Design and Implementation of Multi-Channel NVM Express Controller

A Project Report

submitted by

KEERTHI KIRAN H PUJAR

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2014

THESIS CERTIFICATE

This is to certify that the thesis entitled **Design and Implementation of Multi-Channel NVM Express Controller**, submitted by **KEERTHI KIRAN H PUJAR**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

The successful completion of this project work would not have been possible without the guidance, support and encouragement of many people.

First of all I would like to express deepest sense of gratitude to my guide, **Dr. V. Kamakoti** whose energy, enthusiasm and constant support inspired me from the beginning till the end of my project.

I would also like to express my sincere gratitude and thanks to **Mr. G.S. Madhusudan** who has provided me with lots of ideas and suggestions throughout the project. I am very much thankful for the valuable inputs that he has provided to the project, through the Industry experts.

I would like to thank my faculty advisor **Dr. Deleep R.Nair** for his support and guidance throughout our course. I express my sincere gratitude to my fellow project mates Senthil, Sarath, Chaitanya, Sachin, Chidambaranathan and Neel who have been very supportive.

I would like to thank my classmates who made my life in IITM a memorable experience.

Finally, I take this opportunity to thank my parents for their support and encouragement without which learning in and becoming a part of such a prestigious institution would not have been possible.

ABSTRACT

KEYWORDS: NAND Flash, NVM Express, multi-channel, I/O Parallelism

With more data-intensive applications appearing in the present social life, the NAND flash memory acting as a replacement candidate of hard disk drives is popularly used in some data centres due to its lower power consumption, faster random access, and higher shock resistance. But the traditional solid state disk exposes the limitation of bandwidth. To this end, the scalable Multi- Channel NVMe controller architecture is used to exploit the parallelism of multiple chips. The proposed system consists of multiple independent channels, where each channel has multiple NAND flash memory chips. On this hardware, we have three optimization techniques to exploit I/O parallelism: Striping, Interleaving, and Pipelining. The present project deals with the design and implementation of Multi-Channel NVMe controller.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

ABSTRACT **ii**

LIST OF FIGURES **vi**

ABBREVIATIONS **vii**

1 Introduction **1**

1.1 Overview of Processor Architecture 1

1.2 Overview of contents 3

2 Background **4**

2.1 Flash Memory 4

2.2 NAND Flash vs NOR Flash 4

2.3 NAND Flash Memory Organization 5

2.4 Motivation for Multi-Channel NVMe Controller 5

2.5 NVMe Controller 6

2.5.1 Need for NVM Express 7

2.5.2 Key Attributes of NVM Express Controller 8

2.5.3 NVMe Command Set 8

2.5.4 NVMe Functional Description 8

2.5.5 Controller Interfaces 9

2.6 NAND Flash Controller 11

2.6.1 NFC Functional Description 11

2.7 Non Volatile Memory Sub-system 12

2.8 BlueSpec System Verilog Language Background 13

3 Design and Implementation of Multi-Channel NVMe Controller 15

3.1 Multi-Channel NVMe Controller Architecture 15

3.1.1 The Switching Fabric 16

3.1.2 Inbound Feeder 17

3.1.3 Scheduler 17

3.2 Modules Description 18

3.2.1 Command Dispatch 18

3.2.2 Command Execution 20

3.2.3 Command Completion 22

3.2.4 Command Arbitration 23

3.2.5 NAND Control State Machine 23

4 Exploiting I/O Parallelism 26

4.1 Striping 27

4.2 Interleaving 28

4.3 Pipelining 29

4.4 Implementing Striping, Interleaving and Pipelining 30

4.5 Design Challenges 31

4.6 Verification 31

4.6.1 Verification Setup 32

4.6.2 Verification Test Cases 32

4.7 Simulation Results 33

4.8 Synthesis Report 34

5 Conclusions and Future Work 36

A NVM Subsystem with FTL Processor 37

LIST OF FIGURES

1.1 Overview of Processor Architecture 1

2.1 Target Memory Organization 5

2.2 NVMe Command Set 9

2.3 NVMe Modules and Interfaces 10

2.4 NAND Flash Controller Block Diagram 12

2.5 NVM Subsystem 13

3.1 Multi-Channel NVMe Controller Architecture 16

3.2 Multi-Channel NVMe Modules and Interfaces 17

3.3 State Machine describing the working of Scheduler 18

3.4 Multi-Channel NVMe Modules and State Machines 19

3.5 Command Execute State Machine-without Erase 20

3.6 Command Execute State Machine-without Erase 21

3.7 Nand Control State Machine 23

4.1 Three optimization techniques to exploit I/O parallelism. (a) Striping technique; (b) interleaving technique; and (c) pipelining technique. 26

4.2 The striping technique with the striping level of two. (a) Read operation and (b) write operation. 27

4.3 The interleaving technique with the interleaving level of two. (a) Read operation and (b) write operation. 29

4.4 The Pipelining technique. (a) Read operation and (b) write operation. . 30

4.5 Verification Setup 32

4.6 Creation Of Completion Queue 33

4.7 Creation Of Submission Queue 33

4.8 Initiate Write 34

4.9 Initiate Read 34

4.10 Read Write Command Execute Parallely 35

A.1 Future Work on NVM Subsystem 39

ABBREVIATIONS

NVM	Non Volatile Memory
NVMe	Non Volatile Memory Express
PCIe	Peripheral Component Interconnect Express
SATA	Serial Advanced Technology Attachment
AHCI	Advanced Host Controller Interface
SQ	Submission Queue
CQ	Completion Queue
ASQ	Admin Submission Queue
ISQ	I/O Submission Queue
ACQ	Admin Completion Queue
ICQ	I/O Completion Queue
MSI	Message Signalled Interrupt
ONFI	Open Nand Flash Interface
TLP	Transaction Level Packet
BAR	Base Address Register
SSD	Solid State Drives
MMIO	Memory Mapped Input Output
HBA	Host Bus Adaptor
FTL	Flash Translation Leyer

CHAPTER 1

Introduction

1.1 Overview of Processor Architecture

The overview of the processor architecture is shown in the figure 1.1

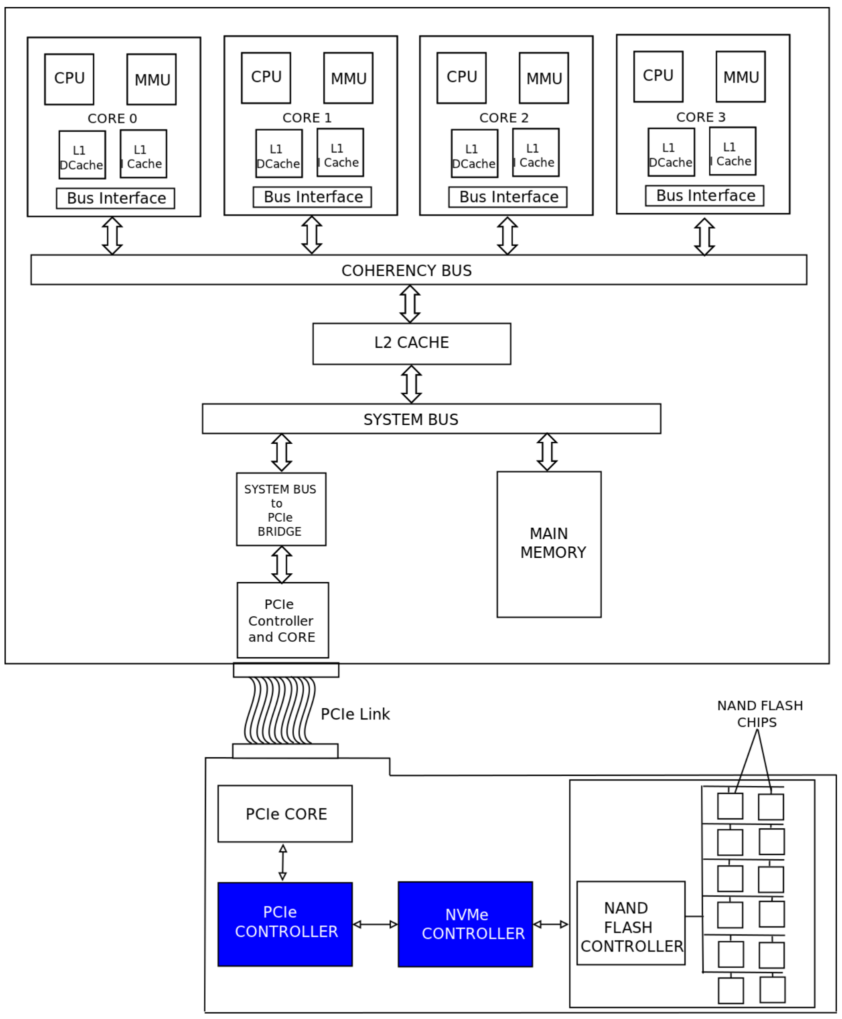


Figure 1.1: Overview of Processor Architecture

Overview of Processor

The processor design team in the Reconfigurable and Intelligent Systems Engineering (RISE) lab has been actively involved in the design of a high-end in-house processor for defence and security related applications. The figure 1.1 shows an overall view of the targeted processor. The processor system has a quad-core architecture, with a support for two levels of cache hierarchy and a single on-chip DRAM. It implements cache coherency at L1 cache level, with coherency bus. The CPU core is based on 64 bit PowerPC Instruction Set Architecture. It supports "dual-issue" and "out of order execution". A Memory Management Unit was designed for an efficient data transfer between the Processor Core and the Main Memory (DRAM). It supports NVM Express based I/O Subsystem with a PCI Express interface. This I/O subsystem typically implements the "File System" for the processor. The NVMe controller supports multiple channels, where each channel can have multiple NAND flash memory chips. This multichannel architecture is used to exploit the parallelism offered by chips.

My Contribution

Previously, NVMe contoller Nand Flash Controller were designed. In order to store large amounts of data and to overcome the bandwidth bottleneck of traditional solid state drives, Multi-Channel NVMe controller has been designed. The highlighted portion in the figure 1.1 shows my contribution to the NVMe subsystem. Also apart from Multi-Channel architecture, a logic for erasing the NAND Flash blocks is implemented. This is because Flash devices are divided into erase units, also called blocks. Writing information to a specific block, in any flash device, can only be performed if that block is empty/erased. In most cases, this means that an erase operation must precede a write operation.

1.2 Overview of contents

Chapter 2 explains the background and motivation for designing Multi-Channel NVM Express based I/O subsystem, with a PCIe interface to it. It explains the overall architecture of NVM subsystem and NAND Flash Controller and concludes with a brief description of the language used to code the design.

Chapter 3 describes the design and implementation of Multi-Channel NVMe controller.

Chapter 4explains the techniques to exploit I/O Parallelism and their implementation.

Chapter 5 concludes with a short description of the future prospects of the designed subsystem.

Appendix A provides little more insight into the proposed modifications to the NVM Subsystem, with the inclusion of FTL unit.

