Design and Implementation of ONFi 4.0 compliant "Nand Flash Controller"

A Project Report

submitted by

Laxmeesha S

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

Master Of Technology

Under the guidance of

Dr.V.Kamakoti

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2015

THESIS CERTIFICATE

This is to certify that the thesis titled **Design and Implementation of ONFi 4.0 compliant "Nand Flash Controller"**, submitted by **Laxmeesha S**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our 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
Project 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. I take this opportunity to express my sincere gratitude and appreciation to them.

First and foremost, I offer my earnest gratitude to my guide, Dr. V. Kamakoti whose knowledge and dedication has inspired me to work efficiently on the project and I thank him for motivating me, and allowing me freedom and flexibility while working on the project.

I would also like to express my sincere gratitude to Mr. G.S. Madhusudan whose persistent efforts and invaluable suggestions have been motivational and helpful. I appreciate his prolonged support and his unprecedented help throughout the project period.

My special thanks and deepest gratitude to Neel, Santosh and Vishvesh who have been very supportive. They have enriched the project experience with their active participation and invaluable suggestions.

I thank my friends at IIT Madras for a great experience. I would like to thank my parents for supporting and encouraging me throughout my stay at IIT Madras and I dedicate this project to my family and friends.

ABSTRACT

The physical limitations in the performance of magnetic media such as Hard Disk Drives has propelled a performance gap between the processor and the disk based storage units. The storage media performance has been the bottleneck of progress for quite a time. The performance gap has worsen since the introduction of multi-core processors.This is where Solid State Drive(SSD) comes into picture. Although SSDs are slightly costlier and provide lesser storage than HDDs, they provide a greater performance improvement as it eliminates the mechanical limitations of HDDs.

NAND Flash Memory is being adopted in SSDs. In the initial days, certain features of NAND Flash 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. Standardization was clearly needed, and the first industry accepted standard was ONFi 1.0 and the latest being ONFi 4.0.

In order to completely utilize the performance of SSDs, a Non Volatile Memory Subsystem was designed(based on the NVM Express Specification) along with a NAND Flash Controller(based on the ONFi specification). The communication to NVMe subsystem is through PCI Express interface and the command set is based on NVMe Specification. The communication to NAND Flash Memory is through NAND Flash Controller and the command set is based on ONFi Specification. The present project deals with the design and implementation of NAND Flash Controller using the ONFi 4.0 standard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CPU	Central Processing Unit
BSV	Bluespec SystemVerilog
NFC	NAND Flash Controller
ONFi	Open NAND Flash Interface
NVMe	Non-Volatile Memory Express
PCIe	Peripheral Component Interconnect Express
LUN	Logical Unit Logical Unit Number
ECC	Error Correction Code
SSD	Solid-State Drive
HDD	Hard Disk Drive
BRAM	Block Random Access Memory
CLE	Command Latch Enable
ALE	Address Latch Enable
SLC	Single-Level Cell
MLC	Multi-Level Cell
TLC	Triple-Level Cell
DUV	Design Under Verification

CHAPTER 1

INTRODUCTION

1.1 Motivation

In the past the main aspect expressed about the storage media was the capacity of storage and the performance of a system was attributed to the CPU's parameters. However with the rapid advancement in the processor systems, especially with parallel computing and other services that offer immediate access to large amounts of data, the traditional hard drive hits its limits. Due to this the speed of storage media becomes a major bottleneck in system design. Hence Non Volatile Memory(NVM) based storage devices were introduced. One such kind of memory is the Flash based Solid State Drives(SSD). These drives offer a significantly higher capacity for their price. But as storage media became faster, the focus needs to be pointed to the communication, too. The traditional protocols, such as SATA, are developed and optimized for mechanical hard-drives and not for high speed Solid-State Disk. As a result, interfaces such as PCI Express have to be introduced for storage-media communication and other appropriate standards have to be introduced.

A NAND flash memory brings advantages in terms of size, weight, reliability, and energy use when compared to the hard disk drives. But it is not easy to extract the maximum performance from a 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.

The present developed Multi-Channel Non Volatile Memory Sub-system consists of a PCIe controller, NVMe Controller and Nand Flash Controller to access the NAND Flash Chips[3]. The Sub-system is shown in the figure 1.1

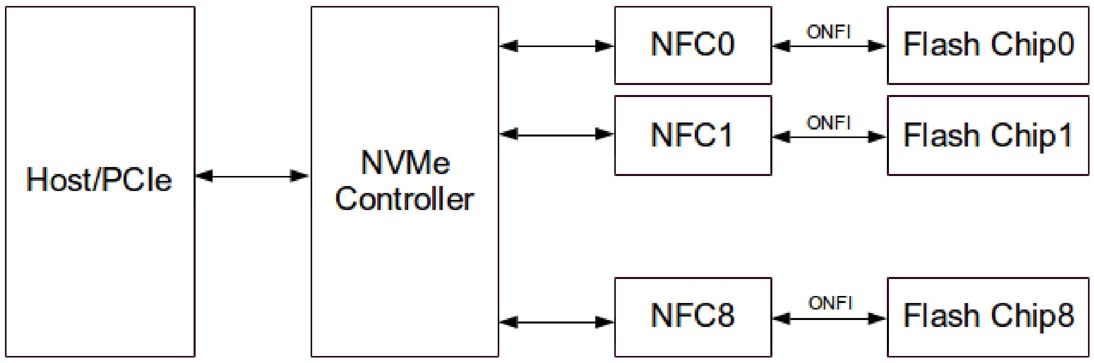


Figure 1.1: SSD Memory Subsystem[3]

1.2 Scope of Work

The importance of NAND Flash memory is growing in everything from mobile phones to industrial systems. But intitally there was no standard, and thus no simple way to introduce new NAND Flash components into existing designs.The lack of a standard caused serious design problems. The host systems had to accommodate differences between vendors devices and adapt to generational changes in parts from a single vendor. All of this made incorporating new or updated NAND Flash components extremely costly.ONFi works to solve all these issues by standardizing the NAND Flash interface reducing vendor and generational incompatibilities and accelerating the adoption of new NAND products.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.The newest Open NAND Flash Interface (ONFi 4.0) standard offers many benefits for high-performance NAND Flash applications.

The first phase of the work included examining fundamentals of the ONFi specfications ,Nand Flash Controller ,Memory architecture of Nand Flash Memory through available literature. Additionally knowledge on Bluespec System Verilog(BSV) had to be acquired since the design was to be implemented using BSV as the hardware-description language.

In the next phase the ONFi 4.0 specifiation was studied and the design concept was developed for the Nand Flash Controller. The concept was to be designed for modularity and low latency. In the next step a model was developed for the Nand Flash Memory using Bluespec System Verilog and it was verified for various cases through BSV. The NFC was implemented using Bluespec System Verilog language. Finally, test scenarios for verification of the Nand Flash Controller along with the developed Nand Flash Memory model are to be developed and executed and the hardware design has to be examined in synthesis. Additionally the Nand Flash Controller was integrated with a Micron Bus functional model of the nand flash memory and verified.

1.3 Overview of content

Chapter-2 describes the background of Flash memories and BSV.
Chapter-3 discusses the background of ONFi 4.0 standard and the description of NAND flash architecture and design.
Chapter-4 explains technique to interleave commands to exploit parallelism.
Chapter-5 includes the design and implementation of NAND flash controller and verification of the design.
Chapter-6 concludes with a short description of the future work.

CHAPTER 2

Background

In this chapter we discuss the some of the fundamentals that are needed to understand the following chapters. We start with a basic Nand Flash memory storage unit at the transistor level and discuss the whole memory as a unit. Then a small introduction is given into the Bluespec System Verilog language.

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. Flash memory based SSDs(Solid State Drives) have 100x greater throughput and instantaneous access times for quicker boot ups, faster file transfers, and overall better 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. Clearly, NAND Flash offers several compelling advantages. The one challenge is that it is not well-suited for direct random access[8].

The transistor level schematic and layout of the cells that build a Nand Flash Memory is shown in figure 2.1 and that of a Nor Flash Memory is shown in figure 2.2.

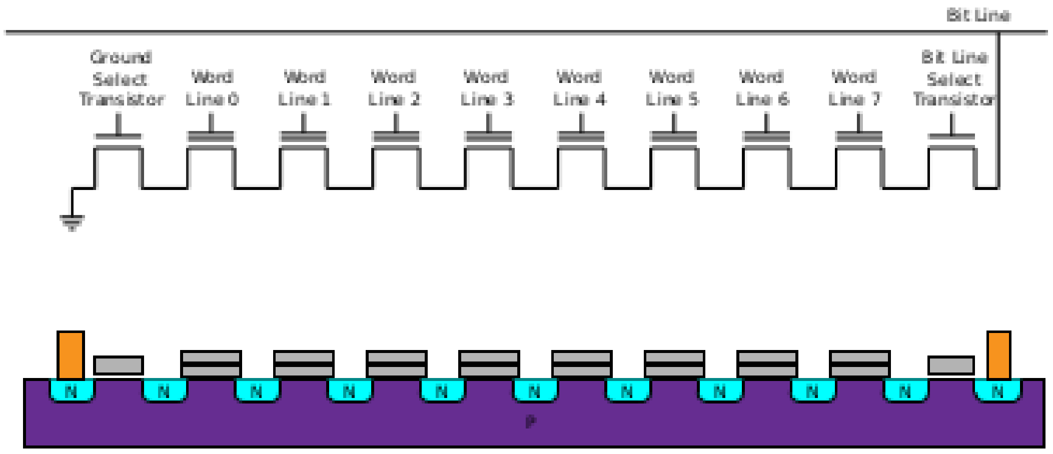


Figure 2.1: Nand Flash cell structure[3]

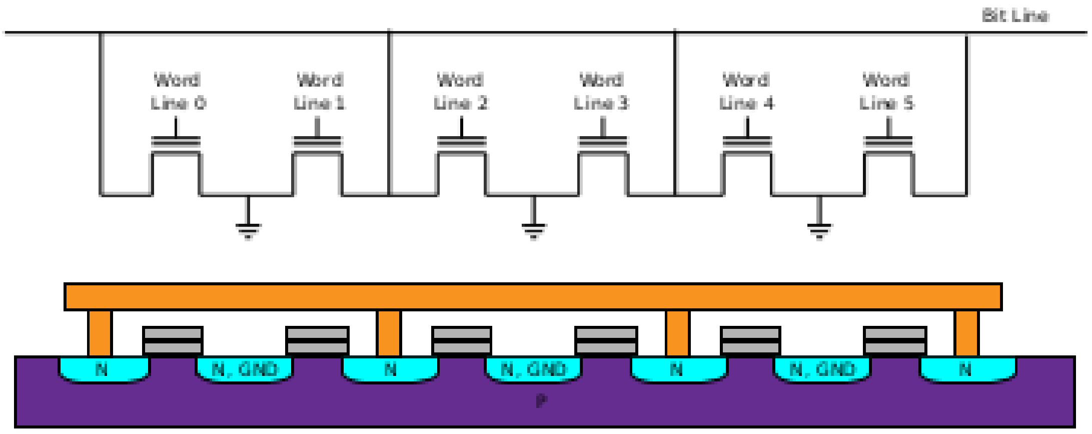


Figure 2.2: Nor Flash cell structure[3]

As shown in figure 2.1 the cell resembles a Nand gate NMOS structure for a Nand Flash and that of a Nor gate NMOS structure for a Nor Flash as seen n figure 2.2. In both the Nand cell and Nor cell the Drain/Source between adjacent transistors are shared. However every alternate connection has a ground potential in Nor cell. Hence an extra ground line has to be passed on in the layout compared to that of Nand cell. Hence very high level of integration can be achieved with the Nand Flash Memory. However, Nand Flash memories are not randomly accessible and thus have to be programmed and read in larger sectors, called pages. These pages usually have a size of four to eight kilobytes. As erase operations are realized by connecting the whole well of a block to the erase-voltage, erase operations always delete the whole block [10].

2.3 Types of NAND Flash

Each Flash cell consists of a single transistor, with an additional "floating" gate that can store charges.The amount of charge on the floating gate affects the threshold voltage of the cell Vt. Based on this we have two types of Nand Flash.

2.3.1 SINGLE LEVEL CELL (SLC) Nand Flash

A SLC Flash stores one bit value per cell, which basically is a voltage level. The bit value is interpreted as a "0" or a "1"[11]. Since there are only two states, it represents only one bit value. A "0" or "1" is determined by the threshold voltage Vt of the cell. The threshold voltage can be manipulated by the amount of charge put on the floating gate of the Flash cell. Placing charge on the floating gate will increase the threshold voltage of the cell. When the threshold voltage is high enough, the cell will be read as programmed. No charge, will cause the cell to be sensed as erased.

2.3.2 MULTI LEVEL CELL (MLC) Nand Flash

As the name suggests, there are multiple values that an MLC cell can represent. The values can be interpreted as four distinct states: "00", "01", "10", or "11"[11]. These four states yield two bits of information. Since the change in threshold between each level has decreased, the sensitivity between each level increased. Thus, more rigidly controlled programming is needed to manipulate a more precise amount of charge stored on the floating gate.

2.4 Bluespec System Verilog language

BlueSpec System Verilog is a Hardware centric language based on industry standard SystemVerilog and Verilog. Using Bluespec language leads to shorter, more abstract, and verifiable source code, as well as type-checked numeric code. BSV provides significantly higher level of abstraction than Verilog, SystemVerilog and VHDL.

2.4.1 Constructs in Bluespec System Verilog[1]

Interfaces: Interfaces provide a means to group wires into bundles with specified uses, described by methods. An interface is a reminiscent of a struct, where each member is a method. Interfaces can also contain other interfaces.

Methods: Signals and buses are driven in and out of modules with methods. These methods are grouped together into interfaces. There are three kinds of methods:

- Value Methods: Take 0 or more arguments and return a value.
- Action Methods: Take 0 or more arguments and perform an action(side-effect) inside the module.
- ActionValue Methods: Take 0 or more arguments, perform an action, and re-turn a result.

Modules: A module consists of three things: state, rules that operate on that state, and an interface to the outside world (surrounding hierarchy). A module definition specifies a scheme that can be instantiated multiple times.

Rules: Rules are used to describe how data is moved from one state to another, instead of the Verilog method of using always blocks. Rules have two components:

• Rule conditions: Boolean expressions which determine when the rule is enabled.

• Rule body: a set of actions which describe state transitions.

Functions : Functions are simply parameterized combinational circuits. Function application simply connects a parameterized combinational circuit to actual inputs.

2.4.2 Overview of the BSV build process[2]

The following are the steps involved in building a BSV design:

1. A designer writes a BSV program. It may optionally include Verilog components.
2. The BSV program is compiled into a Verilog or Bluesim specification. This step has two stages:
   - Pre-elaboration - parsing and type checking.
   - Post-elaboration - code generation.
3. The compilation output is either linked into a simulation environment or processed by a synthesis tool.

Once the Verilog or Bluesim implementation is generated, the workstation provides the following tools to help analyze the design:

• Interface with an external waveform viewer with additional Bluespec provided annotations, including structure and type definitions.

• Schedule Analysis viewer providing multiple perspectives of a modules schedule.

• Scheduling graphs displaying schedules, conflicts, and dependencies among rules and methods.

CHAPTER 3

Nand Flash and ONFi Description

NAND Flash devices include two data interfaces synchronous interface and an asynchronous interface. These devices use a highly multiplexed 8-bit bus to transfer commands, addresses, and data. Current implementation will target the asynchronous interface.There are five control signals that are used to implement the NAND Flash protocol. Additional signals control hardware write protection and monitor device status. This hardware interface creates a low-pin-count device with a standard pin-out that is the same from one density to another, allowing future upgrades to higher densities without board redesign[4].These signal pins are listed in table 3.1 along with their operation for the asynchronous interface. A logical unit (LUN), is the minimum unit that can independently execute commands and report status.

Table 3.1: Table of signal definitions.

Symbol	**Type**	**Description**
ALE	Input	**Address latch enable**: Loads an address from DQ[7:0] into the address register.
CE	Input	**Chip enable**: An asynchronous-only signal that enables or disables one or more logical units.
CLE	Input	**Command latch enable**: Loads a command from DQ[7:0] into the command register.
DQ[7:0]	Inout	**Data inputs/outputs**: The bidirectional I/Os transfer address, data, and command information.
RE	Input	**Read enable**: Transfers serial data from the NAND Flash to the host system.
WE	Input	**Write enable**: Transfers commands, addresses, and serial data from the host system.
R/B	Output	**Ready/Busy**: Indicates if all LUNS are busy.
WP	Input	**Write protect**:Enables or disables array program and erase operations.

3.1 Nand Flash Architecture

The LUN functional block diagram is shown in figure 3.1. The data, commands and addresses are multiplexed onto the same pins and received by I/O control circuits. This provides a memory device with a low pin count. The commands received at the I/O control circuits are latched by a command register and are transferred to control logic circuits for generating internal signals to control device operations. The addresses are latched by an address register and sent to a row decoder to select a row address or to a column decoder to select a column address.

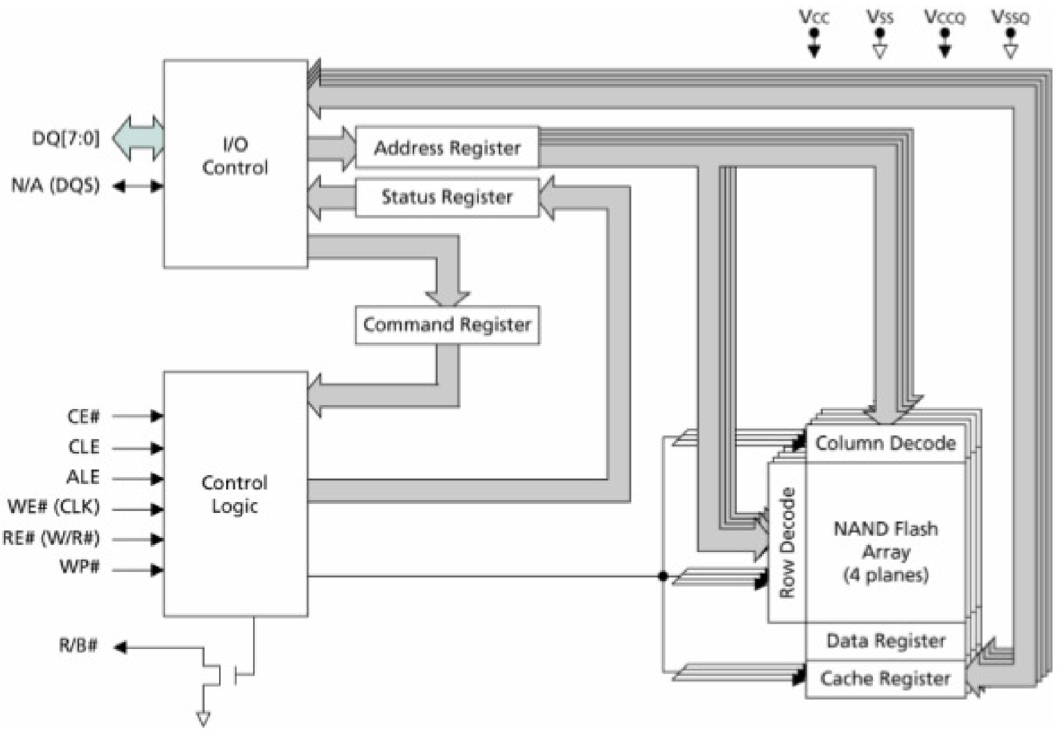


Figure 3.1: Nand Flash LUN functional block diagram[4]

The data are transferred to or from the NAND Flash memory array, byte by byte, through a data register and a cache register. The cache register is closest to I/O control circuits and acts as a data buffer for the I/O data; the data register is closest to the memory array and acts as a data buffer for NAND Flash memory array operation. The NAND Flash memory array is programmed and read using page-based operations and is erased using block-based operations. During normal page operations, the data and cache registers act as a single register. During cache operations the data and cache registers operate independently to increase data throughput. The status register reports the status of LUN operation.

3.1.1 Memory Organisation

Figure 3.2 shows an example of a Target memory organization. In this case, there are two logical units where each logical unit has two planes. A device contains one or more targets. A target is controlled by one CE signal. A target is organized into two logical units(LUNS)[5]

.

