# SAS EXPANDER FPGA EMULATION

by

Ivy Chow
Bachelor of Applied Science in Computer Engineering
University of British Columbia 2000

Project submitted in partial fulfillment of
the requirements for the degree of

Master of Engineering

In the
School of Engineering Science



SIMON FRASER UNIVERSITY

Fall 2005

# APPROVAL

Name: Ivy Chow

Degree: Master of Engineering

Title of Project: SAS Expander FPGA Emulation

Examining Committee:

Chair: Dr. Jim Cavers
Professor of the School of Engineering Science

---

Dr. Rick F. Hobson
Senior Supervisor
Professor of the School of Engineering Science

---

Sylvia Yu
Supervisor
Leader, Product Development
PMC-Sierra Inc.

---

Raymond Lam
Supervisor
Senior Engineer, Product Development
PMC-Sierra Inc.

Date Defended/Approved: Dec. 01/2005

# ABSTRACT

This paper presents my work on building an FPGA emulation system for prototyping PMC-Sierra's PM8387 SXP 36x3G 36-Port SAS expander. The employment of the FPGA emulation system allows rapid prototyping of designs, concurrent hardware and firmware development, and early interoperability and performance testing. This will provide significantly higher design confidence and ultimately will reduce the number of revisions and the time to market.

This paper starts by introducing the SAS and SATA technologies and describes the SAS expander's design specification. It then discusses the alternatives, benefits and the cost of an FPGA emulation system. Next it describes the emulation platform of SAS expander and outlines the entire process of implementing the emulation system, from HDL source code modification to the generation of the FPGA load. In the final sections, challenges, future developments and enhancements of the FPGA emulation platform are discussed.

*To my parents*

*for their love and support.*

# ACKNOWLEDGEMENTS

I would like to thank the following people for their support on this project:

- Dr. Rick F. Hobson
- Sylvia Yu
- Raymond Lam

# TABLE OF CONTENTS

**Approval** ..... ii

**Abstract** ..... iii

**Dedication** ..... iv

**Acknowledgements** ..... v

**Table of Contents** ..... vi

**List of Figures** ..... viii

**List of Tables** ..... ix

**Glossary** ..... x

**1 Introduction** ..... 1

**2 PM8387 SXP 36x3G SAS Expander Overview** ..... 3

2.1 SAS and SATA Overview ..... 3
2.2 SAS Protocol ..... 4
2.3 SAS Bandwidth ..... 6
2.4 SAS Expander ..... 7
2.5 PM8387 SXP 36x3G SAS Expander ..... 10
2.6 SAS Connection ..... 13

**3 Alternatives to FPGA Emulation** ..... 15

**4 FPGA Emulation Feasibility Considerations** ..... 18

4.1 FPGA Emulation Benefits ..... 18
4.1.1 Verification ..... 19
4.1.2 Concurrent Hardware and Firmware Development ..... 19
4.1.3 Interoperability Testing and Marketing Competitiveness ..... 20
4.1.4 Reduced Post-Silicon Test Time ..... 21
4.2 FPGA Emulation Efforts and Limitations ..... 22
4.2.1 Cost ..... 22
4.2.2 Debug Difficulty ..... 22

**5 FPGA Emulation Platform** ..... 24

5.1 SEAD-2TM ..... 25
5.2 SAS Expander Emulation Board ..... 26
5.2.1 Xilinx XC2V8000 Overview ..... 27
5.2.2 Xilinx XC2V8000 Architecture ..... 28

**6 SAS Expander Emulation System ..... 30**
6.1 HDL Source Code Modification ..... 30
6.1.1 Decreased the number of channels/ports ..... 31
6.1.2 Re-generated RAMs and removed all RAM bist logic ..... 32
6.1.3 Replaced all the DesignWare components ..... 32
6.1.4 Modified clock distribution ..... 33
6.2 Synthesis ..... 33
6.2.1 Design Constraints ..... 35
6.2.2 Device and Technology Options ..... 37
6.2.3 Synthesis Run ..... 38
6.3 FPGA Design Implementation ..... 41
6.3.1 NGDBuild ..... 41
6.3.2 MAP ..... 42
6.3.3 Place and Route ..... 43
6.3.4 TRACE ..... 44
6.3.5 BitGen ..... 44
6.3.6 PROMGen ..... 45
**7 Project Contributions and Challenges ..... 46**
**8 Conclusions and Recommendations ..... 48**
**Reference List ..... 50**

# LIST OF FIGURES


Figure 1 Serial Attached SCSI Protocols ..... 5
Figure 2 Expander Device Block Diagram ..... 8
Figure 3 A Maximum SAS Domain ..... 10
Figure 4 PM8387 SXP 36x3G Block Diagram ..... 12
Figure 5 PM8387 based SAS Expander System ..... 13
Figure 6 SAS Connection Sequence ..... 14
Figure 7 Solution Fit for Different Verification Requirements ..... 15
Figure 8 SAS Expander Emulation Platform ..... 24
Figure 9 SOC ASIC to SEAD-2 Board Mapping ..... 25
Figure 10 Virtex-II Architecture Overview ..... 28
Figure 11 FPGA Synthesis Flow ..... 34

# LIST OF TABLES


Table 1 Xilinx XC2V8000 Specification ..... 27

Table 2 Input and Output Files of FPGA Synthesis Process ..... 35

Table 3 Synthesis Performance Summary ..... 38

# GLOSSARY

| | |
|---|---|
| ASIC | *Application-Specific Integrated Circuit.* A semiconductor device targeted for a specific application. |
| ASSP | *Application-Specific Standard Product.* An ASIC that is sold off the shelf as a general product. |
| ATA | *Advanced Technology Attachment.* A disk drive interface standard for Integrated Drive Electronics (IDE). |
| DRC | *Design Rule Check.* The process that verifies that the implemented design meets all physical design rule requirements (spacing, drive strength, etc). |
| Formal Verification | The process that verifies two designs (e.g. pre synthesized and post synthesized) are equivalent by converting both designs into mathematical representations. |
| HBA | *Host Bus Adapter.* An adapter that sits at the host and allows a computer bus to be attached to another bus or a channel device (e.g. SATA). |
| LOS | *Loss of Signal.* |
| MRP | *Map Report File.* A Xilinx proprietary file that contains mapping and component usage information. |
| NCD | *Native Circuit Description.* A Xilinx proprietary file that contains the circuit information. |
| OOB | O*ut Of Band.* A way to describe signals that are not transferred through the main datapath but through a dedicated path. |
| PAR | *Place and Route.* The process that places the components of a design and routes the connections between the components. |

| | |
|---|---|
| PCF | *Physical Constraint File.* A Xilinx proprietary file that contains physical constraint information of the design. |
| RPC | *Remote Procedure Call.* A protocol that a program (client) can use to request a service from a program located in another computer (server) in the network where the networking details are abstracted from the protocol. |
| SATA | *Serial ATA.* An evolution of Parallel ATA that uses serial link to achieve higher speed. |
| SAS | *Serial Attached SCSI.* An evolution of SCSI that uses serial link to achieve higher speed. |
| SCSI | *Small Computer Serial Interface.* A system of connecting a series of computer peripherals to a computer. |
| SMP | *Serial Management Protocol.* A SAS management protocol which is used by SAS devices to communicate management information with other SAS devices. |
| SOC | *System On a Chip.* A chip that integrates multiple individual components of a system together. |
| SSP | *Serial SCSI Protocol.* A SAS protocol that defines the mapping of SCSI to support multiple initiators and targets. |
| STP | *Serial ATA Tunnelled Protocol.* A SAS protocol that defines the method by which a SAS host makes a connection to a serial ATA drive when a SAS expander is used. |
| Synthesis | The process that converts logical functions (e.g. in the form of RTL description) into physical gates (e.g. in the form of a netlist). |
| TWI | *Two Wire Interface.* An interface on a PC that is used by the host to communicate with low bandwidth devices such as sensors. |
| UART | *Universal Asynchronous Receiver-Transmitter.* A computer component that handles asynchronous serial communication. |

# 1 INTRODUCTION

The complexity and density of today's state-of-the-art Application-Specific Integrated Circuits (ASICs) have made proper verification of ASICs increasingly difficult. Quite often, those ASICs have to be verified in a system environment with embedded firmware, which makes it almost impossible to create an error free product on time using traditional Hardware Description Language (HDL) simulation. A more hardware-oriented verification solution is required.

PMC-Sierra's PM8387 SXP 36x3G SAS expander is a sophisticated and high density device where HDL simulation alone is not enough to provide confidence of the completeness of the verification effort. With the system level verification requirement, prototyping is the best solution. Over the past decade, the improvement of Field Programmable Gate Array (FPGA) device capacities has enabled the development of FPGA prototyping environments capable of implementing millions of logic gates, like the SXP 36x3G device. This FPGA emulation platform is the focus of this paper.

Section 2 provides an overview of the Serial Attached SCSI (SAS) and Serial ATA (SATA) technologies and the features of the SXP 36x3G device. Section 3 discusses alternatives to FPGA emulation. Section 4 discusses how FPGA emulation could benefit and enhance the development of sophisticated ASICs and the associated cost and efforts required to implement an FPGA emulation system. Section 5 describes

the SXP 36x3G emulation platform, which includes an off-the-shelf SEAD-2TM development board and an in-house designed emulation board. Section 6 describes the process of implementing the SXP 36x3G emulation system, which includes HDL source code modification, synthesis, and place and route. Section 7 describes my project contributions and challenges and section 8 concludes the paper by providing suggestions to future development and enhancement of the SXP 36x3G FPGA emulation platform.

# 2 PM8387 SXP 36X3G SAS EXPANDER OVERVIEW

Today's storage solution is dominated by SCSI (for enterprise level systems) and ATA (for consumer level systems). Although some enterprise storage systems adopt both standards, the fundamental differences between SCSI and ATA technologies require separate subsystems to be built. On the other hand, as disk storage density continues to increase with technological advancement, existing SCSI and ATA's parallel technology is limiting the throughput performance. Signal skew, crosstalk, signal termination and device addressability are all limiting factors to performance.

Serial Attached SCSI (SAS) and Serial ATA (SATA) interfaces are introduced to overcome these barriers. They both use serial technology to deliver faster, more reliable, and more scalable devices [1].

## 2.1 SAS and SATA Overview

SAS combines the proven functionality of SCSI with the performance of serial technology. In SAS-1.1, the SCSI protocol is transported over a serial interface capable of full duplex operation with a transfer rate of 3.0 Gb/s per direction. The next generation, SAS-2.0, is even faster with a transfer of 6.0 Gb/s per direction. Narrow ports allow for a single serial link, while wide ports support multiple links, allowing the aggregation of multiple links to increase the total available bandwidth. Designed to

leverage as much as possible from the existing SCSI effort, SAS provides faster device interconnect speeds, simpler point-to-point cabling that eliminates the shared bus bandwidth slowdowns, and improves system reliability.