CHAPTER 2

Background

2.1 Flash Memory

Flash memory is widely used for code and data storage of consumer electronics products due to its versatile features such as non-volatility, solid-state reliability, low power consumption, and shock resistance. Flash memory based SSDs(Solid State Drives) have 100x greater throughput and instantaneous access times for quicker boot ups, faster file transfers, and overall snappier performance than hard drives. The most popular flash memory types are NOR and NAND.

2.2 NAND Flash vs NOR Flash

The real benefits of NAND Flash are faster PROGRAM and ERASE times, as NAND Flash delivers sustained WRITE performance exceeding 7 MB/s. Block erase times are an impressive 500s for NAND Flash compared with 1 second for NOR Flash. The hardware pin requirements for NAND Flash and NOR Flash interfaces differ markedly. NOR Flash requires approximately 44 I/O pins for a 16-bit device, while NAND Flash requires only 24 pins for a comparable interface. Although NOR flash memory offers random access capability and high read performance, it suffers from extremely low write and erase performance and it is more expensive per MB than NAND flash memory. Clearly, NAND Flash offers several compelling advantages. But NAND Flash has its own drawbacks limiting the maximum throughput obtainable from it. Hence we go for Multi-Channel NVM Express controller.

2.3 NAND Flash Memory Organization

A NAND Flash device contains one or more targets. A target is controlled by one CE (Chip Enable) signal. A target is organized into one or more logical units (LUNs). LUN or Die is the minimum unit that can independently execute commands and report status. A set of LUNs that share one CE(Chip Enable) signal within one NAND Flash device form a NAND Target. The Flash array contains a number of blocks. A block is the smallest erasable unit of data within the Flash array of a LUN. There is no restriction on the number of blocks within the LUN. A block contains a number of pages. A page is the smallest addressable unit for read and program operations. A page consists of a number of bytes or words.The Target Memory Organization is shown in the figure **??**.

includegraphics[width=11 cm,height=12 cm]./images/image$_pngs/Target_Memory_Organization.png$

Figure 2.1: Target Memory Organization

2.4 Motivation for Multi-Channel NVMe Controller

Although replacing hard disks with NAND flash memory brings advantages in terms of size, weight, reliability, and energy use, it is not easy to draw the maximum performance from NAND flash memory due to its unique operational characteristics. In NAND flash memory, the write operation requires a relatively long latency compared to the read operation. In addition, the previous data should be erased first in order to write another data in the same physical area. The worse problem is that the erase operation cannot be performed on the particular data selectively, but on the larger unit containing the original data with much longer latency. The MLC (Multi-Level Cell) technology, which is recently introduced to multiply the capacity of a NAND flash memory chip, further decreases the operation speed. Thus, developing a high-performance NAND flash-based storage system remains a technically challenging area. In order to alleviate the computer system I/O bottlenecks, we need high-bandwidth flash array architectures. In this project, architecture for high-performance NAND flash-based storage system is designed. The hardware architecture of the designed system consists of multiple independent channels, where each channel can have multiple NAND flash memory chips. Further, three optimization techniques that can exploit I/O parallelism in various ways: Striping, Interleaving, and Pipelining are implemented. The other part of this project is to facilitate erase operation as a prerequisite to write operation. Flash memory cannot perform in-place update of data, so an erase operation must be executed before a program/write operation.

2.5 NVMe Controller

NVM Express (NVMe) is a register level interface that allows host software to communicate with a non-volatile memory subsystem. This interface is optimized for Enterprise and Client solid state drives, typically attached to the PCI Express interface. NVM Express is a scalable host controller interface designed to address the needs of Enterprise and Client systems that utilize PCI Express based solid state drives. The interface provides optimized command submission and completion paths. It includes support for parallel operation by supporting up to 64K I/O Queues with up to 64K commands per I/O Queue.

2.5.1 Need for NVM Express

NVM Express is architected from the ground up for Non-Volatile Memory (NVM). NVM Express significantly improves both random and sequential performance by reducing latency, enabling high levels of parallelism, and streamlining the command set while providing support for security, end-to-end data protection, and other Client and Enterprise features users need.

Standardized Interface

- A standardized interface is need for the easy adoption of PCIe based SSDs. NVM express specification provides this standard interface for PCIe SSDs.

Scalable

- NVM Express is scalable host controller interface standard, which is designed for Enterprise and Client systems that use PCI express SSDs.

Efficient Command Set

- It provides a simple, streamlined and efficient command set which eliminates the legacy HDD to SSD command conversion overhead.

Optimized Register Interface

- It provides an optimized register interface, that allows the host software to communicate with the non volatile memory subsystem.

Industry Support

- It was developed by industry consortium of 80+ members and hence enjoys a very wide industry support.

2.5.2 Key Attributes of NVM Express Controller

1. Support for up to 64K I/O queues, with each queue supporting up to 64K commands.
2. Simplified command decoding and processing with fixed size(64B) command format.
3. All information to complete a 4KB read request is included in the 64B command itself, ensuring efficient small I/O operation.
4. Support for 2k MSI-X interrupts or 32 multiple message MSI.
5. Support for simple and efficient Interrupt Aggregation.

2.5.3 NVMe Command Set

The NVMe controller defines two sets of commands namely Admin Command Set and the NVM Command Set. The Admin Command Set takes care of all the administrative tasks related to the controller. This includes setting up the controller, creating or deleting Queues, aborting commands etc. The NVM Command set is responsible for the actual data transfer between the Host and the Nand Flash Device. The list of Admin Commands and NVM commands supported by the designed controller is shown in the figure 2.2

2.5.4 NVMe Functional Description

This chapter explains how the NVM express controller is implemented in hardware. The controller is designed according to the NVMe Spec 1.0c [6]. Controller interface signals and their brief description is provided along with the detailed description of all the logical modules involved in the design. The chapter also provides various test cases that are used to verify the functionality of the design. Synthesis results for the controller are provided

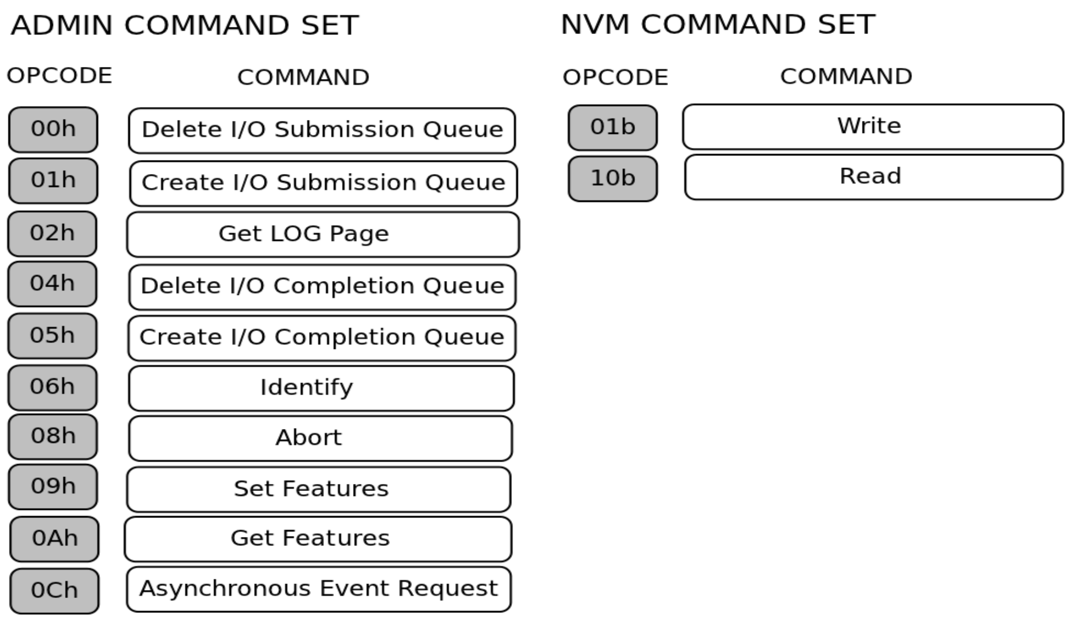


Figure 2.2: NVMe Command Set

at the end. The controller interfaces and logical module partitions are shown as a block diagram in the figure 2.3

2.5.5 Controller Interfaces

The controller has five interfaces to connect PCIe controller on one side and NAND Flash Controller on the other. The description of the interfaces and the signals involved in each interface are given in the following sub-sections.

Completion Data Interface

- This interface is used to connect the controller to the PCIe controller. It provides interface signals necessary to receive the incoming data from the PCI express.

Interrupt Interface

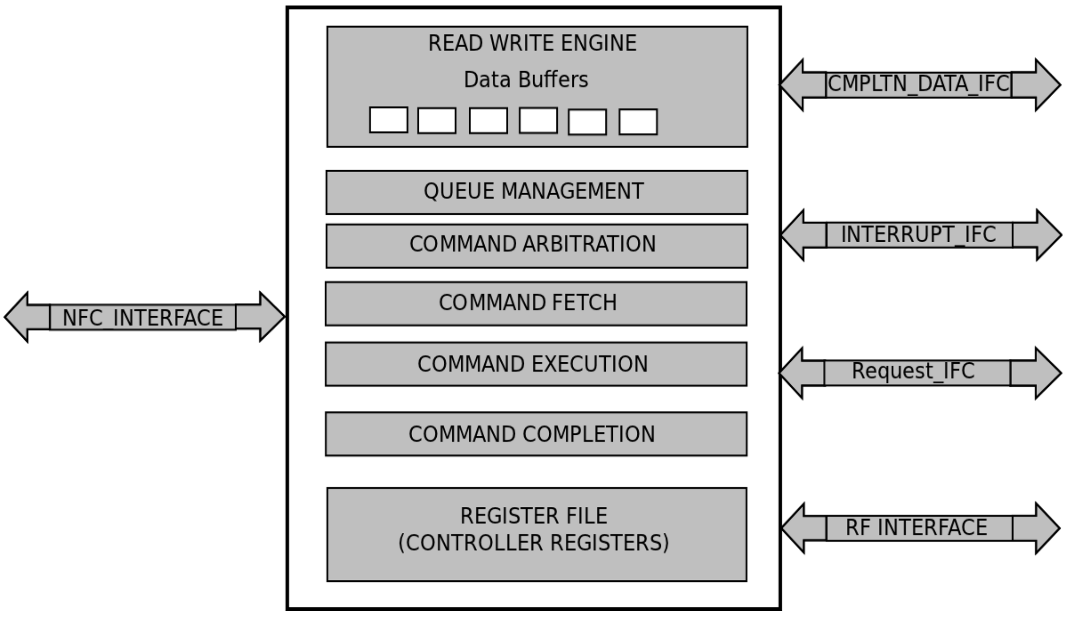


Figure 2.3: NVMe Modules and Interfaces

- This interface provides signals necessary to send interrupt vector information to the PCIe controller. This vector information is eventually sent as multiple message MSI over the PCIe channel, by the PCIe controller.

Request Interface

- This interface is also a part of the connection between the NVMe controller and PCIe controller. It provides interface signals necessary to Request the PCIe controller to transmit Read, Write or Completion TLPs over the PCIe channel. This interface also helps the NVMe controller to transmit the data, as a payload to Write and Completion TLPs.

