Implementation and Verification of NAND flash Controller using ONFI4.0

A Project Report

submitted by

Y RATHAIAH
EE16M047

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

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2018

THESIS CERTIFICATE

This is to certify that the thesis entitled **Implementation and Verification of NAND flash Controller using ONFI4.0**, submitted by **Y.Rathaiah** bearing roll number of **EE16M047**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

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

Place: Chennai

Date: 17 May 2018

ACKNOWLEDGEMENTS

My foremost gratitude and respects to my guide Dr. V. Kamakoti for his guidance, motivation and moral support in my execution of this project work. Despite a very busy schedule, he always gave me a patient hearing.

I would like to extend a special thanks to my Co-guide Prof. Saurabh Saxena who supported me in my academcs with his advise. My special thanks to Mr. G.S. Madhusudan for his inputs and motivation through out the project.

I thank Vishvesh for his passionate talks about the subject and many hour-long discussions which renewed the zeal in me to keep trying for results. His patience has been incredible.

My special thanks to Dr.Neel Gala for giving me this project. I thank Arjun, Rahul, Rama Muni and other rise labmates for lending their immense help in my work.

I thank my fellow batchmates Naveen, Shashi, Wing Commander Sandeep Singh, Neelesh, Shiva, Mani krishna and others for their help and support.

Finally I thank God for making all of this happen.

ABSTRACT

KEYWORDS: SSD, HDD, PCIe, NVMe, NFC, ONFI

Solid State Drives (SSDs) are gaining momentum in memory applications, replacing Hard Disk Drives(HDDs) by offering higher performance and lower power. CPU performance growing exponentially for the past two decades in the case of SSD, while HDD performance has improved linearly for the same period. Additionally, multi-core CPU designs have increased randomness of storage I/Os. These trends have shifted from HDD to SSD.

The storage controller used for Shakti processor has host, PCIe, NVMe, NFC and NAND flash chips. For maximum utilization of SSD performance NVMe is being used. Communication between host and NVMe is through PCIe. Between NVMe and NFC, ONFI interface is used. ONFI4.0 specification defines a standardized NAND Flash device interface that provides the means for a system to be designed that supports a range of NAND Flash devices.

The NAND Flash Controller(NFC) part has been verified for basic operations such as program, read and erase in NAND flash chips. AXI4 interface has been used to send request from testbench to NFC for operations. Multi-plane operations also verified.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SSD	Solid-State Drive
HDD	Hard Disk Drive
ONFI	Open NAND Flash Interface
NFC	NAND Flash Controller
NVMe	Non-volatile memory Express
PCIe	Peripheral Component Interconnect
AXI	Advanced eXtensible Interface
LUN	Logical Unit / Logical Unit Number
CLE	Command Latch Enable
ALE	Address Latch Enable
RPM	Rotations per minute
IOPS	Input/output operations per second

CHAPTER 1

INTRODUCTION

Semiconductor technology is advancing everyday by which processing power also increasing. Moore's law states that number of transistors in an Integrated circuit would double every two years. This effectively means that processing power of computers will also double every two years. With this rapid growth, chip capacity has grown very fast and SOC(System-On-Chip) has become very crucial to integrate hardware,software and other applications. But there is no much improvement in IOPS(Input/output operations per second). Hard disk is where everything else is stored. It stores all of your programs, documents and other files. To access a file or run a program, the system needs to load it from the hard disk and into memory. But there is vast difference between above two speeds. Processing speed is in nanoseconds while HDD takes milliseconds. Magnetic HDDs have been used for database applications mostly. These HDD have been becoming slower as the write and read speed depends on RPM.

Here comes Non Volatile Memory(NVM) based storage devices. Flash based SSD is opted for better performance speed. So the storage has become little faster. NAND flash Memory is selected. A memory management unit was designed for an efficient data transfer between the processor and the main memory. It supports NVM Express based I/O Subsytem with a PCI Express interface. NAND Flash Controller is included in the design for non-volatile memory management. It acts as a controller interface between NVM Express and NAND Flash memory.

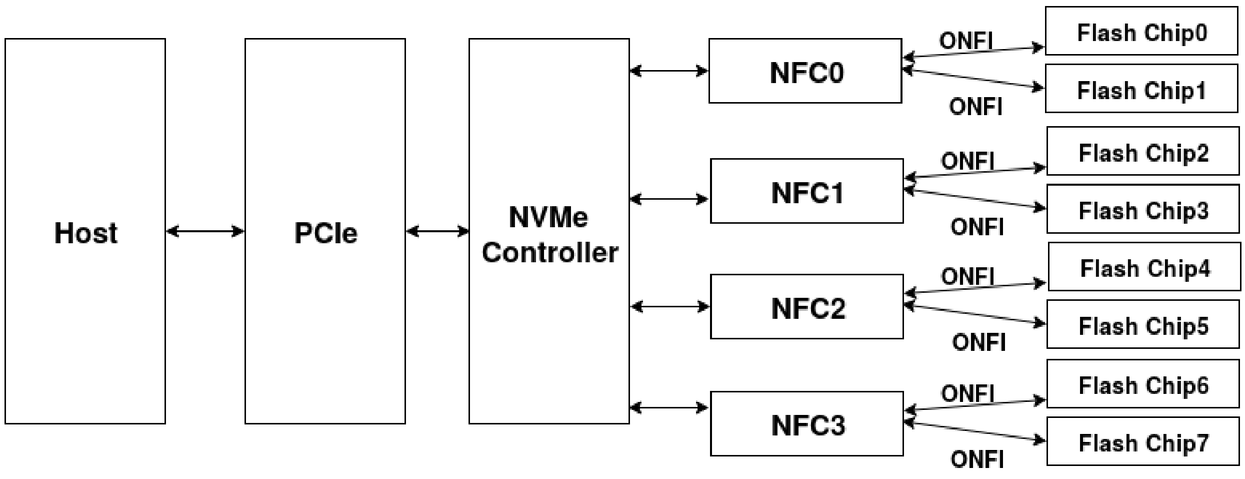


Figure 1.1: Memory Subsystem

1.1 Overview

Host/CPU is a single or multi processing entity which uses this memory system for I/O access to non-volatile memory. This sets I/O submission and completion queues and submission of I/O commands to the appropriate queues based on the priority of the I/O command. For full PCIe (Peripheral Component Interconnect express) is an interconnect between host and NVMe controller. It is designed to connect peripheral devices to a host processor and has a tree-based structure with a central root complex and switches connecting the nodes flowing down the tree.

NVM Conttroller is a bridge between host and NFC. Fetching, execution and dispatch of I/O commands issued by the host into the I/O queues. Pipelined execution and parallelization of IO commands across channels, deciding the IO bandwidth and channel utilization.

NAND flash controller is a bridge between NVM controller and ONFI interface of NAND flash chips. Every channel has an independent NAND flash controller and each NAND flash controller can control maximum of two NAND chips. It executes NAND commands issued by the NVM Controller and returns the status of the NAND flash command to the NVM controller. It also implements error correction for reliable NAND data reads, and also maintains persistent bad block tables and returns them to the NVM Controller.

1.2 My contribution

NAND flash controller has been implemented previously but not fully legitimate. So, this has been corrected fully according to ONFI4.0 specification. The focus of present thesis will be on functioning of basic commands like Write, read and erase from NAND flash chips. These commands are implemented using ONFI4.0 by maintaining timing issues. And later they are fully tested for paticular test cases.

1.3 Organization of the Thesis

The remaining part of Thesis has been divided into following chapters for ease of understanding and coverage.

- Chapter 2 deals with basics NAND flash memory and Bluespec System Verilog.
- Chapter 3 discusses AXI4 interface and related aspects used.
- In chapter 4 deals with fuctioning, addressing, memory organization of NAND flash and ONFI commands used.
- Chapter 5 deals with mega block concept used for obtaining parallelism.
- Chapter 6 shows the implementation of NFC and simulation results.
- Chapter 7 concludes thesis and dicusses future work.

CHAPTER 2

BACKGROUND

This chapter discuss about the basic building block of Nand flash memory i.e., nand flash cell. And then brief introduction of Bluespec System Verilog.

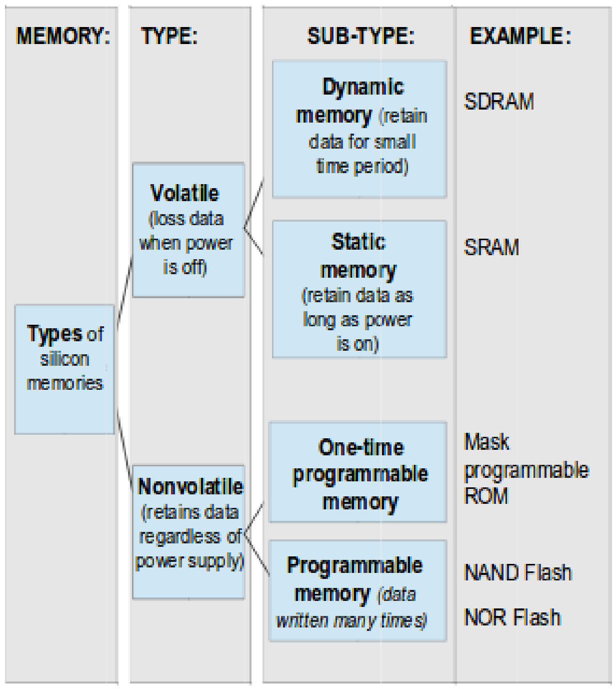

MEMORY:
TYPE:
SUB-TYPE:
EXAMPLE:
Dynamic memory (retain data for small time period)
SDRAM
Volatile (loss data when power is off)
Static memory (retain data as long as power is on)
SRAM
Types of silicon memories
One-time programmable memory
Mask programmable ROM
Nonvolatile (retains data regardless of power supply)
Programmable memory (data written many times)
NAND Flash
NOR Flash

Figure 2.1: Types of silicon memories

Silicon can mainly be divided into Volatile and non-volatile memories. When power is lost, volatile memories will lose data. While there will be no loss of data although power is lost. Speed of volatile memories may be greater than non volatile memory. NAND flash memory is Non volatile memory.