SATA applies the serial technology on top of the ATA technology that is the common disk interface technology in desktop computers, entry-level servers and networked storage systems. SATA I is defined to be half-duplex and has a transfer rate of 1.5 Gb/s. The next generation, SATA II, increases the transfer rate to 3.0 Gb/s. SATA allows devices to be hot-plugged for easier replacement and it uses thinner cables to improve cooling in the box. Cyclical redundancy checking (CRC) is also used to enhance data reliability and the point-to-point cabling eliminates the confusing master/slave configuration of parallel ATA.

Both SAS and SATA are backward compatible with previous generation of SCSI and ATA software applications. In addition, SAS is developed to maximize compatibility with SATA by using a common physical and electrical connection interface with SATA. Therefore, it enables a single storage subsystem to accept either high performance, reliable enterprise-class SAS drives, or high capacity, cost-effective SATA drives.

## 2.2 SAS Protocol

The SAS protocol is divided into six layers: the physical layer, the phy layer, the link layer, the port layer, the transport layer, and the application layer [12]. The physical layer defines physical hardware such as cables and connectors and the SAS port consists

of all the layers except the application layer. Applications used to access parallel SCSI ports could be used to access SAS ports with no modification if only legacy features are used. Figure 1 shows the SAS architecture layers and the relationships between them.

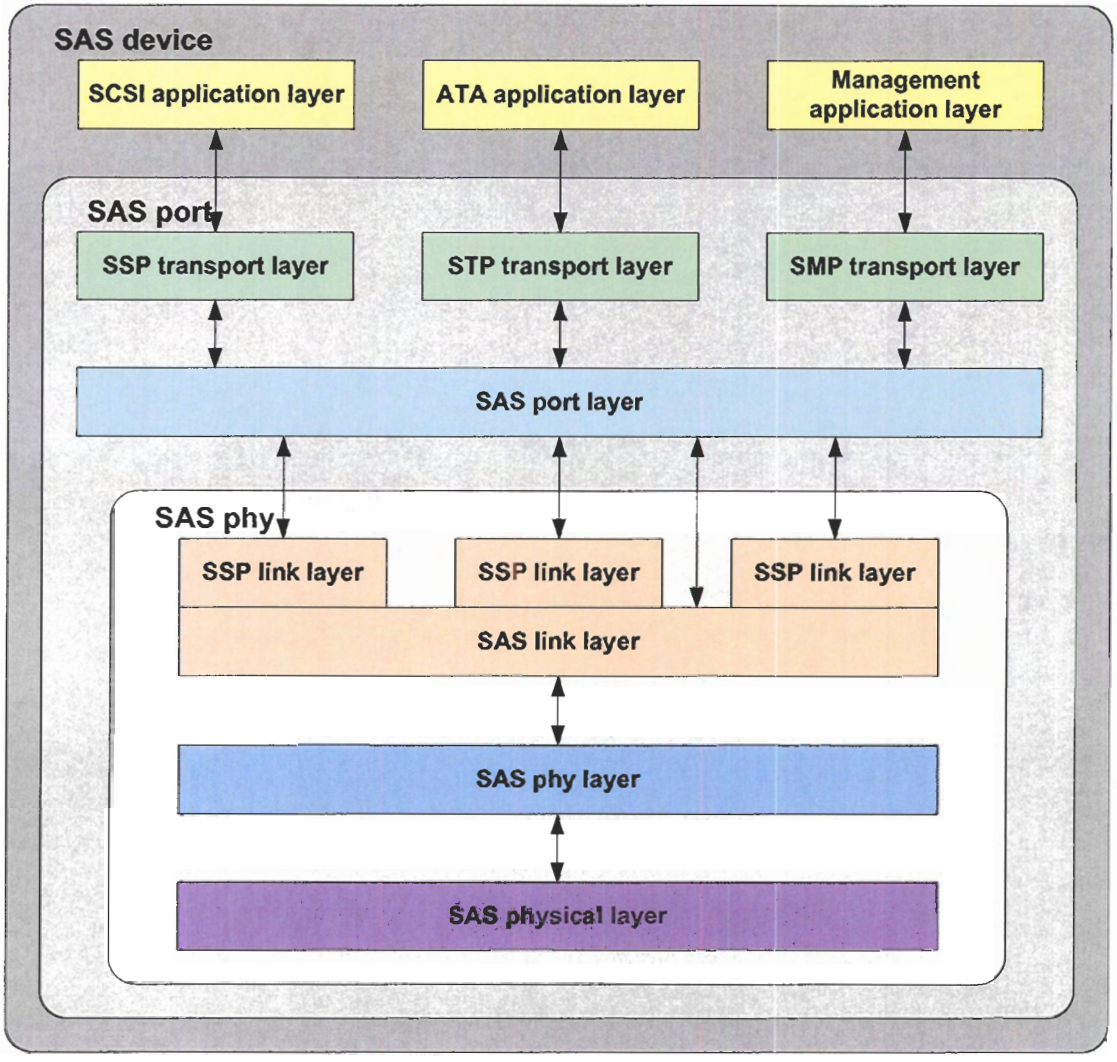


**Figure 1 Serial Attached SCSI Protocols** [1]

[1] Based on Maxtor, Serial Attached SCSI Architecture: Part 1—Introduction, the SAS Physical Layer and the SAS Phy Layer [4], page 3, figure 4

To support the co-existence of SATA and SAS drives, SAS defines three transport protocols. Serial SCSI Protocol (SSP) defines the mapping of SCSI to support multiple initiators and targets in a SAS domain. Serial ATA Tunnelled Protocol (STP) defines the method by which a SAS host makes a connection to a serial ATA drive when a SAS expander in a SAS domain is used. Serial Management Protocol (SMP) is a management protocol which is used by SAS devices to communicate management information with other SAS devices in a SAS domain.

Within a SAS domain, a SAS initiator port can function as a STP initiator port to communicate with SATA drives. The SAS port establishes a STP connection on behalf of the SATA drives so that SATA frames can pass through the connection from the initiator to the drive as if they were directly attached on a physical link. The SAS standard defines the process used during device discovery to determine the type of drive attached to any given port.

## 2.3 SAS Bandwidth

First-generation SAS link rate is 3 Gb/s (300 MB/s). The full-duplex, point-to-point architecture of SAS features simultaneous active connections among multiple SAS initiators and targets. SAS drives can transfer data in both directions at once to effectively double the usable bandwidth of the link rate to 6 Gb/s. For example, a device can simultaneously transfer data from a previously queued read operation while receiving data for a write operation. Narrow ports allow for a single serial link, while wide ports dramatically improve throughput by enabling several disks to communicate with a single

port address at the same time. Grouping four or eight links together, which is typical, can generate bandwidth of 12 Gb/s or 24 Gb/s respectively.

Currently, a 15,000-RPM disk drive will sustain data rates up to 75 MB/s [1]. At these sustained data rates, two disk drives will saturate a SATA 1.5 Gb/s bus. The shared Ultra320 SCSI bus supports a total of 320 MB/s or the sustained data rates of four to five disk drives. By contrast, a 4-wide SAS port supports as many as 16 hard drives before becoming saturated.

System architects normally optimize performance by removing bandwidth bottlenecks – an objective normally met by matching interleaving technologies with complementary efficiency and availability levels. Similar to SAS, PCI Express supports scalable performance by combining multiple data links to create wide data paths. Each PCI Express lane supports 2.5 Gb/s with scalability up to a 32-wide lane configuration [1]. This common capability is the key to optimizing performance between SAS and PCI Express. By combing SAS with PCI Express, the aggregate storage system performance can be maximized. For example, a 4-wide SAS port supports 1200 MB/s for up to 16 hard disk drives. With the SAS initiator attached to an 8-lane PCI Express slot, the entire data path through the system is optimized to support 2 GB/s.

## 2.4 SAS Expander

SAS ports that are physically a part of a SAS Host Bus Adapter (HBA) can directly address either SAS or SATA drives. The number of addressable drives would be restricted by the number of physical ports integrated into the HBA itself if there were no

mechanism for expanding the topology. Integrating a large number of SAS ports into one device could be costly for systems not requiring all the ports, however not providing enough SAS ports would significantly limit the utility of the systems. In light of this limitation, the SAS standard also defines a type of intermediary device called an Expander, as shown in Figure 2 [13].

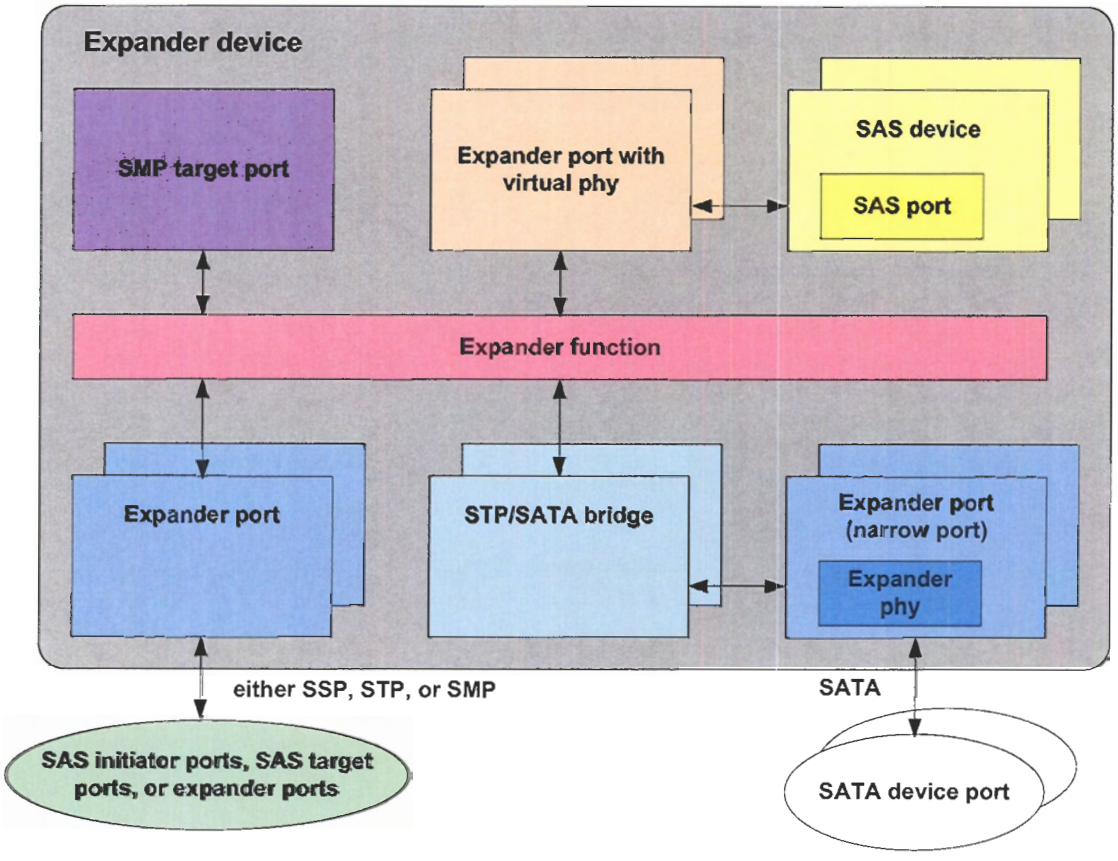


**Figure 2 Expander Device Block Diagram** [2]

[2] Based on American National Standard, Information Technology – Serial Attached SCSI -1.1 [3] page 42 , figure 18

The SAS Expander device can communicate with multiple SAS devices and it functions as a switch to simplify configuration of large systems that can be scaled with minimal latency while preserving bandwidth for increased workloads. It enables highly flexible storage topologies of up to 16,256 mixed SAS and SATA drives in a single SAS domain as shown in Figure 3 [4]. SAS edge expanders may connect up to a total of 128 devices in one domain, including SAS initiators, SAS targets, SATA devices, another SAS edge expander, or a SAS fan-out expander. SAS fan-out expanders may connect up to a total of 128 SAS initiators, SAS targets, SATA devices, or SAS expander devices in one domain, although only one SAS fan-out expander device is allowed in a single SAS domain. SAS expanders can incrementally expand in-box and near-box storage capabilities in systems requiring greater bandwidth connections, as additional expanders provide redundancy and support a large number of devices. In order to establish a connection with a SAS/SATA device or another expander device, all SAS expanders support an addressing mechanism for routing requests to manage the connections between devices and the ability to broadcast primitives across all the expanded connections they support.