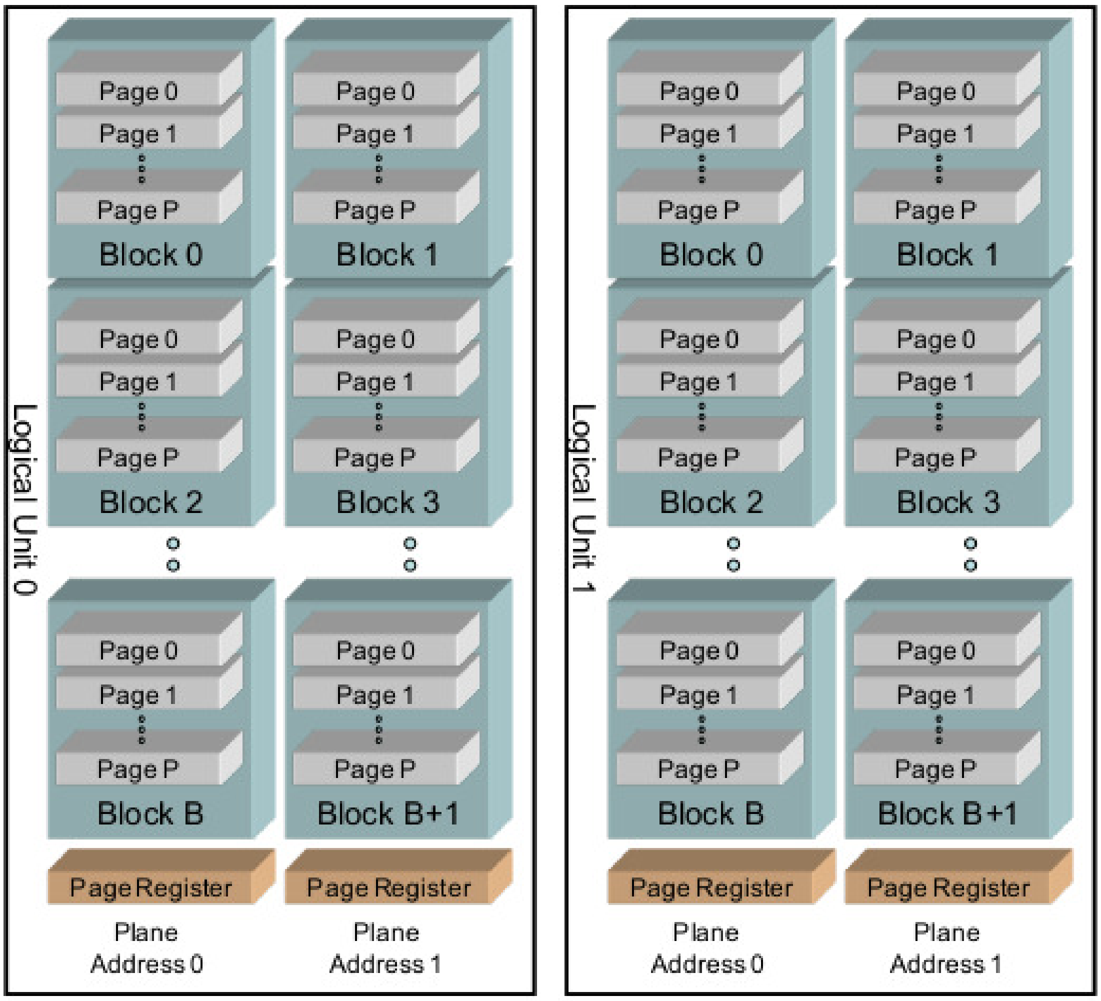


Figure 3.2: Memory organization[5]

Specifically, separate LUNs may operate on arbitrary command sequences in parallel. For example, it is permissible to start a Page Program operation on LUN 0 and then prior to the operation's completion to start a Read command on LUN 1.A LUN contains at least one page register and a Flash array. The number of page registers is dependent on the number of multi-plane operations supported for that LUN. 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 number of user data bytes per page, not including the spare data area, shall be a power of two. The number of pages per block shall be a multiple of 32.There are two mechanisms to achieve parallelism within this architecture. 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.

3.1.2 Array Organisation

An example for a 16Gb array organization per LUN is represented in figure 3.3. A LUN consists of 2 planes, an odd plane and an even plane. Each plane has 2048 blocks and each block has 128 pages.Each page has 4314B where 218B are spare columns used for ECC and other software functions. Hence a total of 17,248Mb of memory[6].

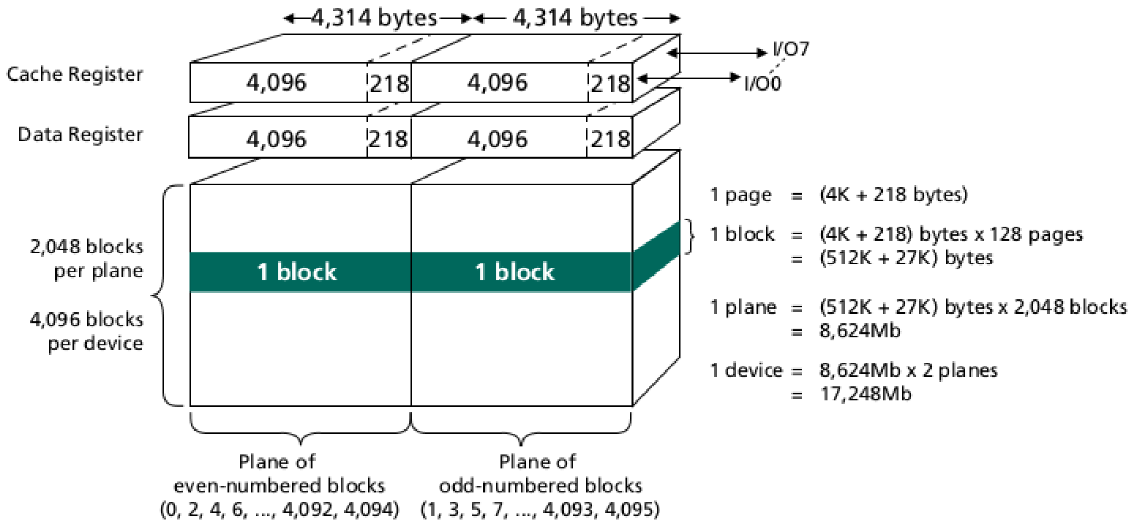

4,314 bytes
4,314 bytes
Cache Register
4,096
218
4,096
218
I/O7
I/O0
Data Register
4,096
218
4,096
218
2,048 blocks per plane
4,096 blocks per device
1 block
1 block
1 page = (4K + 218 bytes)
1 block = (4K + 218) bytes x 128 pages
= (512K + 27K) bytes
1 plane = (512K + 27K) bytes x 2,048 blocks
= 8,624Mb
1 device = 8,624Mb x 2 planes
= 17,248Mb
Plane of even-numbered blocks (0, 2, 4, 6, ..., 4,092, 4,094)
Plane of odd-numbered blocks (1, 3, 5, 7, ..., 4,093, 4,095)

Figure 3.3: Array organization per LUN for 16Gb[6]

3.1.3 Device Organization

The device organization that is targeted is as shown in figure 3.4. The target has 2 LUNS and 2 chips,to be controlled by a single Nand Flash Controller[4].

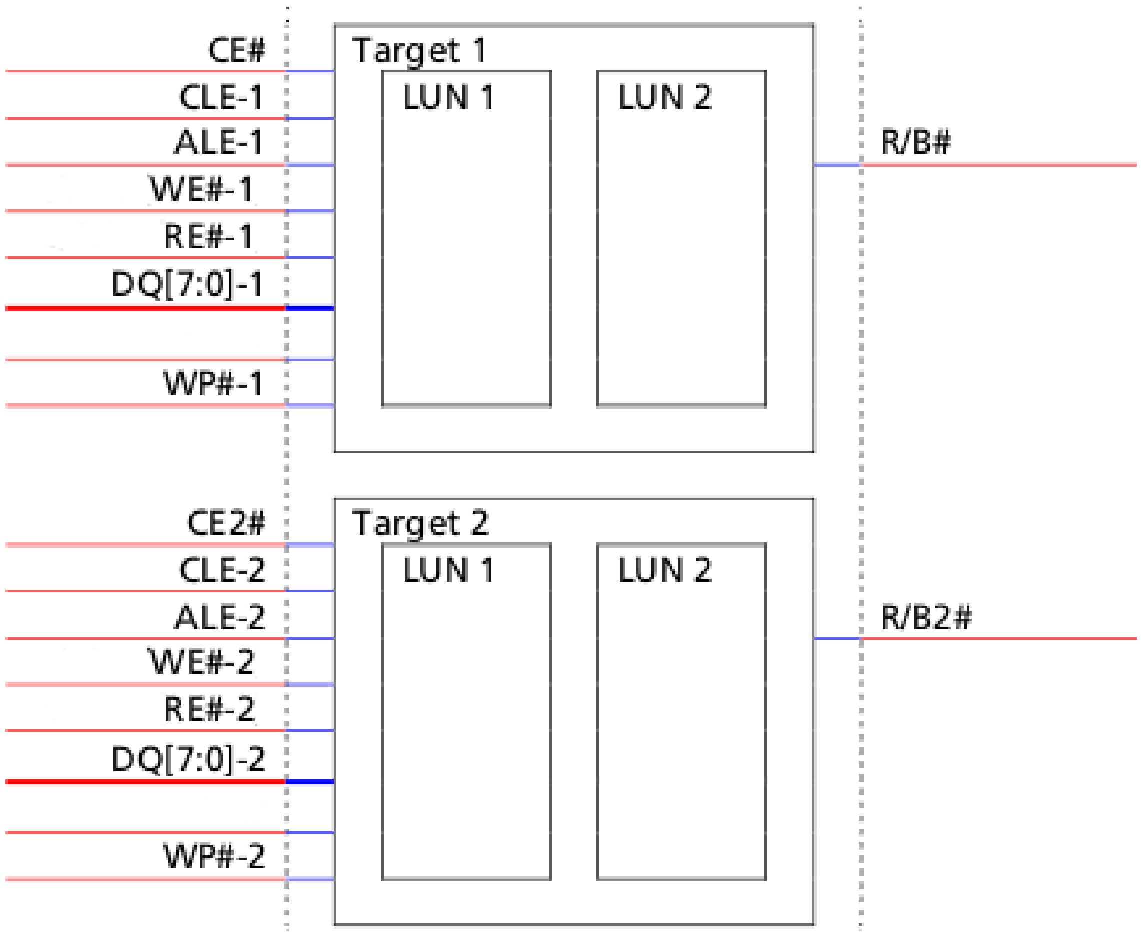


Figure 3.4: Device organization[4]

The asynchronous nature and the block memory structure of NAND Flash devices support combining NAND Flash devices across chip enables in a relatively straightforward process.

Shorting CE's together is a common practice when combining NAND Flash blocks. By not shorting CE's together, as shown in figure 3.5 the design retains the flexibility to communicate with only one NAND device at a time[7].

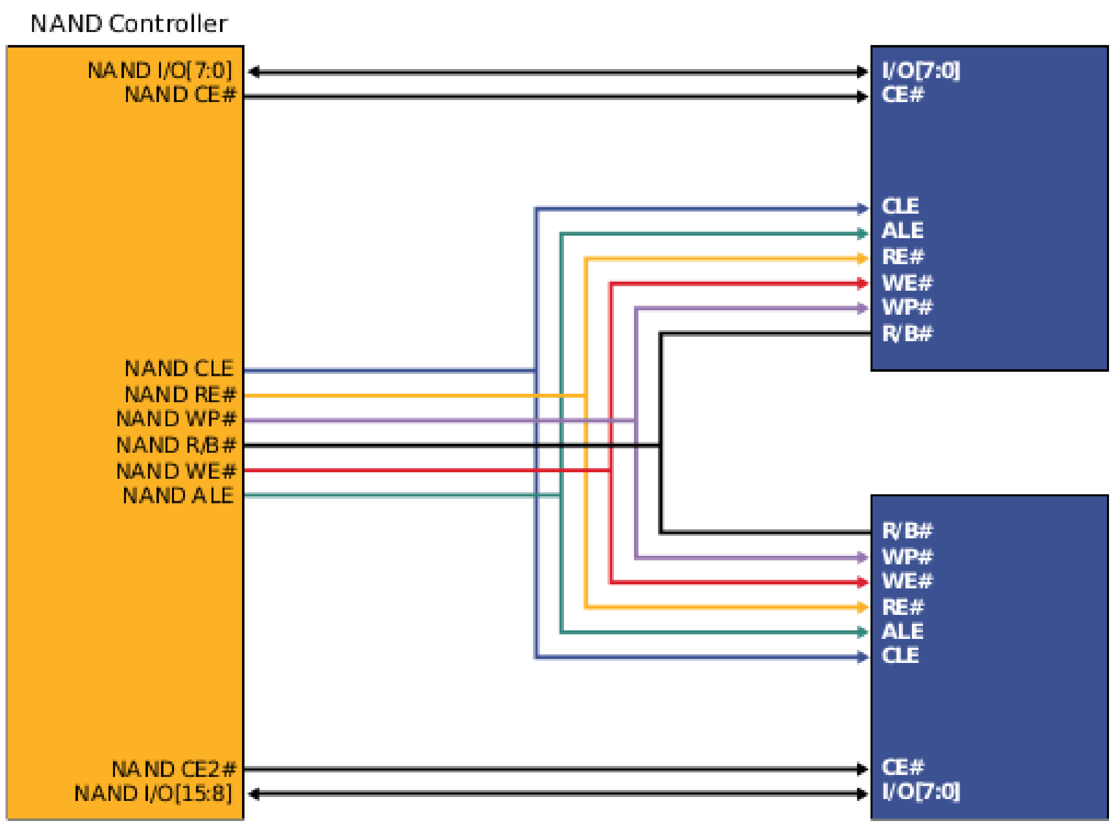


Figure 3.5: Nand Flash blocks combined across CE's[7]

3.1.4 Page storage methods

The two common methods for storing data and spare information in the same page are shown in Figure 3.6. The first method in figure shows a data area of 512 bytes plus the 16-byte spare area directly adjacent to it; 528 bytes for the combined areas. A 2112-byte page can contain four of these 528-byte elements. The second implementation as in figure involves storing the data and spare information separately. The four 512-byte data areas are stored first,and their corresponding 16-byte spare areas follow, in order, at the end of the page. The ECC data is obtained for every 512B of data. This ECC data is to be then stored in the page. If the first method is used the ECC obtained at the end of every 512B is written continuously in the spare area.On the other hand if the second method is employed the column address has to changed after every 512B to store ECC data or a separate register be kept for the ECC data to be written at the end of complete page write[8].

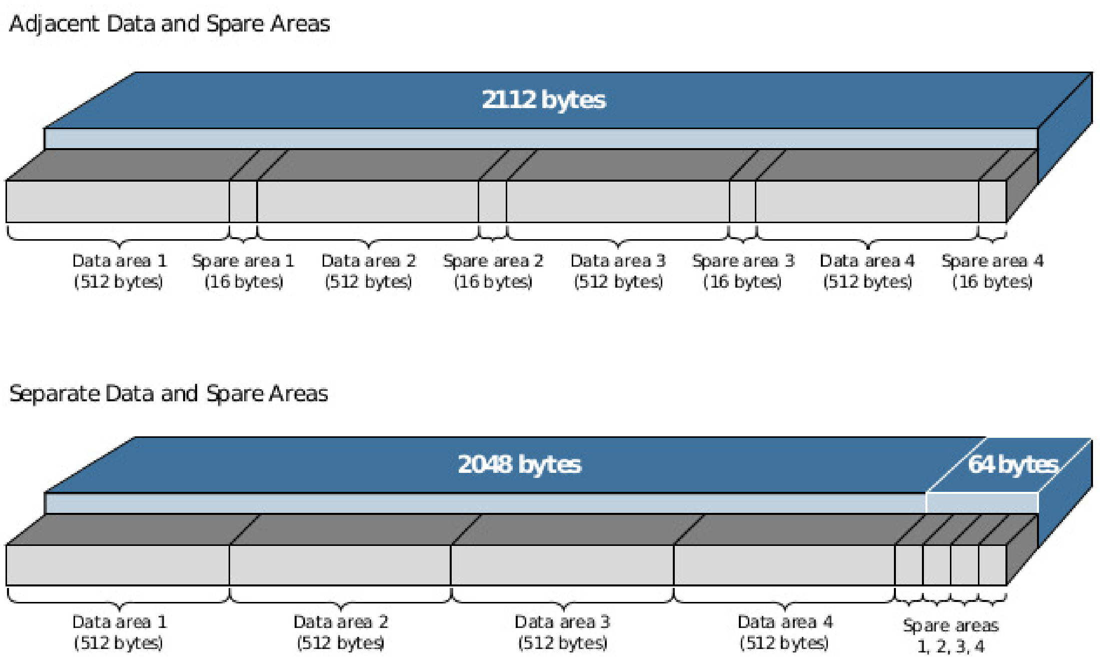


Figure 3.6: Page storage methods[8].

3.2 Addressing

There are two address types used: the column address and the row address. The column address is used to access bytes or words within a page, i.e. the column address is the byte/word offset into the page.The row address is used to address pages, blocks, and LUNs.

When both the column and row addresses are required to be issued, the column address is always issued first in one or more 8-bit address cycles. The row addresses follow in one or more 8-bit address cycles. There are some functions that may require only row addresses, like Block Erase. In this case the column addresses are not issued. For both column and row addresses the first address cycle always contains the least significant address bits and the last address cycle always contains the most significant address bits[5]. If there are bits in the most significant cycles of the column and row addresses that are not used then they are required to be cleared to zero.The row address layout is shown in figure 3.7.

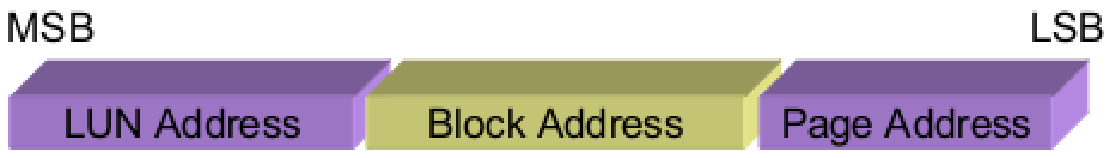


Figure 3.7: Row address layout[5].

3.2.1 Multi Plane Addressing

The multi-plane address comprises the lowest order bits of the block address as shown in figure 3.8.

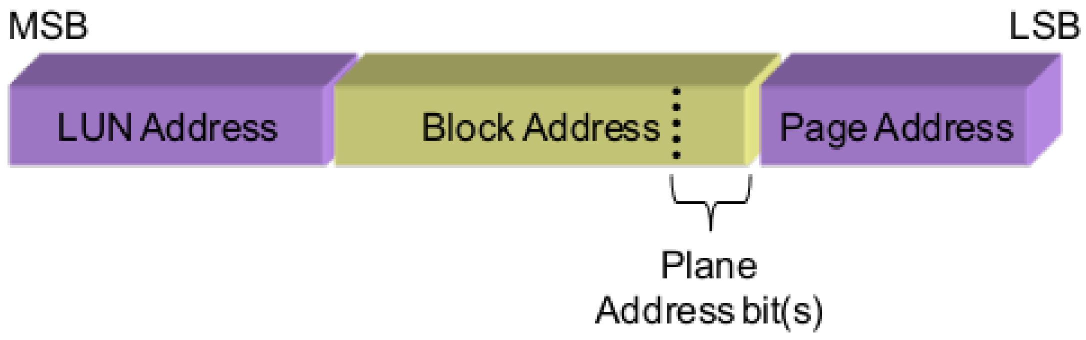


Figure 3.8: Plane Address bits location[5].

The following restrictions apply to the multi-plane address when executing a multi-plane command sequence on a particular LUN[5]:

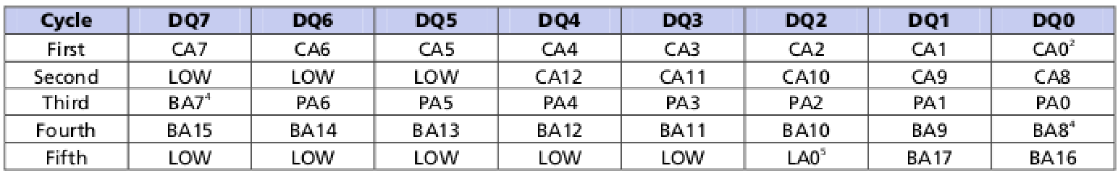
- The plane address bit(s) shall be distinct from any other multi-plane operation in the multi-plane command sequence.
- The page address shall be the same as any other multi-plane operations in the multi-plane command sequence.

There are also plane memory block address restrictions. The specific cases of this are:

- There is no restriction if all the block address bits are different between two plane addresses.
- There is complete restriction if all the block address bits (other than the plane address bits) are the same between two plane addresses. As an example, this means there can be no multi plane operation between block 0 of plane 0 and block 0 of plane 1.

3.2.2 Array Addressing

Addressing is carried out in 5 address cycles through the Data(x8) bus. Some commands require less than 5 address cycles. When an address is being sent, the ALE and WE is to be enabled. The address break up is shown in figure 3.9. The first two cycles are dedicated for the Column Address(CA) and the remaining three cycles for row address. The row address includes Page Address(PA) , Block Address(BA) and LUN Address(LA) bits. As observed in figure 3.8 the plane address in encapsulated in the LSB of the Block Address bits.



