VME SLAVE IMPLEMENTATION ON FPGA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TOLGA ZORER

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

NOVEMBER 2008

Approval of the thesis:

VME SLAVE IMPLEMENTATION ON FPGA

submitted by **TOLGA ZORER** in partial fulfillment of the requirements for the degree of **Master of Science** in **Electrical and Electronics Engineering Department**, **Middle East Technical University** by,

Prof. Dr. Canan ÖZGEN
Dean, **Graduate School of Natural and Applied Sciences** __________

Prof. Dr. İsmet ERKMEN
Head of Department, **Electrical and Electronics Engineering** __________

Prof. Dr. Murat AŞKAR
Supervisor, **Electrical and Electronics Engineering Dept., METU** __________

Examining Committee Members:

Prof. Dr. Hasan GÜRAN
Electrical and Electronics Engineering Dept., METU __________________

Prof. Dr. Murat AŞKAR
Electrical and Electronics Engineering Dept., METU __________________

Assist. Prof. Dr. Cüneyt BAZLAMAÇCI
Electrical and Electronics Engineering Dept., METU __________________

Assist. Prof. Dr. Şenan Ece SCHMIDT
Electrical and Electronics Engineering Dept., METU __________________

Dr. Selim EMİNOĞLU
Micro and Nanotechnology Department, METU __________________

Date: 11.11.2008

ABSTRACT

VME SLAVE IMPLEMENTATION ON FPGA

ZORER, Tolga
M. Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Murat AŞKAR

November 2008, 134 pages

In today's complex technological systems, there is a need of multi tasking several units running in accordance. Each unit is composed of several intelligent microcontroller cards. Each intelligent card performs a different task that the unit is responsible of. For this reason, there is a need of common communication bus between these cards in order to accomplish the task duties. VME (Versa Module Euro-Card) bus is a well known, the most reliable and the commonly used communication bus, even if it was standardized three decades ago. In this thesis work, the world wide accepted VME parallel bus protocol is implemented on FPGA (Field programmable Gate Array). The implementation covers the VME standard slave protocols. The VME Slave Module has been developed by VHDL (Very high level Hardware Description Language). The simulations have been carried over a computer based environment. After the verification of the VHDL code, an Intellectual Property (IP) core is synthesized and loaded into the FPGA. The FPGA based printed circuit board has been designed and the IP core's function has been tested by bus protocol checkers for all of its functionality. The designed hardware has several standard serial communication ports, such as; USB, UART and I2C. Through the developed card and the add-on units, it is also possible to communicate with these serial ports over the VME bus.

Keywords: VME SLAVE, VHDL, FPGA, USB, UART, I2C

ÖZ

FPGA ÜZERİNDE "VME SLAVE" GERÇEKLEŞTİRİLMESİ

ZORER, Tolga
Yüksek Lisans, Elektrik Elektronik Mühendisligi Bölümü
Tez Yöneticisi: Prof. Dr. Murat AŞKAR

Kasım 2008, 134 sayfa

Günümüzün karmaşık teknolojik sistemlerine, birden çok görevi yerine getiren birimlerin uyum içerisinde çalışmaları gerekmektedir. Her bir birim, birden fazla mikrokontrolör karttan oluşmaktadır. Her bir akıllı kart, birimin yükümlü olduğu farklı görevleri yerine getirmektedir. Bu nedenden dolayı; mikrokontrollör kartlarının üzerinde haberleşerek görev gerekliliklerini yerine getirebilecekleri ortak bir haberleşme veriyoluna ihtiyaç vardır. VME (Versa module Euro-Card) veriyolu otuz yıl önce standart haline getirilmiş olsa da, en iyi bilinen, en çok güvenilirliği olan ve en çok kullanılan haberleşme veriyoludur. Bu tez çalışmasında, dünya çapında kabul görmüş olan VME paralel veriyolu, Alan Programlanabilir Kapı Dizisi (FPGA) üzerinde gerçekleştirilmiştir. Gerçeklenme VME standardı "Slave" protokolünü kapsar. VME "Slave" birimi VHDL (Çok yüksek seviyede Donanım Tanımlama Dili) ile geliştirilmiştir. Simulasyonlar bilgisayar tabanlı ortamda yapılmıştır. VHDL kodu doğrulandıktan sonra, Akıllı program sentezlenir ve FPGA'ya yüklenir. FPGA tabanlı Baskı Devre Kartı tasarlanmıştır ve Akıllı Program, veriyolu kontrolörleri tarafından, Akıllı Programın tüm fonksiyonları için fonksiyonel testler ile kontrol edilmiştir. Tasarlanmış olan donanım standard seri haberleşme veriyollarını içerir, bunlar; USB, UART, I2C'dir. Tasarlanmış kart ve ek kart üzerinden, VME veriyolu ile bu seri kanallar arasında iletişim kurmak mümkündür.

Anahtar Kelimeler: VME SLAVE, VHDL, FPGA, USB, UART, I2C

To My Family

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Prof. Dr. Murat AŞKAR for his encouragements, guidance, advice, criticism, and insight throughout the research.

I would like to thank to ASELSAN Inc. for facilities provided for the completion of this thesis.

I would like to forward my appreciation to all my friends and colleagues who contributed to my thesis with their continuous support.

This thesis is dedicated to my father Mehmet, my mother Birgül and my sister Fulya, without whom none of this would have been even possible. I would also like to express my profound appreciation for their continuous support, encouragement and love throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	: Application Specific Integrated Circuit
ANSI	: American National Standards Institute
BLT	: Block Transfer
CPLD	: Complex Programmable Logic Device
DTB	: Data Transfer Bus
FPGA	: Field Programmable Gate Array
I2C	: Inter Integrated Circuit Bus
IACK	: Interrupt Acknowledge
IDE	: Integrated Design Environment
IEEE	: Institute of Electrical and Electronics Engineers
IP	: Intellectual Property
ISE	: Integrated Software Environment
JTAG	: Joint Test Action Group
MTBF	: Mean Time Between Failure
PAL	: Programmable Array Logic
PCB	: Printed Circuit Board
PLA	: Programmable Logic Array
PROM	: Programmable Read Only Memory
RAM	: Random Access Memory
RMW	: Read Modify Write
ROAK	: Release On Acknowledge
ROM	: Read Only Memory
RORA	: Release On Register Access
SCT	: Single Cycle Transfer
UART	: Universal Asynchronous Receiver Transmitter
USB	: Universal Serial Bus
VHDL	: Very High Level Hardware Description Language
VME	: Versa Module Eurocard

CHAPTER 1

INTRODUCTION

VME (Versa Module Eurocard) Bus architecture has been developed nearly three decades ago. It was proposed as a general microcomputer bus, since microcomputers were very popular at that time. There were several attempts to define universal microcomputer bus that many manufacturers could design products that are compatible to this universal bus and their products can communicate with each other on a defined universal bus. Unfortunately those attempts failed until the Motorola engineers defined the VME Bus architecture for 68000 CPU series [1]. The newly proposed bus had many advantages to the other ones, and it was easily accepted by several other companies including Signetics, Philips, Thomson and Mostek. Although microcomputers had limited capabilities, such as the maximum data path width was limited to 16 bits at those years, the proposed VME Bus architecture could be easily upgraded to 32 bit address and data widths. Moreover, the bus structure is not limited to any global clocks defined for bus, on the contrary the residents communicate asynchronously. Thus they were free to communicate as fast as they could.

The VME bus structure was first standardized as IEEE standard for a versatile backplane bus: VME Bus 1987 [2]. Then, ANSI/VITA released new standards on VME as [3] "ANSI/VITA 1-1994 American National Standard for VME 64", 1994 and [4] "ANSI/VITA 1-1997 American National Standard for VME 64X", 1997.

After 1997, there were new improvements and standardization still continues on VME Bus structure nowadays.

At early years of VME Bus, the microcomputers were directly interfacing with VME Bus; however this bus structure limited the speed performance of microcomputer busses because of the other residents of Bus. Afterwards, several ASIC (Application specific Integrated Circuit) solutions for interfacing with VME Bus were developed. They have capabilities to interface the microcomputer with the VME Bus, but the slave residents of VME Bus still need an intelligent interface for VME Bus.

While years were passing, the reliability and redundancy of VME Bus was proved. Therefore, the VME Bus became the number one universal bus for military, aerospace, communication, industrial, nuclear and control applications [5]. The military, aerospace, communication or industrial areas could afford the expensive VME bridge components since they have big budgets for developing structures. However, a great amount of time is spent in order to develop experimental setups in nuclear or control applications. For instance, in nuclear science experiments there is a need for global triggers or detectors. Every laboratory has to develop own customized VME communication structures. [6] - [11]. Not only for nuclear are but also for control area is lack of custom VME communication solutions. [12] - [15]

Through 2000's, new trend called programmable logic started to replace the old ASIC and dedicated circuit solutions. The new trend brings reprogrammable logic circuit IC's (Integrated circuit) for applications. This really reduced time-to-market and decreased the prototyping costs since the designer could develop program at any step of the design process without any PCB (Printed Circuit Board) changes.

The programmable logic is a very ideal solution for developing low cost, reliable and custom applications. Thus, several academic research projects were developed as a VME slave application on FPGA's [16],[17],[18]. Moreover, there are several solutions arrived to the market [19] - [25].

In this thesis work, a VME Slave Module was developed using VHDL (Very high level Hardware Description language) coding style and a VME Slave Card was implemented for tests. Previously proposed and market available solutions were analyzed in detail and the open items in those solutions were covered. Moreover, new design techniques in VHDL coding of VME Slave Module application are proposed to increase the redundancy and performance of the VME Slave Module implementation running on FPGA. Furthermore, the developed VHDL code is not a vendor specific core. Therefore, the designed VME slave module can be synthesized and function on any vendor's FPGA.

The designed VME Slave Card has ability more than just communicating via VME bus. The FPGA on card are connected to a peripheral USB (Universal Serial Bus) controller via PMC connectors, communicates with a temperature sensor on I2C (Inter Integrated circuit Bus) bus and communicates with an RS-485 serial controller via UART (Universal Asynchronous Receiver Transmitter). Thus, the FPGA can work as a bridge between VME bus and above mentioned serial ports. Therefore, the designed FPGA code is responsible of bridging communication between VME bus and mentioned serial ports: USB, RS-485 and I2C.

Following chapters give more details on the thesis work. In Chapter 2, background of the thesis can be found. In chapter 3 the implemented VME Slave Module and in Chapter 4 the VME Slave card will be explained in detail. Then, in Chapter 5, the simulation results will be discussed and finally, in Chapter 6 the summary and conclusion will be made.

CHAPTER 2

BACKGROUND

2.1 PROGRAMMABLE LOGIC

2.1.1 History of Programmable Logic

Before 1970's, the printed circuit boards were using standard logic devices. These standard logic devices have complex interconnects in order to build complex algorithms. Moreover, any change in the logic implementation would cause new layout for printed circuit board (PCB). Afterwards, a solution was found for that problem: Read-Only Memory (ROM) chips were used for combinational logic functions [1]. As the usual functionality of ROM, the inputs (the address lines) results the outputs (the data lines). Now, consider the inputs (the address lines) as independent logic signals and the outputs (the data lines) are as the "Boolean Functions" of inputs stored previously in ROM. Thus, the most general-purpose combinatorial logic device was implemented. On the other hand, there were several disadvantages of this type logic implementation. First of all, memory devices are slower than standard logic elements. Secondly, PROM's outputs may glitch by changing input signals, so the outputs are not reliable for critical applications. Finally, memory devices power consumption is greater than standard logic devices.

Because of the disadvantages of Memory units used as logic devices, other methods were proposed. The most common method was to implement different interconnections in a bigger device. Thus, many standard logic devices could be implemented in one big device. This would increase design flexibility. Ron Cline from Signetics^{TM} created the idea of two programmable planes [27]. This would allow to implement any combinations of “AND” and “OR” gates. The structure called **P**rogrammable **L**ogic **A**rray (PLA). Although the architecture was very flexible, from inputs to outputs propagation delay was high due to the technology limitations at that time.

Not only the propagation delay of PLA’s, but also the problems of the outputs did not have dedicated product terms; put the researches into fixing one programmable plane and creating **P**rogrammable **A**rray **L**ogic (PAL). MMI^{TM} proposed to fix the OR plane in PAL architecture. As a result, less propagation delays achieved, however the design flexibility gets worse compared to PLA architecture. Other architectures also proposed and a general term Programmable Logic Device (PLD) is named to those architectures at all.

Since the chip densities are increased, the next item for PLD developers is to produce new products with larger logical densities. For this improvement, PLD’s evolved into macrocells. The PAL’s technology depended on combinational and registers structures. The PAL’s are extended with additional flip-flop’s (FF) in order to implement the macrocells [1]. Complex Programmable Devices concept is basically having several PLD blocks or macrocell’s in a single device and connecting them with a programmable general-purpose-interconnect block [27]. The switch matrix implemented in CPLD is usually not able to connect all the macrocells or I/O’s to each other. Therefore, %100 utilization of a CPLD macrocells is impossible because of switch matrix. However, CPLD’s are still best choice for high performance control logic implementations and address decoding schemes. This is because they have low and predictable propagation delay times [28].

In1985, Xilinx totally changed the programmable logic design technology by introducing the first Field Programmable Gate Array (FPGA) ever. The new technology is based on Configurable Logic Blocks, surrounded by I/O modules and

Switch Matrixes. Figure 2-1 shows the FPGA architecture which is composed of CLB's, I/O Modules and Switch Matrices.

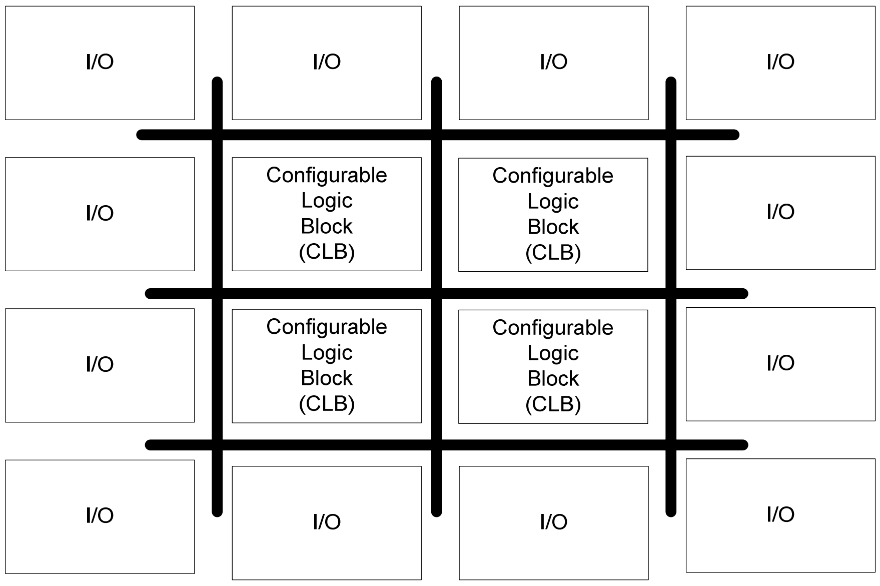

I/O
I/O
I/O
I/O
I/O
Configurable Logic Block (CLB)
Configurable Logic Block (CLB)
I/O
I/O
Configurable Logic Block (CLB)
Configurable Logic Block (CLB)
I/O
I/O
I/O
I/O
I/O

Figure 2-1 FPGA architecture

The configurable logic cells are the heart of the new technology, they are composed of gates, look-up tables and flip-flops (FF). The switch matrices are programmable wiring. The switch matrices are responsible of connecting CLB's to CLB's and CLB's to I/O Blocks. Finally, the I/O blocks provide selectable input, output or bi-directional access to the I/O pins of the FPGA.

Since there are extra flip-flops (FF) in FPGA, the architecture of FPGA is flexible than CPLD. Thus, FPGA is a better choice for a heavy-register based or pipelined applications [28].

2.1.2 Advantages of Programmable Logic Design

Since 1970's Programmable Device technology is developed much further than it was predicted. In today's FPGA and CPLD devices, there are nearly 10 million gates present. For example, ALTERA's Stratix IV family FPGA's and XILINX's Virtex V family FPGA's can have more than several millions of gates. On the other hand, those devices are as small as their ancestors and the space their package occupies is not more than 1000 mm^2. There are several reasons why the programmable logic technology wins the competition against dedicated logic and even Application Specific Integrated Circuit (ASIC) devices. Here are the several advantages of programmable design: [27]

- Ease of design: Once the design is described by schematic, flow chart or Hardware Description Language (HDL), the design process starts. Afterwards, the design could be optimized, analyzed, synthesized and fit into off-the-shelf CPLD device with the help of development tool. Moreover, you can simulate the design and debug the problematic issues without the need of the hardware [27].

- Lower development costs: Since the programmable devices are re-programmable, you can enhance your design in any step of the design process. This lowers the cost of development because there is no need to change PCB. Moreover, the development tools are very inexpensive (ALTERA, XILINX or ACTEL) [27].

- More Product revenue: Programmable logic devices included designs developed in shorter times, which shortens the time to market, and begin to revenue sooner [27].

- Support for Input/Output Standards: Many FPGA/CPLD vendors provide standard I/O functionality on their devices, including electrical and timing characteristics of those standards [27]. For instance, PCI, CMOS, LVTTL etc.

2.1.3 Programmable Logic Design Process

The main difference of Hardware and Software design is that in software code is executed sequentially, however in hardware design; all the inputs are processed in parallel. This means, the input signals are executed in macrocells or PLD's in parallel and routed towards their final destination through switch matrices and then they reach their final destinations output pins. As it is stated in previous sections, the programmable logic designs are mostly entered using HDL (Hardware Description Language)'s, since it is hard to describe complex designs with schematics or flow charts. Moreover, in HDL based design entries, the programmer should think the parallel execution of inputs. Therefore, the statements of HDL create, at the same time executed structures. In Figure 2-2, the programmable logic design process can be seen [29].

- The design is entered with HDL as VHDL (Very high level Hardware Description Language) or VerilogHDL (Verilog Hardware Description Language).

- Required simulations are carried out and if necessary, design entry changes are made to qualify the design.

- With the help of development tool, necessary analysis and synthesis is done. In that step, specific design library and design constraints are also considered.

- The development tool, place and routes the design into the off-the-shelf device you chose to import your design. If there are problems in fitting design into the device a bigger device can be chosen or several design entry changes can be done.

- The timing analysis and simulations are carried on the output of the development program. These simulations are carried out in order to understand if the timing requirements are met or not. If there are problems, necessary improvements are done on design entries, simulations or fitter settings.
- Finally, the qualified design programmed into device and hardware tests can be started on PCB.

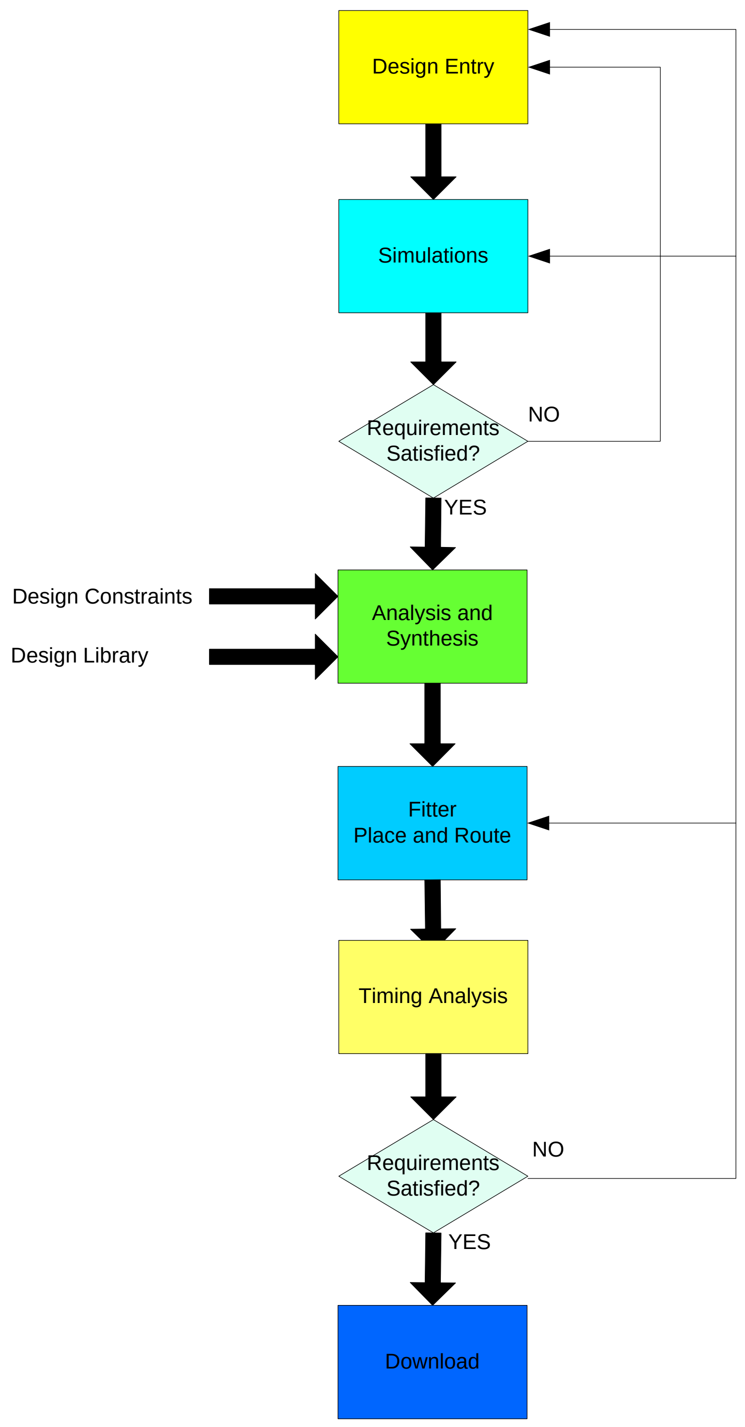


Figure 2-2 Flow Chart of Programmable Logic Design [27]

2.2 RELATED RESEARCH on VME

2.2.1 History of VME

The microcomputer bus industry had begun with the advent of microcomputers [1]. At 1980's there were several busses present. However, the drawbacks of those defined buses were, they were supported one or two types of microprocessors and they had limited addressing range. In such a situation, a few engineers working at Motorola decided to define a bus which is microprocessor independent. As a result, a universal bus could be defined and independent vendors could design products communicating on same bus. Thus, they created the VME bus system starting with MOTOROLA 68000 CPU series. Actually, the main idea was creating a bus system which is compatible to Motorola 68000 CPU bus system. Moreover, not only the architecture but also the card mechanical form factors needed to be defined for standardization. Therefore, VERSA Module Eurocard which is mostly named VME Bus was developed. At that point, the companies including Signetics, Philips, Thomson and Mostek agreed to use this standard. Soon, the bus architecture was officially standardized by ANSI and IEEE as ANSI/IEEE 1014-1987 [2]. The VME bus opened new horizons for designers because, it was microprocessor independent, it could have 16/32 bit data path, a reliable mechanical standard was defined and independent vendors are allowed to build compatible products [1]. Today, the available standards of VME are ANSI/VITA 1994 [3] which is known as VME64 and ANSI/VITA-1997 [4] which is also known as VME64X.

2.2.2 VME BUS

2.2.2.1 Objectives of VME Bus Specification

As it is defined in ANSI/VITA-1994, also known as VME64, VME Bus specifications define an interfacing system used to interconnect microprocessors, data storage, and peripheral control devices in a defined hardware configuration. Moreover, the system specifications have following objectives [3]:

a. To allow devices communicate on VME Bus without disturbing the internal activities of other devices interfaced to the VME bus.

b. To specify the electrical and mechanical system characteristics required to design devices that will reliably and unambiguously communicate with other devices interface to the VME Bus.

c. To specify protocols that precisely defines the interaction between the VME Bus and devices interface to the VME bus.

d. To provide terminology and definitions that describe system protocol

e. To allow a broad range of design latitude so that the designer can optimize cost and/or performance without affecting system compatibility.

f. To provide a system where performance is primarily device limited, rather than system interface limited.

2.2.2.2 Terminology Used For VME Bus

In this section several VME terminologies will be explained.

- Data Transfer Bus (DTB) is the one of the four defined sub busses on VME Bus. DTB is responsible for data transfers between Masters and Slaves.
- Master is a VME Card that is capable of initiating a transfer on VME Bus.
- Slave is a VME Card that is responsible of responding Master VME Card when the Master VME Card addresses itself.
- Priority Interrupt Bus is the one of the four defined sub busses on VME Bus. Priority Interrupt Bus is responsible of interrupt request transfers from interrupter module to interrupt handler module.

- Interrupt is the VME Card that can generate interrupt requests and when the interrupt handler requests it must provide necessary STATUS/ID information to interrupt handler.

- Arbitration Bus is the one of the four defined sub busses on VME Bus. Arbitration Bus is responsible of providing media for Arbiter Module and Requester to coordinate the usage of DTB.

- Utility Bus is the one of the four defined sub busses on VME Bus. Utility Bus is responsible of coordinating power-up, power-down sequence and provides periodic timing for VME bus system.

- System Controller is VME card resides in slot 1 of VME Backplane. System controller has a central arbiter for DTB allocation, IACK Daisy-Chain driver for interrupt acknowledge daisy chain, Bus timer as DTB watchdog timer and also Power Monitor for monitoring power and coordinating power-up and power-down sequences.

In Figure 2-3, four defined sub buses on VME backplane and their interaction with defined functional modules can be seen.

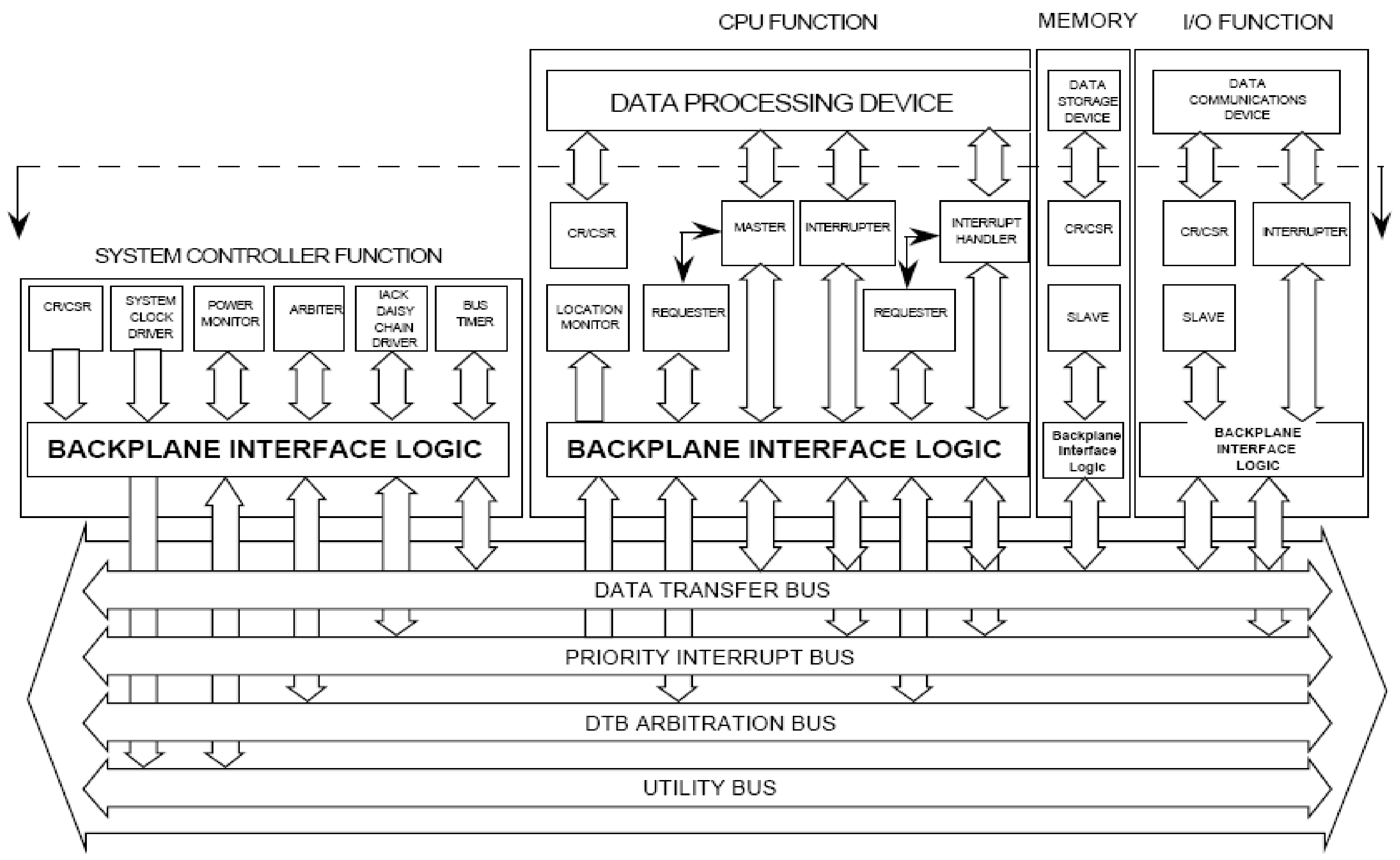


Figure 2-3 Functional Modules and Defined Buses on VME Bus [4]

2.2.2.3 VME Bus Structure

VME Bus is a multiprocessing bus based on master-slave architecture. The VME Bus is multiprocessing bus since there are many masters that can reside on backplane at the same time. Masters are allowed to initiate data transfers from or to slave. Before starting a transfer, VME Master should acquire the DTB, using Arbitration bus from a central arbiter. The arbiter module resides in System controller, which resides in slot 1 on VME Backplane. The job of this central arbiter is to process the data transfer bus usage requests from master VME cards, then decide and allocate the usage of DTB to the appropriate master VME Card.

VME bus is an asynchronous bus that means the date transfers realized by handshaking methodology. Therefore, there is no general clock to coordinate transfers. Moreover, the speed of transfer is stated by the slowest card that participates in transfer cycle. The main features of VME Bus are in Table 2-1.

Table 2-1 Features of VME Bus

ITEM	**SPECIFICATION**
Architecture	Master/Slave
Transfer Mechanism	Asynchronous, Multiplexed/Non-Multiplexed
Addressing Range	16 / 24 / 32 / 64 bits
Data width	8 / 16 / 24 / 32 / 64 bits
Unaligned Data Transfers	Yes
Error Detection	Yes (BERR signal, RETRY signal (VME64x))
Interrupts	7 level (1-7)
Multiprocessing	Yes (1-21 processors)
System Diagnostic Capability	Yes (SYSFAIL signal)
Hot-Swap Capability	Yes
Geographic Addressing	Yes
Mechanical Standard	3U / 6U / 9U

2.2.2.4 Multiplexed and Non-multiplexed Architecture

Multiplexed busses share the same pins for both address and data information transfers. Non- multiplexed busses have different pins for address and data information transfers. Both multiplexed and non-multiplexed bus configurations are showed in Figure 2-4.

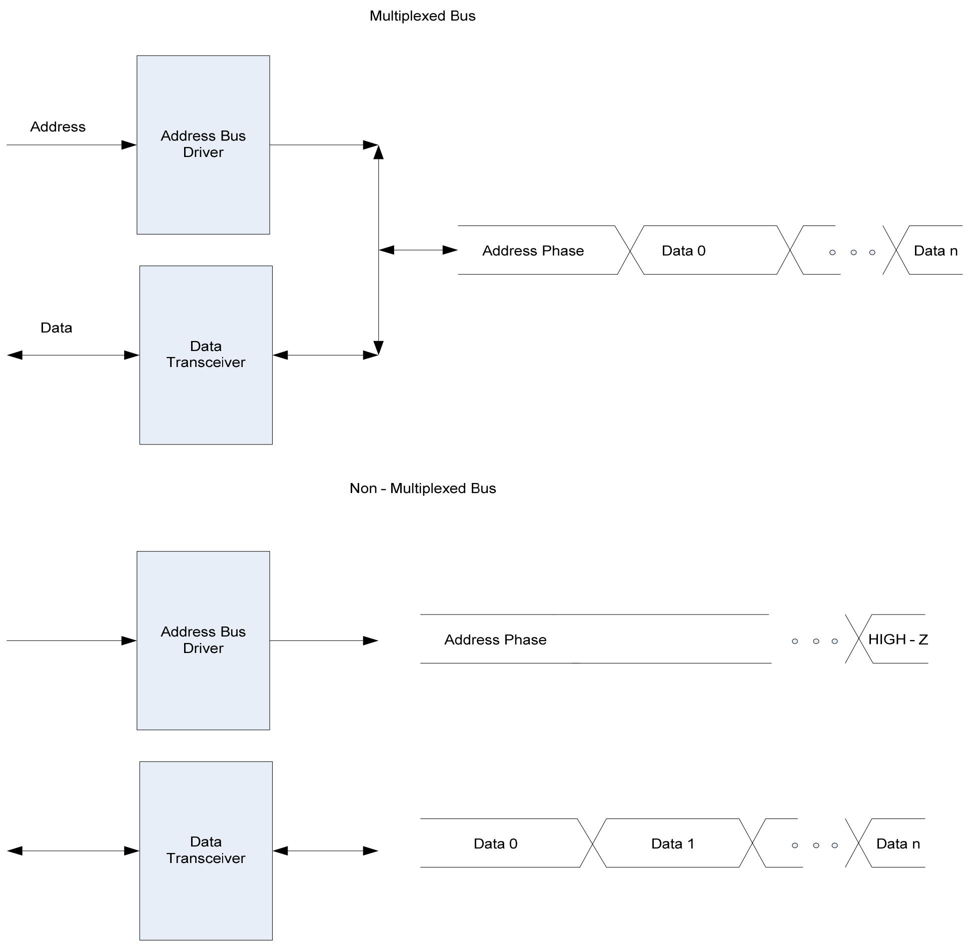


Figure 2-4 Multiplexed and Non-Multiplexed Bus

VME Bus architecture supports both architectures. Up to 32 bit address or data transfers the non-multiplexed architecture is supported, on the other hand for 40 and 64 bit addressing, or 64 bit data transfers, multiplexed bus architecture is supported.

2.2.2.5 Mechanical Standard

VME Bus cards are mainly designed in 6U form factor. In Figure 2-5, the mechanical drawing of 6U – double height VME Card is shown.

The double height VME card is 160 mm x 233 mm. This 6U card contains 3 connectors on bottom of card. Those connectors named P0, P1 and P2. P0 is a 95(19 x 5) pins connector and all of its signals are user defined. P1 and P2 are 160 (32 x 5) pins connectors. The pins on P1 are VME specific signals; however most of the pins on P2 connector are user defined. In the following sections the pin mapping of all three connectors will be given.