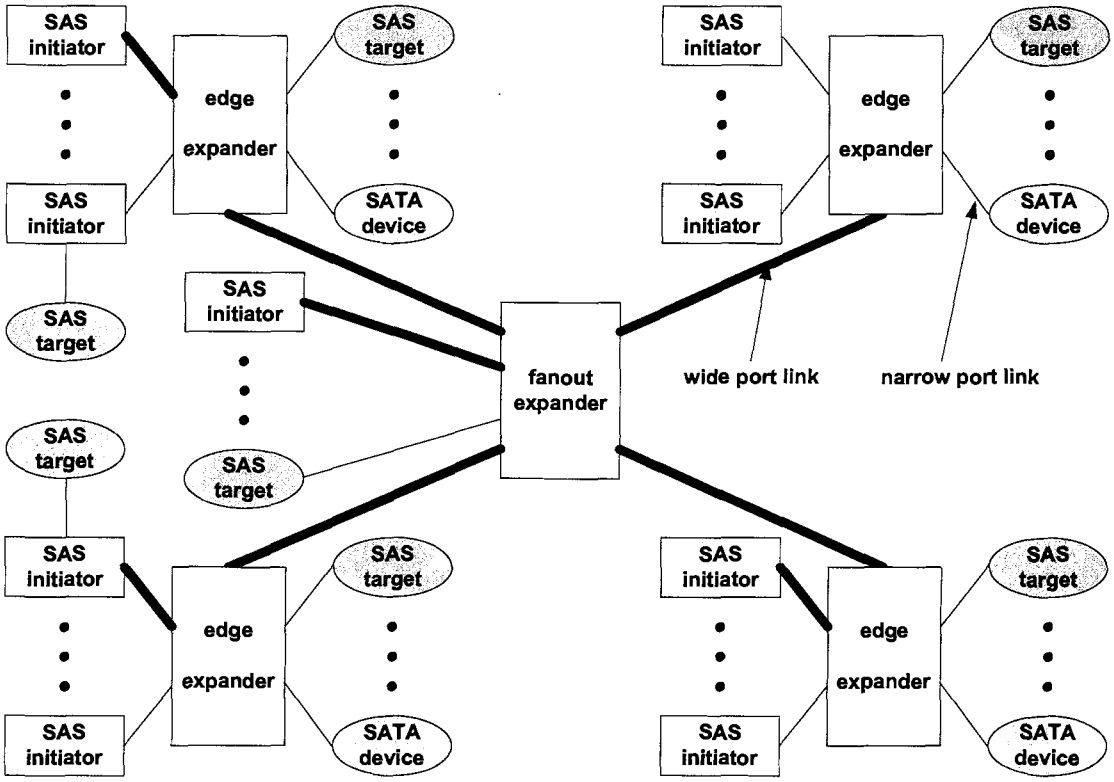


**Figure 3 A Maximum SAS Domain** [3]

## 2.5 PM8387 SXP 36x3G SAS Expander

The PMC-Sierra's PM8387 SXP 36x3G device is implemented with 0.13-micron CMOS technology in a 27 mm x 27 mm 352-pin CSBGA package [9]. It is a SAS Expander device with integrated storage enclosure processor that conforms to the standard developed by the ANSI International Committee for Information Technology Standards (INCITS) T10 Technical Committee [3].

[3] Based on Maxtor, Serial Attached SCSI Architecture: Part 1—Introduction, the SAS Physical Layer and the SAS Phy Layer [12], page 2, figure 2

Figure 4 shows the block diagram of the SXP 36x3G device. It contains 36 independent external expander ports with integrated non-blocking crossbar switch that allows point-to-point connection with 1.5 Gb/s and 3 Gb/s data transfer rates. It features low latency connection arbitration and arbitrary SAS wide port and narrow port configurations. Different routing methods are supported, including table routing, direct routing, and subtractive routing. The STP bridge function allows SATA devices to be attached to any target ports concurrently. The integrated RISC processor supports SMP functions, and allows connection to external components such as fan sensors and temperature sensors to complete the SCSI Enclosure Services (SES) functions. Three configurable Two Wire Interfaces (TWI) are provided for device configuration and control of peripheral devices. An Universal Asynchronous Receiver-Transmitter (UART) interface is also implemented to support serial debugging and a serial GPIO (SGPIO) interface is implemented to provide expansion for up to 144 GPIOs.

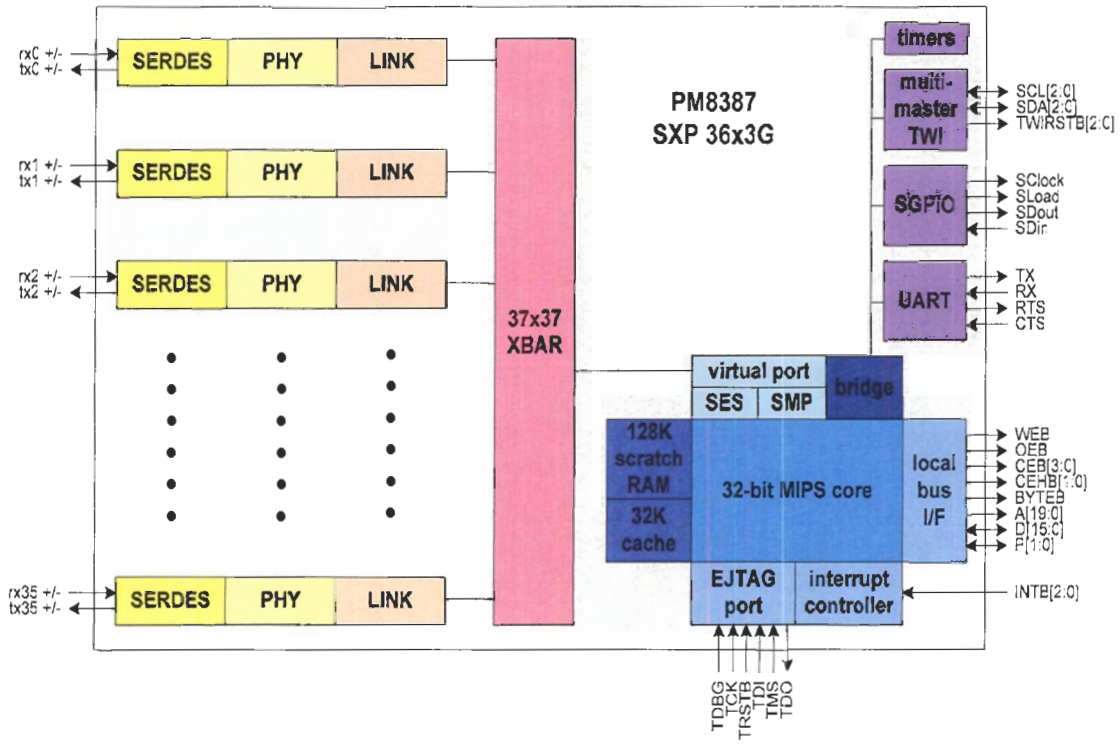


**Figure 4 PM8387 SXP 36x3G Block Diagram** [4]

PM8387 SXP 36x3G offers tiered storage systems increased system performance, higher capacity, and scalability with an embedded storage enclosure processor. It enables storage system architectures to extent to hundreds of SAS and SATA hard disk drives with low-latency and high performance connections. It provides a low cost solution to large and manageable tiered storage systems such as near-line backup, email archive, imaging and financial data retention. Figure 5 shows PM8387 based SAS Expander System in typical server storage application.

[4] Based on PMC-Sierra Inc., PM8387 SXP 36x3G 36-Port SAS Expander Short Form Data Sheet [9], page 1, figure 1

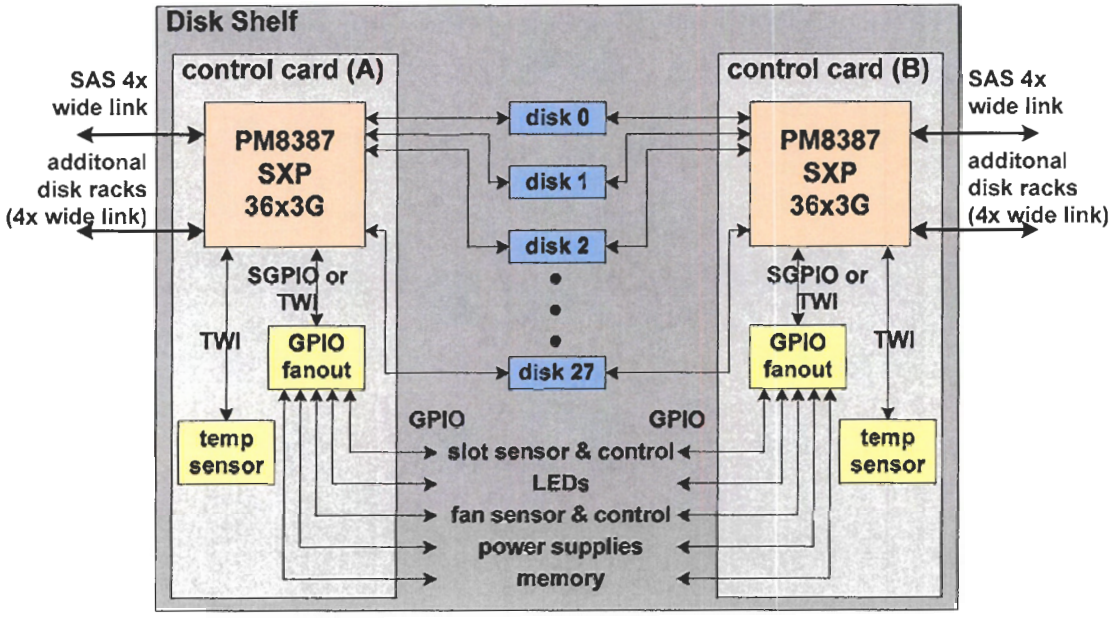


**Figure 5 PM8387 based SAS Expander System** [5]

## 2.6 SAS Connection

In SAS, each connection is a temporary association between an initiator phy (HBA) and a target phy (SAS drive). Figure 6 shows a typical SAS connection sequence. The source phy transmits an OPEN address frame, which contains a destination SAS address. When the expander receives the OPEN address frame, it looks up the destination SAS address in its routing tables and arbitrates for internal access to an outgoing expander port with a path to the destination SAS port. The expander transmits Arbitration In Progress (AIP) primitives to the initiator while it is performing arbitration. Once arbitration is completed, the expander routes the OPEN address frame to the

[5] Based on PMC-Sierra Inc., PM8387 SXP 36x3G 36-Port SAS Expander Short Form Data Sheet [9], page 2, figure 2

matching destination port. When the target device receives the OPEN address frame, it validates the frame and replies with an OPEN_ACCEPT primitive if it is able to accept the connection. The connection is established when the initiator receives the OPEN_ACCEPT primitive forwarded by the expander. There are three types of connections determined by the OPEN address frame, one for each type of protocol (SSP, STP, and SMP) supported by the port. Each of the connection types uses its own protocol for the SAS link layer once the connection is established. The connection is closed when both sides exchange CLOSE primitive.