Cycle	DQ7	DQ6	DQ5	DQ4	DQ3	DQ2	DQ1	DQ0
First	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0[2]
Second	LOW	LOW	LOW	CA12	CA11	CA10	CA9	CA8
Third	BA7[4]	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8[4]
Fifth	LOW	LOW	LOW	LOW	LOW	LA0[5]	BA17	BA16

Figure 3.9: Address cycle break up[4].

3.3 ONFi command descriptions

The command sets of ONFi 4.0 standard that are used in the design of the Nand Flash Controller, are given in table 3.2 along with the number of address cycles that each command that needs and whether input data cycles exists or not[4]. For example to execute the PROGRAM PAGE operation, a command of 80h needs to be sent first followed by 5 address cycles as discussed previously. This is followed by the input data that needs to be programmed into the Nand Flash memory. The command sequence is ended by sending the command 10h. Each of the operations are discussed briefly in the following section.

Table 3.2: Table of command sets.

Command	**Command cycle1**	**Address cycles**	**Data Input cycles**	**Command cycle2**
RESET	FFh	0	-	-
SET FEATURES	EFh	1	4	-
READ STATUS	70h	0	-	-
SELECT LUN WITH STATUS	78h	3	-	-
SELECT CACHE REGISTER	06h	5	-	E0h
READ MODE	00h	0	-	-
READ PAGE	00h	5	-	30h
READ PAGE MULTI-PLANE	00h	5	-	32h
PROGRAM PAGE	80h	5	YES	10h
PROGRAM PAGE MULTI-PLANE	80h	5	YES	11h
PROGRAM PAGE CACHE	80h	5	YES	15h
ERASE BLOCK	60h	3	-	D0h
ERASE BLOCK MULTI-PLANE	60h	3	-	D1h

3.3.1 RESET operation

The RESET (FFh) command is used to put a target into a known condition and to abort command sequences in progress. This command is accepted by all LUNs, even when they are busy.All pending single-plane and multi-plane operations are cancelled. If this command is issued while a PROGRAM or ERASE operation is occurring on one or more LUNs, the data may be partially programmed or erased and is invalid. The command register is cleared and ready for the next command. The data register and cache register contents are invalid.This command must be issued as the first command to each target after power-up. The timing diagram is shown in figure 3.10.

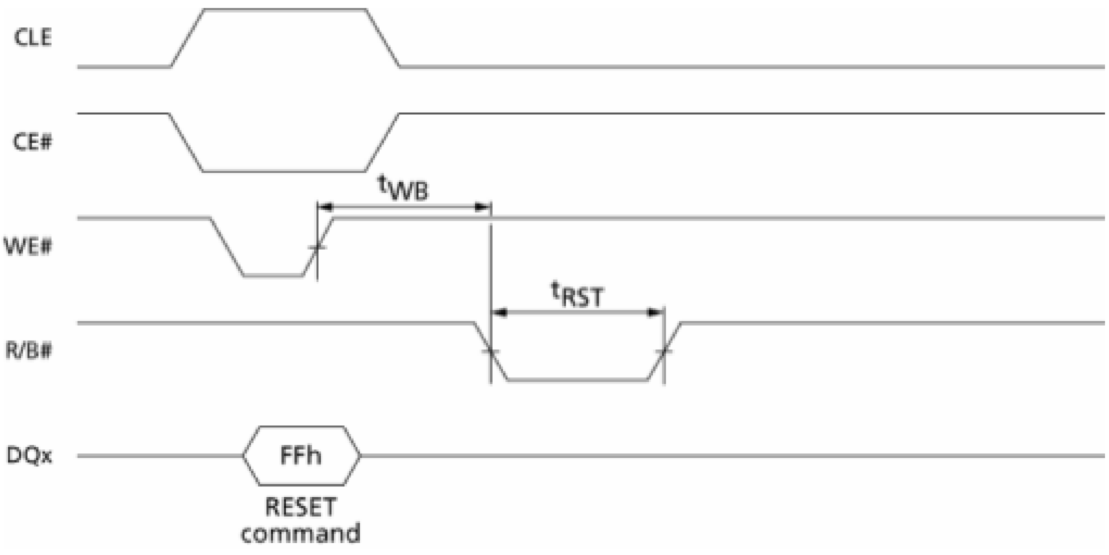


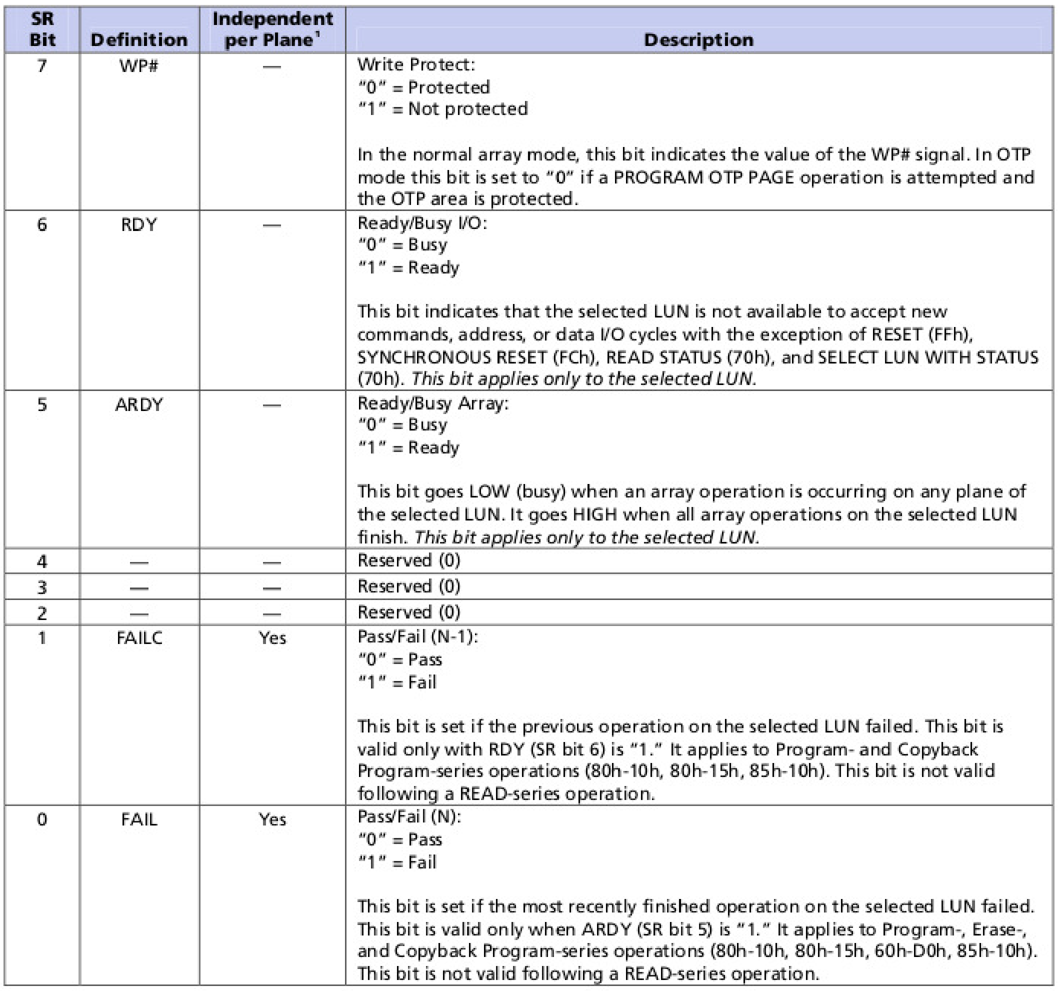
Figure 3.10: RESET cycle[4].

3.3.2 STATUS operation

Each LUN provides its status independently of other LUNs on the same target through its 8-bit status register. Once the READ STATUS (70h) or SELECT LUN WITH STATUS (78h) command is issued, status register output is enabled. The contents of the status register are returned on DQ[7:0] for each data output request.While monitoring the status register to determine when a data transfer from the Flash array to the data register is complete,the NFC must issue the READ MODE (00h) command to disable the status register and enable data output.

The READ STATUS (70h) command returns the status of the most recently selected LUN. To prevent data contention during or following a Multi-LUN operation, the host must enable only one LUN for status output by using the SELECT LUN WITH STATUS (78h) command.The status register definition is shown in figure 3.11.

The timing diagram for READ STATUS operation is shown in figure 3.12 and that of SELECT LUN WITH STATUS command is shown in figure 3.13.



SR Bit	Definition	Independent per Plane[1]	Description
7	WP#	—	Write Protect: "0" = Protected "1" = Not protected In the normal array mode, this bit indicates the value of the WP# signal. In OTP mode this bit is set to "0" if a PROGRAM OTP PAGE operation is attempted and the OTP area is protected.
6	RDY	—	Ready/Busy I/O: "0" = Busy "1" = Ready This bit indicates that the selected LUN is not available to accept new commands, address, or data I/O cycles with the exception of RESET (FFh), SYNCHRONOUS RESET (FCh), READ STATUS (70h), and SELECT LUN WITH STATUS (70h). *This bit applies only to the selected LUN.*
5	ARDY	—	Ready/Busy Array: "0" = Busy "1" = Ready This bit goes LOW (busy) when an array operation is occurring on any plane of the selected LUN. It goes HIGH when all array operations on the selected LUN finish. *This bit applies only to the selected LUN.*
4	—	—	Reserved (0)
3	—	—	Reserved (0)
2	—	—	Reserved (0)
1	FAILC	Yes	Pass/Fail (N-1): "0" = Pass "1" = Fail This bit is set if the previous operation on the selected LUN failed. This bit is valid only with RDY (SR bit 6) is "1." It applies to Program- and Copyback Program-series operations (80h-10h, 80h-15h, 85h-10h). This bit is not valid following a READ-series operation.
0	FAIL	Yes	Pass/Fail (N): "0" = Pass "1" = Fail This bit is set if the most recently finished operation on the selected LUN failed. This bit is valid only when ARDY (SR bit 5) is "1." It applies to Program-, Erase-, and Copyback Program-series operations (80h-10h, 80h-15h, 60h-D0h, 85h-10h). This bit is not valid following a READ-series operation.

Figure 3.11: Status register definition[4].

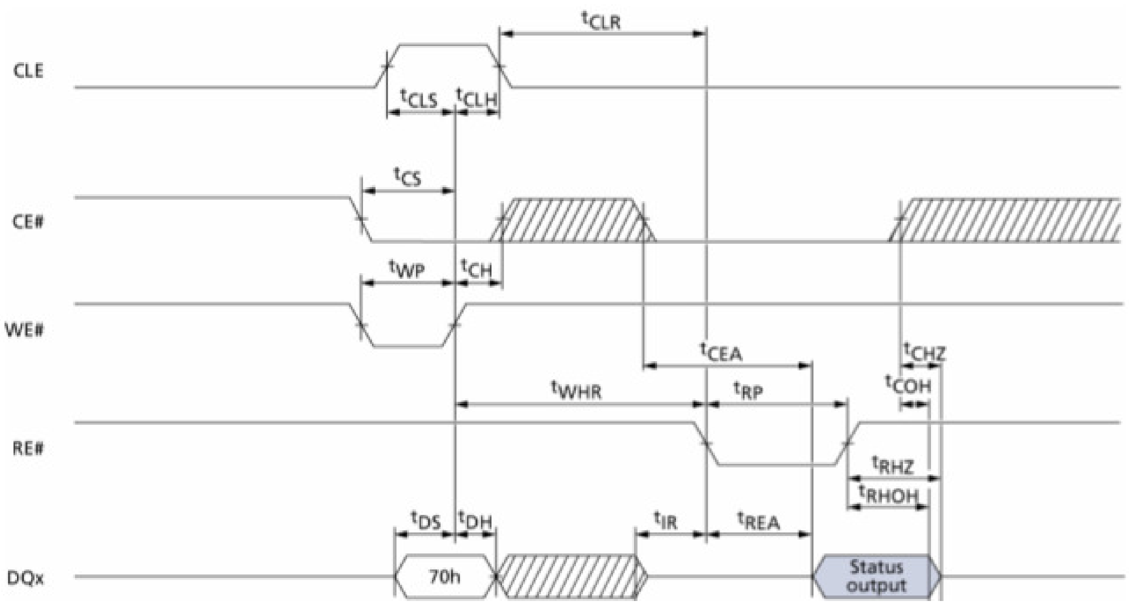


Figure 3.12: READ STATUS cycle[4].

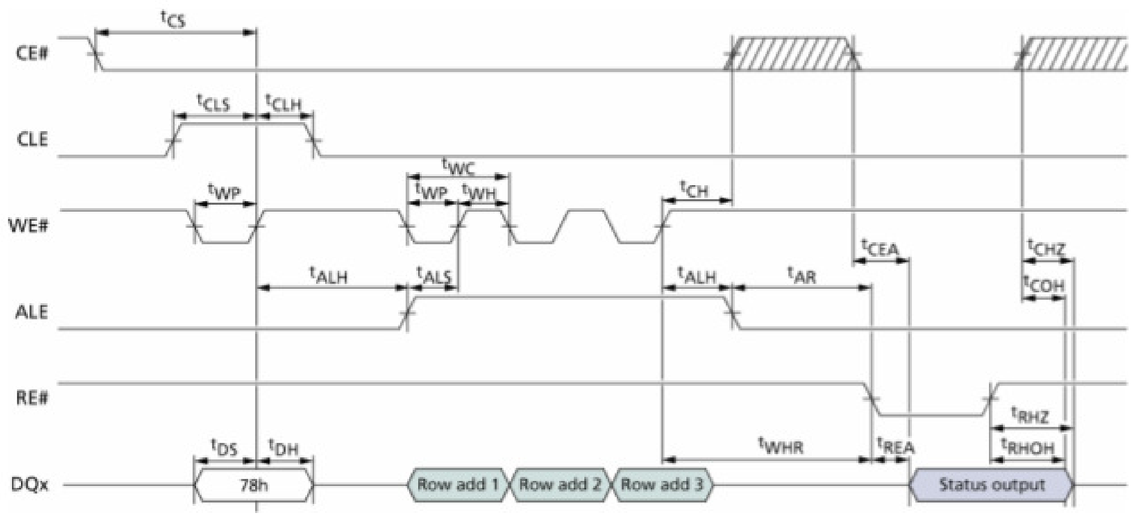


Figure 3.13: SELECT LUN WITH STATUS cycle[4].

3.3.3 SELECT CACHE REGISTER operation

The SELECT CACHE REGISTER command enables data output on the addressed LUN and cache register at the specified column address. This command is accepted by a LUN when it is ready. Following a multi-plane read page operation the SELECT CACHE REGISTER command is used to select which cache register is enabled for data output. After data output is complete on the selected plane it may be issued again to begin data output on another plane.In devices with more than one LUN per target, the SELECT CACHE REGISTER command may be used following a multi-LUN read operation after all of the LUNs on the target are ready LUNs that are not addressed are deselected to avoid bus contention.

3.3.4 READ MODE operation

The READ MODE command disables status output and enables data output for the last selected LUN and cache register after a READ operation has been monitored with a STATUS operation . This command is accepted by the LUN when it is ready. In devices that have more than one LUN per target, during and following multi-LUN operations the SELECT LUN WITH STATUS command must be used to select only one LUN prior to the issue of the READ MODE command. This prevents bus contention. The timing diagram for this operation is shown in figure 3.14.

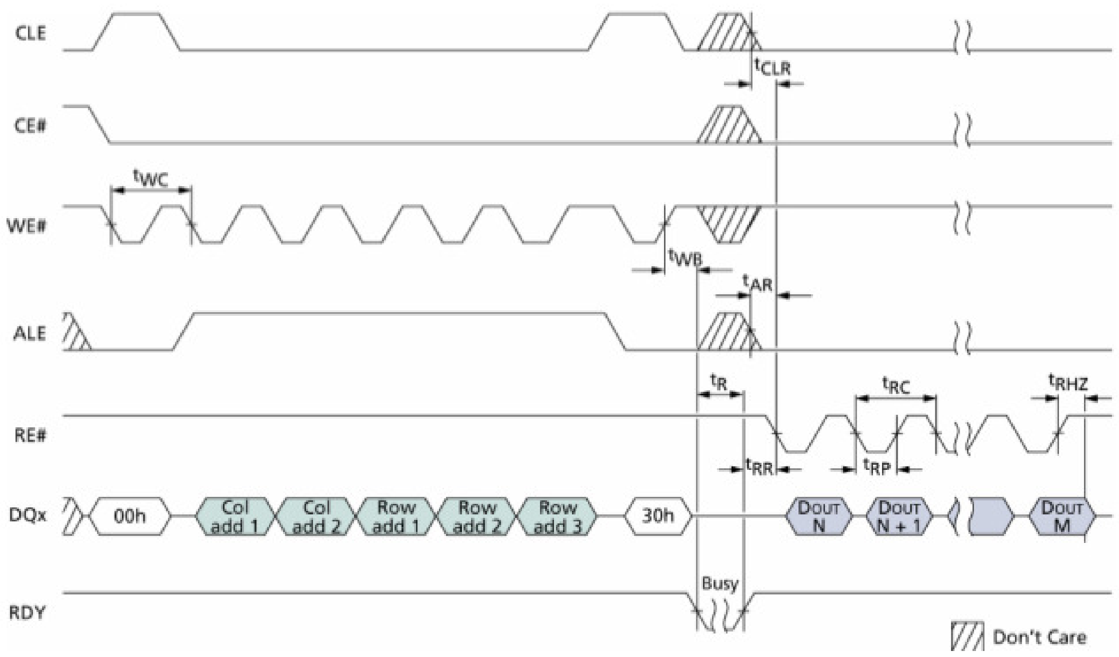


Figure 3.14: READ MODE cycle[4].

3.3.5 READ PAGE operation

The READ PAGE (00h-30h) command copies a page from the NAND Flash array to its respective cache register and enables data output.This command is accepted by the LUN when it is ready.To determine the progress of the data transfer, the host may monitor the target through status operations.When the LUN is ready the host disables status output and enables data output by issuing the READ MODE command.
The READ PAGE command is used as the final command of a multi-plane read operation. It is preceded by one or more READ PAGE MULTI-PLANE commands. Data is transferred from the NAND Flash array for all of the addressed planes to their respective cache registers.

3.3.6 READ PAGE MULTI PLANE operation

The READ PAGE MULTI-PLANE command queues a plane to transfer data from the NAND array to its cache register. This command can be issued one or more times. Each time a new plane address is specified that plane is also queued for data transfer. To select the final plane and to begin the read operation for all previously queued planes, issue the READ PAGE command. All queued planes will transfer data from the NAND array to their cache registers.

3.3.7 PROGRAM PAGE operation

The PROGRAM PAGE command allows the host to input data to a cache register and moves the data from the cache register to the specified block and page address in the array of the selected LUN.The PROGRAM PAGE command is used as the final command of a multi-plane program operation. It is preceded by one or more PROGRAM PAGE MULTI-PLANE commands. Data is transferred from the cache registers for all of the addressed planes to the NAND array. The host should check the status of the operation by using the status operations.The timing diagram for this operation is shown in figure 3.15.

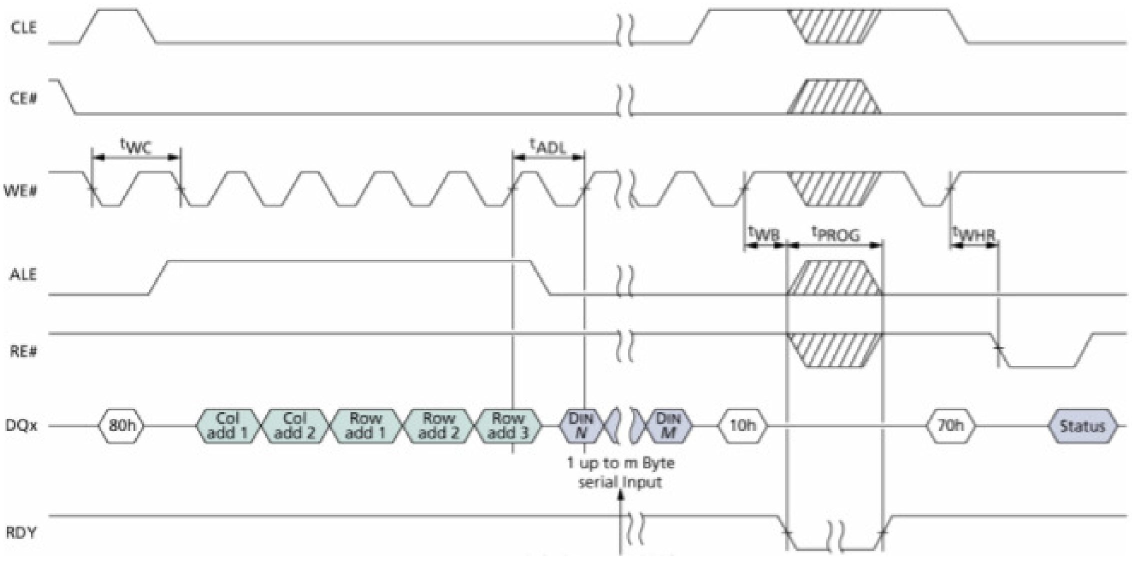


Figure 3.15: PROGRAM PAGE cycle[4].

