Design and Implementation of Reconfigurable VMEbus Exerciser

Makhamisa Senekane

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of Master of Science in Engineering.

Cape Town, December 2010

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town

3 December 2010

Abstract

The VMEbus[1] is an industry-standard high-performance bus that supports 8-, 16- and 32-bit data transfers over a non-multiplexed non-multiplexed 32-bit data bus. Additionally, VMEbus supports 16-, 24- and 32-bit addressing over a separate 32-bit address bus. The main features of VMEbus are:

- A high-speed asynchronous data bus to transfer 8-, 16- or 32-bits at a time;
- Four buses/subbuses: data transfer, arbitration, priority interrupt and utilities buses;
- Supports several bus controllers, such as central processing unit (CPU), direct memory access (DMA), input/output (I/O), and any other device that needs to control the bus. The arbitration bus avoids conflicts;
- Priority-interrupt buses where devices can request services from a VME interrupt handler;
- A utililies bus which provides power distribution, clocks, initialization and failure detection.

The purpose of this dissertation is to describe a project intended for design and implementation of a reconfigurable VME exerciser[2] at iThemba Laboratory for Accelerator-Based Science (iThemba LABS)[3]. This is realized by providing a design and implementation of VME printed circuit board (PCB) as the main board, hardware description language (HDL) for VME daughter-card and finally, the graphical user interface (GUI) is designed and implemented as well. Next, these designs are tested to verify that they conform to the specified user requirements. From the results, this project project proves to meet the user requirements, hence it can be said that it was successful. In addition to meeting user requirements, data rates of up to 26 Megabytes per second (MBPS) were realized.

[1] VME is an acronym for Versa Module Eurocard. Note that VMEbus is used interchangeably with VME

[2] VME exerciser is a VME module that is used to test functionality of VME slave Boards

[3] iThemba LABS is a group of multi-disciplinary laboratories administered by the National Research Foundation (NRF) based in Gauteng and Western Cape Provinces. This project was carried out at the Western Cape site.

Dedication

This is dedicated to my family, my friends and all those people who work hard to make this world a better place.

Acknowledgements

I would like to thank the following parties for their contributions towards successful completion of this research project:

- Professor Michael Inggs, my supervisor, for his guidance and encouragement during the entire research project
- National Research Fund(NRF) through iThemba LABS, for financial assistance
- Dr. Petyu Petev, my co-supervisor at iThemba LABS, for his support and guidance
- Mr. Lebelo Serutla, for his guidance on how to prepare this dissertation
- Nick Thorne and Jason Salkinder at CHPC, for their help on soft-core implementation and PCB design respectively
- Norlan Klaasens and Pam Jones at iThemba LABS, for their untiring technical support
- Sehlabaka Qhobosheane and Thabo Ramosie, for their profound advice throughout the entire research project, and for proof-reading this dissertation
- my family and friends, for always being there for me in times of need
- RRSG group members at UCT, for their contribution in this research

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ADO	—	Address Only cycle
ALU	—	Arithmetic Logic Unit
CPU	—	Central Processing Unit
DMA	—	Direct Memory Access
DXP	—	Design eXPlorer
FPGA	—	Field Programmable Gate Array
FR-4	—	Flame Retardant with dielectric loss of 4
GUI	—	Graphical User Interface
HDL	—	Hardware Description Language
I/O	—	Input/Output
ISE	—	Integrated Services Environment
LED	—	Light Emitting Diode
MBPS	—	Mega-Bytes Per Second
PC	—	Personal Computer
PCB	—	Printed Circuit Board
PROM	—	Programmable Read-Only Memory
RISC	—	Reduced instruction Set Computer
RWD	—	Release When Done
SCT	—	Single Cycle Transfer
SGL	—	SinGle Level
TTL	—	Transistor-Transistor Logic
UART	—	Universal Asynchronous Transmitter Receiver
UCF	—	User Constraint File
VME	—	Versa Module Eurocard
VHDL	—	VHSIC Hardware Description Language
VHSIC	—	Very High Speed Integrated Circuit

Nomenclature

Daughter-card — A circuit board that is used to extend functionality of PCB main boards.

Exerciser — VMEbus exerciser is a service module that is used to generate test commands on the bus.

Field Programmable Gate Array (FPGA) — An integrated circuit that is designed to provide flexibility by allowing the user to reconfigure it for desired application.

Flame Retardant with a Dielectric Loss of 4 (FR-4) — A woven glass fabric that is primarily used for PCB fabrication.

Gerber File — A standard electronics output file used to communicate design information to PCB manufacturing machines.

Graphical User Interface — A computer program that is used to communicate with other computer programs through the use of symbols, visual metaphors and pointing devices.

Hardware Description Language — A programming language that is used to model digital logic.

Interrupt Handler — A VME functional module that is used by VME controller to manage interrupts from other VME modules.

Isotopes — Atoms of the same element which differ in the number of neutrons they contain.

Microblaze —A 32-bit Harvard architecture soft processor core designed for Xilinx FPGAs, from Xilinx.

Picoblaze — An 8-bit Harvard architecture soft core processor with separate data bus and instruction port. It is designed by Xilinx and is available free of charge.

Printed Circuit Board (PCB) — A printed circuit board is an electrical component that is used to mechanically support and electrically connect electronic components using conductive pathways, tracks or traces, etched from copper sheets laminated on a non-conductive substrate.

PyQt — A set of Python bindings for Qt C++ library. This is a product of Riverbank Computing Limited.

Soft Core Processor — An Intellectual Property(IP) core that is implemented using logic primitives of the FPGA.

Universal Asynchronous Transmitter Receiver (UART) — A module/macro that is used to translate data between parallel and serial forms, usually for serial communications.

Versa Module Eurocard (VME) — A computing systems architecture that consists of electrical specifications for data bus and mechanical specifictions for describing the backplane, bus connector board sizes and enclosers.

VHSIC Hardware Description Language (VHDL) — A hardware description language that is commonly used to write text models that describe a digital logic circuit.

Chapter 1

Introduction

This chapter provides an introduction to the project. It starts off with a problem statement. This is followed by background information of technology used in this project. Following that section is an overview of this dissertation.

The purpose of this project was to design and implement a reconfigurable VME exerciser for application at iThemba Laboratory for Accelerator-Based Sciences (LABS). iThemba LABS is a high-energy particle physics centre which extensively implements VME boards for data acquisition purposes, hence a need to use a VMEbus exerciser to stimulate such boards.

iThemba LABS is an institution for accelerator-based sciences. It is aimed at providing research into particle-based sciences, for use in particle radiotherapy treatment of cancer , research in the areas of particle beams, and production of radio-isotopes for use in radio-medicine and research [42]. It has two sites; one in Gauteng Province, and another one in the Western Cape Province. iThemba LABS facility in the Western Cape[1] is shown in Figure 1.1.

Figure 1.1: This Figure shows iThemba LABS site in Cape Town, Western Cape [24].

[1] where this project was carried out

1.1 Problem Statement

The purpose of this project was to design and implement a modern, reconfigurable[2] single-cycle VME exerciser/tester that emulates a VMEbus master. This means that the exerciser should be able to exercise VMEbus using standard VME cycles (read and write) and/or address broadcasting [address-only (ADO)] cycle.

1.2 Project motivation and Background Information

1.2.1 Project Motivation

iThemba LABS is a high-energy research facility which requires high-performance[3] data acquisition systems. Since its introduction, VMEbus[4] has been known to play a leading role in embedded computing, real-time control and data acquisition systems for high-energy physics [5]. This standard is widely used because it is open-ended and it out-performs general-purpose processor architectures in data acquisition applications.

In order to ensure optimal operability of VMEbus systems, VMEbus exerciser modules are used. Further details on VMEbus exerciser are given in Subsubsection 1.2.2.2 and [7].

The previous module of VMEbus exerciser (at iThemba LABS) was a manually controlled module in standard VME based on low level of integration components[5]. This, unfortunately, posed the following limitations:

- lack of user-friendliness (the setting-up of commands is carried out manually by control swiches), and
- lack of program memory storage (the commands were entered one after the other, manually, and that resulted in slow systems).

Based on these limitations, it was decided that a new, low-cost exerciser should be constructed, which would address these limitations.

The front panel of the previous exerciser at iThemba LABS is shown in Figure 1.2.

[2] using Field Programmable Gate Array(FPGA)
[3] high data rates
[4] further information on VMEbus is provided in Appendix A and the references provided
[5] microcontroller-based and discrete logic

Figure 1.2: Front panel of the previous VMEbus tester, with all control switches. The switches were used to carry out VMEbus operations.

1.2.2 Background Information

1.2.2.1 VMEbus Evolution

In the early years of Versa Module Eurocard (VME) design, interfaces typically required a substantial number of discrete transistor-transistor logic (TTL) devices and printed circuit board (PCB) space to decode and generate necessary timing signals for proper VME communication [4]. Such interfaces' lack of flexibility (scalability) was their main drawback. With the introduction of field programmable gate arrays (FPGAs), VMEbus systems designs began to take on a more generic appearance. Their (FPGAs') capability to be re-programmed allows for logic corrections to be made to the board designs even after board fabrication [4]. This results in quick turn-around times and cheap designs.

1.2.2.2 VMEbus Exerciser Brief Description

VMEbus exerciser is a module in a bus architecture which is used to generate extensive test commands on the bus. It can be used to exercise arbitration bus, priotity interrupt bus, and/or utility bus, either within the constraints of VMEbus specification, or to generate spurious signals [7]. It is useful for testing existing (as well as new) boards, and ensuring that interrupt handlers operate properly.

Basically, VMEbus exerciser is a service module that emulates a master module on the bus. It achieves this by initiating commands on the bus appropriate for slave modules to test their functionality. Exerciser functions include reading, writing, address broadcast and memory comparison.

1.3 Dissertation Overview

The rest of the dissertation is arranged as follows:

Chapter 2: this chapter provides detailed explanation of VME exerciser design process. It is divided into five sections, namely; user requirements and constraints section, requirements section analysis section, section detailing exerciser functions, design specifications section and acceptance test plan section. The first section mentions user requirements and design constraints. User requirements include functional requirements (e.g, ADO cycles) and non-functional requirements (e.g, design of a low-cost exerciser). This is then followed by a section on requirements analysis. This section translates user requirements into realizable engineering goals. The next section is on the exerciser functions. These functions are divided into both hardware and software functions. Hardware functions include creating a VME main board which houses VME interface logic, and a daughter-card which is used to carry out VME operations. The next section outlines design specifications, based on user requirements and constraints, requirements analysis and function outlined in the previous sections. The last section of this chapter describes a test plan to be employed in order to test the

success of the project. The test plan is divided into four categories; namely mechanical, functional, graphical user interface (GUI) and performance test processes.

Chapter 3: This chapter provides details on the design of this VMEbus exerciser. The design is carried out in concordance with design specifications outlined in Chapter 2. This chapter is divided into four sections, namely; design tradeoffs section, hardware design section, section on firmware design and GUI design section. The first section deals with design tradeoffs in both hardware, firmware and GUI designs. This is followed by a section which covers hardware (main board) design. Following this section is a section on firmware design. The section gives description of the design of soft processor cores which perform the required VMEbus functions, in order to meet the user requirements. The last section of this chapter provides GUI design considerations, together with GUI design.

Chapter 4: The chapter gives details on the procedure followed to implement this design. These details are given in three sections, with the first section detailing hardware implementation (it covers issues such as printed circuit board's (PCB's) schematic design, board layout, manufacturing and assembly), followed by a section which provides details on firmware implementation (it covers issues such as ADO, read and write cores' implementation and universal asynchronous transmitter/receiver (UART) macros). The final section provides information on the implementation of the graphical user interface (it gives details for implementation of GUI, covering issues such as PyQt4 toolkit, Pyserial module and sending/reading VMEbus commands to/from the serial port).

Chapter 5: The chapter presents the results of varied tests that were carried out on the exerciser. These tests were based on the acceptance test plan outlined in Chapter 2. The tests are divided into four categories, namely; mechanical, functional, GUI and performance tests. Mechanical tests were carried out to test mechanical characteristics of exerciser PCB against VMEbus specification. Functional tests were carried out to determine exerciser system's capability to meet user's functional requirements. GUI tests were performed to determine user interface's functionality and usability. Performance tests were run to determine the extend to which a function was performed (by the exerciser), against the given benchmarks. Furthermore, these results are analysed in this chapter, so that conclusions could be drawn, in the chapter that follows.

Chapter 6: Based on the results obtained in Chapter 5, Chapter 6 represents conclusions drawn on whether the project was successful or not. From this project, the following conclusions could be drawn:

- a high-performance (data rates of up to 26 MBPS), robust, low-cost VMEbus exerciser was developed
- this exerciser is capable of performing ADO, write and read operations
- GUI was built, and this GUI would be used to run VMEbus from the computer to the daughter-card

- system tests proved the project to be a success (meaning that the goals of the project were realized)
- there was a need to upgrade from Xilinx integrated services environment (ISE) 9.2 to ISE 11.1, especially for performing read transactions

Also, recommendations for future work are posited in this chapter. These recommendations are given as thus:

- use of Xilinx ISE 11.1 or later versions for all VME operations
- introducing arbitration macro in VME firmware, so as to reduce overhead between VME cores and the input/output (I/O) circuitry
- use of Python design patterns to enhance GUI performance and usability
- implementation of endianness conversion to improve GUI's user-friendliness
- creation of a log file to store all executed VME commands on the disk

Appendices: There are four Appendices. Appendix A presents a thorough account of the VMEbus specification. Appendix B provides GUI top-level source code. Appendix C provides design errata which should be incorporated in the subsequent versions of this exerciser module. Appendix D is a compact-disc (CD) which contains resources which were used towards ensuring success of this project.

1.4 Summary

This chapter provided an introduction to this project. The problem statement was stated, followed by technology background information of the project. The next chapter discusses in detail the design process that was followed to carry out this project.

Chapter 2

Exerciser Design Process

This chapter provides a detailed explanation of VMEbus exerciser design process. In the chapter, an outline of user requirements and constraints is mentioned, followed by user requirements analysis. This is then followed by description of exerciser board functions, which is followed by design specifications. Finally, acceptance test plan is outlined.

2.1 User Requirements and Constraints

2.1.1 User Requirements

The user requirements are divided into both functional and non-funtional requirements. Functional requirements are:

- ADO VME cycles
- write VME cycles
- read VME cycles

non-functional requirements include:

- design of a low-cost exerciser
- the module should yield high performance
- exerciser board should be portable
- the exerciser should use a user-friendly graphical user interface (GUI), instead of using control switches to set up commands

2.1.2 Design Constraints

The VMEbus exerciser module should be created within the following constraints:

- operating system: the project should be carried out in Windows XP professional
- PCB design tool: Altium Design Explorer (DXP) version 08
- software (firmware) environment: Xilinx[1] integrated services environment (ISE) version 9.2i
- main board dimensions: 6U main board should be created.
- daughter-card: Xilinx Spartan 3 development board from Digilent inc. [45] should be used as a daughter-card

2.2 Requirements Analysis

The previous VMEbus exerciser at iThemba LABS renders exercising job cumbersome because of its primitive style of design[2]. In addition to this, the board was limited in flexibility (hence it was not scalable), because its entire design was based on discrete logic, instead of reconfigurable logic. The solution to this problem[3] was to design a new, advanced reconfigurable exerciser so as to:

- Create a VME main board which provides interface logic between VME backplane and FPGA daughter-card;
- Use a low-cost FPGA daughter-card for:
  - writing a simple programming language to test VMEbus functionality using ADO, write and read cycles;
  - communication with the personal computer (PC), where the test commands are entered using the created graphical user interface;
- Graphical user interface (GUI), to enable transfer of test commands (ADO, write and read commands) to the daughter-card in a more graphical (hence less cumbersome), hence manner. This GUI is also used to store recently entered commands.