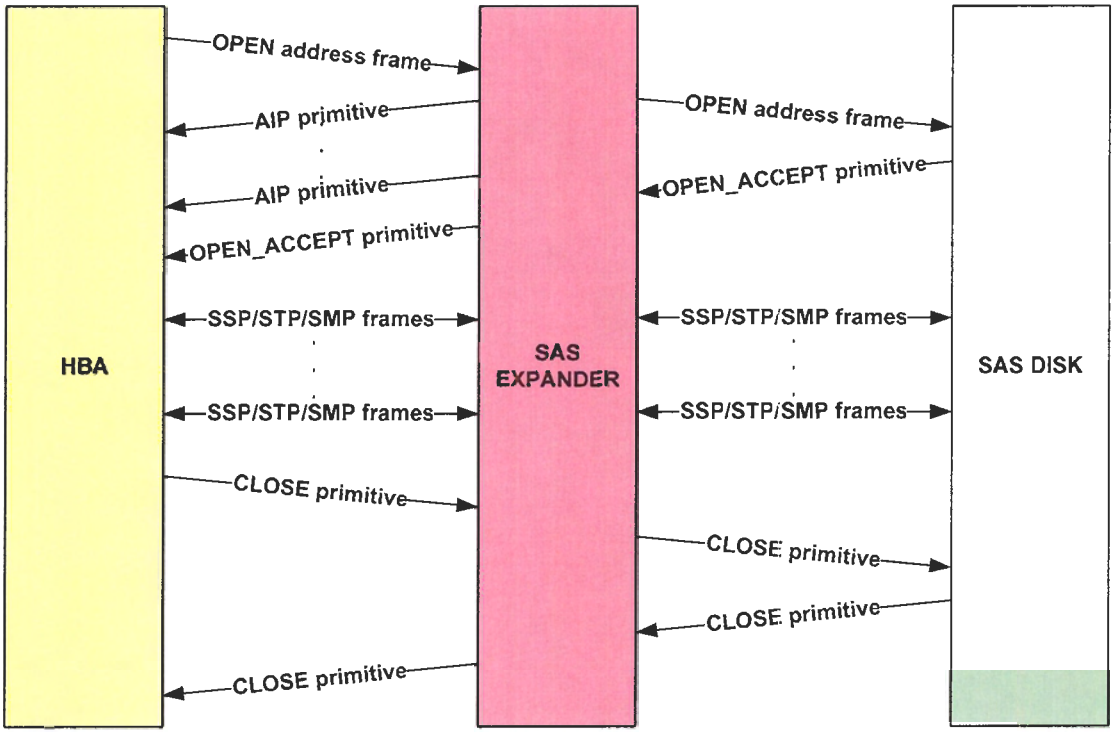


**Figure 6 SAS Connection Sequence**

# 3 ALTERNATIVES TO FPGA EMULATION

Many types of ASIC verification environment exist and they serve different purposes. Figure 7 shows the solutions fit for different verification requirements [8].

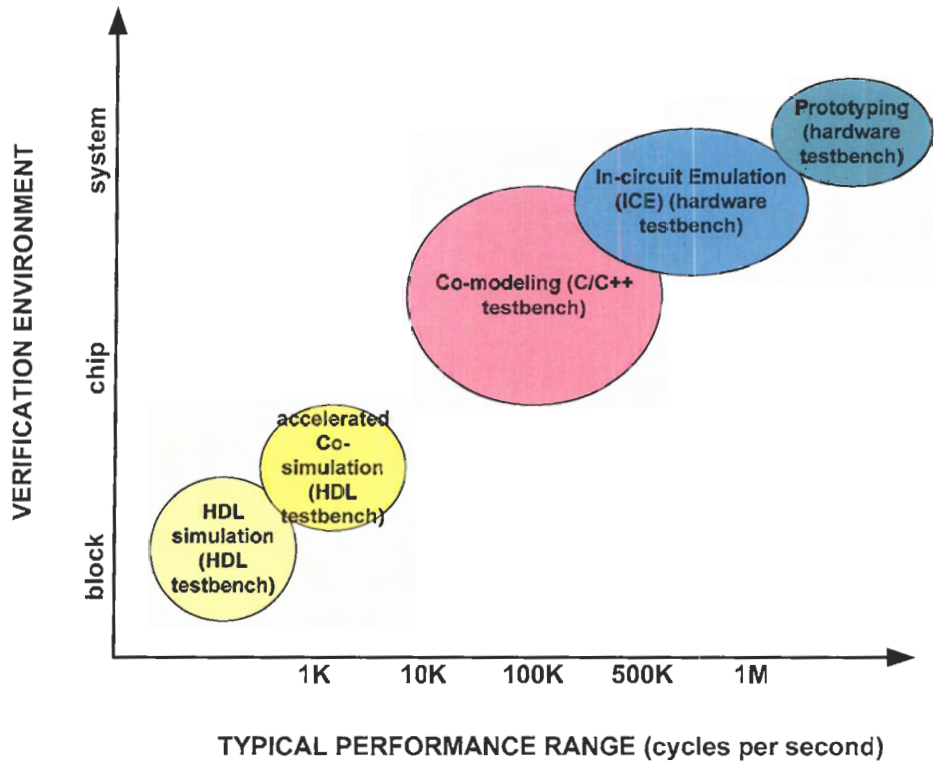


**Figure 7 Solution Fit for Different Verification Requirements** [6]

In HDL simulation, testbenches and testcases are created and a software simulator is used to exercise different features of the design. Both the design and the testbench are

[6] Based on Mitch Dale, The Value of Hardware Emulation [8], page 1, figure 1

compiled into a proprietary database that the simulator understands. In recent years, testbench languages like Vera and Specman have become very popular because they simplify the creation of the testbench by providing features like random data generation and object inheritance. HDL simulation serves as the best way to verify a low level block of a design because a lot of low level corner cases cannot be easily tested at a higher level.

In accelerated co-simulation, the design is compiled into a piece of hardware called the accelerator which allows simulations to run much faster. Some accelerators allow testbenches to be compiled into hardware to provide even faster acceleration. Accelerators have lost their appeal in recent years because they require separate compilation and the cost associated with the accelerated simulation cannot be justified all the time.

Co-modelling involves simulating the system (device and its surrounding environment). Quite often, different pieces of the system are written as C models because they are easier to write and they simulate faster. Third party models are also available to allow designers to focus on the actual design. Co-modelling is also effective at the architecture phase where complex algorithms can be implemented in C models first to see how the different parameters of the design affect performance and interaction with the rest of the system.

An In-Circuit Emulation (ICE) system typically consists of hundreds of FPGAs or processors inside a box to provide hardware acceleration and in-circuit emulation in a

single system. For example, the Incisive Palladium II system by Cadence has a maximum capacity of 256 million gates and operates up to 1.5MHz [4]. Compared to an FPGA prototyping system, the main advantage of an ICE system is that it supports dynamic probes to allow interactive debug during run-time so that designers can get visibility into the design to root-cause issues faster than an FPGA system once the bug can be reproduced in the ICE system. However, the huge cost of an ICE system (typically a few million dollars) and the slower speed are the big disadvantages.

# 4 FPGA EMULATION FEASIBILITY CONSIDERATIONS

Although FPGA technology has improved dramatically since the last decade, it has not been used for emulation until recently. Many people see FPGA emulation as a complicated step that adds additional cost to tightly budgeted projects. However, with the advancement of the software used to program FPGAs, the availability of many common Intellectual Properties (IPs) and the large capacity of FPGAs, FPGA prototyping is becoming a very attractive option. In addition, a number of third party systems are available that allow designers to focus on the actual verification task by providing boards with FPGAs that are connected to an advanced software debugger and are capable of interacting with existing simulators and testcases [13].

In order to obtain the full benefit of FPGA emulation, planning needs to start at the beginning of a project and feasibility studies need to be performed on a project-by-project basis. The following sections discuss the factors to consider.

## 4.1 FPGA Emulation Benefits

In the semiconductor industry, success in the marketplace depends on the ability to produce novel, complex, and working designs faster than the competition. Employing a well planned FPGA emulation system can definitely contribute to higher quality design and verification cycles and reduce the time to market.

### 4.1.1 Verification

As the design gets more complicated, simulation effort increases exponentially in terms of resources required and simulation time. Quite often, random traffic testing is required to exercise the design extensively and simulation is simply too slow to provide enough cycles. Although some of the popular simulators allow simulation to be run on multiple parallel processors, significant effort and planning are required to partition the design and resolve the problem of linking and synchronizing separate simulations running on different processors. For example, Remote Procedure Call (RPC) can be used to handle the synchronization between different portions of the design in separate processors through ModelSim's Foreign Language Interface (FLI) [6]. The complexity of the synchronization and requirements to have balanced partitioning of the design prevent parallel computing to be leveraged in current simulation environment. As a result, simulation is still the bottleneck of most design cycles. Although emulation speed is not as fast as the real device, it provides a promising solution to cut down on the verification cycle.

For the SXP 36x3G device, full random traffic is being passed in FPGA emulation, revealing a few bugs that otherwise will not be caught in simulation.

### 4.1.2 Concurrent Hardware and Firmware Development

Traditionally, because of the unavailability of the actual hardware device, firmware debugging can only begin after silicon arrival. This is acceptable when the device is hardware oriented and its firmware is relatively straightforward. However, as

ASIC designs get more complicated and many of those are system-on-a-chip (SoC) solutions, many of the major processing tasks are off-loaded and handled by firmware. Consequently, the required firmware becomes more complicated and takes more time to verify. It is therefore preferable to start firmware development concurrently with the hardware development despite the difficulty to debug firmware in the absence of the actual hardware. There is also the possibility that while verifying the firmware, a design flaw is discovered in the hardware and changes need to be made. FPGA emulation provides an excellent and reliable environment for firmware engineers to start code development and testing before the final ASIC device is available.