3.3.8 PROGRAM PAGE CACHE operation

The PROGRAM PAGE CACHE command allows the host to input data to a cache register, copies the data from the cache register to the data register, and then moves the data register contents to the specified block and page address in the array of the selected LUN. After the data is copied to the data register the cache register is available for additional PROGRAM PAGE CACHE or PROGRAM PAGE commands.The PROGRAM PAGE CACHE command is used as the final command of a multi-plane program cache operation. It is preceded by one or more PROGRAM PAGE MULTI-PLANE commands. Data is transferred from the cache registers for all of the addressed planes to the corresponding data registers and then moved to the NAND array. The host should check the status of the operation by using the status operations.

3.3.9 PROGRAM PAGE MULTI-PLANE operation

The PROGRAM PAGE MULTI-PLANE command allows the host to input data to the addressed plane's cache register and queue the cache register to be moved to the NAND array. This command can be issued one or more times. Each time a new plane address is specified that plane is also queued for data transfer. To input data for the final plane and to begin the program operation for all previously queued planes, issue either the PROGRAM PAGE command or the PROGRAM PAGE CACHE command. All of the queued planes will move the data to the NAND array.

3.3.10 ERASE BLOCK operation

The ERASE BLOCK command erases the specified block in the NAND array.The ERASE BLOCK command is used as the final command of a multi-plane erase operation. It is preceded by one or more ERASE BLOCK MULTI-PLANE commands.The timing diagram for this operation is shown in figure 3.16.

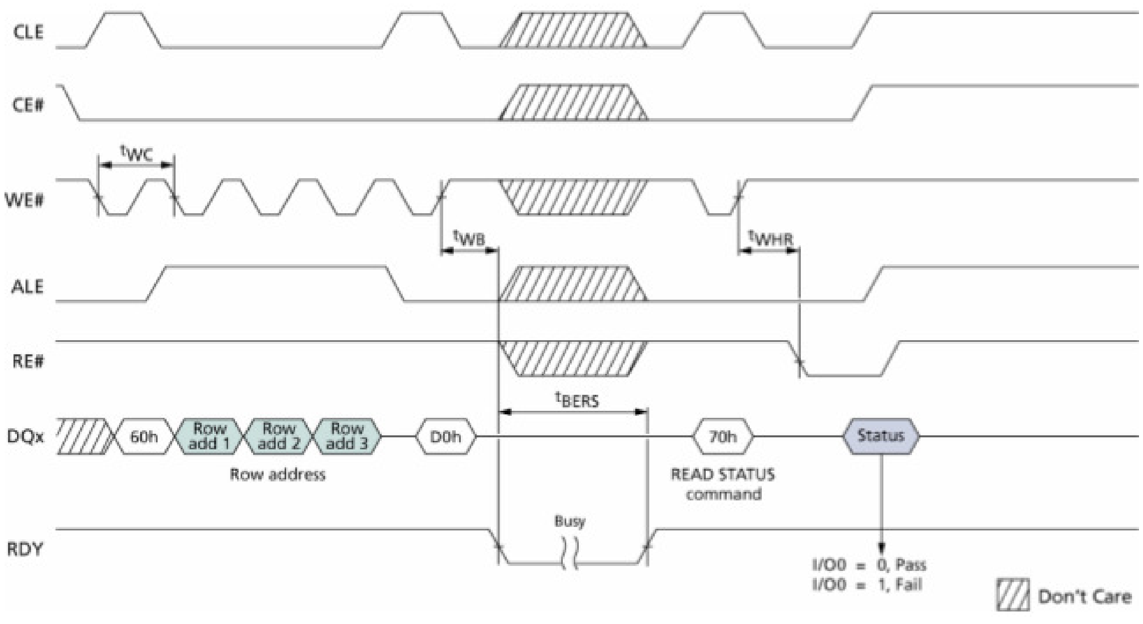


Figure 3.16: BLOCK ERASE cycle[4].

3.3.11 ERASE BLOCK MULTI-PLANE operation

The ERASE BLOCK MULTI-PLANE command queues a block in the specified plane to be erased in the NAND array. This command can be issued one or more times. Each time a new plane address is specified that plane is also queued for a block to be erased. To specify the final block to be erased and to begin the erase operation for all previously queued planes, issue the ERASE BLOCK command .

CHAPTER 4

Interleaving Commands

The most efficient approach for increasing NAND Flash READ, PROGRAM, and ERASE performance with minimal effort is to implement the multi plane commands. Multi plane commands can be used for interleaved die operations to further increase performance in NAND Flash devices that support these commands[12].

4.1 MULTI-PLANE Interleaving

Consecutive page operations can be split between adjacent planes in the same LUN. This ensures that the latency in the case of operations including more than pages will be reduced. This means the pages inside a LUN needs to be scrambled so that adjacent pages fall under unique blocks inside the LUN. As an example a write on 2 pages with address 0 in a classical case implies PROGRAM operation on page 0 and page 1 of block 0. In case of multi-plane operations this needs to be translated into a PROGRAM command involving page 0 in block 0 and another page which falls in a block that is present in different plane.

As discussed in section 3.2.1 there are certain restrictions when executing a multi-plane command sequence. In particular two page operations cannot take place on two adjacent blocks, i.e the block address bits other than the plane select bit must be different. The next restriction being that the page address in the multi-plane operation must be exactly the same. Taking this into consideration the above example of the multi-plane command can now be restated as follows. A two page request on an address 0 can be split into operation on page 0 of block 0(plane 0) and page 0 of block 1(plane 1). Owing to this requirement a "Plane Superblock" is created by combining two flash blocks from separate planes in a LUN as represented in figure 4.1.

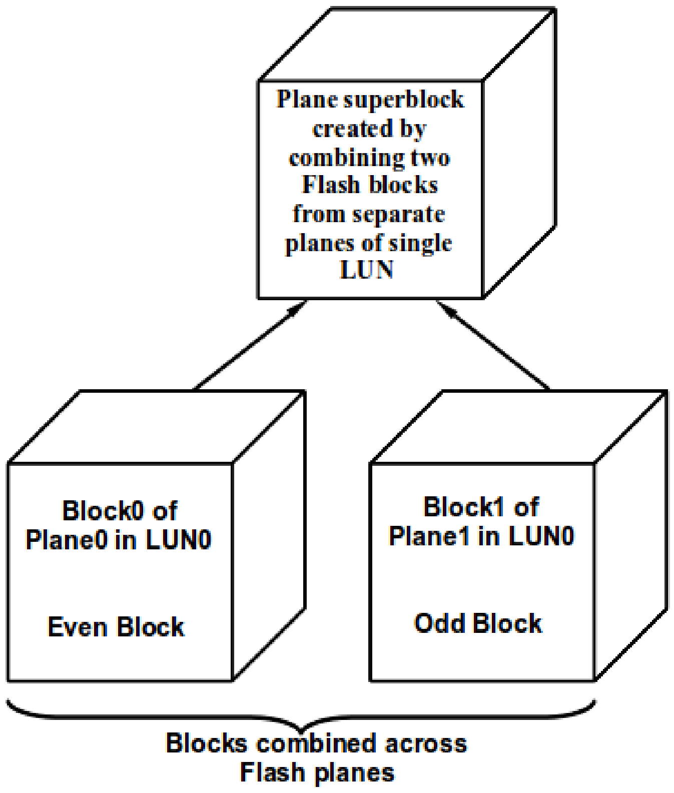

Plane superblock created by combining two Flash blocks from separate planes of single LUN
Block0 of Plane0 in LUN0
Even Block
Block1 of Plane1 in LUN0
Odd Block
Blocks combined across Flash planes

Figure 4.1: Nand Flash Blocks Combined to create a Plane Superblock.

4.2 Interleaving across LUNs

Since two LUNs can operate independently, interleaving can be also be achieved across LUNs in way similar to that of plane interleaving. Any operation that requires more than 2 pages can be efficiently handled by splitting the command into two 2-plane commands and then executing the two commands in two different LUNs controlled by one CE pin. Hence a "LUN Superblock" is created by combining two flash blocks across LUNs as shown in figure 4.2.

4.3 Creating Mega Block

The "Plane Superblock" and "LUN Superblock" can be combined in a target with minimum 2 LUNs and 2 planes per LUN to obtain a "Mega Block". This block is helpful in interleaving commands across planes and LUNs for efficient data transfer. The idea of "Mega Block" is shown in figure 4.3. Hence four consecutive logical pages are now being present as four consecutive pages in a Mega block. To achieve this, the address

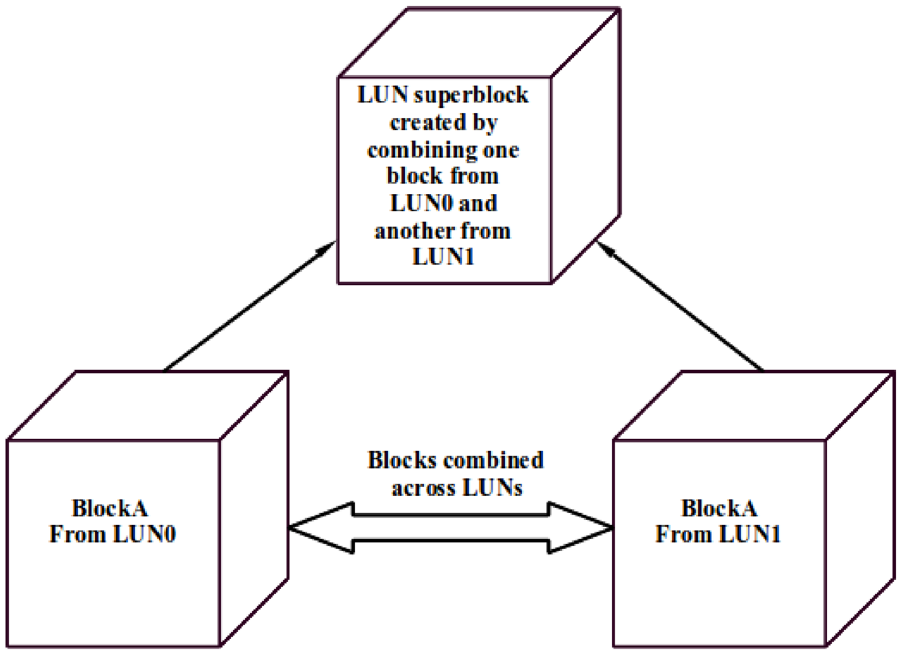

LUN superblock created by combining one block from LUN0 and another from LUN1
Blocks combined across LUNs
BlockA From LUN0
BlockA From LUN1

Figure 4.2: Nand Flash Blocks Combined to create a LUN Superblock.

needs to be scrambled in a particular fashion. So the logical address is mapped onto a function to obtain the physical address on the target. This is further discussed in section 4.3.1.

4.3.1 Logical to Physical Address Mapping

To understand the logical to physical address mapping, in order to arrive at a Mega Block we shall take a specific example. Consider a target Nand Flash with two LUNs, two planes per LUN, four blocks per plane and four pages per block. So there are two bits for page addressing, three bits for block addressing(along with one bit for plane select) and one bit for LUN addressing. As seen in figure 3.2 the blocks are addressed adjacently inside a LUN and the pages are addressed consecutively inside a block. The "physical addressing" of such a Flash memory is shown in table 4.1.

As discussed in 4.3, to obtain a Mega Block we need to scramble the logical address across LUNs. A simple way is to consider block "B" and block "B+3" in both LUNs as constituting a Mega Block. Similarly block "B+1" and block "B+2" across LUNs as constituting another Mega Block. This satisfies all the restrictions that are imposed in multi-plane operations. The logical address table for this kind of addressing is shown in table 4.2.

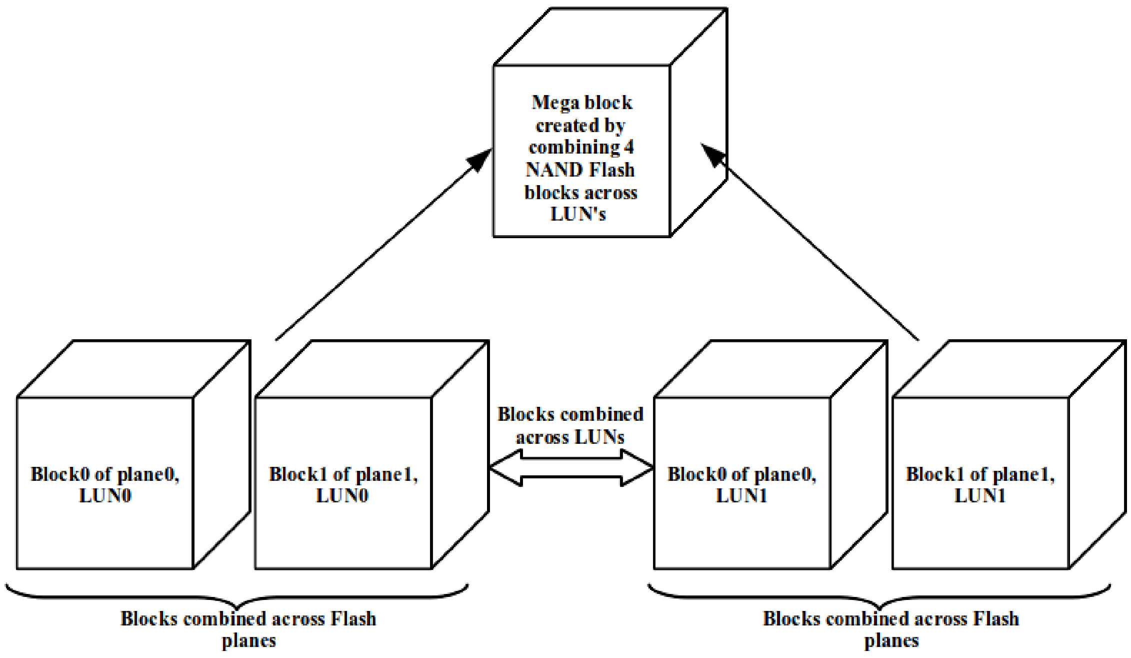

Mega block created by combining 4 NAND Flash blocks across LUN's
Block0 of plane0, LUN0
Block1 of plane1, LUN0
Blocks combined across LUNs
Block0 of plane0, LUN1
Block1 of plane1, LUN1
Blocks combined across Flash planes
Blocks combined across Flash planes

Figure 4.3: Nand Flash Blocks Combined to create a Mega Block.

Table 4.1: Physical address table for a target with 2 LUNs, 2 planes per LUN, 4 blocks per plane and 4 pages per block.

LUN 0		**LUN 1**	
Plane 0	**Plane 1**	**Plane 0**	**Plane 1**
000000	000100	100000	100100
000001	000101	100001	100101
000010	000110	100010	100110
000011	000111	100011	100111
001000	001100	101000	101100
001001	001101	101001	101101
001010	001110	101010	101110
001011	001111	101011	101111
010000	010100	110000	110100
010001	010101	110001	110101
010010	010110	110010	110110
010011	010111	110011	110111
011000	011100	111000	111100
011001	011101	111001	111101
011010	011110	111010	111110
011011	011111	111011	111111

Table 4.2: Logical address table for a target with 2 LUNs, 2 planes per LUN, 4 blocks per plane and 4 pages per block.

LUN 0		**LUN 1**	
Plane 0	**Plane 1**	**Plane 0**	**Plane 1**
000000	010001	000010	010011
000100	010101	000110	010111
001000	011001	001010	011011
001100	011101	001110	011111
010000	000001	010010	000011
010100	000101	010110	000111
011000	001001	011010	001011
011100	001101	011110	001111
100000	110001	100010	110011
100100	110101	100110	110111
101000	111001	101010	111011
101100	111101	101110	111111
110000	100001	110010	100011
110100	100101	110110	100111
111000	101001	111010	101011
111100	101101	111110	101111

The logical to physical address mapping is thus obtained from the table 4.1 and 4.2 as below.

- Page Address Map:

$$PhysicalAddr[0] = LogicalAddr[2]$$
$$PhysicalAddr[1] = LogicalAddr[3]$$

- Block Address Map:

$$PhysicalAddr[2] = LogicalAddr[0]$$
$$PhysicalAddr[3] = LogicalAddr[0] \oplus LogicalAddr[4]$$
$$PhysicalAddr[4] = LogicalAddr[5]$$

- LUN Address Map:

$$PhysicalAddr[5] = LogicalAddr[1]$$

In general if PW represents the width of page address, BW represents width of block address(along with plane select) and LW represents the width of LUN address then the mapping is as follows:

- Page Address Map:

$$PhysicalAddr[PW-1:0] = LogicalAddr[(PW+LW):(LW+1)]$$

- Block Address Map:

$$\begin{gathered} PhysicalAddr[PW] = LogicalAddr[0] \\ PhysicalAddr[PW+1] = LogicalAddr[0] \oplus LogicalAddr[PW+LW+1] \\ PhysicalAddr[(BW+PW-1):(PW+2)] = LogicalAddr[(BW+PW): \\ (PW+LW+2)] \end{gathered}$$

- LUN Address Map:

$$PhysicalAddr[LW+BW+PW-1] = LogicalAddr[1]$$

Each time an address is received for a particular operation , the address translation is done based on this mapping function to perform operation on Mega Block. In the preceding chapters we will discuss in detail about the various operations.

CHAPTER 5

Design and Implementation

In this chapter we will discuss the design of the Nand Flash Controller and its implementation in BSV along with the implementation of a Nand Flash Memory model which is ONFi 4.0 compliant.

5.1 Conception and Design of Nand Flash Controller

The Nand Flash Controller that is developed has to follow the ONFi protocol as discussed in section 3. The Nand Flash Controller talks to the host through the NMVe interface on one side and with the Nand Flash memory through the ONFi on the other. The NVMe interface provides operation requests such as Read, Write and Erase which needs to be translated into ONFi specific commands by the NFC. The data bus on the NVMe side is parameterized for extensibility. The block diagram of the NFC is shown in figure 5.1. It has a Read and Write buffer that takes data to/from the NVMe interface. The Write buffer is needed since the data needs to be serialized to 8 bits at the Nand Flash Memory end and the Read buffer is needed to de-serialise the data from 8 bits to the required data width at the NMVe side. The control logic is a state machine which receives the request from NVMe as input and completes the required operation.

The NVMe specifies the number of pages that needs to be read or written into in a request through a length parameter. This length must then be used to pick the efficient commands from the ONFi command set so as to utilize the concept of Mega Block. For the Erase operation, a Mega block needs to be erased in place of a single block and hence proper command set needs to be picked for this operation. Along with these basic operations of Read , Write and Erase there are other power-on operations that needs to be done by the NFC. The general data flow of the NFC in shown if figure 5.2.

On power-on the NFC needs to wait for the Nand Flash devices connected to be free and once free it must issue the Reset command to the Nand Flash memory. No

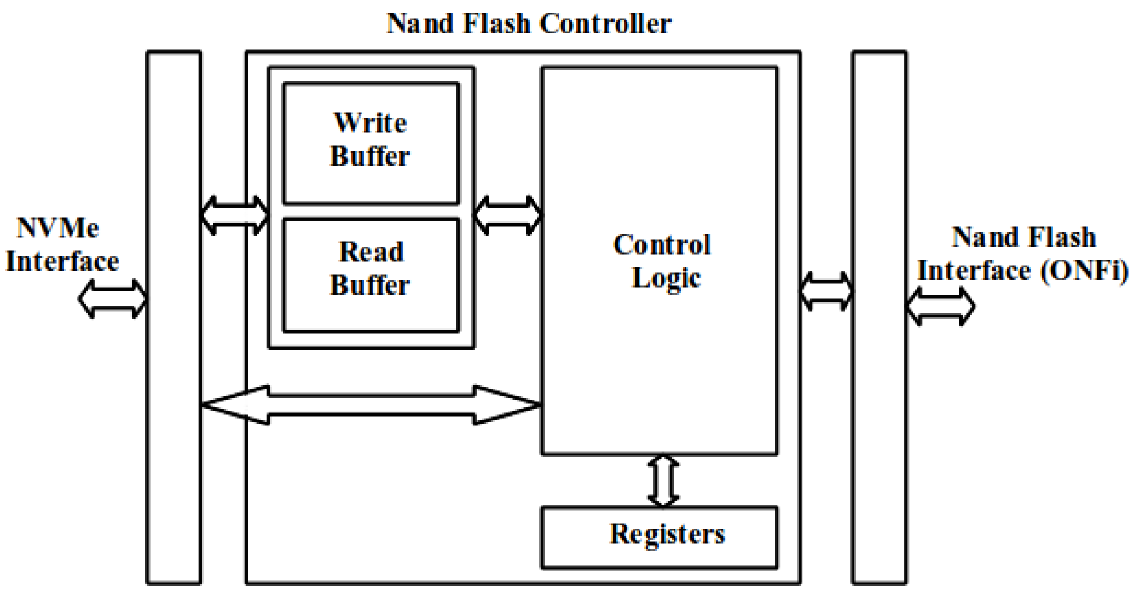


Figure 5.1: Nand Flash Controller Block Diagram.

operation must be done without issuing a Reset command on power-on. Once the Reset is issued the Nand Flash target is monitored till the operation finishes. Once the target is free, certain features such as the operating timing mode of the Nand flash device needs to be set using the SET FEATURE command. Then the Nand Flash memory has to be scanned for bad blocks and a bad block table has to be created. This is discussed in detail in section 5.1.1. Once the bad block table is ready, the controller then asserts its ready/busy pin indicating the host/NVMe that it is free to take operations. Upon an operation request which can be read, write or erase the NFC takes the request completes it and notifies the NVMe.