2.3 Exerciser Functions

Exerciser functions are divided into both hardware functions and software functions.

[1] Xilinx is the world's largest supplier of logic devices
[2] because its design was carried out solely on discrete logic, which resulted in a cumbersome design
[3] hense why this project was carried out

2.3.1 Hardware Functions

Hardware functions include:

- creating a printed circuit board (PCB) [main board] which is used to interface VMEbus backplane and VMEbus processor (FPGA daughter-card)
- a daughter-card that is used to provide VMEbus communication with the PC, using universal asynchronous receive/transmit (UART) specification. Additionally, this daughter-card is used as VMEbus processor, since the firmware is targeted to this hardware

2.3.2 Software Functions

Software functions include:

- writing a program (firmware) which is used to simulate the VMEbus commands (ADO, write and read commands) using hardware description language (VHDL[4]).
- creating a GUI which is used to enter VME test commands and store recently used commands

2.4 Design Specifications

Based on the user requirements and constraints, requirements analysis and functions of the exerciser outlined above, the following specifications were outlined:

- Protocol supported: the exerciser module should support a single cycle transfer (SCT) protocol;
- Addressing modes: A16, A24 and A32 features should be implemented by this module;
- Data widths: the board should support D08 (EO), D08 (O), D16 and D32 data widths;
- Commands supported:
  - address-only: the exerciser should be able to perform address broadcasting (without data transfer);
  - write: the module should be able to successfully write data to the slave board;
  - read: the exerciser should be able to read data from the slave module;

[4]VHDL is an abbreviation for VHSIC Hardware Description Language. VHSIC is an abbreviation for Very High-Speed Integrated Circuit

- Modes of operation:
  - bus request;
  - data transfer;
  - bus release;
- Arbitration scheme (arbiter option): single-level (SGL) priority scheme (level 3 arbitration);
- Bus release scheme (requester option): release when done (RWD);
- Graphical user interface and documentation (user manual);
- Dimension: VME 6U board;
- Mechanically robust;
- Low-cost exerciser;
- operating temperature: 0 to 55 degrees Celsius;
- The exerciser should be portable.

2.5 Acceptance Test Plan

In order to determine success of this project, an acceptance test plan/process was formulated. This plan is divided into four categories, namely mechanical, functional, GUI, and performance test processes.

2.5.1 Mechanical Test Plan

This test process is aimed at testing mechanical characteristics of the main board against VMEbus specification [1]. These tests are mentioned below:

- Board depth: standard 30 cm (300 mm) measuring ruler is to be used to measure the board depth. The expected board depth is 160 mm, as per VMEbus standard;
- Board height: like in the case of board depth, a standard 30 cm measuring ruler is to be used to measure exerciser board height. The expected board height is 233 mm, in compliance with VMEbus specification;
- Board thickness: a micrometer is to be used for this test. As per VMEbus standard, the expected thickness is 1.6mm with an error margin of 12.5%;

- Mechanical robustness: this determines mechanical strength of the designed board. This test is to be carried out by the manufacturer[5] during board fabrication.

2.5.2 Functional Test Plan

This test plan is intended to test functionality of the exerciser, against user requirements. The tests for this process include testing ADO, write and read VME cycles. C.A.E.N VME slave module[6] is to be used to test functionality of the exerciser board. Also, Xilinx Spartan 3 light emitting diodes (LEDs) are to be used to indicate success of the tested VME cycles. The left-most LED is to be active as an indication that the tested cycle is complete (successful). As for read cycles, Spartan 3 on-board 7-segment display is to be used to display the read data.

The following cycles are to be tested:

- ADO cycles
  - 16-bit addressing
  - 24-bit addressing
  - 32-bit addressing
- write cycles
  - 16-bit addressing
    - 8-bit data transfer
    - 16-bit data transfer
    - 32-bit data transfer
  - 24-bit addressing
    - 8-bit data transfer
    - 16-bit data transfer
    - 32-bit data transfer
  - 32-bit addressing
    - 8-bit data transfer
    - 16-bit data transfer
    - 32-bit data transfer
- read cycles

[5] W. H Circuits in this case

[6] for more information concerning this, visit www.caen.it

- 16-bit addressing
  - 8-bit data transfer
  - 16-bit data transfer
  - 32-bit data transfer
- 24-bit addressing
  - 8-bit data transfer
  - 16-bit data transfer
  - 32-bit data transfer
- 32-bit addressing
  - 8-bit data transfer
  - 16-bit data transfer
  - 32-bit data tranfer

2.5.3 GUI Test Plan

These tests are to be carried out to establish whether GUI could communicate with serial port(s) of the host machine (PC). Huawei E220 modem[7] is to be used to test the operation of the GUI. This testing involves sending AT commands through the GUI to the modem, and reading such commands from the modem. When the test is successful, Huawei E220 modem sends an 'ok' message, that is read from the GUI. Success of these tests implies that the host machine could communicate with Spartan 3 serial port. Tests carried out include:

- Using GUI to send commands to the serial port of the PC;
- Using GUI to read commands from the PC's serial port.

2.5.4 Performance Test Plan

Performance test procedure is intended to establish the extend to which the project mission should be executed, against the given benchamarks. The tests are given as thus:

- Electrical specifications: digital multimeter is to be used to measure voltage for interface logic. The expected voltage (according to VMEbus standard) is 5V, with tolerance of 5%;
- Short-circuit test: a digital multimeter is to be used to measure resistance between main board's 5V power pins and the ground pins. This test would be successful if resistance is to be found to be greater than 1 kilo-Ohm;

[7] for futher details about the modem, visit www.huawei.com

- Maximum trace length: from VME specification, maximum trace length should be less than 50.88 mm. This is to be measured using Altium 08 DXP, by showing report of all traces on the main board;
- Interface logic response times: oscilloscope is to be used for measurements of these respone times. The current VMEbus exerciser at iThemba LABS (the one to be replaced) has interface response times of 300 nanoseconds. This would be used as a benchmark for the response time of the new exerciser board. Thus, the new board's response time should be less than 300 nanoseconds, for this project to be successful;
- Program execution time: oscilloscope is to be used to determine average execution time of the program. The execution time should be less than 10 microseconds for this program to be successful;

2.6 Summary

This chapter discussed VMEbus exerciser design process. User requirements, together with design constraints, were given. Next, these requirements were analyzed, in order to establish how such requirements could be met. Additionally, design specifications were established. After establishing design specifications, acceptance test plan was outlined. The next chapter provides details on the design of VMEbus exerciser.

Chapter 3

System Design

This chapter gives a description of VMEbus exerciser design. The design is carried out in concordance with design specifications outlined in Chapter 2. The first section of this chapter deals with design tradeoffs. This is followed by a section which covers hardware (main board) design. The main board is used to house VMEbus interface logic[1]. Following this section is a section on firmware design. This section gives description of the design of soft processor cores which perform the required VMEbus functions, in order to meet the user requirements. The last section provides a description of GUI design.

One of the advantages of VMEbus standard is its flexibility in enabling the system to be upgraded to meet specific performance requirements. The complexity of VMEbus design, coupled with its modularity, makes no simple right or wrong way to design, and the implementation may vary from application to application [29].

3.1 Design Tradeoffs

3.1.1 Hardware Design Tradeoff

In hardware design, two options were considered, namely integrating both main board and daughter-card into one unit, or using these boards separately. Also, single-slot and double-slot board options were considered, and a choice was made.

3.1.1.1 Main Board and Spartan 3 Daughter-Card Separate

The advantage of this approach is simplicity of design and implementation. The downside this arrangement requires a lot of space, resulting in a bulky VMEbus exerciser. Spartan 3 daughter-card from Digilent inc. is shown in Figure 3.1.

[1] VMEbus interface logic consists of buffers and receivers of VMEbus signals

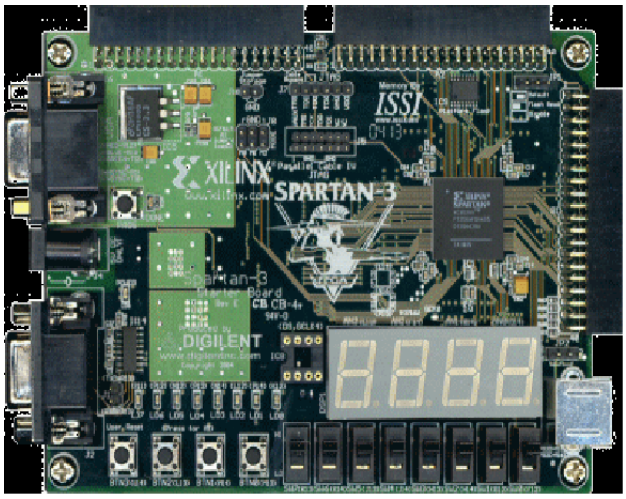

Figure 3.1: This development board was used as a daughter-card of the exerciser [45].

3.1.1.2 Main Board and Spartan 3 Daughtercard Integrated Together

This design option is shown diagrammatically in Figure 3.2.

The disadvantage of this arrangement is complexity in design, because of electromagnetic intefe-rence that would result from housing two different boards in the same unit. On the other hand, it is a good option because it results in a compact/portable product.

Verdict

Based on the criterion of compartibility, the second option (integrating main board and daughter-card into one unit) was chosen.

3.1.1.3 VME 6U Single Slot or Double slot

Another hardware design challenge was whether to use a double slot or a single slot board for the unit. A single-slot board result in a compact design, but would introduce serious electromagnetic inteference which would negatively affect the functioning of the board. On the other hand, a double-slot board would be too wasteful for VME system slots, but would result in a higher-performance, fully operational board.

Verdict

Based on the functionality criterion, 6U double-slot board design was chosen over a single-slot design.

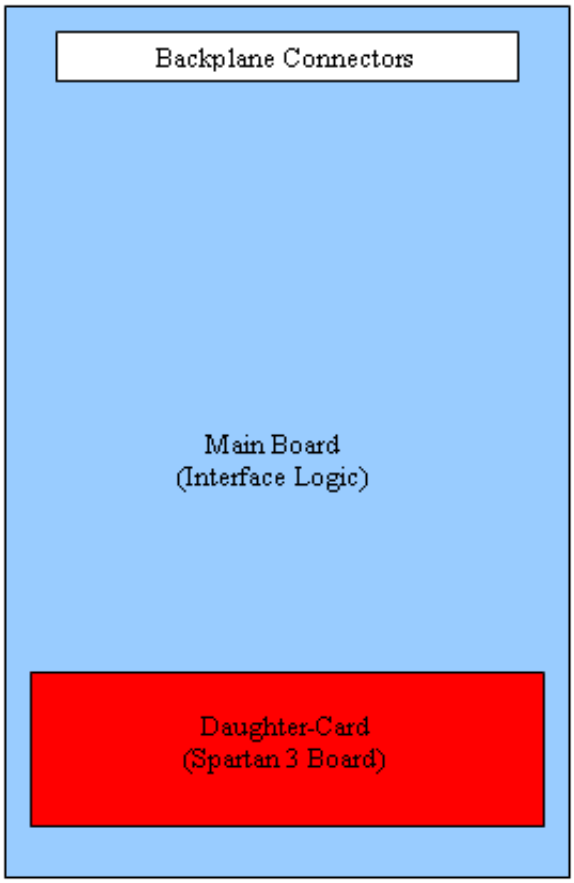

Figure 3.2: This Figure shows a design option where Spartan 3 Daughter-card is integrated together with main board. The Figure is not drawn to scale.

3.1.2 Firmware Tradeoffs

Firmware options included implementing VMEbus processing directly on FPGA logic or using a soft-core processor. Also, a decision was made on whether to use Microblaze or Picoblaze soft-core.

3.1.2.1 FPGA Logic or Soft-Core

Using FPGA logic-level design for processing is difficult because a hardware description language (VHDL in this case) is verbose, making the modelling of intended hardware very cumbersome. On the other hand, using soft-core designs results in rapid development of a system (because of simplicity associated with soft-cores' employmment) while flexibility (due to the use of reprogrammable FPGAs) is still maintained.

Based on this observation, soft-core design was chosen over FPGA logic-level design.

3.1.2.2 MicroBlaze or Picoblaze Soft Core Processor

MicroBlaze is a 32-bit Reduced Instruction Set Computer (RISC), Harvard architecture, high performance (200 MHz) Xilinx soft core that is optimized for embedded applications. Though it yields high performance, the downside of this core is that it is not cost-effective, since a license fee is paid in order to use it (MicroBlaze).

Picoblaze is a low-cost, low-performance (88 MHz) 8-bit RISC microcontroller core from Xilinx, oprimized for Spartan 3 and Virtex families of FPGA. In addition to its cost-effectiveness, this core is simpler to implement (compared to MicroBlaze) and uses less FPGA slices than MicroBlaze. Picoblaze processor is shown in Figure 3.3.

Verdict

Based on the product specification, board performance was traded off for system cost. Hence Picoblaze soft-core was chosen over MicroBlaze.

3.1.3 GUI Tradeoffs

For GUI design, the decision was on whether to use a compiled programming language or an intepreted programming language.

Compiled languages like C++ and Java are fast to execute, but have a very steep learning curves. On the other hand, an intepreted language like Python is more verbose than compiled languages, meaning it is easier to get things done in Python than it is in either C++ or Java.

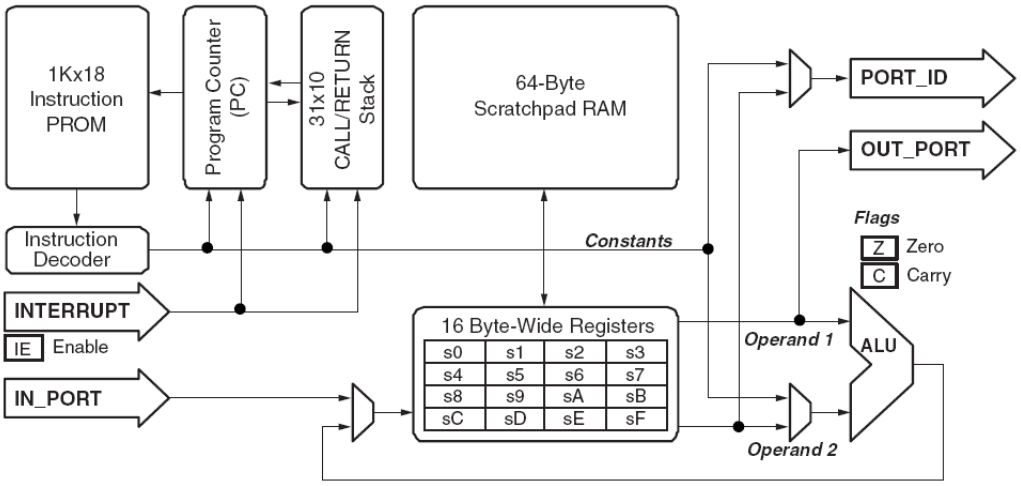

Figure 3.3: This Figure shows a block diagram of Picoblaze soft-core, with its (Picoblaze's) different functional blocks such as general-purpose registers, instruction program storage, Arithmetic Logic Unit (ALU), scratchpad Random Access Memory (RAM) and program counter [22].