2.2.2.6 P1 and P2 (32x5 pin) Connectors

The P1 and P2 are 5 row connectors and they are the difference, between VME64 and VME64X. On VME64 standard those connectors defined as 3 row connectors so they have 32 x 3 = 96 pins. Advantages of the new 5 row pins are new signal connections for additional functionalities and more ground pins for signal integrity. Moreover, four pre-charge pins are provided on connector, which form “make-first, break-last” connection. Thus, hot-swap capable boards can be implemented. The P1 and P2 connector positions on board are shown in Figure 2-6. The pin assignments of P1 and P2 connectors can be found in Table 2-2 and Table 2-3.

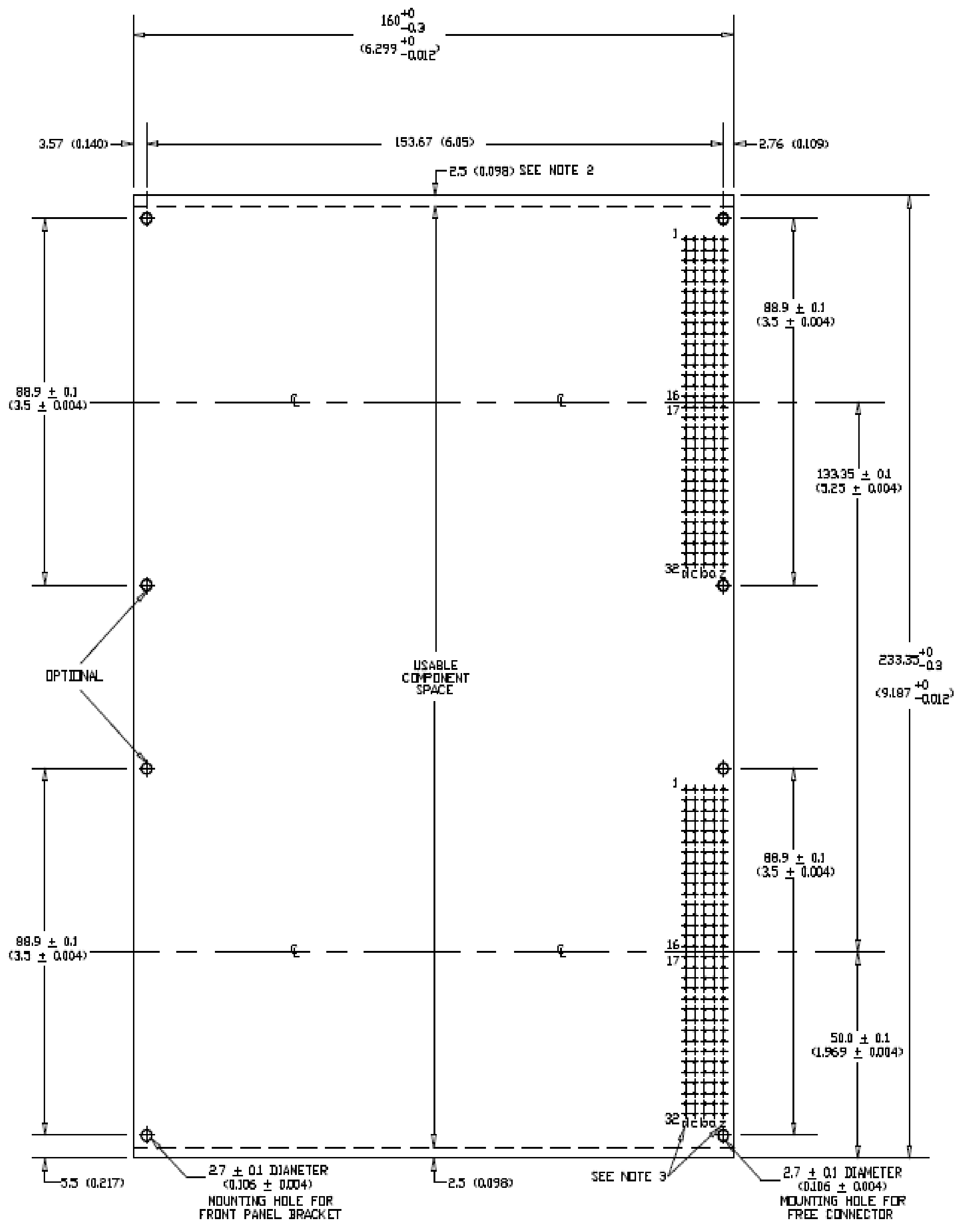


Figure 2-5 Double Height Board (6U) Dimensions

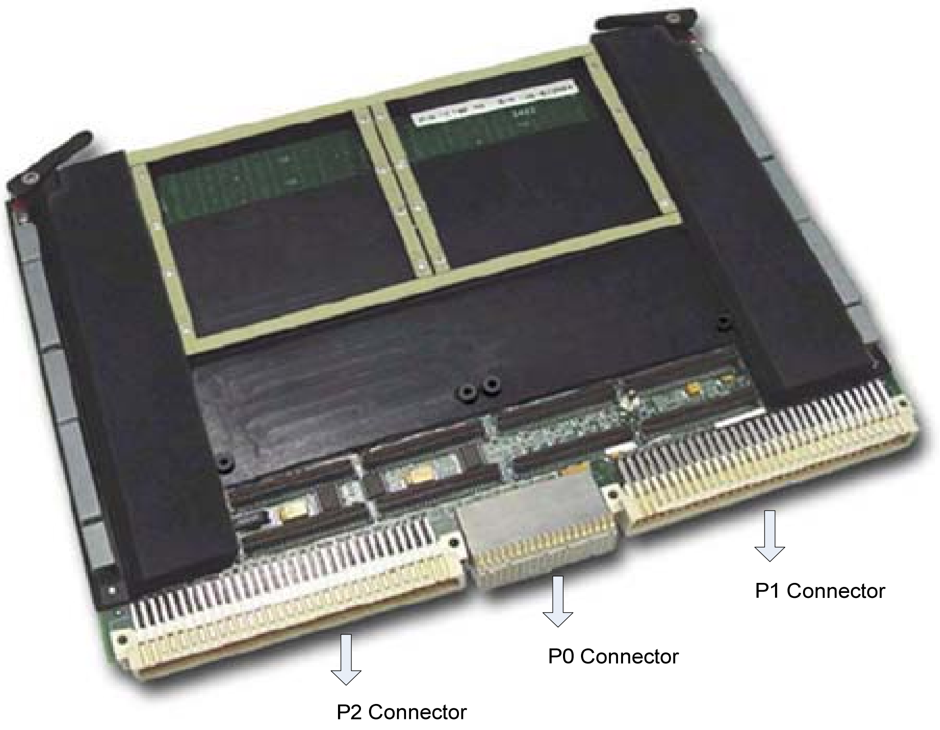


Figure 2-6 Double Height Board Connectors [1]

Table 2-2 VME Bus P1 Pin Assignments [7]

VME P1 Connector					
Pin	**Row D**	**Row C**	**Row B**	**Row A**	**Row Z**
1	VPC	VME_DATA(8)	VME_BBSYn	VME_DATA(0)	VME_MPR
2	GND	VME_DATA(9)	VME_BCLRn	VME_DATA(1)	GND
3	+ V1	VME_DATA(10)	VME_ACFAILn	VME_DATA(2)	VME_MCLK
4	+ V2	VME_DATA(11)	VME_BG0INn	VME_DATA(3)	GND
5	RsVU	VME_DATA(12)	VME_BG00UTn	VME_DATA(4)	VME_MSD
6	- V1	VME_DATA(13)	VME_BG1INn	VME_DATA(5)	GND
7	- V2	VME_DATA(14)	VME_BG1OUTn	VME_DATA(6)	VME_MMD
8	RsVU	VME_DATA(15)	VME_BG2INn	VME_DATA(7)	GND
9	VME_GAPn	GND	VME_BG2OUTn	GND	VME_MCTL
10	VME_GA0n	VME_SYSFAILn	VME_BG3INn	VME_SYSCLK	GND
11	VME_GA1n	VME_BERRn	VME_BG3OUTn	GND	RESPn NC
12	POS3V3	VME_SYSRESETn	VME_BR0n	VME_DS1n	GND
13	VME_GA2n	VME_LWORDn	VME_BR1n	VME_DS0n	RsvBus
14	POS3V3	VME_AM(5)	VME_BR2n	VME_WRITEn	GND
15	VME_GA3n	VME_ADDR(23)	VME_BR3n	GND	RsvBus
16	POS3V3	VME_ADDR(22)	VME_AM(0)	VME_DTACKn	GND
17	VME_GA4n	VME_ADDR(21)	VME_AM(1)	GND	RsvBus
18	POS3V3	VME_ADDR(20)	VME_AM(2)	VME_Asn	GND
19	RsvBus	VME_ADDR(19)	VME_AM(3)	GND	RsvBus
20	POS3V3	VME_ADDR(18)	GND	VME_IACKn	GND
21	RsvBus	VME_ADDR(17)	VME_SERCLK	VME_IACKINn	RsvBus
22	POS3V3	VME_ADDR(16)	VME_SERDAT	VME_IACKOUTn	GND
23	RsvBus	VME_ADDR(15)	GND	VME_AM(4)	RsvBus
24	POS3V3	VME_ADDR(14)	VME_IRQ7n	VME_ADDR(7)	GND
25	RsvBus	VME_ADDR(13)	VME_IRQ6n	VME_ADDR(6)	RsvBus
26	POS3V3	VME_ADDR(12)	VME_IRQ5n	VME_ADDR(5)	GND
27	VME_LI_In	VME_ADDR(11)	VME_IRQ4n	VME_ADDR(4)	RsvBus
28	POS3V3	VME_ADDR(10)	VME_IRQ3n	VME_ADDR(3)	GND
29	VME_LI_On	VME_ADDR(9)	VME_IRQ2n	VME_ADDR(2)	RsvBus
30	POS3V3	VME_ADDR(8)	VME_IRQ1n	VME_ADDR(1)	GND
31	GND	POS12V	POS5VSTDBY	NEG12V	RsvBus
32	VPC	POS5V	POS5V	POS5V	GND

Table 2-3 VME Bus P2 Pin Assignments [4]

VME P2 Connector					
Pin	**Row D**	**Row C**	**Row B**	**Row A**	**Row Z**
1	User Defined	User Defined	POS5V	User Defined	User Defined
2	User Defined	User Defined	GND	User Defined	GND
3	User Defined	User Defined	VME_RETRYn	User Defined	User Defined
4	User Defined	User Defined	VME_ADDR(24)	User Defined	GND
5	User Defined	User Defined	VME_ADDR(25)	User Defined	User Defined
6	User Defined	User Defined	VME_ADDR(26)	User Defined	GND
7	User Defined	User Defined	VME_ADDR(27)	User Defined	User Defined
8	User Defined	User Defined	VME_ADDR(28)	User Defined	GND
9	User Defined	User Defined	VME_ADDR(29)	User Defined	User Defined
10	User Defined	User Defined	VME_ADDR(30)	User Defined	GND
11	User Defined	User Defined	VME_ADDR(31)	User Defined	User Defined
12	User Defined	User Defined	GND	User Defined	GND
13	User Defined	User Defined	POS5V	User Defined	User Defined
14	User Defined	User Defined	VME_DATA(16)	User Defined	GND
15	User Defined	User Defined	VME_DATA(17)	User Defined	User Defined
16	User Defined	User Defined	VME_DATA(18)	User Defined	GND
17	User Defined	User Defined	VME_DATA(19)	User Defined	User Defined
18	User Defined	User Defined	VME_DATA(20)	User Defined	GND
19	User Defined	User Defined	VME_DATA(21)	User Defined	User Defined
20	User Defined	User Defined	VME_DATA(22)	User Defined	GND
21	User Defined	User Defined	VME_DATA(23)	User Defined	User Defined
22	User Defined	User Defined	GND	User Defined	GND
23	User Defined	User Defined	VME_DATA(24)	User Defined	User Defined
24	User Defined	User Defined	VME_DATA(25)	User Defined	GND
25	User Defined	User Defined	VME_DATA(26)	User Defined	User Defined
26	User Defined	User Defined	VME_DATA(27)	User Defined	GND
27	User Defined	User Defined	VME_DATA(28)	User Defined	User Defined
28	User Defined	User Defined	VME_DATA(29)	User Defined	GND
29	User Defined	User Defined	VME_DATA(30)	User Defined	User Defined
30	User Defined	User Defined	VME_DATA(31)	User Defined	GND
31	GND	User Defined	GND	User Defined	User Defined
32	VPC	User Defined	5 V DC	User Defined	GND

2.2.2.7 P0 (19 x 5 pin) Connector

The VME64x standard allows 95 pin P0 connector. This connector is placed between P1 and P2 as shown in Figure 2-6. This is a difference between VME64 and VME64X, the new connectors all pins are user defined. Moreover, the new connector describes new high speed user defined signaling over VME backplane. This is suitable for needed high speed signaling in military and telecommunication applications. For instance, a gigabit Ethernet could be routed over backplane using P0 connector. The pin assignments of P0 connector can be found in Table 2-4.

Table 2-4 VME Bus P0 Pin Assignments [4]

VME P0 CONNECTOR					
Pin	**Row E**	**Row D**	**Row C**	**Row B**	**Row A**
1	User Defined	User Defined	User Defined	User Defined	User Defined
2	User Defined	User Defined	User Defined	User Defined	User Defined
3	User Defined	User Defined	User Defined	User Defined	User Defined
4	User Defined	User Defined	User Defined	User Defined	User Defined
5	User Defined	User Defined	User Defined	User Defined	User Defined
6	User Defined	User Defined	User Defined	User Defined	User Defined
7	User Defined	User Defined	User Defined	User Defined	User Defined
8	User Defined	User Defined	User Defined	User Defined	User Defined
9	User Defined	User Defined	User Defined	User Defined	User Defined
10	User Defined	User Defined	User Defined	User Defined	User Defined
11	User Defined	User Defined	User Defined	User Defined	User Defined
12	User Defined	User Defined	User Defined	User Defined	User Defined
13	User Defined	User Defined	User Defined	User Defined	User Defined
14	User Defined	User Defined	User Defined	User Defined	User Defined
15	User Defined	User Defined	User Defined	User Defined	User Defined
16	User Defined	User Defined	User Defined	User Defined	User Defined
17	User Defined	User Defined	User Defined	User Defined	User Defined
18	User Defined	User Defined	User Defined	User Defined	User Defined
19	User Defined	User Defined	User Defined	User Defined	User Defined

2.2.2.8 VME Signal Definitions

In this section the VME bus signals are explained in Table 2-5.

VME Bus uses an ‘n’ following a signal name shows it is an active low signal. That means the signals active state is logic zero.

Table 2-5 VME Bus Signal Definitions [1]

Signal Name	Definition
VME_ADDR (31:01)	The address bus of VME bus is driven by masters or interrupt handlers. There is 31 bits present on address bus. Thus, the zeroith bit is encoded on two data strobes on VME bus which are DS0n and DS1n. For data transfer cases, the address bus can be 16, 24, 32 or 64 bits. For 64 bit addressing both address bus and data bus are used. The width of the address bus is stated by address modifier code (AM5 – AM0). Interrupt handlers drive A1-A3 to state what level of interrupt is being acknowledged.
VME_ACFAILn	ACFAILn signal is driven by power monitor. When the ACFAILn signal asserted, it states that the power is going to be quit. The use of ACFAILn is optional.
VME_AM (5:0)	The address modifier code is driven by master. These signals indicate both the address size and type of the data transaction.
VME_ASn	The address strobe signal is driven by masters or interrupt handlers. When the ASn signal is asserted, it states that the address and address modifier code bits are valid for transaction.
VME_BBSYn	The bus busy signal is driven by masters and interrupt handlers. The masters and interrupt handlers assert this signal while they are using data transfer bus. The arbiter monitors this signal to decide when to allocate the bus another master.
VME_BCLRn	The bus clear signal is driven by bus arbiter. The bus arbiters asserts this signal to inform the master or interrupt handler that is using data transfer bus that another requester requests the bus.

Cont'd Table 2-5 VME Bus Signal Definitions [1]

Signal Name	Definition
VME_BERRn	The bus error signal is driven by slave or bus timer module. This signal inserted to inform the transfer initiator that an error occurred during transaction
VME_BG3Inn- VME_BG0INn / VME_BG3OUTn - VME_BG0OUTn	The bus grants signal are occupied is bus grant daisy chain. Those signals are driven by arbiter and bus requesters. These signals are used for bus arbitration between masters.
VME_BR3n – VME_BR0n	The bus request signals are driven by requesters. Those signals are used to request the data transfer bus from arbiter.
VME_DATA (31:0)	The VME data bus signals are driven by masters or slaves or interrupt handlers. Those signals are bidirectional. There can be 8, 16, 32 and 64 bit data transfers. The VME_ADDR (1), VME_ADDR (2), DS0n, DS1n, LWORDn signals are used to encode size and type of data transfer is going to be taking place.
VME_DS0n and VME_DS1n	The data strobe signals are driven by masters or interrupt handlers. Those signals are used to validate the data on data bus. Moreover, they are used as address zeroith bit and stating size and type of data transfer with the combination of other signals.
VME_DTACKn	The data acknowledge signal is driven by masters, slaves or interrupt handlers. This signal is used for handshaking between master and slave during data transactions.
VME_GA4n - VME_GA0n	The geographical address signals. They are used to inform the VME card that at which slot it is in on VME bus.
VME_GAPn	The geographical address parity signal is indicating parity of the geographical address.
GND	The ground signal is a signal reference or power return path.
VME_IACKn	The interrupt acknowledge signal is driven by interrupt handlers. The interrupt handlers assert this signal to inform interrupters that this interrupt handler is going to handle the interrupt request.
IRQ7n – IRQ1n	The interrupt request signals are driven by interrupters. The interrupters assert one of seven levels of IRQ lines in order to request interrupt handling. Level 7 has the highest priority.

Cont'd Table 2-5 VME Bus Signal Definitions [1]

Signal Name	Definition
LI/I	The live insertion in signal is used to carry control information for hot swap capability.
LI/O	The live insertion out signal is used to carry control information for hot swap capability.
VME_LWORDn	The long word signal is driven by masters. It is used as stating size and type of data transfers with other signals as well as being zeroith bit of address and data on 64 bit address or data transfers.
MCLK	The module clock signal is used for boundary scan test.
MCTL	The module control signal is used for boundary scan test.
MMD	The module data signal is used for boundary scan test.
MPR	The module pause request signal is used for boundary scan test.
MSD	The slave data is used for boundary scan test.
RESERVED	The reserved signals are not used and for future prospects.
RESPn	The response signal is used to carry information.
RsvB	The reserved bus signals are not used and for future prospects.
RsvU	The reserved un-bussed signals are not used and for future prospects.
VME_RETRYn	The retry signal is driven by slave. A slave asserts this signal to inform the transfer initiator to retry this transaction again later.
SERA and SERB / SERCLK and SERDATn	The SERA and SERB signals re used for optional serial bus such as I2C.
VME_SYSCLK	The system clock signal is used as an optional 16 MHz global clock and driven by system controller
VME_SYSFAILn	The system fail signal can be driven by masters or slaves. This signal is asserted to inform other bus residents that the VME Bus fails.
VME_SYSRESETn	The system resetn signal is driven by any module. This signal resets the whole VME Bus system.
User Defined	The user defined signals are totally defined by user needs.
VPC	The voltage pre-charge pins form a make-first/break-last type connection for hot swapping.

Cont'd Table 2-5 VME Bus Signal Definitions [1]

Signal Name	Definition
V1/V2 Auxiliary Power	The auxiliary power pins are used to supply 38-75 V auxiliary power to the VME Card.
VME_WRITEn	The write signal is driven by masters. When, this signal is asserted, a write transaction is going to take place that means a data transfer from master to slave. On the other hand when it is de-asserted, a read transaction s going to take place, that means a data transfer from slave to master.
+5V STANDBY	It is a 5V optional stand by power source.
+3.3 V DC	It is a 3.3 V main power source.
+5 V DC, +12 V DC, -12 V DC	They are 5 V, 12 V and -12 V main power sources.

2.2.3 VME Bus Transfer Cycles

The VME bus has several transfer cycles. In these sections data transfer cycles and interrupt cycles will be explained. VME bus data transfers can be spitted into two phases, at first addressing phase and finally data transfer phase. In the following sections both phases will be explained in detail.

2.2.3.1 VME Bus Addressing

A master addresses a slave on every read/write transaction on VME Bus. The addressing is done by VME_ADDR (31:1). The addressing size and type is declared by address modifier code signals AM (5:0). Moreover, IACKn and LWORDn signals are participated in addressing. All of these signals are qualified by falling edge of address strobe ASn. Also the zeroith bit of address is encoded on DS1n and DS0n signals on VME Bus. The DS1n and DS0n signals are used to decide which bytes of the 4 byte group data are addressed.

There are 5 possible address widths. They are 16, 24, 32, 40 and 64 bit addressing. The address width can be changed on every other cycle, which is also called dynamic address sizing. This brings great flexibility for bus utilization. Thus, old and new cards could reside on same VME backplane with different addressing schemes.

For 64 bit address size, not only VME_ADDR (31:1) bits but also VME_DATA (31:0) bits are used.

2.2.3.1.1 Address Modifier Code

During a bus cycle, the master tags each address with an address modifier code. The slave monitors AM (5:0) signals so it could determine which address lines it should monitor. Moreover, the address modifier code also states the type of transactions. The transaction could be instruction fetches data cycles or another type. In fact, all those types of transactions are developed in early days of VME, since at those times CPU instructions were fetched through VME backplane because memory elements were so big and expensive and not easily presented at the same card that CPU presented. However, at today's technology, fetching mechanism is handled inside the card and VME backplane is mainly used for data transfers.

The address modifier code signals are neglected at interrupt acknowledge cycles. Interrupt acknowledge cycles are validated by IACKn signal. Thus, IACKn signal is important for addressing mechanism. When IACKn is asserted, it is stated that the next transaction is going to be interrupt acknowledge cycle and there will not be addressing and the slave should not decode the AM (5:0) or VME_ADDR (31:1) signals. The defined address modifier coding can be found in Table 2-6.

Table 2-6 Address Modifier Codes for VME64X

VME Bus Address Modifier Codes Under VME64x		
AM(5:0)	**Address Size**	**Description**
0x3F	24	A24 supervisory block transfer (BLT)
0x3E	24	A24 supervisory program access
0x3D	24	A24 supervisory data access
0x3C	24	A24 supervisory 64-bit block transfer (MBLT)
0x3B	24	A24 non-privileged block transfer (BLT)
0x3A	24	A24 non-privileged program access
0x39	24	A24 non-privileged data access
0x38	24	A24 non-privileged 64-bit block transfer (MBLT)
0x37	40	A40BLT [MD32 data transfer only]
0x36	-	Unused / reserved
0x35	40	A40 lock command (LCK)
0x34	40	A40 access
0x33	-	Unused / reserved
0x32	24	A24 lock command (LCK)
0x30 - 0x31	-	Unused / reserved
0x2F	24	CR / CSR space
0x2E	-	Unused / reserved
0x2D	16	A16 supervisory access
0x2C	16	A16 lock command (LCK)
0x2A - 0x2B	-	Unused / reserved
0x29	16	A16 non-privileged access
0x22 - 0x28	-	Unused / reserved
0x21	32 or 40	2eVME for 3U bus modules (address size in XAM code)
0x20	32 or 64	2eVME for 6U bus modules (address size in XAM code)
0x10 - 0x1F	-	User defined
0x0F	32	A32 supervisory block transfer (BLT)
0x0E	32	A32 supervisory program access
0x0D	32	A32 supervisory data access
0x0C	32	A32 supervisory 64-bit block transfer (MBLT)
0x0B	32	A32 non-privileged block transfer (BLT)
0x0A	32	A32 non-privileged program access
0x09	32	A32 non-privileged data access
0x08	32	A32 non-privileged 64-bit block transfer (MBLT)

Cont'd Table 2-6 Address Modifier Codes for VME64X

VME Bus Address Modifier Codes Under VME64x		
AM(5:0)	**Address Size**	**Description**
0x06 - 0x07	-	Unused / reserved
0x05	32	A32 lock command (LCK)
0x04	64	A64 lock command (LCK
0x03	64	A64 block transfer (BLT)
0x02	-	Unused / reserved
0x01	64	A64 single access transfer
0x00	64	A64 64-bit block transfer (MBLT)

2.2.3.2 Data Sizing

During a bus cycle, VME master decides the size of data access. This dynamic data width change done by DS0n, DS1n, VME_ADDR (1), VME_ADDR (2) and LWORDn signals. VME data bus sizing is achieved by splitting data lines into four banks as VME_DATA (7:0), VME_DATA (15:8), VME_DATA (23:16) and VME_DATA (31:24). Moreover, the banks are categorized as:

BYTE (0): VME_DATA (31:24)
BYTE (1): VME_DATA (23:16)
BYTE (2): VME_DATA (15:8)
BYTE (3): VME_DATA (7:0)

VME Bus using BYTE (n) convention to specify, how data is stored and read from memory, where n denotes the address offset from any 32 bit even address boundary.

Moreover, unaligned byte transfers are allowed in VME Bus specification. This means, the modules can place 2, 3 or 4 bytes of data at virtually any address boundary and that allows 32 bits of data to be transferred at odd addresses [1]. VME Bus data length encoding can be found in Table 2-7.

Table 2-7 VME Bus Data Length Encoding

Defined Data length codes for VME							
	DS1	**DS0**	**Address1**	**LWORD**	**Address2**	**Byte #**	**Description**
Data 8 Odd	1	0	0	1	X	1	Single: Byte 1 read or write
Data 8 Odd	1	0	1	1	X	3	Single: Byte 3 read or write
Data 8 Even	0	1	0	1	X	0	Single: Byte 0 read or write
Data 8 Even	0	1	1	1	X	2	Single: Byte 2 read or write
Data 16	0	0	0	1	X	0-1	Double : Byte 0-1
Data 16	0	0	1	1	X	2-3	Double : Byte 2-3
Data 32	0	0	0	0	X	0-1-2-3	Quad : Byte 0-1-2-3
Data 16	0	0	1	0	X	1-2	Double : Byte 1-2
Data 24	0	1	0	0	X	0-1-2	Triple : Byte 0-1-2
Data 24	1	0	0	0	X	1-2-3	Triple : Byte 1-2-3

2.2.3.3 VME Bus Data Transfer Cycles

VME64X Data transfer bus allows 8 types of bus cycles. Those cycles are:

- Read/Write Cycle
- Read-Modify-Write Cycle
- Address-Only Cycle
- Block Transfer Cycle
- Multiplexed Block Transfer (MBLT) Cycle
- Multiplexed Data-32(MD32) Cycle
- Address-Only With Handshake (ADOH) Cycle
- Two-Edge VME Bus (2eVME) Cycle

- Two-Edge Source Synchronous Transfer (2eSST) Cycle

2.2.3.3.1 Read/Write Cycle

A read/write cycle is the most typical cycle on VME Bus. VME master starts the transfer by qualifying A (31:1), AM (5:0), LWORDn, WRITEn and IACKn. Those signals are qualified by asserting ASn signal. If WRITEn and IACKn are de-asserted, it is a typical read cycle on VME Bus. Then for the desired data transaction size and type, DS0n and DS1n signals validated. Afterwards, slave encodes the information on bus and responds by putting the data on D (31:0) and the data on data bus is qualified by slave asserting DTACKn signal. With the assertion of DTACKn signal, master collects the data on data bus and finishes the read cycle by de-asserting, ASn, DS0n and DS1n signals. A typical read cycle is shown in Figure 2-7.

For a write cycles, the master initiates the bus as it is read cycle but with one difference. It asserts the WRITEn signal. Thus, this means the new transaction will be write cycle. Then the master drives the data bus and qualifies the data by asserting DSXn signals to appropriate positions. Then the master waits slave to acknowledge the write process by asserting DTACKn signal. After the DTACKn signal is asserted by slave, master de-asserts ASn, DS0n and DS1n signals and terminates the cycle. A typical write cycle is shown in Figure 2-8.

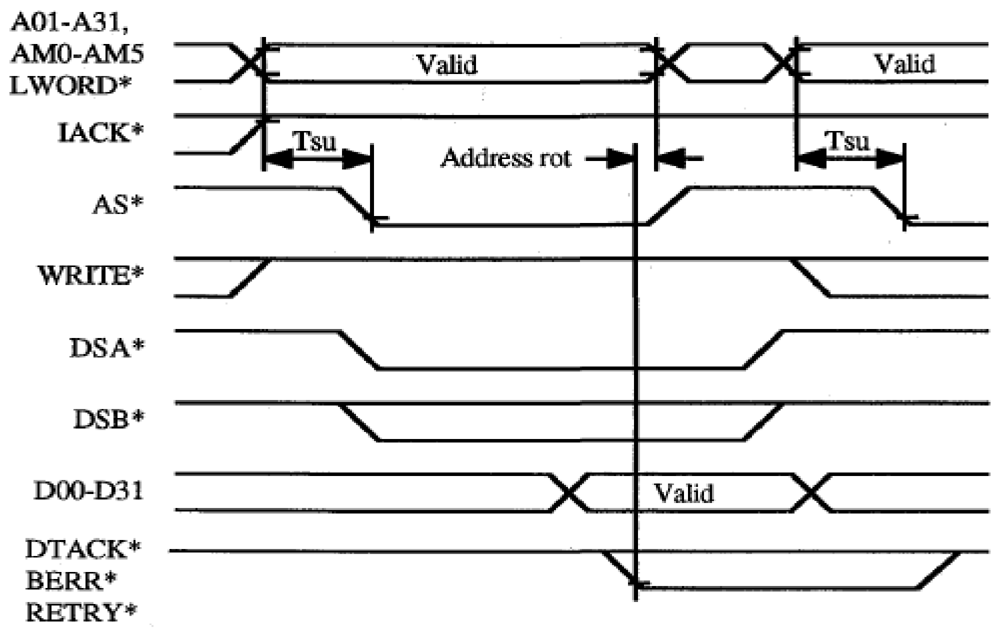


Figure 2-7 VME Bus Read Cycle [1]

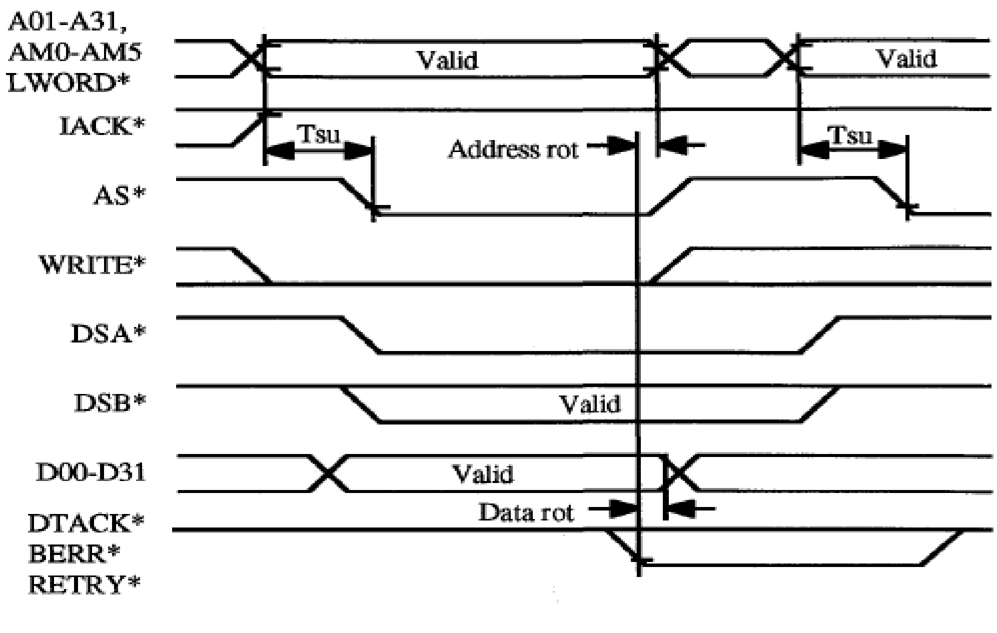


Figure 2-8 VME Bus Write Cycle [1]

2.2.3.3.2 Read-Modify Write Cycle

The read-modify write cycle is like adjacent read and then write cycles. However, in read-modify write cycle, the ASn signal is not de-asserted between read and write phases. Therefore, A (31:1), AM (5:0), IACKn and LWORDn signals are valid for whole transaction period and they could not be changes. The read-modify write cycle is shown in Figure 2-9.

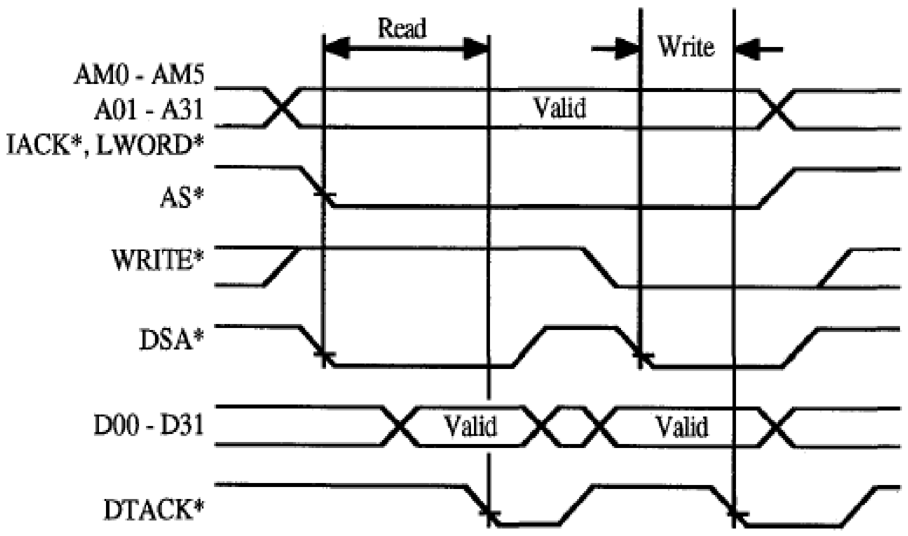


Figure 2-9 VME Bus Read-Modify Write Cycle [1]

2.2.3.3.3 Address-Only Cycle

Address-only cycle is used just for address broadcasting; it does not include data transfers. The address-only (ADO) cycle is shown in Figure 2-10.