2.1 Nand Flash Memory

2.1.1 Flash Memory

Flash memory is a type of non-volatile memory. This can be electrically erased and re-programmable. As over-write is not possible, erase must happen before program in a cell which is already programmed. It is not possible to erase each cell in a flash memory individually. It may be possible if we add a large additional circuitry. This will increase cost. So, this approach is dropped and large part is erased namely block-wise erase.

2.1.2 Basic NAND Flash Cell

Nand flash cell shown in Figure 2.2 is the basic building block of almost every solid state drive. For storing a single bit of data on a solid state drive, the smallest building block ″single NAND flash cell″ is required. This single cell can be set to either a ′0′ or ′1′ and it will continue to store that state even after power is removed as it is a type of Non-volatile memory.The floating gate will remain in it state of charged or uncharged until it is surrounding circuitory is changed.

Table 2.1: State of NAND flash cell based on charge

Floating gate state	**Referred as**	**Binary value assigned**
Charged	Programmed	′0′
No Charge	Erased	′1′

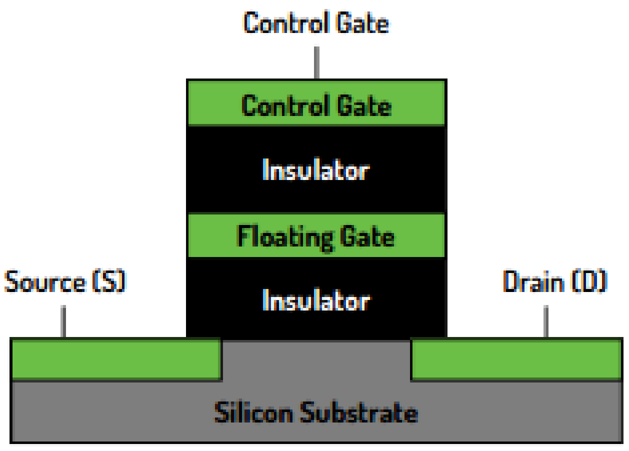


Figure 2.2: Basic NAND flash cell

2.1.3 Reading a NAND flash cell

To read a NAND flash cell, voltage is applied to control gate. Then by knowing the current flow from source to drain will tell whether it is charged or not.

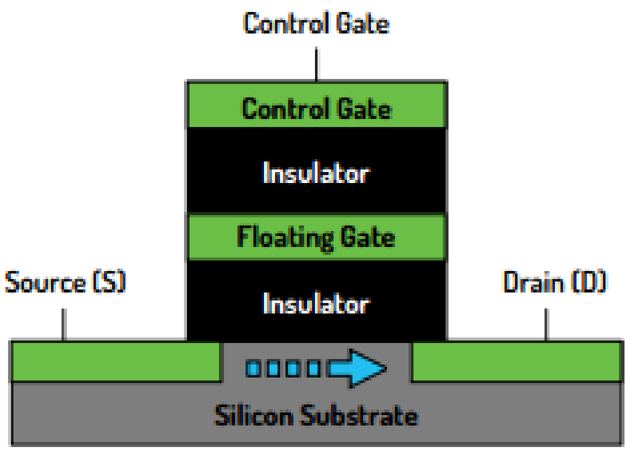


Figure 2.3: Value ′1′ is stored as current is flowing from Source to Drain

If there is current flow, it signifies that floating gate is not charged i.e., value stored is ′1′ as in Figure 2.3

If there is no current flow, it signifies that floating gate is charged i.e., value

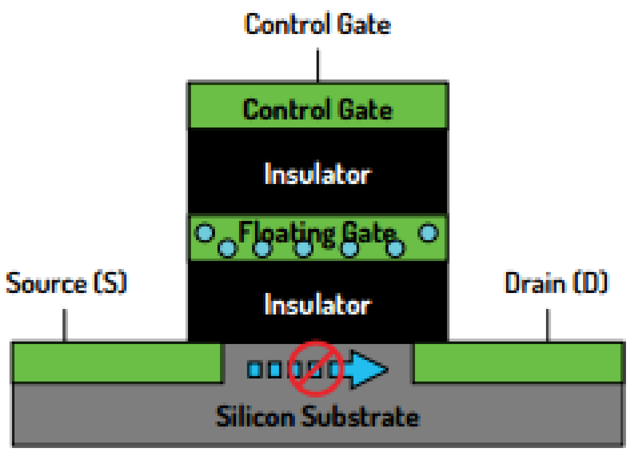


Figure 2.4: Value ’0’ is stored as no path for current from Source to Drain

stored is ’0’ as in Figure 2.4

2.1.4 Write in a NAND flash cell

For writing in NAND flash cell, a high voltage is applied to control gate. By this electrons move from silicon substrate to floating gate. This process is known as ”Tunneling” as electrons tunnel from substrate to floating gate through oxide insulator.After write happened, the value in NAND cell is ’0’. Refer Figure 2.5

2.1.5 Erase a NAND flash cell

For erasing NAND flash cell, a high voltage is applied to silicon substrate. By this electrons move from floating gate to silicon substrate. This also uses Tunneling process. After erase happened, the value in NAND cell is ’1’. Refer Figure 2.6

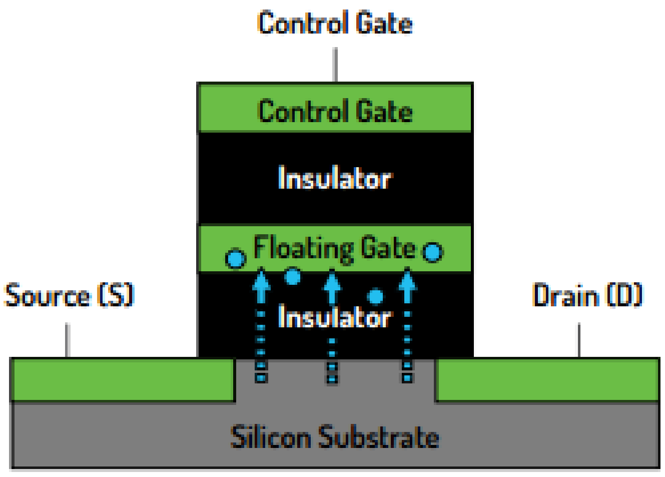


Figure 2.5: Writing/programming a NAND flash cell

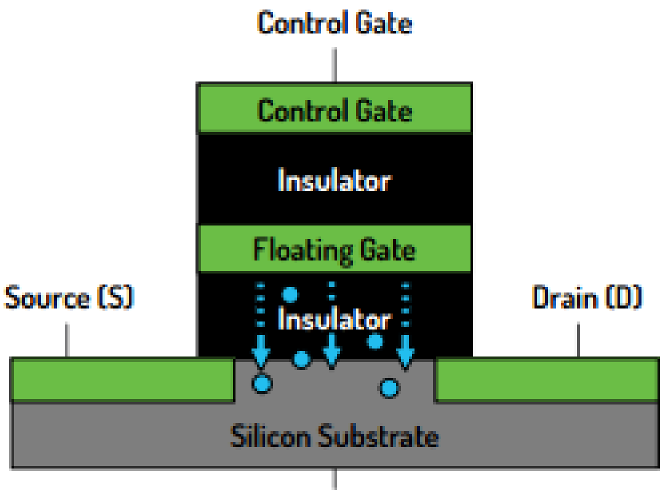


Figure 2.6: Erase a NAND flash cell

2.1.6 NAND cell life

Due to Tunneling process which is required for Write and Erase functions to happen, stress will be developed on oxide insulating layer. As Tunneling takes place multiple times, this stress breaks down the oxide layer and floating gate can't maintain the charge. At some point the cell is no longer usable and cell must be removed. This is the reason for finite number of write/erase per cell of NAND flash. So, all these cells which are no longer useful must be noted for no problems further. Bad block management is done which will be discussed in later chapters for maitaining these all bad cells. In NAND flash there is no concept of overwrite. To write in an address in which already once written, first erase should happen and then write. By this cell life reduces. Endurance is 100,000 PROGRAM/ERASE cycles.

2.2 Bluespec System Verilog

BSV is a HDL used in design of electronic systems such as FPGA, ASIC etc. It is high level language and results in synthesizable hardware which can run on FPGA emulation platforms. BSV substantially extends the design subset of System Verilog and also increases the programmers coding efficiency. It uses rules and interface methods for behavioral description, which adds a powerful way to express complex concurrency and control.

2.2.1 Key Features of BSV

- High level atomic rules in place of Verilogs always block
- High level interfaces instead of Verilogs port list
- Powerful Parametrization and Polymorphism.
- Fully synthesizable at all levels of abstraction.

2.2.2 Bluespec SystemVerilog Constructs

Rules

Rules are used to explain how the data shifts from one state to another state, instead of the Verilog method of uses always blocks. Every Rule has two components:

Rule conditions : In rule condition we declare condition like in while in c. If condition satisfied then goes to rule body.

Rule body : It is a set of actions these explains state transitions.

Modules

A module has of three kind of things: state, rules that operate on that state, and an interface that has inputs and outputs of module. A module definition specifies a scheme that can be instantiated multiple times.

Interfaces

Interfaces give a means to group wires into bundles with mentioned uses, explained by methods. An interface is a tend to remind one of something of a struct, where each member is a method. Interfaces may have other interfaces also.

Methods

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

- Value Methods: It takes zero or more parameters and returns a value

- Action Methods: It takes zero or more parameters and it performs an action inside of module
- Action Value Methods: It takes Zero or more parameters, and performs an action, and returns the result

Functions

Functions are simply parametrized combinational circuits. Function application simply connects a parametrized combinational circuit to actual inputs. Can be used multiple times.

Data Types

Common scalar types like Bit(n), Bool, UInt(n)and Int(n) in bits type class. Other common data types include Integer, which belongs to the Arith type class.