For the SXP 36x3G device, there is an integrated RISC processor to handle SMP functions. Because of the deployment to the FPGA emulation, several revisions of firmware are done prior to the silicon arrival. By the time silicon is available, firmware is already robust enough to support all the major features.

### 4.1.3 Interoperability Testing and Marketing Competitiveness

Most ASIC devices do not operate as standalone devices. Devices developed by different vendors within a system need to communicate with each other through a standardized protocol or specification. Despite efforts by different device vendors to implement the design according to specification, it is not uncommon for two devices implementing the same standard to be incompatible with each other due to a slightly different interpretation of the standard. Early interoperability testing using FPGA emulation reduces the risk of failure in interoperation with peer systems. FPGA

emulation can also be used as a prototype for customers to start board development at an earlier time. Hence, FPGA emulation becomes an important strategic component for marketing to raise customers' confidence in the design and seek early commitment from customers.

For the SXP 36x3G device, the emulation platform successfully demonstrated system interoperability with multiple SAS server, storage enclosure and controller vendors at the interoperability forum SAS Plugfest. Achieving this with multi-vendor SAS systems assures OEMs of an interoperable solution for next-generation SAS servers and server attached storage systems.

### 4.1.4 Reduced Post-Silicon Test Time

All new devices must be validated according to internal and customer requirements by performing a series of planned tests. Without any kind of pre-silicon prototyping, the tests themselves cannot be verified. FPGA emulation helps to pull-in the time required for the device validation process by providing a reliable environment to ramp up product validation engineers on the device and to debug software and firmware, prior to silicon arrival. Some low-risk features can also be validated using the emulation platform.

For the SXP 36x3G device, the FPGA emulation system allowed the product validation engineers to finish validation of the device ahead of schedule.

## 4.2 FPGA Emulation Efforts and Limitations

Despite all the benefits, FPGA emulation does require significant effort and additional cost to build the hardware platform and has several limitations.

### 4.2.1 Cost

The cost of implementing an FPGA emulation system includes the actual cost of the FPGAs and the associated hardware components, software tools and licenses, and labour in driving the entire process. Using the SXP 36x3G project as an example, each emulation platform consists of two boards: a custom board with a Xilinx XC2V8000 FPGA and an off-the shelf SEAD-2TM board. Each of those two boards costs about $20,000. The synthesis tool for FPGA emulation is different from the tool used for ASIC design and therefore additional funding needs to be allocated. The FPGA emulation lasts as long as the chip development and involves labour from different groups: product validation engineers to design the custom board, CAD engineers to support the tools and flows, and product development engineers to build the FPGA load, bring up the emulation platform, and run the tests. Nevertheless, the overall cost of building an FPGA emulation system should not be as costly as a chip revision when critical bugs are found that require re-spinning of the chip.

### 4.2.2 Debug Difficulty

Compared to simulation, FPGA emulation is relatively difficult to debug because of very limited visibility into the internal signals. Making internal signals observable

requires changes to the source code and an additional place and route cycle for wiring the internal signals to the spare debug ports. In addition, because of the lack of accessibility to force the internal signals into certain states, it is hard to hit corner cases predictably using the FPGA emulation platform. Therefore, FPGA emulation is good for finding issues with the design, but not good for root cause investigation. It cannot replace conventional verification effort, which provides post simulation debugging and relatively easy reconstruction of different events by putting the device in a specific state. This is the exact use model of FPGA emulation for the SXP 36x3G project.

# 5 FPGA EMULATION PLATFORM

The SXP 36x3G emulation platform consists of two boards: off-the-shelf SEAD-2TM development board and in-house designed SAS Expander emulation board with Xilinx XC2V8000 FPGA. The two boards are connected by 32-bit wide Mictor cables as shown in Figure 8.

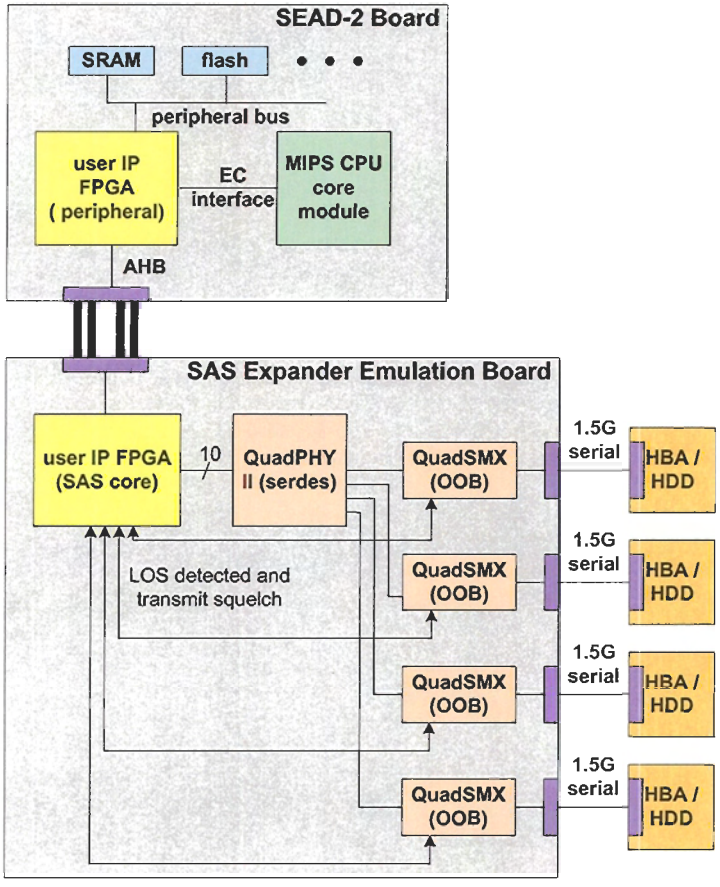


**Figure 8 SAS Expander Emulation Platform**

## 5.1 SEAD-2TM

The SEAD-2TM board is developed by MIPS Technologies to provide a platform for integration of the user's hardware IP design with MIPS32® core. There are two FPGAs on the SEAD-2 TM board. One FPGA is preloaded with MIPS4KEm CPU from an unencrypted source, the other is a two-million gate Xilinx SCV2000E FPGA where the user's design is implemented. The SEAD-2 board also provides a USB interface, an UART interface, 4 MB of SRAM, 32 MB of flash memory, and 32 MB SDRAM modules. In addition, the SEAD-2 board features low-level debugging aids by supporting EJTAG debugger connectivity. Figure 9 shows how a SoC ASIC design can be mapped into the SEAD-2 TM board and communicates with different peripherals.

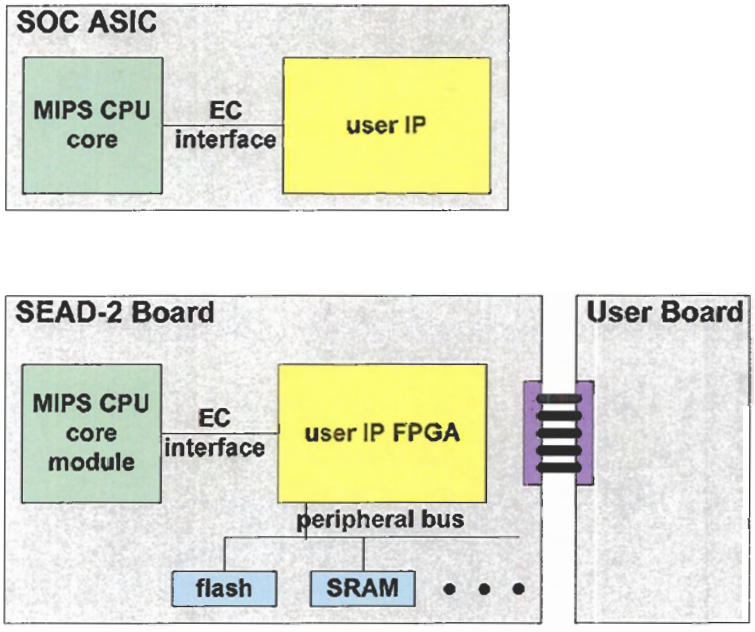


**Figure 9 SOC ASIC to SEAD-2 Board Mapping** [7]

[7] Based on MIPS Technologies, Sead-2 Board User's Manual, Revision 01.02, March 31, 2004 [7], page 4, figure 1

In the SXP 36x3G emulation platform, the Xilinx FPGA on the SEAD-2TM board implements the UART interface, the Local Bus interface, watchdog and general timers, EC/AHB Bridge, AHB Decoder, and AHB/APB Bridge.

## 5.2 SAS Expander Emulation Board

The SAS Expander Emulation Board mainly consists of a Xilinx XC2V8000 FPGA, PM8354 QuadPHY II device [10], and PM8380 QuadSMX 3G device [11]. Xilinx's XC2V8000 is an eight-million gate FPGA where most of the SXP 36x3G's functionality, including the SAS physical layer, link layer, and routing functions, are implemented. Because SXP 36x3G emulation platform requires 1.5 Gb/s links and the Xilinx FPGA is not capable of these I/O rates, a SERDES device QuadPHY II is used as a serializer/deserializer. It serializes the 10-bit parallel data and transmits it differentially at 1.5 Gb/s in the transmit direction and recovers the serial differential data and converts it back to 10-bit parallel data in the receive direction. To support Out-Of-Band (OOB) signalling, QuadSMX 3G is used on the serial-side of the QuadPHY II. QuadSMX 3G supports loss of signal (LOS) detection and is capable of generating OOB squelch events to define the messages during OOB signalling period in the transmit stream. These signals are routed to the Xilinx FPGA so that SXP 36x3G can control OOB signalling events. Depending on the time between LOS conditions, SXP 36x3G can determine the OOB states of the hard drive or HBA that is transmitting to SXP 36x3G. Based on the incoming state, SXP 36x3G will generate outgoing OOB squelch events through the QuadSMX 3G.

### 5.2.1 Xilinx XC2V8000 Overview

Xilinx XC2V8000 is a member of the Virtex-II family developed for high-performance and high-density designs, especially those that have lots of IP cores and customized modules. Xilinx XC2V8000 uses 0.15μm CMOS technology with 8-layer of metal. It can implement eight million gates with clock speed running at up to 400MHz, and it supports I/O speed up to 840Mb/s. The following table summarizes the specification of Xilinx XC2V8000 [17].

**Table 1 Xilinx XC2V8000 Specification** [8]

| | |
|---|---|
| Maximum clock speed | 400 MHz |
| I/O speed | 840 Mb/s |
| Maximum number of User I/O pads | 1108 |
| Number of logic cells | 104832 |
| Number of SelectRAM blocks | 168 |
| Total SelectRAM Memory | 3,024 Kbits |
| Number of digital clock managers | 12 |
| Number of multipliers | 168 18x18 |
| CLB Array: Row x Column | 112 x 104 |
| Number of Slices | 46,592 |
| Number of LUTs | 93,184 |
| Number of Flip-Flops | 93,184 |

[8] Xilinx, Virtex-II Platform FPGAs: Complete Data Sheet, DS031 (v3.4) [17], October 14, 2003

### 5.2.2 Xilinx XC2V8000 Architecture

Xilinx Virtex-II architecture is optimized for high-density and high performance logic designs. As shown in Figure 10, the programmable device is comprised of input/output blocks (IOBs) and internal configurable logic blocks. Programmable I/O blocks provide the interface between package pins and the internal configurable logic. The internal configurable logic block includes the Configurable Logic Blocks (CLBs), Block SelectRAM, Multiplier blocks, and Digital Clock Manager (DCM) blocks organized in a regular array [17].