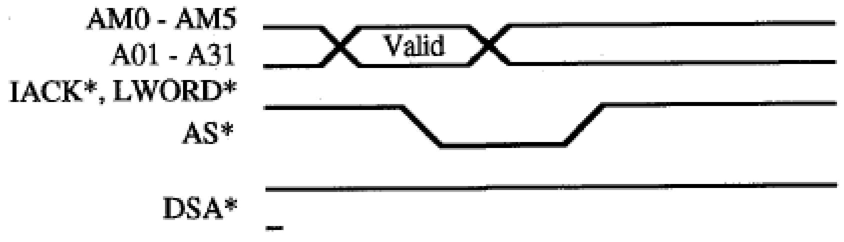


Figure 2-10 VME Bus Address-Only Cycle [1]

2.2.3.3.4 Block Transfer Cycle

The Block Transfer cycle is to move large blocks of data on VME Bus. This cycle also named burst cycle on VME bus. The main advantage of this cycle is when the master gets the DTB ownership; it could transfer up to 256 cycles of data. This increases the utilization of bus and speed of transfer. Moreover, this transfer is useful for controlling peripherals like disk controllers or network devices since they require large blocks of data at once. On the other hand, the block size is limited to 256 bytes, in order not to cross card address boundaries on VME Bus. The limitation is because of bus arbitration technique. If one master allocates the VME bus more than 256 cycles, the global arbitrator should push that master off the DTB and then re-coordinate bus ownership. However if a master is limited to 256 cycles, it would get out of bus by itself and the work load on arbitrator would be less. The Block Transfer Cycle is shown in Figure 2-11. Block Transfer cycle resembles the single read/write cycles; however, master validates the address and address modifier code once during all cycle and than it provides data on every falling edge of data strobe signals. The slave is responsible of incrementing the address by itself where the data targets.

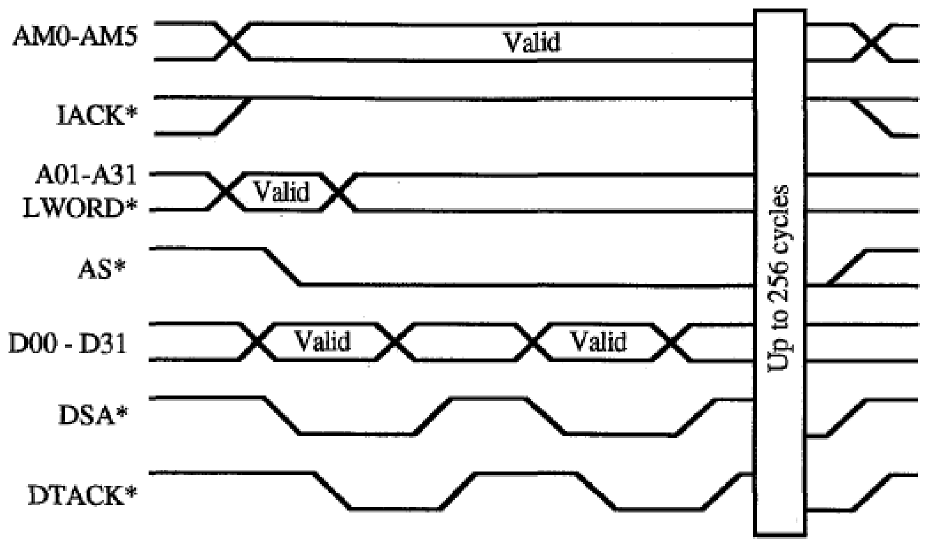


Figure 2-11 VME Bus Block Transfer Cycle [1]

2.2.3.4 VME Bus Interrupt Cycles

VME Bus has seven levels of interrupts IRQ1 to IRQ7. Those interrupts are prioritized and the seventh level interrupt has the highest priority. A module that generates interrupt requests by asserting any of the seven levels of interrupts is called interrupt requester and the module that services the interrupt is called interrupt handler. Every interrupt handler must handle only one level of interrupt [3].

The interrupt acknowledge cycle begins with any level of interrupt assertion of interrupt requester. Then, the corresponding level's interrupt handler tries to acquire the data transfer sub-bus on VME bus for status_id read cycle. Then, the handler puts the level of interrupt on VME address lines 3-1 and, asserts ASn and IACKn to broadcast all VME bus residents about the interrupt acknowledge cycle of corresponding level. The interrupt level encoding on address lines is show in Table 2-8.

Table 2-8 Interrupt Line codes on VME address(3:1)

Interrupt Line	**VME Address 3**	**VME Address 2**	**VME Address 1**
IRQ7	1	1	1
IRQ6	1	1	0
IRQ5	1	0	1
IRQ4	1	0	0
IRQ3	0	1	1
IRQ2	0	1	0
IRQ1	0	0	1

Afterwards, the IACK daisy-chain driver in system controller on slot 1 of VME bus starts interrupt IACKIN / IACKOUT Daisy chain on backplane. The daisy-chain propagates from module to module until it reaches the interrupt requester that resides closer to the system controller. To explain this phenomenon, for instance, both the SLOT4 and SLOT5 VME modules requests IRQ3, the handler acknowledges this request and daisy-chain driver starts daisy-chain. The SLOT4 card will participate in daisy-chain mechanism at first and will respond back with status_id information. However SLOT5 card would still be waiting for interrupt cycle and its IRQ3 line will stay asserted. The daisy-chain mechanism is shown in Figure 2-12.

After the daisy chain arrive the interrupt requester card, it provides necessary status_id information on data transfer bus and finishes the interrupt cycle with asserting DTACKn signal. Therefore, the interrupt acknowledge cycle finishes. Moreover, there are two ways to terminate the interrupt request on IRQ lines. Those methods are Release on Acknowledge (ROAK) and Release on register Access (RORA).

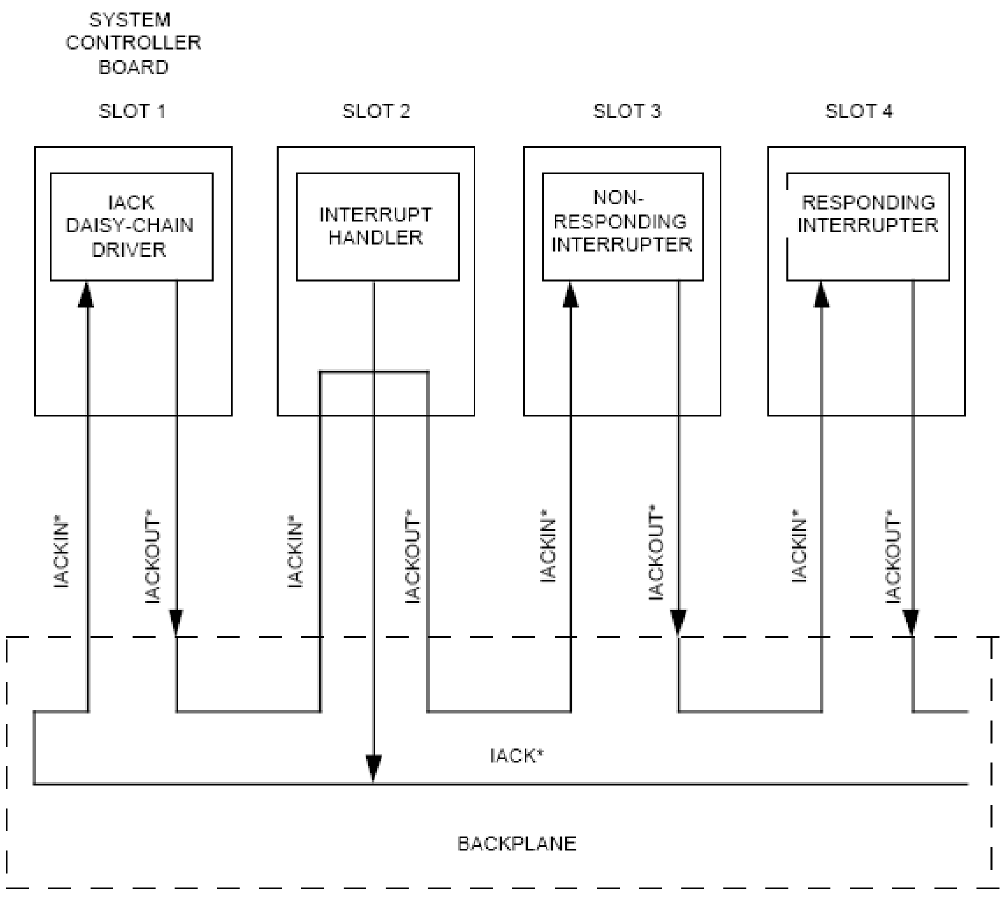


Figure 2-12 VME Bus IACKIN / IACKOUT Daisy-Chain [3]

2.2.3.4.1 ROAK Mechanism to Terminate Interrupt Request

In Release on Acknowledge mechanism, the interrupt request on IRQ lines are terminated when the IACKIN input of the VME Card is asserted. The termination is done by de-asserting the corresponding IRQ line. A ROAK example for IRQ3 is shown in Figure 2-13.

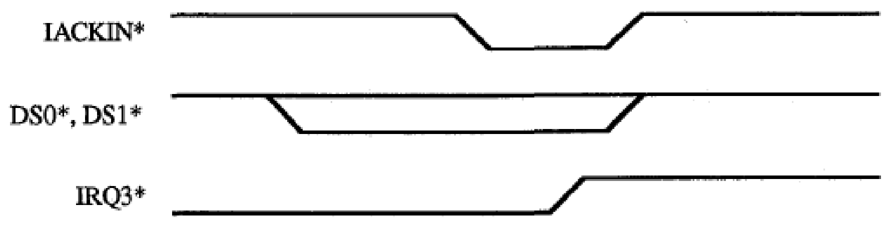


Figure 2-13 ROAK Mechanism to Terminate Interrupt Request [1]

2.2.3.4.2 RORA Mechanism to Terminate Interrupt Request

In Release on Register Access mechanism, the interrupt request on IRQ lines are terminated when the interrupt acknowledge cycle is finished by rising edges of DS0n or DS1n. The termination is done by de-asserting the corresponding IRQ line. A RORA example for IRQ3 is shown in Figure 2-14.

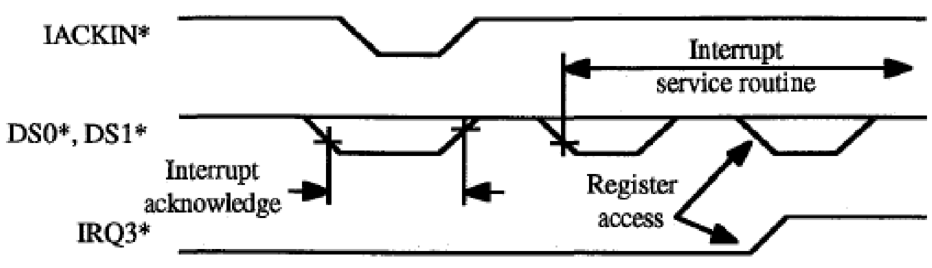


Figure 2-14 RORA Mechanism to Terminate Interrupt Request [1]

2.3 PARALLEL BUS TO SERIAL BUS INTERFACING

Parallel busses are designed for mass data transfers and handling other utility or interrupt services defined inside bus architecture. On the other hand, serial busses are designed for simple and quick data transfers with less number of signal paths. Although, there is a mentality difference between parallel busses and serial busses, in some conditions there is a need to interfacing them.

In order to interface two different types of busses, there should be a control unit implementation which is capable of understanding both sides' protocols. In fact, this would require hard work and the control unit design would be complex. Moreover, this solution would be a problem specific so could not be a generic solution since this way only a one bus to another bus bridge could be implemented.

On the other hand, bus interfaces could be designed that have a common interface on one side, and a standard protocol interface on another side. By that way, the bus interfacing would be much easier. Therefore, many generic modules could be implemented and those modules can communicate with each other on the same defined bus easily.

For the thesis work, there is a general user side interface is developed for intra FGA communication. This structure is explained in the following chapters. For VME Slave Card, there are several peripheral components are connected to FPGA where the FPGA should communicate with them in serial interfaces. Those interfaces are Universal Asynchronous Receiver Transmitter (UART) interface, Inter Integrated Circuit Bus (I2C) interface and Universal Serial Bus (USB) interface. In the following section these interfaces are briefly explained.

2.4 SERIAL BUSES ON VME SLAVE CARD

2.4.1 Universal Asynchronous Receiver Transmitter (UART)

Universal Asynchronous Receive Transmitter (UART) is a serial communication protocol considered as voltage signaling. The examples of voltage signaling standards could be RS-232, RS-422 and RS-485. A standard UART frame is shown in Figure 2-15. There is one start bit, and then five to eight bits of data are sent. If desired, a parity bit must be sent before stop bits, finally one or two stop bits are sent and a five to eight bit data transfer is completed.

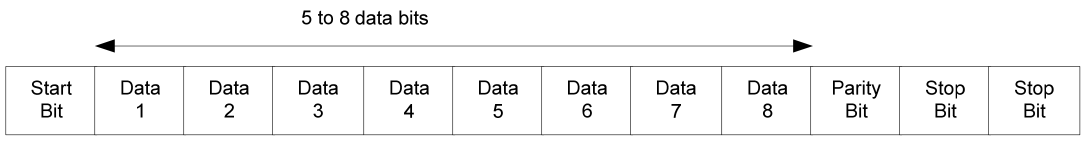


Figure 2-15 Standard UART Frame

2.4.2 Inter Integrated Circuit (I2C) Bus

Inter Integrated Circuit (I2C) Bus is defined by Philips Semiconductors in 1982 and recently updated and 3rd revision is released by NXP [39]. NXP is a company owned by Philips. It is also known as two wire bus since there are two wires to carry the data between IC’s. One signal is called SCLK and other is called SDATA. I2C is a multi-master bus. Thus, every IC connected to I2C bus must have a unique address and it should respond to that addressed transactions.

There are 50 or more companies designing peripheral components communicating via I2C bus [39]. Thus, having an I2C communication port would be great advantage for general purpose modules to have a communication port with various peripheral components including, SEEPROM's, real time clocks (RTC) or temperature sensors.

2.4.3 Universal Serial Bus (USB)

Universal Serial Bus (USB) was developed in 1995 to implement an external expansion bus for Personal Computer's (PC). In fact, USB is a serial bus standard to interface devices [40]. It was designed to allow many peripherals to be connected using a single standardized interface socket and to improve the plug-and-play capabilities by allowing devices to be connected and disconnected without rebooting the computer which could be also called "hot swapping". There are 4 data transfer rates defined for USB standard [40].

- Low Speed (USB 1.1 or USB 2.0) rate of 1.5 Mbits/s
- Full speed (USB 1.1 or USB 2.0) rate of 12 Mbits/s
- High speed (USB 2.0) rate of 480 Mbits/s
- Super Speed (USB 3.0) rate of 4.8 Gbit/s

The data communication on USB is accomplished on one differential pair [40].

There are 4 pins defined for USB communication ports [40].

- PIN 1: 5V voltage supply
- PIN2: D+ (data positive)
- PIN3: D- (data negative)
- PIN4: Ground

CHAPTER 3

VME SLAVE IMPLEMENTATION ON FPGA

3.1 VME SLAVE MODULE DEFINED IN VME64 STANDARD

A block diagram of VME Slave is shown in Figure 3-1 according to the to the VME64 standard. The asterisk (*) is used to denote the active low signals. The dashed lines state the various types of VME Slave modules. As it is seen on block diagram, the VME Slave module is only capable of communicating with Data Transfer Bus according to the standard definition. On the other hand, it should monitor IACKn signal to differentiate, data transfers from interrupt handling cycles. Moreover, it could monitor SYSRESETn signal and the VME slave should be reset by assertion of this signal.

In Table 3-1, how the various types of VME Slave modules use those signals is described.

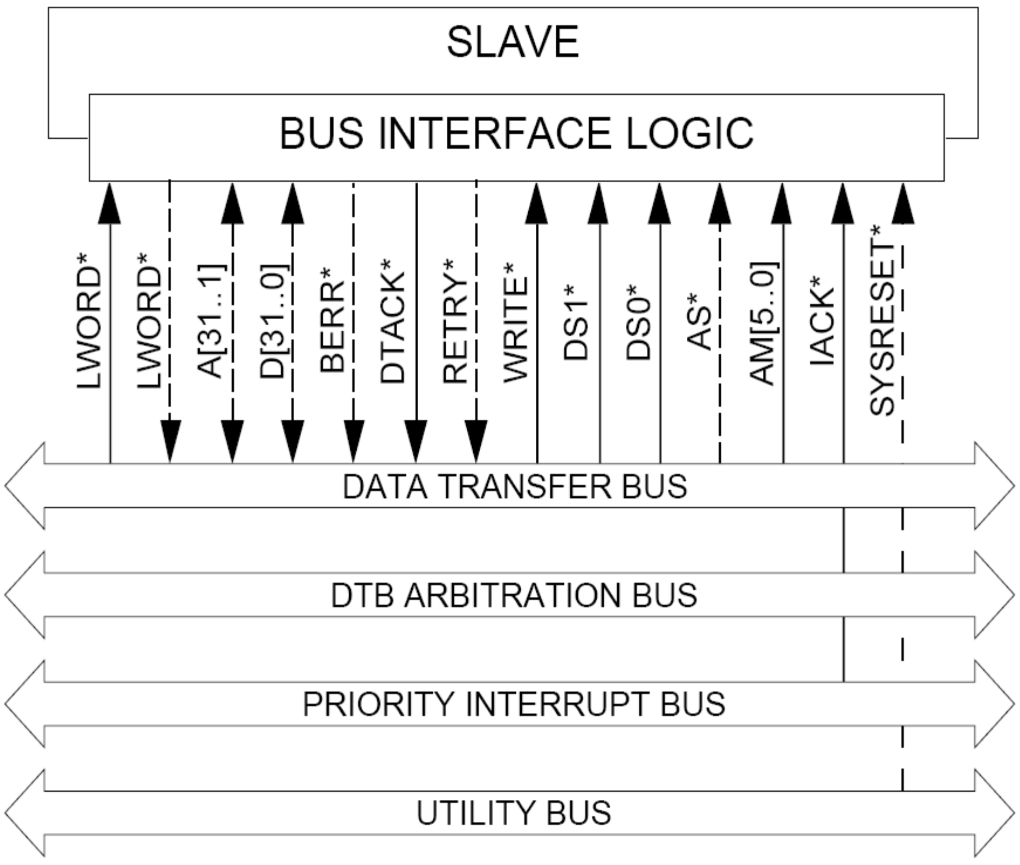


Figure 3-1 A Block Diagram of VME Slave Module According to VME64 Standard [1]

Table 3-1 Rules and Permissions That Specify the Use of Dashed Lines by VME Slave Modules [3]

Type of Slave	Use of dashed lines
D08	MUST monitor and/or drive D(7:0) MAY monitor ASn MAY monitor or drive D(31:8)
D16	MUST monitor and/or drive D(15:0) MAY monitor ASn MAY monitor or drive D(31:16)
D32	MUST monitor and/or drive D(31:0) MAY monitor ASn
MD32	MUST monitor and/or drive A(15:1) MUST monitor and/or drive LWORDn MUST monitor and/or drive D(15:0) MUST monitor ASn
MBLT	MUST monitor and/or drive D(31:0) MUST monitor and/or drive A(31:1) MUST monitor and/or drive LWORDn MUST monitor ASn
A16	MUST monitor A(15:1) MAY monitor A(31:16)
A24	MUST monitor A(23:1) MAY monitor A(31:24)
A32	MUST monitor A(31:1)
A40 & A40MBLT	MUST monitor D(15:0) MUST monitor A(23:1) MUST monitor ASn
A64	MUST monitor A(31:1) MUST monitor D(31:0) MUST monitor ASn
ALL	MAY drive RETRYn MAY drive BERRn MAY monitor SYSRESETn

3.2 PROBLEM DEFINITION

The VME architecture was developed nearly three decades ago and it has been used extensively in military, aerospace, industrial, and communication systems.

Moreover VME systems are also used in high energy physics and nuclear physics experiments. This is because; the VME Bus is flexible, open ended bus system. [16] [5]. According to the Venture Development Corporation (VDC), that is an independent technology market research and strategy consulting firm that specializes in a number of industrial, embedded, component, retail automation, RFID, AIDC, datacom/telecom, and defense markets, the market shares of bus architectures is show in Table 3-2.

Table 3-2 Market Share of Bus Architectures in 2006 (US$ in Millions) [30]

	VME	**Compact PCI**	**Others**
North America	303.8	69.9	37.9
Europe	100.2	43.4	14.9

According to the Table 3-2, the market share of VME Bus Architecture is nearly % 73.4 in North American and % 63 in European markets. Thus, this research of market shares proves the extensive usage of VME Bus in every market. Although there is an extensive usage of VME bus in systems, there are only several companies' offer solutions for VME interfacing. Tundra is one of the major companies that offer VME interfacing, and their solutions are Tsi148 and UniverseII ASIC's (Application Specific Integrated Circuit). Those IC's are capable of pretending as Master, Slave or System controller on VME bus. They are working as VME to PCI/X bridges. Thus, the users should build their modules to communicate with PCI/X bus architecture. On the other hand, there are several companies offering IP (Intellectual Property) solutions. The IP solutions investigated in [19]-[25]. All of those are mostly designed for just one address or data mode. Moreover, they still leave the address decoding side to the user side they defined. Therefore, the user should be aware of the VME address modifier codes for address decoding. Furthermore, mentioned VME IP solutions are vendor specific IP's. This means, those cores are developed for specific products of a specific vendor.

On the other hand, laboratories spend lots of time in order to develop VME Modules at low costs to satisfy their specific experimental requirements [16].

To conclude, in this thesis work, a general VME SLAVE Module is developed, in order to overcome the problems explained above. Moreover, this new module has an advantage of easy and quick re-programmability of FPGA's and modularity of VME standard. On he other hand a reference VME Card is designed for testing the implemented IP Core.

3.3 AVAILABLE VME SLAVE SOLUTIONS

There are several companies offering IP solutions for VME interfacing [19] - [25]. Moreover, researches also have interest in developing VME Modules for generic purposes [16] [5] [17]. The features of in market IP solutions for VME Slave are listed in Table 3-3.

As it could be seen in Table 3-3, there is only one VME slave core with capability of block transfer, this is a draw back if the designer would like to have mass data transfers. Moreover, no VME slave module responds to address only cycles.

When the IP's are compared in interrupter side, Inicore's IP's have selection for ROAK or RORA mechanisms, however they only provide 8 bit STATUS/ID for interrupt acknowledge cycles. On the other hand, MemecCore's and Intelop's have IP's that could provide 8 / 16 /32 bit STATUS/ID for interrupt acknowledge cycle but they can not have RORA interrupt request termination mechanism.

Bus errors are only monitored for MemecCore's and Intelop's products and there is no bus error monitoring for IniCore products. This is also a drawback for those products since they are not monitoring if any transaction errors are occurred and not inform masters for transaction errors on VME bus. Moreover, for user side there is no error detection mechanism or a signal to acknowledge user if there is an error occurs in VME Bus.

Moreover, on user side, the address decoding mechanisms are left for user. Thus, user needs to implement address decoding to interface with VME Slave IP's.

On the other hand, the proposed VME Slave mechanisms [16] [5] [17] [18], the proposed VME slave communication protocols are just for specific purposes. There is no generic structure that everybody could adapt their design into those structures.

Table 3-3 Features of VME slave IP's in Market

Company		**IniCore**			**MemecCore**		**Intelop**			
IP Name		**A24D16 Slave Controller**	**A24D32 Slave Controller**	**A32D32 Slave Controller**	**MC-ACT-VME2416**	**MC-ACT-VME32**	**VME Slave Core**	**Proposed [17]**	**Proposed [16]**	**Proposed [5]**
Data Size	**8**	x	x	x	x	x	x	x		
	16	x	x	x	x	x	x	x	x	
	24							x		
	32		x	x		x	x	x	x	x
Address Size	**16**	x	x	x	x	x	x			
	24	x	x	x	x	x	x		x	
	32			x		x	x	x	x	x
Cycles	**SCT**	x	x	x	x	x	x	x	x	x
	Block						x	x		
	RMW		x	x			x	x		
	AO									

Cont'd Table 3-3 Features of VME slave IP's in Market

Company		**IniCore**			**MemecCore**		**Intelop**			
IP Name		**A24D16 Slave Controller**	**A24D32 Slave Controller**	**A32D32 Slave Controller**	**MC-ACT-VME2416**	**MC-ACT-VME32**	**VME Slave Core**	**Proposed [17]**	**Proposed [16]**	**Proposed [5]**
Interrupter	**8**	x	x	x	x	x	x	x		
	16				x	x	x	x		
	32					x	x	x		
Interrupt Termination Mechanism	**ROAK**	x	x	x	x	x	x	x		
	RORA		x	x						
Rescinding DTACK			x	x						
External Transceiver Control		x	x	x	x	x	x	x	x	x
VME_BERRn										
VME_RETRYn										
VME_SYSRESETn										

3.4 DESIGNED VME SLAVE MODULE FEATURES

The VME SLAVE module is designed to overcome all the problems defined in section "PROBLEM DEFINITION". The features of VME Slave Module are:

- ANSI-VITA-1994 [3] and ANSI-VITA-1997 [4] standard compatible
- A16 / A24 / A32 width address bus
- D8 (EO) / D16 / D32 width data bus
- 1-7 level interrupt generating

- Configurable 8, 16 or 32 bit interrupter
- Supported Cycles :
    - Single Read Cycle (R)
    - Single Write Cycle (W)
    - Read Modify Write Cycle (RMW)
    - Block transfer Cycle (BT)
    - Address Only Cycle (AO)
- Synchronous Local Side interface

First of all, the VME SLAVE module has a generic architecture. Thus, the user could compile the VME SLAVE module in 16/24/32 bit address modes and connect only necessary widths of address and data signals to the VME Bus. This brings flexibility in user design and overcomes the dependency issue on IP core. Moreover, the user could easily upgrade the design by just compiling the VME Slave Module with new address width.

There are 3 different pages for local side. This brings flexibility on dynamic addressing capabilities of VME Bus structure. For instance, the user could select 16 bit addressing for page 1, 24 bit addressing for page 2 and 32 bit addressing for page 3. All of the pages could have different base addresses and address offsets added to those base addresses. Thus, the VME slave module in fact combines three different Slave Cards in one card.

The data transaction widths accepted by the VME slave module can be set. For instance, the user could choose just 16 bit and 32 bit wide data bus transactions. Thus 8 bit transactions to VME Slave module would be forbidden. This is also an important feature since there could be components on the card which has only 16 bit data access. So, the VME Slave Module protects misusage of the 16 bit wide data accessed components.

The address decoding is implemented in VME Slave Module. Therefore, the local side does not need to decode address modifier codes of VME.

There is a defined simple “Local Bus” where the user only needs to connect address and data bits to this bus and implements a basic control unit for the added components to system. This defined “Local Bus” is a big advantage for user.

The VME Slave Module includes interrupter that is capable of driving any of 7 levels of interrupt signals on VME bus. Moreover, in interrupt handling cycle, the VME Slave Module could provide 8, 16 or 32 bit status_id information to interrupt handler on VME bus, which defines the cause of the interrupt request.

The VME Slave Module has capability to terminate interrupt request by Release on Acknowledge or Release on register Access mechanisms.

Finally, the VME slave module is totally compliant to VME64 [3] and VME64X [4] standards.

3.5 COMPARISON OF DESIGNED VME SLAVE MODULE WITH AVAILABLE AND PROPOSED IP’S

In this section a detailed comparison of designed VME Slave Module with other available cores will be given.

3.5.1 Features Comparison of VME Slave Module with Available IP’s

At first, the VME Slave Module is compared with IP solutions available in market [19] - [17]. The comparison is shown in Table 3-4 and Table 3-5.

As it was shown in Table 3-4 and Table 3-5, the VME Slave Module is designed with more features than any of the available IP solutions in market. On the following paragraphs the main advantages of the VME Slave Module are explained.

The VME Slave module provides 3 different pages on local side with their own base addresses, address offsets and bus widths defined for VME Bus. Thus, VME slave Module not only handling VME address decoding by itself, but also supporting three different solution cases with just one solution. Therefore, it could be said three cards could be controlled with just one VME Slave Module. This is a great advantage.

Table 3-4 VME Side Comparison of Designed VME Slave Module and In Market IP's

Company		**IniCore**			**MemecCore**		**Intelop**				
IP Name		**A24D16 Slave Controller**	**A24D32 Slave Controller**	**A32D32 Slave Controller**	**MC-ACT-VME2416**	**MC-ACT-VME32**	**VME Slave Core**	**Proposed [17]**	**Proposed [16]**	**Proposed [5]**	**Designed VME Slave Core**
Data Size	**8**	x	x	x	x	x	x	x			x
	16	x	x	x	x	x	x	x	x		x
	24							x			x
	32		x	x		x	x	x	x	x	x
Address Size	**16**	x	x	x	x	x	x				x
	24	x	x	x	x	x	x		x		x
	32			x		x	x	x	x	x	x
Cycles	**SCT**	x	x	x	x	x	x	x	x	x	x
	Block						x	x			x
	RMW		x	x			x	x			x
	AO										x
Interrupter	**8**	x	x	x	x	x	x	x			x
	16				x	x	x	x			x
	32					x	x	x			x
Interrupt Termination Mechanism	**ROAK**	x	x	x	x	x	x	x			x
	RORA		x	x							x
Rescinding DTACK				x	x						x
External Transceiver Control			x	x	x	x	x	x	x	x	x
VME_BERRn					M	M	M				D
VME_RETRYn											x
VME_SYSRESETn		M	M	M	M	M	M				D

M:	Monitor
D:	Drive
x:	Available

Table 3-5 Local Side Comparison of VME Slave Module and In Market IP's

	VME Slave Module (Designed)	**In Market IP Solutions** [16] , [19] - [17], [18]
Address Decode	o Inside Core	o User Side
User Watchdog Timer	o Yes	o No
Page	o Yes (3 – with different base addresses)	o No

On the other hand, there are generic registers to decide which data size and type transactions are allowed for VME Slave Module. This is mandatory for special applications and prevents the misusage of VME Bus by illegal data size transactions to and from card.

Moreover, the VME slave modules is capable of terminating data transactions with VME_RETRYn or VME_BERRn signals in communication problems with local side or while the local side is busy with other transactions.

Additionally, VME Slave Module terminated interrupt requests with ROAK or RORA mechanisms defined in standard.

Furthermore, with an intelligent logic implemented on card, system failures could be detected and other VME Bus residents could be informed in fail situations.

3.5.2 Features Comparison of VME Slave Module with Proposed IP's

There are several IP's in literature for VME interfacing [16] [5] [17]. All of those IP's are compared to the designed VME Slave Module. The VME interfaces proposed in references will be named as follows:

[16] will be called P1,

[5] will be called P2,

[17] will be called P3.

First of all, P1 is not a generic IP, it is developed just for the modular VME card and only has user interface with I/O module that could be designed to combine with the general purpose VME Card. There is no local side developed, only 16 bit registers to read and write I/O module interface. Moreover, it is only capable of A24/D16 or A16/D32 interfaces with VME bus and there is no interrupter designed for this IP. Thus, this module is far from reaching the capabilities of the designed VME Slave Module. However, It is specifically designed for the card implemented.

Secondly, P2 is designed just for A32/D32 interface with simple read and write transactions on VME bus implemented. Thus, this approach is a defined solution for just one case. Again, the proposed IP is far from comparing with the developed VME Slave module when the features of VME slave Module is considered.

Finally, P3 is the most capable IP proposed for VME interfacing. Table 3-4 summarizes the VME side comparisons of VME Slave Module and the proposed IP P3.

The local side of proposed VME interface P3 has 32 bit address and 32 bit data width interface with control signals for interrupt requesters. However, as the main VME interface module is investigated in P3, it is a 10 stage state machine. This idea was considered in VME Slave design process. The drawbacks of such a step by step control of VME Bus communication with local side are explained in following sections. Thus, with this proposed method every single cycle on VME Bus would take 30 ns x 10 = 300 ns, since the module functions with 32 MHz global clock

Moreover, in VME Slave Module the metastability problem is also taken into account and every control signal sampled three times to reduce the MTBF (mean time between failure) rate. However, such a problem is not considered in P3 design. This makes the VME Slave Module more redundant IP for space applications as well as P3 with a lot of additional features listed.

3.6 VME SLAVE MODULE

The designed VME Slave Module is capable of providing everything defined on standard. Moreover, it could work as an interrupt requester and it could drive SYSFAILn, BERRn or RETRYn signals to inform other VME Bus residents in the case of system failure or transaction failure. Therefore, the VME Slave Module could be defined as an IP that is responsible for accomplishing data transfers from master cards to slave cards and from slave cards to master cards, in the case of necessity, requesting interrupt, and finally informing system fail or transaction failures to other VME bus residents.

3.6.1 VME Slave Module Architecture

In Figure 3-2, the VME Slave Module Interface is shown. As it is seen in Figure 3-2, the VME Slave Module is working as an interface between inner logic of FPGA, which is defined as local in Figure 3-2, and VME interface. The VME Slave Module is composed of 3 sub-modules in order to provide that interface between VME bus and inner logic of FPGA. Those sub busses are Data Transfer Bus Module, Interrupter Module and Utility Module. The interaction between sub modules and their outer interfaces are shown in Figure 3-3.