Typedef:typedef is used to define new and more reliable synonym.

typedef existingDataType NewDataType;

typedef bit[63:0] Data;

typedef bool flag;

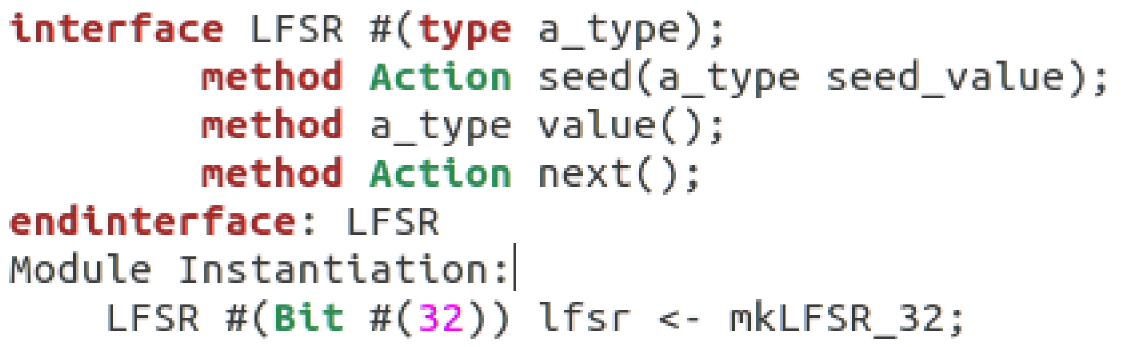
Tagged Union: A tagged union is a composite type made up of a collection of members. A union value only contains one member at a time while a structure value contains all of its members.

2.2.3 LFSR

LFSR means Linear Feedback Shift Registers. LFSR's used to generating pseudo random numbers.

Interfaces and Methods

LFSR package provides an interface which contains three methods namely seed, value and next. For setting the value of shift register the seed method is called with the parameter seedvalue. The value is for reading with the value method. The next method is used to shift the register on to the next value.



```
interface LFSR #(type a_type);
        method Action seed(a_type seed_value);
        method a_type value();
        method Action next();
endinterface: LFSR
Module Instantiation:
    LFSR #(Bit #(32)) lfsr <- mkLFSR_32;
```

Figure 2.7: LFSR interface and Module instantiation

CHAPTER 3

AXI4 INTERFACE

ARM introduced the Advanced Microcontroller Bus Architecture (AMBA)4.0 specifications, which includes Advanced eXtensible Interface (AXI)4.0. The AXI specifications describe an interface between a single AXI master and a single AXI slave, representing how they exchange information with each other. Memory mapped AXI masters and slaves can be connected together using a structure called an Interconnect block. The block diagram of AXI bus having single master and slave is shown in Figure 3.1.

Main components of AXI interface are

- AXI Master
- AXI Slave
- AXI Interconnect

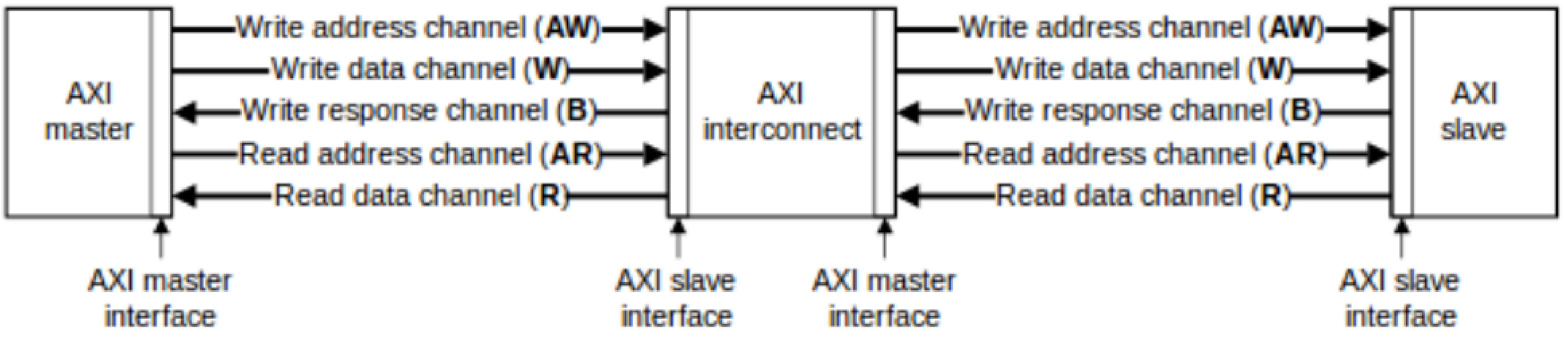


Figure 3.1: AXI Block diagram

AXI Master: Master requests the slave based on Write and Read requirements. And the responses from the slave is received here.

AXI Slave: Slave receives requests from the master based on Write and Read requests. And the response is given to master.

AXI Interface: AXI interconnect can take multiple masters simultaneously and configured with required priority scheme. It has five channels and they are:

- Write Address channel
- Write Data channel
- Write Response channel
- Read Address channel
- Read Data channel

Data is transferred from master to slave using a Write data channel. Starting Address is sent by Write address channel. A response is sent from slave to master by Write response channel. Starting address to read data is sent through Read address channel. Response Data is transferred from slave to master using a read data channel.The AXI protocol supports burst-based transactions.

3.1 Write Transaction

3.1.1 Write Channel

In write channel, Data, Address and control information is sent from Master to slave. Slave receives data and gives response to Master. Figure 3.3 shows how write address and write data channels are used in write transactions.

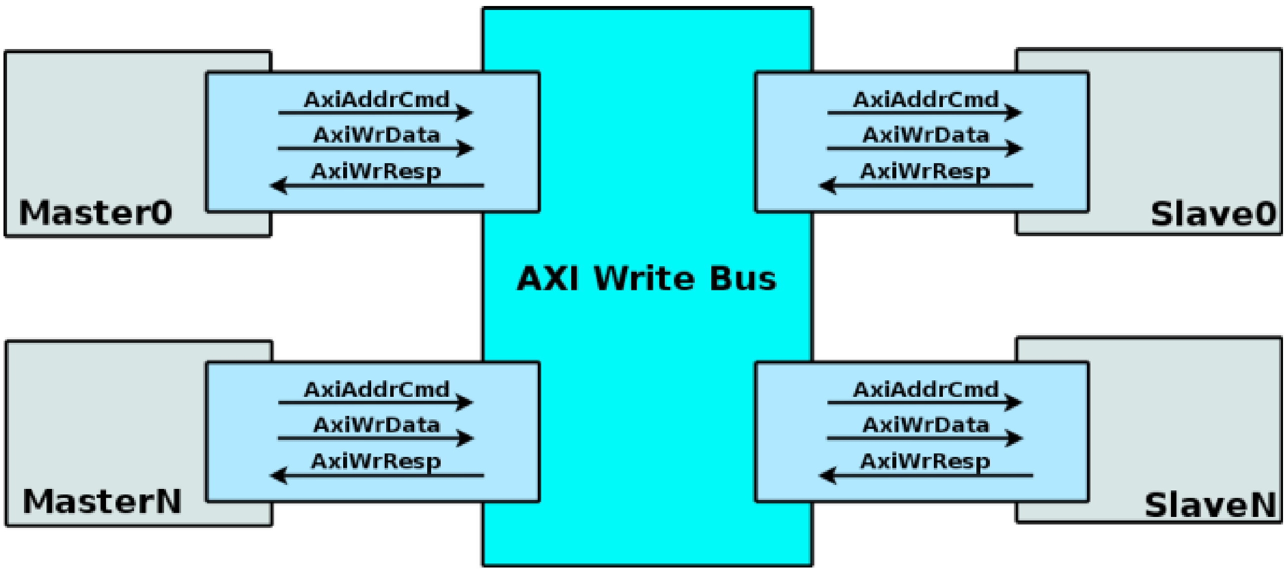


Figure 3.2: Axi Write Bus example

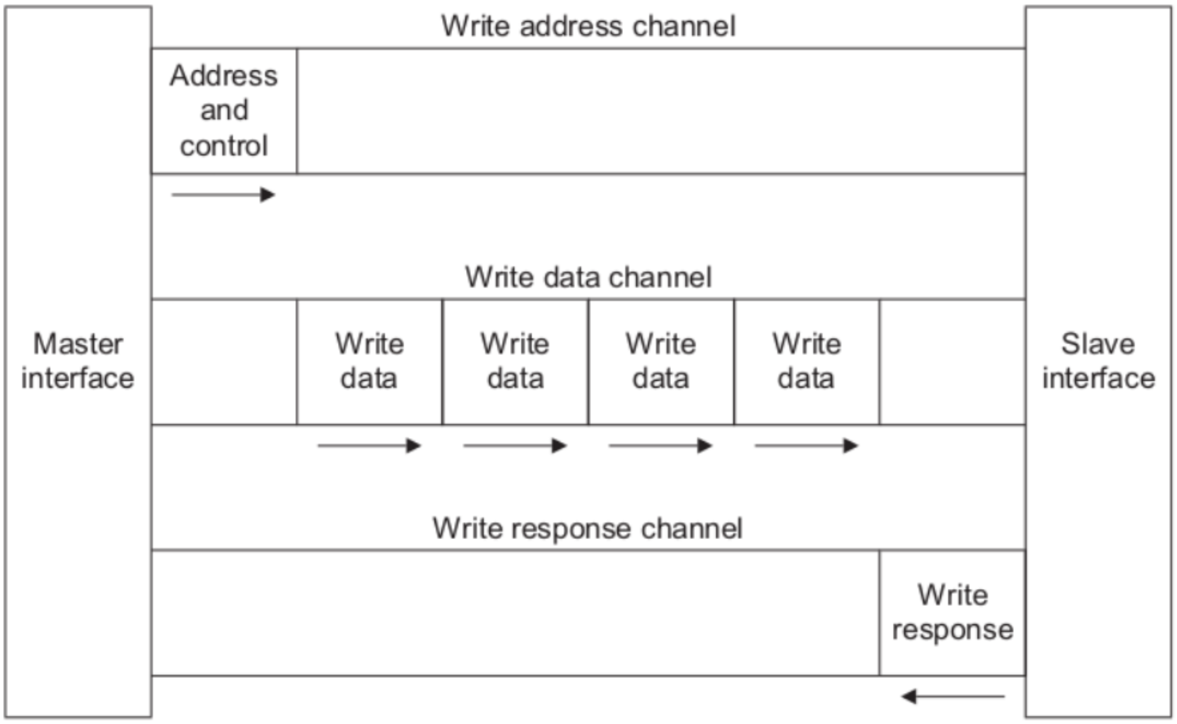


Figure 3.3: Write channel Architecture

Write Address Channel

Through write address channel, address and control information are sent from master to slave. The address is starting address of the required transaction. Control information means length, size and type of transaction. Table.3.1 describes the few signals of Write Address channel.

