory content (through enhanced configuration devices or any other storage device), and then reconfigure itself with the new data. On power-up in remote configuration mode, the Stratix device loads

¹ Rollback is the ability to revert to a previous upgrade (possibly the Default). In a system that has space for more than two configurations, (e.g. using a commodity flash chip), it could rollback to a known good upgrade that was previously installed

the user-specified factory configuration file, located in the default page address 000 in the enhanced configuration device. After the device configures, the remote configuration control register points to the page address of the application configuration that should be loaded into the Stratix or Stratix GX device. If an error occurs during user mode of an application configuration, the device reloads the default factory configuration page.

3.3.3 Triscend E5 - Secure remote updates

Triscend (which was acquired from Xilinx) has implemented a secure remote update system using configurable System-on-Chip devices. The Triscend E5 Configurable System on Chip (CSoC) [90] device is able to host downloadable applications from the internet; it contains both programmable hardware and an industry standard microcontroller. The system is equipped with integrated cryptography techniques providing by this way tools that help to prevent accidental or malicious installation of incorrect updates via download.

3.3.4 Hardware Objects Technology Manager

The HOTMan platform [91] enhances the development, design and control of bitstreams in an embedded system. HOTMan enables bitstream management and remote hardware upgrading using C++ and JAVA programming languages on Windows/DOS or Linux/Solaris Operating Systems. HOTMan treats the programmable hardware as an object within the system, similar to software objects used in C++. As a result, applications that are written using C++ tend to be object oriented, modular, and upgradeable. The remote FPGA configurations are capable of total or partial reconfiguration of single or multiple FPGAs via SelectMAP or JTAG ports. HOTMan represents a method for flexible and reliable remote upgrades including features like, secured and compressed bitstream files.

Discussion

The remote hardware configuration platforms introduced concepts that are related to the security aspects, as described in Chapter 4. Useful specification concerning

requirements and architectural implications were also considered (i.e. the roll-back mechanism for safe reconfiguration of the FPGA device in the Xilinx IRL system). In most of these projects there is reference to the security issues that are raised (i.e. mainly reverse engineering, intrusion detection, misuse of the local and network resources etc). However, there is a limited or inefficient implemented work that provides answers to these problems (i.e. the security countermeasures are limited only to the secure downloads of a bitstream in the FPGA device). Moreover, these systems/projects touch only a relative small portion of the spectrum that concerns error detection, concurrency issues, availability, operability etc.

3.4 Relation to the work of this thesis

The different research efforts presented in this chapter, focus on different aspects of FPGA-based reconfigurable computing. Various projects use FPGAs in order to accelerate the performance of systems and some particular aspects of security and on-chip monitoring of real-time systems are covered. There is also significant research that proposes methods and tools for confronting transient errors, errors that are produced from radiation or other random factors in the low-level wire connections of the FPGA [92], [93]. However, there is no previous work on the application-layer -monitor and control- domain of critical FPGA-based systems (i.e. a robust monitoring and control mechanism for achieving the requirements of security, fault tolerance and failure prevention). Table 3.1 is a summary of the indicative research work that is related in different contexts with the research and implementation targets presented in this thesis.

The research goal in this work is different and at the same time complementary to the general context that was formed so far by the academic research and the commercial solutions/platforms. The aim is to confront the errors that are caused by the non-predicted behaviour of the configured application in FPGA devices. This can include inherent programming bugs, misuse of the available resources, security violations, illegal accesses etc. The implementation of the monitor and control component targets dependable computer-based systems (i.e. PCs). The foundations of an Active Network were used as a real-world system of this type, in order to obtain the necessary architectural design requirements and limitations.

Monitoring and error detection	
Hardware monitoring domains	
1 Performance measurements • Performance Monitoring of Embedded Hw/Swy Systems • RP3 performance monitoring hardware • A Relational Approach to Monitoring Complex Systems	<i>Performance assessment through hardware monitoring</i>
2 Execution monitoring in multi-processor systems • Hardware monitoring of a multiprocessor systems • On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems	<i>On-chip monitoring of multiprocessor systems and data analysis in an external platform.</i>
3 Real time systems • Real-time execution monitoring	<i>Hardware monitoring is interfaced via the CPU socket or via the data and address buses.</i>
Embedded monitor cores for microprocessors	
SPY - Pnhps	<i>Non-intrusive output of internal signals on 12 chip pins.</i>
ETM - ARM	<i>Non-intrusively tracing of instructions and data accesses at core speed.</i>
NEXUS	<i>Debug interface for real-time observability of pre defined features</i>
Hardware monitoring using logic analyzers	
Xilinx - ChipScope	<i>System verification - embedded VHDL cores</i>
Altera - SignalTap II	<i>System verification - embedded VHDL cores</i>
Agilent Logic Analyzers	<i>Several different types of logic analyzers</i>
Catalyst - IA700	<i>PCI bus analyzer, Testing, Data Analysis</i>
Silicon Control - PCI850	<i>PCI bus analyzer, Testing, Data Analysis</i>
Corelis - PI-PCI32E	<i>PCI bus analyzer, Testing, Data Analysis</i>
Temento Systems - Dialecte	<i>System verification and debugging</i>
MAMon (academic project)	<i>Monitoring, Testing, Debugging, Data Analysis</i>
Reconfigurable computing and programmable networking	
The P4 architecture	<i>An FPGA-based acceleration of application in a network environment. Introduction of FPGA security issues.</i>
The Field-programmable Port Extender (FPX)	<i>General-purpose, reprogrammable platform that performs data processing in FPGAs. Standard computer network security countermeasures implemented in FPGAs.</i>
ANN - Active Network Node	<i>Hardware-based reconfigurable router designed to support high performance active applications.</i>
AMnet	<i>FPGAs used in a networked reconfigurable system. The error handling features provide QoS and error control.</i>
PLATO	<i>FPGA-based platform for experimentation with active networks</i>
Active SANs	<i>Reconfigurable architecture used in Active networks. Performance evaluation quantifies the overheads of the system throughput.</i>
Adaptive Fault Recovery for Networked Reconfigurable Systems	<i>Fully-automated fault recovery system for networked systems which contain FPGAs.</i>
Remote FPGA reconfiguration	
Xilinx - Internet Reconfigurable Logic	<i>Upgrading hardware and software across a network.</i>
Altera - Remote System Configuration	<i>Upgrading hardware and software across a network.</i>
Inscend E5 - Secure Remote Updates	<i>SoC able to host downloadable applications from the internet.</i>
Hardware Objects Technology Manager	<i>Bitstream management and secure remote hardware upgrading.</i>
Programming Reconfigurable Hardware Through Internet	<i>Early work on remote FPGA configuration</i>
Effective Use of Networked Reconfigurable Resources	<i>System reconfigured over a network using job scheduling</i>
Hardware Support for Active Networking	<i>FPGA-based reconfigurable Active Router with resources monitor</i>
Internet Connected FPL	<i>Configuration and data transferred to the FPGA via the network.</i>

Table 3.1: Overview of the related research.

CHAPTER 4

Motivation and Framework

This chapter describes the motivation for carrying out research in the emerging field of concurrent on-chip monitoring and control of critical FPGA-based applications. It includes a general discussion of how to design a computer system, taking into account a variety of general security concerns. Also it provides a complete analysis of the particular components (i.e. security framework, policy framework) that have to be integrated in the system or in similar topologies, in order to provide maintainability. The main goal of this research is the timely detection of errors, in order to prevent system failures. For this reason, it was decided that the monitoring and the control of the configured applications has to be realized through an embedded module; the latter should be integrated passively in the execution environment (PCI-based FPGA board) and should also be able to apply a per-cycle, policy-based monitoring. The selection of the appropriate type of monitoring (i.e. software, hardware or hybrid), the need for concurrent detection of errors and other implications are also discussed in this Chapter.

4.1 Processing power versus flexibility

A significant introductory goal of the implementation stage was not only to propose a processing platform for delivering efficiently the research results, but to justify this selection as well. Hence, it was fundamental for the research deliverables of this work to define an architecture that will both satisfy the research goals and the need for innovation and originality. The research scenario that was adopted is not limited in the selection of a dependable computing system, but also in the use of a state-of-the-art processing platform for the applications. Both performance and flexibility requirements had to be satisfied. Therefore, many different solutions were analysed; FPGAs were from the beginning the obvious candidate. The reasons for using an FPGA-based

platform as the execution environment for the applications are stated in this chapter. It is clear that this section is based on common issues concerning application processing and computational complexity. The short review that follows underlines the significance and the potential of the FPGAs in application processing systems.

A system for processing applications has efficient computational power if it could be identified, among all typical and reasonable applications, the one that requires the most computational power. If a system can accommodate data traffic of this application at full link speed, one can safely assume that there is enough computational power to host any other reasonable application. The link speed between peripheral cards and the host microprocessor in a PC is restricted from the throughput of the PCI bus. The PCI bus can have one of the following modes of operation:

- 32 bit / 33 MHz, link speed = 132 Mbytes/sec
- 32 bit / 66 MHz, link speed = 264 Mbytes/sec
- 64 bit / 33 MHz, link speed = 264 Mbytes/sec
- 64 bit / 66 MHz, link speed = 528 Mbytes/sec
- 64 bit / 133 MHz, link speed = 1064 Mbytes/sec

Application Specific Integrated Circuits (ASICs) are specially built ICs optimised for achieving maximum performance under certain conditions and for executing certain computations, and thus they are very fast and efficient. However, once the circuit is fabricated it cannot be altered. To alter a functionality of a given ASIC, redesign and re-fabricate the chip is required. Although the unit cost of an ASIC is fairly cheap, the fabrication process is very expensive and time consuming. Another drawback is the fact that all the existing ASICs, which are to be upgraded, have to be replaced. Traditional network nodes (e.g. routers, switches) are based on ASICs.

A more flexible solution is the use of software-programmed microprocessors. Processors perform a computation when they execute a set of instructions. Changing the instructions the system is altered without changing the hardware. However the cost of the flexibility is lower performance, if not in the clock speed then in the work rate. An inflexible feature of microprocessors is that their instruction set is defined at their fabrication time.

In order to adapt to new protocols, services, standards, and network applications, many modern routers are equipped with general purpose processing capabilities to handle (e.g., route and process) data traffic in software rather than dedicated hardware. The implementation of network processors is a quite challenging area. Network processors distribute the computational processes over several processors (e.g. from two to sixteen). The advances of VLSI technology resulted in network processors, which have multiple processors, with cache and DRAM on a single chip, reducing by this way considerably the memory access latency. The network processing of applications is divided in two categories. The first include header processing applications and the second payload processing applications. Payload-processing applications access and possibly modify the contents of a packet during network node processing. Therefore they have a considerable computational complexity as is shown in figure 4.1. Computational complexity can be defined, as the number of instructions per byte required for an application operating on a given packet length.

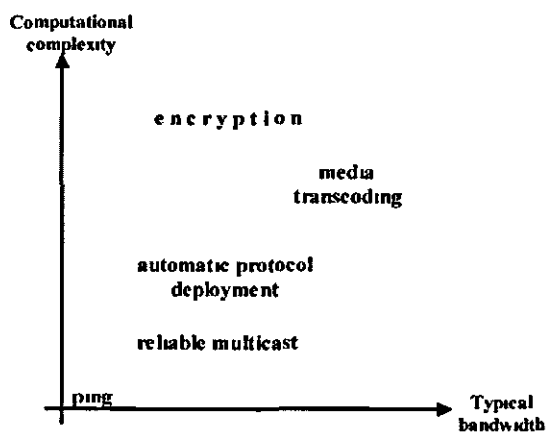


Figure 4.1: Bandwidth and computational complexity of some applications [94].

For a bit intensive application like on-the-fly encryption, with a link bit rate of $R_{link} = 132 \text{ MB/sec}$ (the slowest PCI throughput), the computational complexity N can be 204 instruction/byte according to [95]. The header-processing overhead for the data stream can be ignored since payload processing dominates the computational complexity. Hence: $M = N * R_{link} = 204 \frac{instr}{byte} * 132 * 10^6 \frac{bytes}{sec} = 26,928 \text{ MIPS}$

It is clear that a standard RISC processor by itself is sufficient for header processing at link speed, but will not provide adequate computational power for tasks that perform payload processing. The use of vector processing techniques for streaming applications (i.e., embedded vector processors) or multiple parallel superscalar or VLIW processors on a chip are promising approaches to achieve link-speed payload processing.

However, FPGAs stand in the middle of hardware and software solutions and can be considered the optimal candidate for a series of applications. While processors divide computations across time, the programmable logic devices like the FPGAs divide them across space (figure 4.2). Being far more flexible than hardware platforms like ASICs, and having considerably better performance than software, FPGAs give an answer to the technology gap.

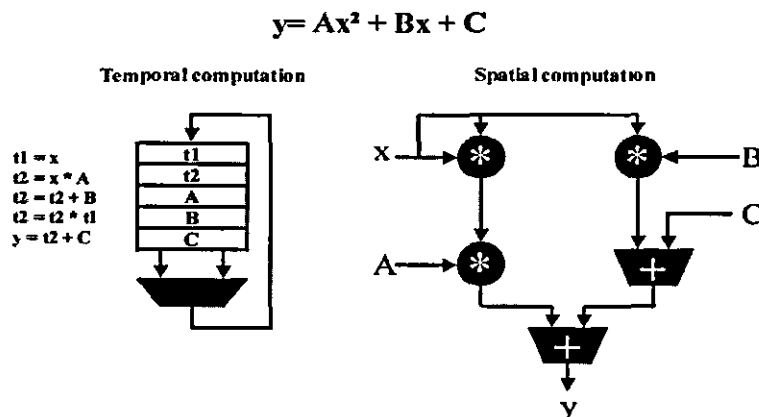


Figure 4.2: temporal versus spatial computation

“Hard-wiring” a variety of IPs that are closely coupled with a microprocessor and programmable logic as well, the advantages of hardware and software platforms are combined. The result is a System-on-Chip (SoC). Hence a technology fusion is accomplished. It is obvious from both figures that the bit-intensive applications should be implemented in a hardware-based solution.

As the VLSI technology progresses, the capacity and the performance of the FPGAs increases, making them even more attractive for a variety of applications and architectures. FPGAs performance increases in a pattern that follows Moore’s Law².

² Moore observed an exponential growth in the number of transistors per integrated circuit and predicted that this trend would continue. Moore’s Law, the doubling of transistors every couple of years, has been maintained, and still holds true today.

The FPGA constraints are related to the area utilization, the in-system programming times and the I/O requirements. The table that follows is a synopsis of area utilisation, performance, and I/O metrics, taken from several applications notes from Xilinx:

Application	Family example device	Fmax (MHz)	Slices	IOB
ADPCM, 1024 Channel Simplex [96]	Virtex-II Pro, XC2VP20-7	56	2753	69
	Virtex-II, XC2V3000-6	51	2824	69
e-Discrete Cosine Transform [97]	Virtex, XCV200-6	66	1,772	23
	Virtex-E, XCV200E-8	83	1,769	23
	Virtex-II, XC2V1000-5	83	1,802	24
	Spartan-II, XC2S200-6	76	1,772	23
	Spartan-IIe, XC2S300E-7	63	2,220	23
JPEG Encoder [98]	Spartan-3, XC3S4000-4	54	6192	151
	Virtex-II Pro, XC2VP20-6	106	6146	151
	Virtex-II, XC2V1500-6	95	6153	151
MPEG-2 HDTV I & P Encoder [99]	Virtex-II, XC2V3000-5	111	6508	113
	Virtex-II, XC2V3000-5	111	5540	58
MPEG-4 Video Compression Encoder [100]	Spartan-3, XC3S1000-4	58	2997	235
	Virtex-II Pro, XC2VP7-7	85	2991	235
	Virtex-II, XC2V1000-6	78	2979	235
MPEG-4 Video Compression Decoder [101]	Spartan-3, XC3S1000-4	62	1351	215
	Virtex-II Pro, XC2VP7-7	82	1352	215
	Virtex-II, XC2V1000-6	75	1350	215
Triple DES Encryption [102]	Spartan-3, XCV3S1000-4	101	790	301
	Spartan-IIe, XCV2S300E-7	117	790	301
		165	705	301
	Virtex-II Pro, XC2VP7-7	182	705	301
	Virtex-II, XC2V1000-6	125	790	301
	Virtex-E, XCV300E-8			
AES Fast Encryption [103]	Spartan-3, XC3S1500-5	119	447	391
	Spartan-IIe, XC2S400E-7	79	499	391
	Virtex-II Pro, XC2VP20-7	217	447	391
	Virtex-II, XC2V1000-6	170	447	391

Table 4.1: Implementation details for several FPGA-based application implemented.

Even the most bit intensive applications, such as media transcoding, can be efficiently implemented in the available FPGA devices as it is proved from the results of table 4.1. The parallel and concurrent nature of the FPGA processing ensures high performance (figures 4.2, 4.3 and table 4.1). It can be claimed that hardware platforms based on FPGAs enable packet processing at link speed. As a general conclusion (and with reference to the previous figures), it can be claimed that it is more "profitable" to implement bit intensive applications and algorithms in FPGA devices rather than in ASICs or general purpose and special purpose microprocessors.

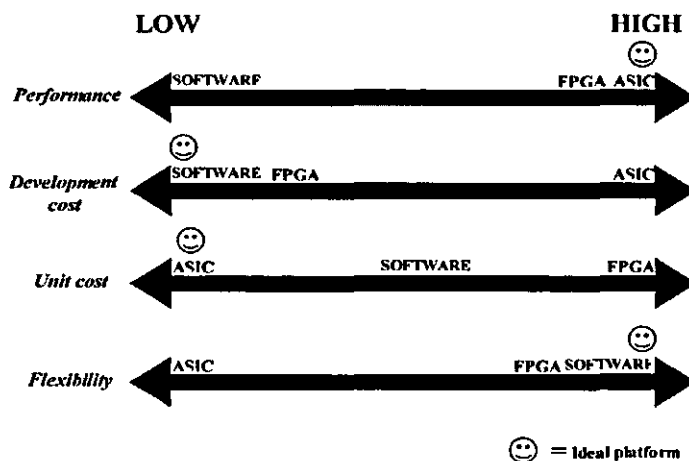


Figure 4.3: Performance and flexibility issues related to development and unit cost.

To summarise, it is important to point out that there is a constant increase in the demand of processing power in order to meet the challenges of the ever-growing electronic market. Some indicative reasons are given below:

- The growth of the internet [104].
- The increasing use of electronic equipment and especially computers, mobile phones and other electronic “gadgets”.
- The increasing number of users of high technology products.
- The continuous introduction of new complex application and technology services.
- The constant demand for applications, systems and services that deliver higher performance.

The increasing number of users, applications and services that are open in the public domain creates security risks and thus new processing-consuming security policies and protocols have to be introduced.

4.2 Active Networks – a critical computing system

The theoretical background of Active Networks was covered in Chapter two. This section is an analysis of the apparent security implications which are involved with the definition and connotation of Active Networks.

Active networking supplies the users with the ability to download and execute code within a node. That is by its nature a security critical activity. In such an infrastructure the security implications are far more complex than in current static environments. In Active Networks the author of the active code, the user who deploys it, the owner of the nodes' hardware and the owner of the execution platform can be different entities governed by different security policies. In such a heterogeneous environment security becomes an extremely sensitive issue. The possibility of loading and executing active code in Active Network Nodes imposes considerable threats on the expected operation of the nodes and/or the network due to flaws in active code, malicious attacks by unauthorized users, conflicted code execution etc. Thus, security in Active Networks deals mainly with protecting the system (infrastructure) from malicious (unauthorized) and erroneous use. The objective for the security architecture in this work is to guarantee robust/secure operation of the computing infrastructure despite the unintentional or intentional misbehaving of the applications. Fulfilling these objectives is fundamental for the usability, integrity and trustfulness of the system. Furthermore, an unintentional error in the design of a new network service, its implementation (active code), or its configuration can degrade performance of the critical system. Finally, a malicious or unintentional misbehaving of an application can severely degrade or even disable the network services perceived by other user(s) of the critical system. It is clear therefore that the Active Nodes (i.e. Active Router) are more susceptible than those in current passive networks. Several scenarios of misuses concerning applications or services could be realised in Active Networks:

- Misuse of an active network node by an active application.
- Misuse of an active application by other active applications.
- Misuse of an active application by an active network node.
- Misuse of an active application and/or execution environment by the underlying network infrastructure.
- Misuse of the Active Network as an entity

Finally a combination of the above categories is possible. These kinds of attacks (the complex and collaborative ones) are very difficult to detect not to mention prevent or be effectively tackled. Classical examples include the co-operation of various hosts

against another Active Node or Active Application. Threats can also be analysed from the perspective of a single Active Network Node, and from the network-wide perspective. Of course, threats to a single node apply also to the whole Active Network (domain). However, network-wide threats can be more subtle and harder to combat, since they are based on the global, distributed nature of network protocols, and thus, their respective active codes. The scope of the initial security framework is limited mainly to the first and last category.

The standard design flow for providing security to Active Networks (actually to Active Nodes like an Active Router) is given below.

When a packet containing executable code arrives at a node, the system must:

- Accept the authenticity of the credentials of the packet,
- Identify the sending network element,
- Identify the sending user,
- Authorize access to appropriate resources based on these identifications and credentials,
- Allow execution based on the authorizations and security policy,
- Monitor and control access to system resources throughout the execution,
- If needed, encrypt the packet to protect its code and data in transit.

A reference monitor may be used to restrict the information, system resources and services that active applications are allowed to access and use. The reference monitor consults a security policy to determine if access is to be granted. Since access-level monitoring places restrictions directly on what an application can do, it is an effective method. However, the decision of granting permission for using some resources is based upon some credentials which are not able to guarantee that a packet is harmless.

The contribution of this thesis focuses in the monitor and control domain, which so far has only been implemented in software space. For the reasons quoted in sections 2.2, 2.3, 2.4.4, and 2.6.3 it is apparent that the monitoring and control in a system like an Active Network must be delivered in a concurrent mode, especially when the execution environment of the application is a FPGA device. This is secured by using a dedicated, embedded hardware monitor.

The general research requirements were already quoted in section 1.1. One of those was to select and study the properties of a real world critical system, based on a computer architecture. Active Networks are subject to “firm” time deadlines; they also demand that the Active Applications practice a fair utilisation of the host resources, they require high levels of integrity, availability and reliability and they have also security as well as performance requirements (see section 2.7.5). Their basic experimentation platform can be based on PC hosts. Therefore an Active Network or better a PC-based Active Router satisfies the above research/design property. Considering also the current infrastructure, expertise and the know-how at Loughborough University [87] as far as the Active Networks are concerned, this particular system was an appropriate selection. The structure, properties and requirements of an Active Router were studied in order to deduce useful design demands that will enable the implementation a robust monitor and control module that it can be reused as a generic component in various critical computing systems.

4.2.1 The security context

The target topology, which will be examined comprises of an Active Router (or Active Node). This is a PC-based router that carries a PCI board with one FPGA device dedicated for applications that need a bit intensive processing environment (figure 4.4).

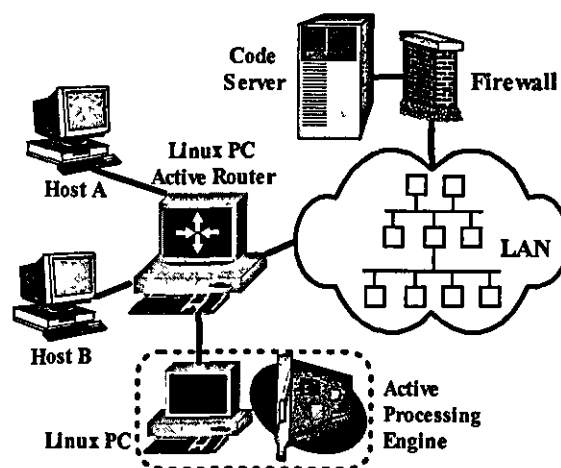


Figure 4.4: An Active Network with a reconfigurable processing engine.

Following the Active Network design approach, the Active Packets [87] may have one of the following operation modes:

1. In the first scenario the Active Packet has a pointer to a certain application (IP core). The requested application is stored either in the Active Router or in a different location (e.g. a Code Server [105]). The Active Router detects and processes the Active Packet so as to serve the request. The requested application is then configured in the hardware platform. The Active Router forwards the user's data to the PCI card, so as to be processed by the Active Application.
2. In the second approach, several Active Packets carry a configuration bitstream and data in their payloads. The Active Router detects and processes the Active Packets so as to serve the request. This processing stage includes authentication of the user, and integrity checks. The application is then configured in the target FPGA board and in the following stage the host transfers the user data.

The Active Packets are identified as active when the Active Router processes their header information (see also Appendix I). The Active Router then decides whether to download the new configuration bitstream in the FPGA device or not, depending on the current activity on the FPGA board. Apparently, as already stated in the previous sections, the new configuration can potentially extend and modify the network infrastructure. The routing and the forwarding of the packets are vital processes of an Active Router and must not be interfered with, by any means since their maintenance is important for the viability of the network. Moreover, the operability of the routers is critical to the proper and correct running of many other important systems and services. Therefore, an Active Network system designer should consider new possible threats on top of the already existent network security issues.

The security threat model has to be identified, considering all the platform-specific characteristics and requirements. The analysis of the security context that applies to the platform used for this research is based on how the basic packet functions of an IP network could affect the services, integrity and performance of the system (or simply, the dependability of the system). Therefore, a set of potential threats (and their possible impact to the host computer) was identified.

4.2.1.1 Packet replication

Replication of packets occurs in many different instances inside the context of a computer network in order to deliver specific services (i.e. multicasting). If an application that is configured in the FPGA device of the platform replicates packets in an uncontrolled manner or with malicious intentions, then as a consequence a side-effect could result in contention of the PCI bus; thus leading to degradation of the services that the Active Router offers. Although this function is not necessarily considered to be malicious, it can even result in Denial of Service (DoS)³. Hardware versions of “hoaxes” and “viruses” could deliberately replicate packets.

4.2.1.2 Packet corruption

There are various situations that can result in the production of corrupted packets (i.e. packet loss, communication channel noise, congestion, low QoS etc). The impact of packet corruption on the reconfigurable platform could be multiple PCI retries, aborts, disconnects and terminations. In a similar way to packet replication, these events are not necessarily treated as malicious. However, deliberately corrupted packets may lead the system in a computational loop. In the worst case scenario a configuration bitstream that targets the FPGA platform, could be intercepted and modified when it is sent through a public computer network. Packet corruption may result in retransmission of packets and therefore in delays for system functions and in an extreme scenario it can cause the hang-up of the system.