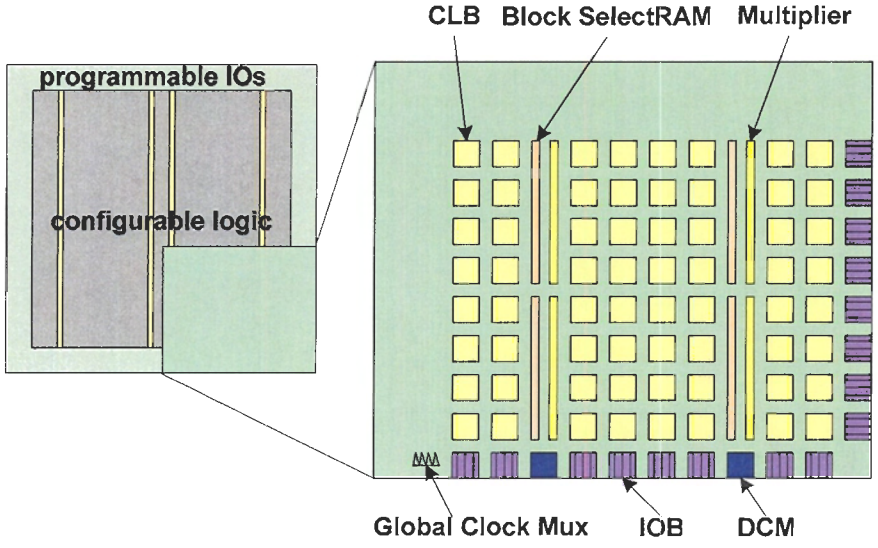


**Figure 10 Virtex-II Architecture Overview** [9]

The CLBs provide functional elements for combinational and sequential logic, including basic storage element. The Block SelectRAM memory modules provide large 18 Kbit storage elements of true dual-port RAM with two independently clocked and

[9] Based on Xilinx Virtex-II Platform FPGAs: Complete Data Sheet [17] page 3, figure 1

controlled synchronous ports. Multiplier blocks are 18 by 18 bits which are optimized for high-speed operations and have low power consumption. The DCM blocks provide self-calibrating, fully digital solutions for clock distribution and delay compensation, clock multiplication and division, and coarse-grained and fine-grained clock phase shifting.

The Xilinx Virtex-II FPGA has 16 global clock buffers and supports 16 global clock domains. Eight clock buffers are in the middle of the top edge and eight are in the middle of the bottom edge. Each device is divided into four quadrants. Any of these 16 global clock buffers can be used in any quadrant, up to a maximum of eight clocks per quadrant.

The remaining sections of this paper are focused on the implementation of the FPGA load for the Xilinx Virtex-II FPGA on the SAS Expander Emulation Board. This includes most of the complex core functions of the ASIC.

# 6 SAS EXPANDER EMULATION SYSTEM

The process of developing an FPGA emulation system is similar to but simpler than the digital IC design flow and they are developed concurrently. Implementation of the SAS Expander emulation system involves HDL source code modifications to scale down the size of the logic to fit the FPGA, synthesis to convert HDL into a netlist, and physical design to place and route the netlist into the FPGA.

## 6.1 HDL Source Code Modification

The major reason of modifying HDL source code is to map the real HDL into the FPGA device, which has limited capacity and runs at a slower speed. However, HDL source code modification should be minimized such that there would be little difference between the design in the prototype and the design in the actual silicon. This is important because one of the goals of FPGA emulation is design verification and risk reduction for the silicon as mentioned in Section 4. If the HDL code differs significantly, the emulation effort would be verifying a different design compared to the actual silicon, and the benefits derived from emulation would be substantially reduced. Therefore, engineers designing the silicon should keep in mind that the design should not only be achievable using the target technology, but also be mappable into the desired FPGA. The engineers should realize that FPGA is register rich but is poor in complex function implementation and runs a lot slower than Application-Specific Standard Product (ASSP)

gates. Although the FPGA tools can balance registers and move logic across clock boundaries, it is better for designers to put additional timing stages for complex logic to alleviate timing and routing problems if the latency hit is acceptable. In addition, adherence to the synchronous design philosophy allows a design to migrate from ASSP to FPGA easily. FPGA does not have good buffer or delay insertion tools available to control the delay between the latches and flip-flops to meet timing requirements for an asynchronous design.

The following are the major areas that require HDL modifications for the SXP 36x3G SAS expander emulation system.

### 6.1.1 Decreased the number of channels/ports

The SXP 36x3G is a 4.8-million gate device which is too big to be mapped entirely into an FPGA. Each channel operates independently and so reducing the number of channels is the best solution to the problem. However, the number of channels should be enough to support testing of different system configurations.

The Xilinx FPGA is composed of four quadrants and each quadrant can fit one port of the design to allow up to four device ports to be available in the FPGA. The top-level HDL of SXP 36x3G is therefore stripped down to include only four ports. Having four ports is enough to enable emulating SXP 36x3G as a fan-out expander and edge expander by cascading two emulation platforms together. Both fan-out and edge functionalities can be tested in narrow-port and wide-port configurations.

### 6.1.2 Re-generated RAMs and removed all RAM bist logic

As the RAM models instantiated in the HDL source code are technology-specific, they cannot be used for FPGA emulation and must be replaced. Xilinx provides a graphical interactive design tool called CoreGen to create different memory models using Block SelectRAM or the basic storage element depending on requirements. For each memory model, CoreGen generates a command file (*.xco), a memory file (*.mif), and a vhdl file (*.vhd). The modules created with CoreGen can be instantiated in Verilog/VHDL source code as "black boxes" for use with Synplify Pro. All the instantiations of RAM in the HDL source code need to be modified to instantiate those RAM models.

Besides the RAM models, the RAM BIST components and logic, and DFT related wrappers need to be removed because they are not supported and are no longer needed.

### 6.1.3 Replaced all the DesignWare components

A DesignWare component is a verified, silicon proven, and synthesizable intellectual property. It can be implemented in a number of ways and is integrated into the Synopsys synthesis environment to provide more optimization flexibility in terms of performance and area. It is commonly used in designs with a large amount of high-speed datapaths to improve design reliability. However, the absence of DesignWare component in the FPGA library forces all usage of those components to be replaced by functionally equivalent logic.

In the SXP 36x3G, the DesignWare component DWF_prienc is used in the design. This is a priority encoder that can be mapped into one of the high-performance

components optimized for either timing or area. It is replaced by a regular priority encoder consisting of generic combinational logic.

### 6.1.4 Modified clock distribution

A Xilinx Virtex-II global clock buffer (BUFG) must be instantiated at the root location of each clock tree, although major synthesis tools can now automatically infer the BUFG when the corresponding input signal is used as a clock in the HDL source code. If the number of clock domains exceeds the number of global clock buffers available, the design must be partitioned carefully to divide individual domains into FPGA quadrants. Also, all internal clock dividers must be replaced by DCMs to guarantee reliable timing between the source clock and the divided-down clock.

## 6.2 Synthesis

For SXP 36x3G emulation system, Synplify Pro (v 7.6.1) developed by Synplicity, Inc. is used to synthesize the HDL into logic gates. It is a high-performance, sophisticated logic synthesis engine that delivers fast and efficient FPGA designs. It compiles the HDL input, removes all redundant logic, and combines all common expressions to create a technology independent optimized netlist. Based on the timing constraints specified by the user, the compiled design is then optimized for a specified technology using functional blocks. Depending on the design priority, the design can be optimized for area in which the number of functional blocks is minimized. The design can also be optimized for speed in which the number of levels of logic in critical paths is reduced to maximize the speed the clock can be run. The optimized netlist is written out

in EDIF format (*.edif). Figure 11 depicts the FPGA synthesis flow using Synplicity Synplify Pro and Table 2 lists the required input files and the generated output files of the synthesis process.