Register File Interface

- This interface provides a Read Write access to the Controller Registers. PCIe controller can write into registers through this interface. However it can only request to read through this interface. The read data from the register file is sent as completions through the NVMe Request interface.

NAND Flash Controller Interface

- This interface is used to connect the controller to the Nand Flash Controller. It provides necessary signals to access the memory mapped register file of the Nand Flash Controller. It has a 32-bit data bus to transfer the data to or from Nand Flash Controller buffer. Nand Flash Chips in general support either a 16 bit bus or a 8 bit bus. However, the data bus to NFC is made 32 bits in order to transfer the data at a much faster rate and hence the controller will ready to process other commands.

2.6 NAND Flash Controller

NAND Flash Controller (NFC) provides an interface for user to communicate with NAND Flash devices. The controller accepts NAND Flash Memory commands from the user interface and generates different cycles on memory interface according to the NAND Flash Memory protocol.

The beginning of NAND Flash was very trying for manufacturers of controllers and host device solutions. Interface timing, page ID location, and format typically were vendor dependent. This made it very difficult for configuration and booting. Also, command sequences were specific to the device, making development of state machines more difficult. Many solutions at the time were software-based due to the changes among vendors. Standardization was clearly needed, and the first industry-accepted standard was ONFi 1.0.

The Open NAND Flash Interface (ONFi) is an industry Workgroup made up of more than 100 companies that build, design-in, or enable NAND Flash memory. They are dedicated to simplifying NAND Flash integration into consumer electronic products, computing platforms, and any other application that requires solid state mass storage.

2.6.1 NFC Functional Description

NAND Flash Controller has two interfaces. The Target interface is the ONFi interface. The Host interface is designed to be compliant with NVM express as well as any host processor. Figure 2.4 shows the interface signals coming from/going to the Host via Host interface and coming from/going to the NAND flash device via ONFi interface.

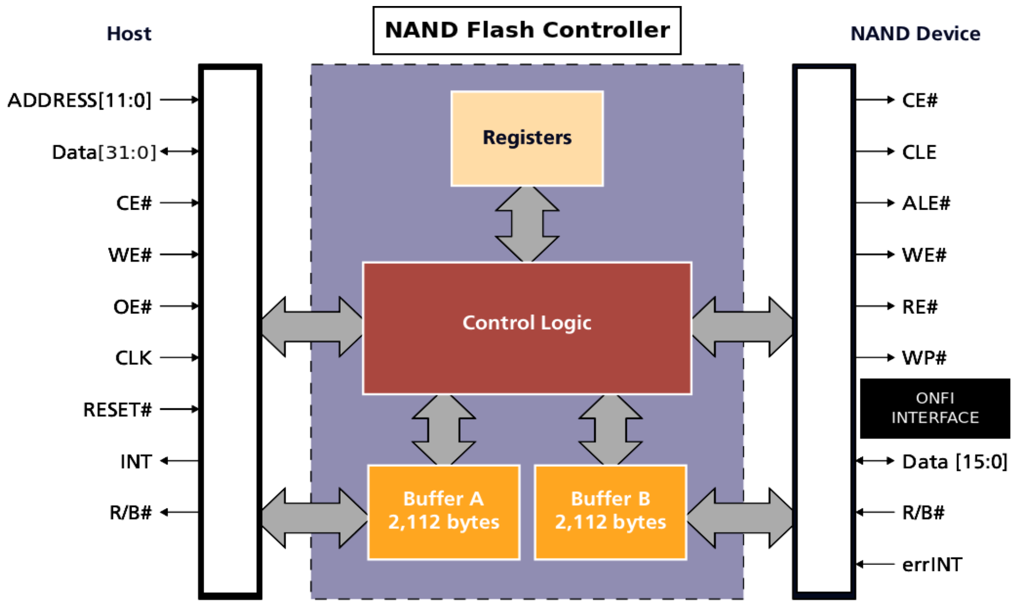


Figure 2.4: NAND Flash Controller Block Diagram

NVM express sends out 32-bit data and hence, the data bus on the Host side is 32-bit wide while NAND Flash supports either 8 bit or 16-bit wide data bus. So, the data bus on the Target side can be either 8-bit or 16-bit wide depending on the device. The host sends out commands on the data bus and addresses on the address bus. According to the command, the NAND Flash Controller takes certain action like read to or write from a certain location on the NAND flash device specified by the address from the host. The data management in the controller is managed by the two buffers.

2.7 Non Volatile Memory Sub-system

The present developed Multi-Channel Non Volatile Memory Sub-system consists of a PCIe Core , PCIe controller, NVMe Controller and Nand Flash Controller to access the NAND Flash Chips. The Sub-system is shown in the figure 2.5

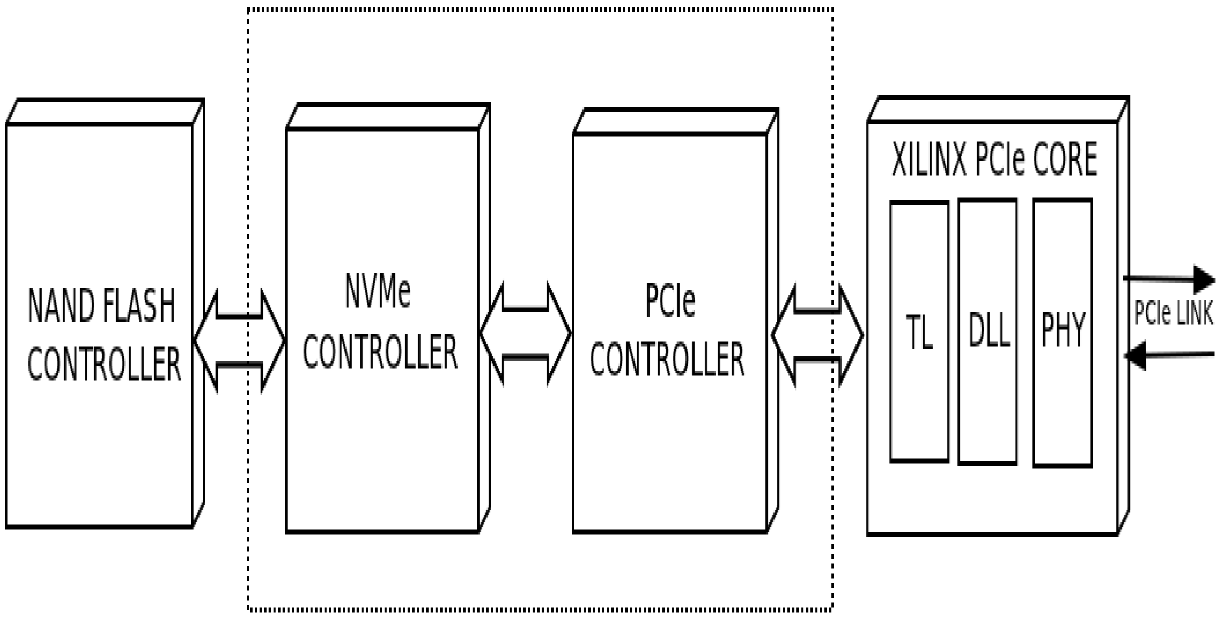


Figure 2.5: NVM Subsystem

2.8 BlueSpec System Verilog Language Background

BlueSpec System Verilog is a Hardware centric language based on industry standard SystemVerilog and Verilog [1]. It uses ***Rules*** and ***interface methods*** for behavioral description, which adds a powerful way to express complex concurrency and control :

- Across multiple shared resources.
- Across module boundaries.

Parallelization

- Concurrent behavior is expressed implicitly.
- It has a traditional hardware semantic model of cooperating FSMs.
- Rules express concurrent operations by simply describing the conditions under which the state element(s) are updated.
- Rules are implemented as unsequenced atomic transactions.
- Compiler introduces the scheduling and the muxing for the shared resources.

Level of Abstraction

- BSV provides significantly higher level of abstraction than Verilog, SystemVerilog, VHDL and SystemC.
- It provides Behavioral description through : Rules and Interface Methods.
- Various attributes for Structural description are:
  - High level abstraction types.
  - Powerful Static checking.
  - Powerful parametrization.
  - Powerful Static elaboration.

Standalone Function Libraries

BSV has a large set of function libraries, some of them are provided below.

- It has Data Containers such as FIFO, registers, BRAMs etc ..
- Circuits such LFSR and completion buffer [2].
- Interface types, GET, PUT transactors.
- Multiple Clock domain circuits such as synchronizers.
- Bus interfaces such as common data bus , Z-bus etc ..

CHAPTER 3

Design and Implementation of Multi-Channel NVMe Controller

3.1 Multi-Channel NVMe Controller Architecture

Figure 3.1 shows the architecture of the scalable multi-channel NVM Express controller. As shown in the Fig. 1, although we use eight flash chips, but all of them are addressed uniformly. That is, the eight chips are considered as a whole one called super-chip in users view. The host/PCIe interface makes read and write requests specified in terms of logical sectors to NVMe controller, which translates host requests into flash requests. And then the NVMe controller issues the commands, addresses and data to one of the channels. Here a channel is a link between the NVMe controller and a NAND Flash Controller(NFC).

The controller interfaces and logical module paritions are shown as a block diagram in the figure 3.2.When receiving the requests, the Scheduler sends the requests to a single NFC decided by the address of the request. A logical sector address is mapped to a particular NFC. A quick lookup tells us to which NFC the request has to be dispatched. Then according to the type of the request, the single NFC sends control signals to NAND flash chip to complete the operation like read page, program page, block erase etc. In the following subsections, we detail the key features of the multi-channel NVMe controller.