4.2.1.3 Packet damage

This type of failure can either be the result or the cause of a system misuse. Packet damage can be realised when an active packet destroys or changes the resources or services of a node by reconfiguring, modifying, or erasing them from the memory. Hence it can be connected with illegal accesses to restricted memory areas. A node may

³ Denial of Service: An active packet may overload a resource or service due to constantly consuming network connections or using a great portion of the CPU cycles available or by causing bus contention. The node cannot function properly under these circumstances and another active packet cannot be executed or forwarded

also erase an active packet before the completion of its job in the node. Finally, active packets that share the same computational environment may attack each other.

4.2.1.4 Packet theft

An active packet may access and steal private information from a node. On the other hand, active packets can be manipulated and violated when they arrive in an Active Node. Even if it is encrypted, it is not totally safe because it usually has to be decrypted in order to execute. Packet theft may also result in application hijacking. Hardware versions of “worms” as well as hardware version of spyware programs may attempt illegal memory reads and memory writes. Reverse engineering may also be the goal of packet theft.

4.2.1.5 Packet transformation

The packet transformation can be part of the processing sequence of many different Active Applications (encryption, compression, coding etc). The impact of packet transformation in the PCI-based reconfigurable platform is limited to PCI retries, aborts, disconnects and terminations if the core that is configured in the FPGA device fails to deliver a service successfully. All the previous PCI transactions events do not necessarily constitute errors but they should be treated with suspicion because if they exceed some limits they can degrade the performance and affect various services of the PC.

4.2.1.6 Packet fission and fusion

Packet fission can result in PCI bus contention and DoS, since a fissional node has the potential of forwarding more packets than it receives (e.g. data decompression). Packet fusion is in a way an opposite packet function. It involves the merging of different data into a union. It is possible that the application that is configured in the PCI-based FPGA board, exercises fusion functions in data; this may result in several undesirable PCI events such as persisting disconnects, retries and in the extreme it can cause performance degradation of the whole node as an entity.

4.2.2 Hardware “bugs” and malicious attacks

A system failure can also be realised by the logic failures of a digital design (or hardware bugs), which generate invalid computational cycles. A hardware bug may have analogous effects with a software bug; it may lie dormant within a rarely used block for a long time until a particular sequence of events satisfies the condition required for its activation. Hardware bugs can monopolise the PCI bus, try to read an empty memory, try to write to a full memory etc. Moreover illegal cycles can violate certain rules of a protocol specification and lead the system to “hang up”.

Malicious attacks can also take place. For example an application, which is configured in the PCI-based FPGA platform, deliberately activates a mechanism that adds noise to the output signals of the core. This could cause PCI protocol errors and generate random delays. Malicious attacks can also take place with hardware versions of viruses, hoaxes, Trojan horses and spyware programs (as already mentioned in the different Active Packet event properties). If the latter malicious attacks are implemented in the heart of the reconfigurable system (i.e. PCI FPGA board) they can even have catastrophic results for the node and possibly for other computer entities that are dependent on the critical services that the host delivers. Considering also the fact that these malicious applications are implemented in FPGAs that have inherent parallelism in their operation, it is easy to understand that is even more difficult to confront and eliminate the consequences of their activity.

4.2.2.1 A generic hardware attack scenario

An application configured in a PCI FPGA device can have full access to the system and can communicate with any part of the address space. This means, among other things, that an attacker can read or write to the BIOS memory on the motherboard or in peripheral hardware. In earlier times, BIOS memory was stored in ROM or in EPROM chips which could not be updated from software. These older systems require the chips to be replaced or manually erased and rewritten. Since this is not very cost effective, new systems employ EEPROM chips, otherwise known as Flash ROM. Flash ROM can be re-written from software. Modern embedded systems often include Flash ROM. A given computer can have several megabytes of Flash ROM on various controller cards

and the motherboard. These Flash ROM chips are almost never fully utilized and leave lots of room to store backdoor information and viruses. To an attacker, the compelling reason for using these memory spaces is that they are hard to audit and almost never visible to software running on a system. To access such hardware memory” requires driver level access. Furthermore, this memory is immune to re-booting and system re-installation. If someone suspects a viral infection, restoring the system from tape or backup will not help. EEPROM memory is fairly common on many computer systems. Ethernet cards, video cards, and multimedia peripherals may all contain EEPROM memory. The hardware memory may contain flash firmware or may just be used for data storage.

4.3 Counteract framework

The threats mentioned in section 4.2.1 do not constitute the complete and absolute security threat model for the platform. However, they can be thought as a solid introduction and a pathway that helps us to build a robust defence system in an incremental way. Table 4.2 is a summary of all the identified Active Application functions that can lead the reconfigurable PCI-based FPGA platform to a failure mode, as well as the effects and the causes of these failures. Although the causes of these failures or security “holes” in different critical computing systems may differ from each other, the effects of these failures are sharing common grounds between several similar computer architectures (i.e. the example systems quoted in section 1.2.5).

For each of the functions in column 1 of table 4.2, a corresponding effect (or effects) were identified and listed in column 2. A failure effect is what the user of the critical FPGA PC platform will experience or perceive once the failure occurs. Examples of effects include: inoperability or performance degradation of the system or process, illegal memory accesses, contention, etc. Aside from its effect(s), the potential cause(s) of every listed failure mode must also be enumerated (column 3). A potential cause should be something that can actually trigger the failure to occur. Examples of failure causes include weak or inefficient design, hardware bugs, hardware version of “viruses”, “warms”, “hoaxes” and spyware programs, incompatible software or hardware functions etc. A severity rating was assigned reflecting the importance

effect of each error. Each critical system has its own severity rating system, depending on the nature of its operation and the degree of dependability that it would like to apply. The error were grouped in a scale of 1-5, with '1' corresponding to 'no error effect' and '5' corresponding to maximum error-severity, such as the breakdown of a public or private critical computing system (i.e. cost and loss of availability impact). The last column presents the event detection effectiveness which the monitoring system intends to apply, with reference to the time that is needed in order to prevent the propagation of the error and hence a potential failure.

Application function	Failure effect	Potential failure cause	Severity rating / 1-5	Detection effectiveness
Packet replication	<ul style="list-style-type: none"> ▪ PCI bus contention ▪ Services degradation ▪ Denial of Service 	Bad design, network demand, hardware version of "hoax", "virus"	4	Event driven data monitor, gradual detection
Packet corruption	<ul style="list-style-type: none"> ▪ PCI retries, aborts, disconnects, terminations ▪ System delays ▪ System hang-up 	Packet loss, noise, congestion, low QoS, malicious interception	4	Concurrent detection, event driven timer
Packet damage	<ul style="list-style-type: none"> ▪ reconfiguring, modifying, or erasing resources from the memory 	Malicious attack, hardware "bug", hardware "virus"	5	Concurrent detection
Packet theft	<ul style="list-style-type: none"> ▪ Loss of private information ▪ Unauthorised memory accesses ▪ Application hijacking 	Hardware versions of "worms" and spyware programs, reverse engineering	5	Concurrent detection
Packet transformation	<ul style="list-style-type: none"> ▪ PCI retries, aborts, disconnects and terminations ▪ Loops and System hang-up 	Bad design, improper use of the resources, hardware "virus"	3 - 4	Concurrent detection, gradual detection
Packet fission/fusion	<ul style="list-style-type: none"> ▪ PCI bus contention ▪ Denial of Service 	Hardware "bug", hardware version of "hoax"	3 - 4	Gradual detection

Table 4.2: A synopsis of the Active Application functions and their effects.

Considering the Active Application effects as these are stated in section 4.2.1 and the sequence of events that results in a system failure (figure 2.8), all the Active Application functions quoted in table 4.2 are treated as potential errors; hence they are defined as the sources of instability, low security and low reliability. In critical computer systems (i.e. public network systems, private corporation networks etc), a part of which could be the Active Networks, dependability and security issues are raised and thus measures must be taken to overcome the effects of potential failures. The field programmable reconfigurable platform that was used applies a set of measures, which are divided into four levels of action:

- Policy-based monitoring,
- Timely detection of system threats,
- Error tolerance,
- Erroneous application removal and stable reconfiguration.

4.3.1 Policy-based monitoring

Dependable computer systems demand a policy-based (embedded) monitoring component because the standard failure avoidance techniques cannot ensure the high dependability requirements of such systems. Failure avoidance techniques aim to prevent errors from entering the system during the design stage. A way to apply this is to use applications that have already been verified and tested. Failure avoidance is also provided if the configuration bitstreams are encrypted with a standard encryption algorithm. Custom built cores (i.e. like the ones that Active Networks are supposed to host) may realize any circuit that can even physically destroy the target FPGA or other embedded components. Therefore, commercial application cores or open source cores are considered to be a much safer option compared to the applications implemented by unknown users. However, all the applications are subject to failures and hence none is considered to be bug-free. This last statement is proved to be true in our everyday experience, when verified and tested software or hardware components are malfunctioning.

For this reason, a set of policy rules has to be designed and implemented in order to deal with the effects of errors. The policy based monitoring of a critical computing

system should be implemented in hardware for the reasons quoted earlier in this chapter two. The observability of the target hardware topology is achieved either by tapping a subset of crucial signals or by tapping all the signals if this is required. These signals can be used to form macros and complementary signal groups which are necessary in order to build the individual policy blocks. Each one of these blocks is feeding the overall policy framework. Next, the signal states are checked for validity according to the implemented policies. Comparator logic has to be developed for satisfying this design requirement.

4.3.2 Timely detection of system threats

The detection of potential threats or errors in critical computing systems should be realised while a service is being delivered. In this way, the effects of the failures are minimized or eliminated within the operational system. In such systems the pattern of monitoring, matching and collecting is impractical and inefficient because these steps involve a considerable and very crucial delay for the detection of the potential errors. In most of the cases these critical systems require a more solid, efficient and concurrent approach for providing protection against application bugs, protocol errors and misuse of resources. Therefore, the collecting of data is replaced with a concurrent application control mechanism while monitoring and matching are kept intact. The time that an error is detected is a critical factor especially if the severity rating of this particular error is high. An error with severity rating 5 should not propagate to other system components because in this case the stability, integrity, security and hence the dependability of the whole system is endangered. Therefore, concurrent methods of error detection with parallel processing of signal states have to be applied. The most common techniques for error detection are:

- Replication checks: In this case, multiple replicas of a component perform the same service simultaneously. The outputs of the replicas are compared, and any discrepancy is an indication of an error in one or more components.
- Timing checks: Typically a timer is started and set to expire at a point at which a given service is expected to be complete. If the service terminates successfully

before the timer expires, the timer is cancelled. However, if the timer times out, then a timing error has occurred.

- Run-time constraints checking: This involves detecting that certain constraints, such as boundary values of variables not being exceeded, are checked at run time.
- Diagnostic checks: These are typically background checks that determine whether a component is functioning correctly. In many cases, the diagnostic consists of driving a component with a known input for which the correct output is also known.

A concurrent monitor detection component can be designed borrowing features of the above mentioned error detection techniques. Indeed, as it will be described in the next chapter, the monitoring mechanism is integrating different controls and methods for error detection (i.e. applying concurrent or timely detection of errors,) preventing the propagation of errors to other functions or services of the system. This requires a very good understanding of the flow of information in the system/component that is under monitor.

4.3.3 Error tolerance

Error tolerance is based on the fact that many digital systems exhibit acceptable behaviour even though they contain defects and occasionally output errors. A circuit is error-tolerant with respect to an application, if it contains defects that cause internal and may cause external errors, and the system that incorporates this circuit produces acceptable results [106].

In short, a class of errors may only have either local effects or minimal impact to the system operability. Thus this type of errors at the system outputs and their presence in the system can be easily tolerated. The policy exercised in this thesis follows the rule: when the severity rating of the failure is lower than five (see table 4.2), then error tolerance policy is applied in a realistic basis. Nonetheless, the presence of such errors should be detected to evaluate invisible degradation and to make reconfiguration decisions in a timely manner. Many digital systems -critical as well as low-end- carry out computations where the results do not always have to be "exactly" correct at all times. Examples of such systems include signal and image processing, voice and image

communication, and robust control systems. In such systems errors either cause no degradation or only cause acceptable degradation in the system outputs.

4.3.4 Erroneous Application removal

The monitoring mechanisms attempt to isolate a potential system error before this enters a service. Some of the errors may endanger the stability of the system. Therefore, when a configured application produces errors with high severity rating, it is considered as a potential threat for the operability of the Active Router. As a first defensive step the monitoring system should attempt to stop the illegal/unauthorised/erroneous output of the configured application locally upon the detection of a predefined error. If the erroneous condition is persisting then the monitoring and control mechanism should safely reconfigure the FPGA device with a stable application. This should take place considering a minimal risk scenario for avoiding contention conditions either in the system bus (i.e. the PCI bus) or in a local bus interface. Therefore, it is highly desirable to employ dedicated system ports or interfaces, instead of utilising system resources (i.e. host handshake, memory transfers), which could lead in a deterioration of the erroneous conditions⁴.

Summary

This chapter presented the development research framework which is the “substrate” of the work presented in this thesis. The functionality of dependable computing system, such as Active Networks and Active nodes, was studied and a forensic analysis was delivered. Thus, the forming of policies and rules was based on architectural requirements and specification of programmable networks.

⁴ The use of a wrong application-removal and reconfiguration method of the erroneous application that is configured in a PCI-based FPGA board, could accelerate the potential system failure. This could happen if for example a packet replication error generated by the configured application results in PCI bus contention conditions; using a dedicated PCI interface to remove the erroneous application and re-configure the FPGA device with a stable one could make the system to “hang up”.

CHAPTER 5

System Architecture & Design

This chapter is a detailed description of how the four-level set of countermeasures were actually implemented in hardware. An initial definition of the requirements of the hardware platform is given, justifying the decisions for the selection of the individual components. The programming and design structure of the policy based monitoring together with the predefined rules is given next. A detailed presentation of the internal components and modules is also analysed in this chapter.

5.1 Design issues

This section reports the reasons for the selection of the hardware platform, including an explanation of the requirements and the system limitations, according to the available resources. It is also a discussion on the decisions that were made, in order to deliver the research results and meet the challenges for building a high-performance platform with FPGA support that satisfies high level of dependability.

5.1.1 The hardware platform

The selection of the appropriate software and hardware for achieving the goals of this research was a complicated issue, since many different considerations and dependencies had to be taken into account. The selection of the execution environment was the first main concern; FPGA devices were selected as the processing environment of the applications in order to satisfy the performance requirements of the system. For this reason, section 2.2 was deliberately extended in order to give a sufficient answer to one of the research targets that was set in section 1.1 (i.e. define a flexible and high performance application-processing platform). The classification of FPGA-based custom computing systems includes the following two categories:

- FPGAs systems which communicate with general-purpose computers through standard serial or parallel communication interfaces
- FPGA systems which are using a high speed multiplexed data/address/control bus in order to communicate with the host.

Though the implementation of the system can work with both, the second category was the specific FPGA system targeted by this research. This type of reprogrammable FPGA systems are designed to support a variety of computing applications through providing designers FPGA resources, memories, and sometimes other hardware such as crossbars, specialized interfaces (video, audio, etc.), and even general-purpose processors. This type of FPGA-based custom computing systems are hosted by general-purpose computers which provide them with programming data and, usually, data to process through a peripheral bus such as PCI or SCSI. In addition, they frequently provide designers and users with development environments for designing applications and controlling the FPGA-based custom computing systems. Early examples of FPGA-based custom computing systems include: Splash [107], Splash 2 [108], DecPerLe-1 [109], Teramac [110], Pamette [111], the Wildforce and Wildstar family of boards [112], and the SLAAC family of boards [113]. Nowadays the market is flooded with several different types of FPGA-based custom computing systems that can satisfy almost any design need.

A PCI board was chosen in order to deliver the required field programmable features. An autonomous FPGA development board (i.e. working without the help of a host PC) [114] was rejected because of the major issues that were introduced: communication link overhead, cross-compatibility with the existent available infrastructure, as well as development problems and added developing cost. However, the implemented monitor component can be used in this particular FPGA-based custom computing system since it can be part of a PCI plug-in card (i.e. many ARM-based Excalibur boards have PCI slots). This can be achieved either when a similar development board uses an embedded Real Time Operating System (RTOS) or when additional logic is implemented in order to add operating functionality. A proof of the reusability and the portability of the platform is given by this way. It is clear, from what is mentioned so far, that the decision to use a PCI board followed the decision to use a

PC as the system host. Although the original idea for taking the above design decisions was made having in mind to implement an FPGA-based network-reconfigurable embedded system, it is clear that the hardwired monitor and control core, which was the outcome of this research, can also be used in a variety of other PC-based systems.

5.1.2 The specific properties of the PCI board

Next, the remaining components that will comprise the FPGA-based PCI board had to be defined according to additional factors and requirements that constrained the implementation options. Thus, the selection of the individual hardware and software components had to be justified with reference to the target topology. After careful consideration of the design issues, the available resources and the technology reuse considerations, a commercial PCI board was chosen. Therefore, the system was selected in such a way in order not to depend on a custom-built prototype but in free-to-access commercial boards. Moreover, as it will be explained later in this Chapter, the design of the system was kept simple. This can help the easy adaptation and integration of this work on currently available commercial products. It is also a practical way to reuse the current technologies without the need to improvise or design solutions that fit only to the exact specifications of this implementation.

A final important task was to define the specifications of the PCI board as far as the needs for programmable logic are concerned. The PCI board has to carry two FPGA devices in order to comply with the following requirements:

- The user applications must be able to use as large FPGA area as possible.
- The physical separation of the hardware modules (i.e. PCI core, monitor core, user applications) is needed in order to apply more distinctively the security and error detection policy as well as manage the system without complex overheads (i.e. scheduling mechanism for partial reconfiguration).
- The application monitor core has to be hardwired in an FPGA and should not interfere with the system processes.
- A PCI-based board with one FPGA that hosts both the PCI core and the user application will result in rebooting the host computer each time the user application needs to be reconfigured. This can be avoided either by using partial

reconfiguration features of modern FPGAs or by using a multiple FPGA board. The use of partial reconfiguration, apart from the many other overheads that introduces, it would have made the research and development work dependent on this particular technology, which is promoted only in specific Xilinx FPGA devices.

- Both the local side signals of the PCI core and the registered versions of the PCI bus signals should be accessible. The PCI core has to be a soft-core in order to enable us to integrate the monitor core, without interfering with its operation. Obviously this is not possible if the PCI core is implemented as an ASIC. The soft PCI core has also the advantage of being more flexible having a reasonable degree of customisation.

Also, the PCI board is required to have a flash type of memory in order to store locally an application that can play actively the role-back mechanism; in case of an error detection the application can be used in order to reconfigure the system and resume a stable condition for the critical computing system. Reconfiguration has also to be achieved via the JTAG port, which anyway is considered to be a standard communication interface in almost all the FPGA-based boards. The choice of the entity to be monitored was made bearing in mind the following issues:

- i) When the application that is configured in the target FPGA device produces an error, which may lead to a system failure, there must be enough time available (clock cycles) for counteracting,
- ii) The monitoring of the transactions should utilize a well-known communication standard, protocol or topology.
- iii) The security model of the Active router requires a per-cycle basis monitoring (software based solutions are insufficient in this context).
- iv) The monitoring realisation must be as simple as possible and should not interfere with the operation of other functional components (i.e. other configured IPs) or applications.

FPGA-based hardwired logic can be developed to monitor the functionality of the system components in real time.

There are many different available commercial boards [115], [116], [117] in various configuration topologies that can satisfy all the design needs described so far.

Considering all the requirements and restrictions that critical systems have, the FPGA board environment could be realized with the potential board layout shown in figure 5.1.

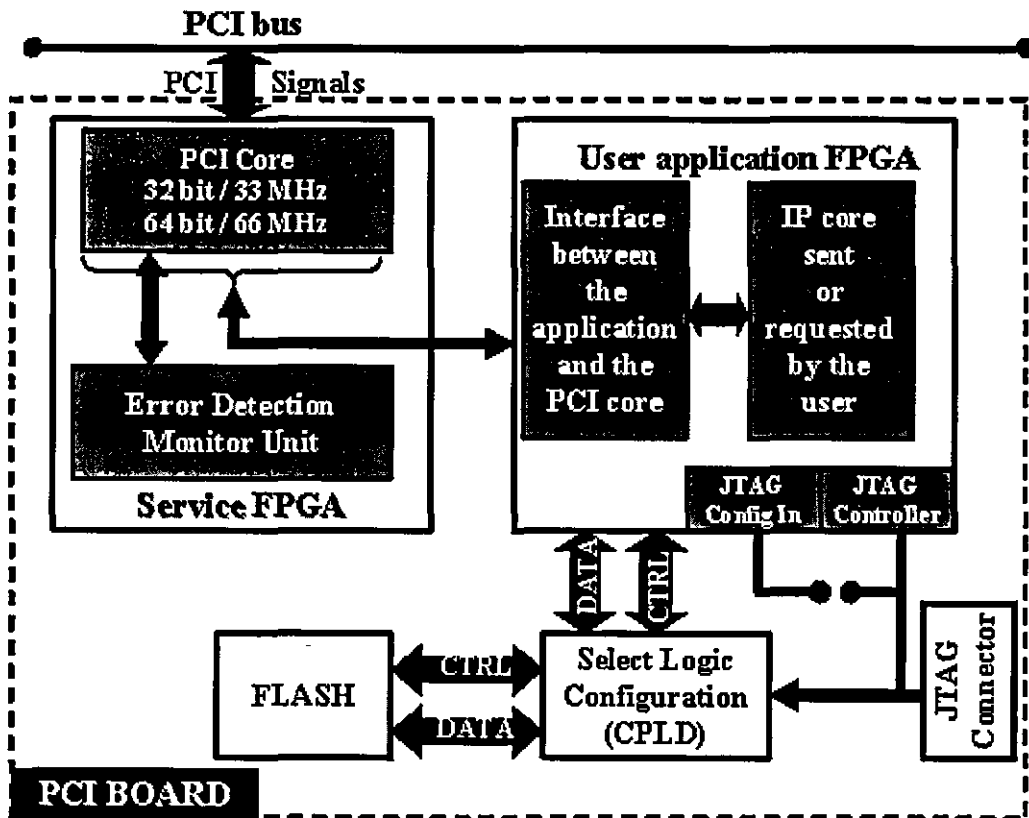


Figure 5.1: The required reconfigurable target components.

An embedded component, permanently configured in one of the two FPGAs of the PCI board (i.e. service FPGA), acts as the system Error Detection Monitor Unit (EDMU). As already mentioned, the specification of the components (i.e. number of FPGAs, communication interface, flash memory) was identified after taking into account many practical aspects concerning the effective implementation of the reconfigurable platform. The board has one FPGA that hosts the PCI core and the EDMU, and a second FPGA that hosts the user applications. The idea of implementing the error detector monitor unit in the user application FPGA is not practical, because in this case every IP core (application) must integrate the error detection monitor unit as part of its design. Furthermore, the single-FPGA implementation of such a system will

imply the hard restart⁵ of the hosting PC every time a new application configures the FPGA device; this, obviously, is the result of reconfiguring the PCI core of the board together with the new application.⁶ Therefore, this design scenario does not meet the scopes and the goals of this project. It would be a different way to apply the same scheme but it introduces an undesirable design and structure overhead.

The implementation of a novel core, embedded in a PCI-based FPGA platform is the basic outcome of this research. The Error Detection Monitor Unit (EDMU) satisfies features like policy-based monitoring, timely detection of system threats, error tolerance when a low severity error occurs and finally stable reconfiguration. The EDMU operates in a "passive" way without interfering with the host functions. The implemented module works as a "firewall" preventing the communication of the configured application with the host, only when an error occurs.

5.2 The policy-based monitoring framework

The monitoring policies were implemented in VHDL (VHSIC Hardware Description Language) as a conjunction of propositional formulas or properties, auxiliary state variable assignments, and other complementary assignments. The properties are used to specify correctness of states and not an explicit behaviour. There is no conversion of this to a state machine; this is precisely the code for the monitor. The policy formation is based on many small rules and intentionally avoids large state machines (i.e. state machine with actions specified for each state), because designing such state machines is a complex and error-prone task. The adopted design approach instead, relies on many small and generic state machines that track one thread of information; the bulk of the policies is done using compact rules.

An example is a 2-state, set-and reset machine which becomes set when a certain event happens and stays set until it is no longer needed. It is used to record certain information such as whether the transaction is a read or a write. Another example is a counter which counts the number of cycles from a certain event, or counts the number

⁵ A strict requirement of the system is to be available and operate uncorrupted, offering by this way maintainability and operability; hence it is highly undesirable to restart the PC system.

⁶ This complication could only be avoided using the partial reconfiguration feature of Xilinx FPGAs [95].

of occurrences of a special event. Only these two types of state machines (i.e. see figure 5.2) were needed in order to implement the monitor for the two identified error categories.

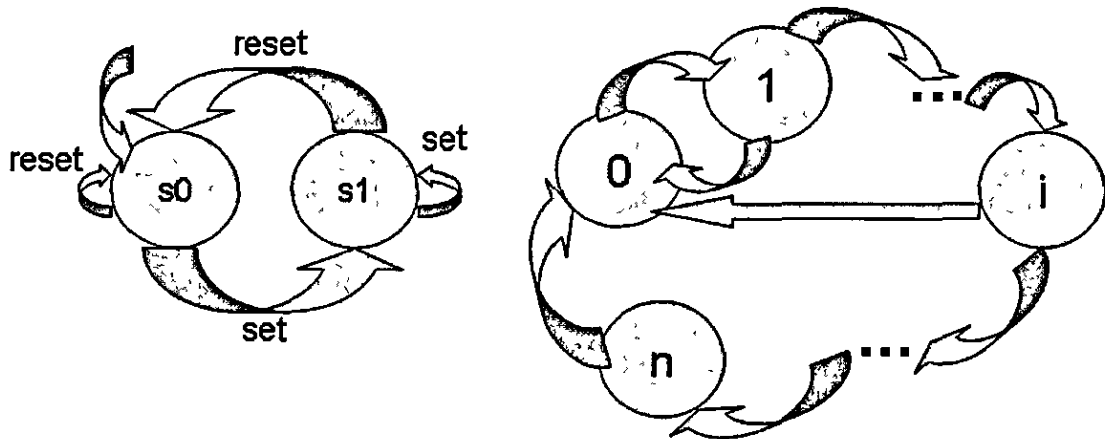


Figure 5.2: The two state machine types used for the monitor logic.