Table 3.1: Signal description of Write Address Channel

Signal	**Source**	**Description**
AWADDR[63:0]	Master	It gives the address of the first transfer in a write burst transaction.
AWLEN[9:0]	Master	This is burst length. This signal gives the number of transfers in a burst.
AWUSER[12:0]	Master	This is user defined signal. This has been used as page length initially. But later removed as it can't be configured directly.
AWSIZE[3:0]	Master	This is Burst size. This signal gives the size of each transfer in the burst.
AWBURST[1:0]	Master	Burst type. This tells how the next address is calculated. AWBURST: 00-FIXED 01-INCR 10-WRAP
AWID[3:0]	Master	This id tag signal. By this we can know which master is sending address and control information on this read address channel.

Write Data Channel

Through write data channel, write data is sent from Slave to Master. Data can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits. Read response indicates the status of transaction whether it is completed or not. Table.3.2 describes the few signals of Write Address channel.

Table 3.2: Signal description of Write Data Channel

Signal	Source	Description
WDATA[63:0]	Master	Write data signal has a write data bus with 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
WSTRB[7:0]	Master	This signal when high,indicates the bytelanes of data bus that has vali information. One write strobe for eight bits of write data bus in which WSTRB[n] corresponds to WDATA[(8n)+7:(8n)].
WLAST	Master	This signal is a Bool type. And will be true for last transfer from Master to Slave

Write response Channel

By Write response channel, the slave responds to master about write transactions.This is done once for each burst but not for individual transfer within burst. Table.3.3 describes the few signals of Write Address channel.

Table 3.3: Signal description of Write Response Channel

Signal	Source	Description
BID	Slave	This signal tells which master is the slave driving the Write Response signal. The value of BID and AWID must be same for the write transaction.
BRESP	Slave	This BRESP signal describes the status of write transaction.

3.2 Read Transaction

3.2.1 Read Channel

In read channel, Address and control information is sent from Master to slave. Slave drives data and response to Master. Figure 3.4 shows how read address and read data channels are used in read transactions.

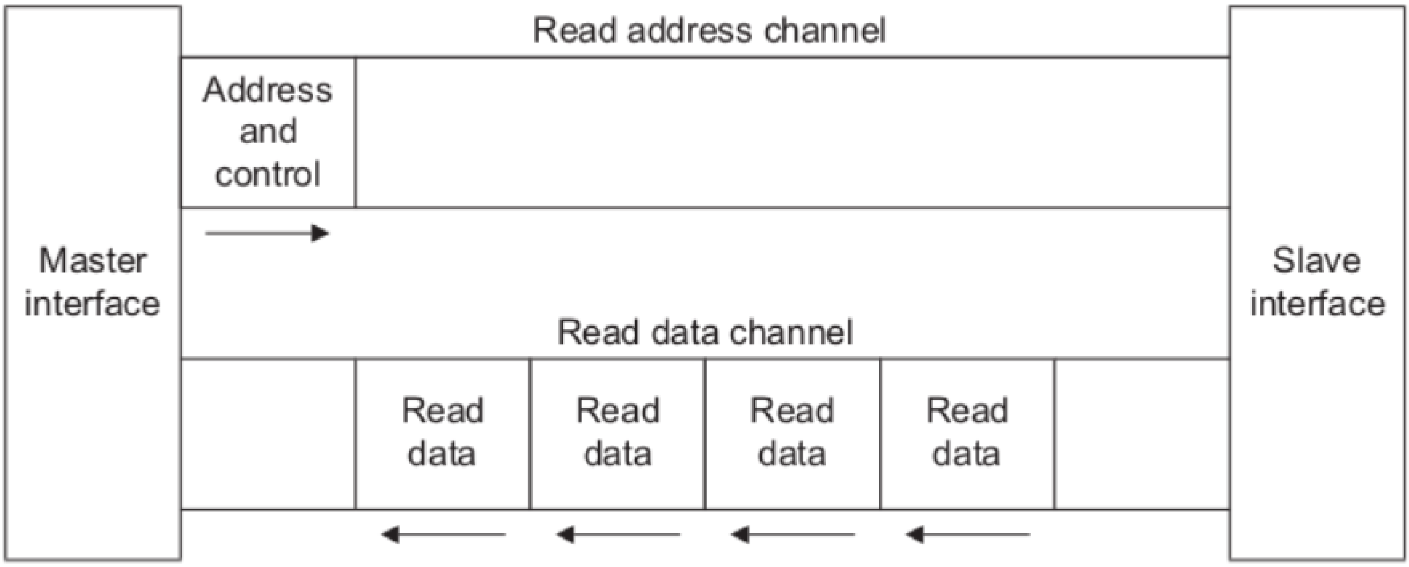


Figure 3.4: Read channel Architecture

Read Address Channel

Through read address channel, address and control information are sent from master to slave. The address is starting address of the required transaction. Control information means length, size and type of transaction.

Table 3.4: Signal description of Read Address Channel

Signal	Source	Description
ARADDR[63:0]	Master	It gives the address of the first transfer in a read burst transaction.
ARLEN[9:0]	Master	This is burst length. This signal gives the number of transfers in a burst.
ARUSER[12:0]	Master	This is user defined signal. This has been used as page length initially. But later removed as it can't be configured directly.
ARSIZE[3:0]	Master	This is Burst size. This signal gives the size of each transfer in the burst.
ARBURST[1:0]	Master	Burst type. This tells how the next address is calculated. ARBURST: 00-FIXED 01-INCR 10-WRAP

Read Data Channel

Through read data channel, both read data and read response are sent from Slave to Master. Data can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits. Read response indicates the status of transaction whether it is completed or not.

Table 3.5: Signal description of Read Data Channel

Signal	Source	Description
RDATA[63:0]	Slave	Read data signal has a read data bus with 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
RRESP	Slave	This signal gives response based on read transaction.
RLAST	Slave	This signal is a Bool type. And will be true for last transfer from slave to Master

3.3 AXI Interconnect

A system may have any number of masters and slaves. They are connected through an interconnect. Interconnect comes into play when there are multiple masters and/or multiple slaves. As this project has only one master and one slave. There will be only one interface between master and slave.

3.4 Summary

In this project data is 64 bit wide. And address also 64 bits. So, AWDATA, AWADDR, RDATA, RADDR all are 64 bit wide.

CHAPTER 4

Nand Flash Description

Each Nand flash controller can have a maximum of 2 Nand flash chips/target. A chip can have a number of LUNs. A LUN has planes which must be multiple of 2. A plane has multiple blocks. A block has pages which are multiple of 32. A page has user data bytes without including spare must be multiple of 2.

NAND Flash device has two data interfaces: synchronous and asynchronous. It highly multiplexed 8-bit bus for transferring data,address and commands. This design will deals with only asynchronous interface. A Chip/target is controlled by CE(chip enable) signal.Along with CE there are some more control signals that are used to implement NAND flash protocol.

4.1 Functional Overview

4.1.1 NAND Flash Architecture

The hardware of NAND flash interface is a low pin count device. As data, address and commands are multiplexed onto same pin and received. A highly multiplexed 8-bit bus (I/O[7:0]) to transfer data, addresses, and commands. So this reduces the pin count. There are five command pins namely CLE, ALE, CE, RE, WE for implementing NAND Flash interface protocol. Additional R/B is used to monitor device status.

Table 4.1: Signal description

Signal	**Type**	**Logic level**	**Description**
CE(Chip Enable)	Input	Active Low	An Asynchronous only signal that enables or disables one or more LUN's. Enables transfer between the host system and the NAND flash device.
ALE(Address latch enable)	Input	Active High	When ALE is HIGH, on the rising edge of WE address istransferred from I/O[7:0] into on chip address register. ALE should be LOW when address is not being sent.
CLE(Command latch enable)	Input	Active High	When CLE is HIGH, on the rising edge of WE the information in I/O[7:0] is transferred from I/O[7:0] to on chip command register. CLE should be LOW, when command is not being sent.
DQ[7:0](Data inputs/outputs) (x8)	In/Out	NA	Bidirectional I/Os transfer address, data, and commands. Data is output only during READ operations. And remaining time I/O are used as input.
RE(Read Enable)	Input	Active Low	Controls transfer of data from the NAND flash device to the host system i.e., reading data from NAND flash device
WE(Write Enable)	Input	Active Low	Controls the transfer of commands, addresses, and serial data from the host system to NAND flash device
R/B(Ready/Busy)	Output	High/Low	This signal indicates if LUN is Ready/Busy

In NAND Flash memory random access of any location in the memory is not allowed.Page is the least addressable for read and write operations.In read operation,the required page is internally transferred from the flash array to the associated page register.Then page data is accessable in a serial way byte by byte starting.The address within the active page can be freely changed to read other bytes from the given memory page.Changing page address requires to send read command.