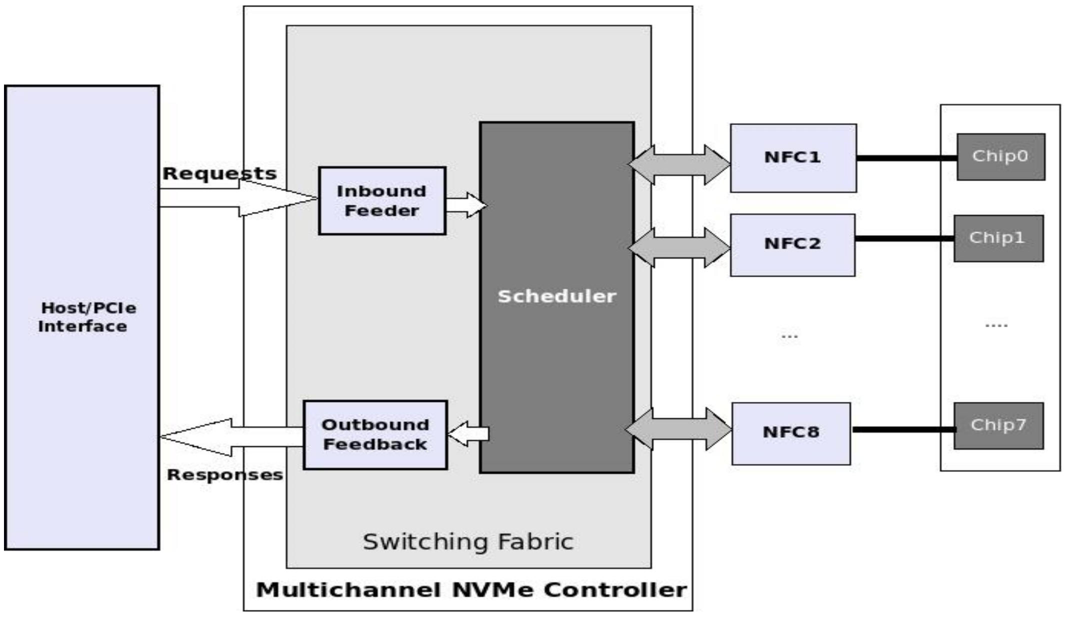


Figure 3.1: Multi-Channel NVMe Controller Architecture

3.1.1 The Switching Fabric

We design the switching fabric to implement our multi-channel parallel access mechanism. The switching fabric can accept multiple requests simultaneously. The requests store in the inbound feeder while awaiting dispatch to the target flash chip. Then according to the busy/idle status of each single NFC and the single NFC number of the request(obtained after mapping the logical address to the corresponding NFC) in the inbound feeder, the scheduler sends select signals to establish a connection between the NVMe controller and a single NFC, which is nothing but a channel has been activated. Through this channel we transmit the data, commands, and addresses between the single NFC and the NVMe controller. And at the same time, the other channels can be used to execute multiple requests concurrently. When the request has been executed completely, i.e. the single NFC is idle, the channel is disconnected by the scheduler.

When the Scheduler dispatches the requests, we have the following collision: If the target of the commands is the same chip, then there is a collision. This situation is that

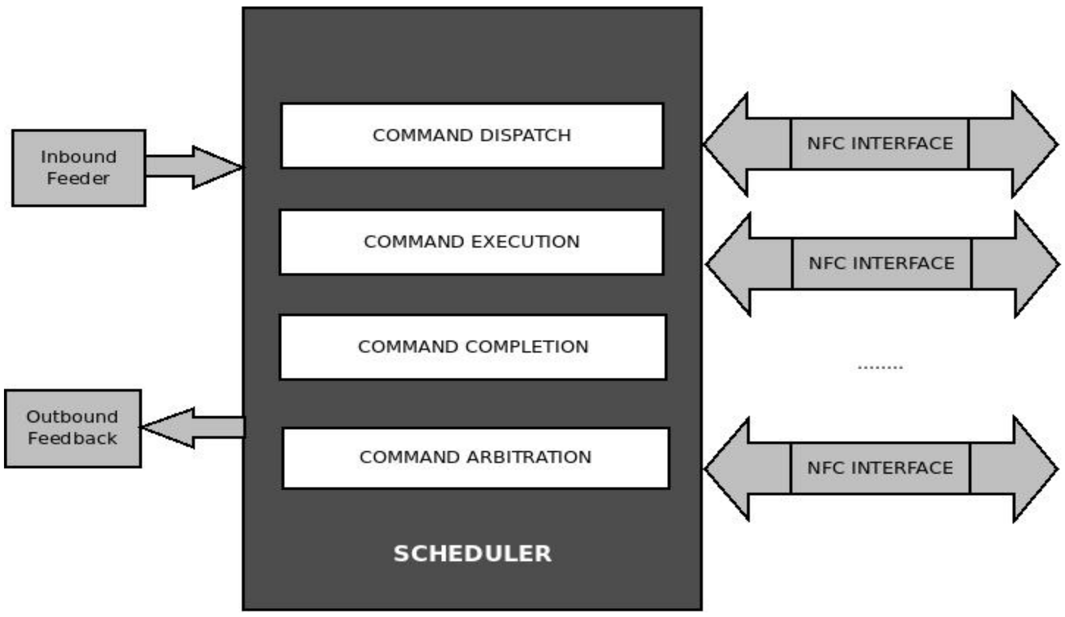


Figure 3.2: Multi-Channel NVMe Modules and Interfaces

while a single NFC is busy executing a command no matter what it is, another command in the inbound feeder wants to use this single NFC. In this case, until that NFC executes the former command we keep the latter waiting.

3.1.2 Inbound Feeder

The commands from the various submission queues are obtained by a simple Round Robin basis and are enqueued in an internal FIFO called the Inbound Feeder feeds the command to the various NFCs through the Scheduler.

3.1.3 Scheduler

The Scheduler maintains a table, mapping a particular logical address to a NFC. Each NFC is allocated a particular number/index, so when a command has to be dispatched to a NFC, we use the logical address in the command to lookup this table to find the right NFC number/index. The following state machine diagram (Figure 3.3)describes the operation of the Scheduler.

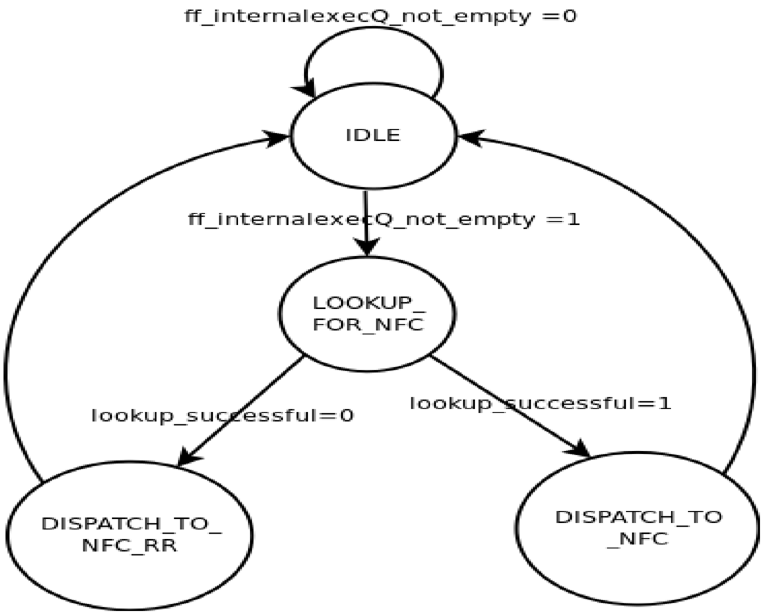


Figure 3.3: State Machine describing the working of Scheduler

3.2 Modules Description

The entire design of the controller is divided into five logical modules.Three state machines are defined to assist the modules in deriving their respective functionality. Detailed block diagram showing the modules and state machines is shown in the figure 3.4

3.2.1 Command Dispatch

This module dispatches the commands waiting in the inbound feeder to the NFCs after looking up the Logical Sector address and NFC mapping table. The state machine diagram below explains the dispatch process.

IDLE State

If the inbound feeder is not empty then we proceed to the lookup state. Else we remain

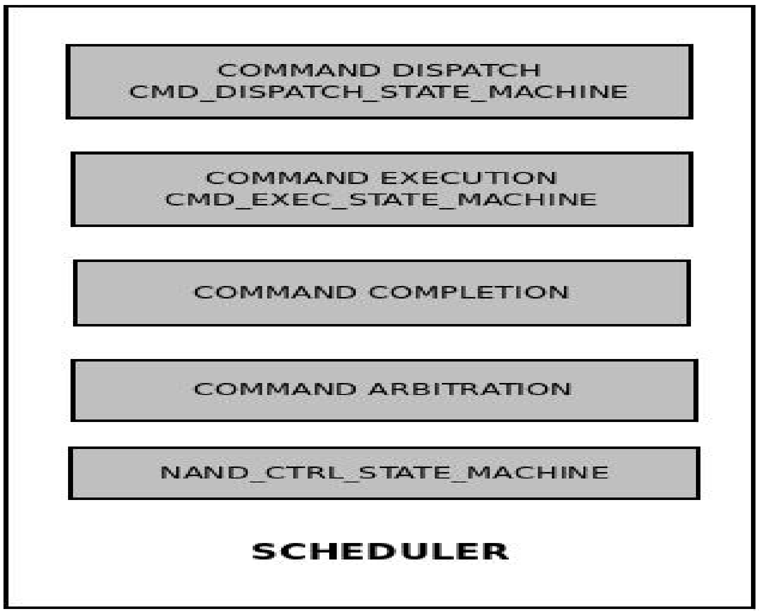


Figure 3.4: Multi-Channel NVMe Modules and State Machines

in this state.

Lookup State

Here, after fetching a command from the inbound feeder we lookup the Logical Address-NFC mapping table to get the NFC index to which the command has to be dispatched. If the lookup is successful then we go to Dispatch Idle NFC state else transition to Dispatch RR .

Dispatch Idle NFC State

We dispatch the command to the NFC obtained in the lookup stage. If its Idle then the command is executed right away else it waits in the NFC buffer.

Dispatch RR

If the lookup isnt successful then we dispatch the command any available idle NFC, preference will be given to that idle NFC with least index. If none of the NFCs are idle, then we dispatch the command to the first NFC.

3.2.2 Command Execution

The NVM Command Execution State Machine executes the three NVM Commands, namely Read from NAND,Write to NAND and Erase. The Host doesn't issue the erase command, but we need this since, an erase operation has to precede write. The state machines for command execution before and after introducing erase operation are shown in the figures fig:beforeerase and 3.6 respectively.

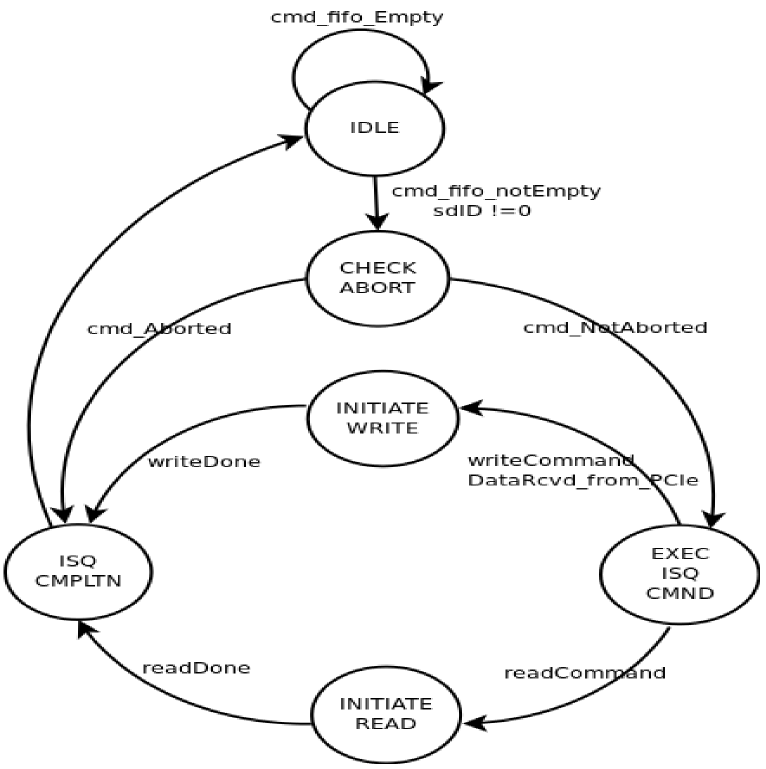


Figure 3.5: Command Execute State Machine-without Erase

Idle State

The command execution state machine is in idle state as long as the internal command execution queue is empty.