Therefore, the structure of the monitor was intentionally kept simple in order to be maintainable, low error-prone, and simpler to extend, since a basic research goal was to provide a monitoring element that can be reusable and could be extended in an incremental way. This means that the hardware monitor was built with the inherent ability to be easily ported to similar critical computing architectures, having at the same time the ability to add more policies that fit to the particular needs of the system. Moreover, the architecture is flexible enough to allow changes in counter based events/policies. This gives the advantage to apply a stricter version of the policy if the design needs of the target critical system require such a thing. For example, creating counter-based events that set shorter deadlines from the ones that a communication protocol specifies (i.e. the rule that an initiator in a PCI transaction must not keep IRDY# deasserted for more than seven PCI clocks during any data phase, can become more firm according to the specific needs of the critical computing system).

The monitor logic is supported by a number of auxiliary signals. These are formed using the basic set of the PCI core signals combined in Boolean expressions. The auxiliary signals form the backbone design of the monitor. Although several of these signals can be taken directly from the ports of the PCI core (i.e. command register), it is

preferable for the needs of the system to build them from the basic PCI core signals. Table 5.1 is an overview of these signals. A generic version of the PCI signals is given figure 5.3 [118]; the particular version of the PCI core signals that were used for the actual implementation of the monitor logic are given in the diagram in *Appendix II*. One basic functionality of the monitor logic is to hold in registers the state of the signals that occurred one clock prior of their current state⁷ (including PCI core signals, auxiliary structure signals and state machines output signals). This is crucial for the whole operation of the policy generation component. Several other states of the internal monitor logic were formed by the above mentioned signal groups using simple multiplexers and encoders. The state machines are triggered from a combination of all the types of signals mentioned so far that form Boolean expressions.

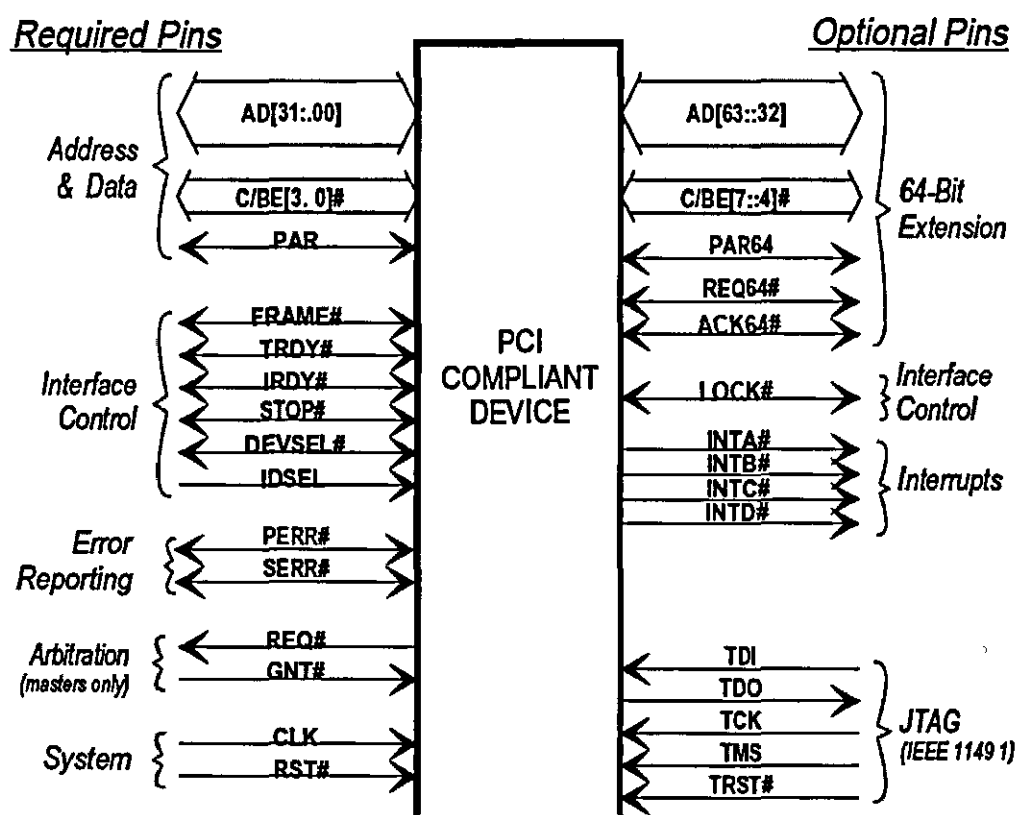


Figure 5.3: A general view of the PCI core signals [118].

⁷ The name of these signals is formed by adding a "p" in front of the original signal name.

Auxiliary signal	Boolean expression in VHDL ⁸
Burst transfer	not (not framen and not irdyn)
Address phase	not(p_framen and not framen)
Bus idle state	not(framen and irdyn)
Data phase	not(not irdyn and (not trdyn or not stopn))
Final data phase	not(((not irdyn and (not trdyn or not stopn)) and framen)
Data transfer	not(not trdyn and not irdyn)
Target abort event	not((devseln and not stopn) and trdyn)
Retry state	not((trdyn and not stopn) and initial_data_phase)
Disconnect with data	not((not trdyn and not stopn) and not devseln)
Disconnect without data	not(((trdyn and not stopn) and not devseln) and not initial_data_phase)
Back-to-back transaction	not((p_framen and not framen) and not p_final_dphase_done)
Master abort event	not(not m_abort_cond Or not m_abort)
Master abort condition ⁹	not((((not p_framen and framen) and not p_irdyn) and not (not p_stopn or not p_trdyn)) or (((not p_irdyn and irdyn) and p_framen) and not (not p_stopn or not p_trdyn)))

Table 5.1: Auxiliary signals and their respective Boolean expression.

The monitoring policies were also formed using Boolean algebra expressions combining all the signals that were described so far. The policies define the correct status of the signals in a given clock instance (examples of these expressions will be given in the next section). In other words, the monitored signals are supplying with their values the Boolean expressions of the policies and they are compared for correctness in a given time-instance as it is shown in figure 5.4 (i.e. the comparator is not included in this diagram because its operation comes after the generation of the policies). Synchronous logic is used for this purpose. To sum up, a policy is formed by using a Boolean expression containing a combination of the following input signals:

- Registered versions of PCI core signals,
- Auxiliary signals (as they are presented in table 5.1),
- Previous state signals,
- The state machines output signals.

⁸ All the signals have negative logic.

⁹ This is needed because of the delay in state machines. Namely, m_abort can only become true a cycle after the occurrence of m_abort but this variable turns true in the same cycle as when a master abort happens.

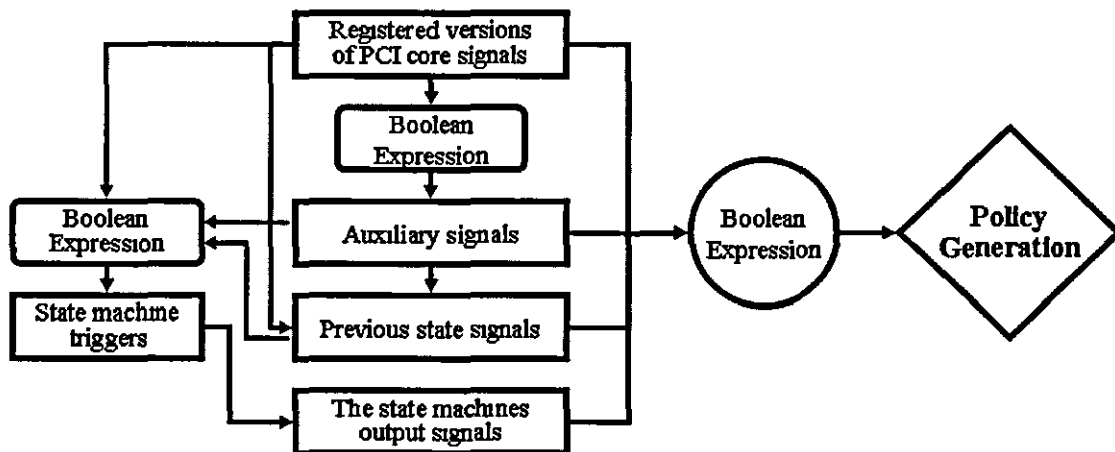


Figure 5.4: Generating policies.

A more detailed explanation of the diagram of figure 5.4 follows. The PCI core signals (e.g. registered versions of local PCI core and PCI interface signals) are the source for the creation of the auxiliary signals through Boolean expressions. Both categories of signals give input to the previous state signals module. Various PCI core signals, auxiliary signals and previous state signals are forming the state machine triggers through Boolean expressions. Finally, the outputs of the state machine together with the other signal categories (e.g. PCI core signals, auxiliary signals, previous state signals) are forming the monitoring policies through Boolean expressions. An alternative representation of figure 5.4 that depicts in a different way the same information, including the specific signals that are the inputs of the policy generation component is given in figure 5.5. This figure has details for the PCI core which were initially monitored and reused in order to form the signal categories and internal structure of monitoring core.

A monitor is an observer in a group of interacting modules, or agents which communicate via a set of protocol rules. In this context, the Error Detection Monitor Unit is a “hardwired” core that detects violations in the communication of a configured IP core with the host, over the PCI bus. The 62 identified errors were grouped in two sub-classes. This policy grouping was made according to the current number of identified errors that can potential result in a system failure, as these were stated in the previous chapter (section 4.3).

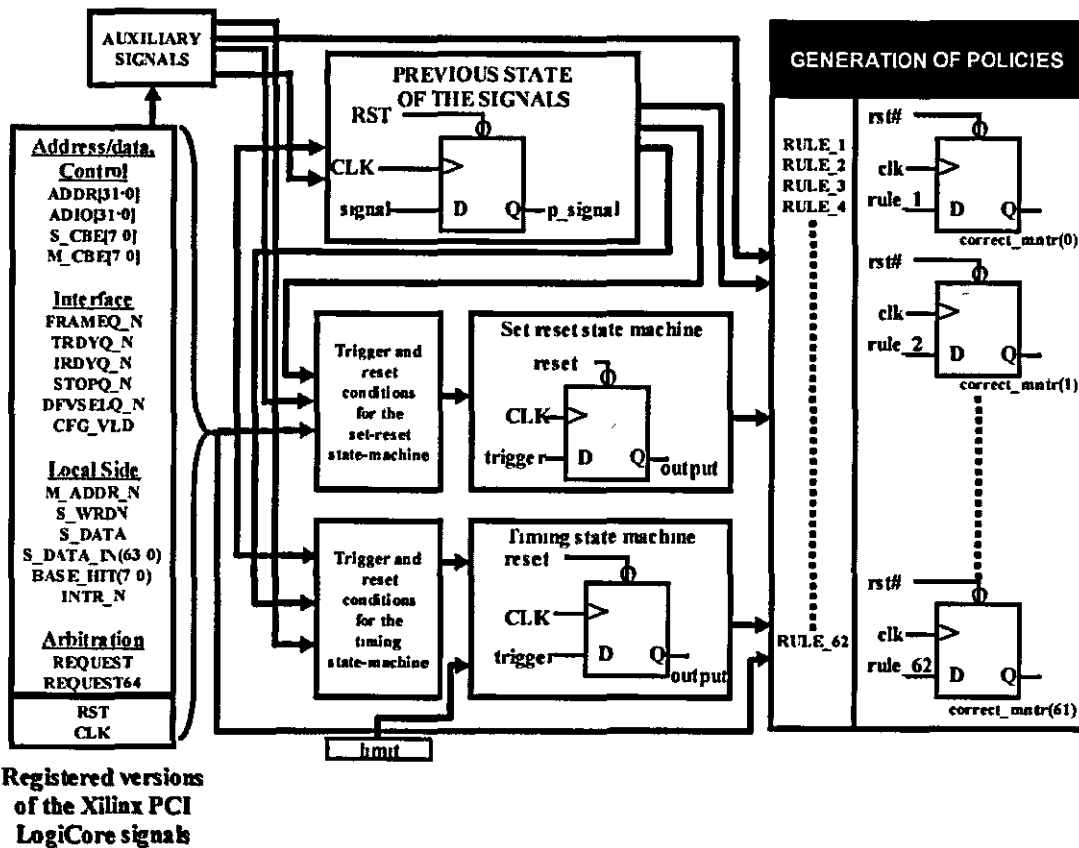


Figure 5.5: Another representation of the policy generation.

5.2.1 PCI Protocol Errors

The PCI (Peripheral Component Interconnect) bus protocol was used as the basis for forming rules written initially in behavioural VHDL. The PCI Local Bus Specification (revision 2.2) [118] includes the protocol, electrical, mechanical, and configuration specification, but the monitor covers mainly the protocol definitions. There are several versions of the PCI specifications which could be used for achieving the research goals of this work. Table 5.2 provides a comparison and justifies the selection that was not made with the present criteria (i.e. the selection of the specification to be studied was an early decision of this research). PCI Specification 2.2 was adopted based on the current market demand. The decision is made based on the majority PC motherboard and PCI device manufacturers. However, the selection of the board was made taking into consideration the research target of extensibility. Therefore, the board was selected to be PCI-X compatible.

Specification	Advantage	Disadvantage
PCI 2.1	<ul style="list-style-type: none"> • Mature technology 	<ul style="list-style-type: none"> • Outdated, bugs
PCI 2.2	<ul style="list-style-type: none"> • Widely use, • Mature technology 	<ul style="list-style-type: none"> • Will be outdated when PCI-X get accepted widely
PCI-X	<ul style="list-style-type: none"> • New technology, • Fast, • backward compatible with PCI 2.X 	<ul style="list-style-type: none"> • Cost, • Market availability and offer¹⁰

Table 5.2: PCI specification comparison.

The PCI bus protocol is being used inside millions of personal computers for communicating video, graphics, and networking data to the processor. A number of these computers are part of a critical system. The protocol is used to interconnect peripheral add-in boards such as audio cards and graphic cards, and controller components such as LAN and SCSI controllers, with the processor/memory system. There are several different configurations for the bus architecture as was first stated in Chapter 4: 32-bit or 64-bit address/data path at 33MHz, or 32-bit or 64-bit address/data path at 66Mhz.

Contradictions are a common problem with multiple-property specifications (i.e. PCI protocol specifications), as two or more properties may unexpectedly place conflicting requirements on signals under certain conditions. The EDMU applies PCI Protocol tracking and also checks for illegal signal changes in order to verify PCI specification compliance. For every clock cycle, the monitor defines protocol correctness instead of defining behaviour. The crucial observation is that a single propositional formula can describe a range of acceptable behaviour. The example property, “*an acknowledge must happen sometime between three to eight clocks*”, can easily be specified in propositional logic with a simple counter and a set-reset variable. In case of PCI protocol errors, the PCI core will respond according to its internal state-machine logic. However, the outcome of this response cannot be considered a “healthy” condition, since the side effects of the PCI protocol errors can threaten the operability, the throughput and thus the dependability of the critical computing system.

The PCI protocol error checking logic acts as a bus analyser that is used to verify compliance of user-designed PCI interfaces. The PCI bus protocol properties were

¹⁰ The majority of the PCI designs are built for 33 MHz/32 bit targets, thus additional development is needed in order to be adapt them in the PCI-X specification.

transformed into monitoring rules, according to the PCI bus specification. The representative PCI protocol policies are followed by the respective description in VHDL.

- Master must raise `irdy` within 8 cycles of the assertion of frame.

```
process ( clk, rst)
begin
  if rstn = '1' then
    correct_mntr(6) <='1';
  elsif clk'event and clk = '1' then
    If (((correct_mntr(6) and ((not masteris) and
    p_irdyn)))='1') and (p_m_initial = "00111")
  then
    correct_mntr(6) <= '0';
  else
    correct_mntr(6) <= '1';
  end if;
end if;
end process;
```

- Master must raise `irdy` within 8 cycles of the last data completion.

```
process (clk, rst)
begin
  if rstn = '1' then
    correct_mntr(7) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(7) and ((not masteris ) and
    p_irdyn)))='1')and (p_m_subseq= "00111")
  then
    correct_mntr(7) <= '0';
  else
    correct_mntr(7) <= '1';
  end if;
end if;
end process;
```

- Once `irdy#` is asserted, `frame#` can never change its state from de-asserted to asserted until the data phase completes. One data phase transaction is also considered.

```
process ( clk, rst)
begin
  if rstn = '1' then
    correct_mntr(5) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(5) and (((not masteris and not p_irdyn) and
    not (not p_stopn or not p_trdyn)) and (((not p_devseln or not
    p_devseln_history) or p_initial_data_phase)))) and not ((not
    irdyn_e and not irdyn_o) or not m_abort))='1') or (p_framen_7 >=
    "00100")
  then
    correct_mntr(5) <= '0';
  else
    correct_mntr(5) <= '1';
  end if;end if;
end process;
```

- Once `irdy#` is asserted, `frame#` can never change from deasserted to asserted until the data phase completes.

```

process ( clk, rstn)
begin
  if rstn = '1' then
    correct_mntr(9) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(9) and ((not masteris and not p_irdyn)
    and p_framen) and p_dphase_done))) = '1')
  then
    correct_mntr(9) <= '0';
  else
    correct_mntr(9) <= '1';
  end if;
end if;
end process;

```

- `irdy` must be deasserted the clock following the completion of the last data phase.

```

process ( clk, rstn)
begin
  if rstn = '1' then
    correct_mntr(15) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(15) and (not masteris and not
    p_final_dphase_done)))= '1')
  then
    correct_mntr(15) <= '0';
  else
    correct_mntr(15) <= '1';
  end if;
end if;
end process;

```

- There must be an idle after a transaction terminated by a retry or disconnect

```

process ( clk, rstn)
begin
  if rstn = '1' then
    correct_mntr(17) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(17) and (((not masteris and not
    p_devseln) and not pp_stopn) and not
    p_final_dphase_done)))) = '1')
  then
    correct_mntr(17) <= '0';
  else
    correct_mntr(17) <= '1';
  end if;
end if;
end process;

```

5.2.2 Common application errors

The EDMU goes beyond a typical validation of the PCI bus protocol at run time. Issues like bus ownership, latency, amounts of data that are transferred from and to the host are also considered in the policy framework. The goal is to control the above transaction events as well to prevent DoS, intended/unintended misuse of the resources and other undesirable events.

The monitor logic allows the analysis of general events like the count of actual data transfers, the matching of specified addresses, measuring and reporting bus utilization, latencies and retries, on-the-fly. In this category of errors the focus is not on PCI protocol-related errors but on PCI transactions and events that are considered either undesired or premature signs of anomalies that affect the dependability of the system. The auxiliary signals (i.e. table 5.1), the PCI core signals and the previous state signals are used in conjunction with the simple state machines in order to acquire useful information or metrics concerning the basic behavioural functions of the reconfigurable platform. An overview of these is given in table 5.3.

Monitoring events	Description
Target latency	Number of waits due to TRDY before first data phase
Master latency	Number of waits due to IRDY not asserted
Target Efficiency	Number of TRDY# asserted over DEVSEL# asserted
Master Efficiency	Number of IRDY# asserted over DEVSEL# asserted
Bus Utilization	Number of cycles DEVSEL# asserted over time
Idle	Number of IDLE cycles
Wait states	Number waits
Reads and Writes	<ul style="list-style-type: none"> ▪ I/O Read/Write Commands ▪ Memory Read/Writes ▪ Configuration Read/Write ▪ Multiple Memory Read
Retry	Number of Target Retry
Abort	<ul style="list-style-type: none"> ▪ Target Aborts ▪ Master aborts
Disconnects	<ul style="list-style-type: none"> ▪ Number of target disconnect with data ▪ Number of target disconnect without data
Latency	The time from REQ# asserted until data transferred
DEVSEL Speed	Reports Target Decode speed

Table 5.3: Monitoring events.

The monitoring events quoted in this table are the inputs in most of the policies that try to address potentially erroneous conditions of the configured FPGA applications of the system. Several of them are dependent exclusively to the auxiliary signals and simple timing state machines.

A general grouping of the conditions that were defined as erroneous or error-prone is given next. It is clear that the following application events are caused from transactions that cannot be considered "officially" or formally as errors. However, the firm requirements of a dependable computing system (i.e. as were described in Chapter 4), create the need to monitor, record and control such events because their impact affects the stability of the reconfigurable system. For example the PCI interface can signal an abort (i.e. target abort); this event often occurs due to incorrect programming or serious system errors. It can also signify that the initiator has incorrectly attempted to burst data beyond the address space of the target. The user application (or in this case EDMU) must respond appropriately. A list of potential error-events initiated by configured applications is quoted next (with reference to the hardware topology shown in figure 5.1).

- Continuous request for bus ownership. A back-up latency timer (apart from the one that is implemented as a PCI configuration register) was implemented in order to deal with the above-mentioned issue. The value of this timer defines in a more custom way, the minimum amount of time (in PCI clock periods) that the bus master is permitted to retain ownership of the bus each time that it acquires bus ownership and initiates a transaction (see also the third block of VHDL code that is given next).
- Access of host's memory areas, which are out of the predefined range, is not allowed. This violation could either cause an error to other computational processes that are running in the host (e.g. memory writes) or can be related to packet theft (e.g. memory reads) and application hijacking (see also the second block of VHDL code that is given next in this section).
- Data corruption resulting in delays (figure 5.6), multiple requests for retry, disconnect or abort. These events are treated as potential errors. The activity is traced because in the extreme it could result in loops and system "hang-up".

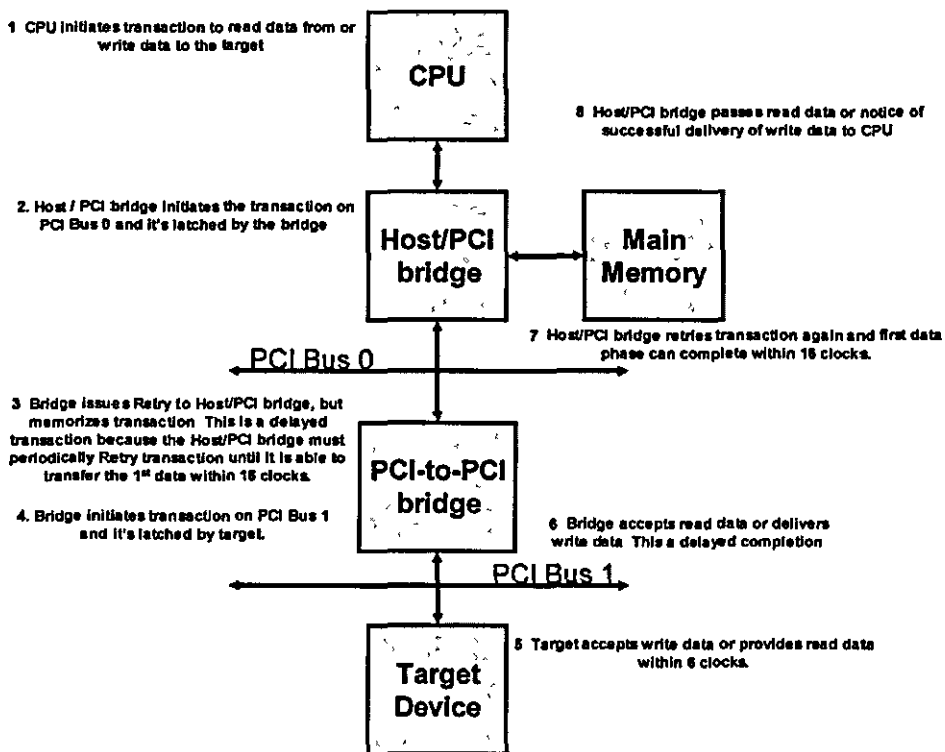


Figure 5.6: A delayed transaction example¹¹ [119].

- System must inevitably return to the idle state; if there are any deadlock states which forbid this from happening, checking for this characteristic should find such a problem. In other words, the system must always force the interface to reset to idle eventually. This characteristic has to hold because only when the bus is idle, can a new agent start a transaction. If the bus is never idle because one agent is constantly driving it, this agent has effectively taken over the bus never allowing other agents to use it. To avoid such a situation, the system must force an agent to eventually relinquish the use of the bus as a master and let the bus state be idle. The FPGA application can remain in a non-idle state and not relinquish the bus in various cases. Essentially such cases occur when a frame is de-asserted while `irdy#` is asserted by the configured application (i.e. the FPGA application while being in idle state, it can assert `irdy#` and remain in this non-idle-bus state forever¹²).

¹¹ Theoretically, if a target is always very slow, a delayed transaction would never take place.

¹² Also, during the data phase of a single data phase transaction, frame is deasserted and `irdy#` is asserted. If a target doesn't respond with a `trdy#` or a stop, the master can remain in this non-idle-bus state forever. Another case could be during the last data phase of a transaction; frame is deasserted and `irdy#` is asserted. If a target does not respond with a final `trdy#` or a stop, the agent can remain in this non-idle-bus state forever.

- Generating illegal read or write cycles. One example of this application error occurs when multiple writes to the same location are performed as a single write on the other side of the bridge. Collapsing is defined as reducing multiple writes posted to the same location to only one write that delivers the final data to the location. Although collapsing is not allowed for any type of write transactions, a bridge may allow collapsing within a specific range when a device driver indicates that this will not cause operational problems. This event is monitored and controlled within the system.
- When the user application requests the bus for an initiator operation, it may not be granted the bus for quite some time. In the meantime, another agent may initiate a target access to the user application. Denying the other agent access by forcing a target retry will be disastrous in a system with priority-based arbitration. The initiating agent may keep retrying the transaction because it has higher priority, and the user application will never access the bus. This results in deadlock. (However, in a round-robin system, forcing a target retry is a good way for the user application to perform its pending transaction first. It can then respond to the target access when it is later retried by the other agent)
- Bus contention is realised when the PCI bus reaches its throughput limit. This problem is not considered to be exactly an error. However, it can result in Denial of Service, or in crashing the operating environment of the PC. In such cases hard re-booting of the PC is needed. This state is highly undesirable for the system. Thus, a mechanism that monitors the transactions was implemented to prevent such incidents. This comprises monitoring events (i.e. table 5.3) combined with dedicated timing state machines.
- Slow targets result in delayed transactions. A target that cannot transfer the first data item within 16 clocks from the assertion of frame#, must issue a retry. Thus, the transaction is rejected and the master must retry the transaction on a periodic basis until the target is able to transfer the first data item within 16 clocks. Theoretically, if the target is always slow, the transaction will never take place. Obviously, this is not allowed because it can end up to a loop.