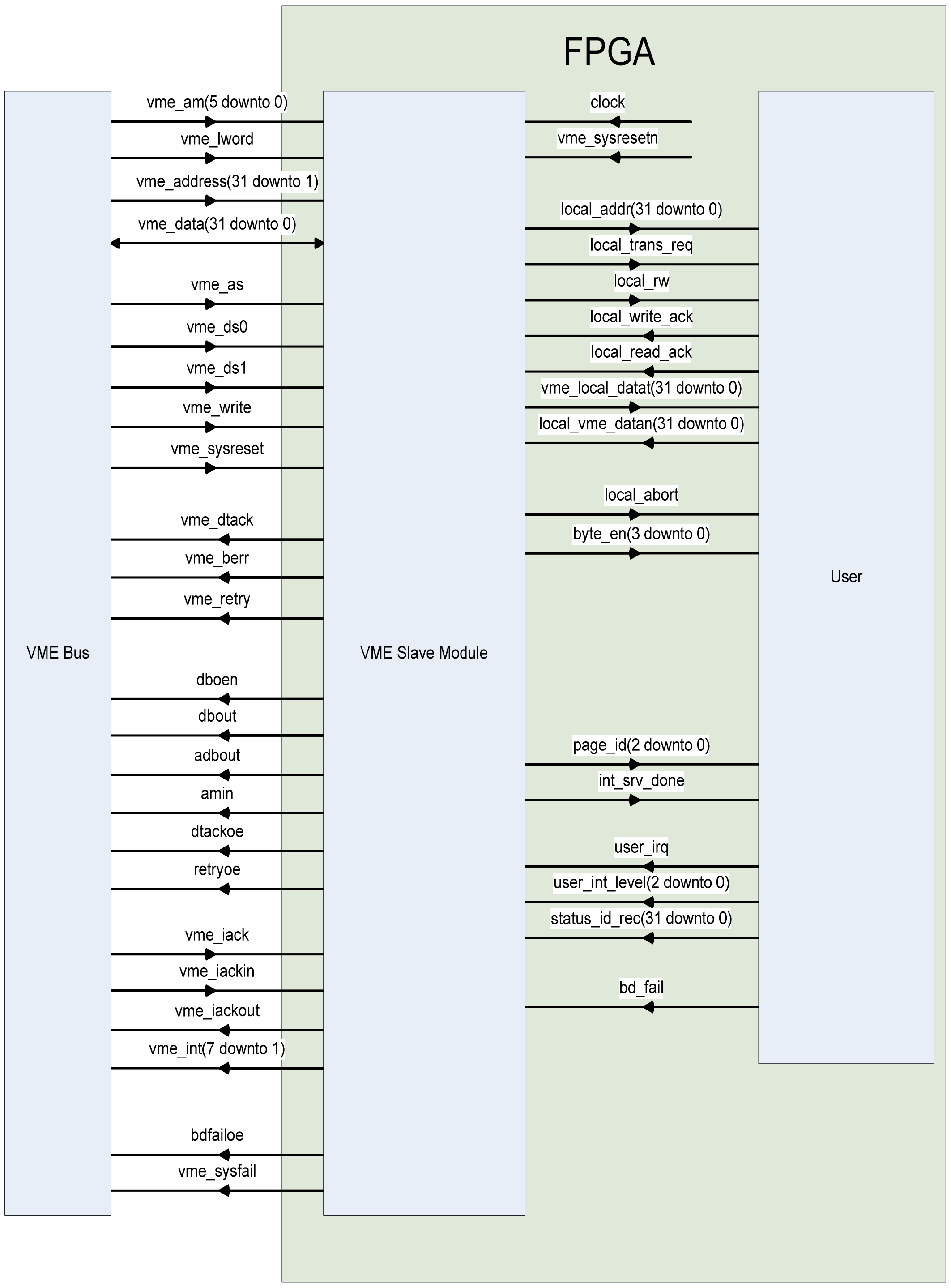


Figure 3-2 VME Slave Module Interface

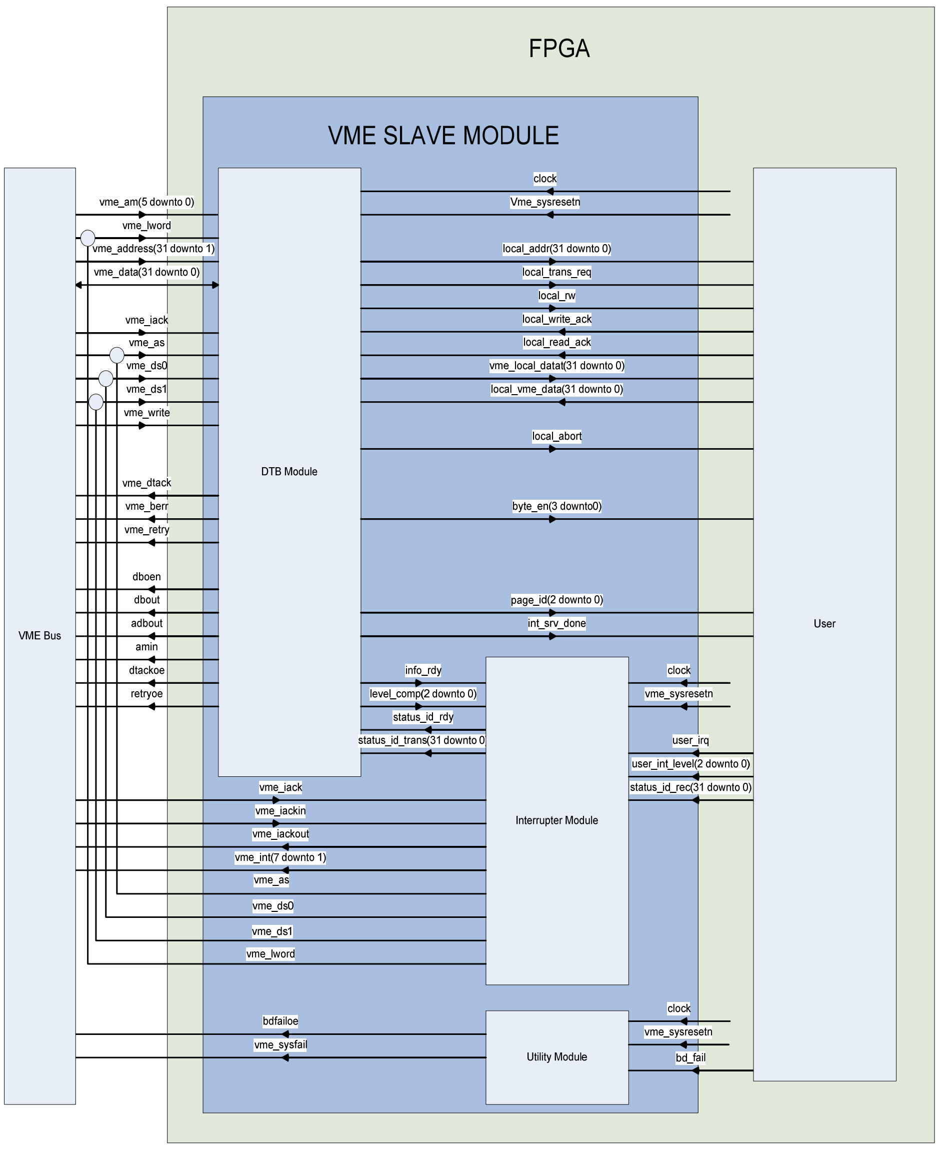


Figure 3-3 VME SLAVE Architecture

The sub-modules are going to be defined in detail on the following sections.

3.6.2 Input / Output Pin Descriptions

The input and output pin descriptions of VME Slave module is defined in Table 3-6.

Table 3-6 VME Slave Module Input Pin Descriptions

Input Signals	
Signal Name	**Description**
Clock	The general clock signal for VME Slave Module
VME_AM[5:0]	Defined in 2.2.2.8 VME Signal Definitions
VME_LWORD	Defined in 2.2.2.8 VME Signal Definitions
VME_Address[31:1]	Defined in 2.2.2.8 VME Signal Definitions
VME_IACK	Defined in 2.2.2.8 VME Signal Definitions
VME_AS	Defined in 2.2.2.8 VME Signal Definitions
VME_DS0	Defined in 2.2.2.8 VME Signal Definitions
VME_DS1	Defined in 2.2.2.8 VME Signal Definitions
VME_WRITE	Defined in 2.2.2.8 VME Signal Definitions
VME_SYSRESET	"System Reset" signal input. This signal is used to reset whole VME Slave Module.
VME_IACKin	Defined in 2.2.2.8 VME Signal Definitions
Local_Write_Ack	Local Write Acknowledge input signal. This signal is used to acknowledge the VME slave module that the written data on vme_local_data bus is captured by the user.
Local_Read_Ack	Local Read Acknowledge input signal. This signal is used to acknowledge the VME slave module that the read data is put on local_vme_databus by the user..
Status_Id_rec[31:0]	"status/id receive" input signal. This signal is used to provide status/id data to the VME Slave Module.
User_Int_level[2:0]	"Interrupt level" input signals. The user side provides the interrupt level that is going to be interrupt requested.
User_IRQ	"Interrupt Request" input signal. This signal is used by user to request interrupt from VME Bus via VME Slave Module.
BD_FAIL	"Board Fail" input signal. This signal is used by user to inform other cards by SYSFAILn signal via VME Slave Module.
local_vme_data[31:0]	User Data in input signals. This signal bus is used by user to carry data to VME Slave Module.

Table 3-7 VME Slave Module Bidirectional Pin Descriptions

Bidirectional Signals	
Signal Name	**Description**
VME_Data[31:0]	Defined in 2.2.2.8 VME Signal Definitions

Table 3-8 VME Slave Module Output Pin Descriptions

Output Signals	
Signal Name	**Description**
VME_BERR	Defined in 2.2.2.8 VME Signal Definitions
local_abort	"local_abort" signal output. This signal is used by VME Slave Module to inform User side that the VME Slave module will terminate VME transaction by VME_RETRYn signal in the following rising edge of clock.
VME_DTACK	Defined in 2.2.2.8 VME Signal Definitions
VME_RETRY	Defined in 2.2.2.8 VME Signal Definitions
VME_IACKout	Defined in 2.2.2.8 VME Signal Definitions
VME_Int[7:1]	Defined in 2.2.2.8 VME Signal Definitions
VME_SYSFAIL	Defined in 2.2.2.8 VME Signal Definitions
DBOEn	"Data Bus Output Enable" output signal. This signal is used to enable VME Buffers to drive the VME Bus.
DBOut	"Data Bus Out" output signal. This signal is used to decide the direction of data flow on VME buffers.
RETRYOE	"RETRY signal Output Enable" output signal. This signal is used to enable the VME Buffer to drive VME_RETRYn signal to VME bus.
ADBOut	"Address Bus Out" output signal. This signal is used to enable VMe buffer to pass VME address information from VME bus to FPGA pins.
DTACKOE	"Data Transfer Acknowledge Output Enable" output signal. This signal is used to enable VME buffer to allow output DTACK signal.
AMIN	"Address Modifier In" output signal. This signal is sued to enable VME buffer to allow AM (5:0) signals to pass through VME buffers.
BD_FAILOE	"Board Fail Output Enable" output signal. This signal is used to enable VME buffer to drive VME_SYSFAILn signal.
local_Trans_req	"Transfer Request" output signal. This signal is used to start a transaction from VME Slave Module to User side inside the FPGA.
local_Addr[31:2]	"Address [31:2]" output signals. These signals are used to carry address from VME Slave Module to User side in FPGA.
local_rw	"Read / Write" output signal. This signal is used to inform user side if it is going to be a read or write cycle inside FPGA.
Int_srv_done	"Interrupt Service Done" output signal. This signal is used by VME Slave Module to inform User side that the requested interrupt cycle is finished successfully.
byte_en[3:0]	This signal is used to describe that nth byte of "local_vme_data" and "vme_local_data" bus must be active.
Page_id	Page id signal. This signal is used by VME Slave module to indicate which page on user side is addressed. This signal is useful when VME slave module is interfacing different modules with different address sizes in FPGA.
vme_local_data[31:0]	User Data Out output signals. This signal bus is used by VME Slave module to carry data to User side.

3.6.3 Functional Description of VME Slave Module

VME Slave Module is mainly responsible for processing data transactions between VME masters and slaves. This is achieved by interfacing between VME bus and inner logic of FPGA. A standard bus is defined for internal logic communication. This bus is inspired from VME bus and designed specially for VME Slave Module. The VME Slave Module is capable of 16/24/32 bit addressing and 8Even/8Odd/16/24/32 bit data transactions. All of those bus widths are supported both single cycle and block transfer cycles. Moreover, unaligned data transfers (24 bit) can be achieved via VME Slave module.

On the other hand, VME Slave module can work as an interrupt requester. The VME Slave module can request any of 7 levels of interrupt and on the interrupt handling cycles, it could provide status/id data to interrupt handlers.

Moreover, VME Slave is capable of monitoring and driving VME_SYSFAILn signal which means it could control system functionality and if the card has a hazardous situation for VME Bus communication; it could inform other VME bus residents.

VME Slave Module also monitors the transactions and informs both master and local side if any transaction violation occurs with local_abort, VME_BERRn AND VME_RETRYn signals.

All of those concepts are handled in 3 sub modules (DTB Module, Interrupter Module and Utility Module) of VME Slave Module. The sub modules will be explained in following sections.

3.7 PROPOSED USER SIDE INTERFACE FOR VME SLAVE MODULE (LOCAL BUS)

There is a need for interior FPGA bus to communicate with VME Slave module. Therefore, there is a need to define and design a user side interface for proper functioning of VME Slave Module. The designed local bus is the main bus inside the FPGA. On this bus only the VME Slave Module is master and all other modules connecting to Local bus must be on slave architecture. The bus is designed to allow VME Bus to communicate with other modules in FPGA. The primary objective of the user side bus is that it should be simple and fast.

3.7.1 Structure of Local Bus

The Local bus can be connected to unlimited number of other local modules. The local bus structure can be seen in Figure 3-4. All of the local (user) signals are multiplexed and each local module is connected to those signals.

The local modules may monitor desired size of address and data signals. This is useful for FPGA place and route sessions. Because unnecessary connections limit the FPGA capabilities and timing issues should be considered in those cases.

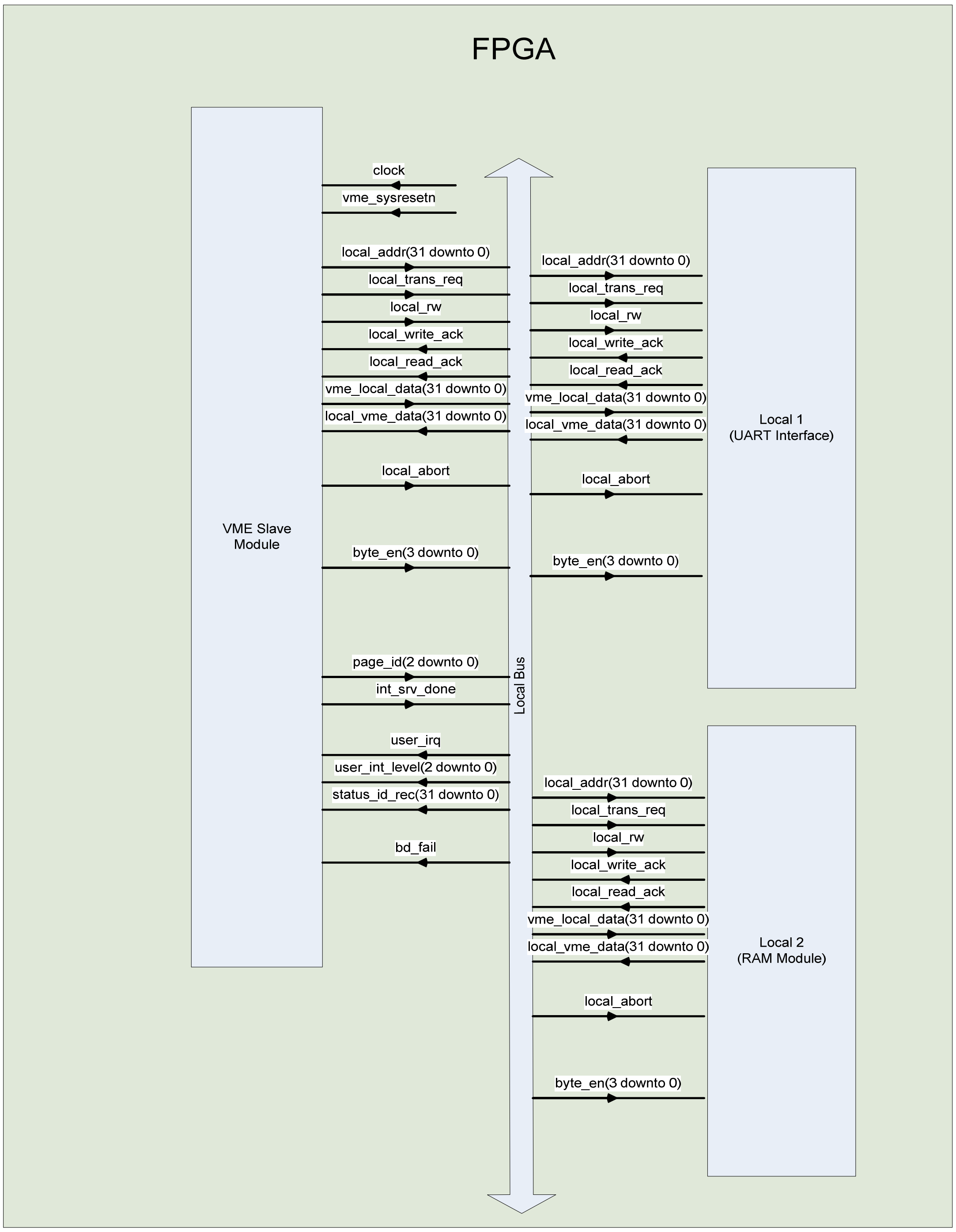


Figure 3-4 Local Bus Structure

3.7.2 Local Bus Signal Definitions

The Local bus signals are explained in Table 3-9.

Table 3-9 Local Bus Signal Definitions

Signal Name	Description
BD_FAIL	"Board Fail" input signal. This signal is used by user to inform other cards by SYSFAILn signal via VME Slave Module.

byte_en[3:0]	This signal is used to describe that nth byte of "local_vme_data" and "vme_local_data" bus must be active.
	Local_byte(0) => Data Bits(32:24)
	Local_byte(1) => Data Bits(24:16)
	Local_byte(2) => Data Bits(15:8)
	Local_byte(3) => Data Bits(7:0)

Int_srv_done	"Interrupt Service Done" output signal. This signal is used by VME Slave Module to inform User side that the requested interrupt cycle is finished successfully.
local_abort	"local_abort" signal output. This signal is used by VME Slave Module to inform User side that the VME Slave module will terminate VME transaction by VME_RETRYn signal in the following rising edge of clock.
local_Addr[31:2]	"Address [31:2]" output signals. These signals are used to carry address from VME Slave Module to User side in FPGA.
Local_Read_Ack	Local Read Acknowledge input signal. This signal is used to acknowledge the VME slave module that the read data is put on local_vme_databus by the user..
local_rw	"Read / Write" output signal. This signal is used to inform user side if it is going to be a read or write cycle inside FPGA.
local_Trans_req	"Transfer Request" output signal. This signal is used to start a transaction from VME Slave Module to User side inside the FPGA.
local_vme_data[31:0]	User Data in input signals. This signal bus is used by user to carry data to VME Slave Module.
Local_Write_Ack	Local Write Acknowledge input signal. This signal is used to acknowledge the VME slave module that the written data on vme_local_data bus is captured by the user.
Page_id	Page id signal. This signal is used by VME Slave module to indicate which page on user side is addressed. This signal is useful when VME slave module is interfacing different modules with different address sizes in FPGA.
Status_Id_rec[31:0]	"status/id receive" input signal. This signal is used to provide status/id data to the VME Slave Module.

Cont’d Table 3-9 Local Bus Signal Definitions

Signal Name	Description
User_Int_level[2:0]	“Interrupt level” input signals. The user side provides the interrupt level that is going to be interrupt requested.
User_IRQ	“Interrupt Request” input signal. This signal is used by user to request interrupt from VME Bus via VME Slave Module.
vme_local_data[31:0]	User Data Out output signals. This signal bus is used by VME Slave module to carry data to User side.

3.7.3 Local Bus Cycles

There are two types of cycles are defined for Local Bus. Those are read/write cycles and interrupt cycles.

3.7.3.1 Local Bus Read/Write Cycles

The most common cycle on Local bus is read/write cycles. Those cycles are needed for data transfer from or to VME Slave Module. Since the VME Slave Module is the only master on Local bus, the transfer initiator is always VME Slave Module and no arbiter is needed for Local bus.

For Read cycles, VME Slave Module should place local_addr (31:2), local_r_w and byte_en (3:0) signals to appropriate states. Afterwards, VME Slave Module drives local_trans_req to high state. Then, local module provides the requested data and put the data on local_vme_data bus and acknowledge VME Slave module by de-asserting local_read_ack signals. With the rising edge of local_read_ack signal, VME Slave module captures the read data and terminates transaction by driving local_trans_req to low. Thus, the read transaction is terminated successfully. The local bus read cycle is shown in Figure 3-5.

For write cycles, VME Slave Module should place local_addr (31:2), local_r_w, vme_local_data (31:0) and byte_en (3:0) signals to appropriate states. Afterwards, VME Slave Module drives local_trans_req to high state. Then, local module captures the data on vme_local_data bus and acknowledge VME Slave module by de-asserting local_write_ack signals. With the rising edge of local_write_ack signal, VME Slave module terminates transaction by driving local_trans_req to low. Thus, the write transaction is terminated successfully. The local bus write cycle is shown in Figure 3-6.

If there local side does not respond via local_read_ack or local_write_ack signals within a defined period. The VME Slave Module terminates the transaction with asserting local_abort signal. Moreover, the VME Slave module informs VME master with VME_RETRYn and VME_BERRn signals about the transaction failure.

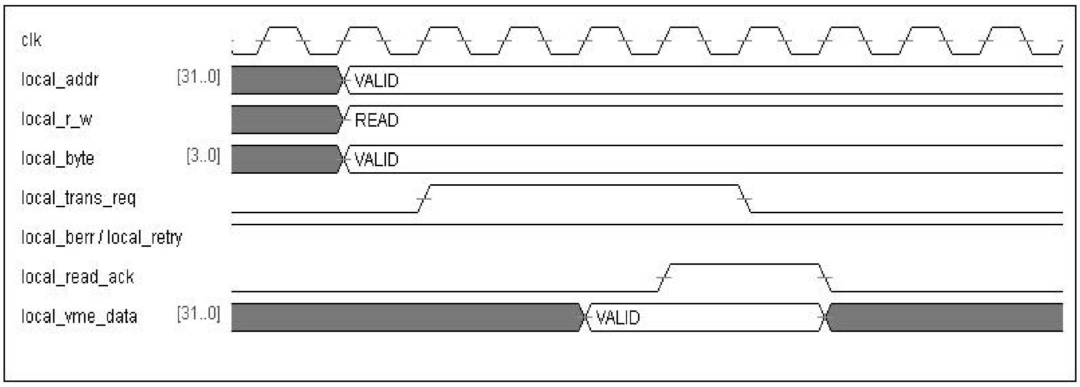


Figure 3-5 Local Bus Read Cycle

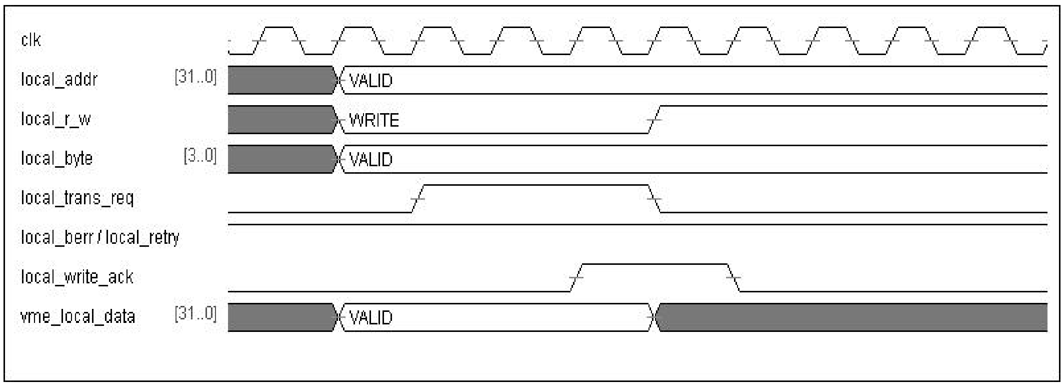


Figure 3-6 Local Bus Write Cycle

3.7.3.2 Local Bus Interrupt Cycles

Local bus interrupt cycles are designed to request interrupt routines from VME Slave Modules. With these cycles, VME Slave Module generates interrupt requests on VME Bus and informs the result, if it is successful or not to the local module.

Local module provides local_int_level (3:0) and statud_id_rec (31:0) signals. Afterwards, it de-asserts local_irq signal and waits VME Slave module to request interrupt on VME bus and then responds interrupt handling cycle and return local module by de-asserting int_srv_done signal. Thus, the interrupt routine successfully finishes. The interrupt cycle on Local bus is shown in Figure 3-7.

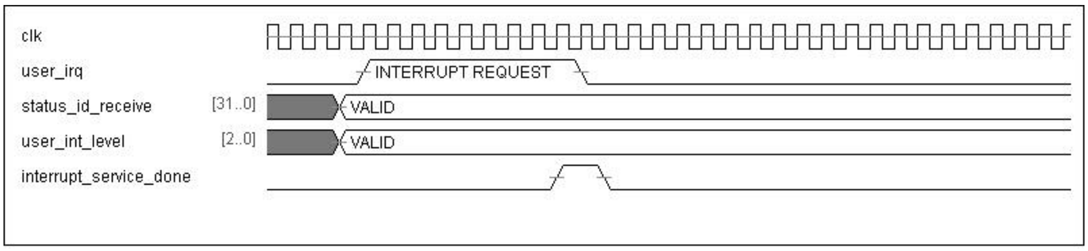
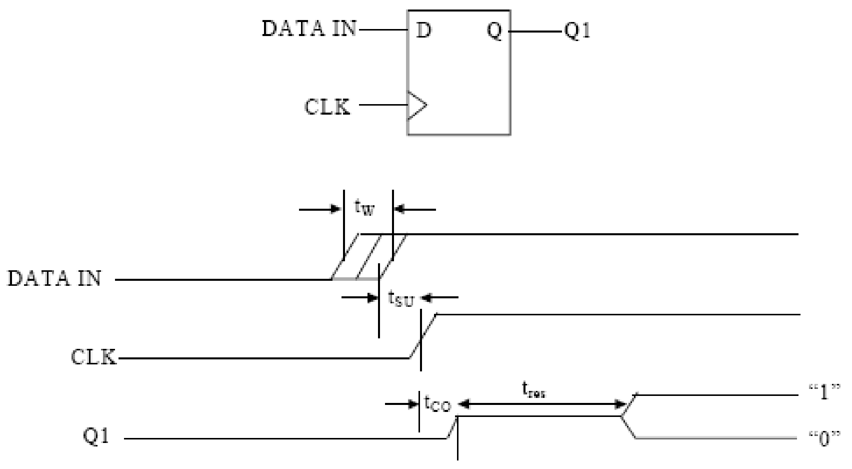


Figure 3-7 Local Bus Interrupt Cycle

3.8 METASTABILITY

Before continuing the design flow of programmable logic design process, an important issue of metastability should be defined. Metastability is defined as the, un-stability on the output of the flip – flop with a small probability, when an asynchronous input is clocked into flip-flop [35]. This problem arises in FPGA based designs when asynchronous designs are established. The metastability situation is visualized in Figure 3-8



t_W = Time window where input transition may cause a metastable condition
t_{SU} = Actual clock setup time for flip-flop
t_{CO} = Actual flip-flop propagation delay
t_{res} = Metastability resolution time

Figure 3-8 METASTABILITY Problem on FPGA Flip - Flop [35]

There were several ideas proposed to overcome this issue. According to Royal and Cheung [36], the clock should be stopped while an asynchronous data is receiving. In this method there should be synchronous handshake signals to decide the time of incoming asynchronous signal and stop the clocks of flip- flops and then start the clocks again. This theory vanish the probability of metastability situation but it is hard to implement and a control logic implementation is mandatory for this application.

On the other hand, the metastability problem can be explained as a reliability criteria as the mean time between failures (MTBF) value. This value is reversely proportional to clock frequency and data frequency. Thus, latching the critical strobe signals three times in pipelined architecture would decrease the sampling frequency of the signal. Moreover it also decreases the chance of occurring metastability at the output of one flip-flip by replacing it with three different flip-flops. Therefore, latching critical signals at least three times would decrease the metastability occurring probability in design.

Since the VME Bus is an asynchronous bus, the VME Slave Module is going to be designed while metastability issues taking into account.

3.9 PROPOSED DESIGN ARCHITECTURES FOR DATA TRANSFER BUS MODULE

The data transfer bus module is responsible for main communication between Local bus and VME bus, and providing status_id information to interrupt handler on VME bus via VME_Data (31:0) pins. Two different approaches were designed for DTB module. Each of them is criticized carefully and one is developed further for VME Slave Module.

In first idea, everything could be synchronous and the transaction from VME to Local side could be controlled step by step. However on the second idea, there would be several processes working in parallel and increasing module performance. While working in parallel it should be noted that not other process would cause problem for another process. Thus, the functions of independent processes should be carefully separated and every process should be aware of the others in one or another way. Both of the design approaches are explained and criticized in this section.

3.9.1 Synchronous Design of DTB (First Approach)

First of all, when normal VME Bus communication sequence is thought there should be a state machine with many states. Thus a first draft of the DTB module is designed and is shown in Figure 3-9. The transaction starts with falling edge of ASn, this validates both address and address modifier code. By the way, the IACKn must be “high” since the DTB should differentiate the interrupt cycles with normal data transaction cycles. Afterwards, the address modifier code is decoded and the type of data transfer is decided. Later on, depending on the cycle if it is a read or write, the necessary local bus transactions are made and finally the master of VME bus is acknowledged by DTACKn assertion on VME bus and VME Bus cycle is terminated. There are 10 steps to finish a VME Bus data write cycle. However for read cycle there should be 11 steps because, there should be one cycle wait to assert DTACKn signal, when the VME Slave Module puts the read data on VME_DATA signals, for data stability on bus. Moreover, there would be metastability problem for strobe signals. Thus, every input data strobe should be latched three times. Therefore, designed first draft of DTB would take 14 or 15 clock cycles for just one VME read or write operation. Moreover there would be watchdog timers in each state in order to check if there are any malfunctioning or not responding modules or functions. This would also increase the logic of the first draft of DTB would occupy and the design would become more complex. Furthermore, there are a lot of junctions and there would be decision mechanisms with a lot of branches. This is also a drawback for communicating with an asynchronous Bus. The only advantage of the first draft of DTB is that the address decoding is handled in DTB and local side is just responsible of responding if it is addressed or not with page_id signals.

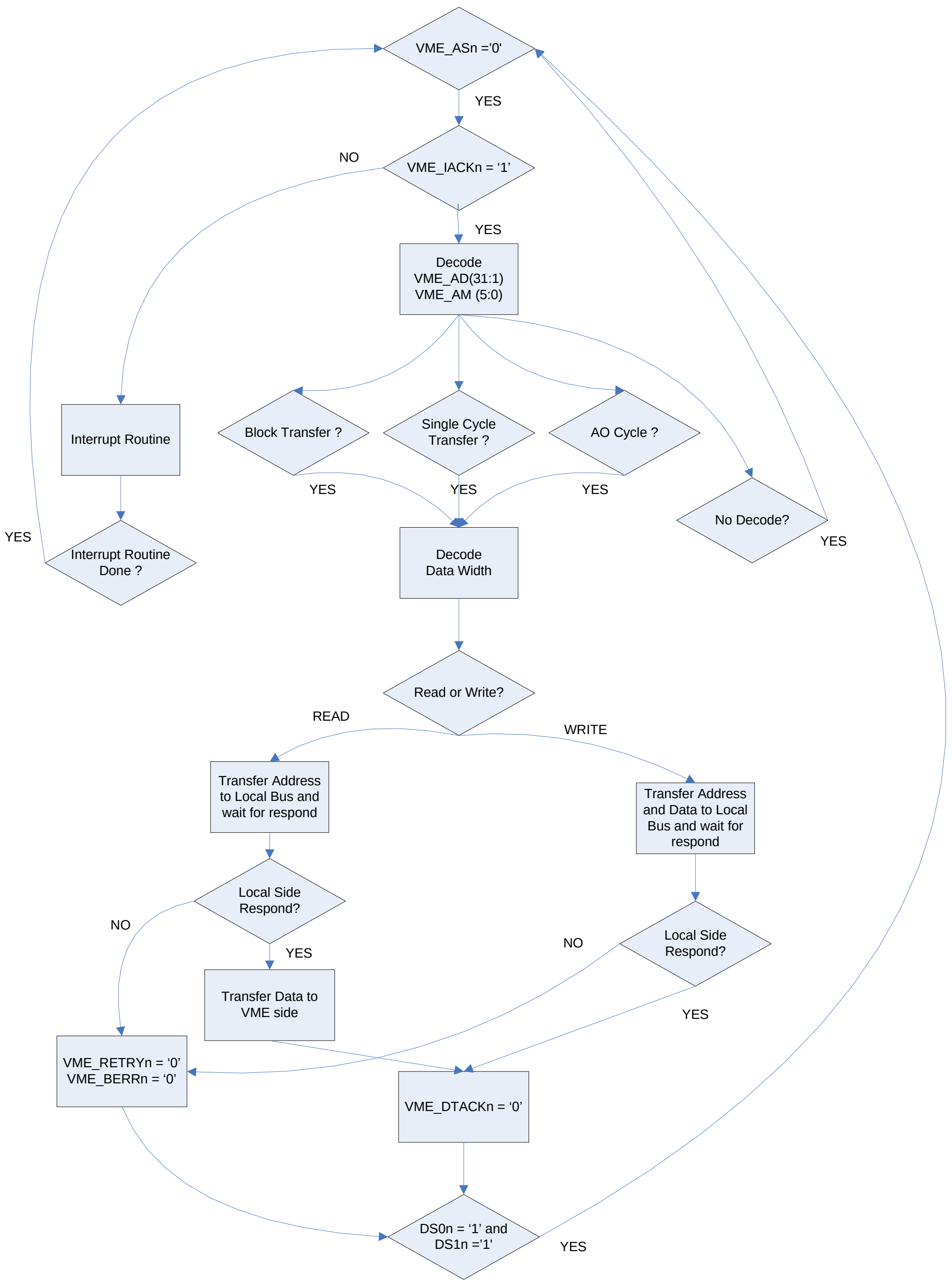


Figure 3-9 First Draft of DTB Module Flow chart

3.9.2 Asynchronous and Synchronous Mixed design of DTB (Second Approach)

Since the FPGA structure is suitable for parallel processing, there would be performance increase if several asynchronous modules are combined with several processes working in parallel. Thus, a new approach was accepted to dividing the functions into several asynchronous blocks and several synchronous processes. The synchronous processes are mandatory for LOCAL_BUS communication. In new approach there are 3 asynchronous blocks, used for address modifier code validating, address validating, data size and type validating. Those asynchronous blocks are shown in Figure 3-10. The asynchronous blocks work continuously and always generate outputs. Those outputs are evaluated by processes when needed. As it is stated above, the functioning mechanism of asynchronous blocks are as follows:

First of all, on one asynchronous block, VME address modifier code is decoded and checked if it is suitable for the defined corresponding page. Moreover, this module also checks if the address is valid for the corresponding page.