ISQ Command Check Abort State

This state checks if the command is to be aborted or not. The Abort command in the Admin Command set decides which command is to be aborted. The command to be

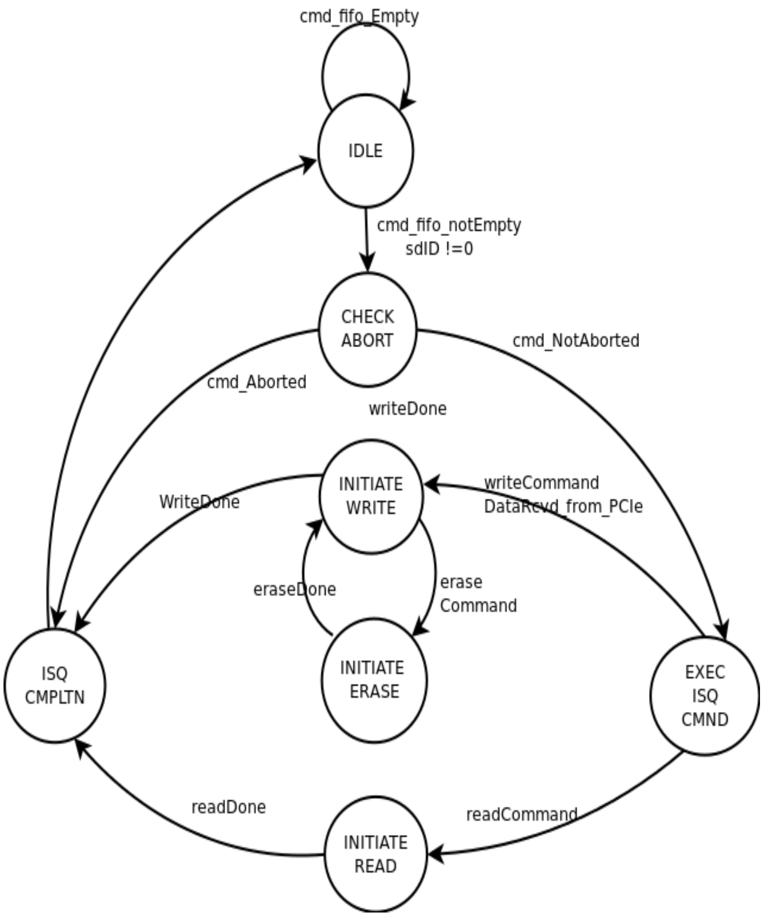


Figure 3.6: Command Execute State Machine-without Erase

aborted is uniquely specified by the command ID and the Submission Queue ID.

ISQ Command Execution State

This State executes the NVM Commands, namely Read and Write. If it is a read command then it immediately transitions to Initiate Read State. However, if it is a Wirte command then the controller raises the wire `wr_req_read_data`. The PCIe request State Machine acknowledges by raising the signal `rg_read_data_accepted`. The controller send request to send read TLP to the PCIe controller, and waits for the data to be filled in the data buffer. Once, the data buffer is filled, execute state machine transitions to Initiate Write State Machine.

Initiate Write State

This state initiates the write operation to the Nand Flash Controller. But before this an erase operation has to happen.Once the erase is over the controller starts the Write operation in the Nand Flash Control State Machine by asserting the signal `rg_initiate_write`. Nand Flash Control state machine acknowledges the execution machine after sending the data in the NVMe buffer to the Nand Flash Controller buffer. Later the controller transitions to ISQ Command Completion State.

Initiate Erase State

This state initiates the erase operation to the Nand Flash Controller.The controller starts the Erase operation in the Nand Flash Control state machine by asserting the signal `rg_initiate_erase`.Once the erase is done we resume to write process.

Initiate Read State

This state initiates the read operation to the Nand Flash Controller. The controller asserts the signal `rg_initiate_read`, to start the Read operation in the Nand Flash Control State Machine. Nand Flash Control State Machine sends the read command to NFC and then comes to its default or idle state. When the NFC is ready with the data to be Read in its Data buffers, then it generates an interrupt to the controller. Upon reception of the interrupt, the Data Transfer State Machine is triggered to transfer the data from the NFC buffers to the Host Memory in the form of Write TLP payload. Controller then transitions to ISQ Completion State.

ISQ Command Completion

This state sends ISQ Command Specific Completions to the Host memory.

3.2.3 Command Completion

Command Completions are a means by which the controller notifies the host about the status of each of the commands that it has issued. This logical module takes the status for every command, forms the command completion structure and then sends it to the host.

3.2.4 Command Arbitration

The Commands are dispatched in Round Robin basis to distribute the load uniformly across the NFCs.

3.2.5 NAND Control State Machine

This state machine generates control and address signals to access the Nand Flash Controller. It writes read/write commands to the memory mapped controller registers of the Nand Flash Controller to initiate read/write operations respectively.

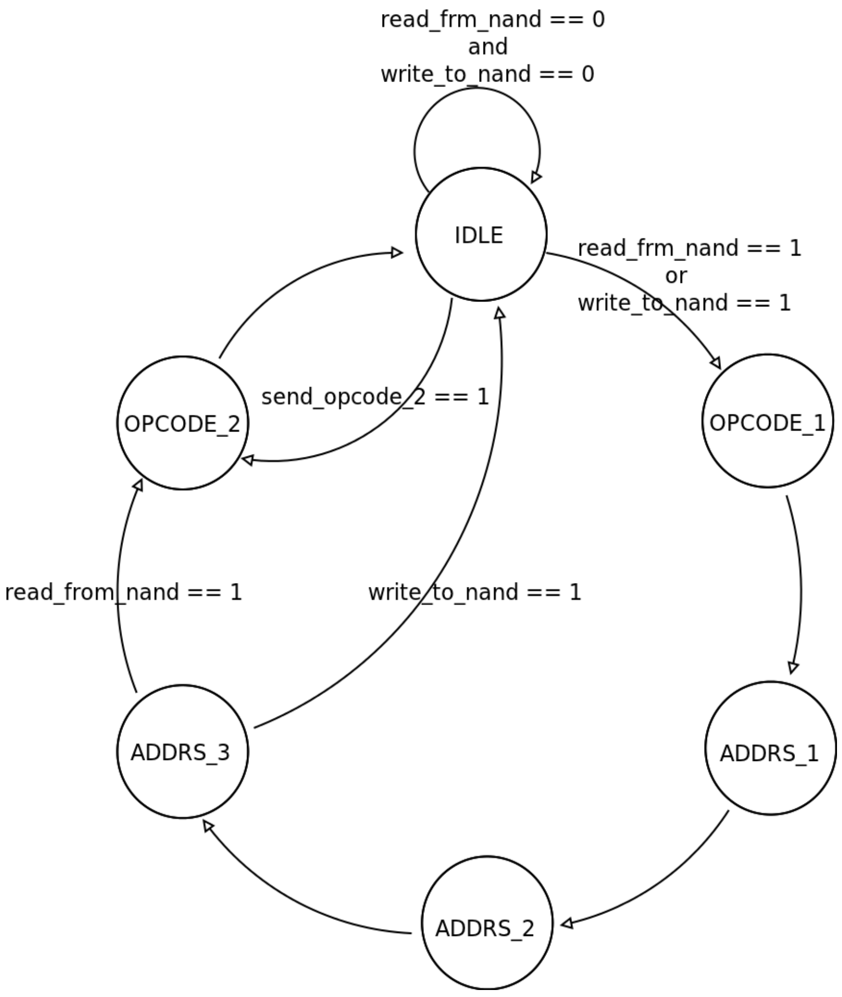


Figure 3.7: Nand Control State Machine

IDLE STATE

The state machine waits for the signals `read_frm_nand` or `write_to_nand` to be asserted . When one of the signals is asserted, state transition to OPCODE1 state occurs. There is no possibility for both the signals to be asserted at the same time.

OPCODE_1 STATE

The nand flash controller needs a start command and end command to initiate either read or write operation. Nand Chip enable and Write Enable are asserted and the opcode is written into the nand control register by supplying address of the "command register" in the address lines and opcode in the data lines.

ADDRESS_1 STATE

Nand Flash requires the "page address" to be sent in three clock cycles. This state provides the first part of the page address.

ADDRESS_2 STATE

This state provides the second part of the page address.

ADDRESS_3 STATE

This state provides the third part of the page address. In the sequence of read operations the next state is OPCODE_2, to supply the End Command which initiates the read operations in the Nand Flash Controller. NVMe controller should then wait until the Nand Flash Controller gets the required data into its Data Buffers. Nand Flash Controller interrupts the NVMe controller indicating that it is ready with the requested page in its buffers. In the sequence in of Write operations, the state machine directly jumps to IDLE state after supplying the third part of address. The controller should then send the data to be written to the NFC buffers in the subsequent cycles, followed by the second opcode or end-command.

OPCODE_2

This state provides the End Command for read and write operations.

CHAPTER 4

Exploiting I/O Parallelism

There are two kinds of I/O parallelism we can exploit. One is intra-request parallelism which denotes the parallel execution of a single request to reduce service time. The other is inter-request parallelism which indicates the parallel execution of many different requests to improve throughput. In order to exploit intra-request parallelism, we can adopt a **striping** technique. The striping technique spreads out a request across multiple channels. Figure 4.1a illustrates that the request 1 is divided into two sub-requests and those sub-requests are handled by two channels in a parallel way.To exploit inter-request parallelism, we use two techniques: **interleaving** and **pipelining**.

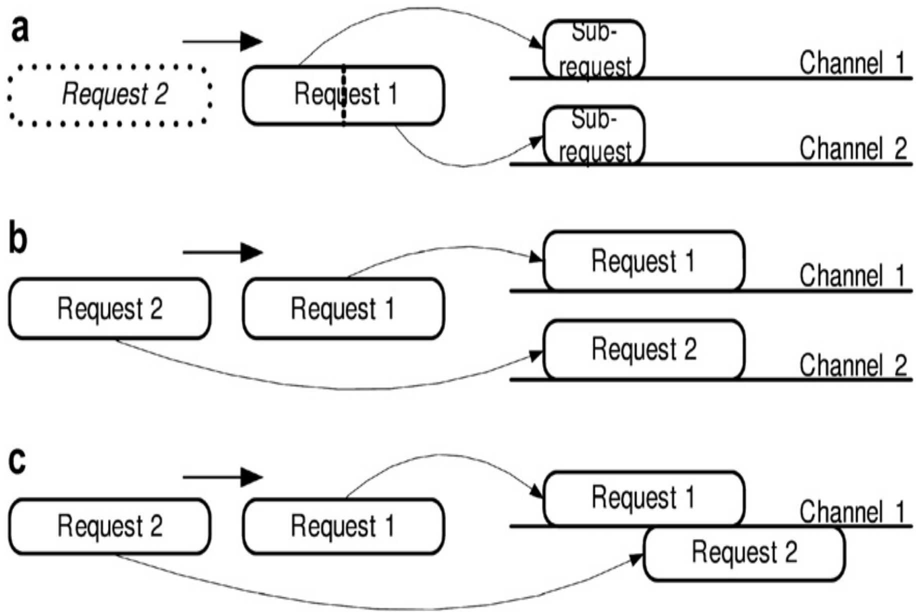


Figure 4.1: Three optimization techniques to exploit I/O parallelism. (a) Striping technique; (b) interleaving technique; and (c) pipelining technique.