An indicative number of policies and concerning erroneous events is given next. The policies were written in synthesizable VHDL.

- If frame# does de-assert and irdy was not asserted on the same clock and a data phase has not just completed, a master abort condition must hold.

```
process ( clk, rst)
begin
  if rst = '1' then
    correct_mntr(4) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(4) and (((not masteris and not (not
      p_stopn or not p_trdyn)) and not p_framen) and not
      p_irdyn) and (((not p_devseln or not p_devseln_history) or
      p_initial_data_phase))))='1') or (p_framen_7 >= "00100")
    then
      correct_mntr(4) <= '0';
    else
      correct_mntr(4) <= '1';
    end if;
  end if;
end process;
```

- The user application is not allowed to access host memory areas that are out of the predefined memory range.

```
process ( clk, rst)
begin
  if rst = '1' then
    correct_mntr(1) <='1';
  elsif clk'event and clk = '1' then
    if (correct_mntr(1) and (masteris and in_addr_phase))= '1'
      and (ad >= "XXXXXXXX" or "YYYYYYYY" <= ad)
    then
      correct_mntr(1) <= '0';
    else
      correct_mntr(1) <= '1';
    end if;
  end if;
end process;
```

- It has to be either a read or write command.

```
process ( clk, rstn)
begin
  if rstn = '1' then
    correct_mntr(24) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(24) and (p_framen)) or (not
      read_command or not write_command))='1')
    then
      correct_mntr(24) <= '0';
    else
      correct_mntr(24) <= '1';
    end if;end if;
end process;
```

- If gnt# is deasserted before the natural completion of the transaction and there is a timeout, continued use of the bus is not allowed.

```
process ( clk, rstn)
begin
  if rstn = '1' then
    correct_mntr(10) <='1'; correct_mntr(11) <='1';
    correct_mntr(12) <='1';
  elsif clk'event and clk = '1' then
    if (((correct_mntr(10) and (((not masteris and not
      p_timeout) and p_gntn) and not p_dphase_done) and not
      p_framen) and not p_irdyn))) = '1')
    then
      correct_mntr(10) <= '0';
    else
      correct_mntr(10) <= '1';
    end if;
    if (((correct_mntr(11) and (((not masteris and not
      p_timeout) and p_gntn) and not p_framen) and p_irdyn))) =
      '1')
    then
      correct_mntr(11) <= '0';
    else
      correct_mntr(11) <= '1';
    end if;
    if (((correct_mntr(12) and (((not masteris and not
      p_timeout) and p_gntn) and not p_framen) and not p_irdyn)
      and p_devseln_history))) = '1'))
    then
      correct_mntr(12) <= '0';
    else
      correct_mntr(12) <= '1';
    end if;
  end if;
end process;
```

5.3 The Error Detection Monitor Unit

The policy generation was a significant part of the research and development time spent in this work. The reconfigurable platform was enforced with a policy framework that defines erroneous conditions, which were written in synthesizable VHDL code. However, another fundamental goal was to detect the errors in a timely (if not concurrent) manner, as well as to provide error tolerance and prevent the propagation of high-severity rated errors in the system. This additional functionality is essential for the reconfigurable platform, if it is going to be used as part of a critical computing system, as the one described in Chapter 4. This section includes information for the remaining important components that form the Error Detection Monitor Unit (EDMU).

The actual implementation of the policy generation, (as this was presented earlier in section 5.2) uses a particular version of a Xilinx PCI core [120]. However, the design of EDMU can be easily extended in order to include the PCI-X core features, by applying minor modifications and additions in its structure. The number of the formed policies is just a representative sample; the design structure of the EDMU was kept simple in order to be able to add more policies in an incremental way.

A basic finding during the implementation and testing of EDMU is that several signals of the PCI core are not accessible (i.e. any attempt to monitor them obviously causes contention). However, it is still possible to access the functionality of the PCI core signals by using their registered versions, while many signals were still not accessible. The solution was to indirectly produce these signals after studying the internal behaviour of the PCI core. A representative block with VHDL code for this type of signals is given next:

```
--reqn is defined
process(rst, request, request64)
begin
    if rst = '1' then
        reqn_ds <= '1';
    elsif (request or request64) = '1' then
        reqn_ds <= '0';
    elsif (request= '0' and request64= '0') then
        reqn_ds <= '1';
    end if;
end process;
reqn <= reqn_ds;
--gntn is defined
process (clk, rst, M_ADDR_N)
begin
    if rst = '1' then
        gntn_ds <= '1';
    elsif M_ADDR_N = '0' then
        gntn_ds <='0';
    elsif clk'event and clk = '1' then
        if irdyn = '0' then
            gntn_ds <= '1';
        end if;
    end if;
end process;
gntn <= gntn_ds;
```

The architecture of the monitor component does not depend on the way that the two (or more) FPGAs of the board are interfaced. This means that the EDMU is independent of the type of local bus that is used to interconnect the two (or more)

FPGAs. That is because every PCI board manufacturer that uses “soft” PCI cores, provides a wrapper that maps the signals of the local bus with the signals of the PCI core local side. Therefore the monitoring activity is triggered first by the local side signals and next by the registered version of the PCI interface signals.

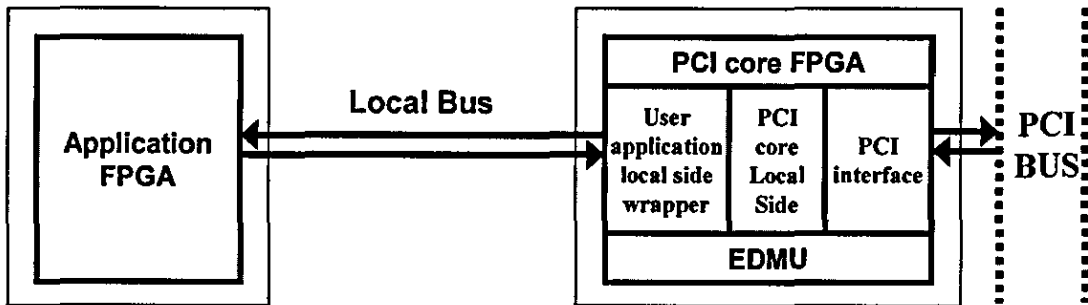


Figure 5.7: The EDMU is passively integrated with the PCI core.

The design of the EDMU as a plug-in component of soft PCI cores was implemented considering all the sensible requirements and constraints of the PCI bus functionality. The hardware monitor can be thought as a hardware firewall that operates passively and does not interfere with the vital services that a PCI core delivers (figure 5.7). The selection of the PCI board components was decided according to this property. The applications are running in a dedicated FPGA device and thus the monitor module does not introduce any realistic intrusiveness to the resources that are allocated to the application FPGA. The EDMU is integrated with the PCI core in a second FPGA of the board, occupying a particular area of it, and therefore it does not interfere with the area of the application FPGA.

Figure 5.8 represents a more detailed view of the EDMU with all its sub-components and modules. As it is seen, both the PCI interface signals (i.e. registered version of the signals) and various local side PCI signals are monitored. Next, the monitored signals are supplying the inputs of the state machines (i.e. timing and set-reset state machines) as well as the inputs of the auxiliary signals block. The outputs of these three discreet blocks are activating the (static, predefined) policies as they were described in the two previous sections. The following stage includes the data values collection and the consequent policy checker. The output of this block supplies the

input to a component that is responsible to trigger an alert whenever an error is detected. The classification of the error importance takes place on the following block based on predetermined assumptions (see also section 4.3). The comparator and error classifier blocks are the inputs of an encoder that forms a vector which in the next block is memory-mapped with an area of a PCI register (i.e. BAR0). This register together with the control component (interrupt generation and application abort) are the outputs of the EDMU that can be sampled by the host (i.e. PCItree [121] or any other dedicated API). The control component takes immediate action, issuing an interrupt as well as initiating an abort sequence of the error (with high importance rating).

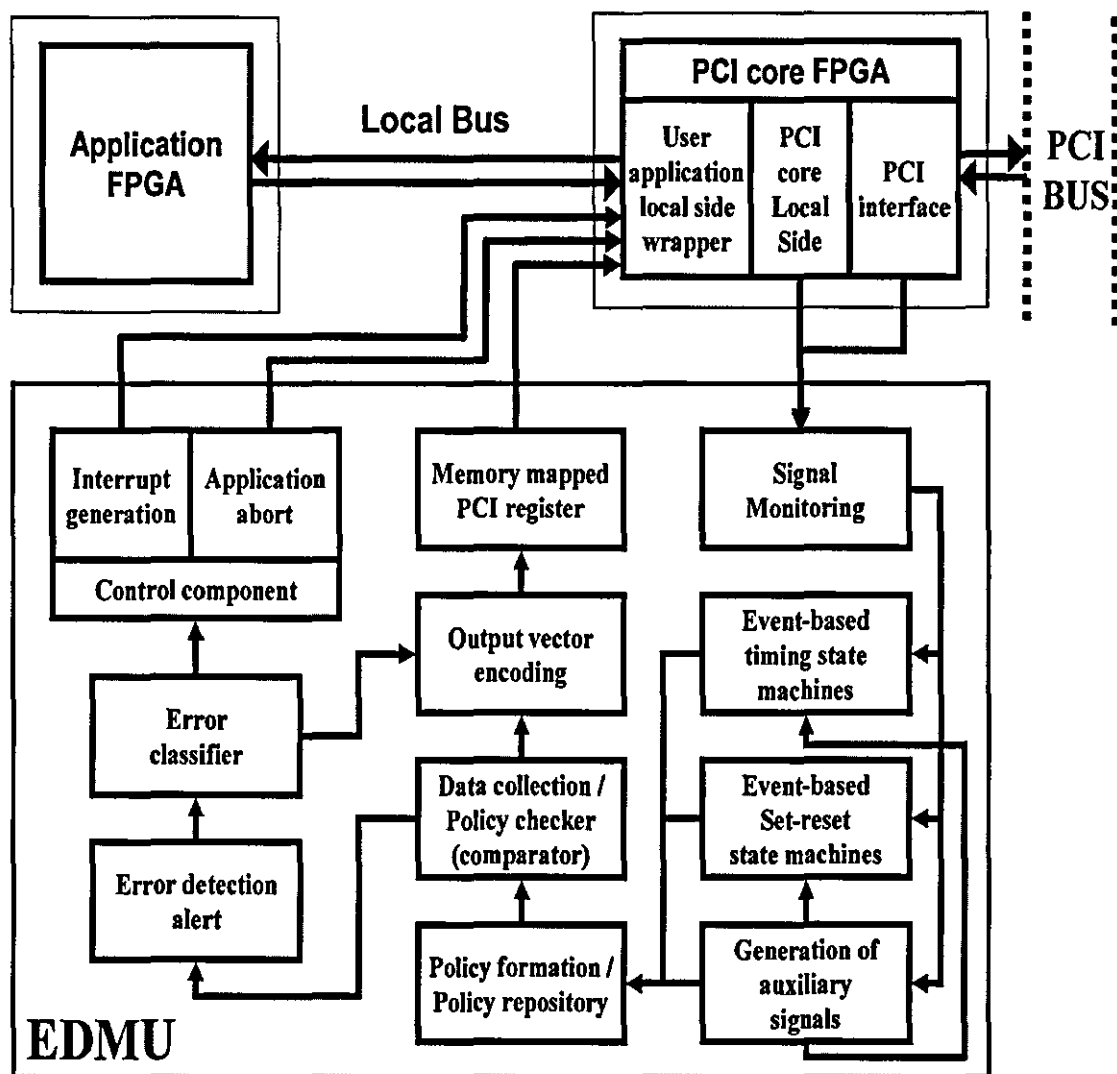


Figure 5.8: An "X-ray" of the EDMU.

An alternative way to describe the main functionality of EDMU is shown in the simple ASM chart of figure 5.9. The first three boxes are also the basis for the formation of each monitoring policy. The first state box is a Boolean expression of the monitored signals (i.e. PCI core signals); the decision box that follows (i.e. the policy checking) compares the output of the first state box with the predefined policy rules. Finally, the conditional output box (i.e. error detection alert) flags the potential system error. The values of each monitored signal are checked for state correctness every clock cycle by using this design approach.

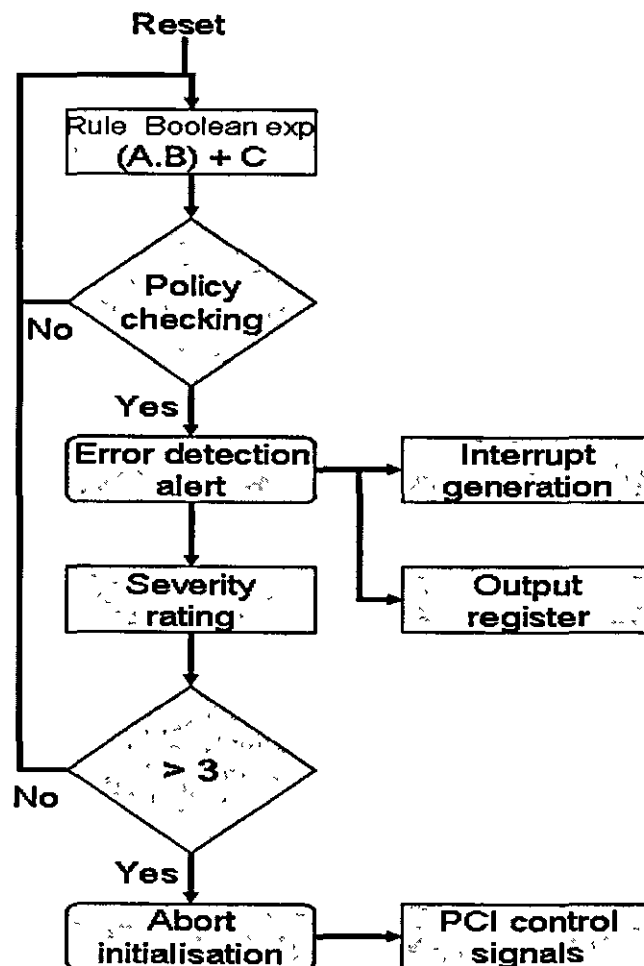


Figure 5.9: An indicative ASM chart of the EDMU functionality.

As it is already mentioned, the monitored signals (together with all the auxiliary signals) are compared with the predefined policies in almost a concurrent mode of

operation; the inherent parallelism of the FPGA devices allows us to check for state correctness in every PCI clock. The outputs of the policy rules (i.e. `correct_mntr(x)`) are grouped in a vector. This vector is encoded into a 32 bit word and stored in an internal register (i.e. `out_reg`) providing by this way the "health" status of the critical reconfigurable system for each clock cycle (see the block with the VHDL code that follows).

```
process (clk, rst)
begin
  if rst = '1' then out_reg <= "00000000000000000000000000000000";
  elsif clk'event and clk = '1' then
    if correct_mntr(0) = '0' then out_reg <=
      "00000000000000000000000000000001";
    elsif correct_mntr(1) = '0' then out_reg <=
      "00000000000000000000000000000010";
      .
      .
      .
    elsif correct_mntr(61) = '0' then out_reg <=
      "0000000000000000000000000000111110";
    else
      out_reg <= "11011110101011011011111011101111";
    end if;
  end if;
end process;
```

As it will be shown in the discussion of the results in the next chapter, when a PCI transaction breaks a predefined rule, the value of the "out_reg" register is updated one clock after the actual policy violation takes place. This register is memory-mapped to a Base Address Register (i.e. BAR0) of the PCI core. The host is able to read the value of the output register by sampling BAR0 (i.e. reading the BAR0 memory space). An offset value was chosen (i.e. 0x7C or "1111100" in binary form), according to the specification of the PCI board that was used for implementing the EDMU in hardware¹³. The block of VHDL code that follows describes the above mentioned processes. A multiplexer is preventing contention when the "S_DATA_IN" signal is used.

¹³ More details for the board are given in Chapter six, while several board specifications are quoted in *Appendix III*

It is clear that each error has a different impact in the operability, availability and maintainability of the system. Therefore, the system errors were grouped into different categories according to the severity of the threat; a scale from one to five indicates the importance of the error. This means that the system can tolerate an error when it poses a low level threat but will not tolerate an important one. Certain errors are identified at a very early stage before their activity becomes threatening to the system. This is achieved by monitoring some signals of the local side of the PCI core (i.e. the address/data bus of the local side). For example as already described before in this chapter, the error detection core prevents an erroneous application from making an illegal memory transfer (i.e. writing to a memory space of the host that is allocated to a different application). In this case the illegal memory access is detected, exactly one clock cycle afterwards. The categorisation and classification of the errors according to their severity level (i.e. represented with an array) and the possible impact to the stability of the system gives this qualitative measure in the host. This sub-component of the EDMU was implemented following the theoretical approach of the error classification that was described in Chapter four (table 4.2).

```
process (clk, rst)
begin
    if rstn = '1' then
        correct_mntr(x) <= '1';
        severity <= "111";
    elsif clk'event and clk = '1' then
        if correct_mntr(x) = '0' then
            severity <= "101";
        else
            severity <= "111";
        end if;
    end if;
end process;
```

Another critical task that the EDMU component delivers is to isolate the PCI-based FPGA board upon the detection of an error. EDMU acts in similar way that firewalls do, “locks” the reconfigurable PCI platform and forces the configured application to an abort. At this stage the host decides if the erroneous application is going to be removed from the FPGA device. The decision is taken according to the importance of the error. The recovery of the system is achieved when the FPGA device is configured with a

stable application, which has previously been exhaustively tested and verified. The abort sequence is implemented using registered versions of the local side signals of the PCI core.

Summary

This chapter is a description of all the implementation and development issues related to the design of Error Detection Monitor Unit. The functionality and structure of this core is analysed and VHDL code examples were given to illustrate the development stages and system components.

CHAPTER 6

RESULTS

This Chapter presents the behavioural simulation and the real time debugging results. Several details concerning the simulation and real time test-benches are given together with detailed description of the applications and the software that was used.

6.1 Design validation

Verifying the functionality of the EDMU was the capstone of the overall testing, debugging, and validation effort. It was important to ensure that the monitor policies of the design (especially the ones that check PCI protocol validity) are not error-prone themselves and that they are correctly implemented in accordance with the PCI bus specification. A thorough study of the PCI protocol was needed for this reason, as already stated before in chapter five. The validation of EDMU was accomplished using very simple and exhaustively tested applications as well as dedicated hardware-debugging tools (i.e. ChipScope Pro, freeware PCI core debugging tools). In more detail, during this stage, it was verified that within the correctly reachable state space, the antecedents of all the properties will each become true at some point during execution. If there is a property where the antecedent is always false, the property is never “fired” and is therefore vacuous. Since it is meaningless to have a property that is never in force, it can be safely assumed that the property is not stated correctly and requires modification.

The open source applications that were used for simulation and real-time testing of the system provided a direct access to the source code as well as reliable simulation test-benches. The source-code level modifications of these applications had as a goal to produce erroneous conditions that will violate specific rules of the policy framework in order to validate through this way the response and efficiency of EDMU. For this

reason the application errors with high severity rating were exercised and applied to the system for verifying its functionality, while this was operating in real-time conditions. Proof of concept was provided in this way; nevertheless, there is undeniably plenty of (design) space for extensions and improvements of this hardware monitor and control core. However, the initial motivation and conception of the need for research has to be followed by realistic design and implementation targets; research without realistic development targets is not acceptable in today's engineering community. This is particularly important in the hardware design cycle where the implementation, simulation and verification of a system or application is significantly more demanding and time-consuming compared to the software design cycle. The development and validation stage is equally important and demanding with the research stage.

6.2 Simulation results

A PCI compliant simulation testbench was used in order to test the behaviour of EDMU and verify its functionality. Testing scenarios test the basic transactions between two agents on a PCI bus; one agent being the PCI device under test (DUT) and other being the behavioural model of a PCI Initiator. The behavioural model of EDMU was built to intervene passively among these two agents. The tests aimed to verify if the DUT and EDMU were PCI compliant or not. There are two types of testing that were performed in order to verify that EDMU design is functional: system-level testing and PCI bus protocol testing. System-level testing set up modules in order to transfer data back and forth within the system (checking to see if the data was actually sent and whether it arrived at the destination or not). However, the correct operating procedure was not checked in system-level testing. PCI bus protocol testing was used to determine if the modules of the system (together with EDMU) operated within the rules of the protocol. PCI protocol tests were performed by EDMU after verifying its PCI compliance.

System behavioural simulation was a time consuming period of the modelling stage that required substantial effort. All the considerations and requirements were initially introduced in the simulation stage; the debugging required a very good understanding of PCI protocol issues as well as of reconfigurable hardware and programmable

network concepts. The intention during the initial design stage was to use Altera products for the final implementation of the system. However, technology issues that Xilinx FPGAs offer for further future development of this research project (i.e. partial reconfiguration of the FPGA device), made them a more suitable solution. Hence, Xilinx products were chosen for the realisation of the platform, although the EDMU was simulated for verification purposes with versions of Altera and Xilinx PCI cores using their respective PCI simulation test-benches.

The EDMU was simulated with the Altera PCI megacore function [122] using the Altera PCI testbench as it is represented in figure 6.1. A number of applications were used in order to test the validity and efficiency of the rules. The Altera reference design [123] is an indicative example of how to establish communication between an SDRAM controller and the PCI MegaCore function; this was used as a guideline for constructing the testbench. The principal application was X-MatchPRO [124] which is a hardware implementation of a high-speed lossless data compressor/de-compressor. The available access to the source code allowed us to make some necessary modifications in order to use this application during the behavioural simulation stage. Therefore, it was a suitable application for demonstrating the functionality of the system.

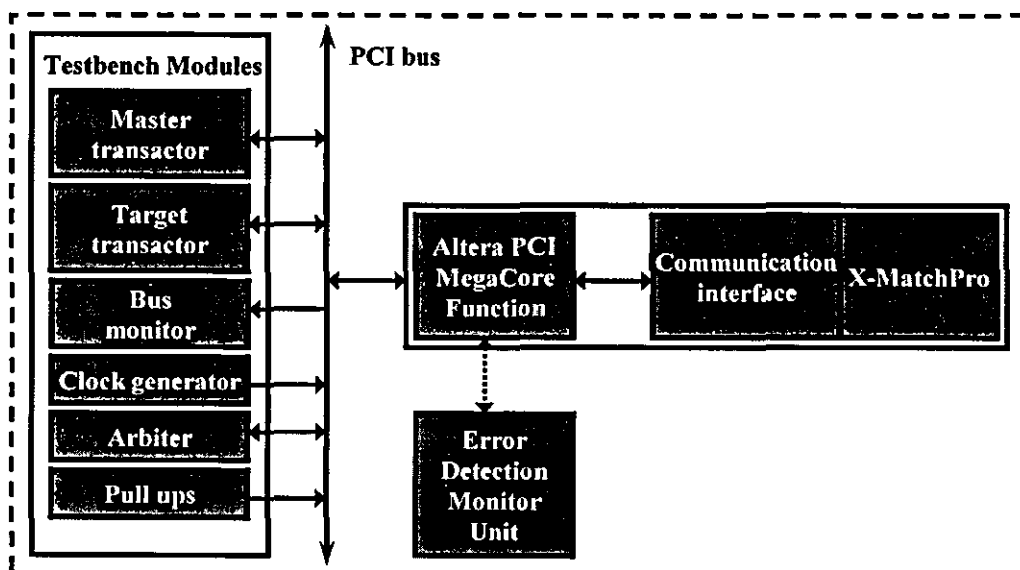


Figure 6.1: The Altera PCI simulation testbench.

The EDMU was also simulated with the Xilinx PCI functional simulation testbench [125] (figure 6.2). The validity and efficiency of the rules was tested using certain applications. For example, the PING64¹⁵ application was used to verify the design flow; PING64 design accepts data as a target PCI device and then uses the data to perform initiator transactions over the PCI bus. The simulation testbench contains HDL behavioural stimulus files and a top-level file, PCIM_TOP. The stimulus files contain behavioural models of an arbiter, two targets (one 32-bit target and another 64-bit target), and a central resource. These behavioural modules interface to PCIM_TOP through the PCI bus. The HDL wrapper, PCIM_TOP, combines three sub-modules. The first sub-module is PCIM_LC, a wrapper for the LogiCORE. The second is the PING64 application. The third is the CFG module, which configures the LogiCORE interface. The PING64 sub-module is an HDL design that interfaces to the user-application signals from the PCIM_LC module. EDMU is also integrated in this simulation testbench monitoring the activity between the user application and the PCI core. Figure 6.2 shows a block diagram of the complete system. Modelsim was used for the modelling and behavioural simulation of the EDMU core.

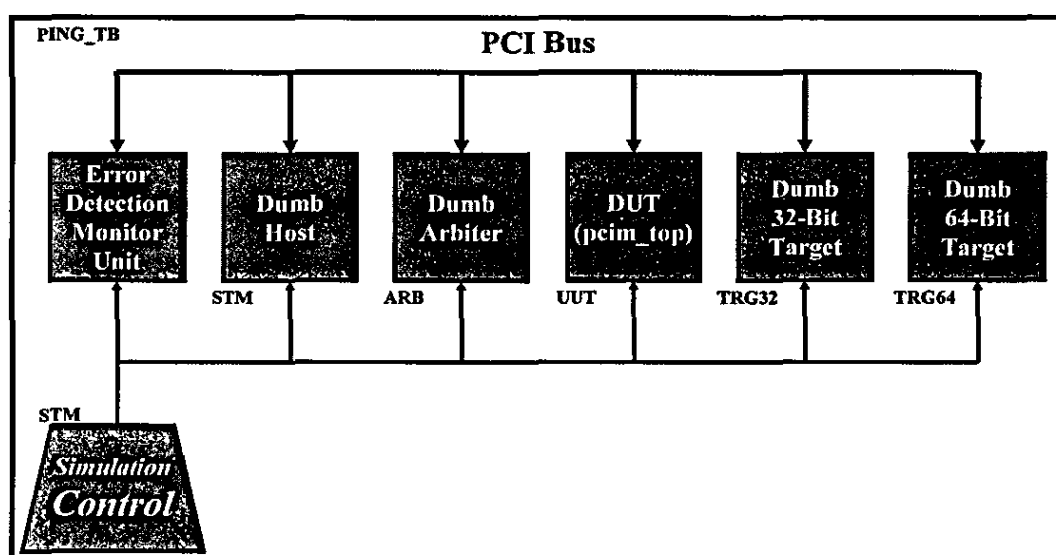


Figure 6.2: The Xilinx simulation testbench.