Data Transfer Bus module always generates an internal address if there is an allowed page and the VME address hits base address of the defined base address of the corresponding page. The inner address is composed of VME address and defined address offset of the corresponding page.

The final asynchronous module checks if the combination of ds0n, ds1n, lwordn, vme_addr (1) and vme_addr (2); since they are used for encoding data size and type, are valid for the corresponding page.

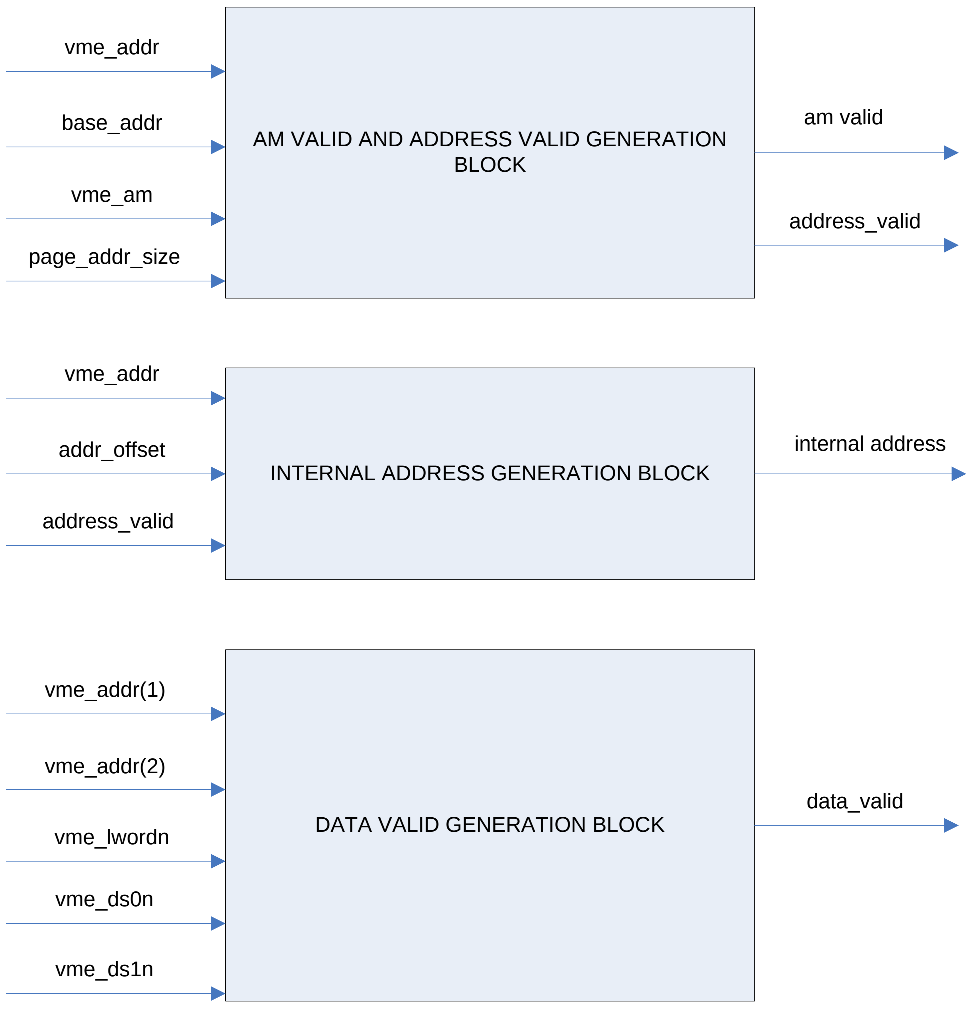


Figure 3-10 DTB Module Asynchronous Blocks

The working mechanisms of processes in DTB are shown in Figure 3-11. The outputs of the asynchronous blocks are evaluated by synchronous processes when the desired conditions encountered. Those processes work in parallel and create outputs in parallel. However, inside the processes, the outputs of other processes are also monitored and each process is aware of other processes and failures. There are

4 different processes, address decode process, data decode process, main check process and interrupt process.

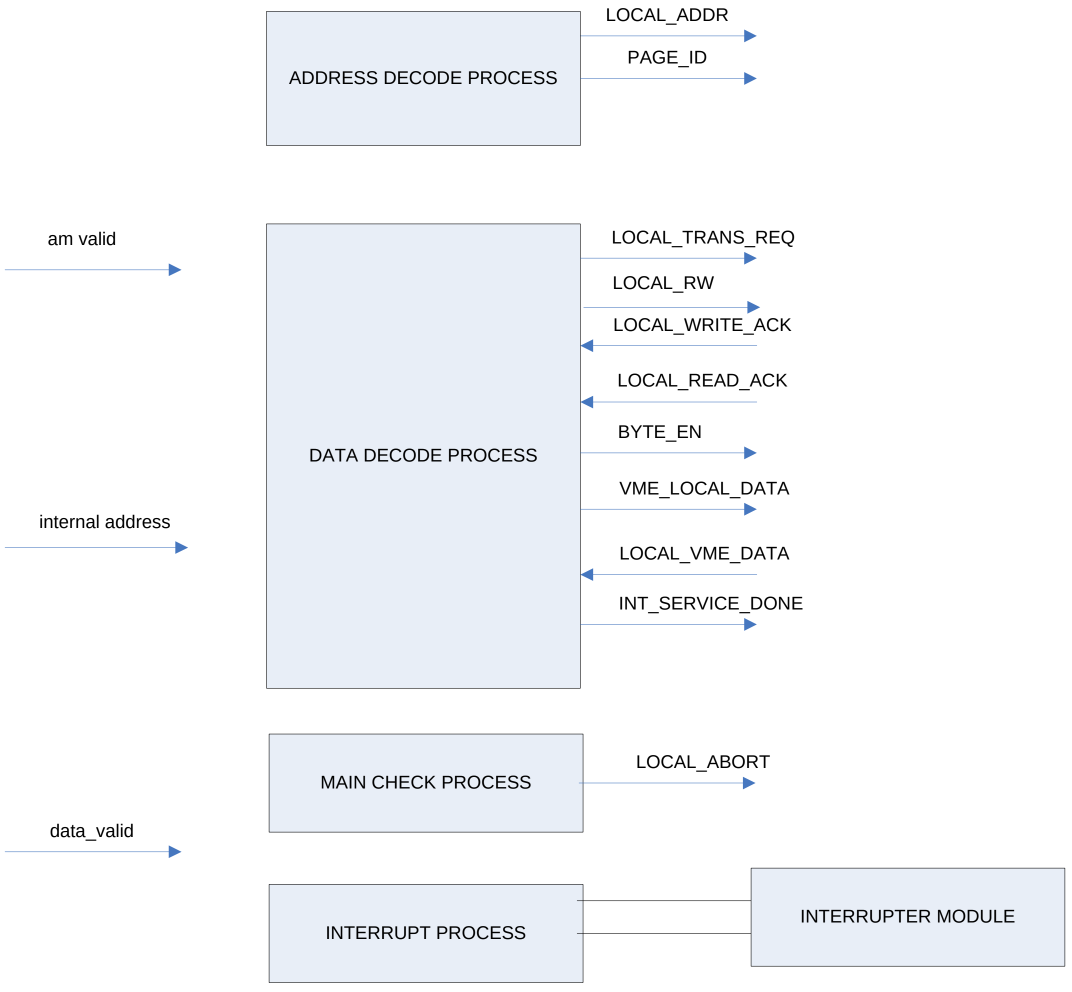


Figure 3-11 DTB Module Parallel Processes

There is a process called address decode state machine, the encoded inner address is driven to the local_addr signals in the light of encoded address modifier codes. Flow chart of this process is shown in Figure 3-12. Checking am valid is mandatory because address modifier codes defines the address size and as it is stated there is dynamic address sizing in VME Bus structure. Therefore, local bus address is always updated with the change on VME address bus. This does not cause a problem because the local transfer request signal is needed for validating local bus address and it is only validated if the VME slave module is selected on VME bus and VME transaction is started with falling edges of ds0n or ds1n.

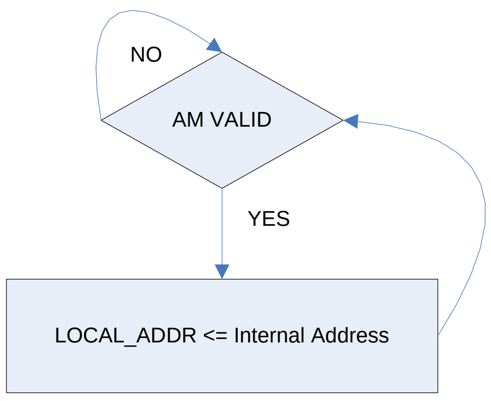


Figure 3-12 DTB Address Decode Process Flow Chart

Now, the second process, which is responsible of data transactions and named data decode process state machine. The flow chart of this process is shown in Figure 3-13. This process starts to work if the following conditions are satisfied:

- If it is not an interrupt cycle (VME_IACKn = ‘1’)

- If address hits VME slave module (VME_ADDR = BASE_ADDRx)
- If data size and type encoding is true
- If address modifier code is true
- If ds0n and ds1n are at correct states

Then this process provides necessary signals for local side. Thus the VME slave module starts a transaction towards local side and then waits the local side to respond back. If there is no error occurs and the local side responds, this process finishes the VME transaction by asserting the vme_dtackn signal and waits the VME master terminates the cycle and the this process de-asserts vme_dtackn and totally terminate cycle on VME bus.

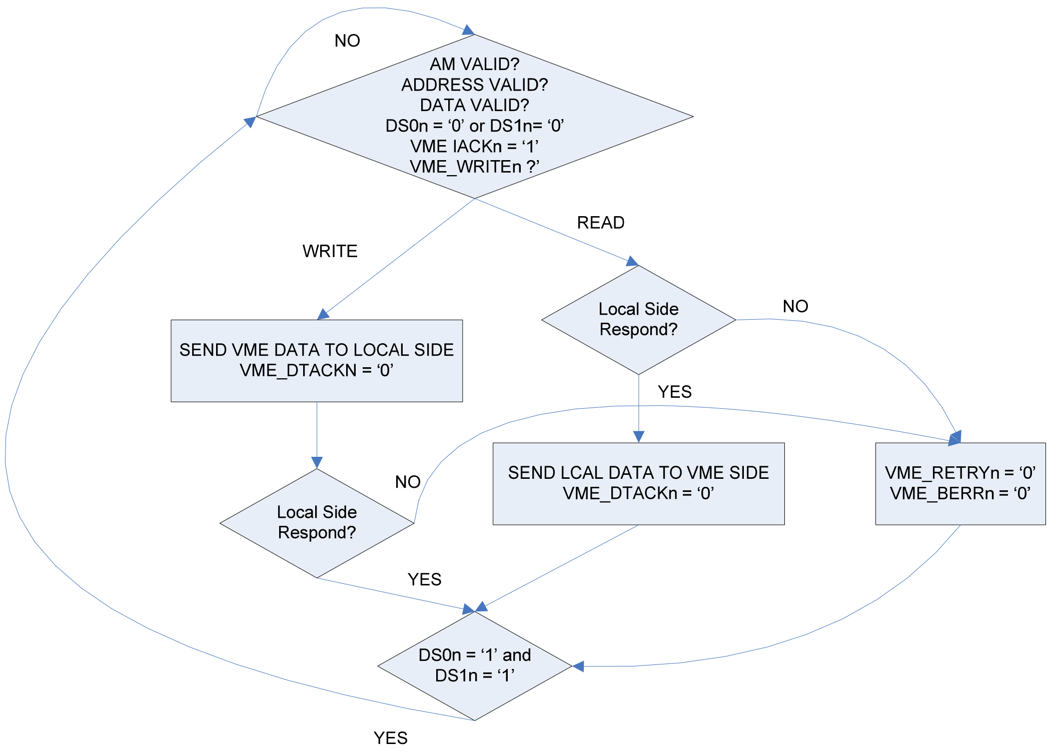


Figure 3-13 DTB Module Data Decode Process Flow Chart

Thirdly, another process works to control if there is an error occurs during the communication with local side. Flow chart of main check process is shown in Figure 3-14. This process is called the main check state machine. There are timers on this process and they started to count down on every cycle begins with a transaction start on VME Bus and addressing VME Slave Module. If the timer reaches zero and there is no response from local side, the process first informs the local side by local_abort and then on the following clock edge the process asserts vme_retryn signal. However, there is no vme_retryn signal on VME64 standard and for backward compatibility; vme_berrn should be asserted to inform the masters of VME64. This vme_berrn assertion is a must, because VME standard requires backward compatibility and, the VME slave module can not decide if it is communicating with a VME64X standard master or not. Thus, after waiting for a defined period, the data transfer bus module asserts vme_berrn signal and informs master that there is an error occurred during transaction.

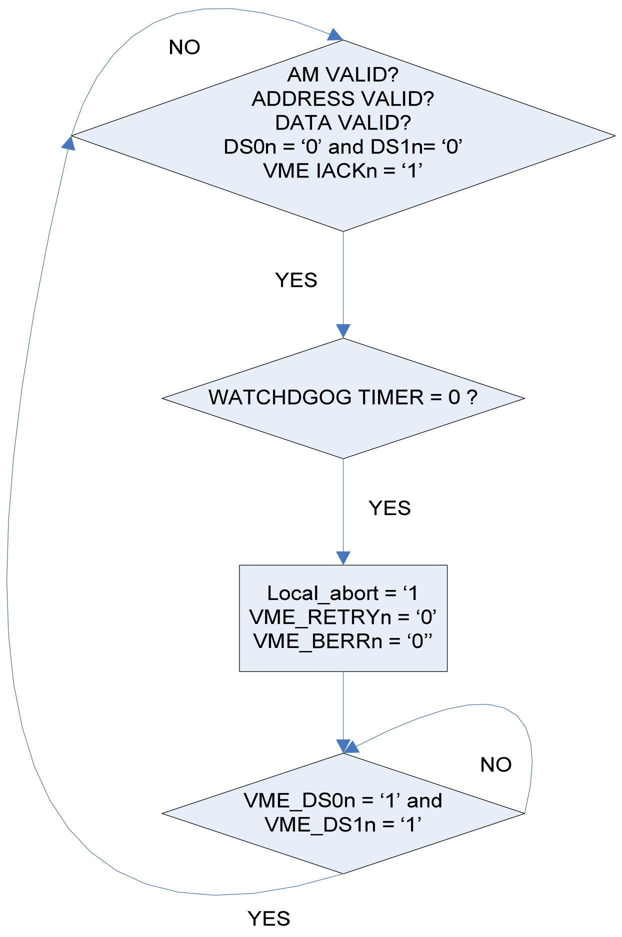


Figure 3-14 DTB Module Main Check Process Flow Chart

Finally, a last process runs for interrupt handling cycles, which is called interrupt state machine. Flow chart of this process is shown in Figure 3-15. It works to carry information from vme_addr (3:1) bits, that indicates the level of interrupt handling to the interrupter module and provide status_id data that the process takes from interrupter module, to the VME data bus.

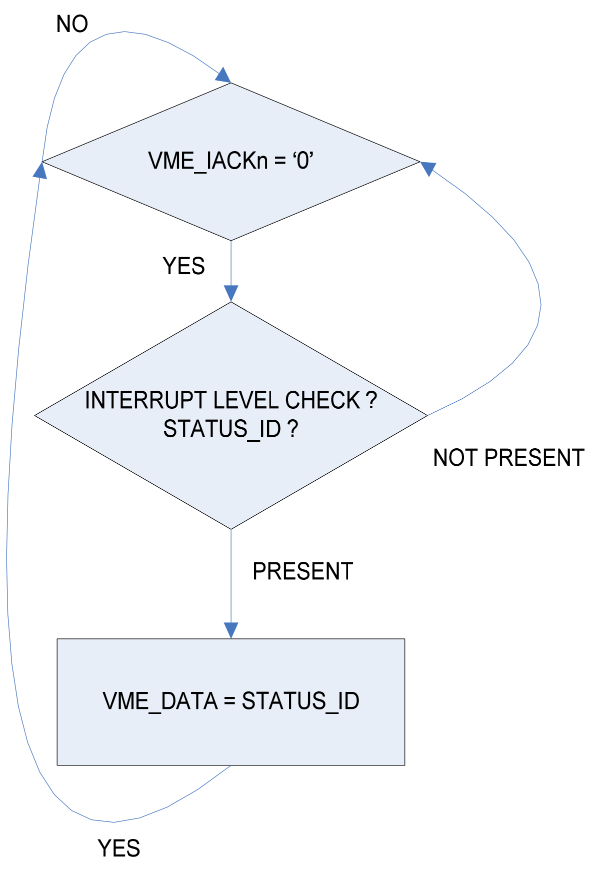


Figure 3-15 DTB Module Interrupt Process Flow Chart

The main advantage of the mixed design approach is its modularity and flexibility to control LOCAL_BUS with VME Bus strobe signals. Moreover, the performance is increased and parallel processing capabilities of FPGA is involved in design. Therefore, the DTB module is implemented by using the asynchronous and synchronous mixed design architecture. Moreover, the other sub-modules are also implemented on asynchronous and synchronous mixed design architecture because of the advantages stated above.

3.10 DESIGN OF DATA TRANSFER BUS MODULE

The Data Transfer Bus module is the heart of VME Slave Module. This module is responsible of interfacing VME Bus and Local Bus. Local Bus is defined in previous sections.

The Data transfer Bus Module interface is shown in

Figure 3-16. The DTB Module is capable of interfacing VME Bus at 16/24/32 bit addressing and 8/16/24/32 bit data widths. Moreover, DTB adds values to VME slave Module, such as driving VME_RETRY and VME_BERRn or defining based addresses for three different pages working independently on LOCAL side. All three different pages have its own base address and address offset. Moreover, each page could have different addressing widths. Thus, DTB behaves as combining three slave modules in one module.

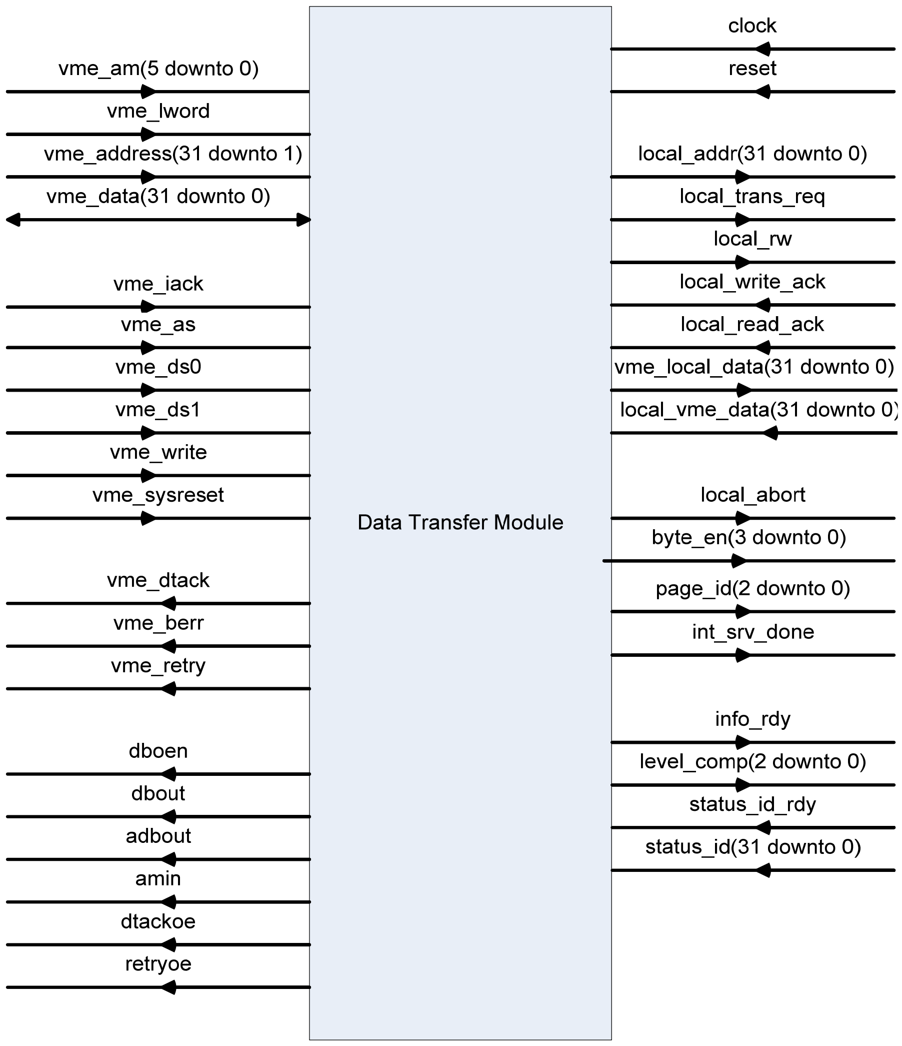


Figure 3-16 Data Transfer Bus Module

3.10.1 Input / Output Pin Descriptions

The DTB Module includes pins interfacing with VME Bus or Local Bus. Most of those signals were described in Table 3-6, Table 3-7 and Table 3-8. The DTB Module specific pins are described in Table 3-10 and Table 3-11.

Table 3-10 DTB Module Specific Input Pin Descriptions

Input Signals	
Signal Name	**Description**
status_id[31:0]	“status/id” input signal.. This signal is used to carry status/id data from Interrupter Module to DTB Module. This data is qualified by rising_edge of status_id_rdy signal.
status_id_rdy	“status/id” ready input signal. This signal is driven by interrupter module to qualify status_id(31:0).

Table 3-11 DTB Module Specific Output Pin Descriptions

Output Signals	
Signal Name	**Description**
Info_rdy	“Info ready” output signal. This signal qualifies the level_comp(2:0) signals that are used to inform interrupter which level on interrupt is handling..
Level_comp[2:0]	“Level compare” output signals. These signals are used to provide information about which level of interrupt is handling on VME Bus to Interrupter Module.

3.10.2 Generic definitions for Data Transfer Bus Module

A generic description is a need in VHDL coding in order to provide flexibility in design. For instance, you have a generic assignment for address size, if you want to interface just 24 bit address you can set this generic to 24 and for 32 bit addressing, you can set this to 32 to compile and synthesis the required address width. As, it is described here, flexibility is important if a core is designed for various applications.

Since VME Slave Module could have several features and in order to support customization, generics are defined for data transfer module.

The list of generics defined for data transfer bus module and their descriptions can be found in Table 3-12.

Table 3-12 Generic Descriptions of DTB Module

Generic Name	Description
vme_addr_size	Vme address size generic is defined as integer. This generic defines the synthesized vme bus address width. 16 => Vme Slave module will be generated with VME_ADDR(15:1) 24 => Vme Slave module will be generated with VME_ADDR(23:1) 32 => Vme Slave module will be generated with VME_ADDR(32:1)
page_addr_size	Page address size generic defined as integer. This generic defines the address width interfacing with local bus. 8 => LOCAL_AD(7:0) 16 => LOCAL_AD(15:0) 23 => LOCAL_AD(23:0) 32 => LOCAL_AD(31:0)
base_addr_1 base_addr_2 base_addr_3	Base address x generic is defined as std_logic_vector. This generic defines the base address of x page. On VME Slave Module there are 3 defined pages. The base addresses and sizes of those3 pages must be defned as not overlapping. Base_addr_x generics size is defined as follows: Ex, vme_addr_size = 24 , page_addr_size = 16 ise base_addr_1 ((24-1) downto 16)
page_1_add_size page_2_add_size page_3_add_size	Page x address size generic is defined as std_logic_vector. This generic defines the allowable address sizes of corresponding page. bit 0 = ‘1’ => 16 bit addressing enabled bit 0 = ‘0’ => 16 bit addressing is disabled. bit 1 = ‘1’ => 24 bit addressing enabled bit 1 = ‘0’ => 24 bit addressing is disabled. bit 2 = ‘1’ => 32 bit addressing enabled bit 2 = ‘0’ => 32 bit addressing is disabled. The user must only set one of 3 bits. Thus, only one address size should be chosen for corresponding page.

Cont’d Table 3-12 Generic Descriptions of DTB Module

Generic Name	Description
page_mask	Page mask generic is defined as std_logic_vector. This generic defines the enabled pages. bit 0 = ‘1’ => page1 is enabled bit 0 = ‘0’ => page1 is disabled bit 1 = ‘1’ => page2 is enabled bit 1 = ‘0’ => page2 is disabled bit 2 = ‘1’ => page3 is enabled bit 2 = ‘0’ => page3 is disabled
add_offset_1 add_offset_2 add_offset_3	Address offset x generic is defined as std_logic_vector. This signal adds offset to the incoming address and address the local module with the new address at the corresponding page. Ex; page_addr_size = 16 ise add_offset_1 ((16-1) downto 0) local_addr = vme_addr(15:1) + add_offset_1
data_modes	Data modes generic is defined as std_logic_vector. This generic defines the data widths that the VME Slave Module is capable of. By setting bits of this generic, VME Slave Module responds the required VME trasactions. bit 0 = ‘1’ => 8 bit odd data is enabled bit 0 = ‘0’ => 8 bit odd data is disabled bit 1 = ‘1’ => 8 bit even data is enabled bit 1 = ‘0’ => 8 bit even data is disabled bit 2 = ‘1’ => 16 bit data is enabled bit 2 = ‘0’ => 16 bit data is disabled bit 3 = ‘1’ => 32 bit data is enabled bit 3 = ‘0’ => 32 bit data is disabled bit 4 = ‘1’ => 24 bit unaligned data is enabled bit 4 = ‘0’ => 24 bit unaligned data is disabled bit 5 = To be determined (Future use) bit 6 = To be determined (Future use) bit 7 = To be determined (Future use) bit 8 = To be determined (Future use)
berr_en	Berr enable generic is defined as std_logic. This generic defines, if the VME Slave Module is going to terminate VME transactions with VME_BERRN signal or not. berr_en = ‘1’ => VME_BERRn function is enabled. berr_en = ‘0’ => VME_BERRn function is disabled.

3.11 PROPOSED DESIGN ARCHITECTURE FOR INTERRUPTER MODULE

Interrupt Module is responsible of interrupt requests from VME Bus. There is one process called interrupt state machine. This process is responsible of both of "interrupt requesting" and "interrupt handling cycle". This process is shown in Figure 3-17.

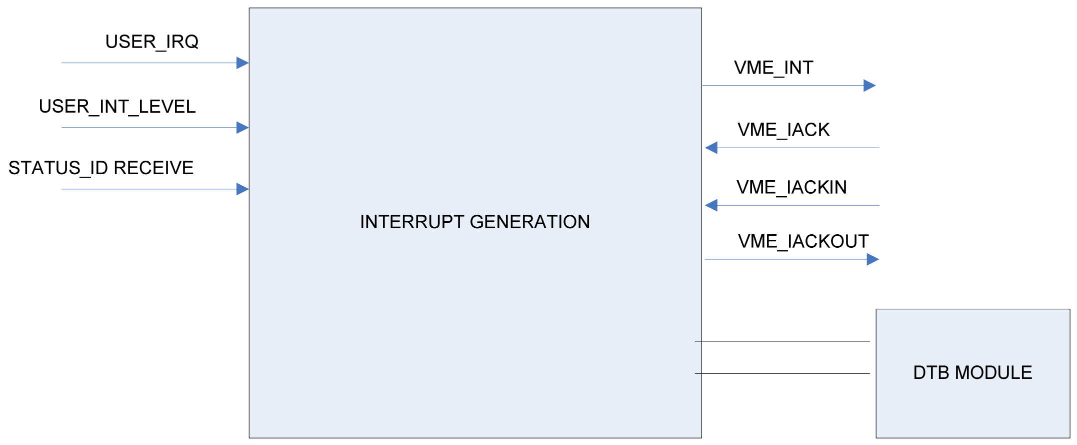


Figure 3-17 Interrupter Module Interrupt Process

The flow chart of interrupt process is shown in Figure 3-18. The process always monitors if there is any user interrupt request. If so, the module creates the appropriate VME interrupt level on VME Bus and waits the interrupt handler, handles the interrupt request. When the interrupt handler receives the interrupt requests, it starts the interrupt daisy chain. During daisy-chain if the IACKINn asserted and the level of interrupt handing is put on VME address (3:1). When the

IACKn is asserted, the Interrupter checks if the level of interrupt handling is the same level of interrupt requested by module, if the requested interrupt level and the handling interrupt level are same; the process goes for another check mechanism. On the second check mechanism, the process checks if the module could provide a status_id as the requested width of status_id by interrupt handler. Then, if the second check also passes, the interrupter module provides status_id and terminates the interrupt request depending on the ak_ra generic defined. If the Release on Acknowledge was chosen, the interrupt request is terminated when the first check is completed and the interrupt module understands the interrupt handler, handles the interrupt request the interrupter module created. On the other hand, if the Release on Register Access mechanism was chosen, the interrupter module doesn't terminate the interrupt request until the interrupt handler gets the status_id of interrupt requester and terminates the interrupt handling cycle.

On the other hand, the interrupter module also participates in interrupt daisy chain. If there is an iackin assertion, and there is no request from interrupter module, or the level of interrupt handling is not the same level that interrupter module requests, the interrupter module asserts the iackoutn signal and informs the next resident on VME bus about interrupt handling cycle.

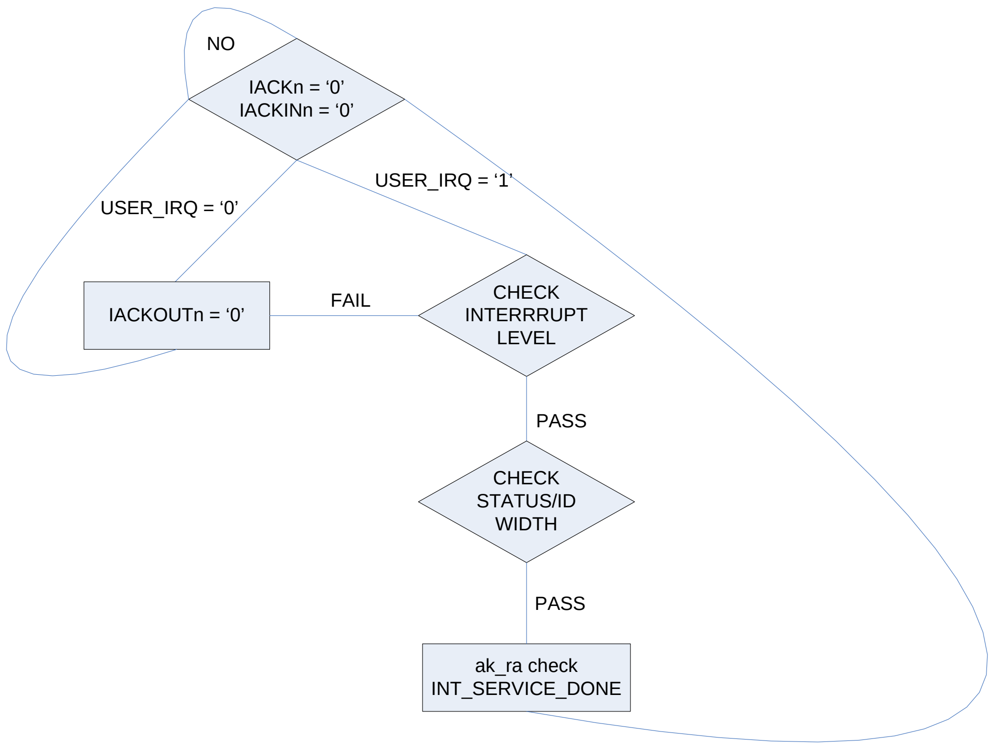


Figure 3-18 Interrupter Module Interrupt Process Flow Chart

3.12 DESIGN OF INTERRUPTER MODULE

The Interrupter Module is responsible of requesting interrupts on VME bus with requests from Local Bus.

The Interrupter Module interface is shown in Figure 3-19. The Interrupter Module is capable of requesting all 7 levels of interrupts on VME Bus, and could provide 8/16/32 bit status_id information on interrupt handling cycles. Moreover, the interrupter module supports both interrupt request termination mechanisms, release on acknowledge (ROAK) and release on register access (RORA).

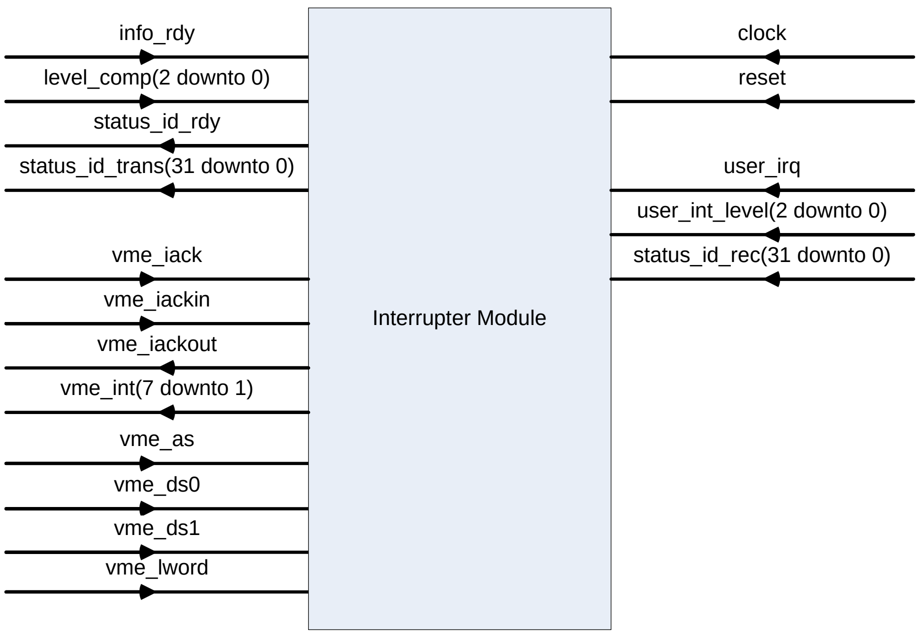


Figure 3-19 Interrupter Module

3.12.1 Input / Output Pin Descriptions

The Interrupter Module includes pins interfacing with VME Bus or Local Bus. All of those signals were described in Table 3-6, Table 3-7, Table 3-8, Table 3-10 and Table 3-11.

3.12.2 Generic definitions for Interrupter Module

The list of generics defined for interrupter module and their descriptions can be found in Table 3-13.