In the interleaving technique, several requests are handled in parallel by using several channels. For example, in Figure 4.1b, two requests are processed simultaneously with two channels. The pipelining technique overlaps the processing of two requests as presented in figure 4.1c. While the interleaving technique and the striping technique occupy more than two channels, pipelining technique occupies only a single channel.

4.1 Striping

We define the striping level to be the number of channels that are used to handle a single request. Those channels form a channel group. Each channel in the same group gets its corresponding portion of data to handle the request. Let us assume that the striping

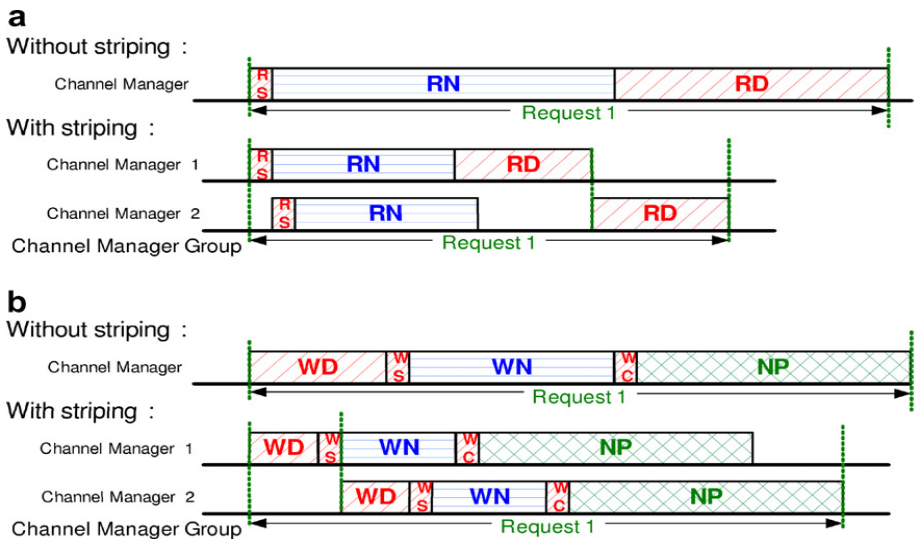


Figure 4.2: The striping technique with the striping level of two. (a) Read operation and (b) write operation.

level is two. A channel group is composed of two channels. A request is divided into two sub requests as illustrated in Figure 4.2.

In case of read operation (Figure 4.2a), commands and addresses of two sub-requests are given to each channel sequentially (RS) through the host interface. Each channel reads data from NAND flash memory to the buffer of the channel concurrently (RN). After that, the data in the buffers are copied to host memory sequentially (RD) through the host interface. The case of write operation (Figure 4.2b) is similar to that of read operation except that the sequence of the data transfer phase and the busy phase is reversed.

4.2 Interleaving

The interleaving technique exploits inter-request parallelism using multiple channels similar to the striping technique. The difference between interleaving and striping is the number of requests handled simultaneously. The interleaving technique handles several requests at once, while the striping technique handles only one request with several channels. We define the interleaving level to be the number of requests that can be handled simultaneously. Let us assume that the interleaving level is two. In Figure 4.3, request 1 and request 2 are handled concurrently by the channel1 and the channel2, respectively. The control order of read and write operations with interleaving are similar to that with striping except that each channel handles its own request separately. For write operation, we redirect a write request to one of the available channels. A simple distribution policy is a round-robin policy, where the channel is decided based on the logical address. In case of read operation, however, we have no freedom to distribute incoming read requests because data are already stored in a specific location. In any case, the throughput of read or write operations can be degraded when the requests are skewed towards a specific channel.

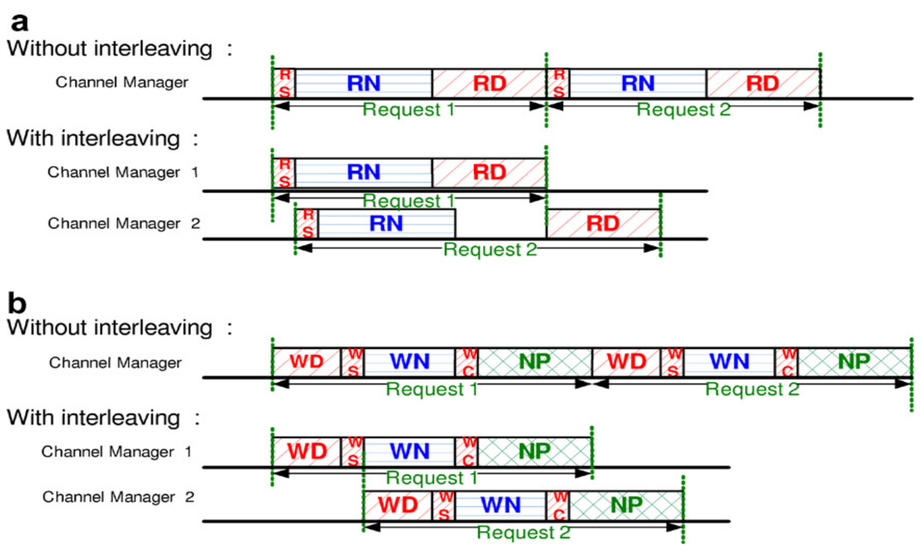


Figure 4.3: The interleaving technique with the interleaving level of two. (a) Read operation and (b) write operation.

4.3 Pipelining

The Pipelining technique utilizes inter-request parallelism between the host interface and the channel manager. Since the host interface and the channel manager operate concurrently, we can send the next command to the same channel manager using the other buffer without waiting for the completion of the previous command as shown in Figure 4.4. The NAND interface of the channel manager handles the previous command and the host interface deals with the next command simultaneously. In case of read operation (Figure 4.4a), when the data transfer from NAND flash memory to the buffer 1 (RN 1) is finished, the channel manager sends an interrupt signal to the host system. Since NAND flash memory is free at this point, we can send request 2 to the channel manager using the buffer 2 before the data in buffer 1 is copied to the host memory (RD 1). Therefore, the channel manager moves the data of request2 from NAND flash memory to the buffer 2 using the NAND interface (RN 2) while the data in buffer 1 is transferred to the host memory using the host interface (RD 1). For write operation (Figure 4.4b), the data transfer of request 1 from the buffer 1 to NAND flash memory (WN 1) overlaps with the data copy of request 2 from the host memory to the buffer 2

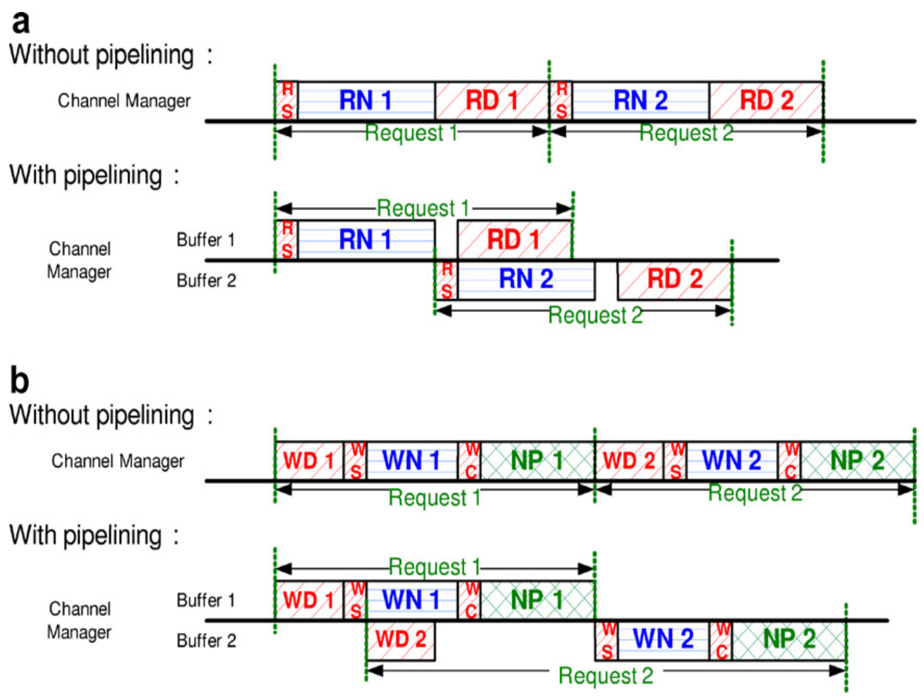


Figure 4.4: The Pipelining technique. (a) Read operation and (b) write operation.

4.4 Implementing Striping, Interleaving and Pipelining

Due to the very nature of the Multi-Channel NVMe controller architecture Interleaving is taken care implicitly. The commands are dispatched in Round Robin basis for uniform load distribution. With regards to Pipelining, if there is a command waiting to be dispatched to a particular NFC which is busy executing a previous command, then that is handled as discussed in 4.1.3. Striping is given the last priority or in other words Striping is not used as much as Interleaving and Pipelining. This is because commands hardly span over multiple dies or chips. When we write or read, the amount of data written or read is restricted to a few pages or blocks or planes in a die. Rare is the case where the data spills over a die and has to be accommodated in the next die. Only in those cases Striping technique comes into picture.

4.5 Design Challenges

Some of the challenges faced and the decisions taken during the design are explained below.

Storing commands waiting to be dispatched

If the NFC obtained after the lookup stage is not idle, then storing the command(waiting to be dispatched) in a FIFO external to that NFC leads to complication. Because i enqueue the command now and then later I've to comeback to dispatch it. Instead, we can have one dedicated buffer to every NFC. The command waiting to be dispatched to a particular NFC can wait in its specific buffer. As and when the NFC becomes free, the command in the buffer can be executed.

Order of dispatch of commands to the NFC

This applies to the new command, about to be dispatched to a NFC. Now in order to distribute the load uniformly we can go for Round Robin distribution. Then in that case irrespective of whether the NFC is idle or not, it gets its share of the load. If the NFC is idle then the command is executed straight away else it has to wait in the NFCs buffer. The other way to dispatch a new command is to deliver it to any available idle NFC right away. If more than one NFC is idle(say NFC0, NFC1..), then deliver the command to the NFC with least index (in this case NFC0).The latter method is followed.

4.6 Verification

The entire verification of the controller was carried out in the BlueSim simulator.The test setup used and the verified test cases are described in the following sub-sections.