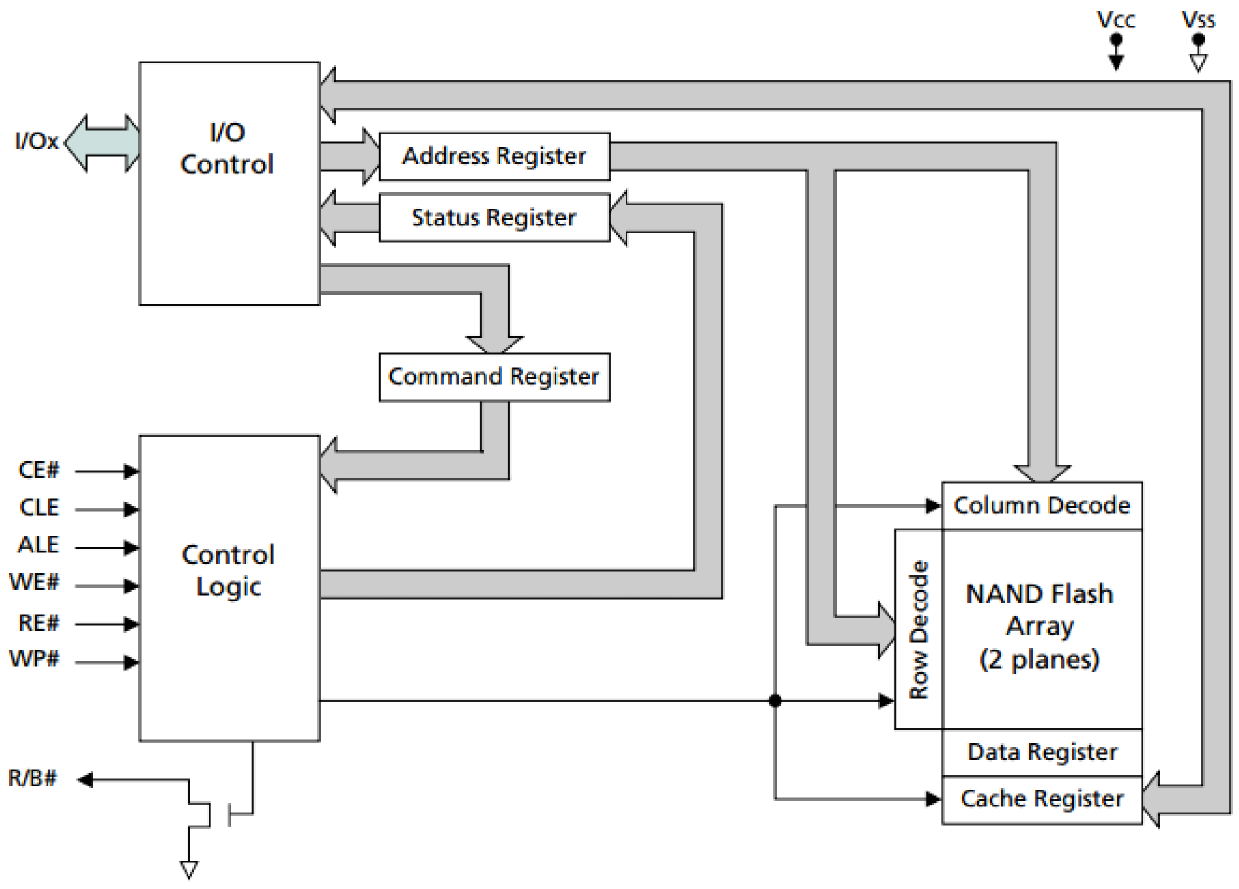


Figure 4.1: LUN functional block diagram

Data, commands and addresses are multiplexed onto the same pins and received by I/O control circuits. 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 or a column decoder to select a row address or a column address, respectively. Address is written to address register on the rising edge of WE and CE,CLE must be low and ALE should be high.

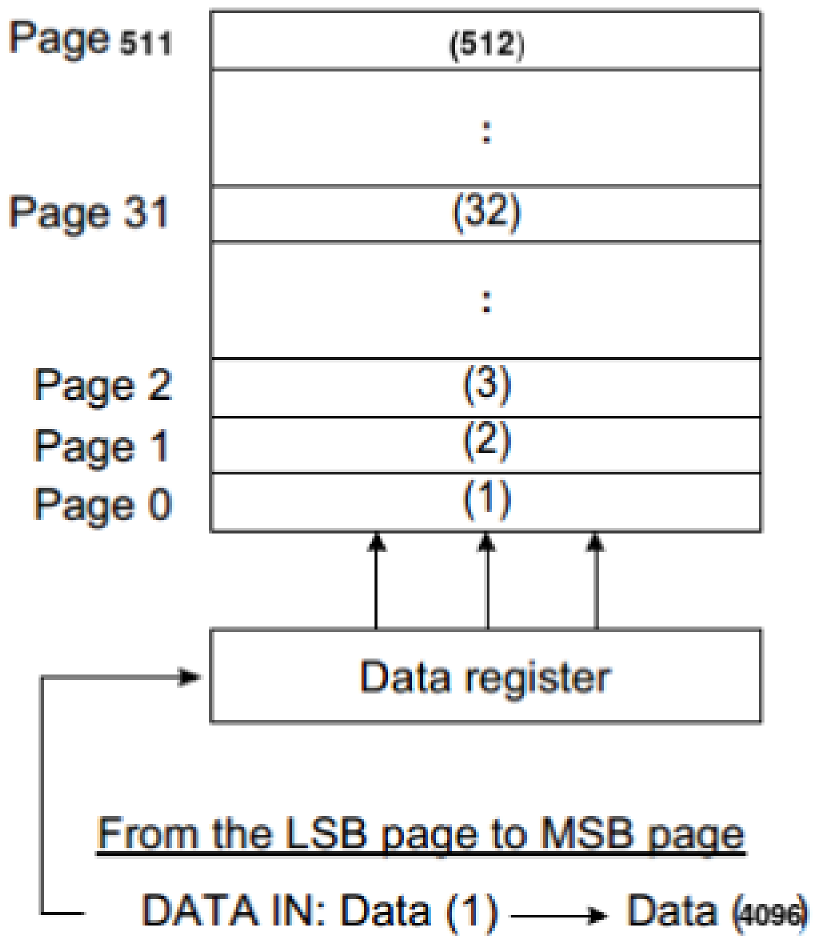


Figure 4.2: Data transfer

The data are transferred to or from the NAND Flash memory array, byte by byte (x8),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, whereas the data register is closest to the memory array and acts as a data buffer for the NAND Flash memory array operation. The NAND Flash memory array is programmed and read in page-based operations and is erased in block-based operations. During normal page operations, the data and cache registers are tied together and act as a single register. During cache operations the data and cache registers operate independently to increase data throughput. These devices also have a status register that reports the status of device operation. Data is written into data register when CE, CLE and ALE are low and the device must not be busy.

4.2 Addressing and Memory map

4.2.1 Memory Organization

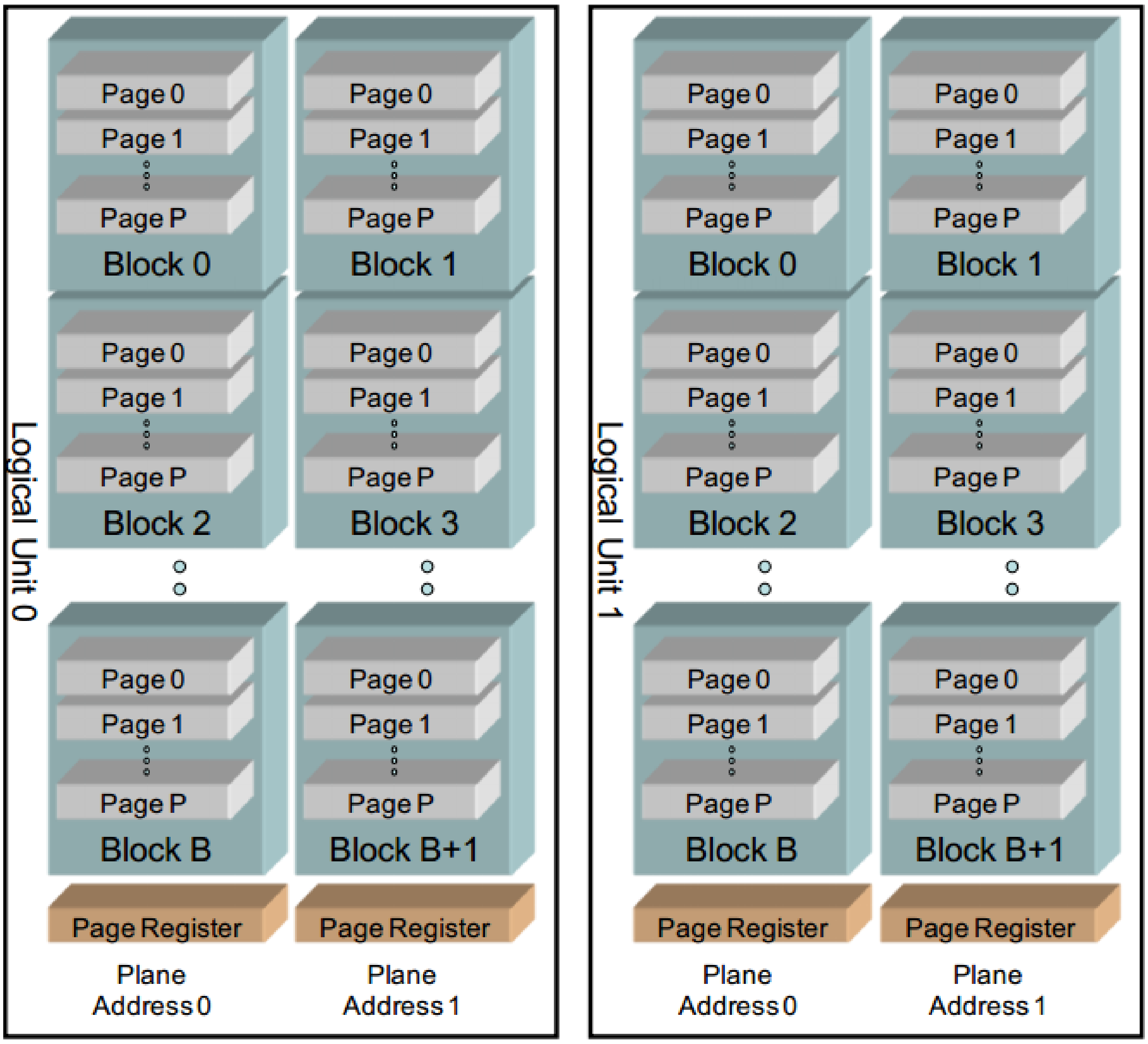


Figure 4.3: Target Memory Organization

Figure 4.3 is a target/chip. A chip is organized with one or more LUNs.A LUN(Logical Unit) is the minimum unit that can execute commands and reporting status independently. LUNs can operate on command sequences in parallel. For example, a program page operation can be started on LUN0 and before completing that operation, Read operation can be started on LUN1. By this parallelism is achieved. This is called Multi LUN operation which will be discussed in later chapters. A LUN contains Flash array. Flash array has multiple number of blocks. Each LUN has atleast one page register. Page register is used for storing data temporarily before it is being moved to a page within flash array or after it is being moved from a page in the flash array.