Table 3-13 Generic Descriptions for Interrupter Module

Generic Name	**Description**

ak_ra	Release on Acknowledge or Release on Register Access generic is defined as std logic. This generic defines the termination of interrupt requesting. There are two ways to terminate interrupt request on VME standard, on one way, the interrupt requester terminates requesting with IACKn assertion, which means acknowledge detection, on another way the interrupt requester terminates requesting with register access which means after providing status_id information to the interrupt handler.
	ak_ra = ‘0’ => Release on Acknowledge is nabled
	ak_ra = ‘1’ => Release on register Access is enabled.

data_size	Data size generic is defined as three bits std_logic_vector. This generic defines the lenth of the status_id information that the interrupter module provides. Moreover, due to the VME standard, if a slave provides less length status_id information than the requested length of status_id information by interrupt handler, this slave could respond the interrupt handler, on the other hand, if the slave provides wider status_id information than the interrupt handler requests ,this slave could not provide status_id information to interrupt handler. For instance, if a slave provides 8 bit status_id, it could respond to 8/16 or 32 bit status_id requests by interrupt handler. Howoever, if the slave provides 16 bit status_id, it could not respond to 8 bit status_id request by interrupt handler. Bit 0 = ‘0’ => 8 bit status_id is enabled. Bit 0 = ‘1’ => 8 bit status_id is disabled. Bit 1 = ‘0’ => 16 bit status_id is enabled. Bit 1 = ‘1’ => 16 bit status_id is disabled. Bit 2 = ‘0’ => 32 bit status_id is enabled. Bit 2 = ‘1’ => 32 bit status_id is disabled. Only one of three lengths should be enabled for proper functioning.

3.13 PROPOSED DESIGN ARCHITECTURE OF UTILITY MODULE

The Utility module is responsible of driving VME_SYSFAILn signal on VME Bus. Thus, there is only one process running in Utility module which monitors, bd_fail signal from local side. If the bd_fail signal is de-asserted, the utility module asserts the VME_SYSFAILn signal until the bd_fail signal is asserted on local side.

3.14 DESIGN OF UTILITY MODULE

The Utility Module is responsible of driving VME_SYSFAILn signal on VME Bus if the necessary conditions occurred. This feature is not defined in VME Standard but the "VME_SYSFAILn signal could be driven by any module" statement is in [3] and [4]. Therefore, if an intelligent component can control VME slave module, it could inform the total VME bus system in the presence of fail condition that could affect total Bus structure. The Utility module is shown in Figure 3-20.

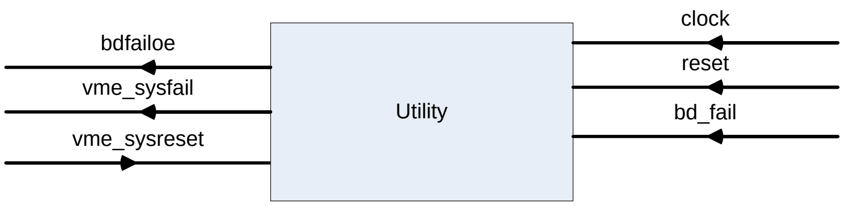


Figure 3-20 Utility Module

3.14.1 Input / Output Pin Descriptions

The Utility Module includes pins interfacing with VME Bus or Local Bus. All of those signals were described in Table 3-6 and Table 3-7.

3.15 DESIGN QUALITY

After the VME SLAVE Module VHL coding is finished, designed VHDL code is tested by HDL Designer Series Design Check. This is a code style check mechanism tool which has a rule set defined for commonly accepted coding styles for VHDL. The version used for checking is ver05.3. In

Table 3-14, on the overall of %84 of the written VHDL codes met the rules. However, the main factor to decrease the overall performance is the GENERICS defined for user flexibility in DTB Module. The defined rule set does not allow GENERICS to define because there are unused if and else branches are occurred because of the non available GENERIC definitions. Therefore, the overall of % 84 design quality performance is a good value for a customizable design of VME Slave Module.

Table 3-14 Design Quality Report of VME Slave Module and Sub Modules

Module	**Quality (%)**
DTB Module	85
Interrupter Module	92
Utility Module	100
Overall	**84**

3.16 TEST MODULES

Several test modules are implemented in order to test the Local bus interface and the communication path between FPGA and peripheral devices. Those modules are bridges between Local bus and peripheral device interface. Here are the implemented interfaces:

3.16.1 RAM Interface

A RAM block is implemented in FPGA in order to test the FPGA and VME functionality. The RAM is memory area is 100 Kbytes. VME Slave Module lets the VME master directly reaches RAM module and initiate write or read transactions. The RAM works synchronously and the clock frequency of RAM block is 33.33 MHz as the VME Slave Module. RAM Test Modules I/O connection is shown in Figure 3-21.

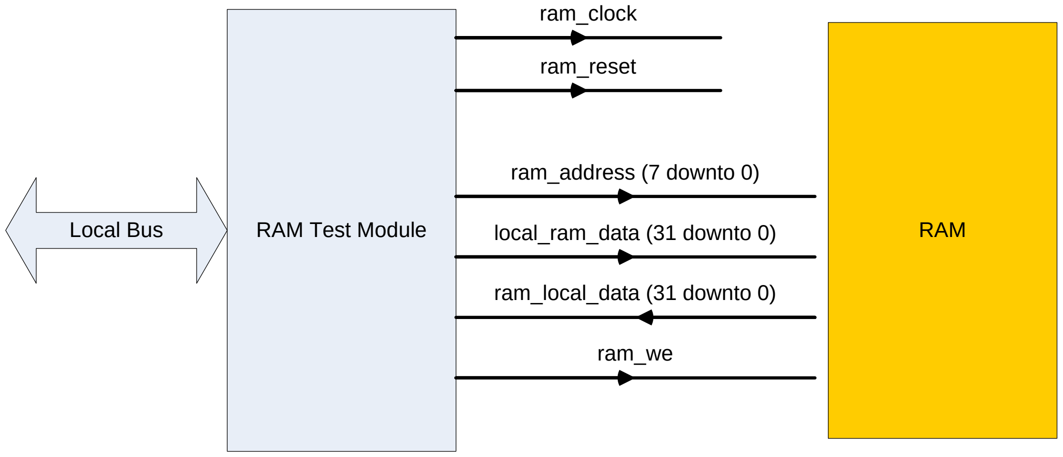


Figure 3-21 RAM Test Module

3.16.2 RS-485 Interface

This interface implemented in order to communicate with RS-485 interface component. FPGA communicates with Texas Instruments SN65HVD08 product numbered serial controller on UART protocol. A test module is implemented to bridge between UART protocol and Local bus of VME Slave Module. VME master could send data over RS-485 interface over VME Slave Module. Moreover, the received data on RS-485 serial port are stored in FIFO of UART Test Module. Therefore, VME Master can read the received data's from RS-485 port. UART Test Module is shown in Figure 3-22

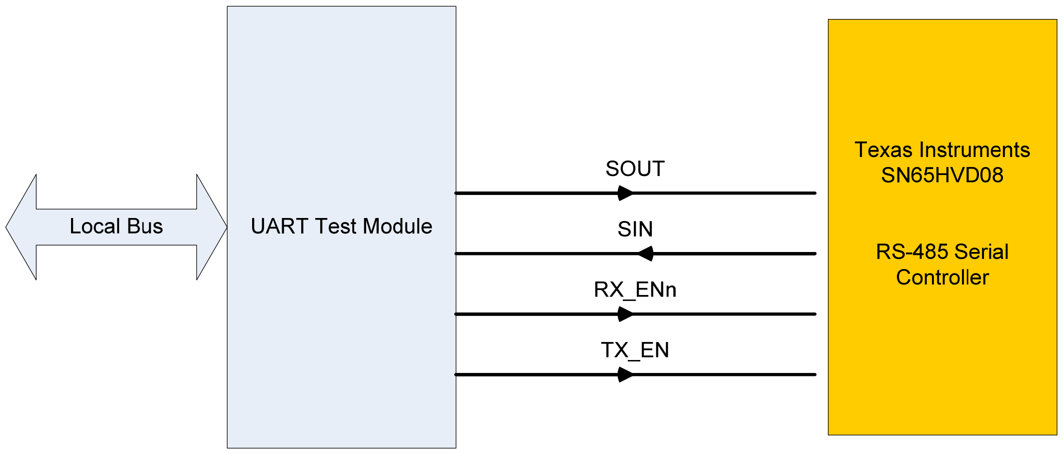


Figure 3-22 UART Test Module

3.16.3 I2C Interface

This interface implemented in order to communicate with Temperature sensors. Analog Devices ADT7461 product numbered temperature sensor is communicating with FPGA on I2C bus. Therefore a test module is implemented in order to bridge Local bus and I2C protocol was implemented according to the I2C standard [39]. defined by Philips semiconductors. The VME Master can read the temperature on card over VME Slave Module. The I2C Test Module is shown in Figure 3-23.

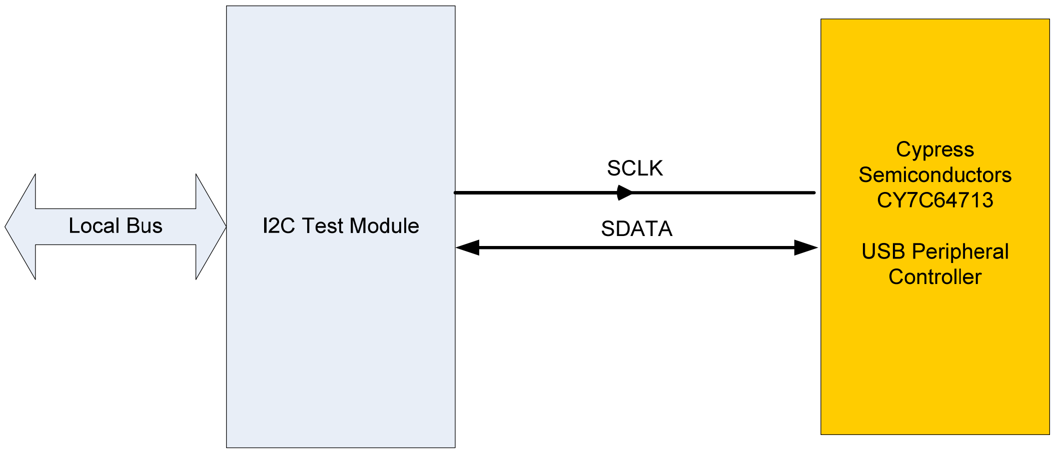


Figure 3-23 I2C Test Module

3.16.4 USB Interface

This interface implemented in order to communicate with USB Peripheral controller component. Cypress Semiconductors CY7C64713 product numbered USB peripheral controller is communicating with FPGA on a cypress semiconductors defined protocol “Slave FIFO” interface. The interface is a synchronous interface and the details for transaction types and signals are in [41]. The interface connection of FPGA and USB peripheral controller is shown in Figure 3-24.

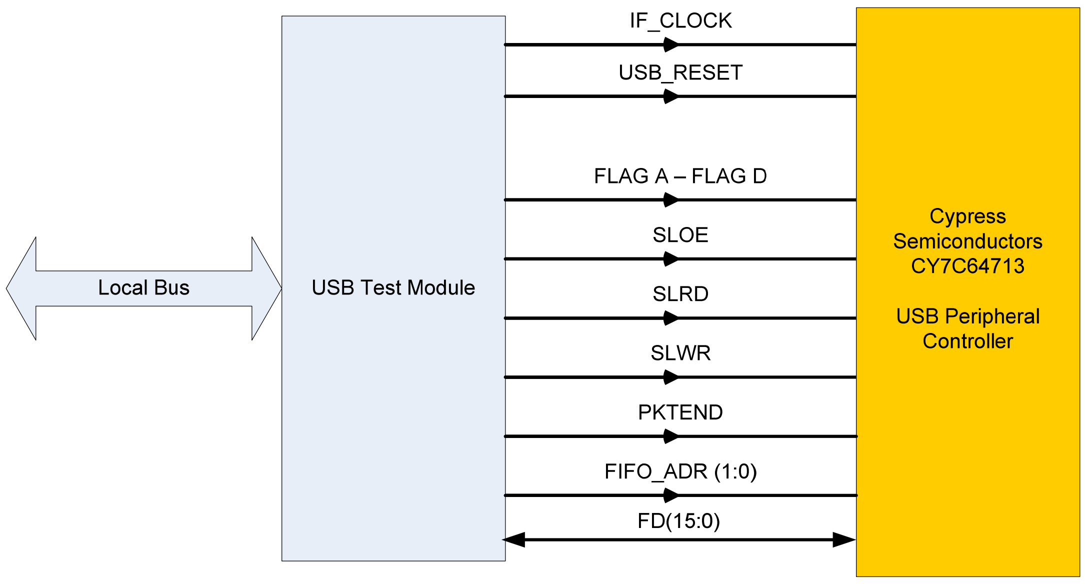


Figure 3-24 FPGA and USB Peripheral Controller "Slave FIFO" protocol interface

The explained modules are implemented and they are used to test hardware of VME Slave Card in hardware tests.

CHAPTER 4

VME MODULE HARDWARE IMPLEMENTATION

After the VME Slave Module is designed, it should be tested in an appropriate environment. Therefore, there was a need to develop simple VME Slave card where the VME Slave Module could be tested.

4.1 BLOCK DIAGRAM OF VME MODULE

The basic needs of VME Slave Module were defined and A24/D16 type VME Slave Card was designed. The functional block diagram of VME slave card is shown in Figure 4-1.

The card is designed with minimum features to occupy less space on VME card. Thus, the user decided components than could be easily placed on VME card. The heart of the card is the FPGA component; it is responsible of communication between VME Bus and other peripherals connected to it. For FPGA configuration at start there is a 10 pin male header and a SEEPROM is available on card. Moreover, for extension cards there are 3 PMC connectors and both of them has 64 pins. These connectors are routed according to ANSI/VITA 20-2001 standard [37]. Thus, the VME Slave card could carry PMC cards if needed. Except those, there is a RS-485 module for UART (Universal Asynchronous Receiver Transmitter) and a temperature sensor for I2C communication via FPGA. Furthermore, a scan bridge is placed on card to route the JTAG's (Joint Test Action Group) boundary scan test interface signals on VME Bus connector to both FPGA and to PMC connectors. Therefore, two JTAG chains are allocated on bus for boundary scan tests.

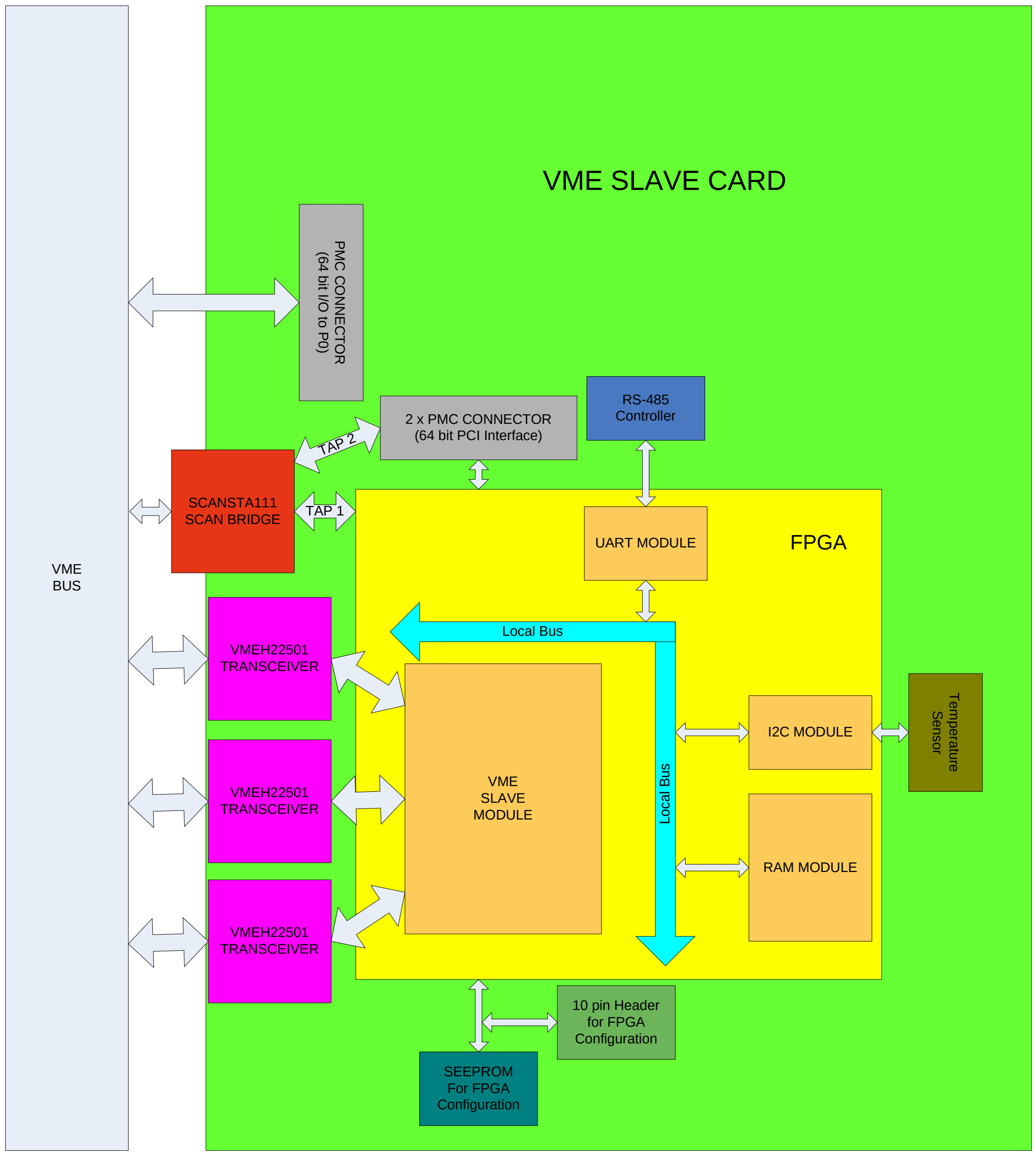


Figure 4-1 VME Slave Card Functional Block Diagram

4.2 COMPONENTS USED ON VME MODULE

In this section, the components placed on VME Slave Card will be explained.

4.2.1 FPGA

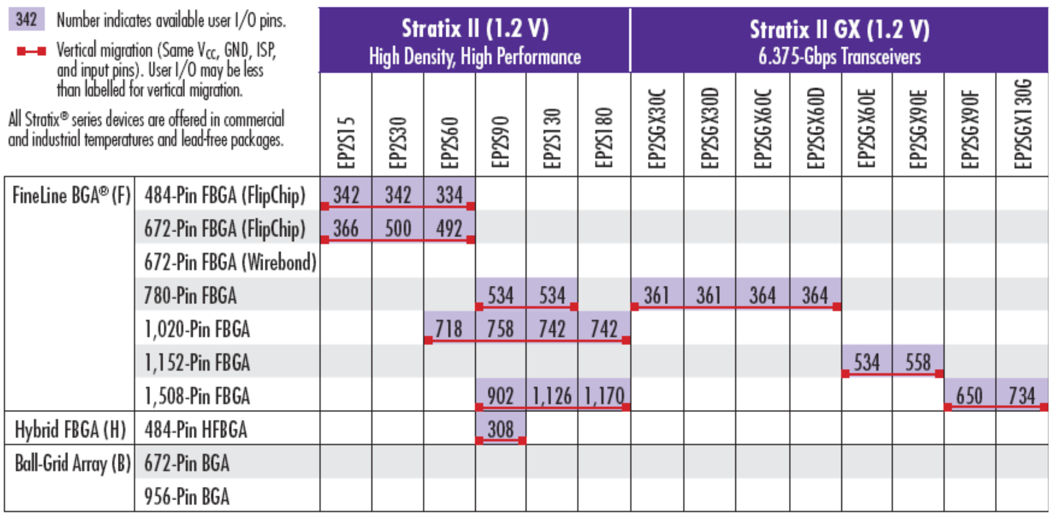
For VME Slave Card, ALTERA's EP2S15F484I4 was chosen. The ALTERA firm is chosen because of it is advanced technology in Stratix II family and also the IDE tools they provide for free when it is compared to XILINX or ACTEL EP2S15 is the cheapest FPGA available in Stratix II family. Moreover, the user could migrate to any other Stratix II family FPGA by only replacing the FPGA component on the VME slave module. As it is shown in Figure 4-2, the EP2S15 series Stratix II FPGA with 484 pins could be easily replaced by EP2S30 or EP2S60 versions of same family FPGAs because of the pin compatibility. Moreover, the device features of Stratix II family are shown in Figure 4-3. As it can be seen in Figure 4-3, the EP2S15 device could be replaced by EP2S60 device where the internal logic could be as four times as in EP2S15 device and this is achieved without making any modifications on card.

4.2.2 SEEPROM for FPGA Configuration

It is obvious that a FPGA needs a PROM (Programmable Read Only Memory) for its initial configuration and initialization. There are several methods to configure Stratix II FPGA's. Although these methods will not be discussed here in detail, these configuration methods can be found in [33]. For VME slave card, "Active Serial configuration" method is chosen. Thus, an EPC16 serial configuration device is chosen. It has 5 million bits to configure the EP2S15 device. Moreover, it could also program EP2S30 and EP2S60 devices. Therefore, there won't be any need to replace serial configuration device if a FPGA migration is needed.

4.2.3 VME Bus Transceiver

There are two kinds of VME bus transceivers available in market. The Texas Instruments VME backplane transceivers are investigated. There are two of them eliminated and they are SN54LVTH162245 and SN74VMEH22501 transceivers. SN74VMEH22501 transceivers are newer products and with recommended compatibility with new features of 2eVME or 2eSST options of VME Backplane. Moreover it is stated that they have balanced timing for high-to-low and low-to-high transitions even in heavily loaded backplanes [34]. This is really important for strobe signals in VME architecture since they are validating other signals by falling edges like DS0n or DTACKn [34]. Therefore, SN74VMEH22501 was chosen for VME Slave card. Moreover, the transceivers wouldn't be changed if 2eVME or 2eSST protocols implemented in VME Slave Module for further development.



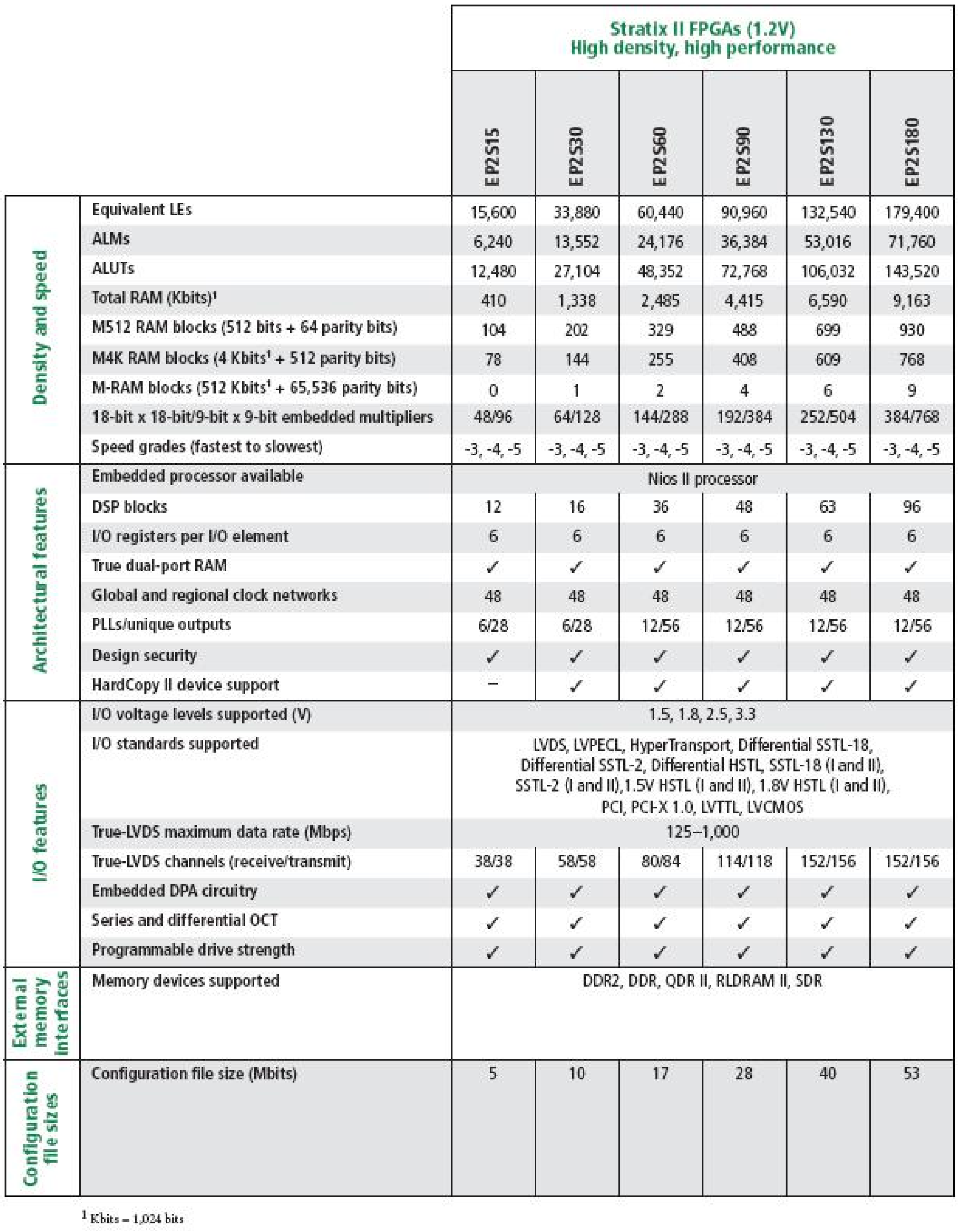
342 Number indicates available user I/O pins.

Vertical migration (Same V_{CC}, GND, ISP, and input pins). User I/O may be less than labelled for vertical migration.

All Stratix® series devices are offered in commercial and industrial temperatures and lead-free packages.

		Stratix II (1.2 V) High Density, High Performance						Stratix II GX (1.2 V) 6.375-Gbps Transceivers							
		EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180	EP2SGX30C	EP2SGX30D	EP2SGX60C	EP2SGX60D	EP2SGX60E	EP2SGX90E	EP2SGX90F	EP2SGX130G
FineLine BGA® (F)	484-Pin FBGA (FlipChip)	342	342	334											
	672-Pin FBGA (FlipChip)	366	500	492											
	672-Pin FBGA (Wirebond)														
	780-Pin FBGA				534	534		361	361	364	364				
	1,020-Pin FBGA			718	758	742	742								
	1,152-Pin FBGA											534	558		
	1,508-Pin FBGA				902	1,126	1,170							650	734
Hybrid FBGA (H)	484-Pin HFBGA				308										
Ball-Grid Array (B)	672-Pin BGA														
	956-Pin BGA														

Figure 4-2 Stratix II Family Device Pin-Out Compatibility [31]



		Stratix II FPGAs (1.2V) High density, high performance					
		EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180
Density and speed	Equivalent LEs	15,600	33,880	60,440	90,960	132,540	179,400
	ALMs	6,240	13,552	24,176	36,384	53,016	71,760
	ALUTs	12,480	27,104	48,352	72,768	106,032	143,520
	Total RAM (Kbits)[1]	410	1,338	2,485	4,415	6,590	9,163
	M512 RAM blocks (512 bits + 64 parity bits)	104	202	329	488	699	930
	M4K RAM blocks (4 Kbits[1] + 512 parity bits)	78	144	255	408	609	768
	M-RAM blocks (512 Kbits[1] + 65,536 parity bits)	0	1	2	4	6	9
	18-bit x 18-bit/9-bit x 9-bit embedded multipliers	48/96	64/128	144/288	192/384	252/504	384/768
	Speed grades (fastest to slowest)	-3, -4, -5	-3, -4, -5	-3, -4, -5	-3, -4, -5	-3, -4, -5	-3, -4, -5
Architectural features	Embedded processor available	Nios II processor					
	DSP blocks	12	16	36	48	63	96
	I/O registers per I/O element	6	6	6	6	6	6
	True dual-port RAM	✓	✓	✓	✓	✓	✓
	Global and regional clock networks	48	48	48	48	48	48
	PLLs/unique outputs	6/28	6/28	12/56	12/56	12/56	12/56
	Design security	✓	✓	✓	✓	✓	✓
	HardCopy II device support	–	✓	✓	✓	✓	✓
I/O features	I/O voltage levels supported (V)	1.5, 1.8, 2.5, 3.3					
	I/O standards supported	LVDS, LVPECL, HyperTransport, Differential SSTL-18, Differential SSTL-2, Differential HSTL, SSTL-18 (I and II), SSTL-2 (I and II),1.5V HSTL (I and II), 1.8V HSTL (I and II), PCI, PCI-X 1.0, LVTTL, LVCMOS					
	True-LVDS maximum data rate (Mbps)	125–1,000					
	True-LVDS channels (receive/transmit)	38/38	58/58	80/84	114/118	152/156	152/156
	Embedded DPA circuitry	✓	✓	✓	✓	✓	✓
	Series and differential OCT	✓	✓	✓	✓	✓	✓
	Programmable drive strength	✓	✓	✓	✓	✓	✓
External memory interfaces	Memory devices supported	DDR2, DDR, QDR II, RLDRAM II, SDR					
Configuration file sizes	Configuration file size (Mbits)	5	10	17	28	40	53

[1] Kbits = 1,024 bits

Figure 4-3 Altera Stratix II FPGA Family Features [32]

4.2.4 Serial Controller (RS-485)

For serial communication RS-485 was chosen and SN65HVD08 RS-485 transceiver from Texas Instruments was selected. There will be UART communication between FPGA and SN65HVD08, then SN65HVD08 will convert the signal level to RS-485 standard level and the data stream from RS-485 line will be converted to FPGA signal levels by SN65HVD08. Table 4-1, the RS-485 specifications are listed.

Table 4-1 RS-485 Specifications

Specification	RS-485
Mode of Operation	Differential
Total number of nodes (One Driver at a time)	32 Driver 32 Receiver
Max Cable Length	4000 FT (~ 1220 m)
Max Data Rate	100Kbps – 10Mbps
Max Driver Output Voltage	-7 V to + 12 V
Driver output Signal Level (Loaded. Min)	+/- 1.5 V
Driver output Signal Level (Unloaded. Max)	+/- 6 V
Driver Load Impedance	54 Ohms
Receiver Input Voltage Range	-7 V to + 12 V
Receiver Input Sensitivity	+/- 200 mV

4.2.5 Temperature Sensor

There would be a need for a temperature sensor on card. Because the master side would need the monitor the temperature on VME Slave card and if needed it would do necessary arrangements to cool down the VME Slave Card. The ADT7461 model digital temperature sensor from analog Devices placed on VME Slave Card. The ADT7461 not only monitors the temperature on its own diodes, but it also has ability to measure the temperature on external diodes. Thus, the ADT7461 could monitor the internal temperature of EP2S15 FPGA on card by FPGA’s internal temperature diodes. The communication between FPGA and the ADT7461 is on I2C bus.

4.2.6 Scan Bridge

The National Semiconductors SCANSTA111 scan bridge is used for boundary scan test of card. As there are boundary scan signals defined in VME64X [4] standard, there is a need to build a boundary scan chain inside the VME Slave Card. Both the FPGA and PMC have boundary scan connections. SCANSTA111 is multiplexing the boundary scan chains. It has one global connection routed to global chain and it connects three sub chains to this global chain. In VME Slave card, one sub chain is connected to FPGA, the other is connected to PMC connecter as PCI standard. The boundary scan chain of VME slave Card is shown in Figure 4-4.

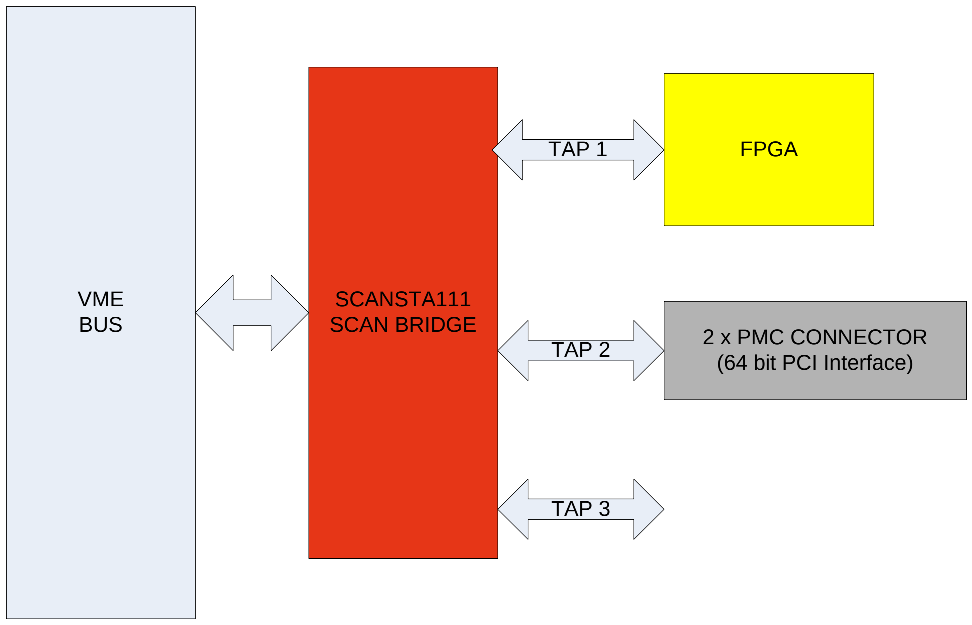


Figure 4-4 VME Slave Card Boundary Scan Chain

4.3 IMPLEMENTED VME SLAVE CARD

The produced VME Slave card is shown in Figure 4-5 and

Figure 4-6. Except the components mentioned in previous section, there are LED's to indicate if the 5 V, 3.3 V and 1.2V voltages are in suitable levels and there are 3 more LED's for user to indicate critical conditions. Moreover, there are 4 switches also attached to FPGA for users.

4.4 DESIGNED ADD-ON CARD (USB Interface)

To test and validate the expansion card capabilities of VME Slave Card, an add-on card is designed. The new add on card is connected to the VME Slave Card with 2 PMC connectors and it consists a USB peripheral controller. The main purpose of the add-on card is to interface the VME Slave Card FPGA with USB. The

Cypress's CY7C64713 part numbered USB Peripheral controller is mounted on add-on card. The Interface between FPGA and the USB Peripheral Controller is accomplished through PMC connectors on VME SLAVE Card. The VME Slave Card and designed Add-On card connection diagram is seen in Figure 4-7. Figure 4-8 shows the component side of the designed Add-On USB Interface Card. Therefore, a VME Bus to USB bridging is also activated by connecting add-on card to the appropriate slots on VME Slave Card supply pins of PMC standard [37].