4.6.1 Verification Setup

The verification setup for verifying the NVMe controller is shown in the figure 4.10

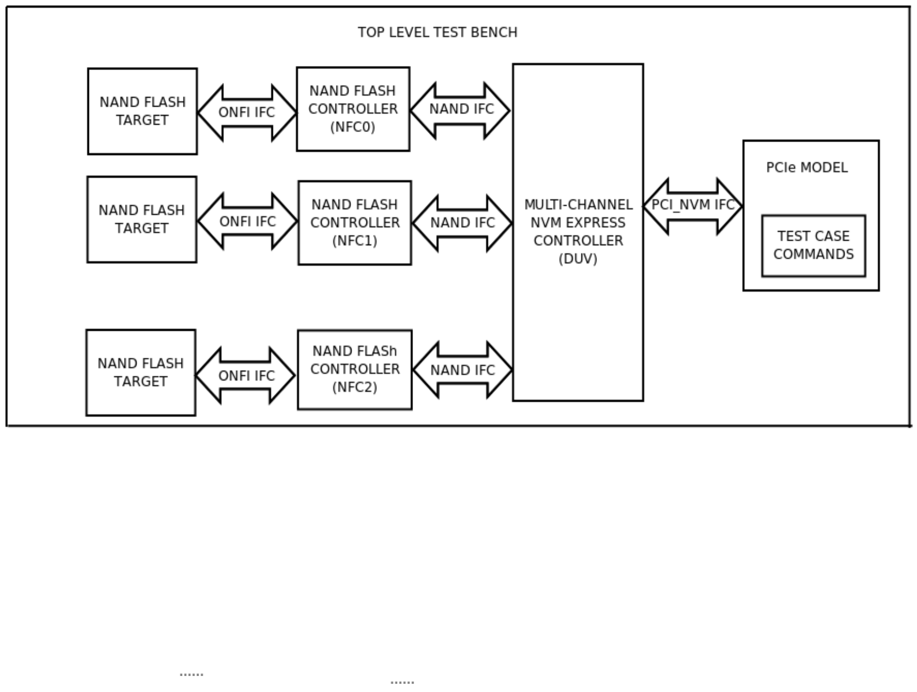


Figure 4.5: Verification Setup

The Multi-Channel NVMe controller is the Design Under Verification(DUV).The Test Case Commands is a memory model in BSV, which stores the required commands for NVM Express. This is accessed just as the Host Main Memory is accessed for commands from the submission queues.

4.6.2 Verification Test Cases

While our main focus is to verify the Multi-Channel aspect of the controller, test cases for normal operation of NVMe Controller have also been verified nonetheless to ensure nothing is broken.The test cases are as follows.

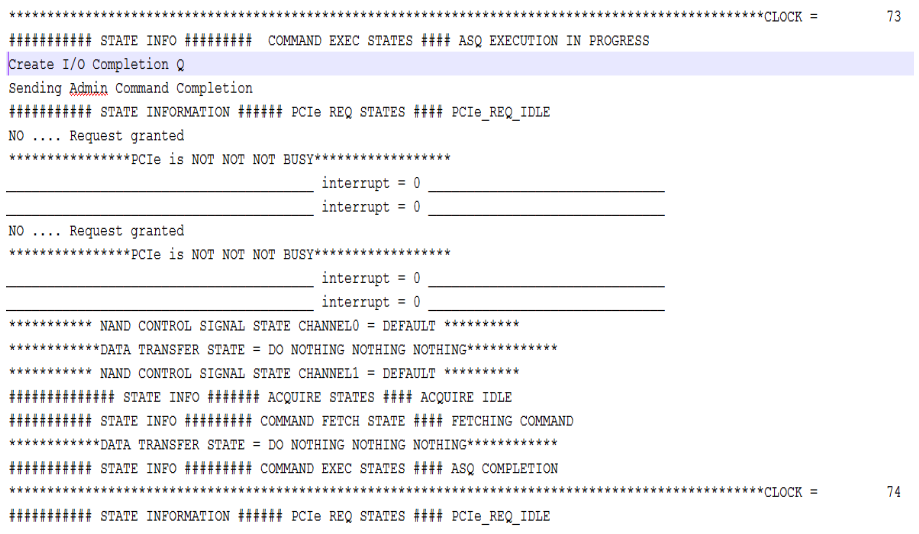
1. Test cases for the I/O command set, that includes Write and Read commands.

2. Test cases for all the commands in the Admin Command set.

3. Test cases for multi-channel operation through issuing multiple commands parallelly.

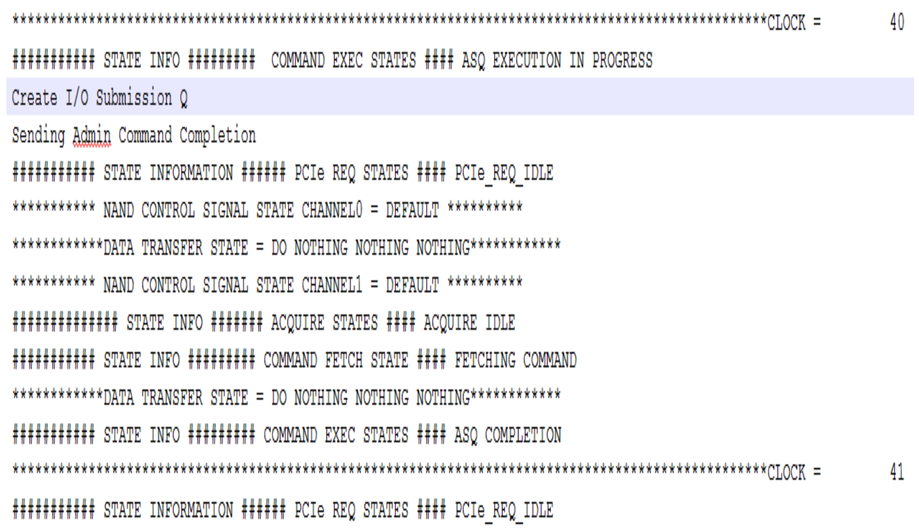
4.7 Simulation Results

The Bluespec Simulation Results are shown below.



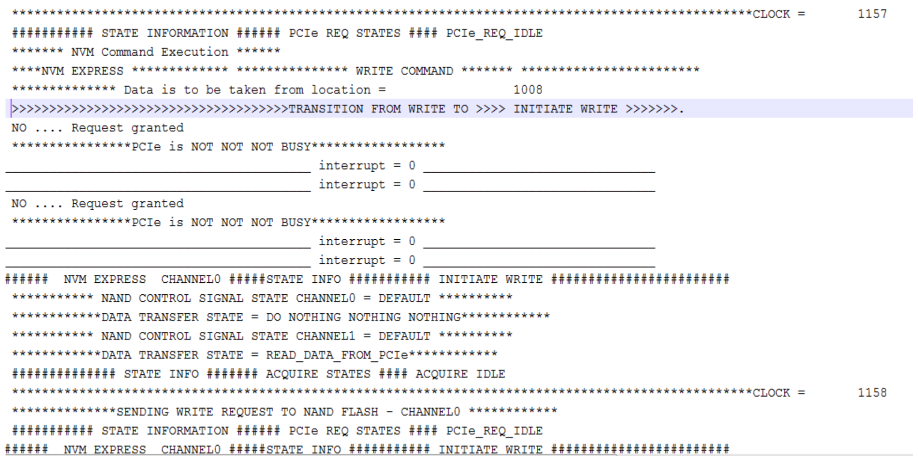
```
****************************************************************************************************CLOCK =         73
########### STATE INFO #########  COMMAND EXEC STATES #### ASQ EXECUTION IN PROGRESS
Create I/O Completion Q
Sending Admin Command Completion
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY*****************
________________________________________ interrupt = 0 ______________________________
________________________________________ interrupt = 0 ______________________________
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY*****************
________________________________________ interrupt = 0 ______________________________
________________________________________ interrupt = 0 ______________________________
*********** NAND CONTROL SIGNAL STATE CHANNEL0 = DEFAULT **********
************DATA TRANSFER STATE = DO NOTHING NOTHING NOTHING************
*********** NAND CONTROL SIGNAL STATE CHANNEL1 = DEFAULT **********
############## STATE INFO ####### ACQUIRE STATES #### ACQUIRE IDLE
########### STATE INFO ######### COMMAND FETCH STATE #### FETCHING COMMAND
************DATA TRANSFER STATE = DO NOTHING NOTHING NOTHING************
########### STATE INFO ######### COMMAND EXEC STATES #### ASQ COMPLETION
****************************************************************************************************CLOCK =         74
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
```

Figure 4.6: Creation Of Completion Queue



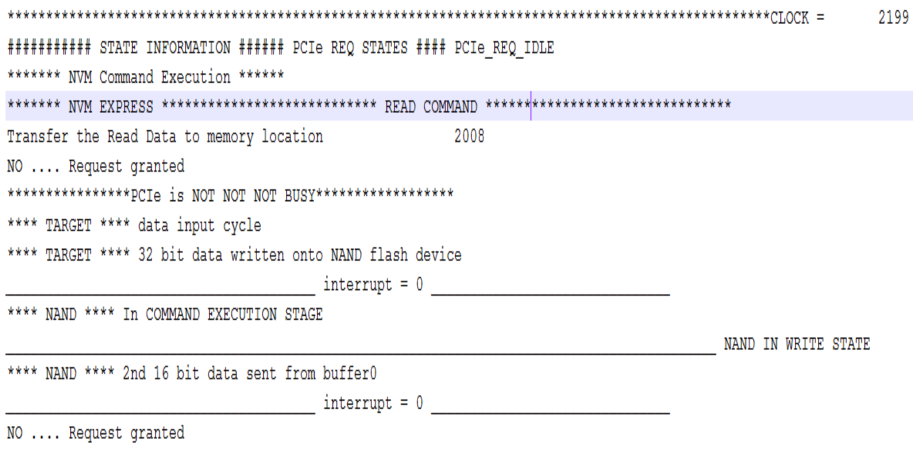
```
****************************************************************************************************CLOCK =         40
########### STATE INFO #########  COMMAND EXEC STATES #### ASQ EXECUTION IN PROGRESS
Create I/O Submission Q
Sending Admin Command Completion
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
*********** NAND CONTROL SIGNAL STATE CHANNEL0 = DEFAULT **********
************DATA TRANSFER STATE = DO NOTHING NOTHING NOTHING************
*********** NAND CONTROL SIGNAL STATE CHANNEL1 = DEFAULT **********
############## STATE INFO ####### ACQUIRE STATES #### ACQUIRE IDLE
########### STATE INFO ######### COMMAND FETCH STATE #### FETCHING COMMAND
************DATA TRANSFER STATE = DO NOTHING NOTHING NOTHING************
########### STATE INFO ######### COMMAND EXEC STATES #### ASQ COMPLETION
****************************************************************************************************CLOCK =         41
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
```