A LUN consists of planes. Each plane has equal number of blocks. Further parallelism can be achieved with multi-plane operations within a LUN. These operations can be used to execute additional commands in parallel. Block is the smallest erasable unit of data. Number of pages in a block should be in a multiple of 32. A page is the smallest unit for read and write operations. A page consists of user data bytes.

4.2.2 16GB NAND flash array organization

A 16GB NAND flash device has 2 LUNs. A LUN has 2 planes. A plane consists of 2048 blocks. So, a LUN has 4096 blocks. There are 512 pages per block with each page having size of 4224bytes. This memory include ECC memory. 4KB is used for data storage and the remaining 128bytes are used for ECC and other sofware functions.

Total memory of current chip from Figure 4.4 = 2*2*2048*512*4KB = 16GB.(This is excluding Spare data)

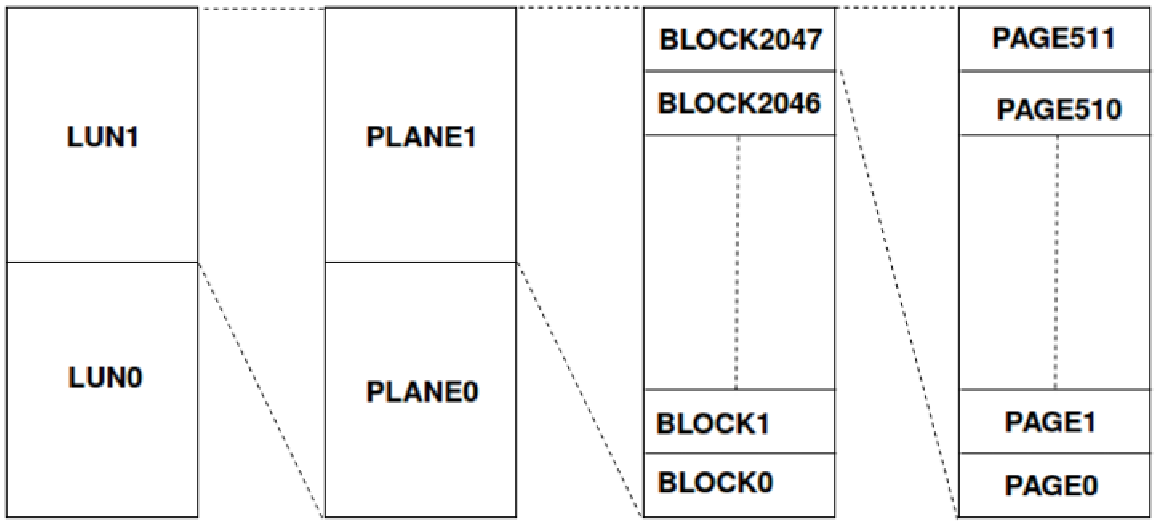


Figure 4.4: LUN flash array

4.2.3 Device Organization

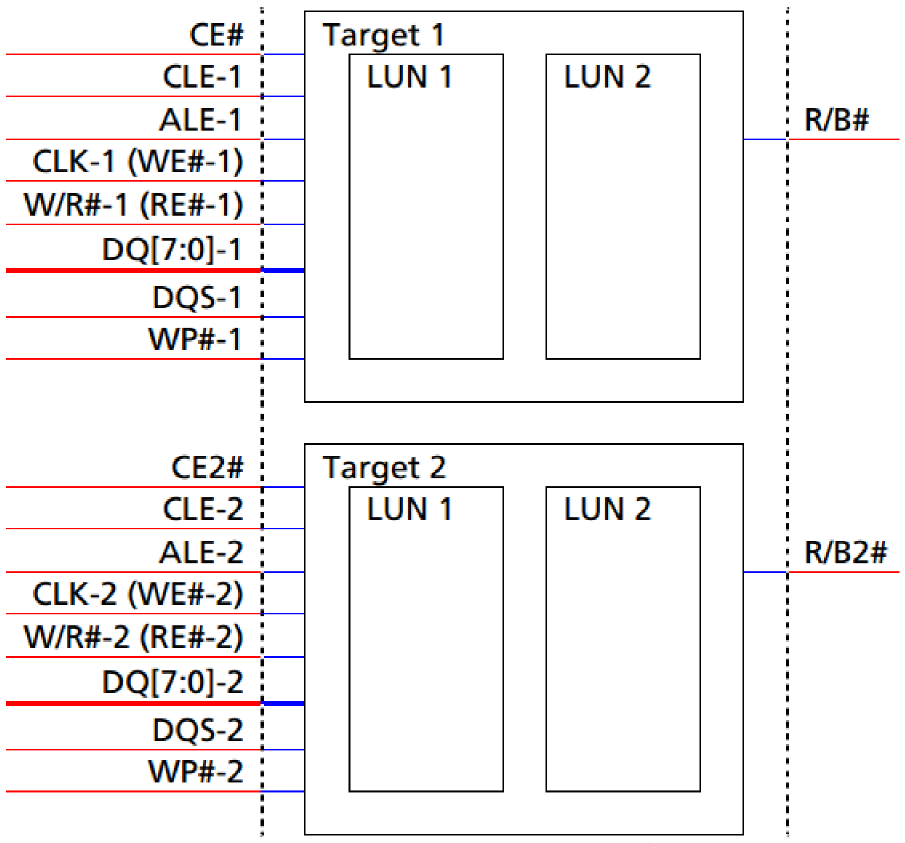


Figure 4.5: Device organization

4.2.4 Addressing

Addresses are loaded using a five-cycle sequence as NAND flash devices doesn't have dedicated address pins. WIdth of LUN is 1 bit, Plane is 1 bit, Block is 11, Page is 9 bit and Column is 12.

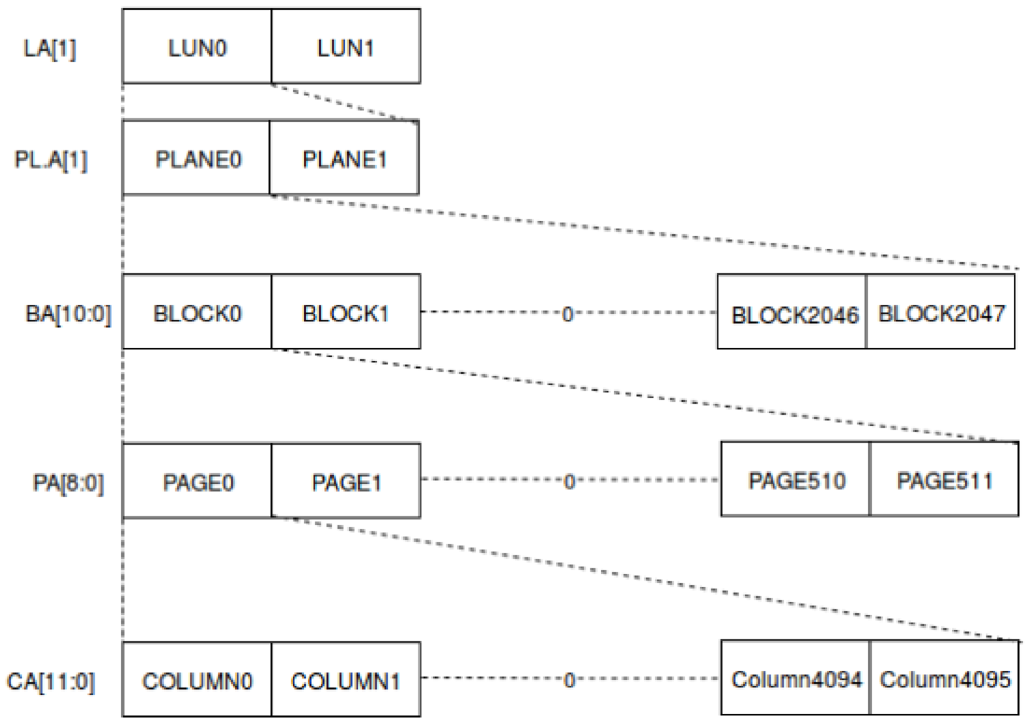


Figure 4.6: Width of address

Addressing mainly divided into two types. Row addressing and Column addressing. By Column addressing, each column can be accessed. So it has all the five addresses CA(Column address), PA(Page address), BA(Block address), Pl.A(Plane address), LA(LUN address). But in row addressing page is least addressable. So, it has PA,BA,Pl.A,LA. Here CA bits are all '0'. Operations like Erase use row addressing. The addressing is done in 8 bit address cycles. While sending five address cycles, first column is sent and then row address. Column addressing is sent first two cycles and row address is sent in next three cycles. For both addressing first address cycle has LSB bits and last address cycle has MSB bits.

Table 4.2: Address cycles

Cycle	DQ7	DQ6	DQ5	DQ4	DQ3	DQ2	DQ1	DQ0
First	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
Second	LOW	LOW	LOW	LOW	CA11	CA10	CA9	CA8
Third	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	BA6	BA5	BA4	BA3	BA2	BA1	BA0	PA8
Fifth	LOW	LOW	LA0	BA11	BA10	BA9	BA8	BA7

Multi Plane Addressing

For achieving parallelism, multi plane addressing is done. As there are 2 planes per LUN, it requires only one bit. If Pl.A ia ′0′ then Plane0 else Plane1. This Pl.A is LSB of BA. Pl.A is represented as BA0.