5.1.1 Bad Block Information

The initial bad blocks that are marked by the flash vendor, could be inadvertently erased and destroyed by a user. To prevent this from occurring, it is necessary to always know where any bad blocks are located. Continually checking for bad block markers during normal use would be very time consuming, so it is highly recommended to initially locate all bad blocks and build a bad block table. This will prevent having the initial bad block markers erased by an unexpected program or erase operation[13]. This can be accomplished by scanning the spare area of all memory blocks and updating a bad block table whenever bad block markers are found. For MLC devices, any block, where

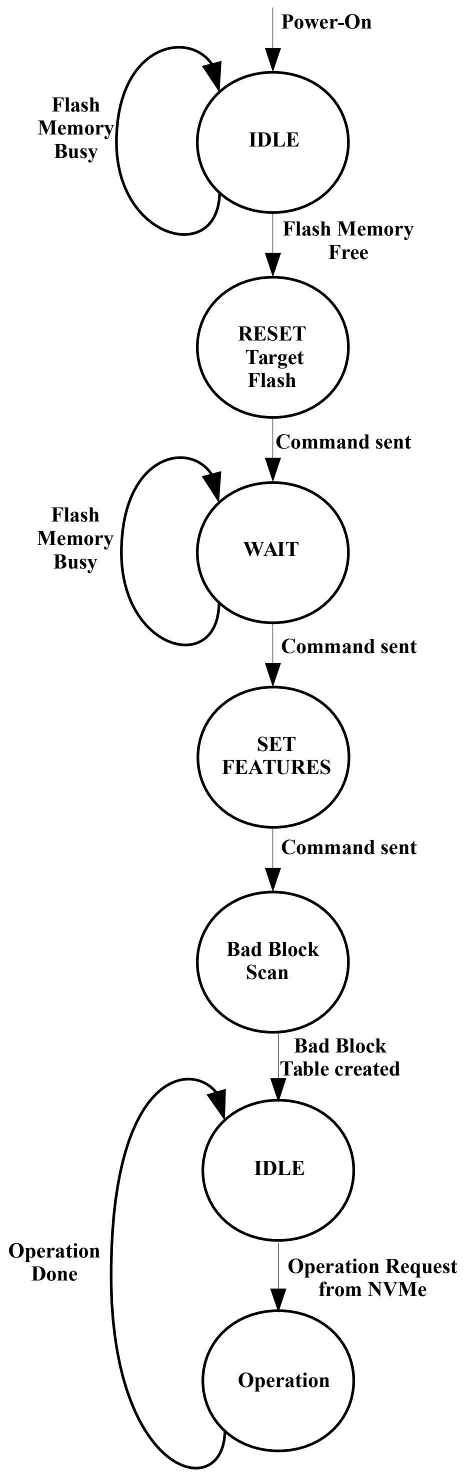


Figure 5.2: General data flow of the NFC.

the 1st byte in the spare area of the last page, does not contain FFh is a bad block[9].

The bad block information must be read before any erase is attempted, because the bad block information is erasable and cannot be recovered once erased. It is highly recommended not to erase the original bad block information. To allow the host to recognize the bad blocks based on the original information, it is recommended to implement the bad block management algorithm shown in figure 5.3.

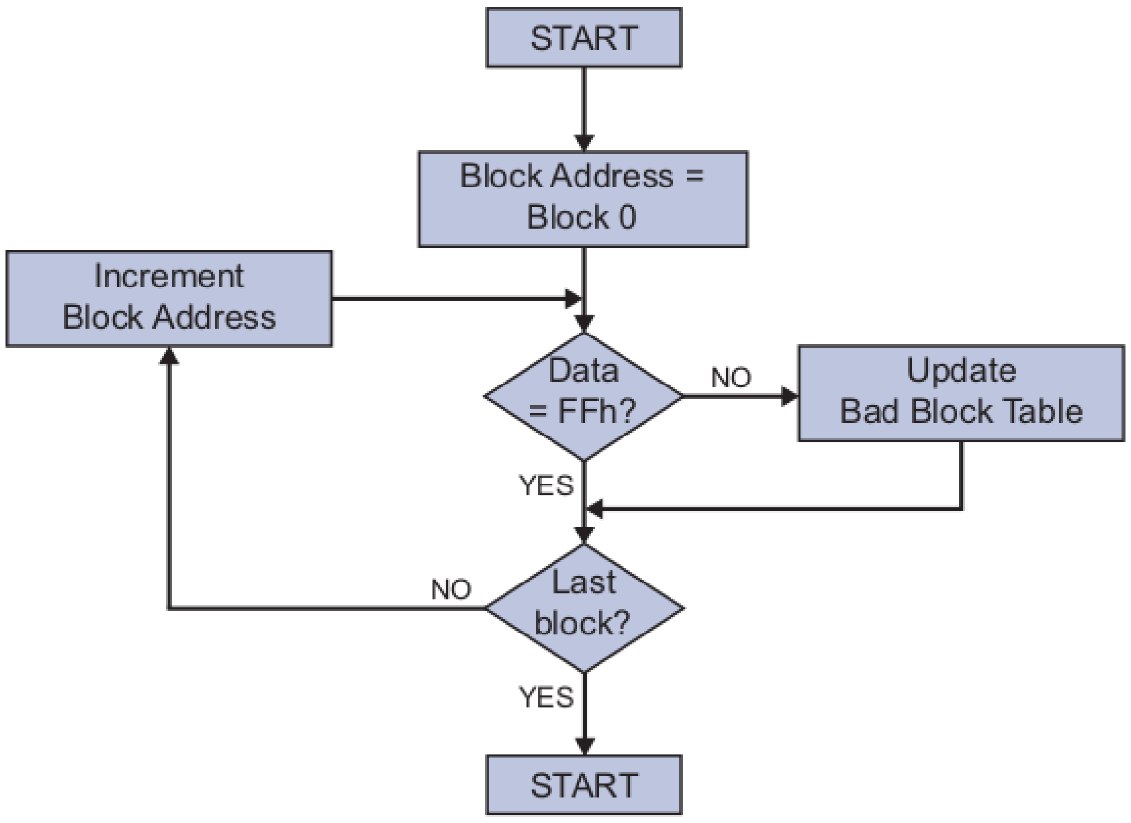


Figure 5.3: Bad Block Management Flow Chart.[9]

5.2 Implementation of Nand Flash Controller

In this section we will discuss the implementation of the Nand Flash Controller by means of detailed state diagrams and code listings. The state machines are implemented using BSV, which provides higher level of abstraction than verilog. A rule in BSV is an atomic block and BSV introduces scheduling of the rules in a module. There must not be any conflict between resources that are shared between rules. State machines are created by combining rules and specifying when to fire a particular rule. Upon achieving no conflict between resource the BSV compiler generates a scheduling logic to obtain the desired functionality.Connections are taken in and out of the module through interfaces. Interfaces in-turn consist of methods. Finally a synthesizable verilog RTL is generated which performs the required function. This RTL generated can be simulated using Modelsim to check the functionality before synthesis. The following sections will discuss the interface definitions that connect the Nand Flash Controller to the outside world and the state machines comprised in the Nand Flash Controller.

5.2.1 Nand Flash Controller Interfaces

NVMe-NFC Interface

The NVMe-NFC interface defines the connection between the Nand Flash Controller and the NVMe. The interface definition is listed in 5.1.

```
interface NandFlashInterface;
    method Action _request_data(UInt#(64) _address, UInt#(11) _length);
    method Action _request_erase(UInt#(64) _address);
    method Bit#(`WDC) _get_data_();
    method Action _write(UInt#(64) _address, Bit#(`WDC) _data, UInt#(11) _length);
    method Action _enable(bit _nand_ce_l);
    method Action _query_bad_block(UInt#(64) _address);
    method bit interrupt_ ();
    method bit busy_();
    method bit write_success_ ();
    method bit write_fail_ ();
    method bit erase_success_ ();
    method bit erase_fail_ ();
endinterface
```

Listing 5.1: NVMe-NFC Interface

The **_enable** method is used to select the channel or enable the NFC. The method **busy_** indicates whether the NFC is ready to receive any requests. The **_request_data** method is used by the NVMe to send a read request. It provides the starting address of the page to be read and the number of pages to be read (payload length) in the form of _length. Once the read request is sent the NFC has to pulse the **interrupt_** method to signal the NVMe that the data is available. The NVMe then calls the **_get_data** method, which returns the data to the NVMe. The NVMe sends a write request through the **_write** method. This passes on the address and the payload length along with the data to the NFC. Once the NFC completes the write it acknowledges the NVMe by either pulsing the **write_success_** or **write_fail_** method. A block erase request is sent through the **_request_erase** method which sends the block address to the NFC. Once erase is completed by NFC it acknowledges either through the **erase_success_** or **erase_fail_** method. The bad block table is accessed by NVMe by sending a request through **_query_bad_block** method. This will have an address and an offset being sent to the NFC. The NFC pulses the **interrupt_** to start sending the bad block table section. The data is sent/received in chunks whose bit-width is set by the parameter WDC (Width of Data Channel).

NFC-Nand Flash memory Interface

The NFC-Nand Flash memory interface must follow the ONFi standard pin configuration. the listing of interface is shown in listing 5.2.

```
interface ONFiInterface;
        method bit onfi_ce0_n_ ();
        method bit onfi_ce1_n_ ();
        method bit onfi_we_n_ ();
        method bit onfi_re_n_ ();
        method bit onfi_wp_n_ ();
        method bit onfi_cle_ ();
        method bit onfi_ale_ ();
        method Action _ready_busy0_n_m ( bit _ready_busy0_l );
        method Action _ready_busy1_n_m ( bit _ready_busy1_l );
         interface Inout#(Bit#(8)) dataio0;
         interface Inout#(Bit#(8)) dataio1;
endinterface
```

Listing 5.2: NFC-Nand Flash Memory Interface

As shown in figure 3.5 the NFC will be controlling two chips. Hence the interface has two methods **onfi_ce0_n_** and **onfi_ce1_n_**. These two methods will enable the target memory. The other methods can be compared directly with the signals in table

3.1. The methods **onfi_we_n_** and **onfi_re_n_** are similar to the WE and RE signals but an active low version. **onfi_cle_** and **onfi_ale_** are equivalent to signals CLE and ALE. The two data I/O bus dataio0 and dataio1, which are defined as two sub interfaces are the DQ bus for each of the chip. Action methods **_ready_busy0_n_m** and **_ready_busy1_n_m** are the R/B pin of the two target chips.

5.2.2 Nand Flash Controller State Machines

The Nand Flash Controller houses that control logic which has a data flow as shown in figure 5.2. This data flow includes smaller state machines such as the RESET state machine, Bad block scan state machine and the operation state machine. The operation state machine has PROGRAM, READ and ERASE state diagrams. Each of these are discussed in the preceding sections.

Before we concentrate on the operation state machines, each of these state machines will be abstracted in way such that, the individual operations as listed in table 3.2 will be shown as a single state. Hence we shall expand on these state machines first.

RESET state machine

The RESET state machine is shown in figure 5.4. Once the Nand Flash Memory is free, the command FFh is sent to both the chips. Then the busy methods of the interface are monitored until the two chips are free again. The state machine then proceeds to the next state as in figure 5.2.

STATUS READ state machine

The status of a LUN can be read through the data I/O pins by issuing the status commands. There are two status commands 78h and 70h. When the status is to be read, the command is sent first , then the RE# is toggled from HIGH to LOW and the status is read from the I/O bits[14]. The state diagram for read status using 78h command is shown in figure 5.5.The address cycles will be absent in case of the 70h command.

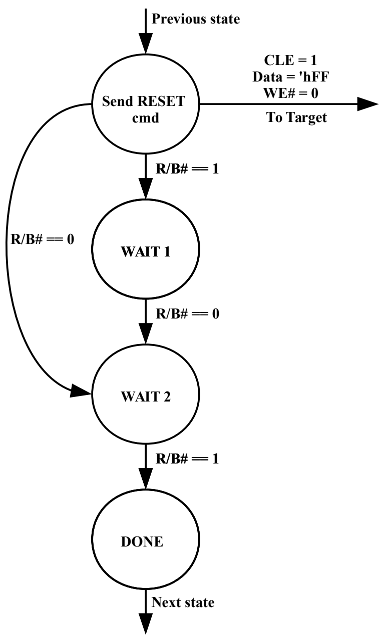


Figure 5.4: RESET state machine.

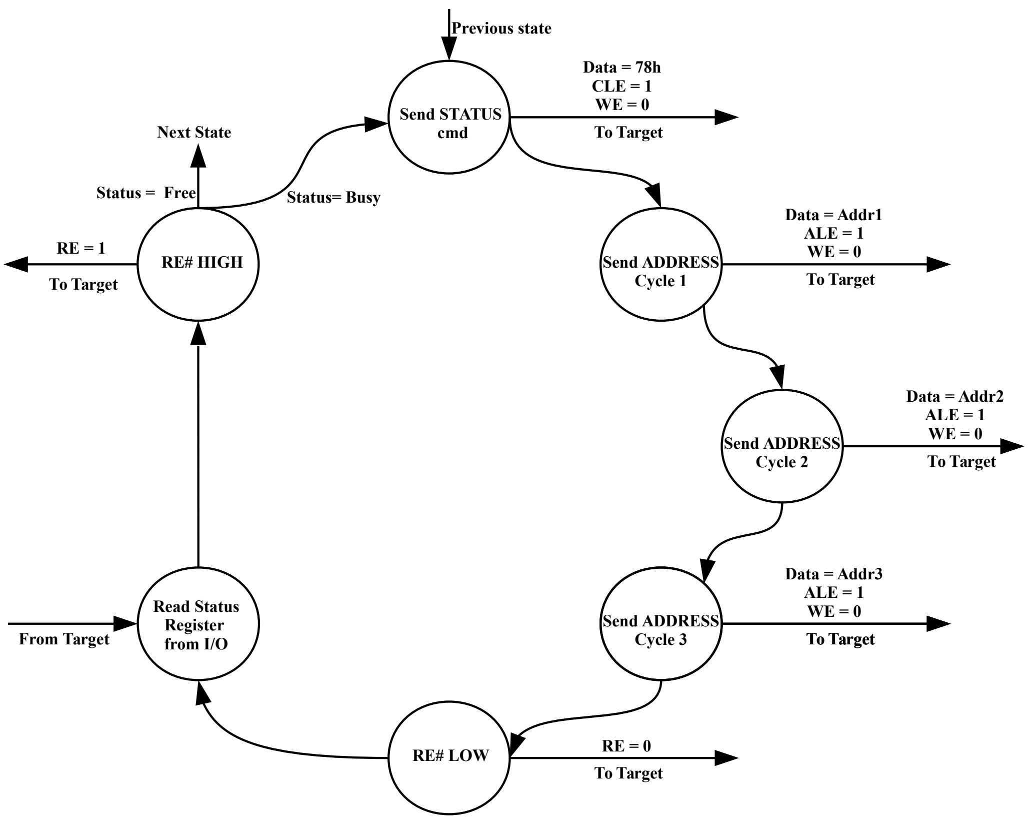


Figure 5.5: READ STATUS state machine.

PROGRAM PAGE series state machine

The operations PROGRAM, PROGRAM PAGE MULTI PLANE, PROGRAM PAGE CACHE all have similar states except for the final command cycle, where each of the operations have a unique command. A PROGRAM series of command can be sequenced first by sending the 80h command, followed by the 5 address cycles. Then data is sent to the target till the column address has reached the final value in the page that is to be programmed while toggling the write enable signal. In case of a full page program, the data sent must be one full page size. In case of an offset, the data sent will be less than a page size. Each page will be divided into sectors of 512B. After every 512B of actual data, the ECC data must be sent so as to store it in the spare area as discussed in figure 3.6. The general state machine for PROGRAM series operation is shown in figure 5.6.

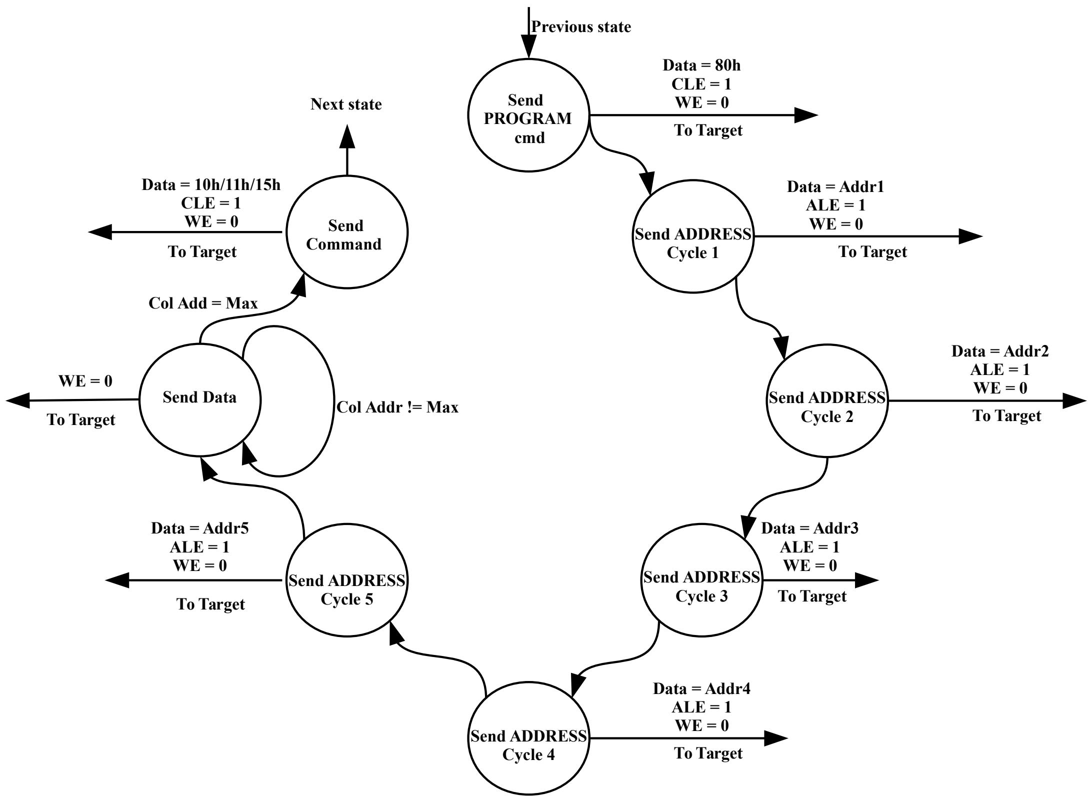


Figure 5.6: PROGRAM PAGE series state machine.

READ PAGE series state machine

The READ PAGE series commands are used to read a page from the target into the corresponding plane registers. The READ PAGE and READ PAGE MULTI PLANE command have similar states except for the final command.This series of command starts with the command 00h followed by the 5 address cycles and ended with the command of 30h or 32h. After this the read enable is toggled and for every toggle a byte of data is received. After every 512B of data , the ECC data is received (since this is how it was programmed in the first place). Toggling read enable until maximum column address will give valid data. The READ PAGE series state machine is shown in figure 5.7. The READ MODE command state machine is simple, in that the previous state being a status read state and the present state being a 00h command issued and toggling of read enable to read the data. The SELECT CACHE REGISTER command is similar to READ MODE except that this need not be preceded by a status operation and can be used to read the data from the cache register.