Figure 4.7: Creation Of Submission Queue



```
************************************************************************************************CLOCK =       1157
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
******* NVM Command Execution ******
****NVM EXPRESS ************* *************** WRITE COMMAND ******* ************************
************** Data is to be taken from location =                 1008
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>TRANSITION FROM WRITE TO >>>> INITIATE WRITE >>>>>>>.
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY*****************
_________________________________________ interrupt = 0 _______________________________
_________________________________________ interrupt = 0 _______________________________
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY*****************
_________________________________________ interrupt = 0 _______________________________
_________________________________________ interrupt = 0 _______________________________
######  NVM EXPRESS  CHANNEL0 #####STATE INFO ########### INITIATE WRITE ########################
*********** NAND CONTROL SIGNAL STATE CHANNEL0 = DEFAULT **********
************DATA TRANSFER STATE = DO NOTHING NOTHING NOTHING************
*********** NAND CONTROL SIGNAL STATE CHANNEL1 = DEFAULT **********
************DATA TRANSFER STATE = READ_DATA_FROM_PCIe************
############## STATE INFO ####### ACQUIRE STATES #### ACQUIRE IDLE
************************************************************************************************CLOCK =       1158
**************SENDING WRITE REQUEST TO NAND FLASH - CHANNEL0 ************
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
######  NVM EXPRESS  CHANNEL0 #####STATE INFO ########### INITIATE WRITE ########################
```

Figure 4.8: Initiate Write



```
************************************************************************************************CLOCK =       2199
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
******* NVM Command Execution ******
******* NVM EXPRESS *************************** READ COMMAND ******************************
Transfer the Read Data to memory location               2008
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY*****************
**** TARGET **** data input cycle
**** TARGET **** 32 bit data written onto NAND flash device
________________________________________ interrupt = 0 _______________________________
**** NAND **** In COMMAND EXECUTION STAGE
___________________________________________________________________________________________ NAND IN WRITE STATE
**** NAND **** 2nd 16 bit data sent from buffer0
________________________________________ interrupt = 0 _______________________________
NO .... Request granted
```

Figure 4.9: Initiate Read

4.8 Synthesis Report

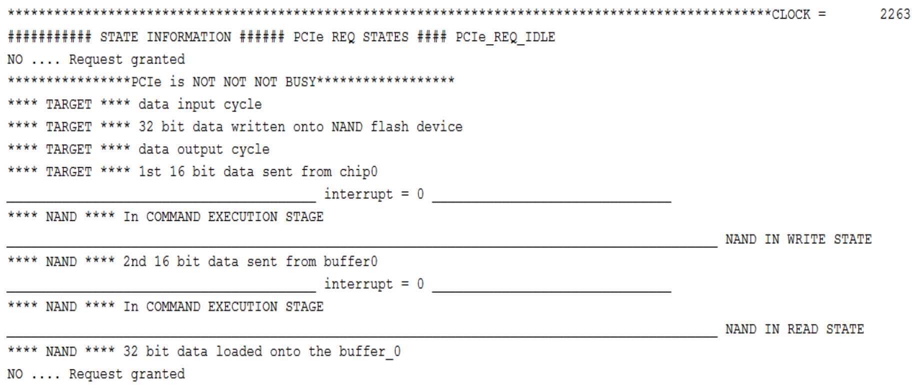
The design was synthesized for the device *Virtex 6 XC6VLX240T-FF1156*. The slice utilization and timing summary is provided below.

Number of Slice Registers used : 4731

Number of Slice LUTs used : 6351

Number of LUT Flip Flop Pairs used : 3126

Number of Block RAMs used : 14



```
***************************************************************************************************CLOCK =       2263
########### STATE INFORMATION ###### PCIe REQ STATES #### PCIe_REQ_IDLE
NO .... Request granted
****************PCIe is NOT NOT NOT BUSY******************
**** TARGET **** data input cycle
**** TARGET **** 32 bit data written onto NAND flash device
**** TARGET **** data output cycle
**** TARGET **** 1st 16 bit data sent from chip0
________________________________________ interrupt = 0 _______________________________
**** NAND **** In COMMAND EXECUTION STAGE
____________________________________________________________________________________________ NAND IN WRITE STATE
**** NAND **** 2nd 16 bit data sent from buffer0
________________________________________ interrupt = 0 _______________________________
**** NAND **** In COMMAND EXECUTION STAGE
____________________________________________________________________________________________ NAND IN READ STATE
**** NAND **** 32 bit data loaded onto the buffer_0
NO .... Request granted
```

Figure 4.10: Read Write Command Execute Parallely

Max Frequency of operation : 205 MHz.

CHAPTER 5

Conclusions and Future Work

The Multi-Channel NVM Express controller was designed retaining all the basic functionality and mandatory command sets. Further to exploit I/O parallelism we implemented three techniques- Striping, Interleaving and Pipelining. Also as a prerequisite to write operation, block erase operation was implemented.The operation of Multi-Channel NVMe controller was verified by issuing multiple commands.

The Multi-Channel NVM subsystem can be improved in the following ways to exploit more I/O parallelsim.

1. There may be multiple commands outstanding to different LUNs at the same time. To get further parallelism within a LUN, multi-plane operations may be used to execute additional dependent operations in parallel [6]
2. A logical unit (LUN) is the minimum unit that can independently execute commands and report status. Specifically, separate LUNs may operate on arbitrary command sequences in parallel. For ex, it is permissible to start a page program operation on LUN0 and then prior to the operation's completion to start a Read command on LUN1.See Multiple LUN operation restrictions in section 3.1.3 [6]
3. An FTL(Flash Translation Layer) processor can be added to the subsystem to carry- out Flash Management activities at software level. A brief detail about FTL and its necessity is provided in the appendix.

APPENDIX A

NVM Subsystem with FTL Processor

Flash Translation Layer is a very essential part of any SSD based Non Volatile Memory subsystem. There are three fundamental limitations for any SSD based system [3]. FTL helps in overcoming these limitations. The three limitations and the steps taken by FTL to overcome it or to hide it from the host are detailed below.

Limitation 1 : Erase-before-write

One of the limitations of the Flash memories is their "Erase before write" for re-writ ting into an already written location. This means that a page data can be written only into a location which is in erased state. FTL is primarily used to manage these activities by hiding them to the Host processor. The typical steps performed by the FTL while performing an over-write operation is shown below :

1. Takes the incoming logical page address, checks if the location pointed by the address in Nand Chip is in "erase" state or "written" state.
2. If it is in erase state then the page is written to that location.
3. If it is in "written" state then, it looks for a "new" location in the same block which is in "erased" state and maps its address to the present logical address. This new address is the actual physical address of the page. The incoming data page is written to this physical page address. There are various algorithms to perform this mapping operation.
4. Whenever the host gives a read command with the previous logical page address, the FTL fetches the page from its "mapped physical page address". In this way the host is unaware of the address translation.

Limitation 2 : Flash can be written in pages but can only be erased as a Block of pages

Flash memories can be written in units of pages, but they can be erased only in larger units called “blocks” of pages. If a particular set of pages are no longer required and are to be erased then the FTL performs a sequence of steps called “*Garbage Collection*”. The steps are detailed below.

1. If a set of pages in a block are no longer needed (stale pages) , then the FTL first reads the other “valid“ pages in that block.
2. Re-writes these valid pages into another ”erased“ block .
3. Then erases the previous block completely.

This is also implemented by the FTL unit, and there are various algorithms to perform it.

Limitation 3 : Flash can be ”Read“ or ”Written” only for a certain number of times

If a particular block were erased and programmed repeatedly without writing to any other blocks, the one block would wear out before all the other blocks, thereby prematurely ending the life of the SSD. For this reason, FTL implements a technique called “*wear leveling*“ to distribute writes as evenly as possible across all the flash blocks in the SSD.

Suggested Modifications to the present NVM Subsystem

The modified NVM Subsystem with the inclusion of the FTL processor is shown in the figure A.1

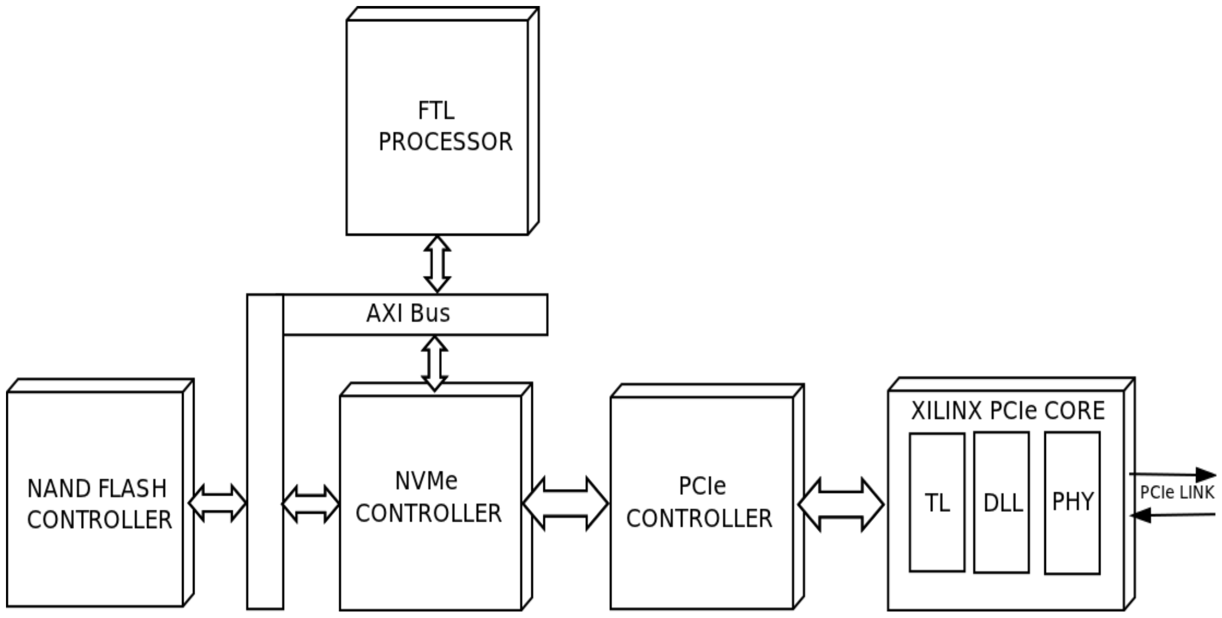


Figure A.1: Future Work on NVM Subsystem

The modifications that could be made are :

1. AXI wrappers could be put around the NVMe controller and Nand Flash Controller, so as to communicate with the FTL processor.
2. FTL processor takes the Logical Address from the NVMe controller and performs the Flash Management operations and sends the ”Control signals“ to the Nand Flash Controller and then initiate the required data transfer between NVMe controller and Nand Flash Controller.

REFERENCES

[1] Bluespec, Inc, *Bluespec System Verilog Reference Guide*, revision: 17 ed., 2012.

[2] R. S. Nikhil and K. Czeck, *BSV by Example*. Bluespec, Inc, 2010.

[3] L. C. Rino Micheloni and A. Marelli, *Inside NAND Flash Memories*. Springer, 2010.