¹⁵ The PING64 module takes its name from the TCP/IP utility named ping which allows network users to verify that a particular machine is on a network and "alive". As such, PING64 is designed to provide "just enough" functionality to verify a design flow.

The access decode logic was used to generate configuration space, memory space, and I/O space select signals. This was achieved by monitoring the CFG_HIT and BASE_HIT signals which indicate that the current PCI transaction is directed at the PCI core configuration space or to an address space mapped by one of the Base Address Registers (BARs) in the PCI interface. In this design, BAR0 is configured as an I/O space, BAR1 is configured as a 32-bit memory space, and BAR2 is configured as a 64-bit memory space. EDMU is using BAR0 register in order to transfer the output values of the monitor as it is shown in figure 6.3 (i.e. signals base_hit, bar0_rd).

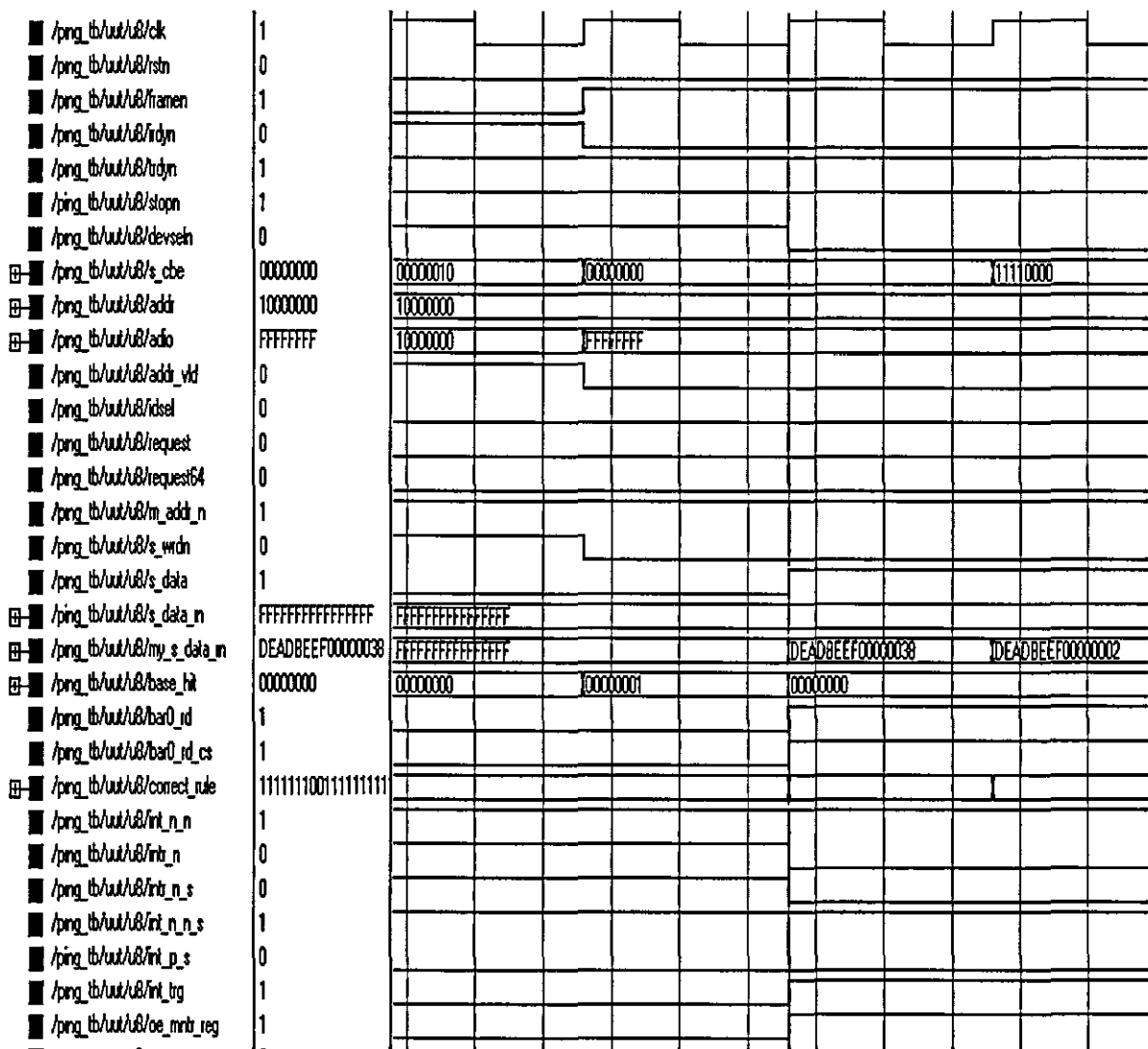


Figure 6.3: When the host sample BAR0 (i.e. signal bar0_rd) the output data of EDMU are transferred to the my_s_data_in vector signal.

The simulation timing diagram of figure 6.3 lists a series of PCI interface signals (e.g. `rstn`, `framen`, `irdyn`, `trdyn`, `devseln`, `stopn`), local-side PCI core signals (e.g. `addr`, `adio`, `addr_vld`, `request`, `request64`, `m_addr_n`, `s_wrtn`, `s_data`, `s_data_in`, `base_hit`, `bar0_rd`, `bar0_rd_cs`, `intr_n`) and just few EDMU signals (e.g. `my_s_data_in`, `correct_rule`, `int_trg`). EDMU detects a PCI protocol error¹⁶ which has the decimal encoded number 38. The same instance that the error is detected (i.e. change in value of `correct_rule`) BAR0 is sampled (the signal `bar0_rd` is asserted), and an interrupt is generated (assertion of `intr_n`, `int_trg`) The EDMU output data, which contain error identification information, are transferred to the host using the `S_DATA_IN` and `MY_S_DATA_IN` signals through a multiplexer. This does not always happen the same clock cycle that an error is detected but when the host generally attempts to read the contents of BAR0 register (i.e. assertion of `bar0_rd` signal).

The application testbench was deliberately set to produce errors. Two different methods were used to test the core in simulation mode. In the first case the application was changed in order to produce simple but at the same time highly undesirable events (i.e. PCI writes out of the predefined memory range). In the second case a simulation library was integrated [126] within the testbench in order to insert a degree of randomness in the system, generating random delays in the signals. This resulted in PCI protocol errors, which were detected by the EDMU. These two types of errors represent the two major categories that were mentioned in the previous chapter (i.e. PCI protocol errors and common application errors).

Another indicative capture of simulation results is given in figure 6.4. This is also a PCI protocol related error (during the address phase for a non-b2b transaction, `irdy#` is not to be driven immediately following an idle). The error severity rating for this policy violation is low and therefore EDMU is applying an error tolerance practice. This waveform capture of the system simulation provides additional evidence of a significant property of EDMU: the per-cycle basis monitoring of the configured application. When an error is detected ($t=2948\text{ns}$, figure 6.4), the “`correct_rule`” is instantly changing value. At the same clock an interrupt (`int_trig`) is generated to signal

¹⁶ Once target has asserted `stop`, it cannot change `devsel`, `trdy`, or `stop` until the data phase completes. This `rule` is divided into 3 sub-rules that describe each individual case.

the host. On the next clock the value is passed to the register (out_reg) that is memory mapped to BAR0. The host at the same clock can sample BAR0 and get a view of the activity in the PCI-based FPGA platform as far as the policy framework is concerned (this is not taking place at this instance). This figure shows therefore the way that vital information concerning the output of the EDMU can be passed in an almost concurrent way to the host.

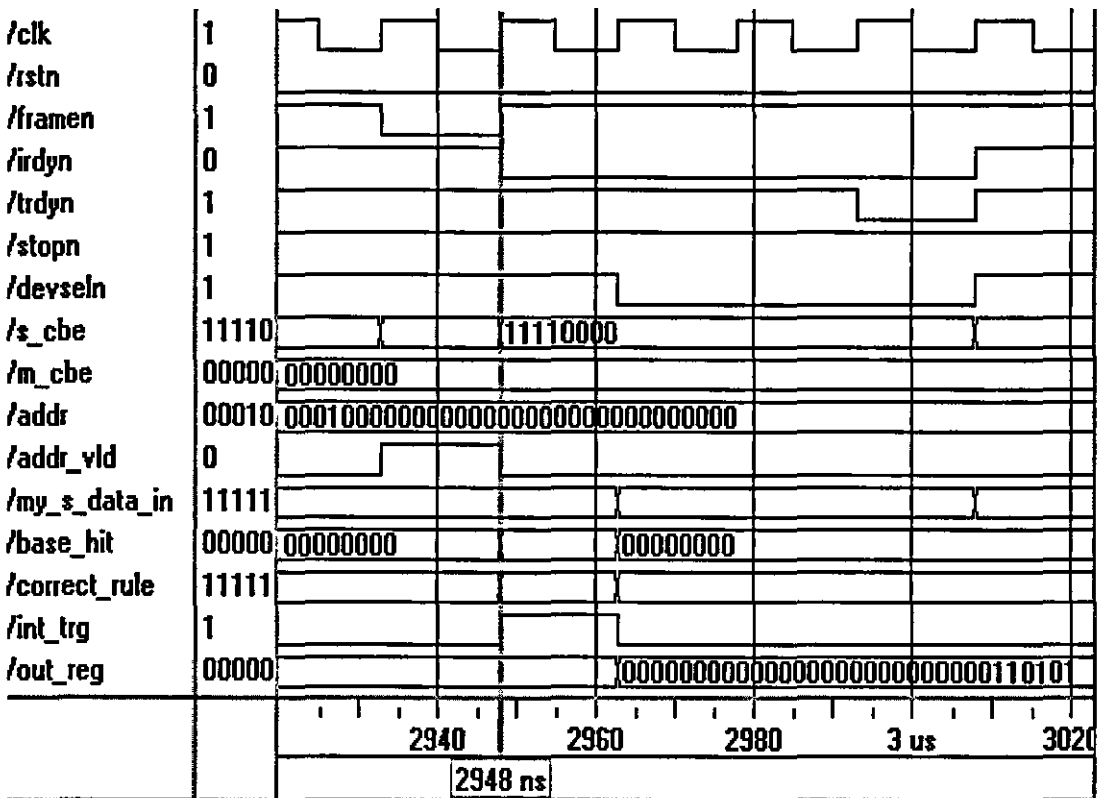


Figure 6.4: The detection of a PCI protocol error in simulation (error tolerance).

The response time of the control component of EDMU during simulation time had to be tested as well. The ability of EDMU to stop the configured application when an error is detected was an equally significant feature that had to be successfully implemented. The obvious goals of the control component are to prevent the propagation of the error in the system and also to prevent transactions (initiated by the configured application) that can later produce an erroneous condition.

The host is able to sample the value of this register since it is memory-mapped in a memory area of BAR0. However, in this case it cannot apply an error tolerance policy because the detected error was pre-classified as a serious one. Consequently the control component of EDMU forces the application to an “abort” (i.e. target abort). As a result the user transaction cannot take place (trdy is not asserted) and the application issues a target abort. This is achieved by taking advantage of control signals on the local side of the PCI core (e.g. S_ABORT a “registered” signal of the local side of the PCI core). The signal mapping of the top-level vhdl file was also changed accordingly. An indicative description of a target abort sequence as this is described in the Xilinx PCI LogiCore user guide is given on *Appendix IV*.

Considering the functionality of the system, it can be claimed that EDMU observes the outputs of the design under verification, flags behavioural errors and assigns blame to the appropriate module during a system-level simulation. The EDMU was originally designed in order to be integrated in a real-time system for fast error detection, rather than for verifying other PCI-based applications. However, this fact reveals another important use of the hardware monitoring component.

6.3 Real time verification of the Error Detection Monitor Unit

Proof of concept for this work was established by constructing a real time testbench using commercially available network and FPGA components. Xilinx products were selected for the implementation of the system; various technology reasons related both with the features¹⁷ of the Xilinx FPGAs and the Xilinx PCI core (i.e. the easier access to the local side signals) made us chose them at that specific time. The current version of EDMU was synthesized with Synplify Pro together with the Xilinx PCI64 core [127] (version V3.0.137). The EDMU performance was tested at 66 MHz although in most application examples the maximum frequency was 33 MHz.

¹⁷ The partial reconfiguration property of Xilinx FPGAs and the variety of embedded “hard-wired” cores (i.e. gigabit transceivers, DSP modules) and specifically the 32bit/33MHz “hard-wired” microprocessor.

Family	Device	Fmax	Slices	GCLK	BUFTs+BUFEs	Register bits	Design Tools
Virtex II	XC2V1000-5 (FG456)	66 MHz	895 (8%)	2 of 16 (12%)	64 of 2560 (2%)	I/O: 64 Non-I/O: 388 (3%)	Synplify Pro 7.3.1

Table 6.1: Implementation results for the EDMU.

The selected FPGA family for the synthesis and implementation of EDMU was the Virtex II (XC2V1000). Table 6.1 summarises the relevant synthesis information for the Error Detection Monitor Core. The Xilinx ISE design tools were used for the place and route process¹⁸.

After the implementation, the Xilinx simulation testbench was used once again for a back-annotated timing simulation. This simulation verified the functional and timing correctness of the behavioural model of EDMU. However, the final and most significant part of the EDMU development and verification is the testing of the system in the actual FPGA PCI board using a real time testbench.

A Xilinx based PCI board [128] was chosen for the realisation of the platform. The selection of the board follows the analysis that was made in section 5.1 and as is shown in figure 6.6 this board satisfies most (if not all) of the design, implementation and future extension requirements. Various other commercial PCI board-level solutions were available at that time. However, it was the optimal choice considering the design requirements, the properties and the specifications of this board (e.g. double PCI FPGA board with a "soft" PCI core, various I/O interfaces and extendibility design options through the embedded microprocessor).

The board (ADM-XPL) supports Xilinx Virtex-II PRO devices, which have a "hard-wired" embedded PowerPC processor. The board utilises a FPGA PCI bridge that supports 64 bit PCI at up to 66MHz. With some enhancements it can also provide compatibility with PCI-X protocol. A high speed multiplexed address and data bus connects the bridge to the target FPGA. Memory resources provided on-board include

¹⁸ Total number of slices. 4875 out of 5120 (95%)
 The average connection delay for this design is: 1.190 ns
 The maximum pin delay is: 6 216 ns
 The average connection delay on the 10 worst nets is: 4.979 ns

DDR SDRAM, pipelined ZBT and flash, all of which are optimised for direct use by the FPGA using IP and toolkits provided by Xilinx.

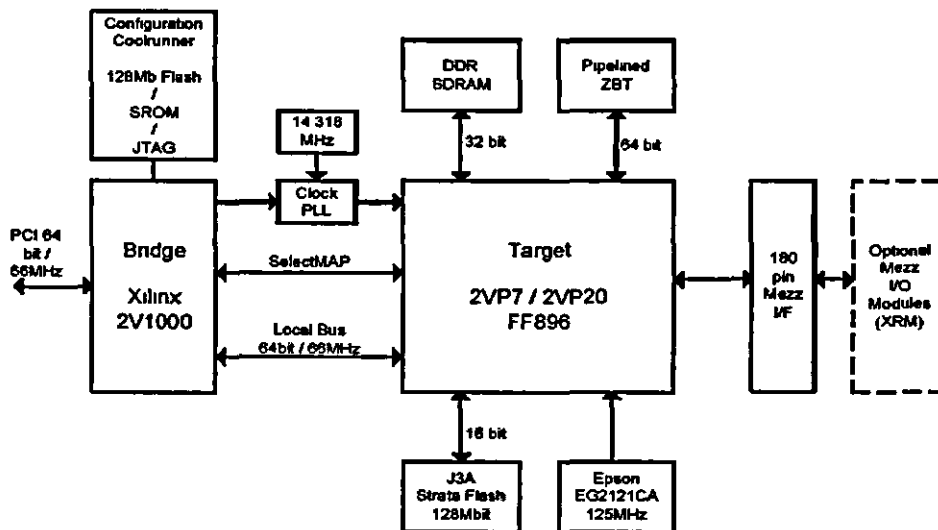


Figure 6.6: The ADM-XPL board topology [129].

The ADM-XRC SDK that comes along with this board is a set of resources including an application-programming interface. The API makes use of a device driver that is normally not directly accessed by the user's application. The API library takes care of open, close and device I/O control calls to the driver. A set of simple, open-source applications is also [130] included within the above mentioned SDK. The following application list helped to build a real-time testbench and verify EDMU in real operating conditions.

- The "DMA" application demonstrates demand mode DMA, using the DDMA¹⁹ sample FPGA design. The application works as follows: it loads the DDMA bitstream into the FPGA, using a DMA transfer. Then it creates two DMA buffers; one for the 'send' direction (host-to-FPGA), one for the receive direction (FPGA-to-host). In the next step it creates a 'sender' thread, which performs demand-mode DMA transfers from the host to the FPGA, using the host-to-FPGA DMA buffer. Finally, it creates a

¹⁹ The DDMA FPGA design demonstrates demand-mode DMA with bursting. Data is read from an application buffer in host memory and then simply written back to another application buffer unchanged. In order to use demand-mode DMA, the host must specify the appropriate mode when performing DMA transfers. This is demonstrated by the DMA sample application.

'receiver' thread, which performs demand-mode DMA transfers from the FPGA to the host, using the FPGA-to-host DMA buffer. The DDMA FPGA design simply performs a loopback operation, placing data from the send thread in a FIFO, to be read out by the receiver thread. The receiver thread checks the data received for correctness.

- The "Master" FPGA design demonstrates direct master access by the FPGA to host memory. The design implements several registers for generating Direct Master transfers to and from host memory. The application allocates a user-space buffer. It then obtains a scatter-gather map of the buffer. Finally, it initialises the user-space buffer to contain known data and waits for the user to enter commands.
- The "Memtest" application uses the ZBT FPGA design to test the SSRAM of the board. The ZBT FPGA design demonstrates how to implement a host interface to the SSRAM in an FPGA design. The design divides the 4MB FPGA space into a lower 2MB region for register and an upper 2MB window for accessing the SSRAM. A page register is provided so that all of the SSRAM on a card is available to the host.
- The "Simple" application, which demonstrates how to implement host-accessible registers in an FPGA design. The registers can be accessed via API calls, or via a memory-mapped region. The user enters hexadecimal values, which the application writes to a register in the FPGA. The application reads the values back from the FPGA and displays them. The FPGA nibble-reverses the values before returning them.
- The "DLL" application demonstrates the clock doubling capability of Virtex DLLs and Virtex-II DCMs. The user gives a frequency for the local bus clock on the command line. The application sets the local bus clock frequency to the value specified, which is doubled and used to clock a 32-bit counter. The application reads the counter once per second, displaying the difference between the current and last readings.

The above set of applications was part of the testing environment of the PCI board. Having free access to the source code, their content was deliberately modified in order to produce erroneous outputs. ChipScope Pro was used in order to apply advanced real-time debugging and verification of the system, by inserting soft debug cores into the

top-level design file. After place and route, these on-chip cores were used to capture real-time signal data. This hardware debugging tool was proved to be very helpful for validating the system functionality, operability and effectiveness.

Figure 6.7 is an indicative waveform capture of the monitoring process of EDMU. ChipScope Pro Analyzer was used for this reason. After applying some small changes in the design of the “master” application (that required re-compiling and re-implementation of the application), a state that is classified as an error for EDMU was deliberately produced (i.e. an illegal access during a write cycle, whenever irdy# is asserted, and the bus is driven by the master).

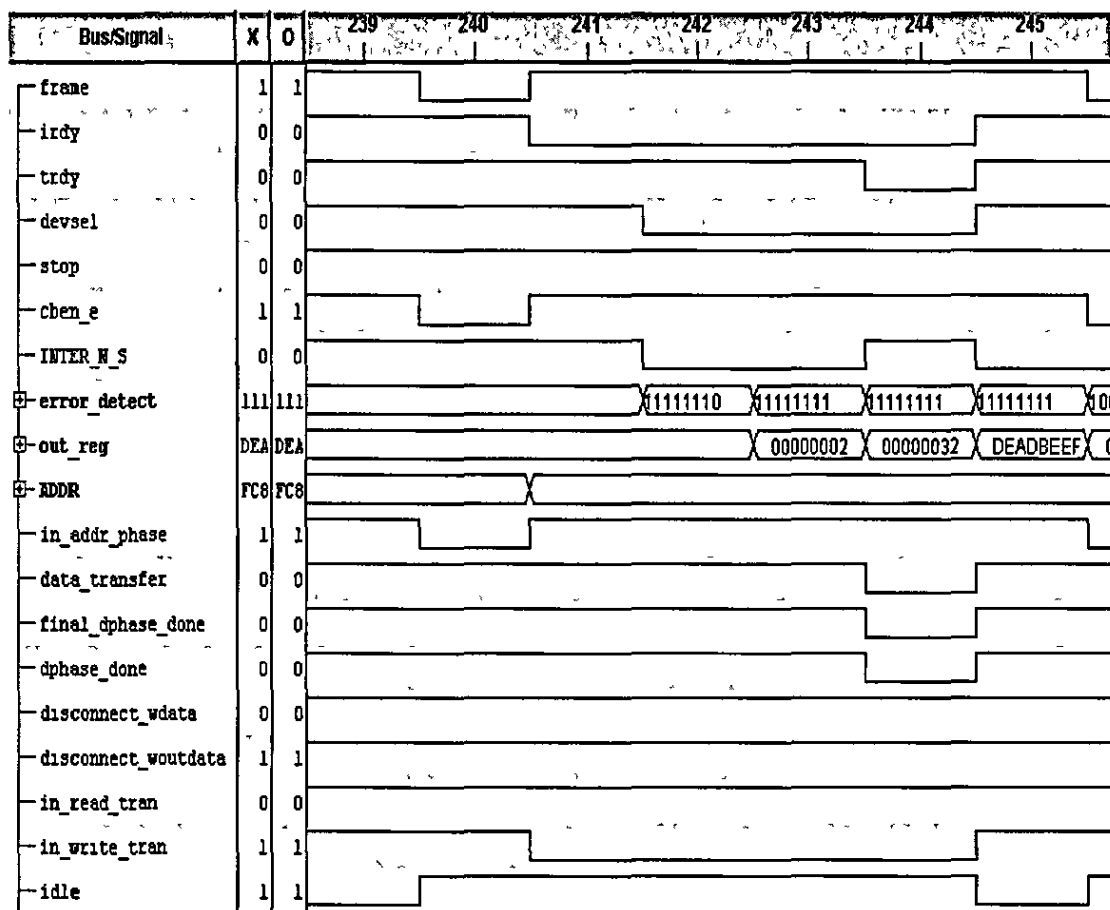


Figure 6.7: Providing error-tolerance in the system operation.

An indicative notice is that the signal “error_detect” remains asserted because another error (PCI protocol this time) is detected after the detection of the first one (i.e.

for b2b transactions since the master needs to be the same for the second transaction). However, the severity rating of both errors is pre-classified as low. Therefore, no action is taken since an error-tolerance policy is applied for these specific errors. This waveform represents signal values captured when the system is operating in real-time; a careful comparison with the simulation results indicates that the EDMU is functioning in a very similar way. An interrupt is generated (assertion of signal INTER_N_S) at the same time when an error is detected (error_detect). The next clock cycle the data are written to the outout register (out_reg) ChipScope Pro Analyzer also gave us the opportunity to verify the correct functionality of the auxiliary signals of EDMU as well as of the monitor-event signals (i.e. in_addr_phase, data_phase, in_write_tran etc).

A similar waveform that was captured from the ChipScope Analyzer is shown in figure 6.8. The different combination of events that is signalled as an error, includes a “disconnect with data” monitoring event.

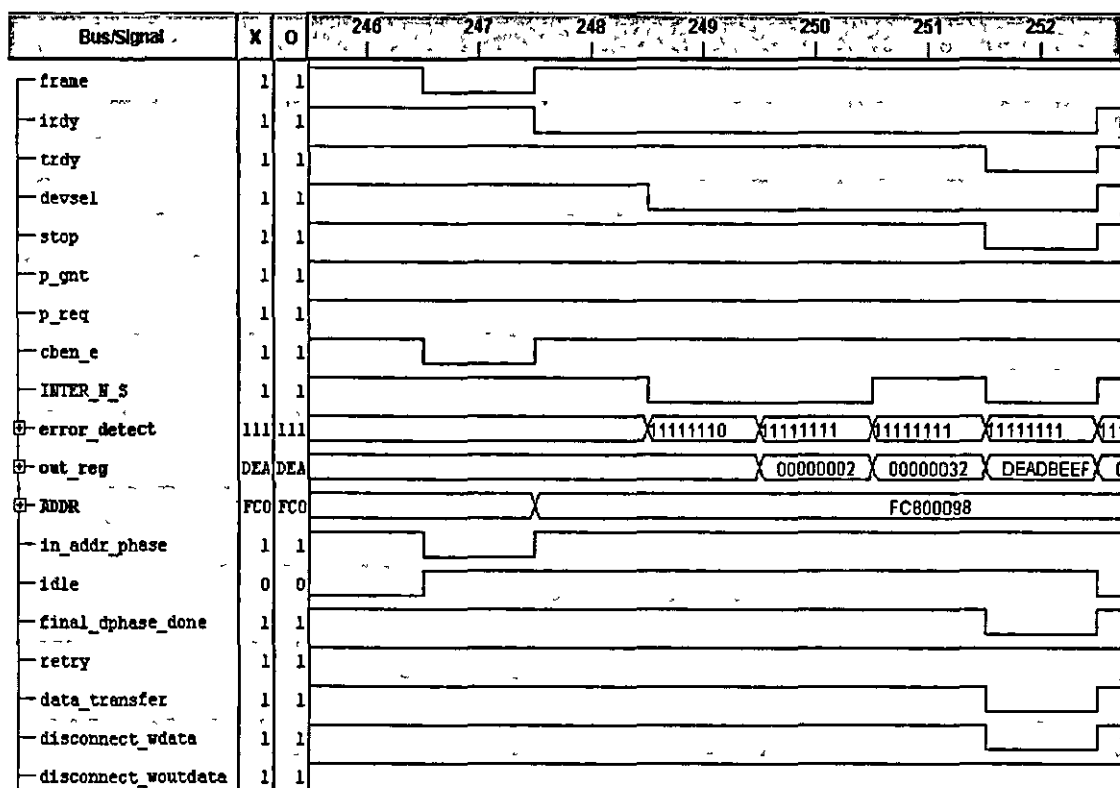


Figure 6.8: Another case of error tolerance.