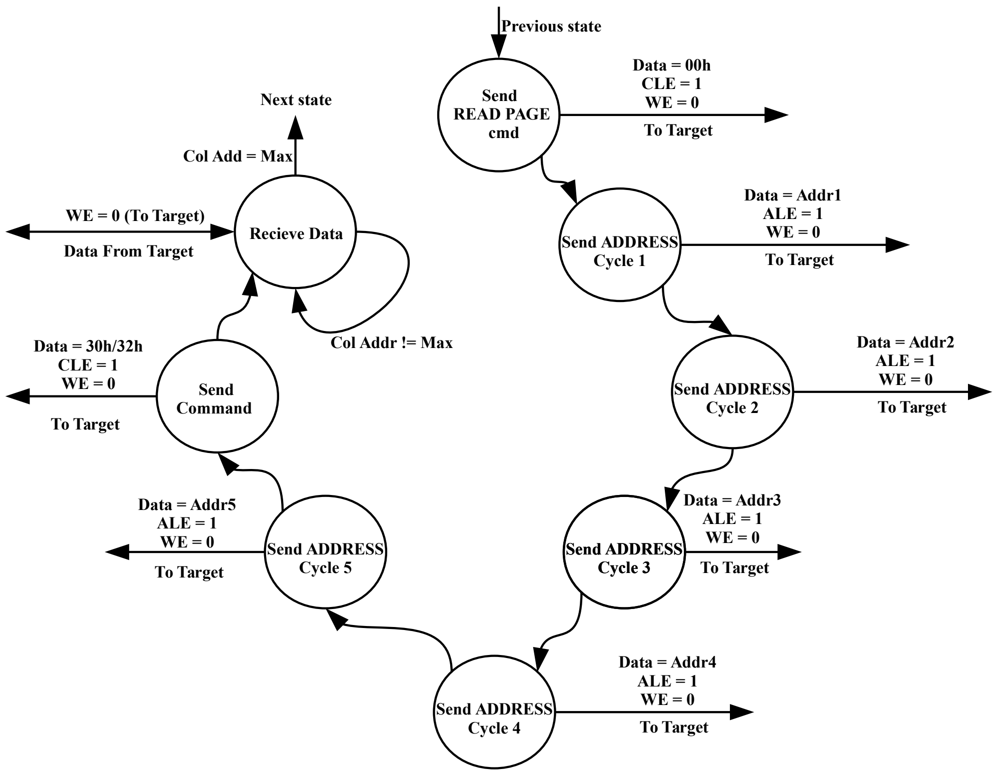


Figure 5.7: READ PAGE series state machine.

ERASE BLOCK series state machine

The block erase operation needs only three address cycles since only the block address needs to be specified. The ERASE BLOCK and ERASE BLOCK MULTI PLANE operations are similar except for the final command. The state machine foe erase series operations is shown in figure 5.8. This series starts with a command of 60h followed by 3 address cycles and then ended with the D0h or the D1h command.

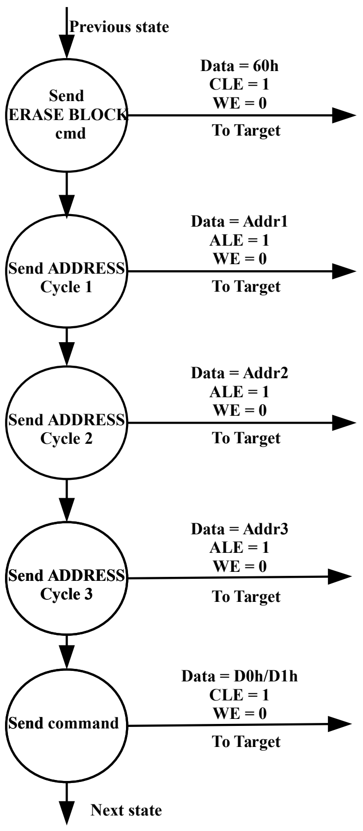


Figure 5.8: ERASE BLOCK series state machine.

The PROGRAM series, ERASE series and READ PAGE series state machines form sub-state machine in larger state machine which is implemented for the write, erase and read operations on the Mega Block. The succeeding sections discuss implementation of each of these state machines in the form of the data flow.

WRITE operation data flow

Program is the highest latency operation in flash memory wen compared to read. Hence a lot of multi-plane and cache operated commands needs to be used. The write operation starts with the write request from the NVMe which sends the start address and the payload length (number of pages). The data will then be buffered by the NFC. The buffer size is one-page size. The data flow from the write request till the completion of request is shown in figure 5.9 and 5.10.

Before a write is done on any LUN its status must be checked for two things. Firstly the LUN must be free and secondly the previous operation that had happened on that LUN was a success or not. In the flow diagram every-time the controls enters node A, the address is mapped so as to obtain the addressing as per Mega block architecture. Once the data is buffered, the data width can be 32,64 or 128 which is parameterized. To send the data to the flash, its has to be split into single bytes starting from the LSB. This is achieved using shift registers. If the length is one, the address is obtained and the status of that LUN is read. Once it is free the PROGRAM PAGE state machine is executed as discussed previously. After this the operation is monitored using status read using READ STATUS state machine. Upon completion, the status bit is checked to see whether the operation was a success or fail. Accordingly the NMVe is sent an acknowledgment in the form of an interrupt and the operation is completed. In case of a two page write, there are two possibilities. Either the two pages fall on the same LUN, in which case the starting address falls on the even plane or the two pages fall on different LUNs, in which case the address of the first page is on the odd plane. In case of even plane address, the status of that LUN is checked first, once it is available, the buffered data is sent along with the PROGRAM PAGE MULTI PLANE command set. Then the next page data is buffered and sent in in the form of a PROGRAM PAGE command. The LUN is then monitored using status command and once complete both the pages are checked for the operation status(whether failed or passed), if any one of the page has failed to program, the fail acknowledgment is interrupted else a success is interrupted, which completes the operation. In case of a three write page again there are two possibilities. For a request if the first address falls on the even plane, then the first two pages will be on one LUN and the next one page will be on the other. Accordingly, the request is being split into two requests. Since the first 2 pages have

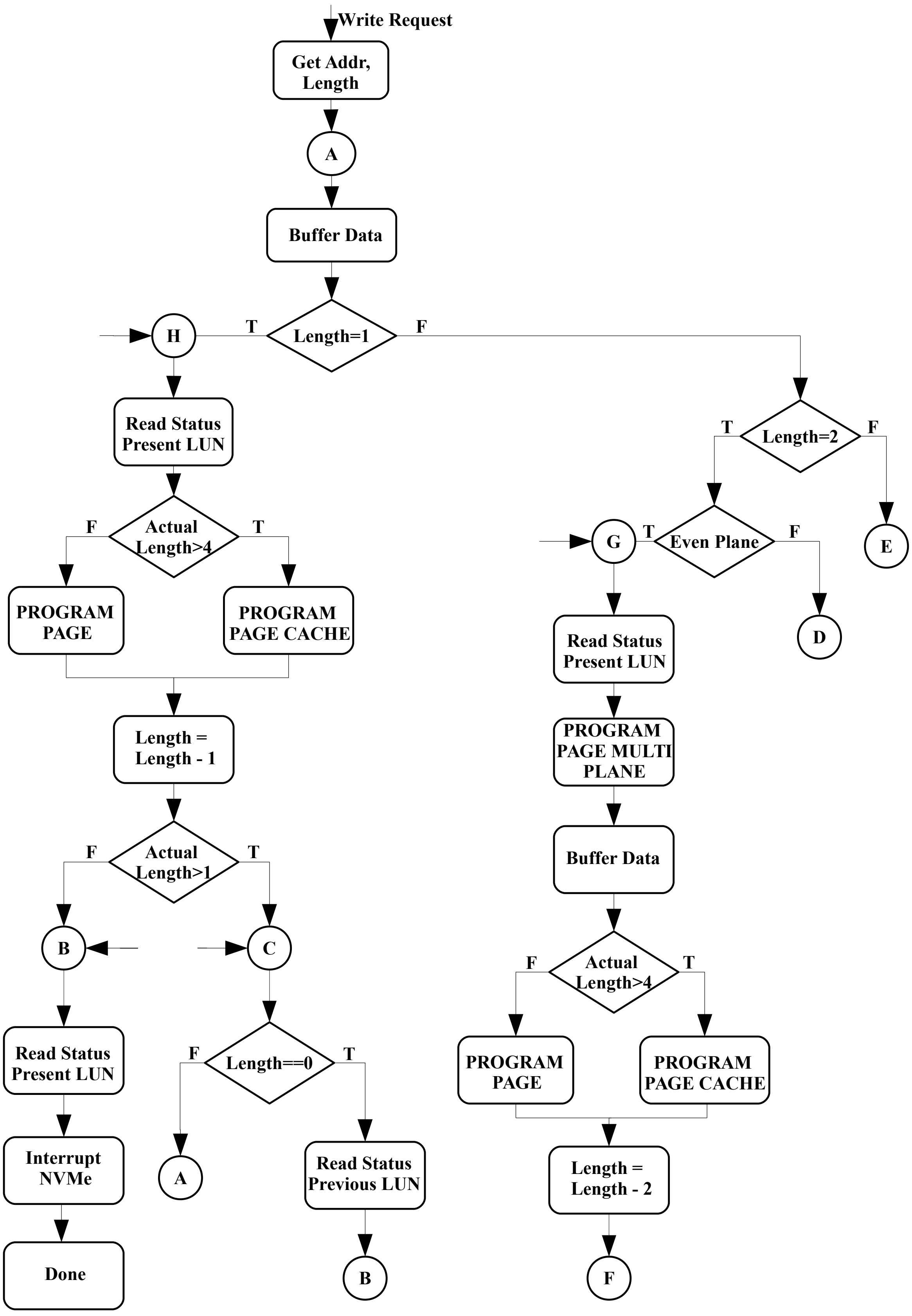


Figure 5.9: Write operation Data flow.

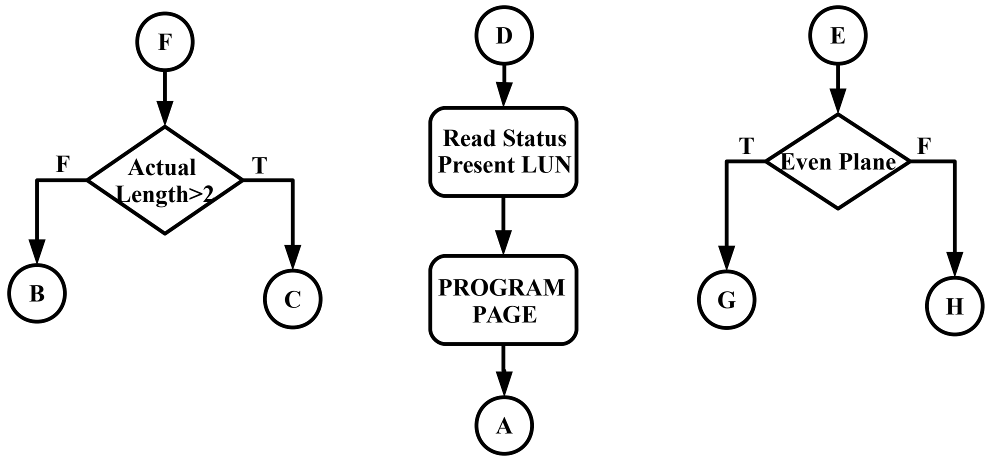


Figure 5.10: Write operation Data flow contd.

to be written on the same LUN, in the even plane address, this is similar to a two page write request on an even plane as discussed before. Once this is executed, the length is now decremented by two since two page operations are done. The remaining one page is then processed as a single page write request. Once this is executed the status is read on the previous LUN, i.e the LUN on which two pages were written. Once this completes, the operation validity is checked. If failed the interrupt is sent and the operation ends, else the status is then read on the other LUN where the single page write was executed. Finally the final status is sent in the form of interrupt. For a four page write, the request may either fall on the even plane of any of the LUN, in which case the first two pages will be on one LUN and the next two pages on the other or it may fall on the odd plane, wherein the first page will be on one LUN, the next two on the other LUN and the remaining one page on the first LUN (similar to wrapping around). In the former case, the request is split into a two page write on one LUN followed by two page write on the other LUN. Before sending the two requests the LUN status must be checked. The first LUN must then be polled using status command and then the next LUN and finally the NVMe must be interrupted. In the latter case, the request is split into a single page write in one LUN and a two page write on the other. Once these two are executed, the first LUN is monitored using status command and then once it completes and successful, another single page write request is given to the same LUN and the status checking is switched to the LUN which was handling the two page write request. Once this status is successful, the status is monitored for the other LUN till it completes successfully and the NVMe is interrupted. At any point of time if the status read indicates a failed operation, the operation stops and the fail interrupt is sent. For any write request that is of length more than four, atleast one LUN is guaranteed to have a consecutive page programs within a physical block(not Mega block). Hence the PROGRAM PAGE CACHE command is used instead of the PROGRAM PAGE command, for all writes of length greater than four. All other data flow remains the same.

The observation from the discussion thus far would be in case of multi LUN write operation, operation must be issued in alternate LUNs and then the status read must switch back and forth between two LUNs. Also as seen in an odd page length four write operation, the status read after the LUN switch may be followed by one more program series operation in the same LUN and then the switch in status read might take place. All these cases are taken care of in the data flow that is being shown. The ready/busy pin of the NFC if indicates free, it takes a write request, takes in one page data and asserts busy status till the data in the buffer is completely off-loaded. Hence in case of multi page write requests, once the first page is sent, the host then has to wait until the ready/busy pin becomes free to send in the next page data. Once the whole operation is complete, the ready/busy will assert a ready status and the write status is returned in the form of whether the operation was a success or fail.

READ operation data flow

In the read operation, the cache commands will not be used, since reading from the flash plane into the respective cache register is not a high latent operation as the program operation. Adding the cache command would only make the control logic critical but does not offer anything on te bandwidth front, since pages have to be read out from the cache registers before any other operation can be executed on the LUN. This time is comparable to the read time from target plane to cache register. Once a read request is placed on the NFC, the length and start address is registered and the new mapped address is generated. All further operation will be based on the mapped address. The data arrives from the flash in terms of bytes and the data to the NVMe is to be sent in 32,64 or 128 bits. Hence data is first packed with the help of shift registers and then buffered before being sent to the NVMe. An interrupt is given to the NVMe at any point data for a page is ready in the NFC. Once the buffer is empty the next page is buffered(in case of payload length more than one) and then again interrupted. The data flow for read operation in shown in figure 5.11 and 5.12.

Before sending the read commands, the LUNs must be checked if they are available for the read operation. In case of a single page request the LUN status is checked using status command and then the READ PAGE command is issued to the LUN. This is monitored using status command. Once it is done, the READ MODE command is issued to read data from the cache register, and the data is buffered. The interrupt is provided and the data can be read by NVMe. For a two page operation, as in write operation there are two cases. In case of an even page read request the pages are in same LUN. Hence the status of that LUN is checked and once free, the READ PAGE MULTI PLANE command followed by READ PAGE command foe next page is issued. This LUN is then polled using the status command and once free, the READ MODE command is issued to read the first page and the data is buffered and sent to the NVMe. Once the data is read out, the SELECT CACHE REGISTER command is issued to read out the remaining page, data is buffered and NVMe is interrupted. For the case where the two page request has starting address in an odd plane, the request is split into two single page read. The first single page read is executed on the first LUN after the status shows its free, then the status is checked on the next LUN and one more single page read is requested from other LUN. The status monitoring is then shifted back to the first page request LUN and once complete the data is read out using READ MODE. Then the status is checked for other LUN and once free, data is read out again through READ MODE and once after the interrupt NVMe reads the data, the request is completely processed. Now consider a three page read request. First let us consider the case where the first page falls on an even plane in any of the LUN. This request is split into two read requests, one of a two page read request and another of a single page read request. The two requests are executed once the LUNs are checked for their status. After both the requests are issued, the status is checked for the LUN where the two page read is happening. Once this is done the data is read from the first cache register(first page

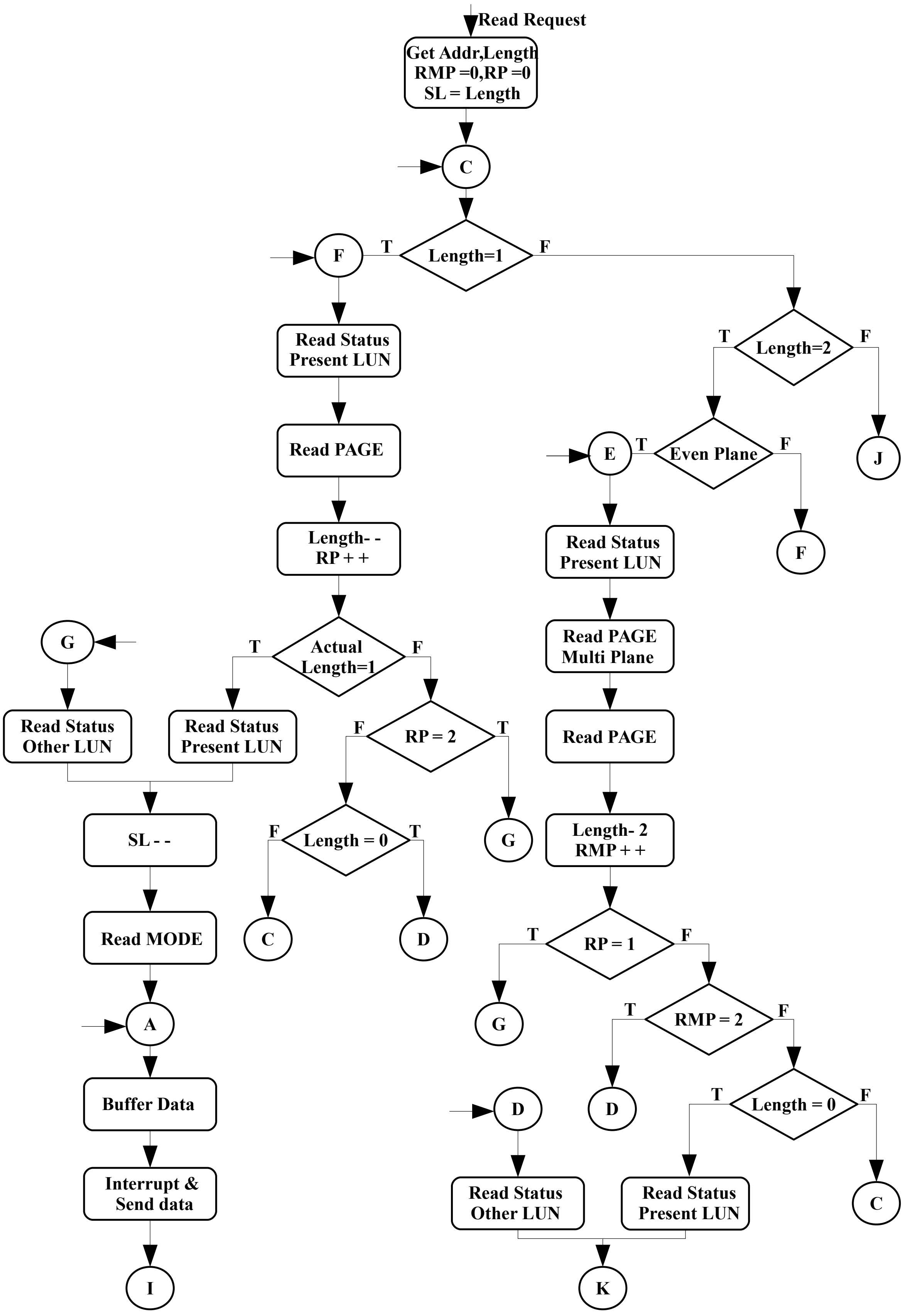


Figure 5.11: Read operation Data flow.