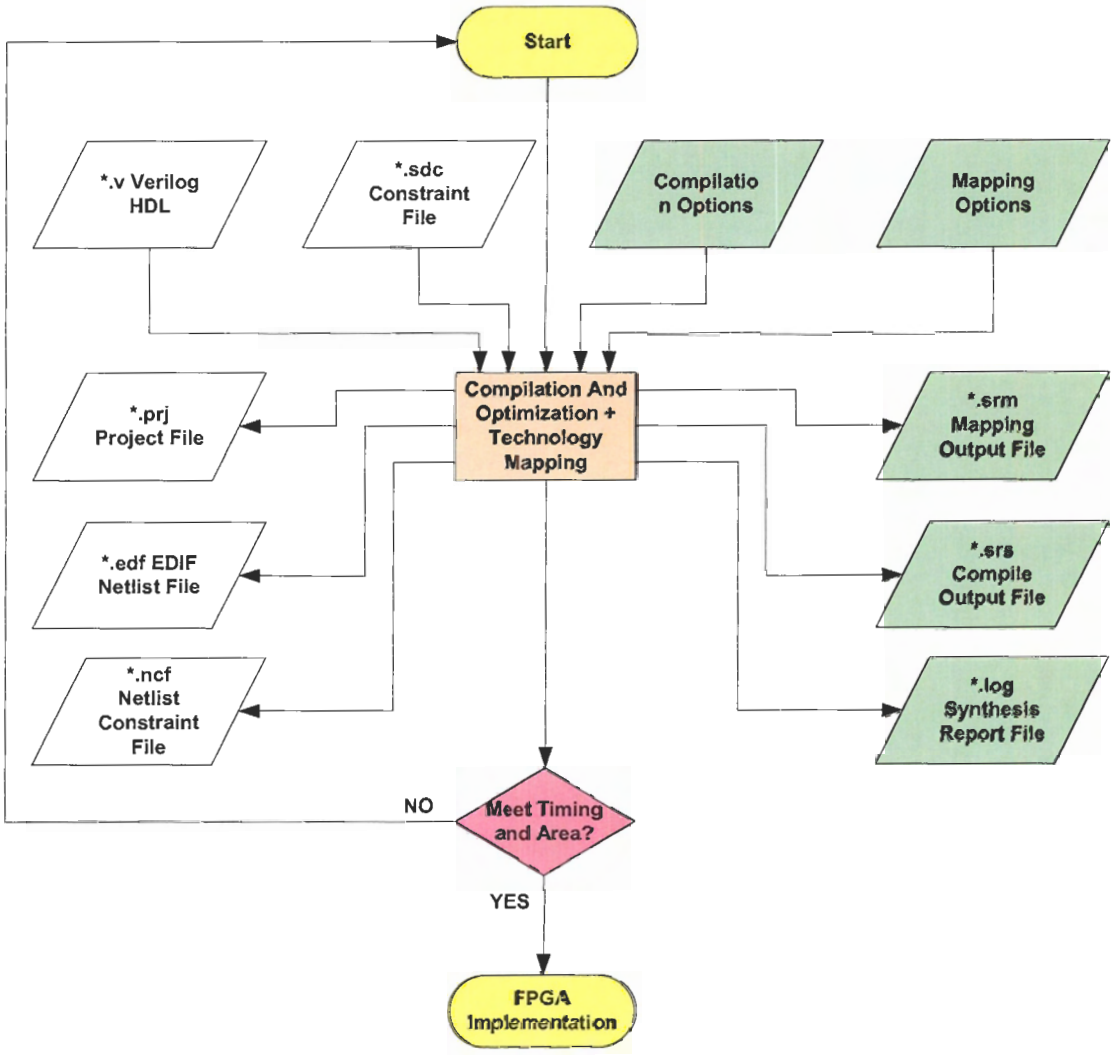


**Figure 11 FPGA Synthesis Flow**

**Table 2 Input and Output Files of FPGA Synthesis Process**

| | |
|---|---|
| Input Files | *.v<br>Verilog HDL Source Files |
| | *.sdc<br>Timing constraint file on clock, input/output delays, timing exception |
| Output Files | *.prj<br>Project file that lists all the commands executed in the synthesis process |
| | *.edf<br>EDIF (Electronic Design Interchange Format) synthesized netlist |
| | *.ncf<br>Netlist Constraint File that is forward annotated to Xilinx Place and Route tool |
| | *.srm<br>Mapping output files generated after mapping to represent the actual technology specific mapped design. It is used in the Technology View by Synplify Pro GUI |
| | *.srs<br>Compile output files generated after compilation to represent the RTL-level of the design. It is used in the RTL View by Synplify Pro GUI |
| | *.log<br>Synthesis log file for synthesis run, timing and area reports |

### 6.2.1 Design Constraints

Synplify Pro supports user-defined design constraints that can be applied to clocks, registers, inputs, and outputs. In addition, it supports attributes which are instructions placed on symbols or nets to show their placement, implementation, and other characteristics. It is recommended to set the design constraints to match the design goal with about 10% margin. Design constraints and attributes are defined in a Synplify constraint file (*.sdc). Using user-specified optimization constraints, annotated constraints can be forwarded to Place and Route tool using a Synplify created constraint file (*.ncf).

#### 6.2.1.1 Clock Constraints

Clock constraints specify a frequency goal for a clock domain. All clocks in the same clock group are synchronous and so "real" paths exist between those clocks. Paths between clocks in different clock groups are "false". Only paths between clock domains in the same clock group are analyzed and optimized. Although the mapper in the Synplify Pro tool can infer clocks by tracing the clock inputs of the registers, it is recommended to use the define_clock timing constraint to declare the clocks.

#### 6.2.1.2 Input/Output Delay Constraints

Input and Output delays outside the device are specified with define_input_delay and define_output_delay. Input delay refers to the delay before the signal arrives at the input pin and output delay refers to the delay of the logic outside the FPGA that is driven by the design outputs. Input/Output delays result in more accurate timing estimation in the Synplify Pro tool as the timing analyzer considers these delays during optimization.

#### 6.2.1.3 Multicycle Path Constraints

The define_multicycle_path timing constraint can be used to specify multicycle paths. These paths are still analyzed and will be reported as timing violation if they cannot meet timing.

#### 6.2.1.4 False Path Constraints

The define_false_path timing constraint can be used to specify paths that are not logically valid and timing of those paths are not analyzed.

#### 6.2.1.5 Fan-out Constraints

Synplify Pro uses the fan-out constraint to optimize designs better. Depending on the value, Synplify Pro can perform replication and buffering. Properly setting this constraint can significantly improve the design performance and routability. In some cases where the global constraint value is not good for specific nets in the design, the syn_maxfan attribute can be used to override the global value and control the fan-out limit for those nets.

#### 6.2.1.6 Pad Type

Xilinx provides a number of input and output pad types. The attribute xc_padtype is used to select the pad type to be used for a pin. Different pad types have different drive strength and slew value that can impact performance and board design.

### 6.2.2 Device and Technology Options

After defining the design constraints, the target technology and device options such as the part, package, speed grade, and fan-out limit need to be specified based on design goals. The Synplify Pro tool has unique mappers for each technology, and the mappers have knowledge of the technology specific architectural features. In addition, the compilation and mapping options such as IO insertion, retiming, pipelining need to be specified.

### 6.2.3 Synthesis Run

After the input source files, design constraints and attributes, and device options are specified, synthesis can be started. The area and speed results of the Synplify Pro run are saved into a log file (*.log). The log file contains an estimate of the operating frequency of the design and an estimated number of device specific resources used. It also reports a list of the critical paths in the design. When the difference between the estimated area or speed and the original design goal is more than 10%, constraints need to be adjusted to improve timing. In some cases, the HDL source code needs to be modified to meet the area or timing requirements.

The following output is copied from the synthesis log file which shows the result of synthesizing SXP 36x3G into the Xilinx FPGA. Although negative slacks are reported, the subsequent Xilinx Place and Route step could continue because the estimated delays of the violations are within 10% of the design goal.

**Table 3 Synthesis Performance Summary**

| Starting Clock | Requested Frequency | Estimated Frequency | Requested Period | Estimated Period | Slack | Clock Type | Clock Group |
|---|---|---|---|---|---|---|---|
| clk0 | 150.0 MHz | 155.4 MHz | 6.667 | 6.436 | 0.230 | declared | default_clk group_3 |
| hclk | 15.0 MHz | 78.0 MHz | 66.667 | 12.823 | 53.843 | declared | default_clk group_4 |
| pm8387_ core_inst. twiss_inst. twi_mpif_ rdb[0] | 5.0 MHz | 287.7 MHz | 200.000 | 3.476 | 196.524 | declared | default_clk group_6 |

| Starting Clock | Requested Frequency | Estimated Frequency | Requested Period | Estimated Period | Slack | Clock Type | Clock Group |
|---|---|---|---|---|---|---|---|
| pm8387_ core_inst. twiss_inst. twi_mpif_ wrb[0] | 5.0 MHz | 50.2 MHz | 200.000 | 19.923 | 180.077 | declared | default_clk group_5 |
| refclk_37mhz | 37.5 MHz | 39.9 MHz | 26.667 | 25.080 | 1.586 | declared | default_clk group_8 |
| refclk_75mhz | 75.0 MHz | 89.9 MHz | 13.333 | 11.120 | 2.213 | declared | default_clk group_9 |
| rxclk_h1_i | 150.0 MHz | 138.1 MHz | 6.667 | 7.241 | -0.574 | declared | default_clk group_2 |
| rxclk_h2_i | 150.0 MHz | 134.4 MHz | 6.667 | 7.440 | -0.773 | declared | default_clk group_0 |
| rxclk_t1_i | 150.0 MHz | 138.1 MHz | 6.667 | 7.241 | -0.574 | declared | default_clk group_1 |
| rxclk_t2_i | 150.0 MHz | 138.1 MHz | 6.667 | 7.241 | -0.574 | declared | default_clk group_7 |
| sxp36x3g_ top_refclk | 1.0 MHz | 472.2 MHz | 1000.000 | 2.118 | 997.882 | inferred | Inferred_clk group_15 |
| System | 1.0 MHz | 80.6 MHz | 1000.000 | 12.401 | 987.599 | system | default_clk group |

Resource Usage Report for sxp36x3g_top

Mapping to part: xc2v8000ff1152-5
Cell usage:

| | |
|---|---|
| DCM | 5 uses |
| FD | 9 uses |
| FDC | 12792 uses |
| FDCE | 18015 uses |
| FDC_1 | 99 uses |
| FDP | 987 uses |
| FDPE | 1621 uses |
| GND | 164 uses |
| LD | 1376 uses |
| MULT_AND | 64 uses |
| MUXCY | 176 uses |
| MUXCY_L | 4284 uses |
| MUXF5 | 5635 uses |
| MUXF6 | 576 uses |
| MUXF7 | 32 uses |
| VCC | 117 uses |
| XORCY | 2767 uses |
| XORCY_L | 20 uses |
| spr32x89 | 1 use |
| tpr_150x33 | 4 uses |
| tpr_16x34 | 4 uses |
| tpr_20x39 | 4 uses |
| trf10x129 | 1 use |
| trf_264x32 | 2 uses |

I/O primitives: 247

| | |
|---|---|
| IBUF | 105 uses |
| IBUFG | 6 uses |
| IOBUF | 6 uses |
| OBUF | 77 uses |
| OBUFT | 13 uses |
| OBUF_F_24 | 40 uses |
| BUFG | 10 uses |

SRL primitives:

| | |
|---|---|
| SRL16 | 3 uses |

I/O Register bits: 146
Register bits not including I/Os: 33377 (35%)

Global Clock Buffers: 10 of 16 (62%)

Mapping Summary:
Total LUTs: 71198 (76%)

## 6.3 FPGA Design Implementation

FPGA Design Implementation is the process of translating, mapping, placing, routing, and generating a BIT file for the design. Xilinx ISE (v 6.2.03i) is used for design implementation.

### 6.3.1 NGDBuild

NGDBuild performs all the steps necessary to read a netlist file and creates an NGD file describing the logical design. NGDBuild invokes a netlist reader program called EDIF2NGD which imports the EDIF netlist (*.edf) generated during the synthesis process. The program also needs the NCF file (*.ncf) which contains timing and layout constraints that are forward annotated from Synplify Pro, the EDN netlist (*.edn) reference memory model files generated by CoreGen, and the UCF User Constraint File (*.ucf) that contains the user defined constraints such as timing and the pin location. The output of EDIF2NGD is a Xilinx proprietary NGO file (*.ngo) which is a binary file containing a logical description of the design in terms of its original hierarchy and components. NGDBuild then reduces the logical components into Xilinx primitives by merging components from other files and identifying appropriate system library components and behavioural models. It then performs a Logical Design Rule Check (DRC) on the converted design and outputs an NGD database file (*.ngd) and a BLD report file (*.bld) that records all the warning and error messages during the build process. The NGD databse file contains both a logical description of the design that is reduced to lower level Xilinx Native Generic Database (NGD) primitives and a

description of the original hierarchy defined in the input netlist. It can then be mapped to the desired device family.

### 6.3.2 MAP

Mapping is the second process in the FPGA implementation flow. In this process, the MAP command is used. MAP first selects the target Xilinx device, package, and speed and then performs a DRC check on the NGD file generated using the NGDBuild program to check the design integrity. After deleting all unused components and nets, it maps all basic logic elements into Xilinx components such as IO cells (IOBs) and Logic Block Cells (CLBs) in the target Xilinx FPGA. It then processes all location and timing constraints, performs target device optimizations, and runs a DRC check on the resulting mapped netlist. Finally it outputs a NGM file (*.ngm) which contains logical information and physical information about how the design was mapped, a PCF file (*.pcf) which contains constraint information specified during the design entry phase, an NCD file (*.ncd) which describes the design in terms of Xilinx components such as CLBs and IOBs, and a MRP file (*.mrp) which contains all the warning and error messages, details about how the design was mapped and statistic about component usage in the design.