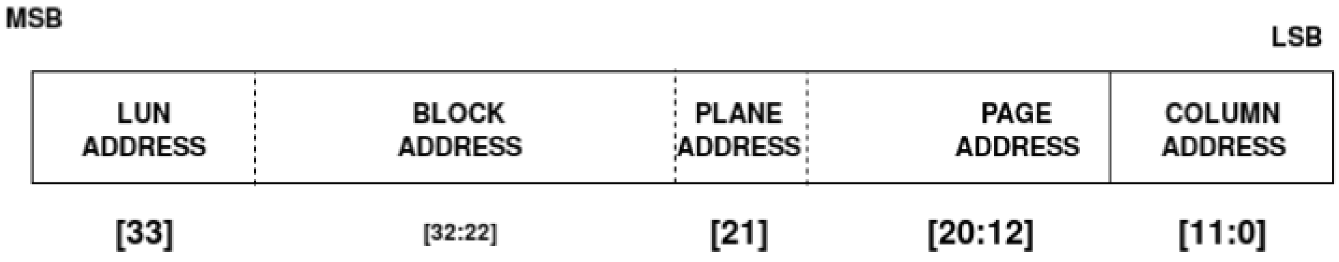


Figure 4.7: Address bits

There are some restrictions in multi-plane addressing in an LUN.

- The plane address bit must be different from any other multi-plane operations
- page address must be the same as any other multi-plane operations

Block address must be different for multi plane operations. This means there can be no multi plane operation between block 0 of plane 0 and block 0 of plane 1 in LUN0.

4.3 ONFI Commands

For designing a NAND flash controller, commands of ONFI4.0 are used. While implementing these commands, NAND flash controller willbe in different modes which are described in Table 4.3.

Table 4.3: Mode Selection

CE	ALE	CLE	WE	RE	Mode
1	X	X	X	X	Standby
0	0	0	1	1	Standby
0	0	1	Falling Edge	1	Command Input
0	1	0	Falling Edge	1	Address Input
0	0	0	Falling Edge	1	Data Input or WRITE
0	0	0	1	Falling Edge	Data output or READ
0	1	1	X	X	Undefined

Here H-Logic level High and L-Logic level Low

For all the modes above R/B should be enabled. Then only these modes can operate. But there are some exceptions like Status, Erase commands can be issued although Busy. Commands have been described in table 4.4. Each command has specific requirements. Some need row address while some need column address. The number of address cycles required for command to execute, starting command, ending command and whether they need data cycles is discussed. Both power up commnads like RESET, SET FEATURES and operational commands like READ, PROGRAM and ERASE and STATUS registers have been discussed.

Table 4.4: Command description

Command	**Command Cycle1**	**Address cycles**	**Data input cycles**	**End command**	**Acceptable while Accessed LUN is Busy**
RESET	FFh	0	-	-	YES
SET FEATURES	EFh	1	4	-	NO
READ STATUS	70h	0	-	-	YES
SELECT LUN WITH STATUS	78h	3	-	-	YES
SELECT CACHE REGISTER	06h	5	-	E0h	NO
READ MODE	00h	0	-	-	NO
READ PAGE	00h	5	-	30h	NO
READ PAGE MULTI PLANE	00h	5	-	32h	NO
PROGRAM PAGE	80h	5	YES	10h	NO
PROGRAM PAGE MULTI PLANE	80h	5	YES	11h	NO
PROGRAM PAGE CACHE	80h	5	YES	15h	NO
ERASE BLOCK	60h	3	-	D0h	NO
ERASE BLOCK MULTI PLANE	60h	3	-	D1h	NO

4.3.1 RESET operation

RESET(FFh) command is used to put a target into a known condition. When reset happens command sequences in progress will be aborted. Reset is issued irrespective of R/B value. As part of the Reset command, all LUNs are also reset. Reset may be issued when target in any state, except during power ON. When this is issued all operations are cancelled. If program or Erase is being happpening when reset is issued then those operations are cancelled although data is partially programmed or erased. Reset is issued as first command to target after power ON. SELECT LUN WITH STATUS(78h) is prohibited during power on reset. Read status(70h) can be used to know when target is ready.

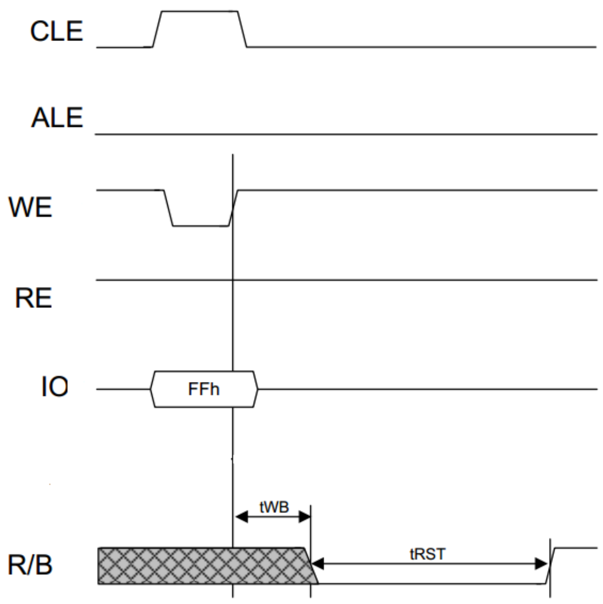


Figure 4.8: RESET Command

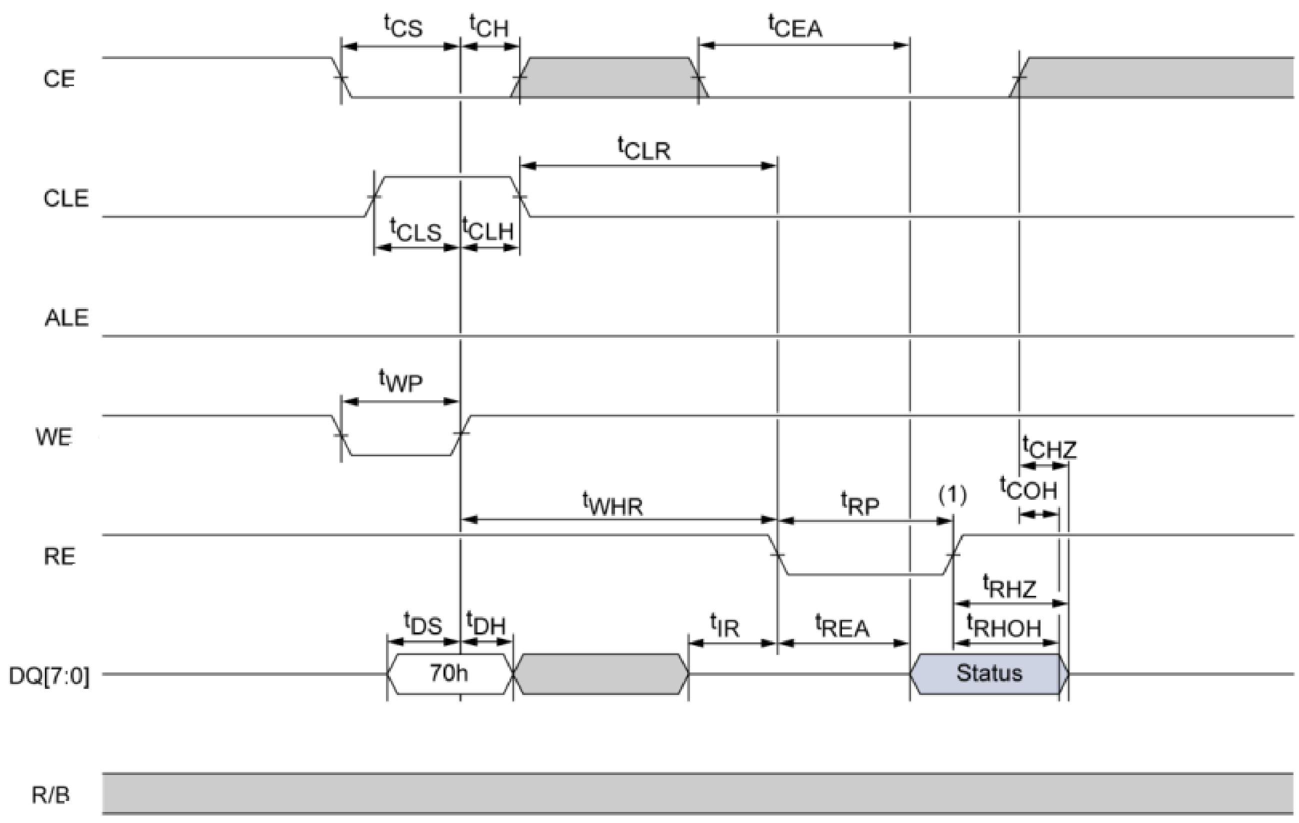
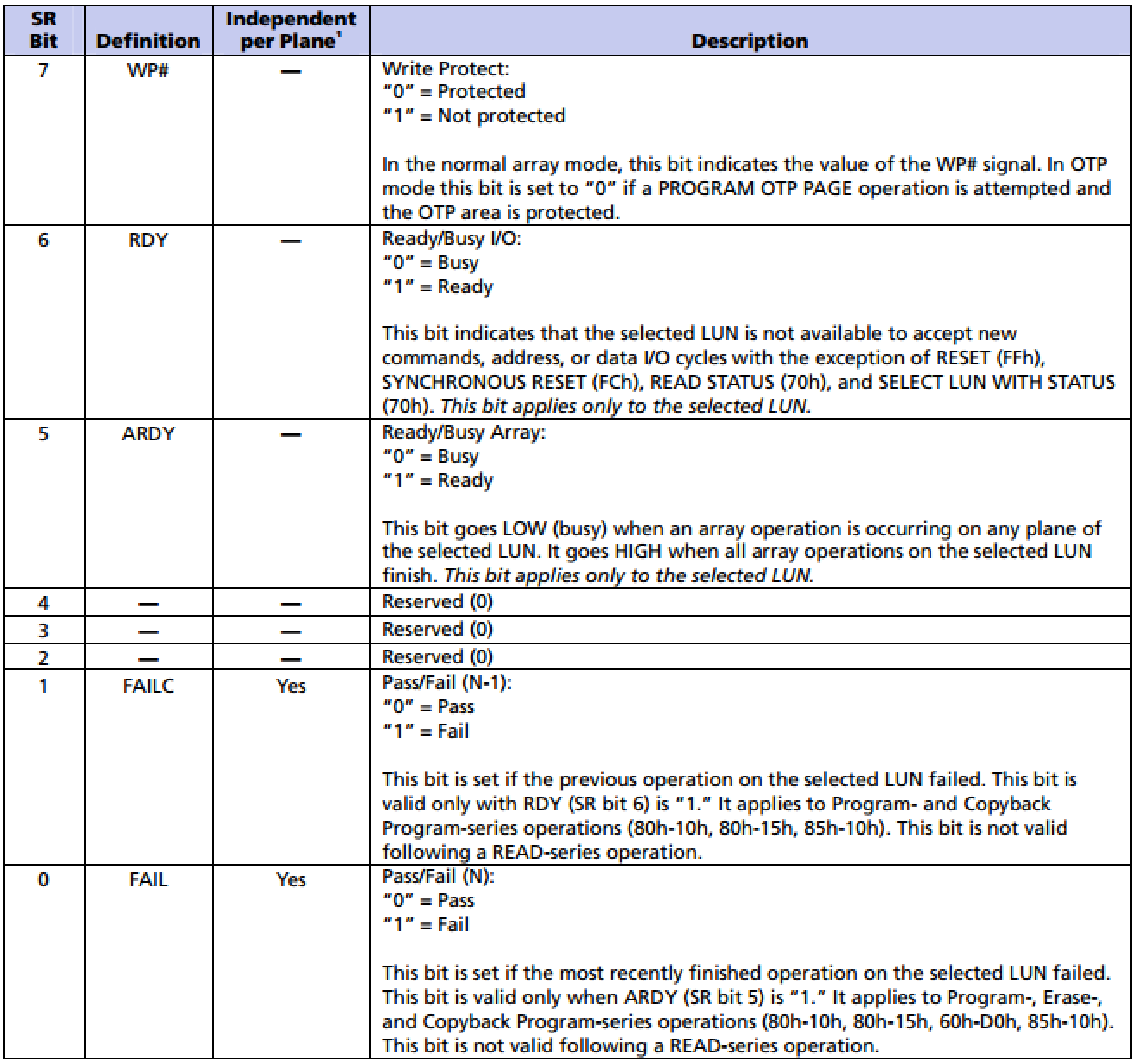