Based on the notion that central processing unit (CPU) time is much cheaper than development time, performance superiority of compiled languages is traded off for simplicity of Python[2].

3.2 Hardware Design

Successful PCB design relies on carefully meeting system time requirements, as well as board lay-out and decoupling in order to cater for signal integrity issues. The following design considerations were taken into account during PCB design:

3.2.1 Main Board Lay-Out

This is a four-layer board lay-out with two outer layers used for signal routing, and two internal layers used as power planes. This lay-out is shown in Figure 3.4.

3.2.2 Use of FR-4[3] Material

FR-4 is the most-used material in printed circuit board design, because of its minimal losses. Additionally, this material is cheap, hence easily affordable.

[2]and this resulted in the rapid development of the GUI
[3]Flame Retardant with dielectric constant of 4

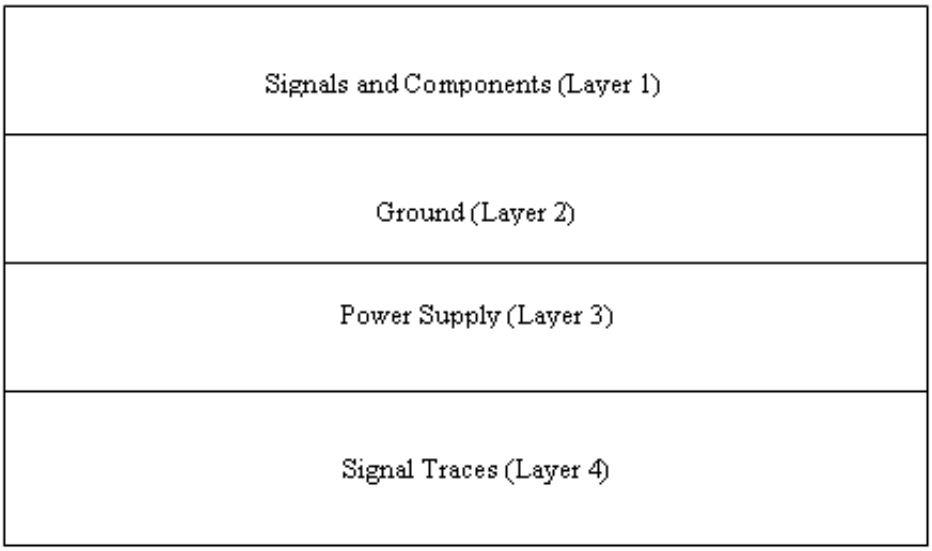

Figure 3.4: A Block diagram of a four-layer PCB. Outer layers are used for signal routing, with layer also being used to mount components. Inner layers are used as power planes.

3.2.3 Power Supply

The main board components were powered from the VME backplane, using power pins defined in the specification [1]. Trace width of 20 mils was used, with copper thickness of 3 Oz. Other non-power traces were designed with a trace width of 6 mils. The backplane was also used to power Spartan 3 daughter-card. These trace widths were also used in order to cater for current-carrying capacity of the board.

3.2.4 Impedance Matching

VMEbus backplane provides a bank of resistors for termination of VMEbus signals to cater for discontinuities caused by plated-through holes/vias and connectors on the main module.

3.2.5 Capacitive Loading of TTL Components

As per VMEbus specification [1], capacitive loading should be kept below 20 pico-Farads. This requirement was taken care of while designing the board, and it was met by ensuring shorter signal distances.

3.2.6 Return Currents

A four-layer (2 signal layers and 2 power planes) was designed. Both signal layers (top and bottom layers) shared a group plane, thus reducing discontinuities in return currents. Also, in order to reduce discontinuities, 45 degrees routing was preferred over 90 degrees routing.

3.2.7 Use of Decoupling (Bypass) Capacitors and Filters

According to VMebus standard, capacitance of bypass capacitors should be in the range of 10 to 100 nano-Farads. These values were used in this design. The capacitors were also connected directly to the power and ground planes, as a way of reducing inductive effects of the traces. Again, in order to mitigate the track inductive effects, shorter distances were employed between bypass capacitors and vias that connect these capacitors to the power planes.

Filtering capacitors were also used to filter out unnecessary signals (frequencies). These capacitors have higher values than those used for decoupling, so as to filter out higher frequency signals.

3.2.8 Slew Rates

Slew rates were taken into account during component placement. Optimal placement of components was carried out to ensure that slew rates do not have a negative impact on the performance of the board.

3.2.9 Use of Ribbon Cables

Two sets of ribbon cables were used; the shorter set used to connect the daughter-card to the main board when both of them were in the VME cage (during operation), and a longer set that was used to connect these modules during the programming of the daughter-card (when main board was in the VME cage, and the daughter-card out of the cage, during its programming).

3.2.10 Interface between Exerciser Board and the Computer

RS-232 was used to connect PC to Spartan 3 development board. RS-232 standard is known for its reliability and elegance. Also, this serial standard is easily supported by different hardware vendors[4]. Its robustness also came in handy during the design of this board.

3.2.11 Even Distribution of Components

Aesthetic design was traded off for high performance design. This is because VME 6U is an extension board, so most of the signal pins are concentrated on the first connector[5], with the second connector using only one (row B) of the three rows available. This is the reason why most components were placed closer to the first connector; in order to reduce signal losses and electromagnetic inteference.

[4] Xilinx also supports the standard, hence why Spartan 3 board has RS-232 port

[5] 96-pin DIN (41612, type C), 3-row connector

3.3 Firmware Design

This section describes the design of firmware (VHDL code)in order to carry out VMEbus exerciser cycles. This firmware design is made up of three parts, namely design of universal asynchronous receiver/transmitter (UART) module, design of ADO, read and Write cores, and input/output (I/O) port assignment using Spartan 3 extension connectors. Firmware architecture is shown in Figure 3.5.

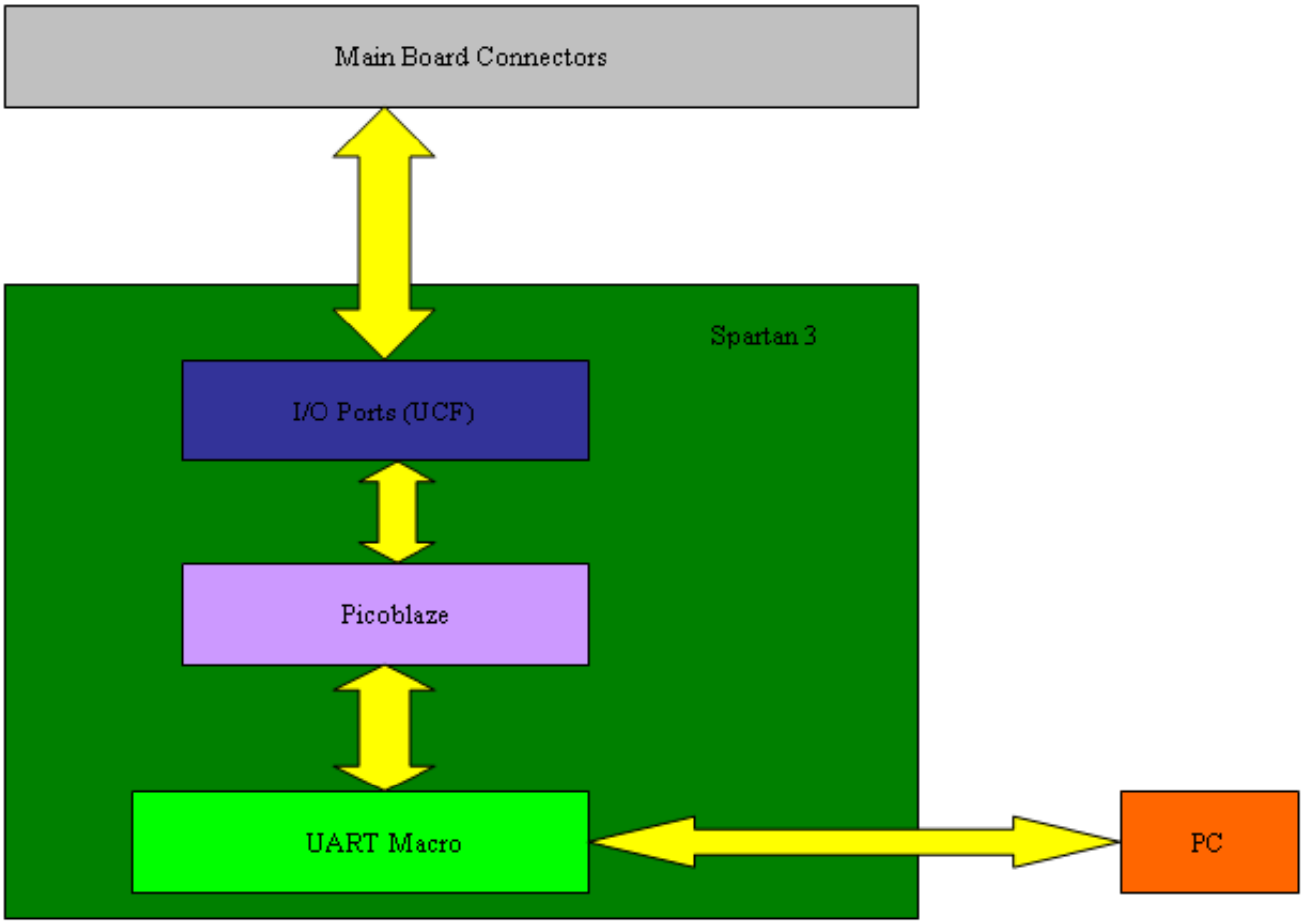

Figure 3.5: A block diagram of firmware architecture. A UART macro is used to provide communication between the PC and Picoblaze. I/O port assignment is carried out using User Constraint File (UCF) for connection between Spartan 3 extension connectors and main board connectors.

3.3.1 UART Design

This UART module is robust,[6] and is easily implemented in Picoblaze due to the fact that it is implemented in VHDL. It was used to define an interface between the computer and the Spartan 3 FPGA.

[6]it is less highly reliable and less noisy

The UART macro used in this project was downloaded free of charge from Xilinx website. Block diagrams for UART transmit (Tx) and Receive (Rx) macros are given in Figures 3.6 and 3.7 respectively.

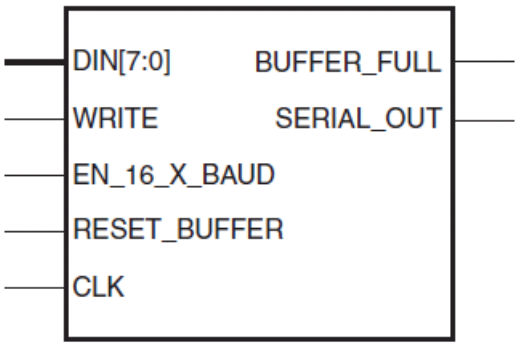

Figure 3.6: UART Tx macro. Its inputs include an 8-bit data_in (DIN) signal, write buffer, reset buffer, clock and timing reference signal (EN_16_X_BAUD). The outputs consist of buffer-full indicator; which is activated when first-in first out (FIFO) buffer is full, and serial data output. A picture courtesy of [9].

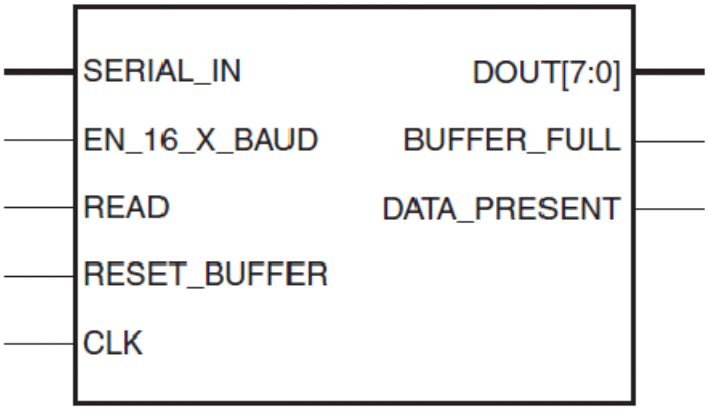

Figure 3.7: UART Rx macro. Its inputs include a clock, reset buffer, read buffer, serial data input and timing reference signals. Outputs include data present indicator, which is used to signal availability of data at the output port, buffer full indicator and data output port (DOUT) signals. A picture courtesy of [9].

3.3.2 Design of VME Cores

ADO, write and read cores were created using Picoblaze soft core designs. Assembly language was used to model finite state machines of these cores, and these were later compiled into VHDL programs. These cores form the integral part of this exerciser module, because they carry out the computation/processing that is required to test VME functionality. As with UART macros, Picoblaze cores were also downloaded free of charge from Xilinx website. Assembly language soft-core algorithm is shown in Algorithm 3.1, while a top-level diagram of Picoblaze is shown in Figure 3.8.

Algorithm 3.1 This Algorithm calls three major routines of the program, namely bus_request (which performs bus requests), data_transfer (which performs data transfer operations) routines and bus_release routine, which facilitates relinquishing of the bus by the exerciser.

forever: CALL bus_request
CALL data_transfer
CALL bus_release
JUMP forever

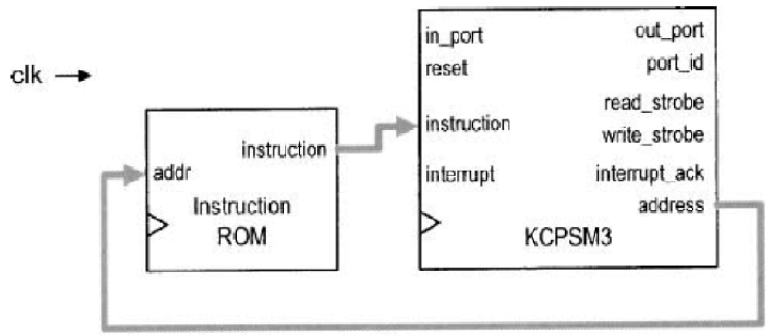

Figure 3.8: Top-level diagram of Picoblaze. Picoblaze is a KCPSM3 processor. A picture courtesy of [10].

3.3.3 I/O Ports

Extension connectors on Spartan 3 were used to route VME signals between the interface logic and the FPGA. During design, user constraint file (UCF) was used to physically connect (route) FPGA I/O signals using extension connectors.