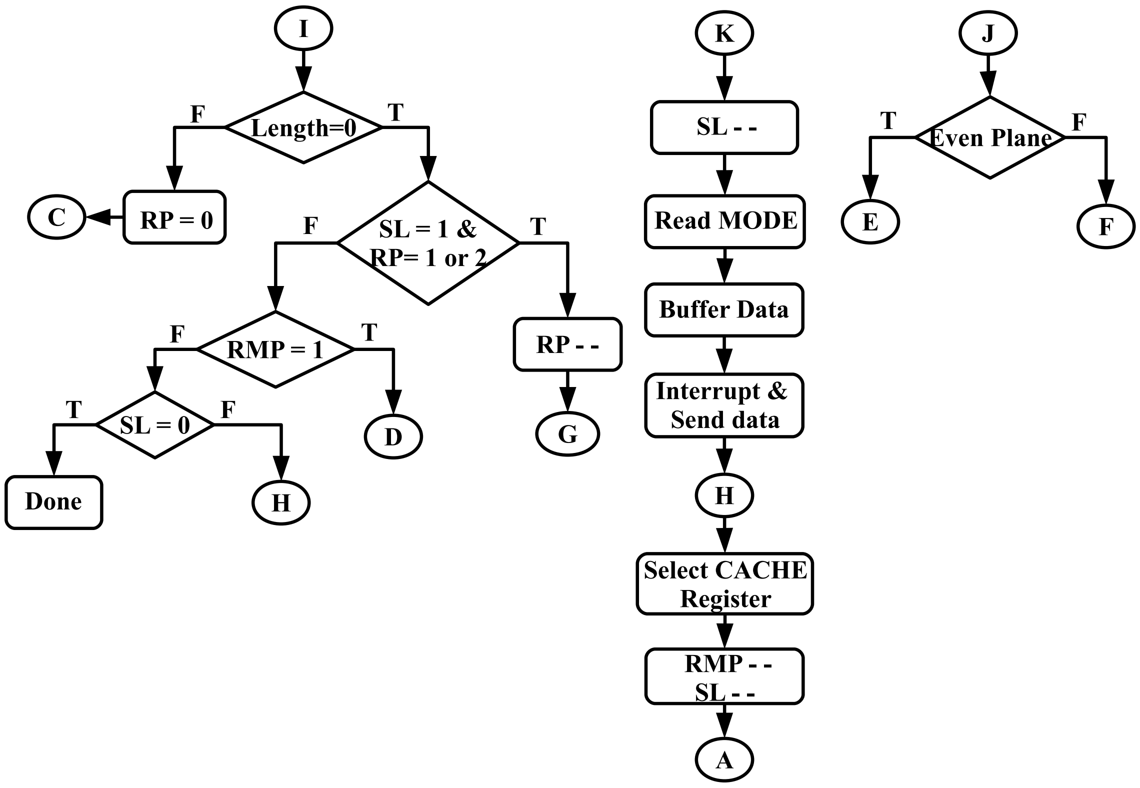


Figure 5.12: Read operation Data flow contd.

data). Once the NVMe reads it out, the next cache register data from same LUN is read out(using SELECT CACHE REGISTER) and the NVMe is interrupted. After this the status is checked again for the LUN where single page request was happening, once free the data is read out into the buffer. In the other case where the first page falls on odd plane, the request is split into a single page read on one LUN and as a two page read in the other LUN. The remaining data flow is similar to the case discussed just before this.

If a four page request is considered with the first page on an even plane, the request is split into two two-page read requests in the two LUNs. The status is read out and the LUNs are made sure to be free and the two page request is sent as discussed previously. On the other hand if we consider the case where-in the first page is on an odd plane, the request is split into three forms. A single page read request on one LUN, two page read request on other LUN and once the first completes, a single page read request on the same LUN. The status checking needs switching between LUNs. But the reading out of data in the order of pages also needs switching between LUNs. Firstly, the one page read request is executed on one of the LUN after the LUN is free. The other LUN status is checked and the two page read request is issued on the LUN. The status for the previous LUN must now be checked and hence it is switched. Once this operation is done, the data must be first read out of the cache register before issuing any more commands on this LUN. hence the data is read out and buffered and then the single page read request is commanded on the same LUN. Now the LUN is switched again to read the status of the LUN that was handling the two page read. Once free, the data is read out and buffered. After buffer becomes free the other cache register is read. After this the LUN is again switched to the single page read LUN and the data is read out from the cache register and buffered. Once the NVMe reads out this buffered data, the requested is termed complete.

For any page more than 4 , the request is split based on even or odd plane into a maximum of two page requests and switching between LUNs to read status and read out the data in an ordered manner. The data flow for the read operation precisely shows this. Once the read request request is obtained the ready/busy pin of the NFC goes busy. At any point a page data is available in the buffer the interrupt is provided, but the ready/busy will still be busy since the request is not yet completed. Only after the last page is read out of the NFC, the ready/busy asserts itself free or ready. Once the ECC decoder is integrated into this, a read status as to whether the read was successful or not can be sent based on the decoder. If the number of errors on any page read is more than the correctable errors a fail status can be sent.

ERASE operation data flow

The erase request has an associated block address with it. Firstly this block address which is logical is mapped into the physical address. Since the Mega Block is now constituted of four blocks, one each from a plane in a LUN; we need to obtain four block addresses. Thereafter the status is read on LUN 0. Once free the BLOCK ERASE MULTI PLANE command sequence is issued along with the first block address. Issuing Block erase commands was discussed in the block erase state machine. Next the BLOCK ERASE command is issued with he next block address. After this the next LUN status is checked and the same commands are sent with the remaining two block addresses. Then the operation is monitored through status and the status of all the four blocks are read. if any one of the status indicated fail, a erase fail interrupt is sent else an erase success interrupt is sent. The data-flow is shown in figure 5.13.

Bad Block scan operation data flow

The bad block scan data flow is shown in an abstract manner in the figure 5.3. A Mega block is constituted of four physical blocks. Hence the bad block table will have bad block information w.r.t to the Mega block, since it is the Mega block that is the abstraction outside of the NFC. Since each NFC can handle two chips, the number of Mega blocks in two times the number of blocks present in a plane. A bit 1 is stored in the table, in case of a bad Mega block(which is nothing but a bad physical block among the four blocks) and bit 0 for a valid Mega block. Considering a MLC flash we need to read the first spare byte of last page in every block. As shown in data flow, we start with zero block address. We generate four addresses that correspond to single Mega block. Then a READ PAGE MULTI PLANE and READ PAGE command sequence is issued on both the LUNs. This operation is monitored by status read and once completed , the first spare byte of the last page of each of the four blocks that constitute the Mega block are compared with FF. If the data is not FF in any one of the block, then the bit 0 is stored in the table corresponding to the Mega Block address. This read is done until all the blocks are covered in both the chips. At the end of this a table is obtained with all the Mega block address and corresponding bit-map indicating whether the Mega block is bad or not.

The bad block request from NVMe has a Mega block address(logical address) along with an offset, as to till what address the bit map is needed[15]. The request obtained by the NFC packs the bit-map corresponding to the start block address upto the bit-map plus offset in register. This bit map is to be sent through the data bus, hence the data is packed with zero at the MSB end if the offset is not same as the data bus width. The maximum offset is data bus width. The Interrupt is sent to the NVMe before placing the bitmap on the bus.

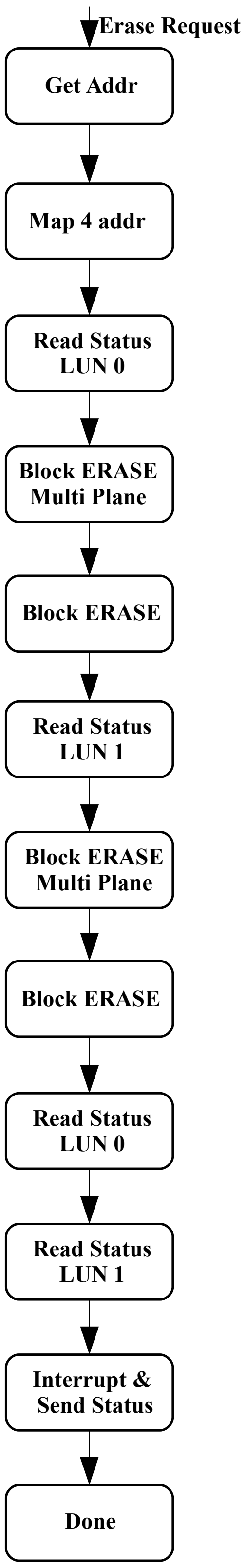


Figure 5.13: Erase operation Data flow.

5.3 Implementation of Nand Flash Memory Model

The Nand flash memory target was developed with the memory organization as shown in figure 3.2. This single target will then be instantiated twice to obtain the overall device organization shown in figure 3.4. Hence the interface corresponding to the target memory model is that of a sinlgle chip. The interface definition is listed in 5.3.

```
interface  TargetInterface ;
        method Bit#(8) _data_to_nfc_m () ;
        method Action _data_from_nfc_m ( Bit#(8) _data_from_nfc ) ;
        method Action _onfi_ce_n_m ( bit  _onfi_ce_n ) ;
        method Action _onfi_we_n_m ( bit  _onfi_we_n ) ;
        method Action _onfi_re_n_m ( bit  _onfi_re_n ) ;
        method Action _onfi_wp_n_m ( bit  _onfi_wp_n ) ;
        method Action _onfi_cle_m ( bit  _onfi_cle ) ;
        method Action _onfi_ale_m ( bit  _onfi_ale ) ;
        method bit t_ready_busy_n_ ;
endinterface
```

Listing 5.3: Nand Flash Target Memory Model Interface

Instead of an I/O interface for the data bus, two other methods **_data_to_nfc_m** and **_data_from_nfc_m** are used. This will be replaced with the actual I/O data bus when dealing with the real Nand Flash bus functional model. The other methods in the interface perform the functionality as mentioned in table 3.1.

Block RAMs(BRAM) are used to form the pages of the target memory. A vector of BRAMs is formed to obtain a single plane in the target memory. The vector size is highly parametrized; where the page size, number of pages per block and number of blocks per plane are parameterized. The number of planes per LUN is fixed to two and the number of LUNs per target is also fixed to two. This single plane is then instantiated four times to obtain the complete memory organizational except for the cache and data register. Separately four more BRAMs are used for forming the cache register four each plane and four more for the data register. LUN address bit along with the plane select bit are used to address which of the planes needs to be selected for an operation and also which cache and data register is selected. The remaining bits of the address are used for addressing the BRAMs inside the single plane. This takes care of the memory organization and address decoding part. The functional block diagram of the model is shown in figure 3.1.

The Nand Flash model supports the command sets that are mentioned in table 3.2. Additionally READ ID, READ UNIQUE ID, GET FEATURES, READ PAGE CACHE SEQUENTIAL, READ PAGE CACHE RANDOM, READ PAGE CACHE LAST commands are also supported. Hence a command register is needed and a address register to store the one/three/five address cycles. For multi plane commands queuing of operations are needed. Hence separate registers are used for queued operation addresses and the operation itself. The timing information is not captured in this memory model. A status register which updates its value every cycle is needed, since this forms the basis for multi plane and multi LUN operations.

Any data present on the input bus is considered a command and put in the command register, if the CLE is enabled. It will be considered an address if the ALE is high and it is put in corresponding address register, since there are five of them. If both these signals are disabled then it is considered as data. For every data intake to be considered the WE must be activated and for every data out to happen the RE must be activated. All these are taken care by the I/O control in the model. The data flow for various operation ranging from program, read, erase, reset,status read and others as well as queuing operations are handled by the control logic. For the verification of the NFC, the parameters of the Nand Flash Model were set in a way such that, each page was of 16KB size. Each plane had 4 blocks and each block had 8 pages.

5.4 Verification

To verify the NFC with the modelled flash memory target, we need to first verify the target memory model itself.

5.4.1 Verification of the Nand Flash Memory Model

The Design under verification, the Nand Flash Memory model, is connected to the testbench by means of the ONFi interface. The verification setup is shown in figure 5.14. The Test cases are input to the DUV and the behaviour of the model is verified.

The test cases involve feeding ONFi command sets along with data to verify the various commands that are supported by the nand model. The various test cases that are targeted are discussed further.

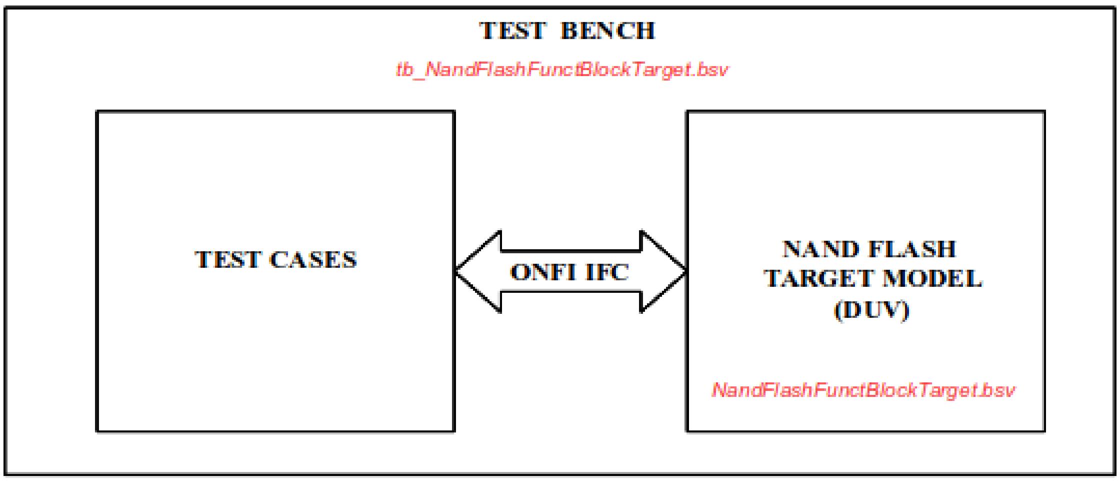

TEST BENCH
tb_NandFlashFunctBlockTarget.bsv
TEST CASES
ONFI IFC
NAND FLASH TARGET MODEL (DUV)
NandFlashFunctBlockTarget.bsv

Figure 5.14: Verification setup for verifying the target memory model.

Reset

The Reset command is issued to the model and the pages are read and verified that they are all FF. This verifies the reset operation in the model.

Write and Read one page in 2 LUNs

After RESET is done, PROGRAM PAGE command is issued once in LUN 0 and then on LUN 1. The status is monitored using status commands. The READ PAGE command is then issued on the same two pages in both LUNs. The status is monitored using the READ STATUS command. Once done the data is read out using READ MODE and the data is compared with written data. This verifies the all the commands that are used and also interleaving of operation between two LUNs in the model.

Write and Read using Multi-plane commands across LUNs

Once again after RESET is successful, PROGRAM PAGE MULTI PLANE and PROGRAM PAGE commands are issued on LUN 0 for writing into two pages. The same is repeated on LUN 1. After the status is monitored successfully, the READ PAGE MULTI PLANE and READ PAGE commands are issued on the pages that was earlier written both in LUN 0 and LUN 1. The status is monitored again and upon completion, the READ MODE and SELECT CACHE REGISTER operations are issued to read out the data from both LUNs and the data is verified with the written data. This test ensures that all multi plane commands and select cache register commands are operational also verifies the multi-plane interleaving along with LUN interleaving.

Write and Read using CACHE operations

Upon completion of reset, the LUN 0 is issued a PROGRAM PAGE CACHE on a page followed by same operation on the next page within the same block. Then the same command is issued on LUN 1. The status is monitored and then the READ PAGE command is issued on LUN 0 followed by READ PAGE CACHE on LUN 0. The same sequence of read operations are then issued on LUN 1. The data from LUN 0 and LUN 1 are read out and compared with the written data. Then the CACHE operation is ended with the READ PAGE CACHE LAST command on both the LUNs and the data is verified. This test case ensures the CACHE operations are verified across LUNs.

Write and Read using extended CACHE operations

In this test we concentrate only on one LUN to make sure an extended cache operation is functional. After the reset is successfully completed, the PROGRAM PAGE CACHE is issued four times on a block in a LUN. This ensures four pages are written back to back on the block. The status is monitored using status operations and then upon completion, the READ PAGE is issued once and followed by READ PAGE CACHE SEQUENTIAL command. Read out the first page data and compare. Then issue READ PAGE RANDOM command on the third page, this will first put the data in data register(second page data) into cache register then start reading third page into data register. Second page data is read out and verified. The command sequence is ended with the READ PAGE CACHE LAST command. The third page is read out and finally after the status shows free, the last page is read and verified. This ensures the CACHE series commands are functional in the model.

Write using Multi plane and CACHE operations, read using Multi plane operation

We need to verify whether cache and multi plane operation happen simultaneously as a part of interleaving. After the reset, the PROGRAM PAGE MULTI PLANE command is issued on a plane in a LUN. This is followed by PROGRAM PAGE CACHE on the same LUN but different plane, once the array status is free again issue the same operation on the consecutive pages in the same block. Wait until the status shows the LUNs are free. Then the READ PAGE MULTI PLANE command followed by READ PAGE is issued to the same addresses as the first write, the data is read out using READ MODE and verified. Again the same read command are issued on the next addresses and the data is read out and checked.

Multi plane and normal Erase operation

Write data other than FF into some pages in two or more pages in a LUN on different plane using PROGRAM PAGE, PROGRAM PAGE CACHE and PROGRAM PAGE MULTI PLANE commands. Once the LUN is free, issue BLOCK ERASE MULTI PLANE and BLOCK ERASE on the blocks that was earlier written into. After the operation is completed read out the same pages that were written into earlier using READ PAGE MULTI PLANE and READ PAGE command. Ensure that the read out data from these pages are FF and not the ones that were earlier written. This verifies the block erase functionality in the model.

5.4.2 Verification of NFC with the Nand Flash Memory Model

The Nand Flash Model is integrated with the NFC for the verification. The test cases are fed through the NVMe-NFC interface and the design is to be verified. The verification setup for verifying the NFC is shown in figure 5.16. The vast majority of the failures are a result of interaction between adjacent memory cells. Often writing a memory cell can cause one of the adjacent cells to be written with the same data. An effective memory test attempts to test for this condition. Therefore, an ideal strategy for testing memory would be (a) write a cell with a zero (b) write all of the adjacent cells with a one (c) check that the first cell still has a zero[16]. A close approximation of this test would be to write consecutive rows in the memory with pattern "A" and "5". After read then write with pattern "5" and "A". Other than this a random pattern can be written in every row followed by its complement and repeated. The read write test cases follow this methodology. The different test cases that are covered are discussed further.

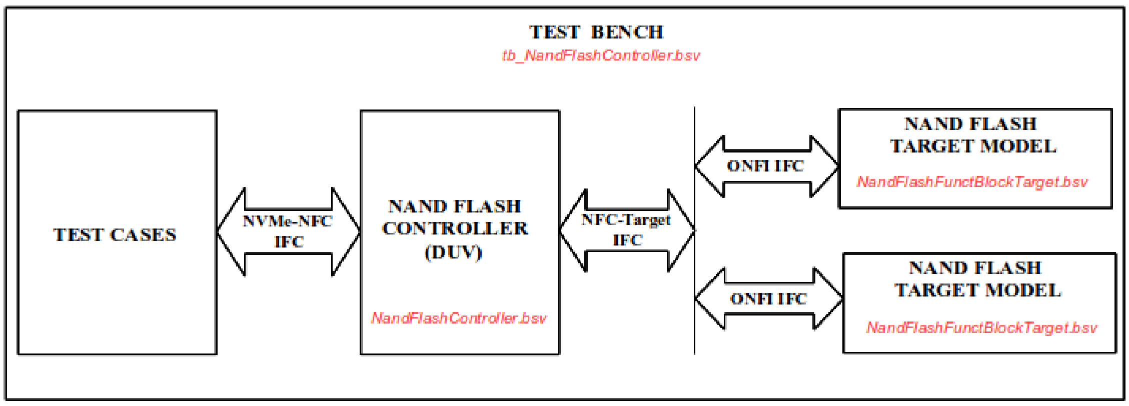


Figure 5.15: Verification setup for verifying NFC.

Bad block operation

To verify whether the bad block scan at power-on is functionally valid, we load some of the BRAMs that form the pages of the block with data other than FF. The bad block table should then have a bit map corresponding to this case. Then a bad block request is sent from the NVMe interface and verified.

Write and Read operation

Write operation is done for various payload length with data pattern as discussed before. For various payload lengths starting address can be in either odd and even plane. Hence write operation with length 1 and lengths from 2 to 13 for even and odd plane is first done in separate test cases. Then the read is done separately for the same payload lengths and the data is compared with the written value.

Erase operation

After a write operation on a page, we send an erase operation through the NVMe. After the erase is successful read the data and ensure it is FF and not the original data.

5.4.3 Synthesis Evaluation

The Nand Flash Controller and the Nand Flash model with 4 pages per block and 4 blocks per plane is executed on xilinx Artix-7 FPGA and verified to be functionally working. The NFC is then synthesized with a NVMe-NFC data bus width of 128 bits. Below is the synthesis report(slice utilization and timing summary) for the same.

Number of Slice Registers used : 1131

Number of Slice LUTs used : 2214

Number of LUT Flip Flop Pairs used : 2262

Number of Block RAMs used : 4

Max Frequency of operation : 299 MHz

5.4.4 Verification of NFC with Micron class-K Bus Functional Model

The timing aspect of the NFC is not verified with the earlier verification wit the bluespec nand flash target model. A micron class-K nand flash bus functional model with a page size of 16KB is integrated with the NFC and all the test cases that were earlier discussed are re-run on this setup. The simulation results for this are shown in the following figures.