The reaction of EDMU is the same with the one described above, applying an error-tolerance policy. One of the auxiliary signals, is asserted indicating a “disconnect with data” PCI event. Disconnects with or without data is a common event during PCI transactions, but it has to be monitored in order to prevent undesirable consequences as these were described in Chapter 4.

The value of the output interface register (namely `out_reg`) that is memory mapped at a certain memory area of BAR0 (i.e. `0x7C`) can be sampled by the host. A freeware program was selected (from the various available), in order to apply this memory read. PCItree [121] is a graphical Windows tool that allows access of all the hardware devices of the PCI bus. Information about the devices and its vendors is obtained from a separate database. PCItree allows read and write access to the configuration registers of each device and even to each device's memory given by the BAR. This tool was used as an additional debugging and verification aid for testing the custom PCI core. The BAR0 memory read gives us a qualitative view of the errors that are detected by the EDMU (figure 6.9). The memory read at location `07xC` gave the hexadecimal value (`x00000002`) that appears also in figure 6.7. However, this BAR0 memory read takes place in a later stage; this hexadecimal value represents the same error, happening at a different time.

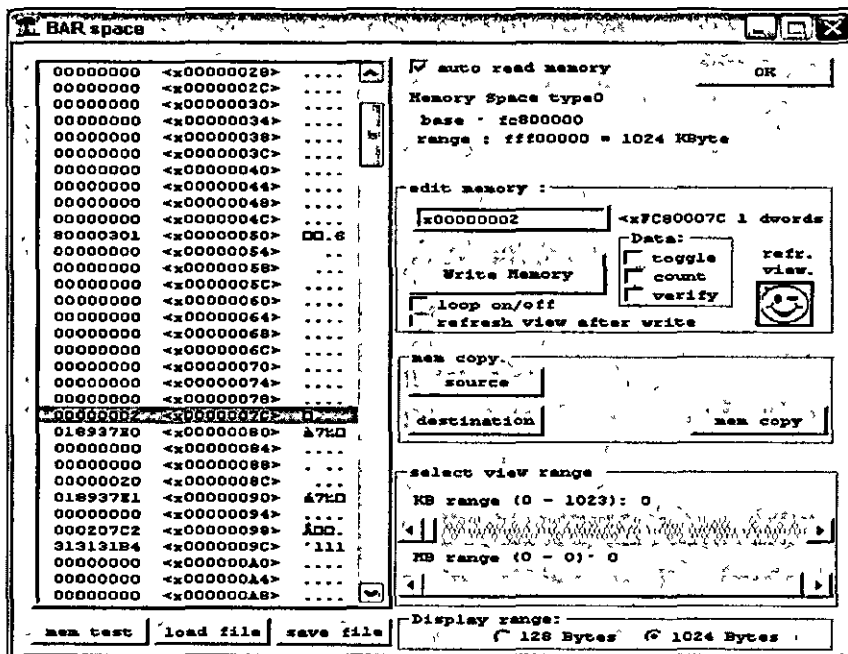


Figure 6.9. The PCItree application “reads” the BAR0 memory space.

Figure 6.10 is another indicative capture of a waveform from ChipScope Analyzer, when a corrupted version of the “memtest” application is configured in the reconfigurable board. The application is trying to access a restricted memory area of the host (i.e. actually, it is the area that was predefined as restricted). This is considered to be an error with high severity rating. The reaction therefore is almost immediate (i.e. according to the details presented so far, the reaction comes one clock cycle after the actual occurrence of the error). Such errors need to be treated drastically, by limiting or ideally, completely stopping their effects. For this reason, EDMU forces the configured application to issue a target abort.

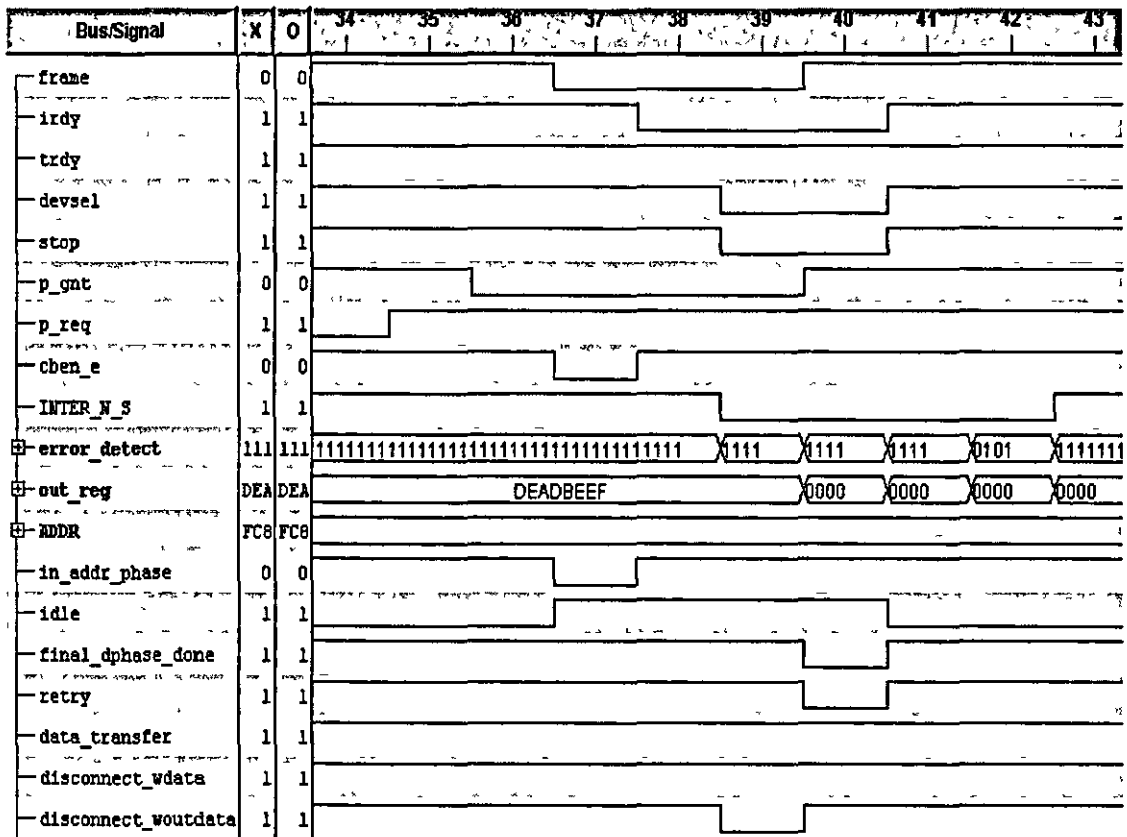


Figure 6.10: A data capture from the ChipScope Analyzer.

The control of several signals (i.e. the s_abort signal of the local PCI core side) with the appropriate signal mapping in the top-level file of the implemented system has as a result that trdy# remains de-asserted, because the EDMU is configured to prevent such an event. As a consequence, the application that is configured in the application FPGA

fails to transfer data in a given time, leading the PCI transaction to disconnect without data and issue at the same time a “retry”. An additional indication of this is shown by an auxiliary EDMU signal (i.e. data_transfer remains de-asserted). This has, as an eventual result the detection of multiple errors (after the detection of the first one), which are related to the “abort” event. Nevertheless, these errors have lower severity ratings. An output vector (error_detect) updates an internal register (out_reg), which is memory mapped to BAR0. An interrupt is generated (inter_n_s) when the errors occur, in order to signal the host. The interrupt line remains asserted during the detection of the consequent errors. It can be claimed that the particular error described above is not malicious, since its impact is not related with a fatal failure of the system. However, it could be connected with application hijacking or other illegal activities related to security issues as was described in Chapter 4.

An example where the control component of EDMU applies both error tolerance and abort policy is shown in Figure 6.11. Violations of policies with different severity ratings are detected.

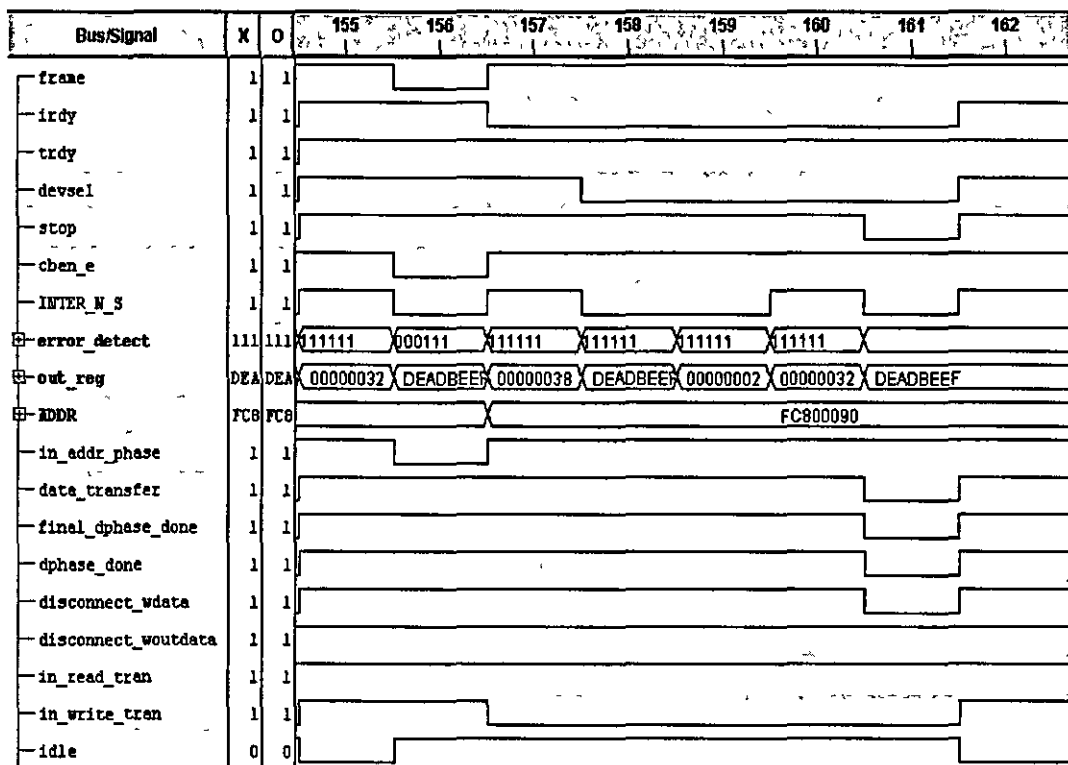


Figure 6.11: Multiple errors with different severity rating are detected by EDMU.

The most serious one comes last and as a result the execution of the application is interrupted because the application is forced to issue a target abort. As a result the application issues a “disconnect with data”. Following the detection of an important error and the consequent target abort that is initiated by EDMU, the application FPGA is reconfigured with a stable application.

Figure 6.12 is a representative screenshot that shows how the execution of the “memtest” application is affected by the way that EDMU is acting in order to stop a detected illegal process.

```

C:\ADMXRC_SDK4.5.1\bin>memtest
Size reg = 0x00000002
Info reg = 0x01100000
Status reg = 0x00000003

Performing memory tests using DMA
Testing 2048 kB in pipelined mode...
Repetition 0
Writing 0x55555555...
Error at index 0x00000000: expected/actual = 0x55555555/0x00000000
Error at index 0x00000001: expected/actual = 0x55555555/0x00000000
Error at index 0x00000002: expected/actual = 0x55555555/0x00000000
Error at index 0x00000003: expected/actual = 0x55555555/0x00000000
Error at index 0x00000004: expected/actual = 0x55555555/0x00000000
Error at index 0x00000005: expected/actual = 0x55555555/0x00000000
Error at index 0x00000006: expected/actual = 0x55555555/0x00000000
Error at index 0x00000007: expected/actual = 0x55555555/0x00000000
Error at index 0x00000008: expected/actual = 0x55555555/0x00000000
Error at index 0x00000009: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000a: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000b: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000c: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000d: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000e: expected/actual = 0x55555555/0x00000000
Error at index 0x0000000f: expected/actual = 0x55555555/0x00000000
Error at index 0x00000010: expected/actual = 0x55555555/0x00000000
Error at index 0x00000011: expected/actual = 0x55555555/0x00000000
Error at index 0x00000012: expected/actual = 0x55555555/0x00000000
Error at index 0x00000013: expected/actual = 0x55555555/0x00000000
Ignore errors... ]
*** FAILED with 524288 error(s)
Performing byte enable tests using programmed I/O
Testing byte enables in pipelined mode...
PASSED

Measuring access speed...
SSRAM to host throughput = 72266.0MB/s
Host to SSRAM throughput = 71159.0MB/s

```

Figure 6.12: The execution of “memtest” is interrupted due to error detection.

The reconfiguration of the application FPGA is accomplished either by using the ADM-XRC SDK application interface (i.e. building a C++ interface that includes SDK functions in order to program the devices through the PCI bus) or through the JTAG

port using a parallel cable connecting the parallel port of the host directly to the JTAG connector of the board together with the Xilinx iMPACT²⁰ (figure 6.13).

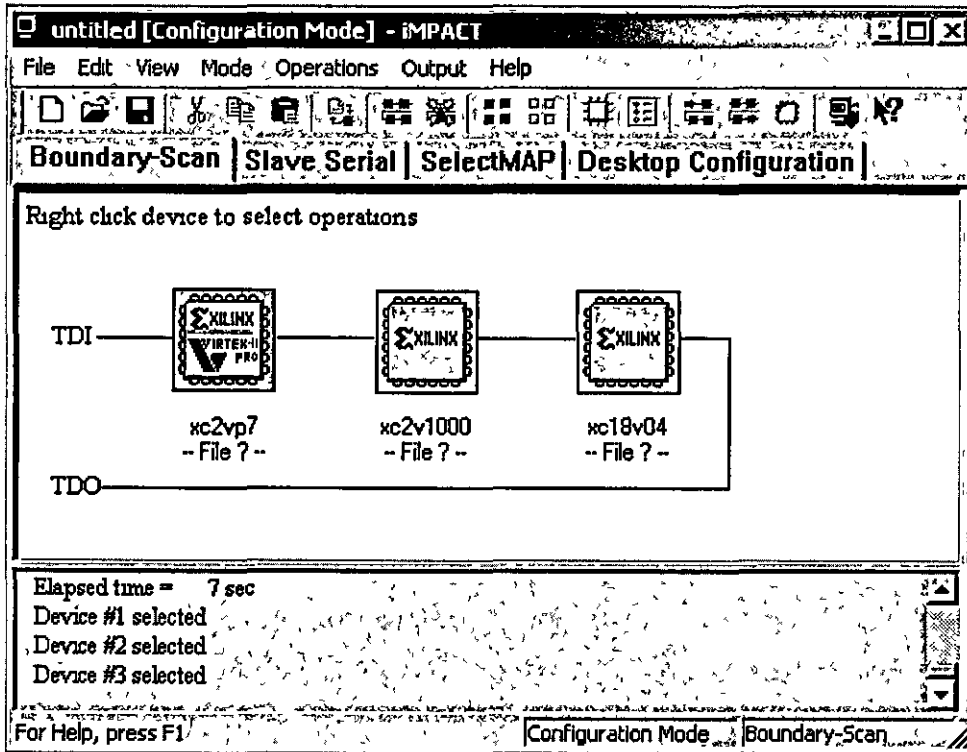


Figure 6.13: The JTAG chain shows the 3 programmable devices of the ADM-XPL board: the application FPGA, the PCI core/EDMU FPGA and the EEPROM.

The easiest and most straightforward solution is to use a script that initiates the JTAG chain. Once the chain has been fully described, it can be saved for later use in a file with .cdf extension. The choice of reprogramming the FPGA devices through this way is preferable (in similar dependable systems) compared to programming them through the PCI bus; the reason behind this selection is the fact that the (erroneous) configured application in the user FPGA is likely to produce errors (PCI protocol or application errors) that will prevent or harden this effort when application replacement is attempted. Although this could only happen under extreme erroneous conditions, it is still preferable to apply the reconfiguration of the FPGA devices through the JTAG port (bypassing the PCI bus), in order to satisfy the dependable computing features of the

²⁰ iMPACT enables you to configure PLD designs through four modes of configuration; Boundary-Scan, Slave Serial, SelectMAP, and Desktop Configuration.

system (the JTAG pins are used for reconfiguring devices of a board or for communicating with embedded cores through hardware Analyzer software like ChipScope Pro). Bus contention conditions could also deteriorate while trying to use the PCI bus for downloading the new configuration bitstream. The reconfiguration through the JTAG port using the Xilinx Impact software is slower compared to the reconfiguration through the PCI bus; however it is reliable enough to serve the scope and the functionality of the EDMU.

The effectiveness of the core was tested and verified using this real time testbench. A stable version was passively integrated with the Xilinx PCI core, offering a robust error monitoring reconfigurable platform. The simulation and real-time results prove that the outcome of this research satisfies the required and desirable system properties (as described in Chapter four) of a PCI-based reconfigurable hardware platform.

6.4 EDMU and Active Networks

Modern System-on-Chip and embedded FPGA-based systems are exhaustively tested and are built specifically for serving testability targets. Part of the evaluation of the system could also be its formal verification²¹. Although formal verification is useful for static verification of system functionality or the validity of a protocol, there is a variety of critical systems that require online monitoring, control and debugging. The external interfaces that were once used extensively for development of System-on-Chip (SoC) based designs have moved on-chip, leaving few points to attach external tools such as logic analyzers. One approach to overcoming development challenges is on-chip monitor support circuits that allow observation of critical system nodes, such as processor cores and buses. Important tools include on-chip trace and triggers. Many interactions within a SoC do not involve a processor core, making data tracing essential. One aspect of the delivered research as far as EDMU is concerned is the formation of a generic PCI-based approach that is compatible with many different system units, and supports design reuse.

²¹ In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property, using formal methods. Formal methods refer to mathematically based techniques for the specification, development and verification of software and hardware systems.

Therefore, EDMU can be integrated in many critical computing environments due to the operation features and the advantages that offers. Various networking PC structures which make use of PCI boards for processing and/or forwarding of packets can take benefit of the EDMU. Online PC systems such as Active Routers/Nodes which are part of programmable and/or Active Networks need to:

- operate constantly,
- operate uninterrupted (i.e. avoid "hard" re-starts),
- operate under the presence of errors and
- demand fast error detection and error isolation/recovery.

The experimental Active network that was implemented at Loughborough University [87] (as it is also presented in Appendix I), can integrate the EDMU in order to monitor and control erroneous and malicious applications. The active node is a fundamental unit of the Active Network which is also part of a standard IP network. Thus, it is configured to work as a router for the standard IP packets, applying at the same time active processing to these IP packets (Active Packets) that either carry executable code in their payload or request an application that is pre-stored (locally in the active router or in a dedicated application server). The Active Applications are downloaded and configured in a FPGA device. The EDMU can add observability in this system by monitoring these applications in real time or in a nearly concurrent way. The system can also continue to operate in the presence of detected errors, as long as the errors do not interfere with vital functions of the system. The EDMU can initiate an application abort sequence, during which the application is not able to communicate with the host microprocessor (the data transactions fail as it is shown in figure 6.12). Next, the erroneous application that is configured in the application FPGA could be replaced with a new configuration (i.e. bitstream). This could be achieved either through the Linux interface presented in [87] (through the PCI bus), or through the JTAG chain as it was presented in the previous section (see also the indicative figure 5.1). The use of JTAG port for reconfiguring the FPGA erroneous application is slower but more reliable and hence more preferable for guaranteeing system stability (i.e. the reconfiguration process overpasses the PCI bus functionality).

The EDMU can be considered as a very useful component of critical PC-based systems; Active Networks can become more robust ensuring the delivery of services in users and therefore provide maintainability and availability to the whole network operation.

Summary

This chapter covered the simulation and real-time debugging results. The functionality of EDMU was verified and tested successfully and a stable version is running on a FPGA-based PCI board providing timely detection of PCI protocol and application errors. The monitor core is applying different policies to the erroneous application which are configured in one of the FPGAs of the PCI board. As a result, it could be claimed that the operation of EDMU resembles that of a “firewall”.

CHAPTER 7

CONCLUSION

7.1 Synopsis of achievements

This work has followed the rationale that the convergence of the computing and communications fields applies to reconfigurable computing and reconfigurable networking. The increasing design-complexity of FPGA-based reconfigurable platforms (i.e. System-on-Chip) makes observability and controllability a significant factor for their proper and correct running, especially when such systems are integrated as part of critical computing application or services. System validation methods such as offline testing, debugging and formal verification have to be complemented by online monitor and control functions which are performed by embedded system components. The research motivation that supports this work claims that the critical computing systems, based on PC architectures, should include in their infrastructure a permanently configured hardware monitor component. This should not introduce any additional costs (i.e. area utilisation) as far as the execution environment of the application is concerned. Hence, it has to be integrated passively, consuming as few local resources as possible. The reasons for choosing a hardware monitor and control module for critical PC-based systems instead of a software or hybrid one, were stated in chapter two (sections 2.2, 2.3, 2.4, 2.5, 2.6). The inherent process-parallelism of the FPGAs satisfied the requirement for immediate response to errors, in order to prevent their propagation (or their trespassing) to security sensitive domains and thus prevent system failures in this way. The concurrent system observability through embedded hardware components is an emerging research area that meets the needs of complex critical PC systems

Although Active Networks were used as the source-model for building the monitor and control core (essentially a PC-based Active Router with FPGA-based hardware

acceleration of the Active Applications), the policy framework can be considered non-deterministic and platform independent. It is true that the hardware monitor logic was built according to the specifications, performance requirements, potential failure operation-modes and the security threats of an Active Router. However, there are many other conceptual domains that EDMU can be applied; similar architectural requirements are found in critical systems that are comprised not only by PCs but SBCs as well. The Error Detection Monitor Unit can be thought of as a monitor and control wrapper for the PCI core that achieves observability of the application transactions with the host. Several PCI-based environments can take advantage of the characteristics of EDMU and integrate it in their structure or even expand its current functionality and operation. EDMU also secures quality in the offered services of the PC-based critical system, since it provides maintainability, availability and dependability features. The constant and simultaneous monitoring (i.e. implemented in hardware) traces the activity between the configured application and the host. Suspicious or undesirable transactions are treated according to the policy framework and the host can maintain its operation; this was one of the main targets of the research and development effort. The following lines include a short description of the characteristics of the EDMU and the achievements that it claims:

- *Reliability*: A PCI-based reconfigurable platform that can be integrated in several critical computing systems (i.e. remote upgrade of FPGAs, Active Networks) and is able to add reliability and operability features.
- *Reusability*: A reusable plug-in module suitable for "soft" PCI cores that adds observability and control features to PCI-based applications.
- *Policy framework*: A failure-resistant policy framework that includes an incremental list of rules, applied to the configured applications of a PCI-based FPGA board.
- *Interference*: A hardware module configured permanently in the programmable logic platform that operates transparently without interfering with the host PC's computational processes.
- *Detection response*: Detection of predefined erroneous conditions that can result in a potential system failure in a nearly concurrent mode of operation.

- *PCI protocol monitoring*: PCI bus transactions monitoring system, tightly coupled with the PCI core, detecting errors initiated by applications which misuse the PCI bus protocol.
- *Categorisation*: System threat identification and classification.
- *Control*: An embedded control component that acts as a “firewall” preventing the communication between the configured application and the host computer, if a policy-violation is detected.
- *Error tolerance*: The communication between the FPGA board and the host is selectively controlled, following an error-tolerant policy.
- *Reconfiguration*: Recovery of the system to a stable state is possible after the detection of an error.

Concluding, it is useful to mention that the EDMU can be used in various ways; it can be integrated and specially configured to serve different design or system-operation targets. The analysis made so far, revealed different applications approaches for using EDMU. The most apparent ones are mentioned in the following lines:

1. Offline FPGA-based, PCI applications debugging, testing and validation. The EDMU can also be thought as a design-for-testability embedded component,
2. System-level integration²² and online operation of the hardware monitor core in critical computing systems.
3. A useful “soft” PCI core plug-in module that can be adapted to fit specific design and/or operational needs.

The research and development results delivered through this work claim a useful contribution to the field of concurrent hardware monitoring applied to dependable PC-based systems with FPGA application support. The outcome of this research in its current stable version can be the starting point for answering more research questions. The growth of the FPGA market and the respective applications, along with the key role of SoCs in future critical systems make observability an important factor. The table 7.1 includes the most significant properties of EDMU as far as their efficiency and impact to the system are concerned.

²² The design and operation requirements of the candidate systems have to be similar to those of a PC-based Active Router or any other networked reconfigurable FPGA PC architecture

EDMU properties	Property effect	Comment
Perturbation	Minimal	EDMU is passively integrated with the PCI core and thus it does not interfere with the application that is configured in a separate FPGA. The overhead of the area utilisation does not surcharge the application execution domain.
Performance	High	The selection of FPGAs for application-processing and application monitoring, secures high performance of the hardware monitor due to the inherent process parallelism of the FPGAs.
Error detection/ policy violation	Fast	Most of the erroneous events are detected almost concurrently, one clock cycle after their occurrence. Several other policies are based on counter-based measurements and their effect follows error-tolerance procedures.
Preventing error propagation	High	If the detected error has a predefined high severity rating, EDMU acts as a firewall by blocking the communication of a configured application with the host, one clock cycle after the detection of an error.
Area utilisation	Low	EDMU was deliberately designed to occupy low FPGA area in order to satisfy the firm area utilisation requirements of dependable systems.
Portability/ reusability/ compatibility	High	The hardware monitor can be easily integrated with other "soft" PCI cores (i.e. by designing a wrapper file). EDMU is also operating system independent. Any configuration environment that uses the PCI protocol, and thus a "soft" PCI core, can integrate EDMU as an embedded monitor component (i.e. PCs, SBCs).
Extendibility	High	The predefined policies can be modified in order to follow a stricter framework. The simple structure of the hardware monitor component allows the integration of additional policies. The design of EDMU can be easily extended in order to include the PCI-X core features
Reduction of cost	High	The combination of the reconfigurable nature of the FPGAs with the monitor and control features of the EDMU minimises the need for spares, replacements and field testing; at the same time the system reliability is increased. Thus, the critical computing systems that integrate this PCI board configuration can achieve a cost-reduction.

Table 7.1: A synopsis of the most important properties of the EDMU.