3.4 GUI Design

A good GUI makes an application easy, practical and efficient to use [25]. High quality GUIs are crucial for successful adoption and use of software applications [40]. Subsections in this section outline points which were considered for design of a successful GUI.

3.4.1 Choice of Python GUI Toolkit

Python has no native support for GUI programming, but Python bindings (wrappers) could be used to develop GUIs in Python. Normal Python GUI toolkits/wrappers are:

- Tkinter;
- PyQt;
- WxPython;

- FxPython;
- PyGTK;
- Pythonwin.

PyQt was used in this project because of its use of innovative signal/slot approach; which couples GUI items and actions.

3.4.2 GUI Usability

The GUI was made to enable the user to easily interact with it. Also, it was ensured (during GUI design) that the interface would let the user accomplish the task of VME exercising effectively. In order to strike a balance between GUI's usability and complexity, depth level of GUI's hierarchy was limited to at most three levels. This is known as a three-click rule.

3.4.3 Functionality over Form

Functionality of theGUI was given a higher preference over its form. Common GUI features such as file menubar and access to help were included in the design.

3.4.4 Avoidance of Clutter

The user interface was kept simple, clear and unambigous. A cluttered user interface has a downside of confusing whoever is using it, because of many widgets (some of which might be unnecessary) concentrated together in the same user interface.

3.4.5 Consistency in Form and Function

The GUI was designed to have the same outlook as other user interfaces on windows XP, so as not to confuse the user. This was ensured by providing a design with a common look with the normal Windows-based user interfaces. Also, the GUI was designed to ensure that data could be sent easily to the serial port and read easily from the serial port. It was also designed to make possible the storing ('caching') of recently used commands.

3.5 Summary

This chapter gave a description on how the exerciser was designed. The design was divided into three categories, namely;

- Hardware design;
- Firmware design; and
- graphical user interface design.

The next chapter covers the implementation of VMEbus exerciser.

Chapter 4

System Implementation

This chapter provides details for VMEbus exerciser system implementation. It is divided into three sections, namely:

- Section One: this section discusses the implementation of exerciser PCB. It covers issues such as schematic design, board layout, manufacturing and assembly
- Section Two: it discusses firmware implementation. It covers issues such as ADO, read and write cores' implementation and UART macros
- Section Three: it gives details for implementation of graphical user interface, covering issues such as PyQt4 toolkit, Pyserial module and sending/reading VMEbus commands to/from the serial port

4.1 PCB Implementation

This PCB was designed to house VMEbus interface logic. It is a four-layer, 6U double-slot module (with a maximum of 32-bit data transfer) VME master interface designed in Altium 08 DXP. The overall thickness of the board is 65.6 mils.

4.1.1 Components Used

The following components were used for interface logic:

- Signal tranceivers (four tranceivers);
- Signal drivers (four buffers);
- A signal receiver.

A summary of these components is provided in Table 4.1[1].

Table 4.1: This table provides a summary of components used to implement PCB.

Component	Description	Quantity
header 20x2	header, 20-pin, dual row	3
cap1	capacitor (RAD-0.1)	12
cap2	capacitor (RAD-0.3)	1
header2	header, 2-pin	1
CNVME96	96-pin connector	2
SN74F245N	octal bus tranceiver with 3-state outputs	6
SN74LS645N	octal bus tranceiver	4
DM74LS245	3-state octal bus tranceiver	1

4.1.2 Schematic Design

This was used to provide details of connections between components, in the form of block diagrams. As is the norm in schematic capture, the following practice was implemented:

- Components were arranged in such a way that inputs are to the left of schematic diagram, and outputs to the right;
- Bypass capacitors were placed next to the components they were meant to provide decoupling to;
- The schematic was broken down into logical blocks;
- Net labels were used to increase readability and traceability of nets on the board.

PCB schematic diagram is provided together with other project material in Appendix E: the compact disc.

4.1.3 Component Placement

This step was carried out so as to ensure simplest routing topology and maximum performance of the board. In order to achieve this, the following steps were taken:

- Wherever possible, the components were divided into functional building blocks;
- Components were also placed based on their trace lengths, so that maximum trace length could be in compliance with VME specification. This also resulted in reduced trace capacitance, inductance and resistance; all of which are undesirable for proper PCB design;

[1] capacitive loading of these components was consistent with VME specification

- Bypass capacitors were placed as close as possible to the components, in order to reduce ground current loops. Also, these capacitors were connected directly to the power planes, in order to reduce inductive effects of the traces.

4.1.4 Board Layering

A four-layer board was designed. The board consists of two signal lagers (upper and lower layers) and two power planes (internal layers). Separate ground and supply planes were used in order to ensure high signal integrity.

Figure 4.1 shows the layer stack of exerciser board.

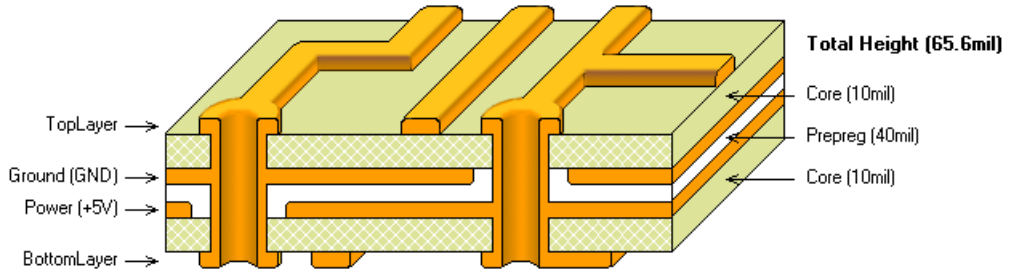

Figure 4.1: Four-layer PCB stack. Outer layers are used for signal routing, and inner layers are used as power planes. Separate ground and supply planes were used in order to ensure higher signal integrity. This figure was generated using Altium 08 DXP.

4.1.5 Signal Routing

Auto-routing was used to route all signals on the PCB. In order to maintain superior performance of the board, design rules were implemented in such a way as to minimize bending of the traces. This means that gradual turns (45 degrees turns) were used instead of 90 degrees turns, resulting in high signal integrity due to reduced signal electromagnetic inteference (EMI). Another routing step which was taken to ensure successful implementation of the board was the uses of through-hole component legs to connect top traces to bottom traces, which in turn reduced number vias required[2].

4.1.6 Board Manufacturing and Assembly

This board was fabricated by W.H Circuits, and its assembly was done at iThemba LABS. Figures 4.2 and 4.3 show Gerber files for both top and bottom routing layers respectively. These Gerber files

[2] vias should be reduced/eliminated because they introduce current discontinuities, which in turn cause signal reflections and hence reduced board performance

were sent to W.H Circuits for fabrication of the exerciser board. Figure 4.4 shows this exerciser main board after completion[3].

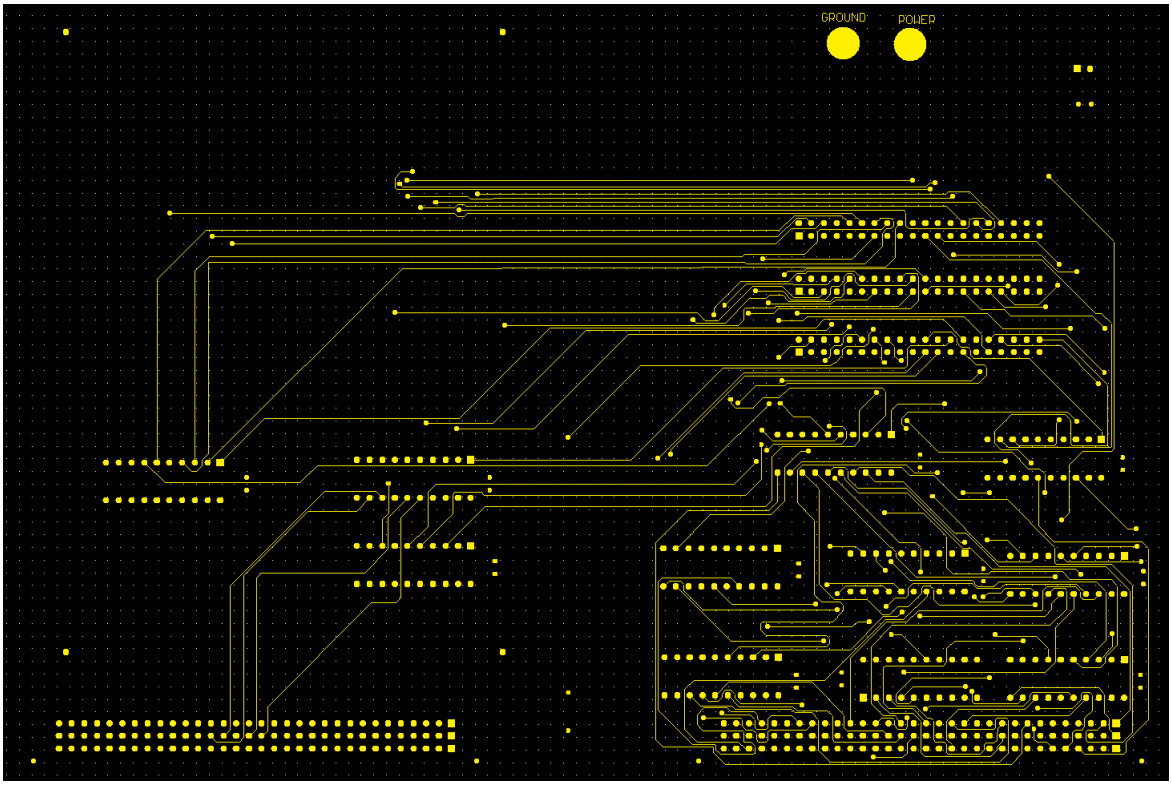

Figure 4.2: Top-Layer Gerber File.

4.1.7 Complete Hardware Integration

The main board was integrated with Spartan 3 daughter-card, to form a single unit of 6U double-slot, as outlined in Chapter 3. This is shown in Figure 4.5.

4.2 Firmware Implementation

Firmware implementation involves building a code for ADO, write and read soft-cores, and UART interface for connection to serial port[4].

[3] after fabrication and assembly
[4] using RS-232 standard

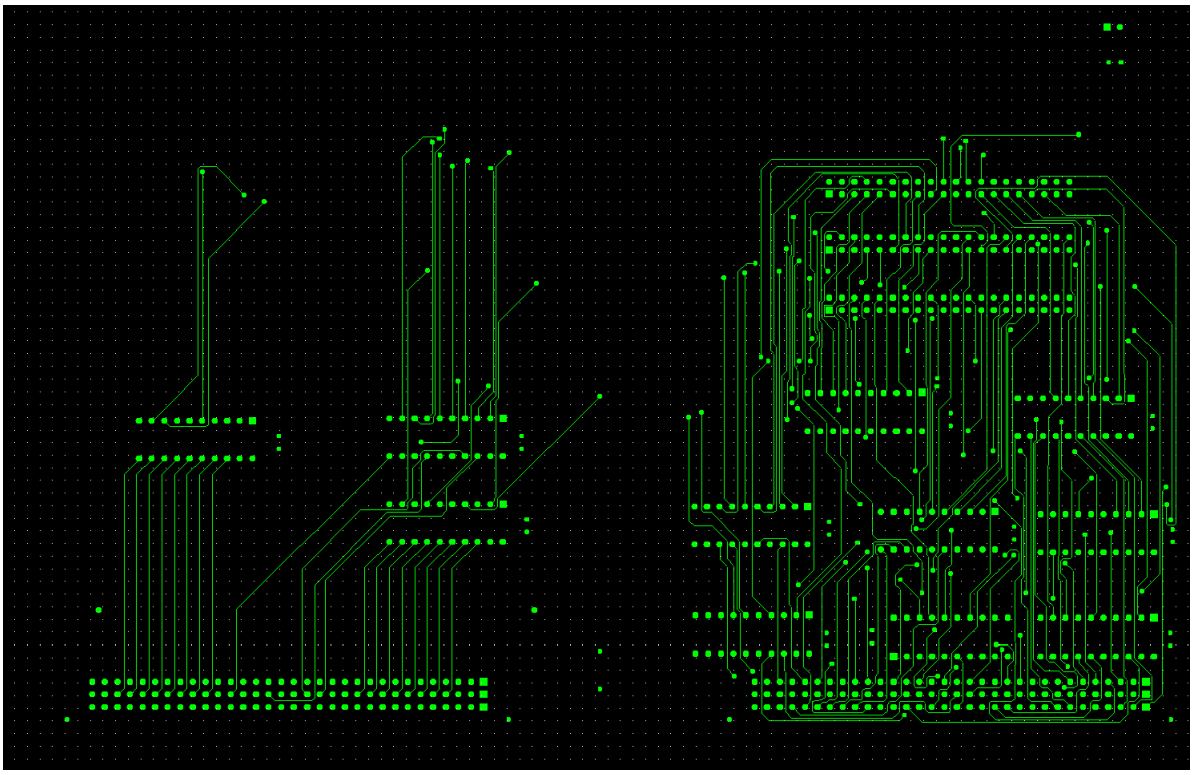

Figure 4.3: Bottom-Layer Gerber File.

4.2.1 VME Cores

These three VME cores were developed using Picoblaze. Picoblaze uses both Assembly and VHDL languages. Due to its simplicity, assembly language is used in Picoblaze designs to build a finite state machine for the application. This state machine is then converted to VHDL Program Read-Only Memory (PROM) using Constant Coded Programmable State machine (KCPSM3) assembler.

4.2.1.1 Assembly Language Component

The assembly language is made up of the following components:

- Declarations part: to provide declarations for all input and output ports used;
- Main program part: this component is used to carry out all the finite state machine computations and UART routine. Finite state machine computations include bus request, data transfer and bus release.

Bus Request Routine

This routine was implemented using the following steps:

- Drive BR3* signal low;

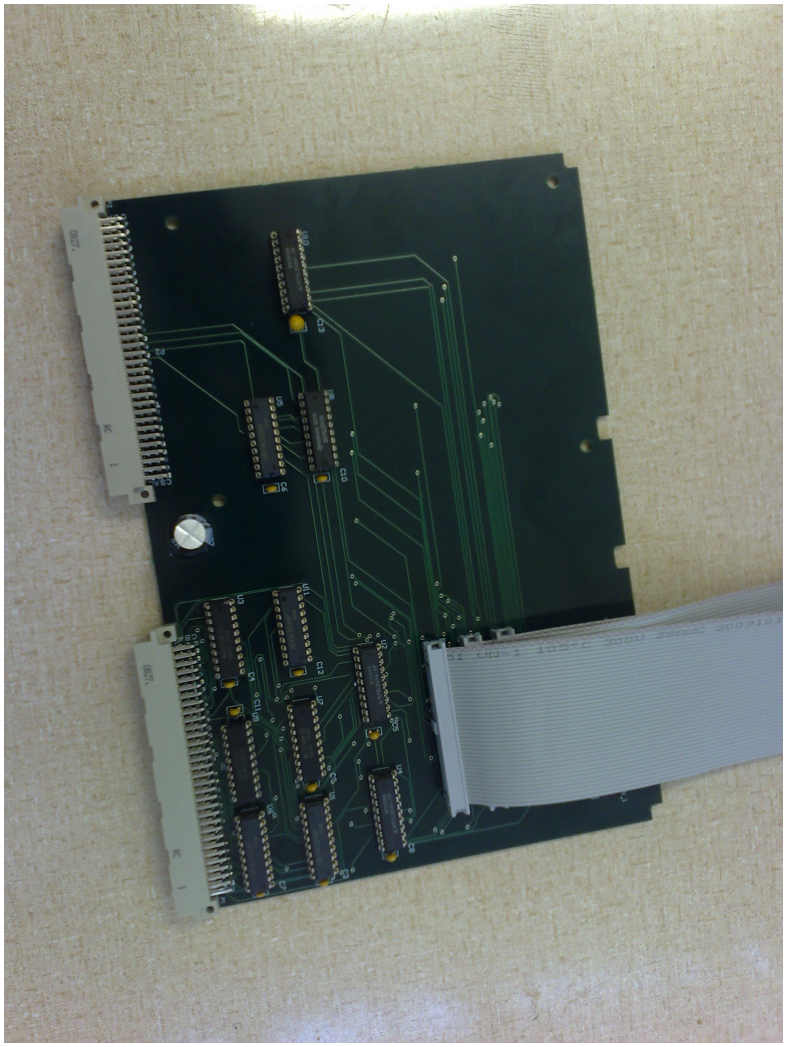

Figure 4.4: Fully Assembled Main Board.