PROGRAM operation

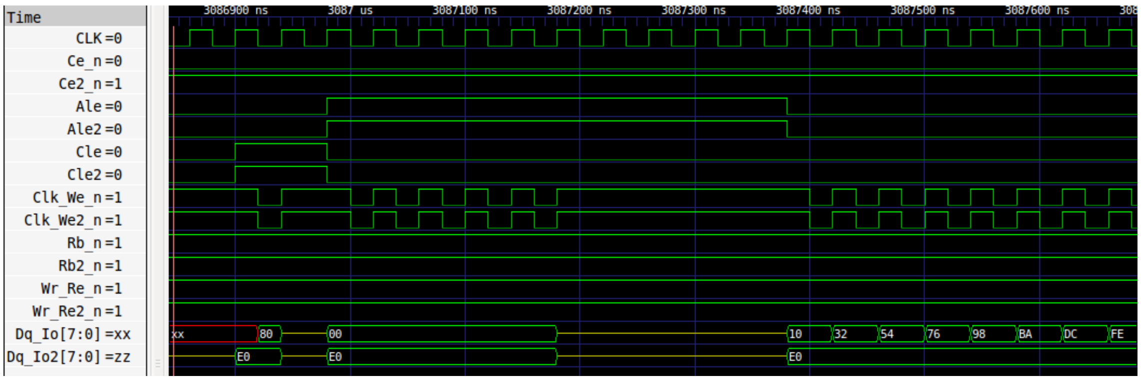


Figure 5.16: Program operation (80h followed by data).

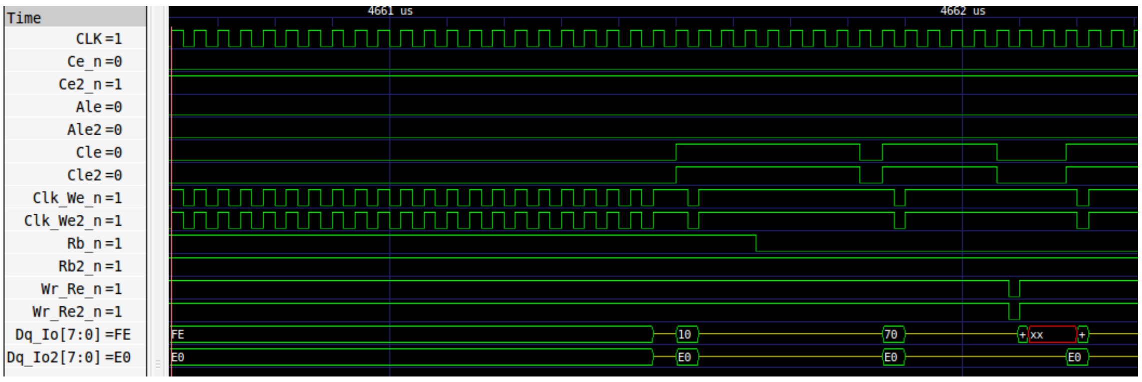


Figure 5.17: PROGRAM PAGE end command sequence(10h after data).

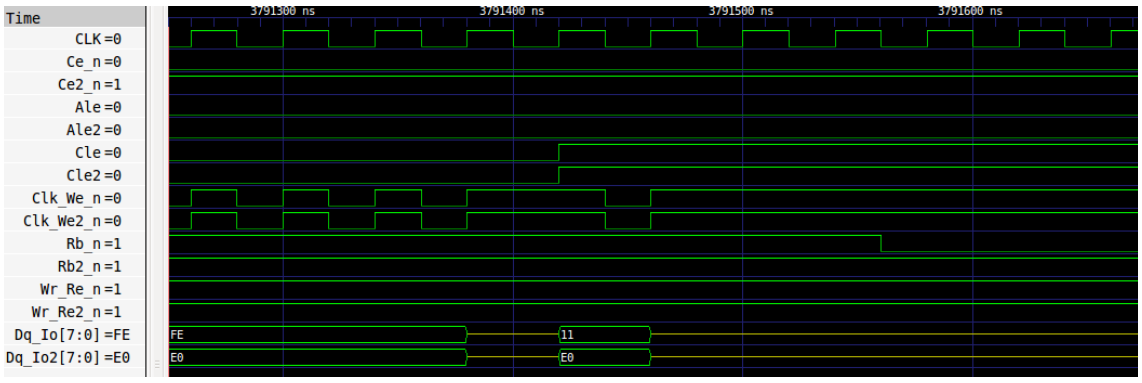


Figure 5.18: PROGRAM PAGE MULTI PLANE end command sequence(11h after data).

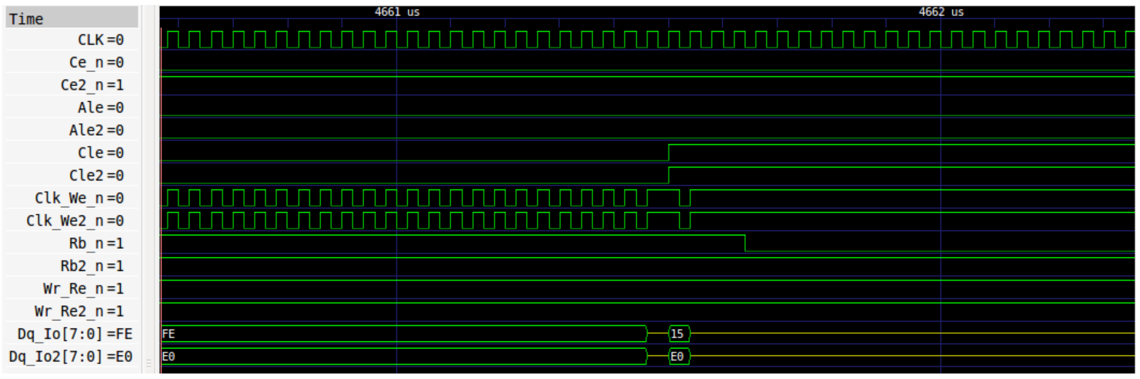


Figure 5.19: PROGRAM PAGE CACHE end command sequence(15h after data).

READ operation

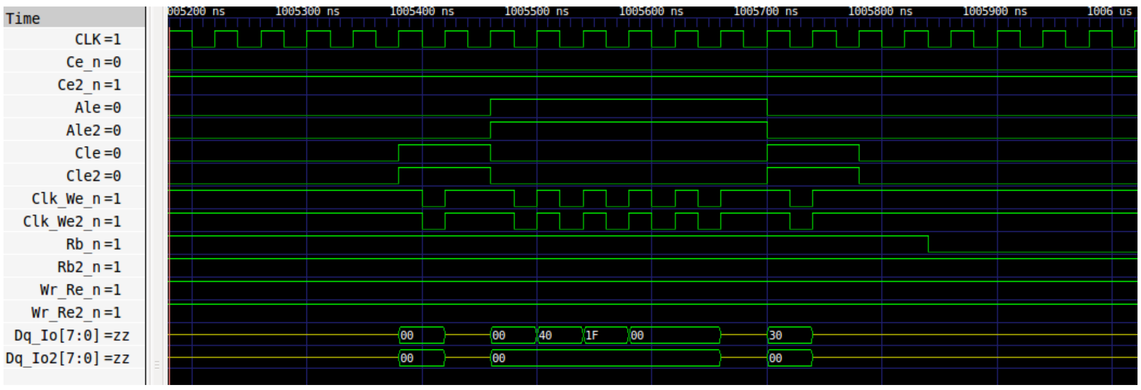


Figure 5.20: READ PAGE command sequence.

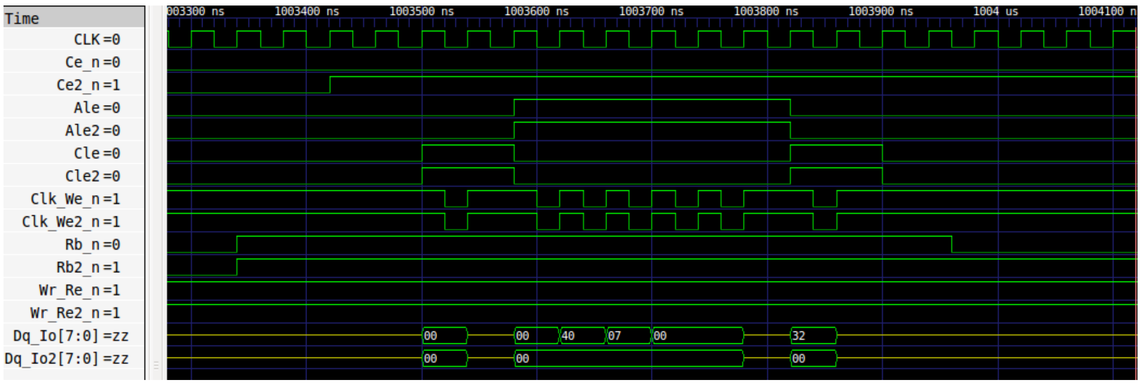


Figure 5.21: READ PAGE MULTI PLANE command sequence.

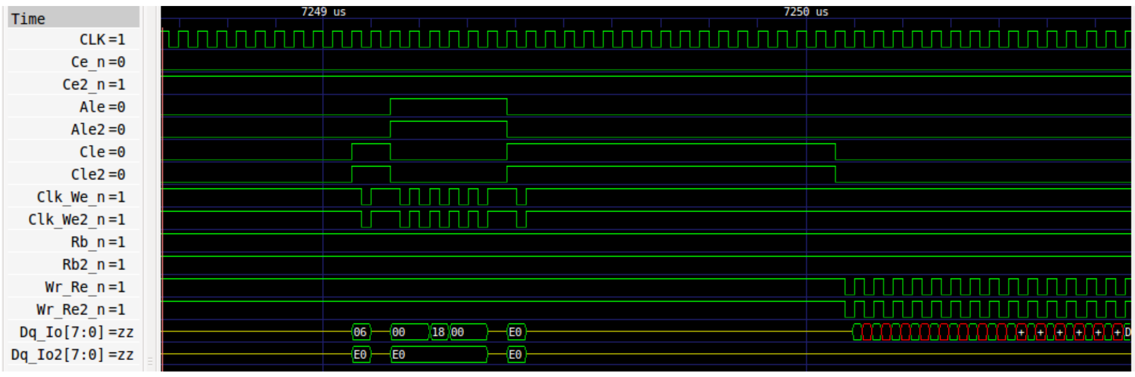


Figure 5.23: SELECT CACHE REGISTER command sequence.

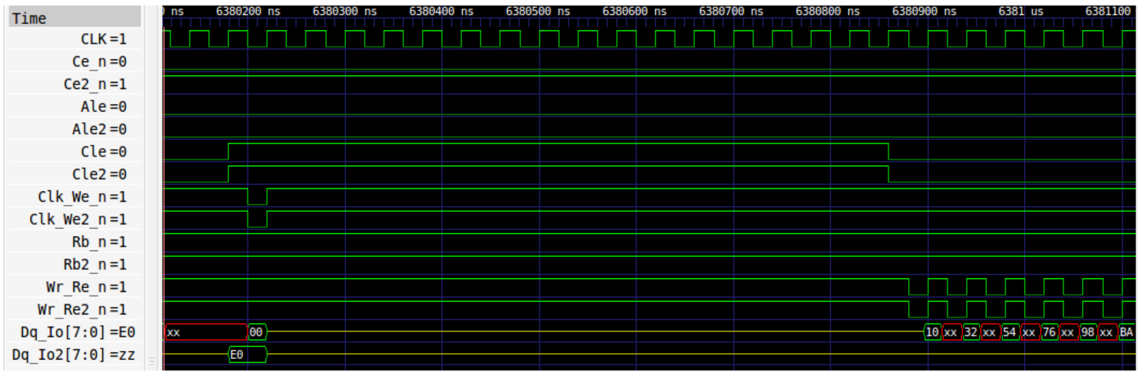


Figure 5.22: READ MODE command sequence(After status check).

STATUS operation

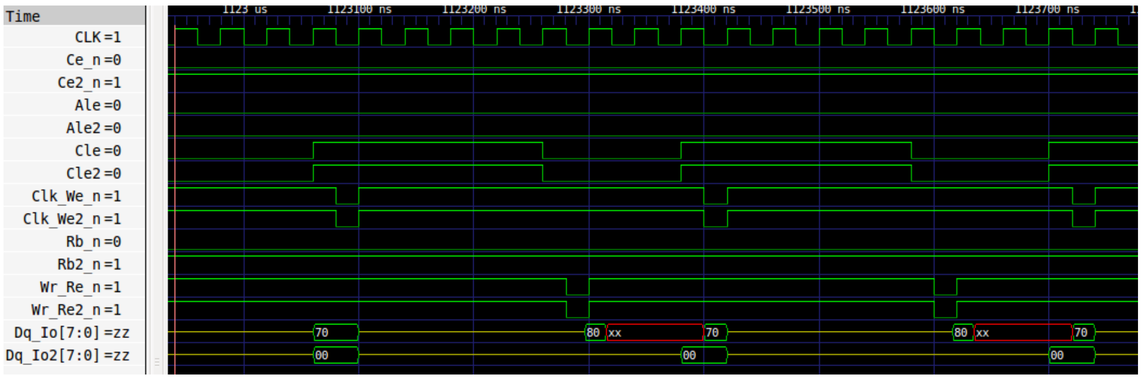


Figure 5.24: READ STATUS command sequence(70h).

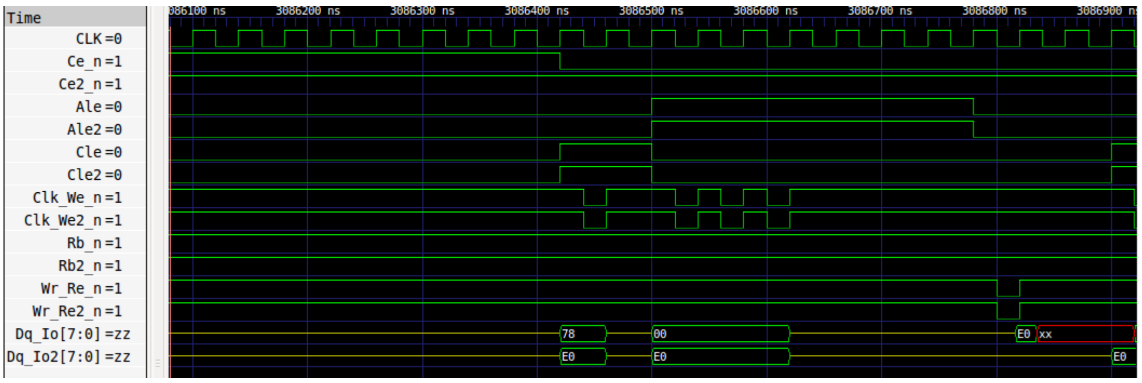


Figure 5.25: SELET LUN WITH STATUS command sequence(78h).

ERASE operation

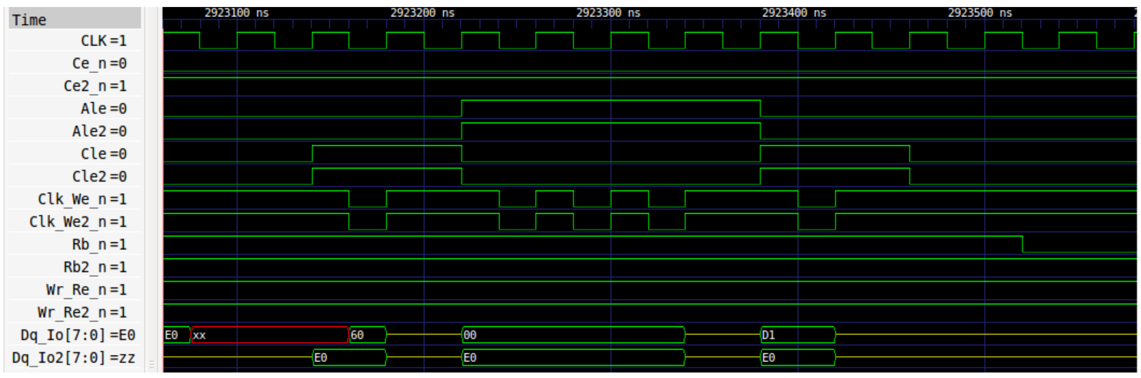


Figure 5.26: ERASE BLOCK MULTI PLANE command sequence.

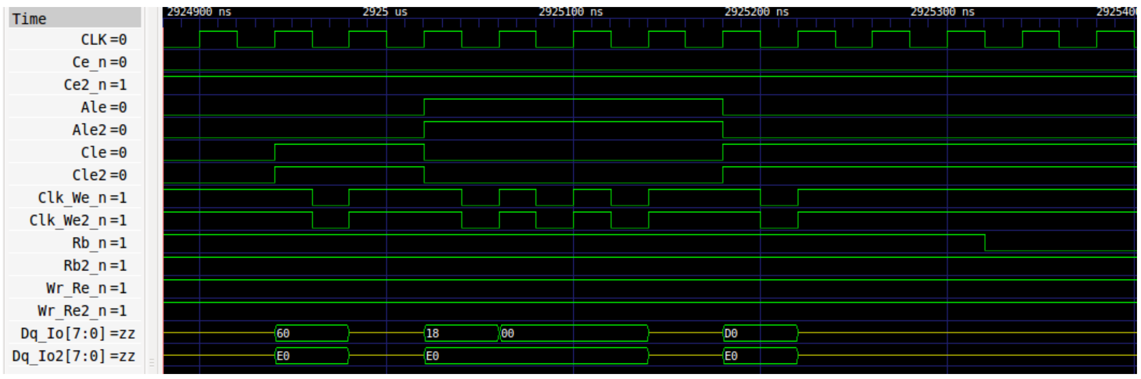


Figure 5.27: ERASE BLOCK command sequence.

CHAPTER 6

Conclusion

6.1 Summary

The concept of Nand Flash Controller which is compliant with ONFi 4.0 standard has been elaborated and implemented in this thesis. The NFC takes the NVMe compliant signalling and translates it into an ONFi compliant signalling to talk to the NVMe and the Nand Flash memory. Every request such as the read request, write request, erase request are processed by means of state machine in the control logic of the NFC. Also in the control logic are the power-on operations such as reset, set feature and bad block scanning. More commands can be integrated into the control logic and can be verified easily with the existing environment. The control logic can be extended by adding more rules at the bluespec abstraction level. Along with the control logic, a verification environment is also setup using bluespec which can be extended to capture more tests. This test environment can be used to verify the NFC both in simulation as well as on the hardware. Also simulation with the actual bus functional model is also covered in the environment. The NFC thus implemented has been tested both through simulation and on real hardware.

6.2 Future work

The NFC shares control signals which are ONFi compliant with the two target chips. Currently the Mega block created has blocks shared across planes in a LUN and then across LUNs. Since the signals are already shared, the Mega block can be extended to add blocks across CE as well. This will add one more level of interleaving along with plane and LUN interleaving.

The present implementation of the NFC concentrates on the asynchronous interface. The design can be extended to also support synchronous interface.

More commands that are available in the ONFi 4.0 standard can be incorporated for better control logic in the NFC. However this is possible only if the level of abstraction at the NVMe-NFC end is increased. A better level of abstraction will allow addition of copyback and other commands in the NFC.

REFERENCES

[1] Bluespec Inc. *Bluespec System Verilog Reference Guide*. Revision 30, January 2012.

[2] Rishiyur S Nikhil and Kathy R. Czeck. *BSV by example*. Edition 1., 2010.

[3] Maximilian Singh. *Development of a FPGA Based Programmable Storage Controller*. October 2014.

[4] Micron technology Inc. *High Speed Nand Flash Memory*. Revision 0.95, April 2008.

[5] ONFi Workgroup. *Open Nand Flash Interface Specification*. Revision 4.0, February 2014.

[6] Micron technology Inc. *Nand Flash Memory*. Revision D, May 2008.

[7] Micron technology Inc. *TN-29-28 Memory Management in Nand Flash Arrays*. Revision A, July 2007.

[8] Micron Technology Inc. *TN-29-19 An Introduction to Nand Flash*. Revision B., April 2010.

[9] Micron technology Inc. *AN1819 Bad Block Management in Nand Flash Memory*. Revision G, July 2010.

[10] Elnec. *Nand Flash Memories and Programming Nand Flash Memories*. Version 2.11, January 2014.

[11] Nina Mitiukhina. Overview of the nand flash high speed interfacing. *IEE 5008 Memory Systems*, 2013.

[12] Micron technology Inc. *TN-29-25 Improving Performance Using Two Plane Commands*. Revision B, September 2008.

[13] Macronix International Co. *Application Note on Bad Block Information*. Version 01, November 2013.

[14] Micron technology Inc. *TN-29-13 Determining Nand Flash Ready/Busy Status*. Revision C, February 2010.

[15] Matias. *Light NVM Support Interface Specification*. February 2015.

[16] MemtestOrg. URL `http://www.memtest86.com/technical.htm`.