The add-on card is supplied from 3.3 Volt pins of PMC standard. Since the Cy7C64713 component needs supply voltage of 3.3 volts, the add-on card is correctly functions with PMC 3.3V

The details of Cypress Semiconductor's CY7C64713 peripheral USB controller can be found in [41].

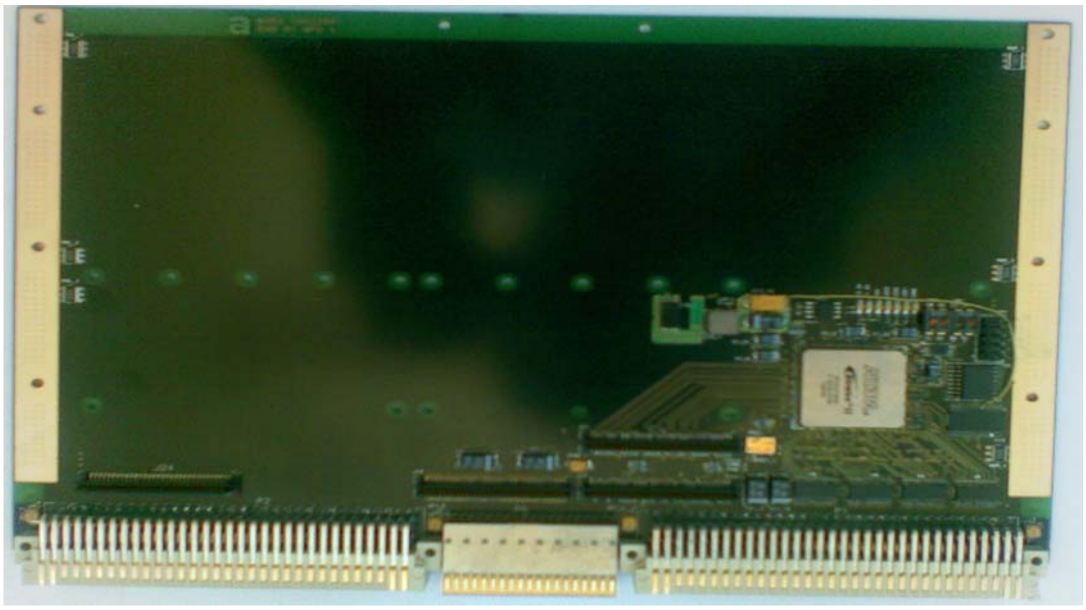


Figure 4-5 VME Slave Card Component Side View

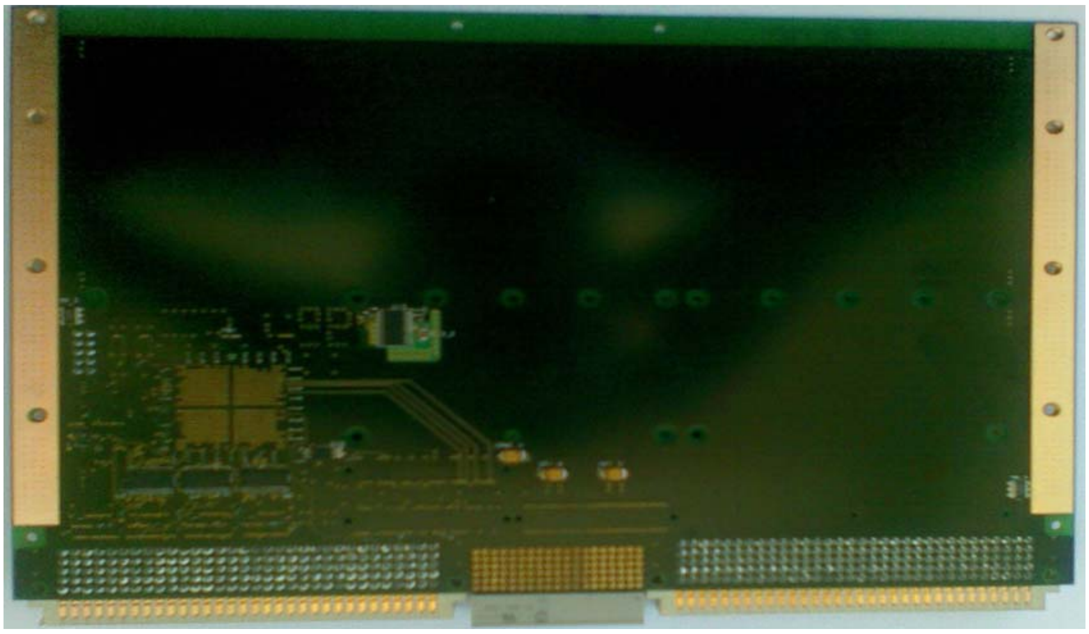


Figure 4-6 VME Slave Card Solder Side View

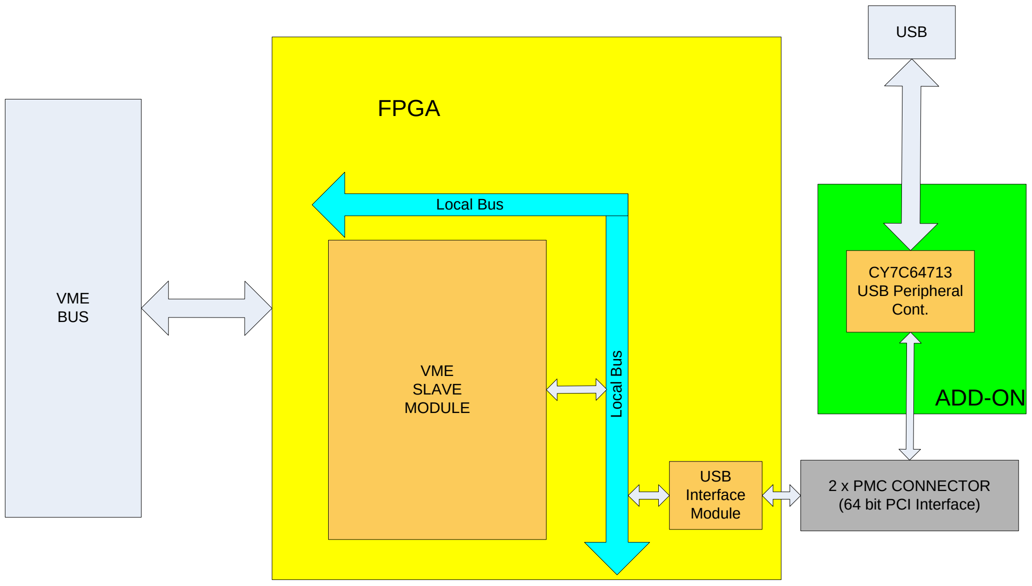


Figure 4-7 VME Slave Card and Add-On Card Connection Diagram

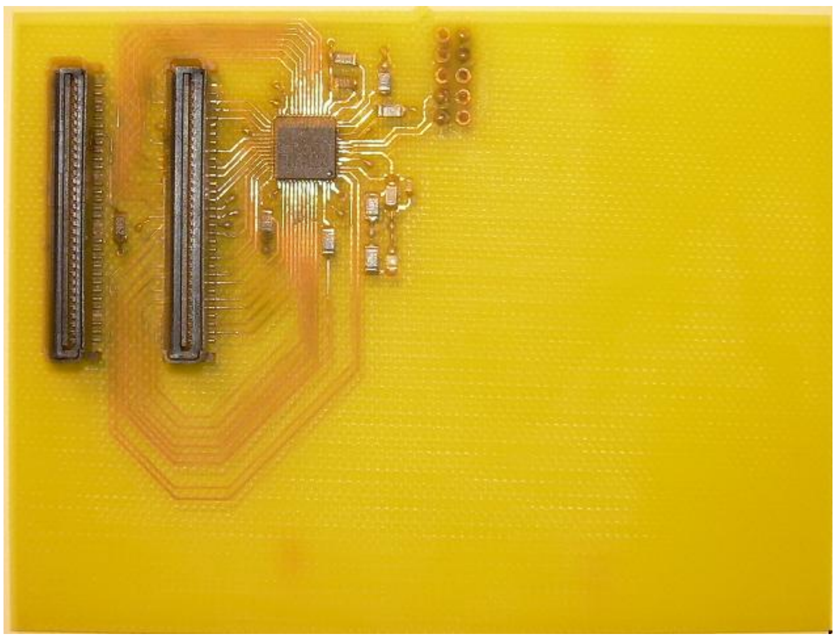


Figure 4-8 Add-On USB Interface Card Component Side View

CHAPTER 5

SIMULATIONS

5.1 SOFTWARE TESTS

In order to test the VME Slave Module, a test bench was created. In this test bench, there is a master block communicates with VME Slave Module via VMEH22501 buffers. Those buffers were modeled in VHDL environment and added to test conditions. Moreover, 1 Kbyte RAM block were also implemented with it’s controller to communicate with VME Slave Module inside the FPGA. The designed test case is shown in Figure 5-1.

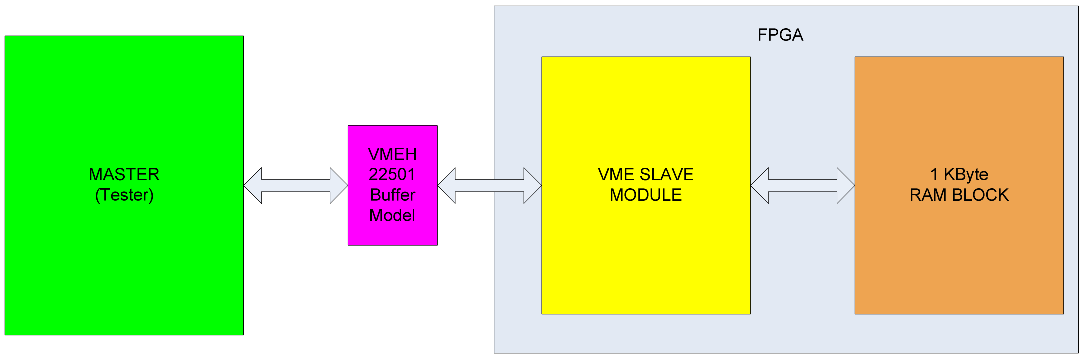


Figure 5-1 VME Slave Module Test Case

5.1.1 Functional Simulations

In functional simulations there are several steps the Master checks in VME Slave Module functionality

1. Master first writes to RAM module in A24/D8 block transfer mode, afterwards it reads the data inside the RAM in A24/D16 block transfer mode and checks if the written data is true.

2. Master first writes to RAM module in A32/D32 block transfer mode, afterwards it reads the data inside the RAM in A32/D16 block transfer mode and checks if the written data is true.

3. Master fills the RAM block in A24/D8 single cycle mode, afterwards it reads the data inside the RAM in A24/D16 single cycle mode and checks if the written data is true.

4. Master fills the RAM block in A16/D16 single cycle mode, afterwards it reads the data inside the RAM in A16/D8 single cycle mode and checks if the written data is true.

5. Master makes a 3 byte unaligned write transfer and reads both address boundary and checks if the data is written.

6. Master applies Read Modify Write Cycle on RAM for 3 addresses and checks if the results are true.

7. Master applies Address Only Cycle.

8. Master tries all 7 levels of interrupt cycles in ROAK mechanism.

9. The Generic in Interrupter module to set RORA or ROAK mechanism is changed to RORA mechanism.

10. Master tries all 7 levels of interrupt cycles in RORA mechanism.

Those 10 step test was applied to the VME Slave Module and it passes all the tests applied. The samples of functional test results are shown in Figure 5-2- Figure 5-4.

5.1.2 Code Synthesis

According to the programmable logic design process, the next step after the simulations is to analyze and synthesis the design in suitable design environment of the Vendor. The VME Slave Module is not vendor specific. Thus, the VME Slave Module is synthesized in 3 different vendors IDE (Integrated Design Environment) programs. Those vendors are ALTERA, ACTEL and XILINX. On the following sub-sections the logic, VME Slave module occupies in each vendor’s products can be found.

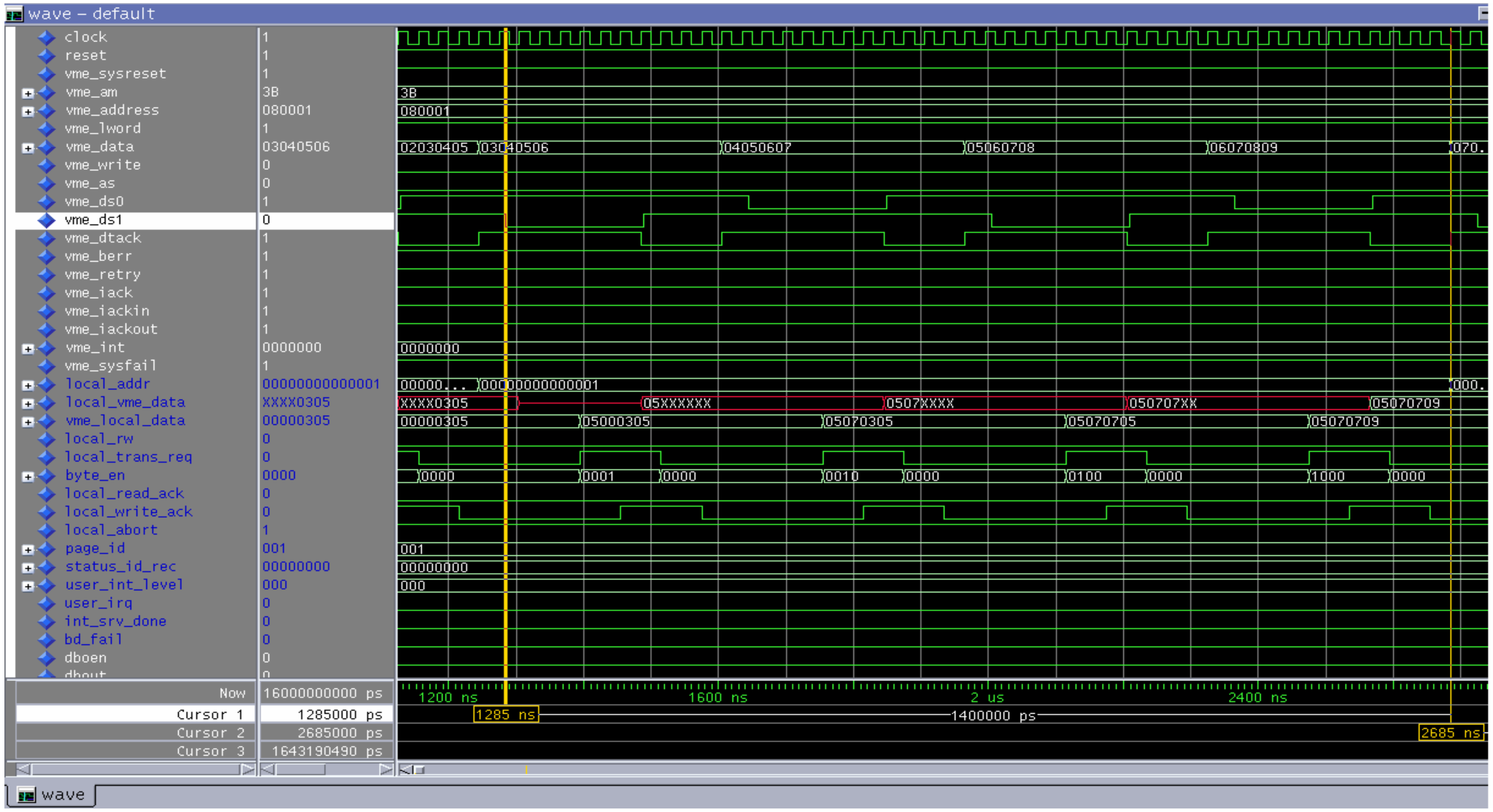
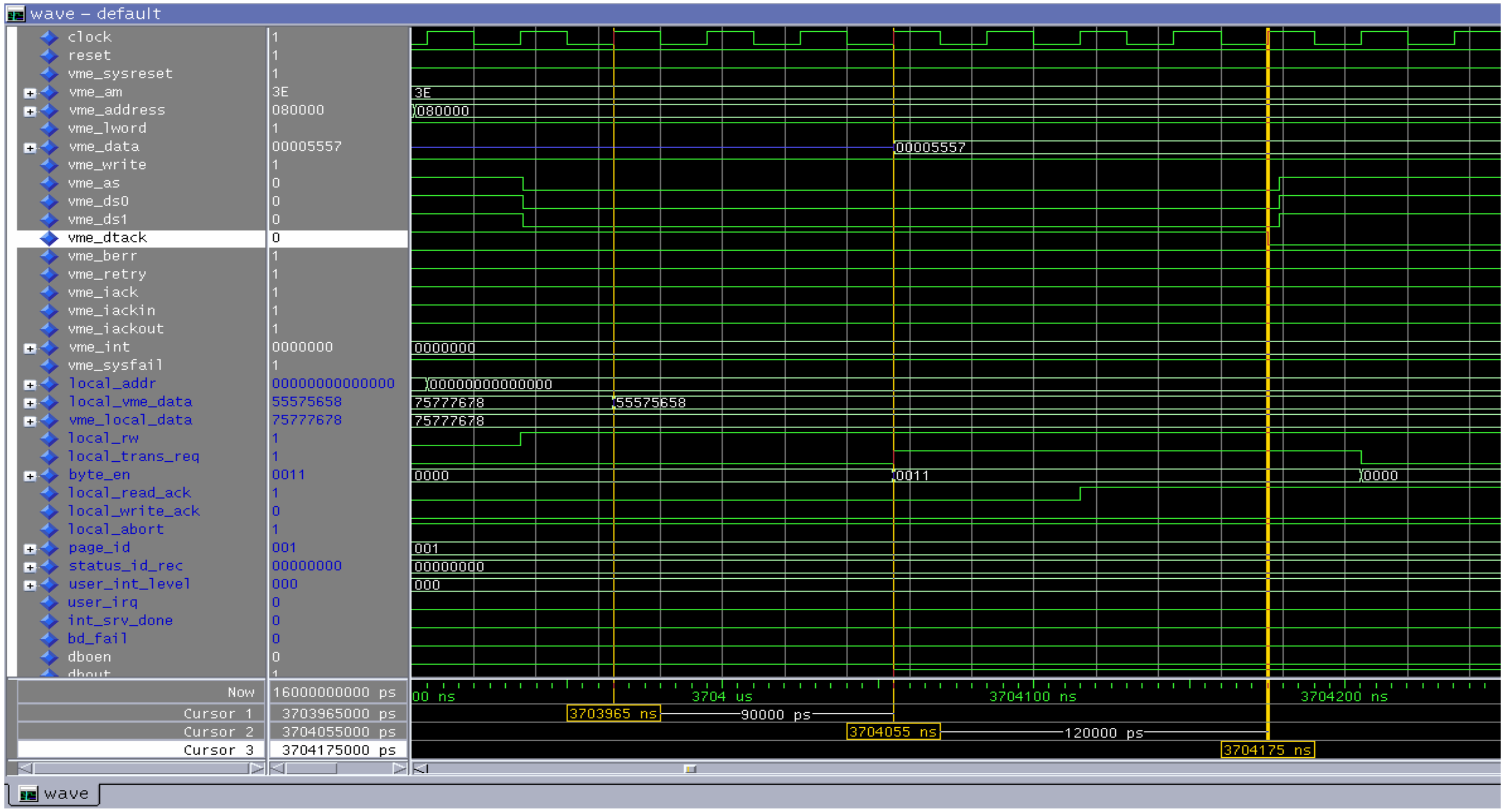
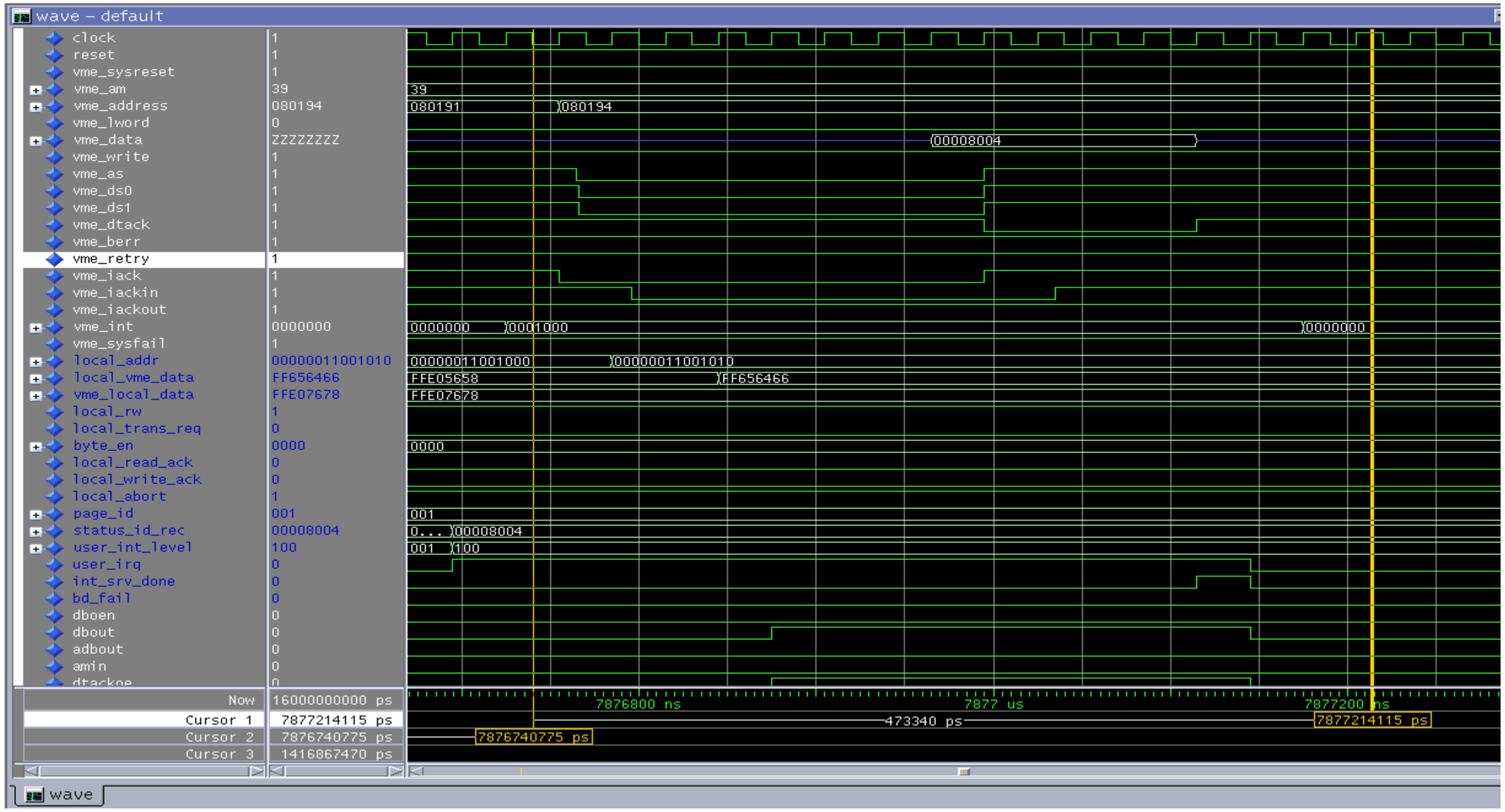


Figure 5-2 VME Block Transfer Cycle (Write - One Cycle)



```
wave - default
clock              1
reset              1
vme_sysreset       1
vme_am             3E
vme_address        080000
vme_lword          1
vme_data           00005557
vme_write          1
vme_as             0
vme_ds0            0
vme_ds1            0
vme_dtack          0
vme_berr           1
vme_retry          1
vme_iack           1
vme_iackin         1
vme_iackout        1
vme_int            0000000
vme_sysfail        1
local_addr         00000000000000
local_vme_data     55575658
vme_local_data     75777678
local_rw           1
local_trans_req    1
byte_en            0011
local_read_ack     1
local_write_ack    0
local_abort        1
page_id            001
status_id_rec      00000000
user_int_level     000
user_irq           0
int_srv_done       0
bd_fail            0
dboen              0
dbout              1

      Now   16000000000 ps
 Cursor 1    3703965000 ps
 Cursor 2    3704055000 ps
 Cursor 3    3704175000 ps
```

Figure 5-3 VME Single Cycle – Read (1 cycle)



```
clock               1
reset               1
vme_sysreset        1
vme_am              39
vme_address         080194
vme_lword           0
vme_data            ZZZZZZZZ
vme_write           1
vme_as              1
vme_ds0             1
vme_ds1             1
vme_dtack           1
vme_berr            1
vme_retry           1
vme_iack            1
vme_iackin          1
vme_iackout         1
vme_int             0000000
vme_sysfail         1
local_addr          00000011001010
local_vme_data      FF656466
vme_local_data      FFE07678
local_rw            1
local_trans_req     0
byte_en             0000
local_read_ack      0
local_write_ack     0
local_abort         1
page_id             001
status_id_rec       00008004
user_int_level      100
user_irq            0
int_srv_done        0
bd_fail             0
dboen               0
dbout               0
adbout              0
amin                0
dtackoe             0

Now        16000000000 ps
Cursor 1   7877214115 ps
Cursor 2   7876740775 ps
Cursor 3   1416867470 ps
```

```
vme_am          39
vme_address     080191  080194
vme_data        00008004
vme_int         0000000  0001000  0000000
local_addr      00000011001000  00000011001010
local_vme_data  FFE05658  FF656466
vme_local_data  FFE07678
byte_en         0000
page_id         001
status_id_rec   0...  00008004
user_int_level  001  100
```

Figure 5-4 VME Bus Interrupt Cycle (Level 4 – RORA mechanism)

5.1.2.1 QUARTUS II (ALTERA)

QUARTUS II is the IDE of ALTERA products. The VME Slave Module is compiled in two versions of QUARTUS II. They are VER 5.0 and VER 7.2.

- QUARTUS II Version 5.0

Table 5-1 VME Slave Module Analysis Result I on QUARTUS II VER 5.0

EP2S15F484I4	**LC Combinationals**	**LC Registers**	**Total ALUT's**
VME Slave Interface	178	141	225 / 12480 (%1.8)
DTB	150	109	
Interrupter	27	29	
Utility	1	3	

Table 5-2 VME Slave Module Analysis Result II on QUARTUS II VER 5.0

EP2S15F484I4	**Required Time**	**Unit**	**Actual Time**	**Unit**
Clock	33.33 MHz (period = 30.00ns)	MHz	128.93 (Period = 7.756 ns)	MHz

- QUARTUS II Version 7.2

Table 5-3 VME Slave Module Analysis Result I on QUARTUS II VER 7.2

EP2S15F484I4	**LC Combinationals**	**LC Registers**	**Total ALUT's**
VME Slave Interface	182	141	184 / 12480 (%1.5)
DTB	151	109	
Interrupter	30	29	
Utility	1	3	

Table 5-4 VME Slave Module Analysis Result II on QUARTUS II VER 7.2

EP2S15F484I4	**Required Time**	**Unit**	**Actual Time**	**Unit**
Clock	33.33 Mhz (Period = 30.00ns)	MHz	142.42 (Period = 7.021 ns)	MHz

VME slave Module also synthesized for EP2S30F484I4 and EP2S60F484I4 and in both devices, the results were same as EP2S15F484I4 so the Table 5-1 to Table 5-4 includes the compilation results of EP2S30 and EP2S60 devices.

VME Slave Module occupies less than %2 of the total logic count in EP2S15 device. Therefore, more than %98 of logic space is available for user development. Moreover, the VME slave Module could reach speeds up to 142.2 MHz. This is really a great performance criteria when it is compared with available products on VME market.

5.1.2.2 LIBERO (ACTEL)

LIBERO is the IDE of ACTEL products. The available version of LIBERO is VER8.0. All of the following results are obtained by LIBERO VER8.0. The IP cores in [19] - [25] includes the compilation results of ACTEL FPGA's. Thus the VME Slave Module is also compiled for ACTEL FPGA's and the compilation reports of the VME Slave Module and it's comparison with other IP's are shown in Table 5-5 - Table 5-10.

Table 5-5 Compilation Report of VME Slave Module Comparison on Axcelerator FPGA

Axcelerator AX500-STD	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	206	145	351 (%4)	132 MHz
IniCore A32D32 Slave Controller [21]	193	159	342 (%4)	119 MHz
MemecCore MC-ACT-VME32 [23]	216	196	412 (%5)	86 MHz

Table 5-6 Compilation Report of VME Slave Module Comparison on ProASIC$^{\underline{PLUS}}$ APA150 FPGA

ProASIC$^{\underline{PLUS}}$ APA150-STD	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	n/a	n/a	750 (%12)	71 MHz
IniCore A32D32 Slave Controller [21]	n/a	n/a	477 (%8)	67 MHz

Table 5-7 Compilation Report of VME Slave Module Comparison on ProASIC$^{\underline{PLUS}}$ APA-600 FPGA

ProASIC$^{\underline{PLUS}}$ APA600-STD	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	n/a	n/a	750 (%3)	71 MHz
MemecCore MC-ACT-VME32 [23]	n/a	n/a	672 (%3)	58 MHz

Table 5-8 Compilation Report of VME Slave Module Comparison on SXA A54SX72A FPGA

SX-A A54SX72A	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	206	151	357 (%6)	71 MHz
IniCore A32D32 Slave Controller [21]	193	163	356 (%6)	83 MHz
MemecCore MC-ACT-VME32 [23]	220	196	416 (%7)	62 MHz

Table 5-9 Compilation Report of VME Slave Module Comparison on RTSX RT54SX72A FPGA

RTSX RT54SX72A	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	**206**	**151**	**357 (%7)**	**71 MHz**
IniCore A32D32 Slave Controller [21]	**193**	**162**	**355 (%6)**	**43 MHz**

Table 5-10 Compilation Report of VME Slave Module Comparison on ProASIC3 A3PE-600 FPGA

ProASIC3 A3PE600-STD	**C-Mod**	**S-Mod**	**Total**	**Estimated Performance**
VME Slave Interface	**n/a**	**n/a**	**609 (%4)**	**77.2 MHz**
MemecCore MC-ACT-VME32 [23]	**n/a**	**n/a**	**536 (%4)**	**82 MHz**

In Table 5-5, VME Slave Module is compared with IniCore's and MemecCore's IP's when they are all compiled in ACTEL's Axcelerator AX500 series FPGA. Although VME slave Module has additional features, it occupies nearly as same space as other IP's and on estimated performance it reaches nearly 132 MHz and this is a significant difference.

In Table 5-6, VME Slave Module is compared with IniCore's IP when they are compiled in ACTEL's ProASIC$^{\underline{PLUS}}$ APA150 series FPGA. Although VME slave Module occupies 1.5 times the space IniCore's IP occupy, there is a performance increase to 71 MHz. Moreover, when the logic implementation on FPGA is thought, the additional features of VME Slave Module should be considered.

In Table 5-7, VME Slave Module is compared with MemecCore's IP's when they are compiled in ACTEL's ProASIC$^{\underline{PLUS}}$ APA150 series FPGA. Although VME slave Module has additional features, it occupies nearly as same space as other IP's and on estimated performance it reaches nearly 71 MHz and this is an performance increase compared to the IP of MemecCore.

In Table 5-8, VME Slave Module is compared with IniCore's and MemecCore's IP's when they are all compiled in ACTEL's SX-A A54SX72A series FPGA. Although VME slave Module has additional features, it occupies nearly as same space as other IP's and on estimated performance it reaches nearly 71 MHz which is the middle estimated frequency when compared to other IP's. This would be because of the architecture of SX-A family FPGA's.

In Table 5-9, VME Slave Module is compared with IniCore's IP when they are compiled in ACTEL's RTSX RTSX72S series FPGA. VME Slave Module occupies the same space as IniCore's IP occupies. However, there is a great performance increase in estimated frequency between 43 MHz and 71 MHz.

In Table 5-10, VME Slave Module is compared with MemecCore's IP's when they are compiled in ACTEL's ProASIC3 A3PE600 series FPGA VME Slave Module occupies a little bit more logic than the IP of MemecCore and this has an affect on performance estimation. Thus, while VME Slave Module reaches 77.2 MHz, MemecCore's IP could reach 82 MHz.

To conclude, VME Slave Module is designed with many additional features to other IP's. Thus, there was an expectation that the VME Slave Module would occupy much more space than the available IP's in market. However, the VME Slave Module occupies nearly as same space as the available IP's on market and its performance estimation is much better than the other ones. Therefore, there should be a comment that the new design approach to VME Slave application succeeded among others.

5.1.2.3 ISE (XILINX)

ISE (Integrated Software Environment) is the IDE of XILINX products. The VME Slave Module is compiled on ISE VER9.0 for SPARTAN3AN Family FPGA's. The compilation report results are shown in Table 5-11.

Table 5-11 Compilation Report of VME Slave Module on SPARTAN3AN FPGA

XC3S1400AN	**SLICES**	**TOTAL**	**ESTIMATED FREQUENCY**
VME Slave Interface	155	155 / 22528 (< %1)	177.4 MHz

In SPARTAN3AN FPGA, the VME slave module just occupies less than %1 of the device logic. Thus, the rest %99 of device would be available for user implementations. Moreover, the design could run as high as 177 MHz in this device. This is really high frequency estimation and proves the capability of parallel processing of the design approach of VME Slave Module.

5.1.3 Timing Simulations

According to the programmable logic design process, after the design compiled in suitable design environment of selected vendor, there should be carried out timing simulations on outputs of IDE software. In this implementation, the ALTERA's EP2S15F484I4 FPGA is target device in the produced VME Slave Card. Thus, all the timing simulations are carried out on the output files of QUARTUS II VER 5.0. Again the Master code is not changed in test case and the following steps were followed to check if the designed VME Slave Module is working appropriately.

1. Master first writes to RAM module in A24/D8 block transfer mode, afterwards it reads the data inside the RAM in A24/D16 block transfer mode and checks if the written data is true.

2. Master first writes to RAM module in A32/D32 block transfer mode, afterwards it reads the data inside the RAM in A32/D16 block transfer mode and checks if the written data is true.

3. Master fills the RAM block in A24/D8 single cycle mode, afterwards it reads the data inside the RAM in A24/D16 single cycle mode and checks if the written data is true.

4. Master fills the RAM block in A16/D16 single cycle mode, afterwards it reads the data inside the RAM in A16/D8 single cycle mode and checks if the written data is true.

5. Master makes a 3 byte unaligned write transfer and reads both address boundary and checks if the data is written.

6. Master applies Read Modify Write Cycle on RAM for 3 addresses and checks if the results are true.

7. Master applies Address Only Cycle.

8. Master tries all 7 levels of interrupt cycles in ROAK mechanism.

9. The Generic in Interrupter module to set RORA or ROAK mechanism is changed to RORA mechanism.

10. Master tries all 7 levels of interrupt cycles in RORA mechanism.

The VME Slave Module passes all the timing simulation cases. Therefore the VME Slave Module code is ready to be downloaded on target board and hardware tests could be started.