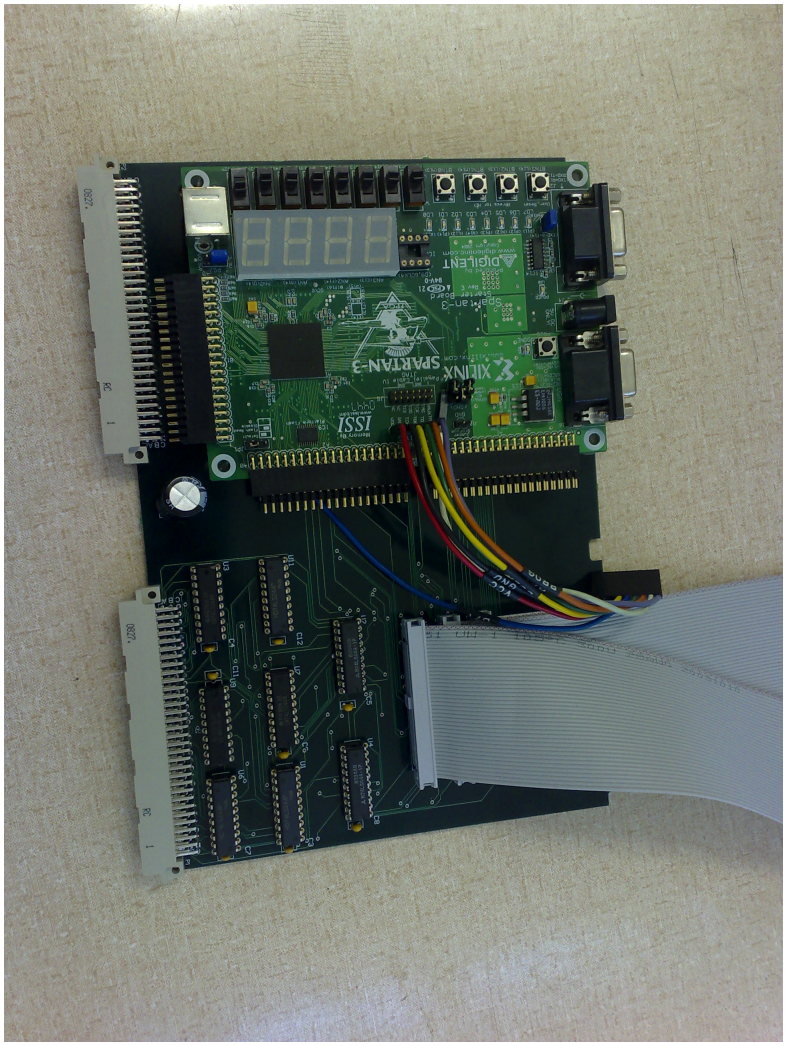

Figure 4.5: A complete implementation of exerciser hardware. Interface logic module and Spartan 3 development board are integrated into a single unit of 6U double-slot system.

- Monitor BG3IN* signal;
- If BG3IN* is low, drive BBSY* low.

Data Transfer Routine

This is the most important routine of the program. It performs ADO, write and read VME operations/cycles. These cycles are given in subsection 2.5.2.

Bus Release Routine

This routine was implemented by driving BBSY* signal high, in order to relinquish bus services.

Serial Connection, Assembly Code Simulation and Conversion to VHDL

Assembly code for reading characters from serial port and writing commands to serial port wass also developed. Again, the simulation of entire assembly code was carried out using a simulation tool from Mediatronix inc. Then after the simulation had been done, assembly code was converted to synthesized to VHDL PROM using KCPSM3 assembler.

Algorithms 4.1 and 4.2 show how to send a character to the serial port and how to read a character from serial port respectively, using assembly language.

4.2.1.2 VHDL Component

VHDL component of the firmware was implemented using Xilinx ISE 9.2. KCPSM3 processor was used to carry out processing of VME operations. It was downloaded as a VHDL file from Xilinx inc. This core was then connected to the PROM generated by KCPSM3 assembler, and UART module.

KCPSM3 implementation in VHDL is shown in Figures 4.6 and 4.7.

4.2.2 UART Macro

This was used to create a circuit to send data through a serial port. It was used in conjunction with RS-232 standard which is used in Spartan 3 board. This macro contains both receive and transmit components, to transfer data to and from the serial port.

UART macro was configured using the following parameters:

- Baud rate: 9600 bits per second;
- Data bits: 8;

Algorithm 4.1 This algorithm is used to send character to the serial port using assembly language.

```
uart_send_data: INPUT s0, uart_status_port
TEST s0, tx_full
JUMP Z, send_data
JUMP uart_send_data
send_data: OUTPUT sF, tx_data
CALL delay_1ms
RETURN
;
delay_1ms: LOAD sD, 19 ; 25*40us=1ms
wait_1ms: CALL delay_40us
SUB sD, 01
JUMP NZ, wait_1ms
RETURN
;
delay_40us: LOAD sC, 28 ; 40*1us=40us
wait_40us: CALL delay_1us
SUB sC, 01
JUMP NZ, wait_40us
RETURN
;
```

Algorithm 4.2 Algorithm to read a character from the serial port.

```
uart_receive_data: INPUT s0, uart_status_port
TEST s0, rx_data_present
JUMP NZ, receive_data
JUMP uart_receive_data
receive_data: INPUT sF, rx_data ; receive character
CALL uart_send_data ; echo received character
CALL delay_1ms
RETURN
```

```
-- KCPSM3 declaration
component kcpsm3
    port ( address : out std_logic_vector(9 downto 0);
           instruction : in std_logic_vector(17 downto 0);
           port_id : out std_logic_vector(7 downto 0);
           write_strobe : out std_logic;
           out_port : out std_logic_vector(7 downto 0);
           read_strobe : out std_logic;
           in_port : in std_logic_vector(7 downto 0);
           interrupt : in std_logic;
           interrupt_ack : out std_logic;
           reset : in std_logic;
           clk : in std_logic);
end component;
```

Figure 4.6: KCPSM3 VHDL Declaration [8].

```
processor : kcpsm3
port map( address => address,
          instruction => instruction,
          port_id => port_id,
          write_strobe => write_strobe,
          out_port => out_port,
          read_strobe => read_strobe,
          in_port => in_port,
          interrupt => interrupt,
          interrupt_ack => interrupt_ack,
          reset => reset,
          clk => clk);
```

Figure 4.7: KCPSM3 VHDL Instantiation [8].

.

- Parity: none;
- Stop bit(s): 1.

4.3 GUI Implementation

The user interface was implemented using PyQt4.6.2 toolkit. Given below is the summary of procedure used to implement this GUI:

- Integrated Development Environment (IDE) chosen: Netbeans 6.9 from Sun Microsystems;[5]
- Programming language used: Python 3.1;
- Python modules used: sys, PyQt4 and pyserial modules
  - Sys module: this is a native Python module;
  - Pyserial: this module is used to establish communication with the serial port;
  - PyQt4: this module is used to build the user interface. It is a Python wrapper which uses C++ Qt library;
- Documentation: the user manual was created together with the user interface, to provide further support (help) on how to use the GUI.

Figure 4.8 shows the main window of the created user interface.

4.4 Summary

This chapter provided the implementation of VMEbus exerciser. It covered the following critical implementation stages:

- PCB implementation;
- Firmware implementation; and
- GUI implementation.

The next chapter covers discusses obtained from running VME exerciser system tests, followed by analysis of such results.

[5] Sun Microsystems is now taken over by Oracle Corporation

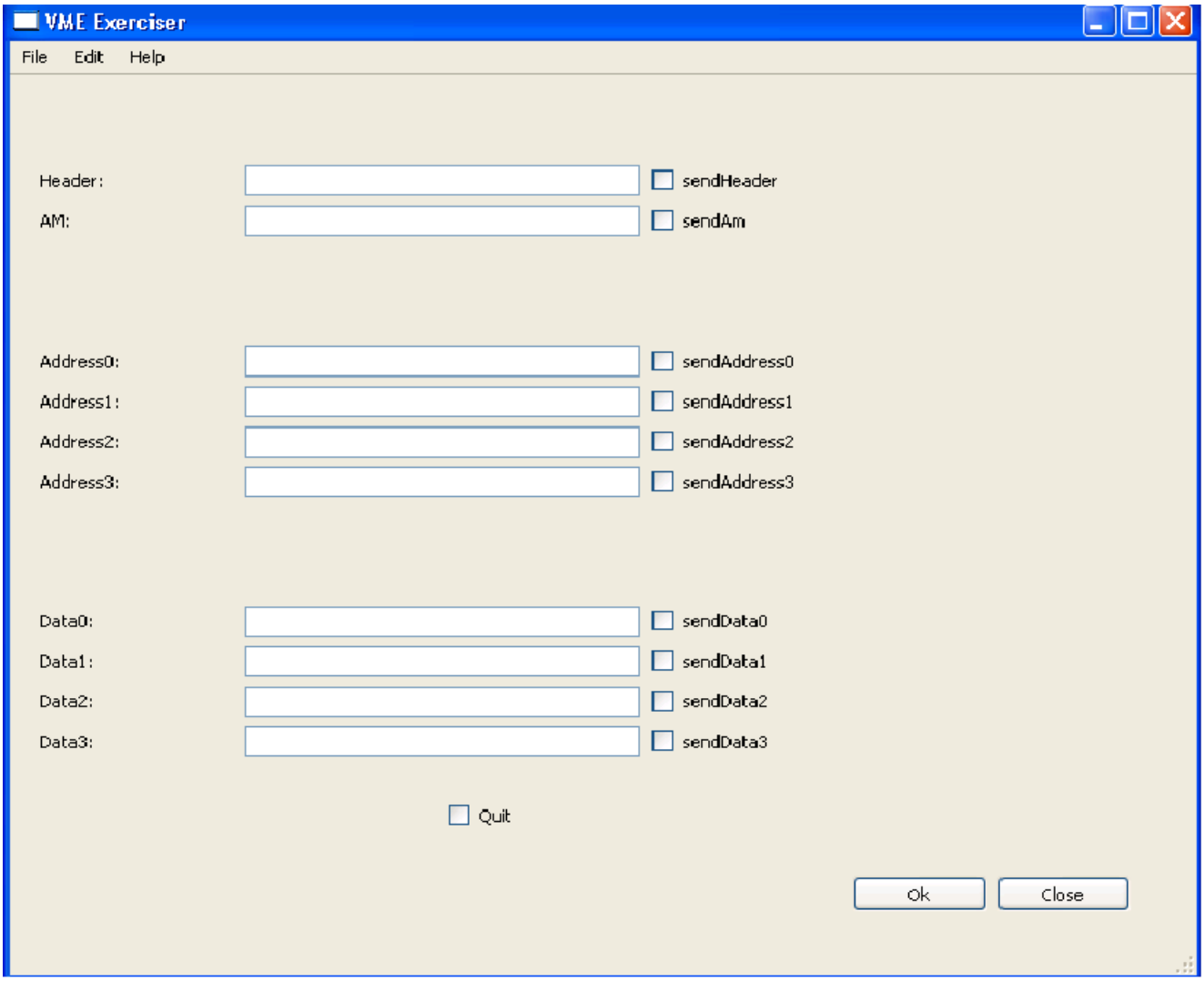

Figure 4.8: GUI Main Window.

Chapter 5

System Verification and Results Analysis

This chapter discusses tests performed on the system to validate its operation. The tests are divided into four categories, namely;

1. Mechanical tests;
2. Functional tests;
3. GUI tests; and
4. performance tests.

The obtained results are then analyzed so that conclusions could be drawn, and further work recommended.

5.1 System Verification

5.1.1 Mechanical Tests

These tests were performed to test mechanical characteristics of exerciser PCB against VMEbus specification. The following results were obtained:

- Board depth: 160 mm;
- Board height: 233 mm;
- Board thickness: 1.7mm.

5.1.2 Functional Tests

These tests were carried out to determine system's capability to to meet user's functional requirements. These functional requirements are:

- ADO;
- Write; and
- Read operations.

The tests were successful, meaning that the system does indeed solve the problems it was intended to solve. Tables 5.1, 5.2 and 5.3 summarize in tabular forms ADO, write and Read operations' findings.

Table 5.1: ADO Operations Results.

ADO Cycle	Status	Remarks
A16	ok	
A24	ok	
A32	ok	

Table 5.2: Write Operations Results.

Write Cycle	status	Remarks
A16D08	ok	
A16D16	ok	
A16D32	ok	
A24D08	ok	
A24D16	ok	
A24D32	ok	
A32D08	ok	
A32D16	ok	
A32D32	ok	

Table 5.3: Read Operations Results.

Read Cycle	Status	Remarks
A16D08	ok	
A16D16	ok	
A16D32	ok	
A24D08	ok	
A24D16	ok	
A24D32	ok	
A32D08	ok	
A32D16	ok	
A32D32	ok	

5.1.3 GUI Tests

The tests were performed to determine user interface's functionality and usability. The functionality included both reading from and writing to the system. These tests were successful, as it was possible to get acknowledgement message from the serial port in tests.

Usability tests were carried out to determine GUI's ease of use. This was assessed by easy with which the user interacted with the interface, and GUI's conformity to three click rule. Figure 5.1 shows interface's help menu.