7.2 Future Work

It is clear that any research work of this type can undergo considerable improvements and adaptations in order to achieve expansion of the functionality, better performance and greater portability and reusability. Many other related research fields can also take advantage of the current results. Taking into account these facts the hardware monitor was built in an incremental and simple way. The outcome is a hardware component with extendibility options and properties.

One obvious extension to this work could be a comparison with software based bus-monitor modules and error detectors. This comparison can underline the usefulness and the need of an embedded core for monitoring and controlling the transactions between the host and the PCI board.

The EDMU can be enforced with more policies that will reflect the needs of other systems, or more complex needs of the currently proposed target architectures. The structure of the EDMU allows the easy system-level introduction of new rules. These rules will refer mainly to application initiated errors and not to PCI protocol errors since this study claims that the PCI protocol is verified in real time. The current simulation and real-time testbenches offer easy integration and testing of the potential new policies.

Another significant expansion to this research work could be the real time performance analysis of the configured FPGA-based application. This can involve statistical analysis of several data samples that are collected at a specified rate or behavioural analysis of the application. The implementation of such an added mechanism in hardware, will require substantial amount of resources (i.e. embedded memory, LUTs). The current design does not occupy significant resources of the FPGA as was shown in table 6.1. Hence, the inclusion of on-chip performance analysis features on the existing EDMU structure will be an additional design, integration and cost issue, since it requires the use of state-of-the-art large FPGA devices. Alternatively, the data can be passed to the host and the whole process can be implemented in software. In this case the overhead in area utilisation is smaller (i.e. FIFO, interface with the host) but the host is overloaded with this processing consuming function.

A very useful and interesting extension to this research work could be the integration and cooperation of an embedded "soft" or "hardwired" microprocessor for delivering a hybrid monitoring scheme. There are many different ways that the microprocessor can be used in conjunction with the EDMU. For example the performance analysis can take place in the local embedded microprocessor and not in the host. Another indicative case could be the formation of the policies in software; writing, compiling and executing the policies in a software environment offered by the embedded microprocessor. The EDMU can be much more flexible in this sense; however, the monitoring component loses at the same time the performance advantage that it has in its current version, where all the computations are implemented in hardware. The integration of a microprocessor for enforcing and extending the monitoring functions of the EDMU will provide a hybrid monitoring system in this sense.

REFERENCES

- [1] "Xilinx chips enable world's first "on-the-fly" reconfigurable satellite", Xilinx Press Release #0317, February, 2003.
- [2] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall and G.J.Minden, "A Survey of Active Network Research", IEEE Communications, vol.35, 1997, pp.80-86.
- [3] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee. "Strong security for active networks". In 2001 IEEE Open Architectures and Network Programming, pp. 63-70, Apr. 2001.
- [4] W. La Cholter, P. Narasimhan, D. Sterne, R. Balupari, K. Djahandari, A. Mani, S. Murphy, "IBAN: Intrusion Blocker based on Active Networks", in Proceedings of the DARPA Active Networks Conference and Exposition (DANCE'02), 2002, pp. 182-192.
- [5] O. Prnjat, T. Olukemi, I. Liabotis, L. Sacks, "Integrity and Security of the Application Level Active Networks", IFIP Workshop on IP and ATM Traffic Management WATM'2001 and EUNICE'200, Paris, France, September 2001.
- [6] Z. Liu, R. H. Campbell, and M. D. Mickunas, "Security as Services in Active Networks", in Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02), 2002, pp. 883 - 890.
- [7] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith, "Security in active networks," in Secure Internet Programming: Issues in Distributed and Mobile Object Systems, Jan, Lecture Notes in Computer Science, vol. 1603, Springer-Verlag Inc., New York, NY, USA, 1999.
- [8] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith, "Safety and security of programmable network infrastructures," IEEE Communications Magazine, issue on Programmable Networks, vol. 36, no. 10, October 1998, pp. 84-92.
- [9] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi, "Seraphim: Dynamic interoperable security architecture for active networks," in IEEE OPENARCH 2000, Tel-Aviv, Israel, March 2000.
- [10] Spartan-II FPGA Family Architecture, Xilinx, www.xilinx.com/technology/logic
- [11] Virtex-II Pro FPGA Family Architecture, Xilinx, www.xilinx.com/technology/logic

- [12] Stratix-II FPGA Family Architecture, Altera
www.altera.com/products/devices/stratix2
- [13] A. Ellis, "Achieving Safety in Complex Control Systems", in Proceedings of the Safety-Critical Systems Symposium, Brighton, England, 1995, Springer-Verlag, pp. 2-14.
- [14] N. G. Leveson, "Safeware - System, Safety and Computers", Addison Wesley 1995. ISBN 0-201-11972-2.
- [15] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems", In IEEE software requirements conference, January 1992.
- [16] "IT Performance Engineering & Measurement Strategies: Quantifying Performance Loss." Meta Group, October 2000.
- [17] David J. Smith, "Reliability Maintainability and Risk", Butterworth-Neinemann 2001, ISBN 0-7506-5168-7.
- [18] J.C Lapde, "Dependability: A Unifying Concept for Reliable Computing", in FTCS82, pp. 18-21.
- [19] Giorgio C. Buttazzo. Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers, 1997.
- [20] G. Farber, Prozeßrechentchnik. 2., völlig Neubearb, Aufl. Berlin: Springer, 1992, 223 S., 116 Abb.
- [21] Leveson N. G. Safeware - System, Safety and Computers. Addison Wesley 1995. ISBN 0-201-11972-2.
- [22] Laprie J.C. Dependability: Basic Concepts and Associated Terminology. Dependable Computing and Fault-Tolerant Systems, vol. 5, Springer Verlag, 1992.
- [23] Felix C. Gartner, "Byzantine failures and security: arbitrary is not (always) random", In Proceedings of INFORMATIK 2003, German Computer Science Congress, Frankfurt, Germany, 2003, pp. 127-138.
- [24] Charles E. McDowell and David P. Helmbold. Debugging concurrent programs. ACM Computing Surveys, 21(4):593-621, December 1989.
- [25] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, and T. Lin. Fault injection based automated testing environment. In Proceedings of the 18th Intl. Symposium on Fault-Tolerant Computing (FTCS18), pages 102-107, Tokyo, Japan, 1988.

- [26] Farnam Jahanian, Ragnathan Rajkumar, and Sitaram C.V. Raju. Runtime monitoring of timing constraints in distributed real-time systems. *Real-Time Systems*, 7(3):247–273, November 1994.
- [27] Henrik Thane, Daniel Sundmark, Joel G Huselius, and Anders Petterson. Replay debugging of real-time systems using time machines. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), presented at the First International Workshop on Parallel and Distributed Systems: Testing and Debugging (PADTAD), pages 288–295, Nice, France, April 2003.
- [28] Jeffrey J.P. Tsai, Yaodong Bi, Steve J.H. Hang, and Ross A.W. Smith. *Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis*. Wiley-Interscience, 1996.
- [29] Charles E. McDowell and David P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, 21(4):593–621, December 1989.
- [30] Calvez J.P., and Pasquier O. Performance Monitoring and Assessment of Embedded HW/SW Systems. *Design Automation for Embedded Systems journal*, 3:5-22, Kluwer A.P., 1998.
- [31] F. Muller, "Timing analysis for instruction caches", *Journal of Realtime Systems*, 18 (2000), pp. 217–247.
- [32] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall, and G. Minden : "A Survey of Active Networking Research". *IEEE Communications Magazine*, vol. 35(1), pp. 80-86, January 1997.
- [33] N. F. Maxemchuk, and S. H. Low, "Active Routing", *IEEE journal on selected areas in communications*, VOL. 19(30), pp. 552-565 MARCH 2001.
- [34] S. Bhattacharjee, K.L. Calvert, and E.W. Zegura, "Congestion Control and Caching in CANES," *The International Conference on Communications (ICC '98)*, 1998.
- [35] S. Bhattacharjee, K. L. Calvert, and E. Zegura, "Self-organizing wide-area network caches," *Georgia Institute of Technology GIT-CC-97/31*, 1997.
- [36] M. Sonoda, K. Okamura, K. Araki, "Design of general reliable multicast architecture with active network framework" In Proceedings of the 15th International Conference on Information Networking, pp. 825 - 830, Feb. 2001
- [37] Ralph Keller, Sumi Choi, Dan Decasper, Marcel Dasen, George Fankhauser, and Bernhard Plattner, "An Active Router Architecture for Multicast Video Distribution", In Proceedings of the IEEE The Conference on Computer Communications, INFOCOM 2000, Vol. 3, pp. 1137-1146, March 2000, Tel Aviv, Israel.

- [38] David L. Tennenhouse and David J. Wetherall, "Towards an active network architecture", In ACM SIGCOMM Computer Communication Review, Vol. 26(1), pp. 5 - 17, April 1996.
- [39] S. Walton, A. Hutton, J. Touch, "High-speed data paths in host-based routers", IEEE Computer, Vol. 31(11), November 1998, pp. 46 - 52.
- [40] J.P. Calvez, O. Pasquier, "Performance Monitoring and Assessment of Embedded Hw/Sw Systems", In Design Automation for Embedded Systems Journal, Kluwer Academic Publishers, Vol 3, pp 5-22, 1998.
- [41] Brantley W.C., McAuliffe K.P. and Ngo T.A. "RP3 performance monitoring hardware", In M. Simmons, R. Koskela, and I. Bucher, eds. Instrumentation for Future Parallel Computing Systems, Addison-Wesley, Reading, MA, 1989, pp. 35-45.
- [42] R. Snodgrass, "A Relational Approach to Monitoring Complex Systems", In ACM Transactions on Computer Systems, Vol. 6, No. 2, May 1988, pp 157-195.
- [43] A.C. Liu and R. Parthasarathi, "Hardware monitoring of a multiprocessor systems". In IEEE Micro, October 1989, pp. 44-51.
- [44] Mohammed El Shobaki, On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems, In proceedings of the 8th International Conference on Real-Time Computing Systems and Applications (RTCSA), Tokyo, Japan, March 2002.
- [45] B. Lozzerini, C.A. Prete, and L. Lopriore, "A programmable debugging aid for real-time software development", In IEEE Micro, 6(3):34-42, June 1986.
- [46] B. Plattner, "Real-time execution monitoring", In IEEE Trans. Software Engineering, 10(6), pp. 756-764, Nov., 1984.
- [47] J.P Tsai, K. Fang, H. Chen and Y. Bi, "Noninterference Monitoring and Replay Mechanism for Real-Time Software Testing and Debugging", In IEEE Trans. on Software Eng. vol. 16, pp. 897 - 916, 1999.
- [48] M. Martonosi, D. Ofelt, and M. Heinrich, "Integrating Performance Monitoring and Communication in Parallel Computers", In ACM SIGMETRICS Performance Evaluation Review, Vol. 2, 1996, pp. 138-147.
- [49] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and Debug Strategy of the NX8525 Nexperia Digital Video Platform System Chip. In Proc. Int'l Test Conference (ITC), pp. 121-130. IEEE, 2001.
- [50] ARM. Embedded Trace Macrocell Architecture Specification. www.arm.com.

- [51] IEEE-ISTO 5001TM. The NEXUS 5001 Forum Standard for a Global Embedded Processor Debug Interface. www.nexus5001.org, 2003.
- [52] "ChipScope Pro Software and Cores User Guide"(version 6.3), Xilinx, www.xilinx.com/ise/optional_prod/cspro.htm
- [53] "Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems", Application Note 323, Altera, September 2003, www.altera.com/literature/an/an323.pdf
- [54] "B4655A FPGA Dynamic Probe Data sheet", Agilent Technologies <http://cp.literature.agilent.com/litweb/pdf/5989-0423EN.pdf>
- [55] "TA700 PCI/PCI-X Analyzer/Exerciser", Catalyst, www.getcatalyst.com/product-ta-series.html
- [56] "PCI850 System Analyser/Exerciser", Silicon Control, www.silicon-control.com
- [57] "PI-PCI32E PCI Bus Analyzer", Corelis, www.corelis.com/products/PCI_Analyzers.htm
- [58] "DiaLite Platform Edition", Temento Systems www.temento.com/files/DLI_Platform_Flyer.pdf
- [59] M. El Shobaki, L. Lindh, "A hardware and software monitor for high-level system-on-chip verification", In 2001 International Symposium on Quality Electronic Design, San Jose, CA, USA 2001, pp. 56-61.
- [60] I. Hadzic and J. M. Smith, "P4: A platform for FPGA implementation of protocol boosters", In Proceedings of 7th International Field-Programmable Logic and Applications Workshop (FPL '97), London, UK, Sep. 1997, pp. 438-447.
- [61] I. Hadžić, J.M. Smith and W.S. Marcus, "On-the-fly Programmable Hardware for Networks", In Proceedings of IEEE Globecom, Sydney, Australia, Nov. 1988 Vol. 2. pp. 821-826.
- [62] I. Hadžić, "Applying Reconfigurable Computing to Reconfigurable Networks". Ph.D. thesis, University of Pennsylvania, Department of Electrical Engineering, USA, 1999.
- [63] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in ACM International Symposium on Field Programmable Gate Arrays (FPGA '2000), (Monterey, CA, USA), Feb. 2000, pp. 137-144.

- [64] J. W. Lockwood, "Evolvable Internet hardware platforms," in The Third NASA/DoD Workshop on Evolvable Hardware (EH'2001), Long Beach, CA, July 2001, pp. 271-279.
- [65] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001), Monterey, CA, USA, pp. 87-93, Feb. 2001.
- [66] F. Braun, J. Lockwood, and M. Waldvogel, "Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware", in Proceedings of Hot Interconnects 9 (HotI-9), Stanford, CA, Aug 22-24, 2001, pp. 93-98.
- [67] E. Horta and J. W. Lockwood, "PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs)," Tech. Rep. WUCS-01-13, Washington University in Saint Louis, Department of Computer Science, July 6, 2001.
- [68] E.L. Hortal, J.W. Lockwood, D.E. Taylor and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration", in Proceedings of the 39th conference on Design automation", New Orleans, Louisiana, USA, June 2002, pp. 343 - 348.
- [69] D.E. Taylor, J.S. Turner, J.W. Lockwood, "Dynamic hardware plugins (DHP): exploiting reconfigurable hardware for high-performance programmable routers", in 2001 IEEE Open Architectures and Network Programming Proceedings (OPENARCH 2001), Anchorage, AK, USA, Apr. 1001, pp. 25-34.
- [70] "Two Flows for Partial Reconfiguration: Module Based or Difference Based", Xilinx Application Note, XAPP290 (v1.2) September 9, 2004
- [71] David V. Schuehler, James Moscola, John W. Lockwood, "Architecture for a Hardware-Based, TCP/IP Content-Processing System", IEEE Micro, Vol. 24, No. 1, Jan 2004, pp. 62-69.
- [72] David Schuehler, John Lockwood, "A Modular System for FPGA-based TCP Flow Processing in High-Speed Networks", in the 14th International Conference on Field Programmable Logic and Applications (FPL), Springer LNCS 3203, Antwerp, Belgium, August 2004, pp. 301-310.
- [73] Haoyu Song, Jing Lu, John Lockwood, James Moscola, "Secure Remote Control of Field-programmable Network Devices", in Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, April 20-23, 2004, pp. 334-335

- [74] Haoyu Song and John W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection using FPGA", in proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'05), Monterey, CA, Feb 20-22, 2005, pp 238-245.
- [75] Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B., "A Scalable, High Performance Active Network Node", In IEEE Network, Volume 13, Issue 1, Jan.-Feb. 1999, pp. 8 – 19.
- [76] Decasper, D. and Plattner, B., "DAN - Distributed Code Caching for Active Networks", In Proceedings of INFOCOM'98, San Francisco, USA, Vol. 2, Mar. 1998, pp.609 – 616.
- [77] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., "Router Plugins - A Modular and Extensible Software Framework for Modern High Performance Integrated Services Routers", Washington University Tech Report WUCS-98-08, February 1998.
- [78] B. Metzler, T. Harbaum, R. Wittmann and M. Zitterbart, "AMnet: heterogeneous multicast services based on active networking", in Proceedings IEEE Second Conference on Open Architectures and Network Programming (OPENARCH '99), March 1999, pp. 98 – 105.
- [79] T. Harbaum, D. Meier, M. Zitterbart and D. Brokelmann, "Flexible Hardware Support for Gigabit Routing", Kommunikation in Verteilten Systemen, 1999, pp. 476-487.
- [80] A. Dollas, D. Pneumatikatos, E. Antonidakis, N. Aslanides, S. Kavvadias, E. Sotiriades, K. Papademetriou, S. Zogopoulos, N. Chrysos, K. Harteros, N. Petrakis, "Architecture and Applications of PLATO, a Reconfigurable Active Network Platform", Proceedings, 9th International IEEE Symposium on FPGA's for Custom Computing Machines, Napa Valley, March 2001, pp. 101-110.
- [81] Craig Ulmer, Chris Wood, and Sudhakar Yalamanchili, "Active SANs: Hardware Support for Integrating Computation and Communication", Proceedings of the Workshop on Novel Uses of System Area Networks at HPCA (SAN 2002), Feb. 2002, Cambridge, Massachusetts, USA.
- [82] Weifeng Xu, Ramshankar Ramanarayanan, Russell Tessier, "Adaptive Fault Recovery for Networked Reconfigurable Systems", in proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, April 2003, Napa, California, pp.143-154.
- [83] Francisco J. Gómez, Javier Garrido and Javier Martínez, "Programming Reconfigurable Hardware Through Internet", in proceedings of First Technical Workshop of the Computer Engineering Department - Universidad Autónoma de Madrid, Madrid, March 2000.

- [84] S. Guccione, D. Verkest, and I. Bolsens, "Design Technology for Networked Reconfigurable FPGA Platforms", In Proceedings, Design, Automation and Test in Europe, March 2002, pp. 994–997.
- [85] A.V. Staicu, J. R. Radzikowski, K. Gaj, N. Alexandridis, and T. El-Ghazawi. "Effective Use of Networked Reconfigurable Resources", In Proceedings of the Military Applications of Programmable Logic Devices, Laurel, MD, USA Sept. 2001.
- [86] H. Fallside and M. Smith, "Internet Connected FPL", In Proceedings of the International Conference on Field Programmable Logic and Applications, Villach, Austria, September 2000, pp. 48–57.
- [87] A.G. Fragkiadakis, N.G. Bartzoudis, D.J. Parish and M.J. Sandford, "Hardware Support for Active Networking", In Proceedings of the International Conference SAM'03: Security And Management, Las Vegas, U.S.A., June 2003, vol. 1, pp. 27-33.
- [88] "Architecting Systems for Upgradability with IRL (Internet Reconfigurable Logic)", Xilinx Application Note, XAPP412 (v1.0) June 29, 2001.
- [89] "Remote System Configuration with Stratix & Stratix GX Devices", Altera Corporation, Stratix GX Device Handbook, Volume 3, September 2004.
- [90] "Implementing Secure Remote Updates using Triscend E5 Configurable System-on-Chip Devices", Triscend Corporation, Application Note (AN-02) v1.00, January, 2000.
- [91] S. Mitra, and E.J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?", in Proceedings of International Test Conference, Atlantic City, NJ, Oct. 2000, pp. 985-994.
- [92] N.R. Saxena, S. Fernandez-Gomez, W.J. Huang, S. Mitra, S.-Y. Yu and E.J. McCluskey, "Dependable Computing and Online Testing in Adaptive and Configurable Systems", in IEEE Design and Test of Computers, Vol. 17, No. 1, Jan.-Mar. 2000, pp. 29-41.
- [93] E. Bohl, Th. Lindenkreuz, and R. Stephan, "The fail-stop controller AE11", in Proceedings of the IEEE International Test Conference 1997(ICT '97), Nov. 1997, Crete, Greece, pp. 567 – 577.
- [94] Tilman Wolf, "A Proposal for a High-Performance Active Hardware Architecture", Technical report, Department of Computer Science, Washington University, 1999.
- [95] Tilman Wolf and Mark A. Franklin, "Design tradeoffs for embedded network processors," in Proc. of International Conference on Architecture of Computing

- Systems (ARCS) (Lecture Notes in Computer Science), vol. 2299, pp. 149–164, Springer Verlag Karlsruhe, Germany, Apr. 2002.
- [96] ADPCM, 1024 Channel Simplex
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/Amphion-ADPCM_1024.pdf
- [97] eDCT e-Discrete Cosine Transform
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/einfo_dct.pdf
- [98] JPEG, 2000 Encoder
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/Barco_JPEG2KE.pdf
- [99] MPEG-2 HDTV I & P Encoder
http://www.xilinx.com/products/logicore/alliance/duma_video/duma_hdtv.pdf
- [100] MPEG-4 Video Compression Encoder
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/4i2i_MPEG-4_Encoder.pdf
- [101] MPEG-4 Video Compression Decoder
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/4i2i_MPEG-4_Decoder.pdf
- [102] Triple DES Encryption Core
http://www.xilinx.com/products/logicore/alliance/cast/cast_des3.pdf
- [103] AES Fast Encryption
http://www.xilinx.com/bvdocs/ipcenter/data_sheet/Helion-AES_Fast_Encryption.pdf
- [104] K. G. Coffman and A. M. Odlyzko, "Internet growth: Is there a "Moore's Law" for data traffic?", AT&T Labs – Research, June 2001.
- [105] S. Choi, D. Decasper, J. Dehart, R. Keller, J. Lockwood, J. Turner, T. Wolf, "Design of a flexible open platform for high performance active networks", In Proceedings of the 37th Allerton Conference on Communication, Control, Computing, 1999, pp. 157-165.
- [106] Melvin A. Breuer, "Intelligible Test Techniques to Support Error-Tolerance", in Proceedings of the 13th Asian Test Symposium (ATS'04), November 2004, pp. 386-393.
- [107] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and using a highly parallel programmable logic array", IEEE Computer, vol. 24, no. 1, Jan. 1991, pp. 81–89.
- [108] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2", in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1992, pp. 316–324.

- [109] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable active memories: Reconfigurable systems come of age", *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 56–69, 1996.
- [110] R. Amerson, R. Carter, B. Culbertson, P. Kuekes, and G. Snider, "Teramac-configurable custom computing", in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, Eds., Napa, CA, Apr. 1995, pp. 32–38.
- [111] L. Moll and M. Shand, "Systems performance measurement on PCI pamette", in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, J. Arnold and K. L. Pocek, Eds., Napa, CA, Apr. 1997, pp. 125–133.
- [112] B. K. Fross, R. L. Donaldson, and D. J. Palmer, "Pci-based WILDFIRE reconfigurable computing engines", in *Proceedings of SPIE—The International Society for Optical Engineering*, Bellingham, WA, November 1996, SPIE, vol. 2914, pp. 170–179, SPIE.
- [113] B. Schott, S. Crago, C. Chen, J. Czarnaski, M. French, I. Hom, T. Tho, and T. Valenti, "Reconfigurable architectures for systems level applications of adaptive computing", *VLSI Design*, vol. 10, no. 3, pp. 265–279, 2000.
- [114] "EPXA10 DDR Development Kit", *Hardware Reference Manual*, April 2003 Version 1.3, Altera Corporation.
(www.altera.com/literature/manual/mnl_epxa10_ddr_devbd.pdf)
- [115] PROCStar II PCI board, Gidel, www.gidel.com (07/2005).
- [116] DN5000k10S PCI board, DN6000k10 PCI board, The Dini Group, www.dinigroup.com (07/2005).
- [117] ADM-XPL PMC board, ADP-XPI PCI board, Alpha Data PLC, www.alpha-data.com (07/2005).
- [118] "PCI Local Bus Specification", Revision 2.2, PCI Special Interest Group (PCISIG), December 18, 1998.
- [119] Tom Shanley and Don Anderson, "PCI system architecture", Mindshare Inc. 1999, Forth edition, ISBN 0201309742.
- [120] PCI 64-bit/(66-33 MHz) and PCI 32-bit/(66-33 MHz) Core, Xilinx LogicCore 3.0
www.xilinx.com/pci
- [121] PCItree version 2.04a, www.pcitree.de

- [122] "pci_mt32 MegaCore Function Reference Design", Altera Application Note, Altera, September 2002. (www.altera.com/literature/fs/fs_fs12-pci_mt32.pdf)
- [123] "PCI-to-DDR SDRAM Reference Design", Application Note 223, Altera, May 2003, version 1.0, www.altera.com/literature/an/an223.pdf.
- [124] J.L. Núñez, C. Feregrino, S. Jones, S. Bateman, "X-MatchPRO: A ProASIC-Based 200 Mbytes/s Full-Duplex Lossless Data Compressor", In Proceedings of FPL 2001, Lecture Notes in Computer Science, Springer, August 2001, pp. 613-617.
- [125] PCI LogiCore v3.0, Getting Started Guide (UG157), Xilinx, November 11, 2004. www.xilinx.com/pci
- [126] Geir Drange, "Random Number Generator Library", Version 1.0, Sep. 2004. www.opencores.org/projects.cgi/web/rng_lib/overview
- [127] PCI64 Virtex and Spartan Series Interface, Xilinx IP Core www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=DO-DI-PCI64-IP
- [128] Alpha data ADM-XPL board, (www.alpha-data.com/adm-xpl.html)
- [129] ADM-XRC-PRO-Lite (ADM-XPL) Hardware Manual, Version 1.7, Alpha Data www.alpha-data.com/pdf/adm-xpl%20hardware%20reference%20manual.pdf
- [130] ADM-XRC SDK version 4.5.2 User Guide, Alpha Data, www.alpha-data.com/pdf/admxrcsdk.chm

APPENDIX I

Active packets have to be distinguished from passive packets. It is also important to have a system that is compatible with the existing IP networks infrastructure. Hence, in order to satisfy both of the aforementioned conditions, an active header encapsulating an IP packet was defined. Related work in Active Packet protocols is presented in [1] and [2]. The active header carries information as it is shown in Table 1.

The Active Header	
Name	Size (bits)
type	8
seq_no	8
options	16
id	32
last node	32

Table 1. The Active Header

Passive IP packets are encapsulated in UDP packets (figure 1). ACT is the active header.

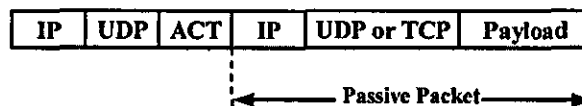


Figure 1: The Active Header.