5.1.4 Test Coverage

The test benches designed in order to test VME Slave Module are evaluated as how many lines of the designed VME Slave Module VHDL code are tested. Mentor Graphics Modelsim 6.1g Code Coverage Report tool is reporting the how many lines of the designed VHDL code is tested with designed test bench. The test coverage report of VME Slave Module testbench is show in Table 5-12. On overall of %89 of VME Slave Module VHDL code is tested with the created testbench. In fact, the % 11 untested regions are the “else” branches in VHDL code which can be activated with different GENERIC definitions.

Table 5-12 Test Coverage Report of VME slave Module Testbench

Module	**Test Coverage (%)**
DTB Module	88.3
Interrupter Module	90.2
Utility Module	94.4
Overall	89

5.2 HARDWARE TESTS

After the VME Slave Module design is verified on every step of programmable logic design processes, the final step is to download the design into the target board and make necessary tests on designed VME Slave Card as hardware tests.

5.2.1 VME Module Test on VME Chassis

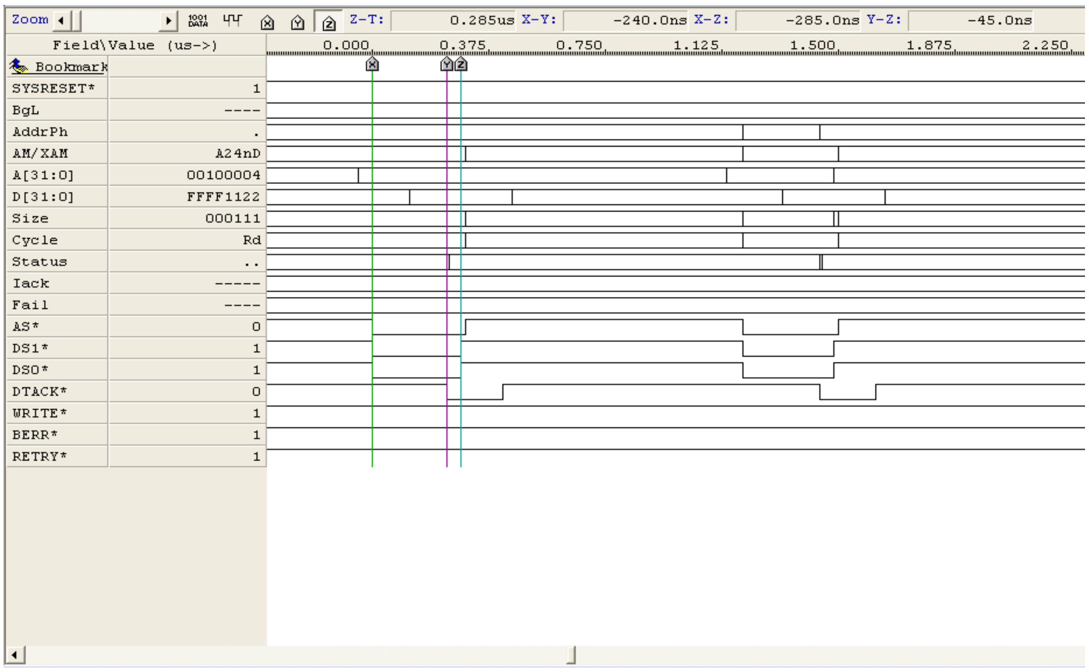
In order to test the designed VME Slave Card with the downloaded VME Slave Module on it, a VME chassis of AP-LABS FS-1112 is used. Moreover, there should be a system control needed to start up VME system. VMETRO's Vanguard VME Analyzer is used as system controller and master. It has abilities to monitor protocol violations, can initiate write or read transfers with desired length and protocol on VME Bus. The Vanguard VME analyzer is controlled by a software program which is called BusView. In the tests, the BusView ver5.2 is used. The Busview program allows a designer to initiate transfers on VME Bus, moreover the VME Bus can be monitored and the waveforms of signals on VME bus can be shown anytime on screen with the help of this program.

VME Slave Card is designed as A24/D16 mode. Thus, all the tests on VME Slave Card are done on A24 mode. Several tests are carried out on VME Slave with Vanguard VME Analyzer şn laboratory environment on VME chassis. The VME Slave Card's FPGA has VME Slave Module and RAM Module downloaded on it.

The results were successful. There are some samples captured from the BusView program. The screen captures of Busview program are shown in Figure 5-5 and Figure 5-6. In Figure 5-5 and Figure 5-6 the samples taken from backplane can be seen. The transactions are single cycle read and write. Also the exerciser menu samples can be seen in Figure 5-7. To explain the situation in Figure 5-7,

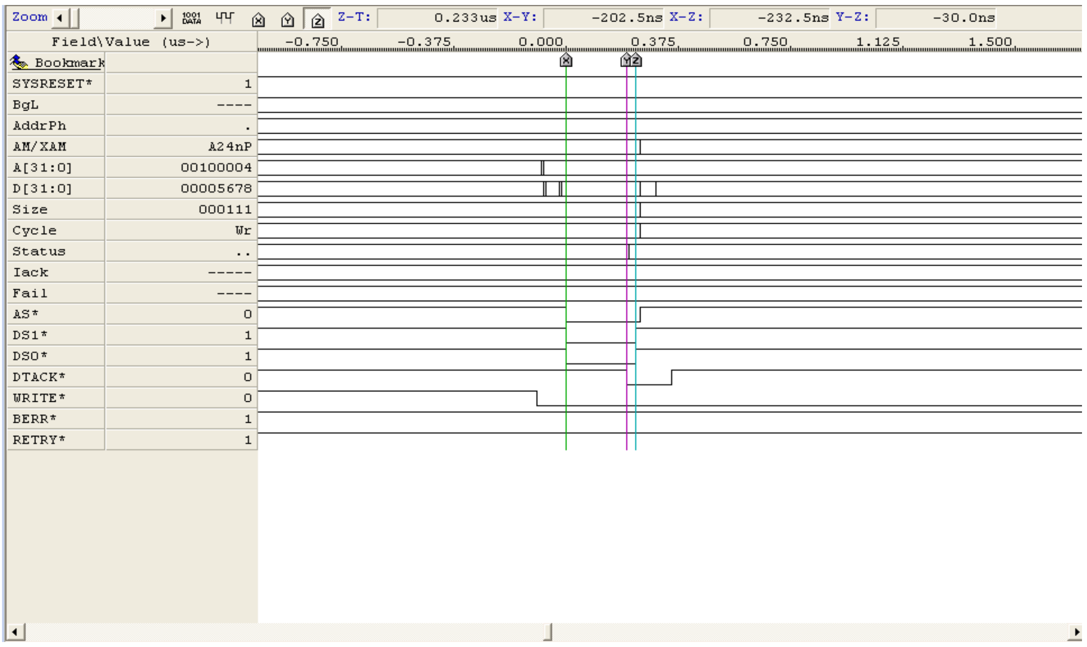
1. VME address from 0x100000 to 0x100010 is read.
2. 2 byte data 0x5678 is written to VME address 0x100000
3. VME address 0x100000 is read and the data 0x5678 is there.
4. VME address range from 0x100000 to 0x1003FC is tested with random values of data for 5 times and the VME slave Card passed that test.
5. VME address range from 0x100000 to 0x1003FC is tested with walking zeros data for 5 times and the VME slave Card passed that test.
6. VME address range from 0x100000 to 0x1003FC is tested with walking ones data for 5 times and the VME slave Card passed that test.

Moreover, during all tests, the protocol checker capability of the Vanguard VME Analyzer was on. There weren't any VME protocol violations on any of the VME transactions. The result of protocol checker is shown in Figure 5-8. Thus, the VME Slave Card and VME Slave Module implementation on FPGA prove the compatibility to VME Standards [3] [4] [2].



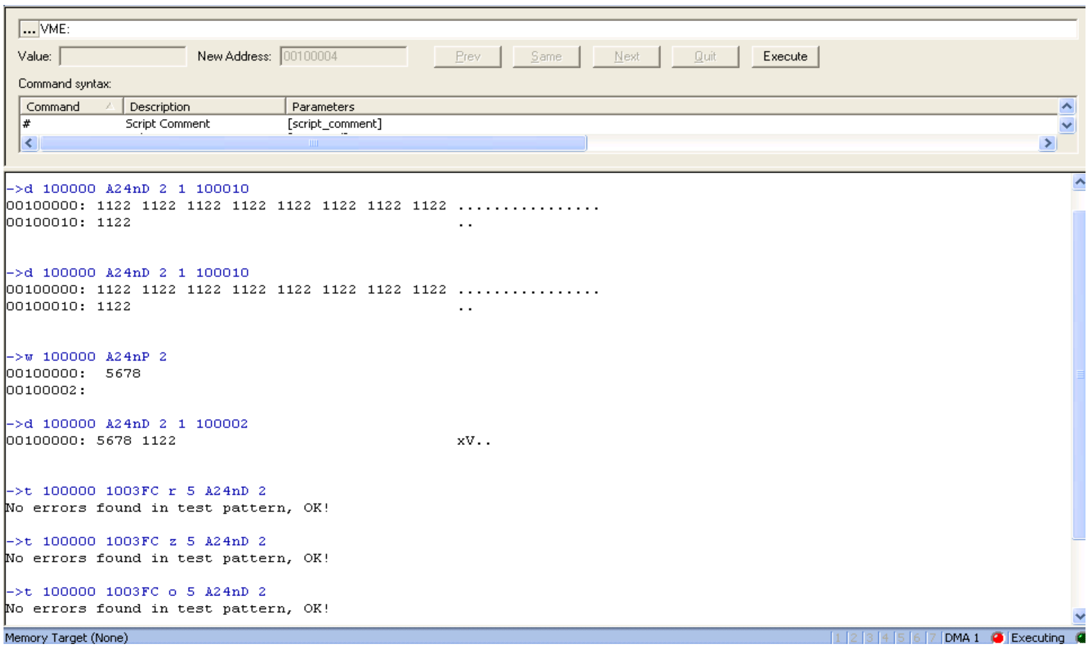
```
SYSRESET*          1
BgL             ----
AddrPh             .
AM/XAM         A24nD
A[31:0]     00100004
D[31:0]     FFFF1122
Size          000111
Cycle             Rd
Status            ..
Iack           -----
Fail            ----
AS*                0
DS1*               1
DS0*               1
DTACK*             0
WRITE*             1
BERR*              1
RETRY*             1
```

Figure 5-5 Single Cycle Transaction (Read)



```
SYSRESET*          1
BgL             ----
AddrPh             .
AM/XAM         A24nP
A[31:0]     00100004
D[31:0]     00005678
Size          000111
Cycle             Wr
Status            ..
Iack           -----
Fail            ----
AS*                0
DS1*               1
DS0*               1
DTACK*             0
WRITE*             0
BERR*              1
RETRY*             1
```

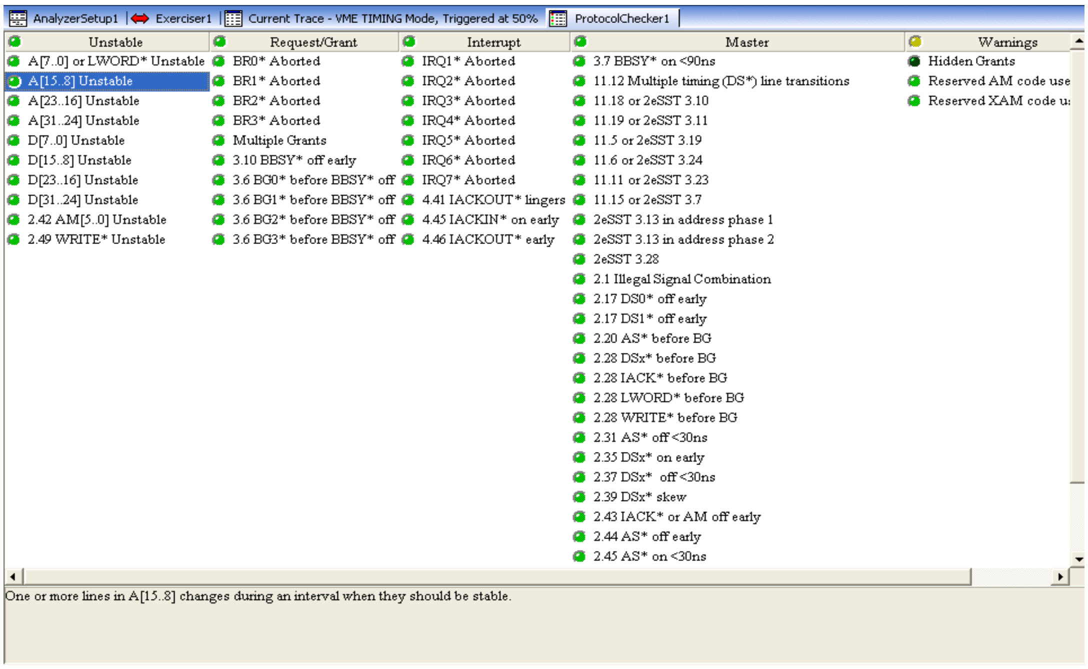
Figure 5-6 Single Cycle Transaction (Write)



Command	Description	Parameters
#	Script Comment	[script_comment]

```
->d 100000 A24nD 2 1 100010
00100000: 1122 1122 1122 1122 1122 1122 1122 1122 ................
00100010: 1122                                    ..


->d 100000 A24nD 2 1 100010
00100000: 1122 1122 1122 1122 1122 1122 1122 1122 ................
00100010: 1122                                    ..


->w 100000 A24nP 2
00100000:  5678
00100002:

->d 100000 A24nD 2 1 100002
00100000: 5678 1122                               xV..


->t 100000 1003FC r 5 A24nD 2
No errors found in test pattern, OK!

->t 100000 1003FC z 5 A24nD 2
No errors found in test pattern, OK!

->t 100000 1003FC o 5 A24nD 2
No errors found in test pattern, OK!
```

Figure 5-7 Exerciser Tests on VME Slave Card



Unstable	Request/Grant	Interrupt	Master	Warnings
A[7..0] or LWORD* Unstable	BR0* Aborted	IRQ1* Aborted	3.7 BBSY* on <90ns	Hidden Grants
A[15..8] Unstable	BR1* Aborted	IRQ2* Aborted	11.12 Multiple timing (DS*) line transitions	Reserved AM code use
A[23..16] Unstable	BR2* Aborted	IRQ3* Aborted	11.18 or 2eSST 3.10	Reserved XAM code u:
A[31..24] Unstable	BR3* Aborted	IRQ4* Aborted	11.19 or 2eSST 3.11	
D[7..0] Unstable	Multiple Grants	IRQ5* Aborted	11.5 or 2eSST 3.19	
D[15..8] Unstable	3.10 BBSY* off early	IRQ6* Aborted	11.6 or 2eSST 3.24	
D[23..16] Unstable	3.6 BG0* before BBSY* off	IRQ7* Aborted	11.11 or 2eSST 3.23	
D[31..24] Unstable	3.6 BG1* before BBSY* off	4.41 IACKOUT* lingers	11.15 or 2eSST 3.7	
2.42 AM[5..0] Unstable	3.6 BG2* before BBSY* off	4.45 IACKIN* on early	2eSST 3.13 in address phase 1	
2.49 WRITE* Unstable	3.6 BG3* before BBSY* off	4.46 IACKOUT* early	2eSST 3.13 in address phase 2	
			2eSST 3.28	
			2.1 Illegal Signal Combination	
			2.17 DS0* off early	
			2.17 DS1* off early	
			2.20 AS* before BG	
			2.28 DSx* before BG	
			2.28 IACK* before BG	
			2.28 LWORD* before BG	
			2.28 WRITE* before BG	
			2.31 AS* off <30ns	
			2.35 DSx* on early	
			2.37 DSx* off <30ns	
			2.39 DSx* skew	
			2.43 IACK* or AM off early	
			2.44 AS* off early	
			2.45 AS* on <30ns	

One or more lines in A[15..8] changes during an interval when they should be stable.

Figure 5-8 Protocol Checker Results

CHAPTER 6

SUMMARY AND CONCLUSION

In this thesis work, an easy interfaced generic VME Slave Module IP is designed. The developed module is implemented in VHDL and optimized for ALTERA Stratix I family FPGA's. However, the developed VHDL code can also be easily synthesized for other vendor's FPGA's.The VME bus is extensively used in industrial, military, aerospace, communication and nuclear applications, there are only several IP solutions and limited research works are carried out on this subject. On the other hand, laboratories spend lots of time in order to develop VME Modules at low costs to satisfy their specific experimental requirements. Therefore in this thesis work, a VME Slave Module is implemented and a VME Slave Card is designed. In fact, the designed VME Slave Module is more capable than the ones proposed and available in market.

The designed VME Slave Module has three paged local side and every page has its own base address, address offset and address width selection registers for VME Bus. Therefore, three different slave cards in fact are gathered in one VME Slave Module.

Address decoding is also handled inside VME Slave Module and the local user needs neither any decoding logic nor address modifier code decoding implementation for local bus communication

The data types and sizes that the VME Slave Module would respond can be selected. This is also an advantage to prevent the misusage of card and prevent illegal transactions to or from the card without a local implementation.

VME Slave Module has ability to terminate VME data transfer cycles with VME_RETRYn or VME_BERRn signals. Those signals are used to inform the master that there is an error occurs during data transfer cycle and it should terminate the transaction as soon as possible. VME_RETRYn signal assertion also indicates that the transaction can not succeed at that time but the master should try the transaction again later. Thus, that means the local side is busy with another function so the master should reach the card again later.

VME Slave Module is also capable of requesting interrupt cycles on each of 7 defined levels in VME bus structure. Moreover, VME Slave Module could provide selectable 8/16/32 bit status/id information to interrupt handler on interrupt handling cycle. On electrically wise, the VME Slave Module can terminate interrupt requesting by ROAK or RORA mechanisms defined on VME Standard.

In addition to that, VME Slave Module has a capability to drive VME_SYSFAILn signal. This signal informs other VME Bus residents in the case of system failure. If an intelligent logic implemented to monitor fail cases on card that could affect total VME Bus functioning, it could drive VME_SYSFAILn via VME Slave Module and inform all the VME bus residents about the system failure.

These above solutions implemented and explained above are unique in VME Slave module when it is compared with other solutions [16], [17], [18], [19] - [25].

On the other hand, the design process of VME Slave Module was difficult. In order to overcome those difficulties, the programmable logic design was investigated in detail. Since the VME bus is asynchronous bus and the FPGA implementations are mostly synchronous implementations, there was a challenge in order to interface VME Bus with FPGA based structure. The problem of metastability also considered while designing VME slave Module and the VME Slave Module is designed as immune to metastability problems

The conceptual design is also compared and discussed with the other designs considering the performance parameters. After careful investigations the combination of asynchronous blocks with synchronous processes are combined for efficient and high performance module design. When the design is compared with [17], VME Slave module could respond VME bus cycles in the half time of the proposed method in [17]. This is a great performance increase. The new design approach is also validated by Mentor Graphics's HDL Designer Series Design Checker Tool. It is a tool to use to estimate the quality of design with appropriate VHDL coding style rules. On overall of % 84 design quality is reached. Therefore, it is validated that the coding style is still kept in well know and accepted VHDL coding style.

Furthermore, the VME Slave Module is not a vendor specific IP. In this thesis, the compilation and timing analysis reports of 3 different vendors can be found. Those vendors are ALTERA, XILINX and ACTEL. When the VME Slave Module is compared with the available IP's in market, it occupies nearly the same logic as the other IP's with its additional features. On the other hand, VME Slave Module has greater performance estimation values and could work with clock frequency as high as 142 MHz in ALTERA, 132 MHz in ACTEL and could reach up to 172 MHz in XILINX FPGA's. Therefore, the vendor independency of VME Slave Module and performance increase in implementation are also proven in this thesis work.

In this thesis work, not only a VME Slave Module IP was designed but also a VME Slave Card designed to test VME Slave Module and offer a concept for generic VME Slave Card design. On the designed card, modularity was the primary concern. The FPGA and peripherals are gathered on a small space on VME Slave Card, and the rest of the card is free for user selectable IC's and ASIC's. The control of those peripherals could be handled in FPGA since VME Slave Module occupies only a small space in FPGA. Moreover, there are PMC connectors for expansion purposes. They are routed according to ANSI/VITA 20-2001 standard

[37] and if the PCI initiator is implemented for FPGA, 64 bit PCI or PCI-X interface could be accomplished between FPGA and PMC. Also those paths between FPGA and PMC could be used for user defined protocols. Moreover, another 64 pin PMC connector is on card for I/O interface of Mezzanine card and P0 connector of VME. Thus, the user defined card could have communication via VME backplane and that paths are also routed according to ANSI/VITA 35-2000 standard [38]. Therefore, a standardized communication between FPGA and the PMC connectors are established.

To conclude, a VME Slave Module is designed as general purpose IP by using hardware description language VHDL. The designed Slave Module is compared with available IP solutions and new design approaches also considered. Moreover the designed card is also compared with the available generic VME Slave card solutions [16], [5] and [18]. As a result, the designed VME Slave Module is much more capable of the previous ones and there is a significant performance increase.

Moreover VME Slave Module and VME Slave Card are open for developments. As a future work, there could be developments on VME Slave Module, for instance new transaction cycles such as 2eSST (2 edge Source Synchronous Transaction – 320 MBytes/sec) and VME Renaissance (> 320 MBytes /sec) cycles could be implemented. Moreover, with some additions, this IP could be used as VME Master and maybe a System Controller. This project could also be considered as a start point for ASIC implementation of VME Module that is capable of being system controller, or master or slave.

REFERENCES

[1] “The VME Bus Handbook”, 4th edition, Wade D. Peterson, 1997

[2] “IEEE Standard for a Versatile Backplane Bus: VMEbus”, 1987

[3] “ANSI/VITA 1-1994 American National Standard for VME 64”, 1994

[4] “ANSI/VITA 1-1997 American National Standard for VME 64X”, 1997

[5] “An Evaluation of the VME Architecture for Use in Embedded Systems Education”, Ricks, Kenneth G. ; Jackson, David J.; Stapleton, William A.; ACM SIGBED Review Archive Volume 2, Issue 4 (October 2005) Special issue: The first Workshop on Embedded System Education (WESE), 2005 Pages: 63 - 69

[6] “Upgrade of the Level 1 Global Trigger System in the Belle Experiment”, Eunil, Won; Hyuncheong, Ha; Iwasaki, Y.;Nuclear Science, IEEE Transactions on Volume 55, Issue 1, Part 1, Feb. 2008 Page(s):122 – 125

[7] “A Digital Signal Processing Module for Gamma-Ray Tracking Detectors”, Cromaz, M.; Riot V.J.; Fallon, P.; Gros S.; Holmes, B.; Lee, I.Y.; Macchiavelli, A.O.; Vu, C.; Yaver, H.; Zimmermann, S.; Nuclear Instruments and Methods in Physics Research, Part A, July 2008

[8] “The CDF Level 2 Calorimetric Trigger Upgrade”, Bhatti, et al: Nuclear Instruments and Methods in Physics Research, Part A, July 2008

[9] “Readout of Silicon Strip Detectors with Position and Timing Information”, Fried, M.; Irmler, C.; Pernicka, M.; Nuclear Instruments and Methods in Physics Research, Part A, July 2008

[10] “A VME-Based Readout System for the CMS Preshower Sub-Detector”, Antchev, G.; Barney, D.; Bialas, W.; Da Silva, J.C.; Kokkas, P.; Manthos, N.; Reynaud, S.; Sidiropoulos, G.; Snoeys, W.; Vichoudis, P.; Nuclear Science, IEEE Transactions on Volume 54, Issue 3, Part 2, June 2007 Page(s):623 - 628

[11] “A VME Module for a Fast Readout of the Monolithics Analog Front-End Chips”, Boiano, C.; Guglielmetti, A.; Romoli, M.; Nuclear Science Symposium Conference Record, 2007, NSS '07, IEEE Volume 1, Oct. 26 2007-Nov. 3 2007, Page(s):340 - 341

[12] “The Micromegas Detector of the CAST Experiment”, Abbon, P. et al; New Journal of Physics, 2007

[13] “Enhancements in the Second Generation DIII-D Digital Plasma Control System”, Piglowski, D.A.; Ferron, J.R.; Gohil, P.; Johnson, R.D.; Penaflor, B.G.; Fusion Engineering and Design, 2007

[14] “A Second Generation Timing System for DIII-D Timing Control”, Deterly, T.M.; Kellman, D.H.; Finkenthal, D.F.; Fusion Engineering, 2007. SOFE 2007, 2007 IEEE 22nd Symposium on 17-21 June 2007, Page(s):1 – 4

[15] “Updating Legacy CAMAC-Based Data Acquisition System for Extended Pulse Duration of JT-60U”, Sato, M.; Kiyono, K.; Oshima, T.; Sakata, S.; Konoshima, S.; Ozeki, T.; Fusion Engineering and Design, 2008

[16] “A General-Purpose VME Module”, Guo, Y.N.; Gao, Z.W.; Chang, M.C.; Li, H.H.; Wang, M.Z.; Nuclear Science, IEEE Transactions on Volume 50, Issue 5, Part 3, Oct. 2003 Page(s):1752 - 1755

[17] “VHDL Synthesis for Space Applications – Implementing VME Core in Actel RadHard Devices Using Actmap VHDL”, McGaugh, Paul; Khan, Kamal, Military and Aerospace Applications of Programmable Devices and Technologies Conference, September, 1998

[18] “A Hybrid PCI/VME Architecture for Space”, Walls, B.; McClelland, M.; Persyn, S.; Aerospace Conference 2001, IEEE Proceedings. Volume 5, 10-17 March 2001 Page(s): 2227-2237

[19] “VME A24D16 Slave Controller”, IniCore, December 2002

[20] “VME A24D32 Slave Controller”, IniCore, December 2002

[21] “VME A32D32 Slave Controller”, IniCore, December 2002

[22] “MC-ACT-VME24/16”, MemecCore, February 2003

[23] “MC-ACT-VME32”, MemecCore, July 2003

[24] “64-bit VME Slave Controller Core, Intelop, 2002

[25] “VMEbus Slave Core Component”, MilLogic, December 2002

[26] “On a Programmable Approach to Introducing Digital Design”, Nixon, Mark S.; IEEE Transactions on Education, Vol. 40, No. 3, August 1997

[27] “Xilinx Programmable Logic Design Quick Start Handbook”, Xilinx, June 12, 2006

[28] “Programmable Logic: What's it to Ya?", Barr, Michael.; Embedded Systems Programming, June 1999, pp. 75-84.

[29] “ALTERA Quartus II Version 8.0 Handbook”, Altera, May, 2008

[30] “VME Growth Outpaces 2005 Projections in Military COTS Market”, Venture Development Corporation, July 12, 2006

[31] “Stratix FPGA Series Package & I/O Matrix”, Altera, July 2006

[32] “ALTERA Product Catalog”, Altera July 2008

[33] “Stratix II Device Handbook Vol1 and Vol2”, Altera, May 2007

[34] “Signal Integrity and Timing Issues of VME64x Double Edge Cycles”, Aloisio, A.; Branchini, P.; Cevenini, F.; Izzo, V.; Loffredo, S.; Lomoro, R.; Nuclear Science Symposium Conference Record, 2005 IEEE Volume 2, 23-29 Oct. 2005 Page(s):711 – 716

[35] “Asynchronous FPGA Risks”, Erickson, Ken; California Institute of Technology Jet Propulsion Laboratory, Pasadena, CA 91109, September 2000

[36] “Globally Asynchronous Locally Synchronous FPGA”, Royal, A.; Cheung, P.Y.K.; 2003

[37] “ANSI/VITA 20-2001 American National Standard for Conduction Cooled PMC”, 2001

[38] “ANSI/VITA 35-2000 AMERICAN NATIONAL STANDARD for PMC-P4 Pin Out Mapping to VME-P0 and VME64x-P2", 2000

[39] “I2C Bus Specification and User Manual”, Rev 03, NXP, June 2007

[40] “Universal Serial Bus Specification”, Revision 2.0, April 2000

[41] “Cypress Semiconductor’s EZ USB Technical Reference Manual”, Rev A, Cypress Semiconductor, 2008

APPENDIX A
DEMONSTRATION SETUP

In order to test the hardware on FPGA, a demonstration is setup. In Figure A-1, the demonstration setup is shown. There are two laptops. LAPTOP 1 is only connected to the VME Vanguard Analyzer in order to control VME Bus. VME Vanguard analyzer is a 6U VME Form factor card which can initiate transfers on VME bus. LAPTOP 2 is connected to the VME Slave Card with RS-485 serial channel. Moreover there is an USB link between LAPTOP2 and VME Slave add-on USB Interface Card.

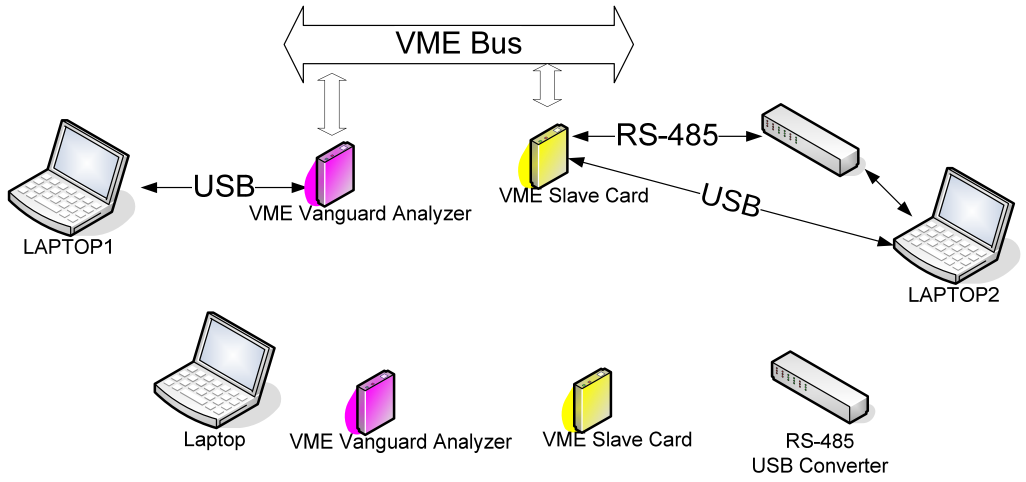


Figure A-1 Demonstration Setup

In Figure A-2, the internal bus connections of FPGA on VME slave Card is shown. There are 5 modules connected to Local Bus as slaves and those modules are responsible for interfacing FPGA with peripheral components. Each module is working as a bridge between Local bus and defined protocol for each peripheral component communication.

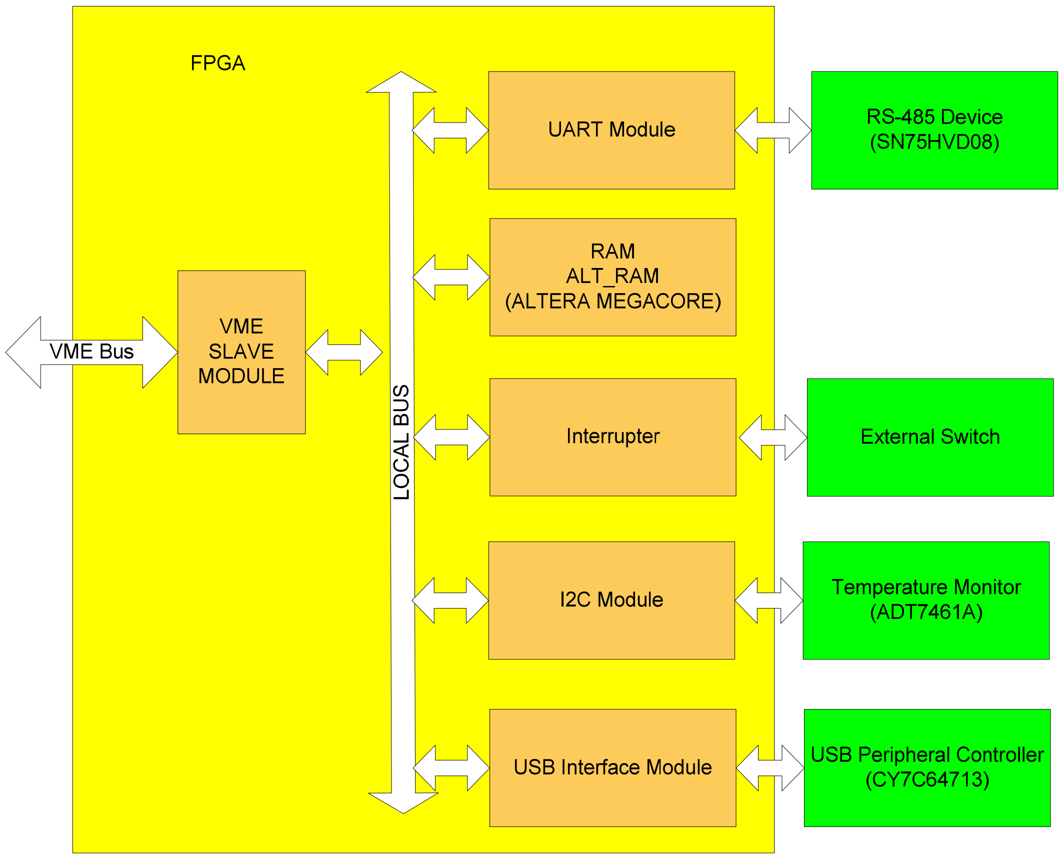

FPGA
VME Bus
VME SLAVE MODULE
LOCAL BUS
UART Module
RAM ALT_RAM (ALTERA MEGACORE)
Interrupter
I2C Module
USB Interface Module
RS-485 Device (SN75HVD08)
External Switch
Temperature Monitor (ADT7461A)
USB Peripheral Controller (CY7C64713)

Figure A-2 VME Slave Card FPGA Internal Bus Diagram

There is only one module which is called RAM module is not connected to the outside of FPGA since the RAM blocks are implemented in FPGA.

The memory map of VME Slave Card FPGA is in Table A-1. Each module can be reached with specific base address.

Table A-1 Memory Map of VME Slave Card FPGA

Adress Mode	**Base Address**	**Page**	**Module**	**Function**
A16	0x0000	1	RAM	RAM
A24	0x100000	2	UART	RS-485
A24	0x101000	2	I2C	Temperature Sensor
A24	0x102000	2	USB	USB Interface