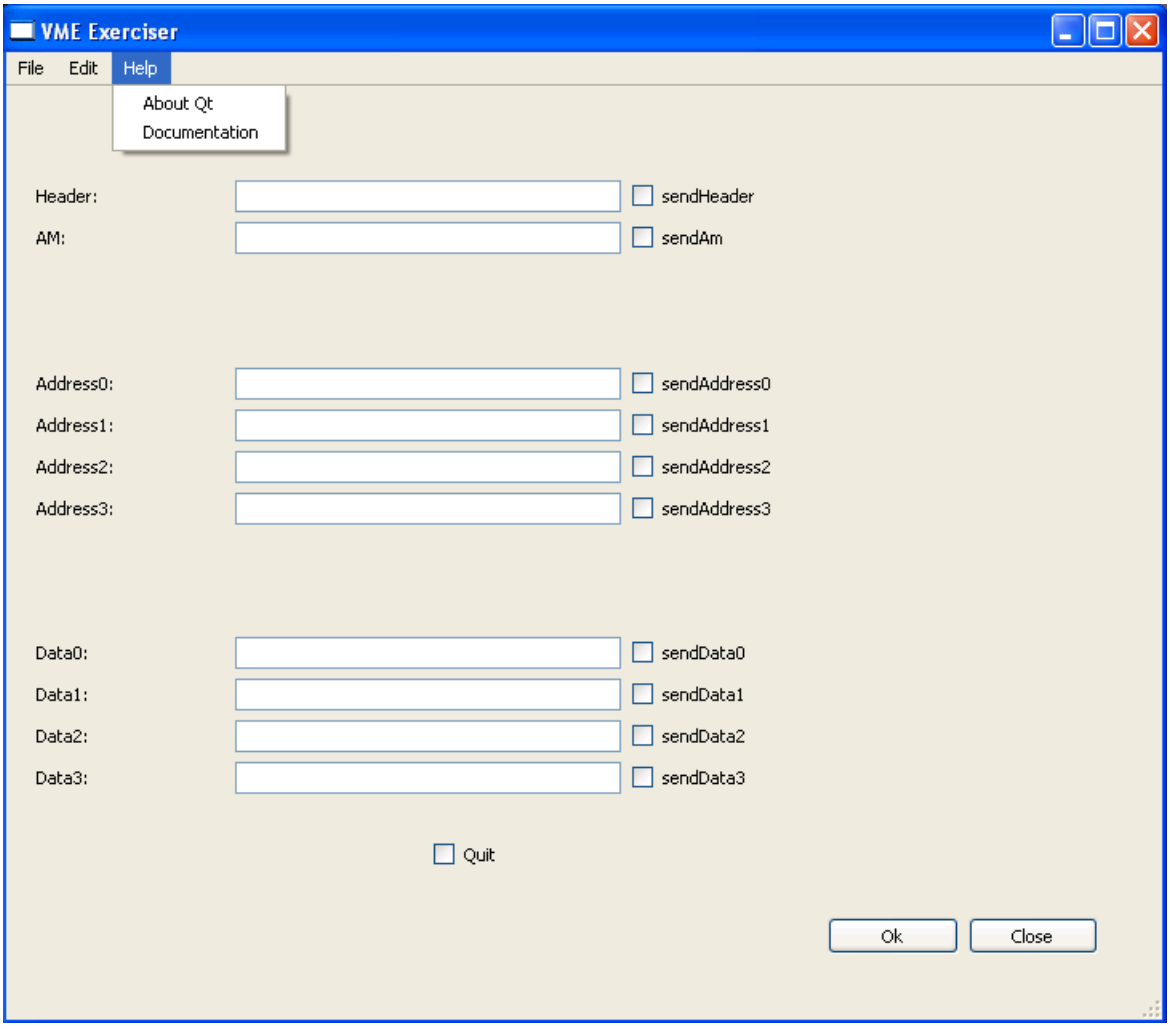

Figure 5.1: GUI file menu. Documentation action is used to access GUI's user manual.

5.1.4 Performance Tests

These tests were run to determine the extend to which a function was performed, against the given benchmarks. The following criteria were used to measure system performance test as per Subsection 2.5.4:

- Electrical tests;
- Short-circuit tests;
- maximum trace length measurement;
- Interface logic response times;
- Program execution time.

5.1.4.1 Electrical Tests

Digital multimeter was used to measure voltage of the main board. These electrical tests are summarized in Table 5.4.

Table 5.4: Summary of Electrical Tests.

Integrated Circuit (IC) Name	Voltage When Off (mV)	Voltage When On (V)
U1	4.6	4.9
U2	4.7	4.8
U3	4.7	4.9
U4	4.9	4.8
U5	4.5	4.9
U6	4.8	4.9
U7	4.7	4.9
U8	5.0	4.8
U9	4.5	4.8
U10	4.5	4.8
U11	5.0	4.9
CONN_1	4.7	5.0
Test Point	4.8	4.9

5.1.4.2 Short-Circuit Tests

Digital multimeter was used to measure resistance between 5V power plane and the ground plane. It was found to be 1.4 kilo-Ohms. Resistance values were also taken between IC 5V pins and IC ground pins. Table 5.5 summarizes these findings.

5.1.4.3 Maximum Trace Length

Altium 08 DXP was used to measure maximum trace length. This length was measured to be 50.76 mm.

Table 5.5: This table provides a summary of different IC resistances between 5V and ground pins.

IC Name	Resistance (kilo-Ohms)
U1	1.4
U2	1.7
U3	1.6
U4	1.8
U5	1.8
U6	1.6
U7	1.6
U8	1.5
U9	1.8
U10	1.4
U11	1.8
CONN_1	1.4
Test Point	1.7

5.1.4.4 Interface Logic Response Times

Oscilloscope was used to measure interface logic response times. The signals which were used for this test include:

- BG3IN*;
- BERR*;
- DTACK*;
- A01;
- D00;
- AM0;
- AS*;
- IACK*; and
- LWORD*.

Figure 5.2 shows a graph of BG3IN*[1] signal's response time. Graphs for other signals' response times are provided in Appendix D.

[1] BG3IN* signal was used because the exerciser module employs a single level arbitration scheme

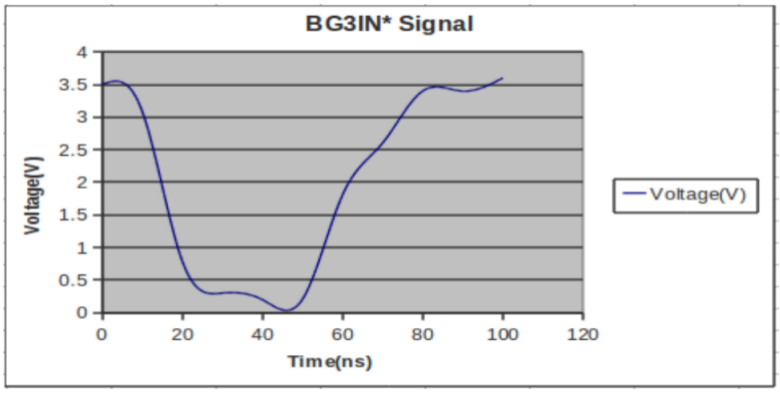

Figure 5.2: This figure shows response time of BG3IN* signal.

5.1.4.5 Program Execution Times

An oscilloscope was used to measure execution times of the program. Three ADO cycles were used, together with three read and three write cycles. Read and write cycles which were measured were maximum data transfer cycles (D32 transfer cycles). The results are provided in Table 5.6.

Table 5.6: Program execution times. For both read and write operations, only maximum data transfer cycles were considered for these tests.

VME Cycle	Time(microseconds)
ADO_A16	6
ADO_A24	7
ADO_A32	8
write_A16_D32	8
write_A24_D32	9
write_A32_D32	10
read_A16_D32	9
read_A24_D32	10
read_32_D32	12

5.2 Results Analysis

5.2.1 Mechanical Testing Analysis

These tests were successful. Board depth conforms to VMEBus specification, so is the board height. The board thickness was within 12.5% of 1.6 mm, as required by VME standard.

5.2.2 Functional Testing Analysis

Board's functionality was tested, and the results show that the board was successful in performing the following VME cycles:

- ADO cycles;
- Write cycles;
- Read cycles.

5.2.3 GUI Testing Analysis

GUI tests show that the user interface could perform the following functions:

- Sending commands to the serial port;
- Reading commands from the serial port.

This shows that GUI implementation was successful. There is one major improvement that could be made on this user interface though. The improvement is adding the endianness converter. The use of this interface requires an individual who is knowledgeable about VME standard; that it uses big endian mechanism, as opposed to conventional method of little endianness for data transfer. In order to make it more user-friendly, an endian converter is hence required to be integrated into the GUI[2]. Another improvement would be to write the stored commands into a log file instead of just storing them on the GUI window.

5.2.4 Performance Testing Analysis

5.2.4.1 Electrical Specification Testing Analysis

Voltage measurements were successful because measured voltages were found to be within 5% of 5V, as specified in the VME standard.

5.2.4.2 Short-Circuit Testing Analysis

All measured resistance values[3] were above 1 kilo-Ohm, showing that there is an open-circuit between ground and power (+5V) pins. This means that there were no short-circuits on the board. Therefore, these tests were successful.

[2] it is worth noting that this improvement was recently (after preparing this dissertation) done in version 3 of GUI, and this version is available in Appendix D

[3] for this testing

5.2.4.3 Maximum Trace Length Analysis

The maximum trace length was measured to be less than 50.88 mm, in concordance with VME specification on the maximum trace length. Hence this test was successful.

5.2.4.4 Response Time Analysis

From Figure 5.3, it takes 55 ns for AS* to be asserted, and it takes additional 55 ns for AS* to be de-asserted (thus, completing a VME cycle). As provided in Appendix D graphs, driver circuit's switching time is 20 ns, and receiver circuit's switching time is 20 ns. Overall response time is then claculated to be 150 ns[4], with the data rate of 26 MBPS.

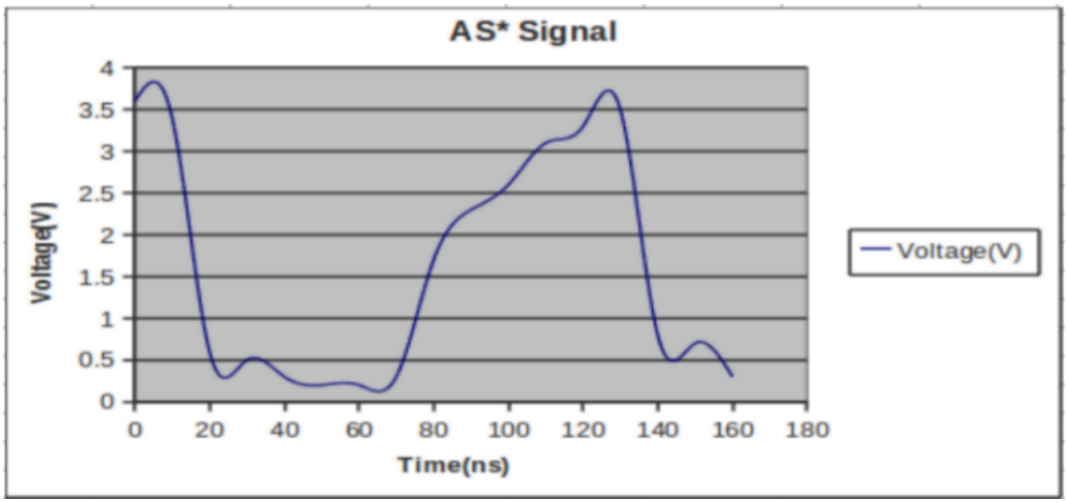

Figure 5.3: AS* signal's graph. This graph shows the time it takes for AS* to be asserted, and the time it takes for that signal to be de-asserted. AS* is an active-low signal, so it is asserted when low, and de-asserted when high. It takes 55 ns for AS* to be asserted, and additional 55 ns for it to be de-asserted.

The response times weere less than 300 nanoseconds. This means that the project was a success, in so far as response times are concerned.

5.2.4.5 Program Execution Time Analysis

Execution times for ADO and write cycles were less than (or equal to) 10 microseconds, as stipulated in the acceptance test plan. However, read cycle execution times exceeded the set threshold of 10 microseconds. This is possibly because the code for read cycles was slightly larger than that of ADO and write cycles. Also, these read cycles could not be implemented in Xilinx ISE 9.2, so they raised some unknown errors (crashed). The whole read core was then implemented in ISE 11.1.

[4] 55 ns + 55 ns + 20 ns + 20 ns

5.3 Summary

This chapter provided results for tests performed on the exerciser unit. The tests were performed based on the acceptance test plan outlined in the second chapter. The following tests were performed:

- Mechanical tests;
- Functional tests;
- GUI tests;
- Performance tests.

Eventually, results obtained were analyzed, to establish whether they were compartible with exerciser expectations or not. The next chapter provides the conclusions drawn from this project, and recommendations for future work.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The goal of this project was to design and implement a reconfigurable VMEbus exerciser. This goal was met, hence the project was successful. Based on the results obtained, the following conclusions could be drawn:

- A high-performance, robust, portable, low-cost VMEbus exerciser was developed. Data rates of up to 26 Megabytes per second were achieved by this exerciser;
- This created exerciser is capable of performing the following functions:
  - ADO operations;
  - write operations; and
  - read operations;
- A graphical user interface was built that would be used to run VMEbus commands;
- System tests were carried out, and the results show that the project was successful;
- Program execution times for some read operations did not meet the expected benchmarks. Moreover, read core crashed when ISE 9.2 was used. This prompted the use of ISE 11.1 for VME read operations.

6.2 Recommendations for Future Work

The following recommendations could be made:

- The use of ISE 11.1 or later versions for future designs;

- Introduction of the fourth Picoblaze core in VME firmware as arbitration core between three VME cores (ADO, write and read cores), and to provide I/O connections to serial port and FPGA I/O pins. This would be to reduce the overhead between VME cores and I/O circuitry. It is worth noting that as much as the introduction of this core would solve a problem, this design unfortunately introduces some sort of design complexity. This proposed firmware architecture is shown in Figure 6.1;
- Use of Python design patterns [50] to enhance GUI performance and usability;
- Implementation of endianness conversion in the user interface[1], to improve GUI user-friedliness;
- A log file creation in GUI so as to store (on the disk) all the commands that were used for VME exercising.