The active packets are distinguished from the passive packets using the destination UDP port 44075 (active port) of the first transport header. The second transport header (UDP or TCP) is application-specific. The type field specifies the type of the active packet. The current active types that have been defined are:

- type 0: UDP active packets (not encapsulated packets),
- type 1: TCP active packets (wrapped TCP packets),
- type 2: UDP active packets (wrapped UDP packets),
- type 3: UDP active packets (administrative packets, not wrapped).

The sequence number (seq_no) is used to help the introduction of reliability schemes where measurement or reliability is required. The options field can be used to specify options for coding modules. The id field specifies the unique code module

number. This refers to the code module of a particular packet which should be handled. The last active node (`last_node`) field specifies the last active node that the packet passed through.

The Active Packets may have one of the following operation modes:

1. In the first scenario the Active Packet has a pointer to a certain application (IP core). The requested application is stored either locally in the Active Router or in a different location (e.g. a Code Server [3]). The Active Router detects and processes the Active Packet so as to serve the request. The requested application is then configured in the hardware platform. The Active Router forwards the user's data to the PCI card, so as to be processed by the Active Application.
2. In the second approach, several Active Packets carry a configuration bitstream and data in their payloads. The Active Router detects and processes the Active Packets so as to serve the request. This processing stage includes authentication of the user, and integrity checks. The application is then configured in the target FPGA board and in the following stage the host transfers the user data.

An experiment has already taken place in the Active Network of Loughborough University. The experiment described in the following lines is based on the topology shown in figure 3. The Xilinx PCI board, which is used in the current experiment [4], simplifies the process of reconfiguring the FPGA multiple times due to the build-in functions of the Linux software API. Rapid implementation of reconfigurable computing applications is made feasible in a time effective way using this PCI board.

Three applications were used for configuring the Active Router (AR) on-the-fly. These applications are:

1. A software-based application that computes the first complement of the data that is contained in the active packets' payload. It is a simple application (just for demonstration purposes) written in C. It has been assigned the module id=1.
2. A hardware-based application that nibble-reverses the data of the active packet's payload. This is done in the FPGA and the nibble-reversed values are passed back to the appropriate active module. This application has been assigned the module id=2.

3. A hardware-based DES (Data Encryption Standard) encryption/decryption algorithm. The ECB (Electronic Codebook) mode is used. The DES IP core was provided by the Free-IP Project [5]. This application has been assigned the module id=3.

The active module that runs in the host splits the packets' payload into 32-bit words and passes them to the FPGA board via the PCI bus. The active module acts as a master and the FPGA as a slave. Each time, the active module performs five "PCI writes" and two "PCI reads". A 32-bit PCI bus is used. The block diagram of the DES algorithm is shown in the figure 2 below. The "PCI reads and writes" are:

- 1st PCI-write: a 32-bit word (from the packet's payload) is passed to the FPGA,
- 2nd PCI-write: a second 32-bit word is passed to the FPGA (these are the 64-bit length input data, "data in" in figure 2),
- 3rd PCI-write: the key for the encryption or decryption is 64 bit long so it is split into two 32-bit words. The 3rd PCI-write passes the last 32 bits of the key,
- 4th PCI-write: the first 32 bits of the key are passed to the FPGA,
- 5th PCI-write: a 32-bit word is passed to the FPGA; this word has the value 0 or 1. If it is 0, the FPGA will decrypt the input data and if it is 1 it will encrypt them ("encrypt" in figure 2).

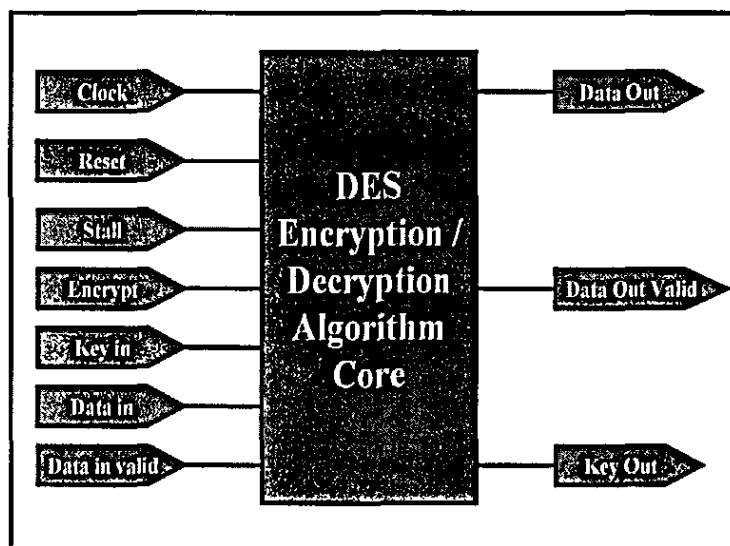


Figure 2: The DES encryption/ decryption IP core.

After all these data are passed to the FPGA (they are stored in registers), two PCI-reads are needed to take the encrypted or decrypted data back (32-bit data in each PCI-read). This procedure is repeated until all the data of the packet's payload have been encrypted (or decrypted). The module id=3 has been assigned to the DES encryption/decryption application.

The Active Router is part of the experimental network shown in figure 9. The Active Router consists of two separate hosts: The Router-Active Forwarding Engine and the Active Element. The Router-Active Forwarding Engine consists of two modules:

- A kernel-space loadable module that forwards the incoming active packets to the AE.
- A user-space process that monitors the AE to check if it is "alive" (by sending ICMP requests). If it is not "alive" it removes the kernel-space module and hence no more packets are forwarded to the Active Element. It reloads it as soon as the Active Element is available again.

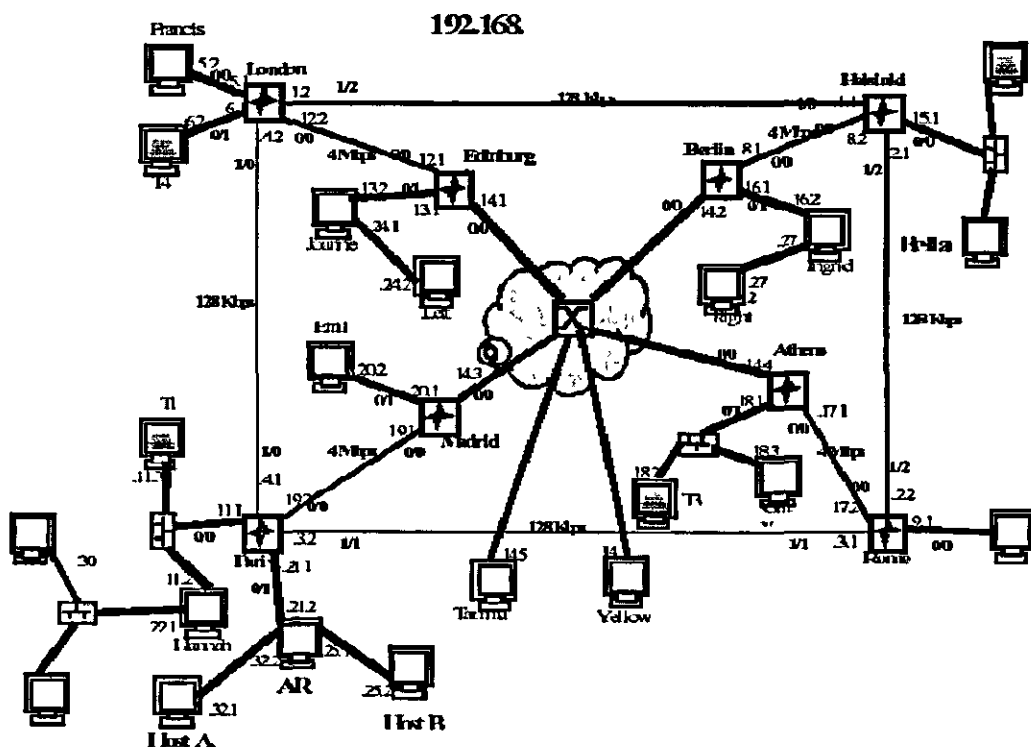


Figure 3: An Active Network with a reconfigurable processing engine.

An application that is running on host A generates UDP packets having as destination host B. These packets are wrapped in host A into active packets of type 2 by an Active Daemon. The module unique identification is 2.

At default, the active application with module id=1 is running in the Active Router (AR). This is an active application that changes the data of the packets' payload to their first complement. When an active packet of type 2 (and module id=2) is transmitted from host A to host B, it will be routed by the AR. Prior to routing, the AR verifies that the active packet has module id=2 and checks which module is currently configured in the reconfigurable platform. In the tested case the configured application has module id=1. At that point the AR has to make a decision: either to stop application 1 and run application 2 or to continue running the current application and just route unmodified the active packet to its destination. The AR has to check the available resources in the target platform. If there are enough, it stops application 1 and loads application 2, configuring the FPGA board with the appropriate bitstream. The AR downloads the bitstream from the code server and invokes the active module no 2. This active module configures the FPGA on the fly with the appropriate bitstream and passes all consequent active packets (with module id=2) to and from the FPGA. The application that 'runs' in the FPGA nibble-reverses the payload data of the packets.

Next, Host A sends an active packet with module id=3. The application with module id=3 is an FPGA-based DES (Data Encryption Standard) encryption algorithm. The AR follows the same procedure as above (resources check, authorised packet etc) and then it stops the application that runs in the FPGA (no 2) and loads application no 3 (DES encryption). Hence, the AR is able to switch from one application to another (1,2,3) and configure the FPGA device on-the-fly.

References

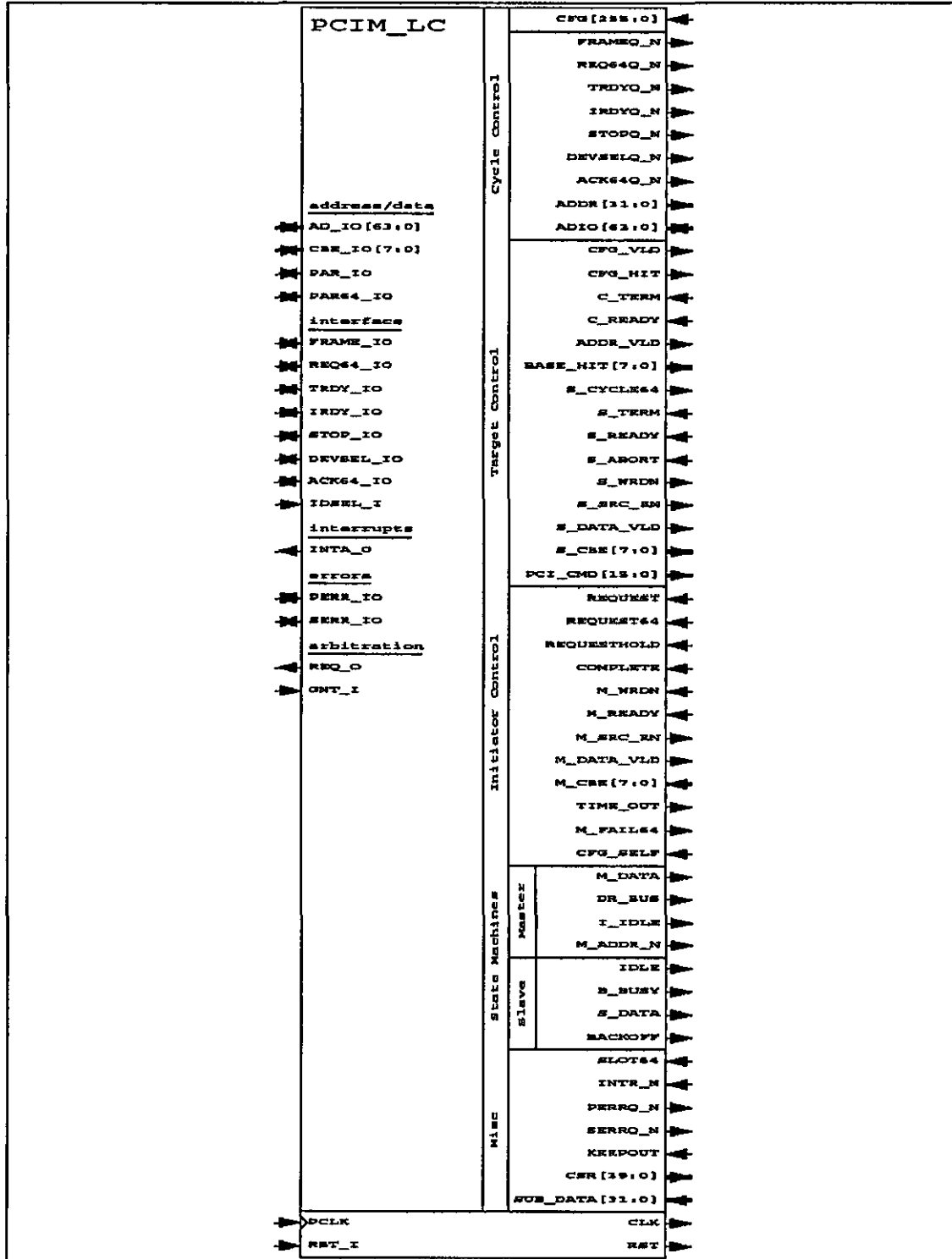
- [1] D.J. Wetherall and D.L. Tennehouse, "The Active IP Option", In Proceedings of the 7th ACM SIGOPS European Workshop, Connemara, Ireland, Sept. 1996.
- [2] D.S. Alexander, B. Graden, C.A. Gunter, A.W. Jackson, A.D. Keromytis, G.J. Minden and D. Wetherall, "Active Network Encapsulation Protocol (ANEP)", July 1997.
(<http://www.cis.upenn.edu/~switchware/ANEP>)

[3] S. Choi, D. Decasper, J. Dehart, R. Keller, J. Lockwood, J. Turner, T. Wolf, "Design of a flexible open platform for high performance active networks", Proc. Of the 37th Allerton Conference on Communication, Control, Computing, pp. 157-165, 1999.

[4] Alpha data ADM-XRC board, (www.alpha-data.com/adm-xrc.html)

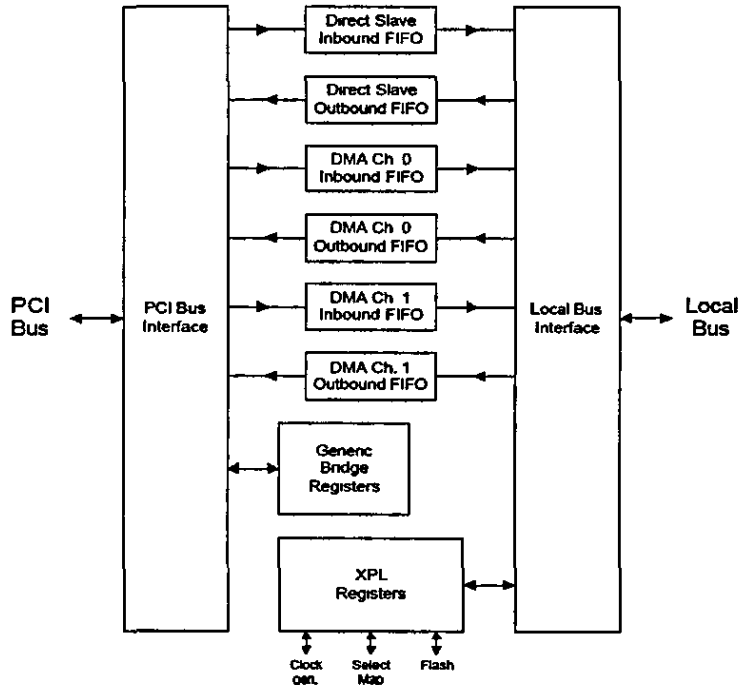
[5] OpenCores (<http://www.opencores.org/projects>)

APPENDIX II



The Signals of the PCI64 Xilinx LogiCore.

APPENDIX III



Block diagram of ADM-XPL.

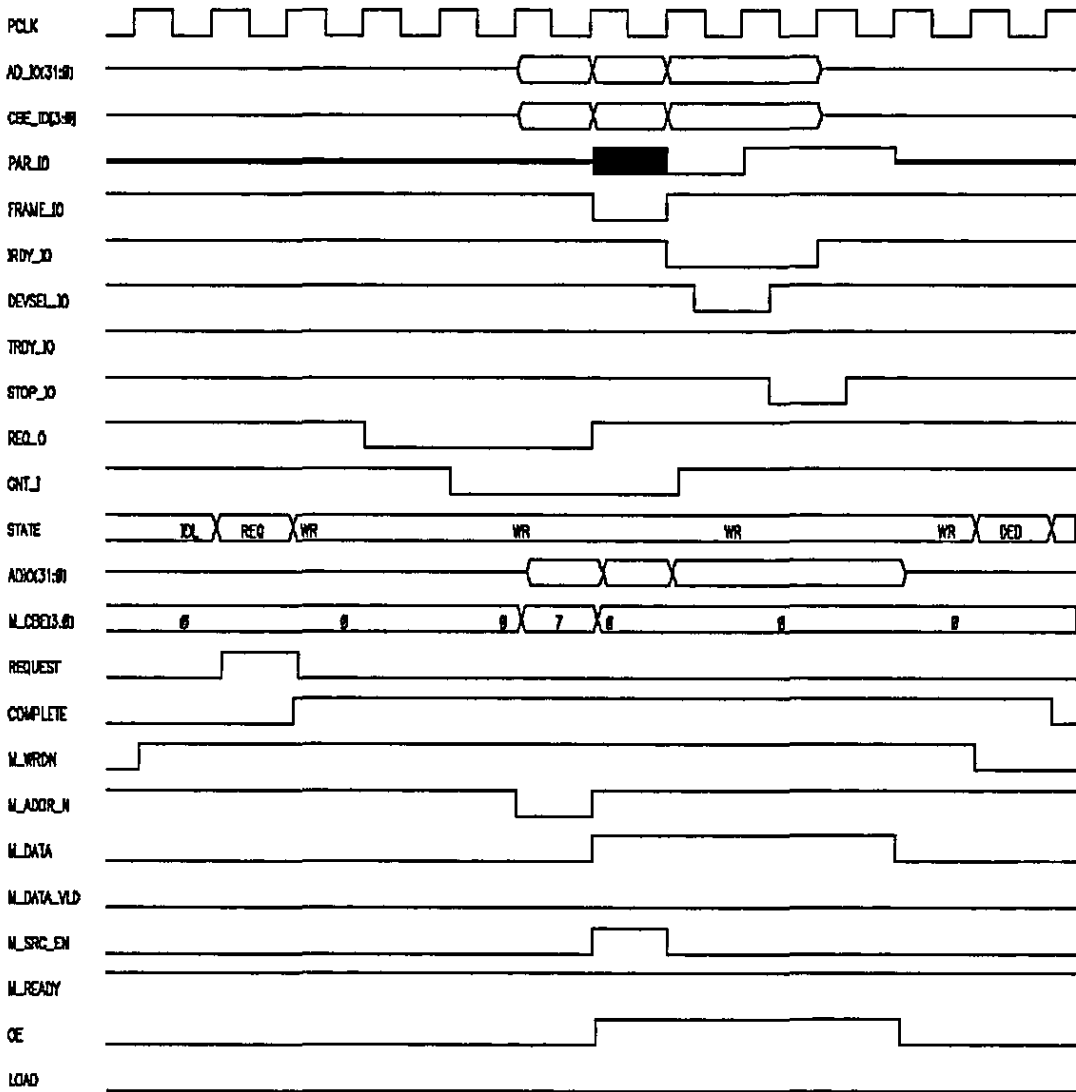
MSb	Bit	Bit	Bit	LSb	Offset
31	24 23	16 15	8 7	0	
Device ID (0x0043)		Vendor ID (0x4144)			0x00
Status Register		Command Register			0x04
Class Code (0x0b4000)			Revision ID (0x02)		0x08
BIST	Header Type (0x00)	Latency Timer	Cache Line Size		0x0C
Base Address Register 0 (BAR0) Generic bridge registers in memory space					0x10
Base Address Register 1 (BAR1) Generic bridge registers in I/O space					0x14
Base Address Register 2 (BAR2) Local bus window 0					0x18
Base Address Register 3 (BAR3) ADM-XPL registers					0x1C
Base Address Register 4 (BAR4) - Reserved					0x20
Base Address Register 5 (BAR5) - Reserved					0x24
Cardbus CIS Pointer - Reserved					0x28
Subsystem Device ID		Subsystem Vendor ID (0x4144)			0x2C
Base Address of Expansion ROM - Reserved					0x30
Reserved			Next Cap Pointer (0x40)		0x34
Reserved					0x38
Max_Lat (0x00)	Min_Gnt (0x00)	Interrupt Pin (0x01)	Interrupt Line		0x3C

The ADM-XPL PCI configuration space header.

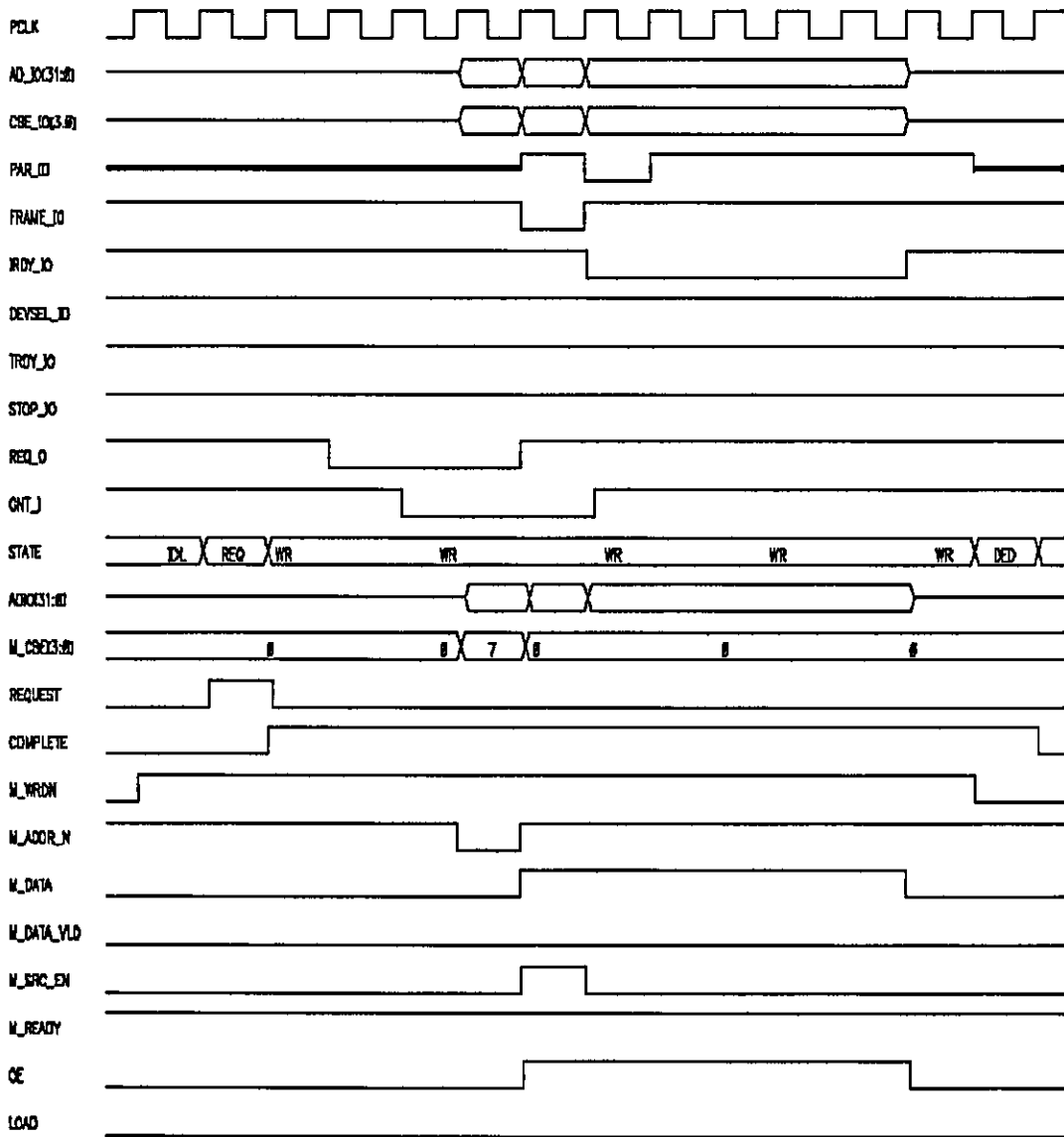
Offset	Name	Function
0x00	LS0B	Local space 0 base address register
0x04	LS0R	Local space 0 range register
0x08	LS0CFG	Local space 0 configuration register
0x0C		<i>Reserved</i>
0x10	LS1B	Local space 1 base address register
0x14	LS1R	Local space 1 range register
0x18	LS1CFG	Local space 1 configuration register
0x1C		<i>Reserved</i>
0x20	LTO	Local bus timeout register
0x24		<i>Reserved</i>
0x28	LCTL	Local bus control register
0x2C to 0x4C		<i>Reserved</i>
0x50	PICTL	PCI interrupt control register
0x54	PISTAT	PCI interrupt status register
0x58 to 0x7C		<i>Reserved</i>
0x80	D0PAL	DMA channel 0 PCI memory address register
0x84		<i>Reserved</i>
0x88	D0LA	DMA channel 0 local address register
0x8C	D0SIZ	DMA channel 0 transfer size and direction register
0x90	D0NDL	DMA channel 0 next descriptor address register
0x94		<i>Reserved</i>
0x98	D0CFG	DMA channel 0 configuration register
0x9C	D0CTL	DMA channel 0 control register
0xA0	D1PAL	DMA channel 1 PCI memory address register
0xA4		<i>Reserved</i>
0xA8	D1LA	DMA channel 1 local address register
0xAC	D1SIZ	DMA channel 1 transfer size and direction register
0xB0	D1NDL	DMA channel 1 next descriptor address register
0xB4		<i>Reserved</i>
0xB8	D1CFG	DMA channel 1 configuration register
0xBC	D1CTL	DMA channel 1 control register
0xC0 to 0xFC		<i>Reserved</i>

Generic bridge registers.

APPENDIX IV



This timing waveform demonstrates the behaviour of the PCI interface when a target signals an abort (target abort). This event often occurs due to incorrect programming or serious system errors. It can also signify that the initiator has incorrectly attempted to burst data beyond the address space of the target. The user application must respond appropriately.



This timing waveform demonstrates the behaviour of the PCI interface when no target responds (master abort). This event is expected during some configuration transactions and special cycle broadcast cycles. However, during normal operation, this would occur due to incorrect programming or serious system errors. It is critical that the user application respond appropriately.