The following is the output in the MRP file which shows how SXP 36x3G is mapped into the Xilinx FPGA:

| | | |
|---|---|---|
| Design Summary: | | |
| Number of errors: 0 | | |
| Number of warnings: 93 | | |
| Logic Utilization: | | |
| Total Number Slice Registers: | 34,601 out of 93,184 | 37% |
| Number used as Flip Flops: | 33,311 | |
| Number used as Latches: | 1,290 | |
| Number of 4 input LUTs: | 68,113 out of 93,184 | 73% |
| Logic Distribution: | | |
| Number of occupied Slices: | 40,079 out of 46,592 | 86% |
| Total Number 4 input LUTs: | 69,846 out of 93,184 | 74% |
| Number used as logic: | 68,113 | |
| Number used as a route-thru: | 1,730 | |
| Number used as Shift registers: | 3 | |
| Number of bonded IOBs: | 235 out of 824 | 28% |
| IOB Flip Flops: | 152 | |
| Number of Block RAMs: | 24 out of 168 | 14% |
| Number of GCLKs: | 10 out of 16 | 62% |
| Number of DCMs: | 5 out of 12 | 41% |
| Total equivalent gate count for design: | 2,331,552 | |
| Additional JTAG gate count for IOBs: | 11,280 | |
| Peak Memory Usage: | 1431 MB | |

### 6.3.3 Place and Route

Place and Route (PAR) is the third process in the Xilinx FPGA flow. It uses the output NCD file from the previous MAP process to place and route the design. During placement, PAR executes multiple phases of the placer and places components into sites based on factors such as constraints specified in the PCF file, the number of connections, and the available routing resources. After placement, PAR executes multiple phases of the router that looks for a converging solution based on the desired methodology that routes the design to completion and meets timing constraints. Once the design is fully routed, a placed and routed NCD file is generated to be used by the bitstream generation (BITGEN). An output PAR Report file is created to provide execution information that records the steps taken as the program converges on a placement and routing solution.

For the SXP 36x3G emulation project, timing-driven PAR is used so that placement and routing are executed according to timing constraints specified in the UCF file at the beginning of the design process. It is set-up to run repeatedly until timing is met.

The following is the device utilization summary copied from the PAR Report file:

Device utilization summary:

| | | |
|---|---|---|
| Number of External IOBs | 235 out of 824 | 28% |
| Number of LOCed External IOBs | 235 out of 235 | 100% |
| Number of RAMB16s | 24 out of 168 | 14% |
| Number of SLICEs | 40079 out of 46592 | 86% |
| Number of BUFGMUXs | 10 out of 16 | 62% |
| Number of DCMs | 5 out of 12 | 41% |

### 6.3.4 TRACE

TRACE is run after the PAR process to provide static timing analysis of the design based on the timing constraint. It verifies that the design meets timing constraint and generates a formatted ASCII file to report statistics on the design. It provides a listing of timing compliance between the routed design and the timing constraint input, and optionally the detailed net and path delay reports.

### 6.3.5 BitGen

BitGen is the Xilinx bitstream generation program. It takes the fully routed NCD file from the PAR process and creates a bit stream binary file (*.bit) for FPGA configuration. In addition to device-specific information from files associated with the target device, the bit stream file contains all of the configuration information that defines

the internal logic and the interconnections of the FPGA. It is downloaded onto the FPGA via various interfaces or it can be used to create a PROM file as described in the next section.

### 6.3.6 PROMGen

For the SXP 36x3G emulation project, the FPGA is not configured directly using the BIT file generated from the BitGen process. Instead, the BIT file is converted into a PROM format file (*.exo) by PROMGen and downloaded over the USB interface.

PROMGen formats a BitGen-generated configuration bitstream BIT file into a PROM format file. It contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix) [15]. It can also generate a binary or hexadecimal file format.

# 7 PROJECT CONTRIBUTIONS AND CHALLENGES

The FPGA emulation system described in this paper takes 18 man-months (3 people) to complete design, implementation, and testing. I was involved in the following tasks of the project:

- Modified the source HDL to map the real HDL into the FPGA.
- Synthesized and built the FPGA load.
- Modified the device testbench to run sanity simulations to verify the correctness of the FPGA RTL.
- Brought up the entire FPGA emulation platform.
- Performed inter-operability testing with different SAS and SATA drives and HBAs in different configurations.
- Performed point testing to verify new hardware blocks in the device.
- Supported firmware development.

An FPGA emulation platform requires significant effort to develop and is particularly true for the SAS Expander device, which is the first device to be emulated using FPGAs at PMC-Sierra.

Although the development flow was similar to a chip design flow, FPGA emulation used different software synthesis and place and route tools and it was

challenging when there was no standard procedure to follow within the company. Each iteration of FPGA synthesis and place and route took 6 hours to complete so a trial and error approach was not acceptable. Each time the design failed to meet timing or route, careful analysis has to be performed to minimize the number of iterations required. It was even more challenging at the early phase of the project where RTL was not stable but we wanted to provide a working FPGA emulation system for firmware and hardware co-development.

The emulation environment consisted of both software and hardware components. Instead of integrating and enabling all the components at once and hoping everything would magically work, the hardware blocks were brought up and tested one-by one. This task required an understanding of the whole system and systematic debugging skills and creativity were essential to the success of the project because visibility into the FPGA was limited.

The tests that we chose to perform with the FPGA system also needed to be carefully planned. Unlike simulation where we had many licenses, the FPGA emulation resource was limited so in order to fully justify the additional cost, we were required to have a carefully thought out test plan that was agreed upon by the design, firmware, and validation teams. By running firmware to perform numerous tests with real HBAs and SAS and SATA drives, we found about 20 hardware bugs. The major bug we found was about data words being swapped within a cache line, which would have required silicon re-spins. Overall, it was a challenging and rewarding experience for me.

# 8 CONCLUSIONS AND RECOMMENDATIONS

For the PM8387 SXP 36x3G device, the primary goal of FPGA emulation is to provide a platform for more robust and thorough pre-silicon verification that will result in a higher quality tape-out. It also provides a platform for pre-silicon validation and concurrent hardware and firmware development.

In order to have successful FPGA emulation, it should be planned and integrated into the design and development cycle as early as possible. Code changes specific for FPGA emulation should be minimized. Replacement strategy for technology specific implementations should be planned and clock and reset distribution should be well thought out.

Also, FPGA emulation is very effective as part of the verification strategy. Although FPGA emulation does not have the same flexibility and high-level controllability as simulation, it is over 50,000 times faster than simulation and is perfect for passing a lot of traffic and catching the corner cases that otherwise will not be possible with simulation [8]. Good coordination between simulation and emulation reduces redundant effort spent in the same area and provides more robust verification of the pre-tapeout design. The FPGA emulation of SAS 36x3G was very successful with all the major functions performing as planned.

One additional thing that FPGA emulation should include is formal verification. Formal verification between the RTL source code and the final ASIC netlist has become an integral part of the ASIC design process to ensure the correctness of the design after synthesis, optimization, placement, and routing. However, the current process does not formally verify the relationship between the HDL source code and the final FPGA design. Although the possibility of having functionally different final FPGA design is small, it is a good engineering practice to perform formal verification on the FPGA design even though the design is not targeted for production.

Another limitation of the current FPGA emulation is that even though it is much faster than simulation, it is still relatively slow compared to the actual chip. This causes data synchronization problems when the FPGA prototype is being tested in a real system. Problems like this prevent some features from being tested in the FPGA emulation platform. One solution is to have a slow clock mode in the device such that rate adaptation FIFOs are used to prevent data overrun and underrun problems and allow the whole system to be slowed down. Although this will require the chip architects to take FPGA emulation into account at the beginning of a project, the return on investment for FPGA emulation will increase significantly with a more comprehensive platform being provided.

# REFERENCE LIST


[1] Adaptec, Inc., Maximizing Server Storage Performance with PCI ExpressTM and Serial Attached SCSI, November, 2003, http://www.adaptec.com/worldwide/product/markeditorial.html?sess=no&language=English+US&prodkey=infostar_sas_article&type=White%20Papers

[2] Adaptec, Inc., Serial Attached SCSI: Meeting the Growing Needs of Enterprise Storage, 2004, http://www.adaptec.co.uk/worldwide/product/markeditorial.html?prodkey=sas_storage_uk

[3] American National Standard, Information Technology – Serial Attached SCSI – 1.1 (SAS-1.1). Project T10/1601-D, Revision 7, November 19, 2004

[4] Cadence, Incisive Enterprise Palladium Series with Incisive XE Software Datasheet, October, 2005, http://www.cadence.com/datasheets/incisive_enterprise_palladium.pdf

[5] Maxtor, Serial Attached SCSI Architecture: Part 1—Introduction, the SAS Physical Layer and the SAS Phy Layer, December 2003

[6] Mentor Graphics, ModelSim's Industry-Leading VHDL Simulator Enables Marconi to Develop 15-CPU, 272 ASIC, Distributed System Simulation Environment, 2001, http://www.model.com/news_events/pdf/gec_success.pdf

[7] MIPS Technologies, Sead-2 TM Board User's Manual, Revision 01.02, March 31, 2004, http://www.mips.com/content/Documentation/MIPSDocumentation/DevelopmentBoards/SEAD-2Boards/MD00064-2B-SEAD2-USM-01.02.pdf?agree=yes

[8] Mitch Dale, The Value of Hardware Emulation, 2002

[9] PMC-Sierra Inc., PM8387 SXP 36x3G 36-Port SAS Expander Short Form Data Sheet, v3, May 25, 2005, http://www.pmc-sierra.com/products/details/pm8387/

[10] PMC-Sierra Inc., PM8354 & PM8354A QuadPHY1G Data Sheet, Issue 5, May 2004, http://www.pmc-sierra.com/products/details/pm8354/

[11] PMC-Sierra Inc., PM8380 QuadSMX 3G Data Sheet, Issue 3, 2004, http://www.pmc-sierra.com/products/details/pm8350/

[12] Robert C. Elliott, Serial Attached SCSI, Revision 5, July 9, 2003

[13] SimPOD, Enhanced FPGA Prototyping, 2003

[14] Synplicity, Synplicity-Xilinx High-Density Methodology, February 2000, http://www.synplicity.com/literature/pdf/high_dense.pdf

[15] Xilinx, Development System Reference Guide, 2005, http://toolbox.xilinx.com/docsan/xilinx4/data/docs/dev/dev.html

[16] Xilinx, Virtex-II Platform FPGA User Guide, UG002 (v2.0), March 23, 2005, http://www.xilinx.com/bvdocs/userguides/ug002.pdf

[17] Xilinx, Virtex-II Platform FPGAs: Complete Data Sheet, DS031 (v3.4), October 14, 2003, http://www.xilinx.com/bvdocs/publications/ds031.pdf