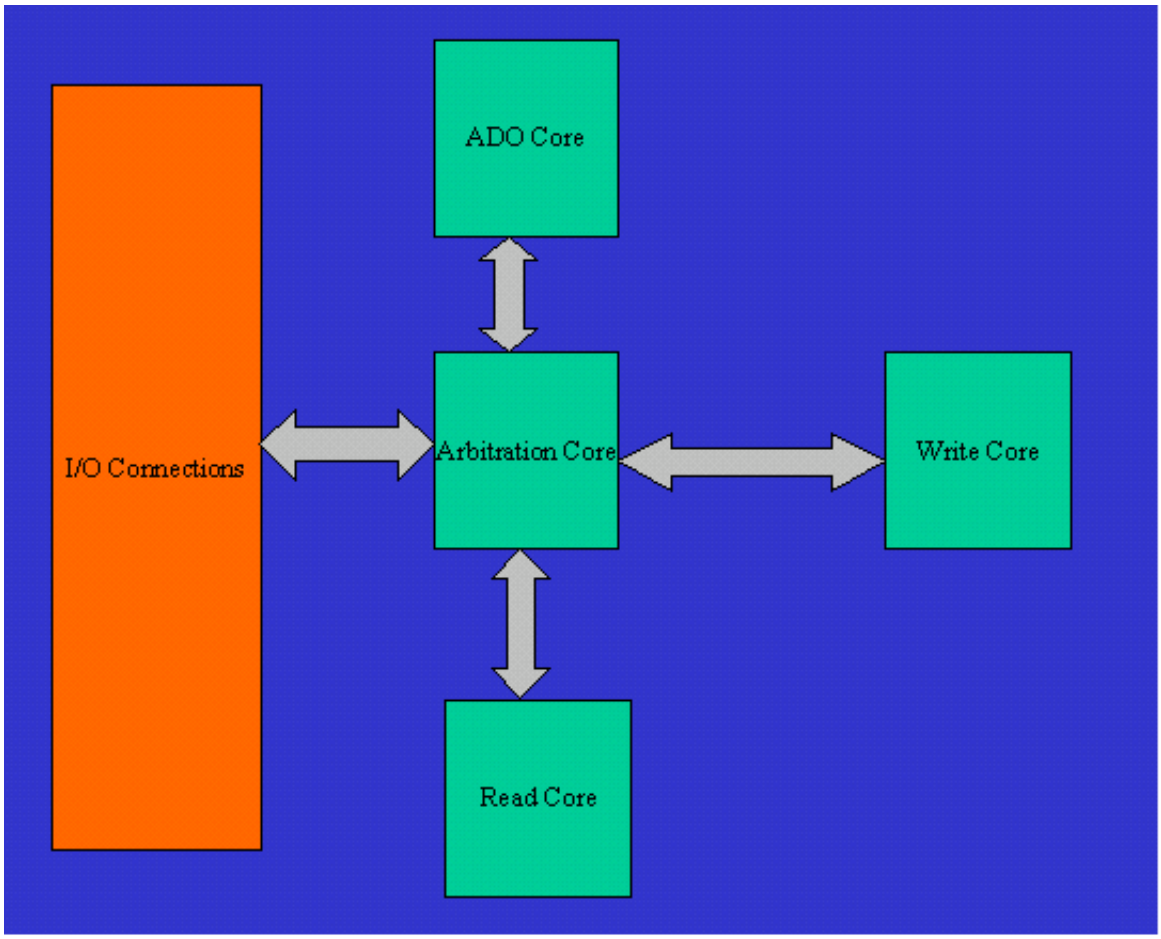

Figure 6.1: A proposed firmware architecture. Arbitration core would be created so as to reduce overhead between VME cores and I/O circuitry.

[1]this is already catered for in the recent version (version 3) of exerciser GUI. The dissertation was prepared before the GUI could be upgraded to version 3. However, as already mentioned, GUI version 3 source code is available in Appendix D

Bibliography

[1] VME Specification Manual, October 1987.

[2] IPC-2221 : Generic Standard on Printed Circuit Design, 1998.

[3] IPC-2152: Standard for Determining Current Carrying Capacity in Board Design, August 2009.

[4] Allison, T. and Flood, R. Versatile Data Acquisition and Controls For EPICS Using VME-Based FPGAs. *CoRR*, cs.AR/0111029:1–4, 2001.

[5] A. Aloisio, P. Branchini, and Cevenini F. Timing Analysis of Asynchronous Block transfer Cycles on VME and VME64x Physical Layer. *IEEE Transactions on Nuclear Science*, 51(3):401–406, June 2004.

[6] M. Balch. *Complete Digital Design*. McGraw-Hill, 2003.

[7] J. Black. *The System Engineer's Handbook*. Associated Press, 1992.

[8] K. Chapman. KCPSM3: 8-Bit Microcontroller for Spartan-3, Virtex-II and VirtexIIPRO. Technical report, Xilinx inc, 2003.

[9] K. Chapman. UART Transmitter and Receiver Macros. Technical report, Xilinx inc, 2003.

[10] P. Chu. *FPGA Prototyping by VHDL Examples*. Wiley, 2008.

[11] C. Coombs. *Printed Circuits Handbook*. McGraw-Hill, 6th edition, 2008.

[12] I. Grout. *Digital Systems Design with FPGAs and CPLDs*. Elsevier, 2008.

[13] Y. Guo. Picoblaze-based Self-organizing Learning Array and Its Experimental Setting. Master's thesis, Ohio University, November 2004.

[14] http://japan.xilinx.com/support/documentation/ipcores.html. Spartan 3A Starter Kit Board User Guide. Technical report, Xilinx inc, 2006.

[15] http://wikipedia.org/wiki/printed-circuit board. Printed Circuit Board. Technical report, Wikipedia, 2010.

[16] http://www.cs.usfca.edu/ afedosov/qttut. PyQt Tutorial. Technical report, University of San Francisco, 2002.

[17] http://www.electronicdesign.com. Ideas for Design: Electronic Design. Technical report, Electronic Design, 2009.

[18] http://www.micrel.com:8000/iphrase/query?query=pcb+design&Image25.x=0&Image25.y=0&Image25= PCB Design Considerations for Ethernet Controllers. Technical report, Micrel, 2009.

[19] http://www.vectorelect.com/catpdf/page72A.pdf. VME Connector Pin Assignment and Signal Descriptions Under VME64X. Technical report, Vectorelect, 2008.

[20] http://www.vita.com/vmefaq.html. VMEbus Frequently Asked Questions. Technical report, VITA, 2000.

[21] http://www.xilinx.com/support/documentation/data sheets/ds099.pdf. DS099 Spartan-3 FPGA Family Data Sheet. Technical report, Xilinx inc, 2007.

[22] http://www.xilinx.com/support/documentation/ip documentation/ug129.pdf. Picoblaze 8-bit Embedded Microcontroller User Guide. Technical report, Xilinx, 2010.

[23] http://xilinx.com/products/design-resources/proc-central/microblaze faq.pdf. Microblaze Soft Processor V 7.20 Frequently Asked Questions. Technical report, Xilinc inc, 2009.

[24] iThemba LABS. www.tlabs.ac.za. Technical report, iThemba LABS, 2010.

[25] B. J. Jansen. The Graphical User Interface: An Introduction. *SIGCHI Bulletin*, 30(2):22–26, 1998.

[26] H. Johnson and H. Graham. *High-speed Digital Design : A Handbook of Black Magic*. Prentice Hall, 1993.

[27] R. Kamat. *Unleash the System-on-chip Using HandelC*. Springer-Verlag, 2009.

[28] C. Maxfield. *The Design Warrior's Guide to FPGAs*. Elsevier, 2004.

[29] P. McCgaugh and K. Khan. VHDL Synthesis for Space Applications - Implementing VME Core in Actel RadHard Devices Using Actmap VHDL. Technical report, Actel Corporation, 1998.

[30] J. Olson. Determining Current Carrying Capacity : Understanding the New Standards for Predicting Temperature Rise. Technical report, Printed Circuit Design and FAB/Circuit Assembly, December 2009.

[31] W. Peterson. *The VMEbus Handbook*. VITA, 4th edition, 1997.

[32] M. Rapacki. *Soft-core Processors as SOC Prototyping Solution for Cryptographic Application*. PhD thesis, Politechnika Gdanska, 2009.

[33] B. Rempt. *GUI Programming With Python : Qt Edition*. Open Source Publisher, 2001.

[34] Kenneth G. Ricks, David J. Jackson, and William A. Stapleton. An evaluation of the VME architecture for use in embedded systems education. *SIGBED Rev.*, 2(4):63–69, 2005.

[35] T. Skaali. Embedded Systems and VMEbus. Technical report, Department of Physics, University of Oslo, 2007.

[36] J. Song, K.E Hoover, and E Wheeler. Effectiveness of PCB Simulation in Teaching High-speed Digital Design. In *IEEE Transactions on Education*, 2008.

[37] B. Stahlin. *Electronic Instrument Handbook*. McGraw-Hill, 2004.

[38] M. Summerfield. *Rapid GUI Programming with Python and Qt*. Prentice Hall, 2008.

[39] M. Summerfield. *Professional Programming in Python 3 : A Complete Introduction to the Python Language*. Addison-Wessley, 2009.

[40] J. Thelin. GUI Development for Advanced Software. Technical report, Integrated Computer Solutions, 2005.

[41] J. Tong, I. Anderson, and M. Khalid. Soft-core Processors for Embedded Systems. In *18th International Conference on Microelectronics (ICM)*, pages 170–173, 2006.

[42] M. Ts'oeu. Proton Beam Steering Control System for High Precision Radio Therapy at iThemba LABS : An Investigation on Actuator Saturation Contraints. Master's thesis, University of Cape Town, 2008.

[43] J. Vartesian. *Fabricating Printed Circuit Boards*. Elsevier, 2002.

[44] www.britannica.com/EBchecked/topic/242033/graphical-user interface. Graphical User Interface(GUI). Technical report, Britannica, 2008.

[45] www.digilentinc.com/products/detail.cfm?prod=s3board. S3Board: Digital Design Engineer's Source. Technical report, Digilent.inc, 2009.

[46] www.mediatronix.com. pBlaze IDE. Technical report, Mediatronix, 2006.

[47] www.riverbankcomputing.com. PyQt Whitepaper. Technical report, Riverbank Computing, 2010.

[48] www.wiki.python.org/moin/pyqt. PyQt- Python Info Wiki. Technical report, Python Community, 2010.

[49] www.zeidman.net/. Introduction to VHDL. Technical report, Chalkboard Network, 2005.

[50] T. Ziade. *Expert Python Programming*. Packt Publishers, 2008.

Appendix A

VMEbus Standard

A.1 VMEbus Background

VMEbus is a computing systems architecture which consists of electrical specifications for databus and mechanical specifications describing the backplane, bus connector, board sizes and enclosures [35]. The standard was conceived with the following objectives [1]:

- To allow communication between devices on VMEbus without disturbing the internal activities of the bus;
- To specify electrical and mechanical system characteristics required to design devices that would reliably and unambigously communicate with other devises interfaced to the VMEbus;
- to specify protocols that precisely define the interaction between the VMEbus and the devices interfaced to it
- To provide terminology and definitions that describe system protocols;
- To allow a broad range of design latitude so that the designer could optimise cost and/or performance without affecting system compartibility;
- To provide a system where performance is primarily device-limited, rather than system interface limited.

A.2 VMEbus Description

VMEbus is a flexible, open-ended computer bus system which is based on Versa module[1] and Eurocard standard[2]. This asynchronous, high-performance, TTL-based bus was created to provide a

[1] for electrical characteristics
[2] for mechanical dimensions

flexible environment supporting intensive tasks, and to help engineers standardize their computer system designs [37]. It is defined by the IEEE-1014-1987 standard with the following features [20]:

- Master/slave architecture;
- Asynchronous bus;
- Variable speed handshaking protocol;
- Non-multiplexed bus;
- Addressing range between 16 and 32 bits;
- Data-path widths of between 8 and 32 bits;
- Bandwidths of up to 40 Megabytes per second;
- Multiprocessing capability;
- Wide variety of mechanical hardware based on IEEE 1101 standard[3];
- Up to 21 card slots can be used in a single chasis.

VMEbus architecture is generally described using the concept of modules [20]. A functional module is a collection of electronic circuitry which reside on a VMEbus board and work together to accomplish a task [1]. VMEbus functional modules are given in Figure A.1. VMEbus cards are formed by including at least one of each type of functional modules in board design. Main types of VME cards are:

- VMEBus system controller: this card provides arbitration and monitors system's state. Only one controller may reside on the VMEbus, and it resides on slot one of VME chasis;
- VMEbus master: it initiates data transfer transactions (read, write, interrupt acknowledge, address broadcast and block transfer). Any number of bus masters may reside on the bus, but only one may have control of the bus at any time;
- VMEbus slave: it responds to read and write transactions within it memory window. Any number of slave boards may reside on the bus.

VMEbus chasis consists of a card cage with1-21 slots, a backplane with two connectors and, normallywith five jumpers per slot.

VMEbus functional modules communicate with each other through conceptual tools known as VME sub-buses. These tools make VMEbus flexible, modular and easily upgradeable (scalable). VME sub-buses include:

[3] this standard is known as Eurocard form factor [17]

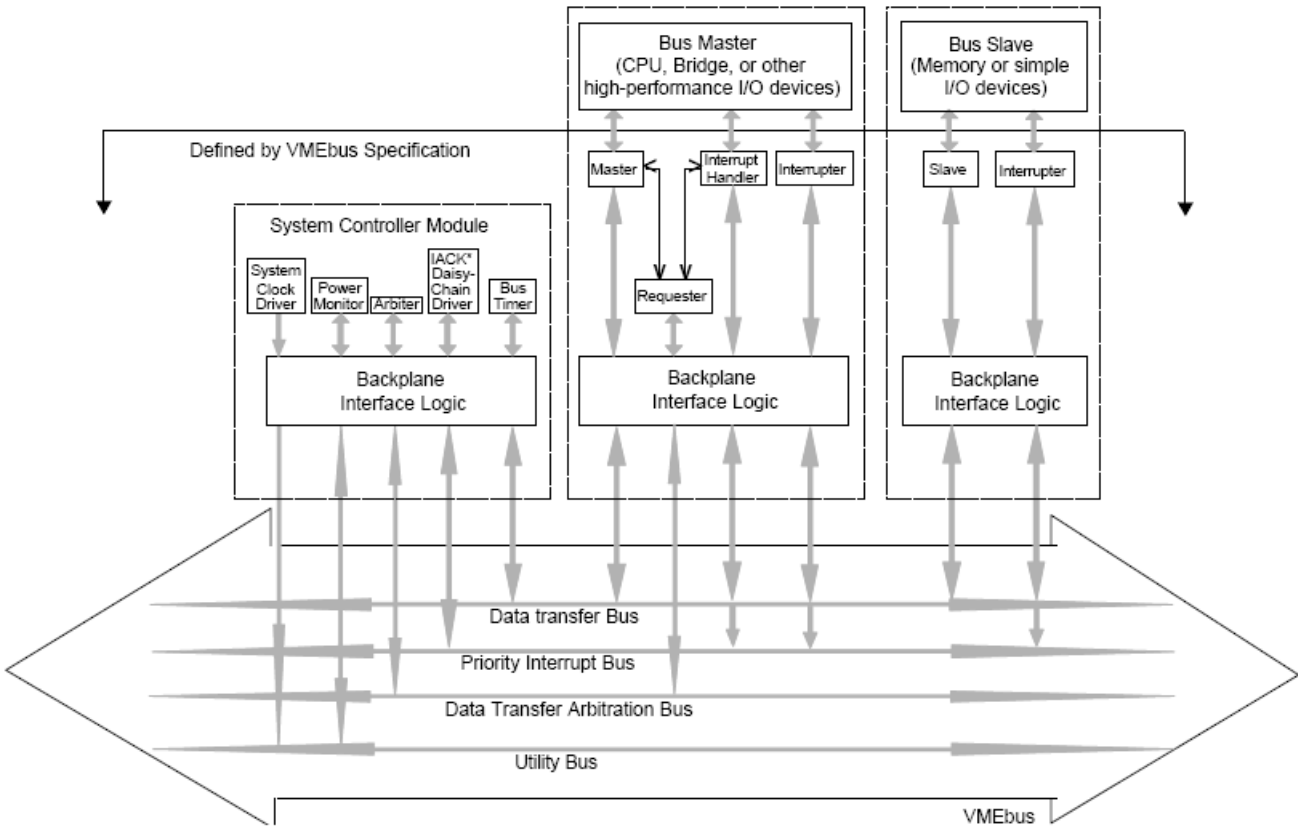

Figure A.1: This Figure shows VMEbus system elements [1].

- Data transfer bus (DTB): it is used to move data to and from the slaves [31]. This sub-bus is composed of address lines, data lines and control lines[4]. Master drives address strobe (AS*)[5] to indicate a valid address on the bus. Data lines are used to transfer information across the bus. Transfer size (8-bit, 16-bit or 32-bit) is determined by the state of LWORD*, DS0*, DS1* and A01 lines. Direction of the transfer is determined by WRITE* line's state. Control lines determine bus' timing (that is, when it starts and stops, and when data is valid and ready to be accepted) [37]. Table A.1 shows a tabular form of these DTB signals;

- Data transfer arbitration bus: it is used by master modules and interrupt handlers to obtain bus ownership from the system controller. A functional module called arbiter is used in conjunction with DTB to determine which master is granted the bus [31]. A module controlling the bus (master module) would drive BBSY* low to show that the bus is in use. When this line (BBSY*) is not low, the arbiter module would sample the bus request lines (BR0*-BR3*) looking for a pending action. BR3* has the highest priority. Requests of equal priority are handled by a daisy-chain using bus grant-in lines (BG0IN*-BG3IN*) and bus grant-out lines (BG0OUT*-BG3OUT*). These signals are shown in Table A.2;

- Priority interrupt bus: it is used to assert an interrupt. It is also used in conjunction with DTB to acknowledge an interrupt. It consists of interrupt request lines (IRQ7*-IRQ1*), with IRQ7* having the highest priority), IACK* line and IACK* daisy chain lines (IACKIN* and IACKOUT*)[6]. These (priority interrupt) bus signals are shown in Table A.3;

- Utility bus: this is a collection of signals used for system reset, periodic updates, timing, system diagnosis and power failures [31]. Utility bus signals include SYSCLK, SERCLK, SEDRAT*, SYSFAIL and ACFAIL*;

Table A.1: DTB Signals.

Address Lines	Data Lines	Control Lines
A01-A31	D00-D31	AS*
AM0-AM5		DS0*
DS0*		DS1*
DS1*		BERR*
LWORD*		DTACK*
		WRITE*

[4]additional information on VMEbus signals can be found in [1], [31], [19] and [35]
[5]VME signals that end with an asterik are active low signals
[6]the reason for using daisy chain in VME is given in [37]

Table A.2: Arbitration Signals.

Daisy-Chained Signals	Common Signals
BG0OUT* to BG0IN*	BBSY*
BG1OUT* to BG1IN*	BCLR*
BG2OUT* to BG2IN*	BR0*
BG3OUT* to BG3IN*	BR1*
	BR2*
	BR3*

Table A.3: Priority Interrupt Signals.

Signal Line(s)	Common Signals
IRQ1*	lowest priority
IRQ2* to IRQ6*	normal priority
IRQ7*	highest priority
IACK*	bussed acknowledgement
IACKOUT* to IACKIN*	daisy-chained acknowledgement

A.3 VMEbus Operation

Typical VMEbus operation includes bus request, data transfer and bus release. Data transfer cycles include:

- Read cycle: transfer of data from slave to master in 1, 2, 3 or 4 bytes;
- Write cycle: transfer of data from master to slave in 1, 2, 3 or 4 bytes;
- block read cycle: transfer of data from slave to master (up to 256 bytes could be transferred per transaction);
- block write cycle: transfer of data from master to slave (up to 256 bytes could be transferred in a single transaction);
- Read-modify-write cycle: it is used to both read from and write to a slave location without permitting any other master to access the location;
- Address-only (ADO) cycle: consists of address broadcast only; without any data transfer;
- interrupt acknowledge cycle: it is initiated by interrupt handler and it reads a STATUS/ID from an interrupter. Interrupt handler generates this cycles whenever it detects an interrupt request from the bus and it has the control of DTB;

In order to fulfill user requirements, this project only concerned with three single cycle transfer (SCT) cycles, namely: ADO, write and read cycles.

A.3.1 ADO Cycle

During ADO cycle, the following steps are followed:

1. Present address;
2. Present address modifier;
3. Drive LWORD* to a valid state;
4. Wait for set-up time, then;
5. Drive AS* low;
6. Wait for slave's response (either of DTACK* or BERR* asserted);
7. Drive DS0* and DS1* high.

A.3.2 Write Cycle

The following steps are followed, in order to perform write operation (cycle):

1. Present address;
2. Present address modifier;
3. Drive LWORD* to a valid state;
4. Drive IACK* high;
5. Wait for set-up time, then;
6. Drive AS* low;
7. Drive WRITE* low;
8. Wait until DTACK* and BERR* are high;
9. Place data on the bus;
10. Drive DS0* and DS1* to their valid states;
11. Receive DTACK* low;
12. Drive DS0* and DS1* high;
13. Drive AS* high.

A.3.3 Read Cycle:

During read cycle, the following procedure is followed:

1. Present address;
2. Present address modifier;
3. Drive LWORD* to a valid state;
4. Drive IACK* high;
5. Wait for set-up time, then;
6. Drive AS* low;
7. Drive WRITE* high;
8. Wait until DTACK* and BERR* are high;
9. Drive DS0* and DS1* to their valid states;
10. Receive DTACK* low;
11. Wait for 25 nanoseconds, then;
12. Receive data on data lines;
13. Drive DS0* and DS1* high;
14. Drive AS* high

Appendix B

GUI Top-Level Source Code

```
import sys
#import pickle #for pickle reading and writing of data types
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import serial
from ui_mainwindow import Ui_MainWindow
ui = Ui_MainWindow()
#ui.setupUi(self)
class TopLevel(QMainWindow):
def __init__(self, parent=None):
QWidget.__init__(self, parent)
#ui = Ui_MainWindow()
ui.setupUi(self)
self.setWindowTitle('VME Exerciser')
# signals and slots
self.connect(ui.actionConnect, SIGNAL("triggered()"), self.connexn)
self.connect(ui.actionDisconnect, SIGNAL("triggered()"), self.disconnexn)
self.connect(ui.actionAbout_Qt, SIGNAL("triggered()"), self.tsa_qt)
self.connect(ui.actionDocumentation, SIGNAL("triggered()"), self.documenta)
self.connect(ui.actionExit, SIGNAL("triggered()"), self.close)
self.connect(ui.headercheckBox, SIGNAL("clicked(bool)"), self.send_Header)
self.connect(ui.amcheckBox, SIGNAL("clicked(bool)"), self.send_Am)
self.connect(ui.address0checkBox, SIGNAL("clicked(bool)"), self.send_Address0)
```

```
self.connect(ui.address1checkBox, SIGNAL("clicked(bool)"), self.send_Address1)
self.connect(ui.address2checkBox, SIGNAL("clicked(bool)"), self.send_Address2)
self.connect(ui.address3checkBox, SIGNAL("clicked(bool)"), self.send_Address3)
self.connect(ui.ra08radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.ra16radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.ra24radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.ra32radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.rd08radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.rd16radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.rd24radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.rd32radioButton, SIGNAL("toggled(bool)"), self.reada)
self.connect(ui.sendpushButton, SIGNAL("clicked()"), self.datasend)
#self.connect(ui.data0checkBox, SIGNAL("clicked(bool)"), self.send_Data0)
#self.connect(ui.data1checkBox, SIGNAL("clicked(bool)"), self.send_Data1)
#self.connect(ui.data2checkBox, SIGNAL("clicked(bool)"), self.send_Data2)
#self.connect(ui.data3checkBox, SIGNAL("clicked(bool)"), self.send_Data3)
self.connect(ui.endianconvertpushButton, SIGNAL("clicked()"), self.endiannes)
def connexn(self):
#if bool :
#ui = Ui_MainWindow()
#ui.setupUi(self)
ui.headerLineEdit.setEnabled(True)
ui.amLineEdit.setEnabled(True)
ui.address0LineEdit.setEnabled(True)
ui.address1LineEdit.setEnabled(True)
ui.address2LineEdit.setEnabled(True)
ui.address3LineEdit.setEnabled(True)
ui.data0LineEdit.setEnabled(True)
ui.data1LineEdit.setEnabled(True)
ui.data2LineEdit.setEnabled(True)
ui.data3LineEdit.setEnabled(True)
ui.headercheckBox.setEnabled(True)
ui.amcheckBox.setEnabled(True)
```

```
ui.address0checkBox.setEnabled(True)
ui.address1checkBox.setEnabled(True)
ui.address2checkBox.setEnabled(True)
ui.address3checkBox.setEnabled(True)
ui.wd08checkBox.setEnabled(True)
ui.wd16checkBox.setEnabled(True)
ui.wd24checkBox.setEnabled(True)
ui.wd32checkBox.setEnabled(True)
ui.ra08radioButton.setEnabled(True)
ui.ra16radioButton.setEnabled(True)
ui.ra24radioButton.setEnabled(True)
ui.ra32radioButton.setEnabled(True)
ui.rd08radioButton.setEnabled(True)
ui.rd16radioButton.setEnabled(True)
ui.rd24radioButton.setEnabled(True)
ui.rd32radioButton.setEnabled(True)
#ui.setupUi(self)
#ui.amLineEdit.setEnabled(False)
conn = "connection established"
print(conn)
#ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout =
5000 )
#ser.open()
return
def disconnexn(self):
if bool :
#ui = Ui_MainWindow()
#ui.setupUi(self)
ui.headerLineEdit.setEnabled(False)
ui.amLineEdit.setEnabled(False)
ui.address0LineEdit.setEnabled(False)
ui.address1LineEdit.setEnabled(False)
ui.address2LineEdit.setEnabled(False)
```

```
ui.address3LineEdit.setEnabled(False)
ui.data0LineEdit.setEnabled(False)
ui.data1LineEdit.setEnabled(False)
ui.data2LineEdit.setEnabled(False)
ui.data3LineEdit.setEnabled(False)
ui.headercheckBox.setEnabled(False)
ui.amcheckBox.setEnabled(False)
ui.address0checkBox.setEnabled(False)
ui.address1checkBox.setEnabled(False)
ui.address2checkBox.setEnabled(False)
ui.address3checkBox.setEnabled(False)
ui.wd08checkBox.setEnabled(False)
ui.wd16checkBox.setEnabled(False)
ui.wd24checkBox.setEnabled(False)
ui.wd32checkBox.setEnabled(False)
ui.ra08radioButton.setEnabled(False)
ui.ra16radioButton.setEnabled(False)
ui.ra24radioButton.setEnabled(False)
ui.ra32radioButton.setEnabled(False)
ui.rd08radioButton.setEnabled(False)
ui.rd16radioButton.setEnabled(False)
ui.rd24radioButton.setEnabled(False)
ui.rd32radioButton.setEnabled(False)
disconn = "Connection terminated "
print(disconn)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 5000
)
ser.close()
#ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout =
5000 )
ser.close()
return
def tsa_qt(self):
```

```
print(" For Qt info, visit : www.qtnokia.com ")
return
def documenta(self):
print("For documentation, consult the manual provided")
def send_Header(self, bool):
#
#
header = "Romela Hlooho"
if bool:
#ui = Ui_MainWindow()
#ui.setupUi(self)
print(header)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
#ui.commandheader.txt = ui.headerLineEdit
#x = ser.open(ui.headerLineEdit, mode = 'wt')
x = ui.headerLineEdit.text()
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return()
def send_Am(self, bool):
am = "Romela AM"
if bool:
print(am)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
x = ui.amLineEdit.text()
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return
def send_Address0(self, bool):
address0 = "Romela Addres0"
if bool:
```

```
print(address0)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
x = ui.address0LineEdit.text()
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return
def send_Address1(self, bool):
address1 = "Romela Addres1"
if bool:
print(address1)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
x = ui.address1LineEdit.text()
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return
def send_Address2(self, bool):
address2 = "Romela Addres2"
if bool:
print(address2)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
x = ui.address2LineEdit.text()
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return
def send_Address3(self, bool):
address3 = "Romela Addres3"
if bool:
print(address3)
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
x = ui.address3LineEdit.text()
```

```
y = ser.write(x.encode('ascii'))
print(y)
ser.close()
return
def reada(self, bool):
if bool:
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 1 )
x = ser.read()
if ui.ra08radioButton.isChecked():
print('A08 is', x)
elif ui.ra16radioButton.isChecked():
print('A16 is', x)
elif ui.ra24radioButton.isChecked():
print('A24 is', x)
elif ui.ra32radioButton.isChecked():
print('A24 is', x)
elif ui.rd08radioButton.isChecked():
print('D08 is', x)
elif ui.rd16radioButton.isChecked():
print('D16 is', x)
elif ui.rd24radioButton.isChecked():
print('D24 is', x)
elif ui.rd32radioButton.isChecked():
print('D32 is', x)
else:
pass
ser.close()
return
def datasend(self):
if bool:
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
if ui.wd08checkBox.isChecked():
x = ui.data0LineEdit.text()
```

```
y = ser.write(x.encode('ascii'))
print('D08 Write is', y)
elif ui.wd16checkBox.isChecked():
x = ui.data1LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D16 Write is', y)
elif ui.wd24checkBox.isChecked():
x = ui.data2LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D24 Write is', y)
elif ui.wd32checkBox.isChecked():
x = ui.data3LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D32 Write is', y)
else:
pass
ser.close()
return
def endiannes(self):
if bool:
ser = serial.Serial('COM1', baudrate = 9600, bytesize = 8, parity = 'N', stopbits = 1, timeout = 0 )
if ui.wd32checkBox.isChecked():
x = ui.data3LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D32 Write is', y)
if ui.wd24checkBox.isChecked():
x = ui.data2LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D24 Write is', y)
if ui.wd16checkBox.isChecked():
x = ui.data1LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D16 Write is', y)
```

```
if ui.wd08checkBox.isChecked():
x = ui.data0LineEdit.text()
y = ser.write(x.encode('ascii'))
print('D08 Write is', y)
else:
pass
ser.close()
return
if __name__ == "__main__":
app = QApplication(sys.argv)
myapp = TopLevel()
myapp.show()
sys.exit(app.exec_())
```

Appendix C

Design Errata

The following design modifications should be performed on the later versions of VMEbus Exerciser:

- BBSY* physical mapping: this VMEbus signal was not physically mapped to FPGA pins during user constraint file (UCF) creation. It is only significant in multi-processor networks, not when a single VME module is exercised
- use of 'Read' button on the interface main window for reading data from the serial port[1]

removal of 'quit' button in GUI mainwindow. This is because this same PyQt slot of 'quit' with 'close' button and 'exit' File submenu, resulting in unnecessary redundancy[2].

[1] this is already taken care of in VME GUI version 3

[2] this design modification is implemented in exerciser GUI version 3, included in Appendix D

Appendix D

Resources Used

This is included in the compact disc (CD) which is submitted together with this dissertation.