Figure 4.9: STATUS Command

Table 4.5: Status Register

Value	7	6	5	4	3	2	1	0
Status Register	WP	RDY	ARDY	-	-	-	FAILC	FAIL



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 4.10: STATUS Command description

4.3.2 Status definitions

Status of a LUN whether it is ready for next operation or busy executing current operation is checked by a 8 bit status register. Starting command of READ STATUS is 70h. No address cycles are given as it checks the status of last selected LUN on a target. Accepted by last LUN although it is busy. This is best command used to check status of LUN, if only one LUN is there per target. For devices having more than one LUN per target, SELECT LUN WITH STATUS (78h) command is used for getting status of particular LUN. 3 address cycles i.e., row address is sent. The contents of the status register are returned on DQ[7:0] after some delay. Can be issued although LUN is busy.

4.3.3 SELECT CACHE REGISTER

When SELECT CACHE REGISTER is issued the data output from the selected LUN and cache register at column address enabled. After multi plane read page commands, SELECT CACHE REGISTER command is issued to select the cache register for enabling data output. This command is accepted by a LUN when it is ready. After data output is completed, the command can be issued to other plane. After dataout of any read command like figure 4.11, 06h is starting address, 5 cycles of address cycles and end command of E0h is given after address. After some delay of end command data out starts from DQ[7:0].

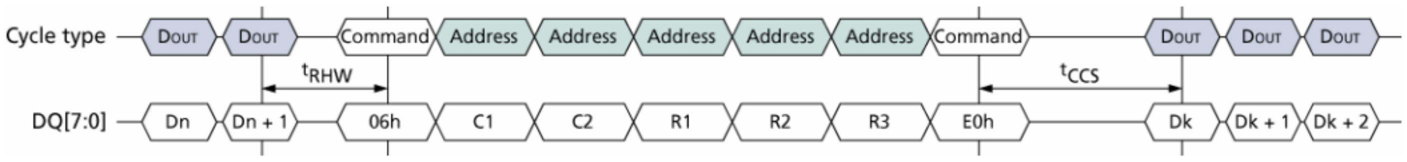


Figure 4.11: Select CACHE register Command

4.3.4 READ Operations

Page is the least addressable or readable unit. Read operations are used to copy data from NAND flash array of any plane to their respective cache registers and enable data out to host by DQ bus only.

READ MODE

The READ MODE(00h) 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.

READ PAGE

The READ page command copies a page from NAND array to its cache register and data output is enabled. LUN should be ready for issuing this command. So, check status of LUN before issuing this command. Starting command of 00h is given which is followed by 5 cycles of address. End command is 30h which indicates normal read page operation. After end command dataout enables and continues packets of data is received. This READ page command is also final command of read page multi plane operation.

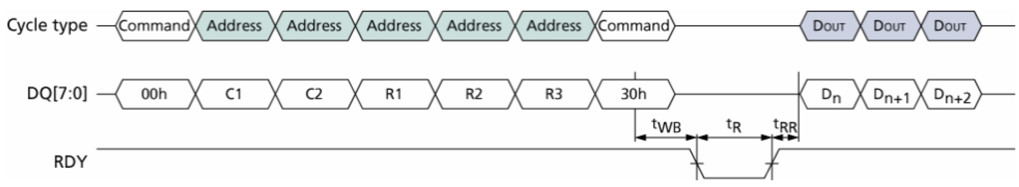


Figure 4.12: READ page command

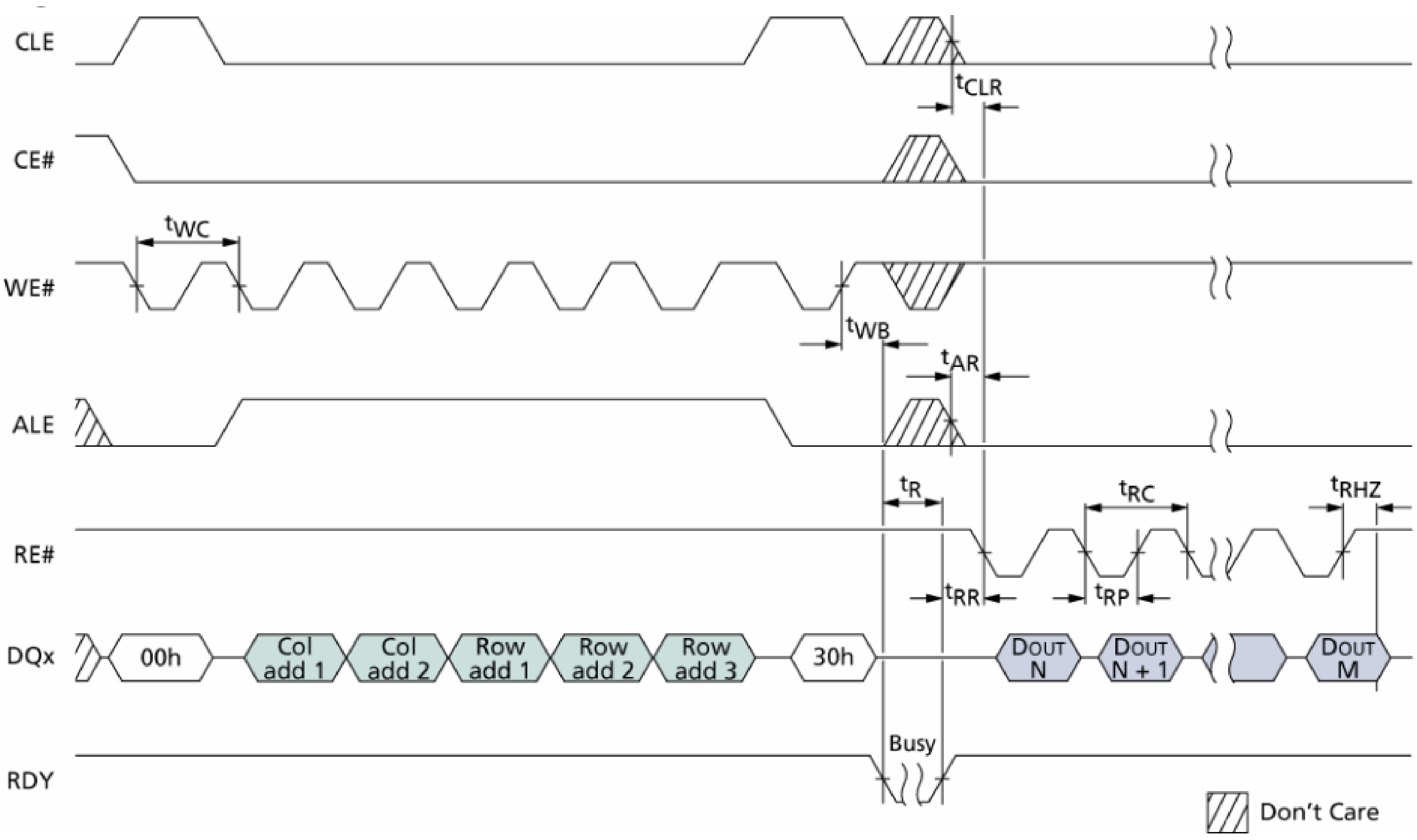


Figure 4.13: Read page command Timing diagram

Read Page Multi Plane

The Read page multi plane(00h-32h) command sends pages of different plane from NAND flash array to its respective cache register. Data is sent to cache register based on plane address. After that data out is enabled at cache register. From cache register the data is read by host through DQ[7:0] control signal. This command can be issued multiple times and ended with READ PAGE command(00-30h). It has five address cycles. As we are not dealing column level, column address is ignored(given '0'). In the figure 4.14, first read page multi-plane is issued. Then second page is read by normal READ page(00-30h) command. The data from NAND array is read into DQ[7:0] after both the commands are issued. So, continuously two pages are read.

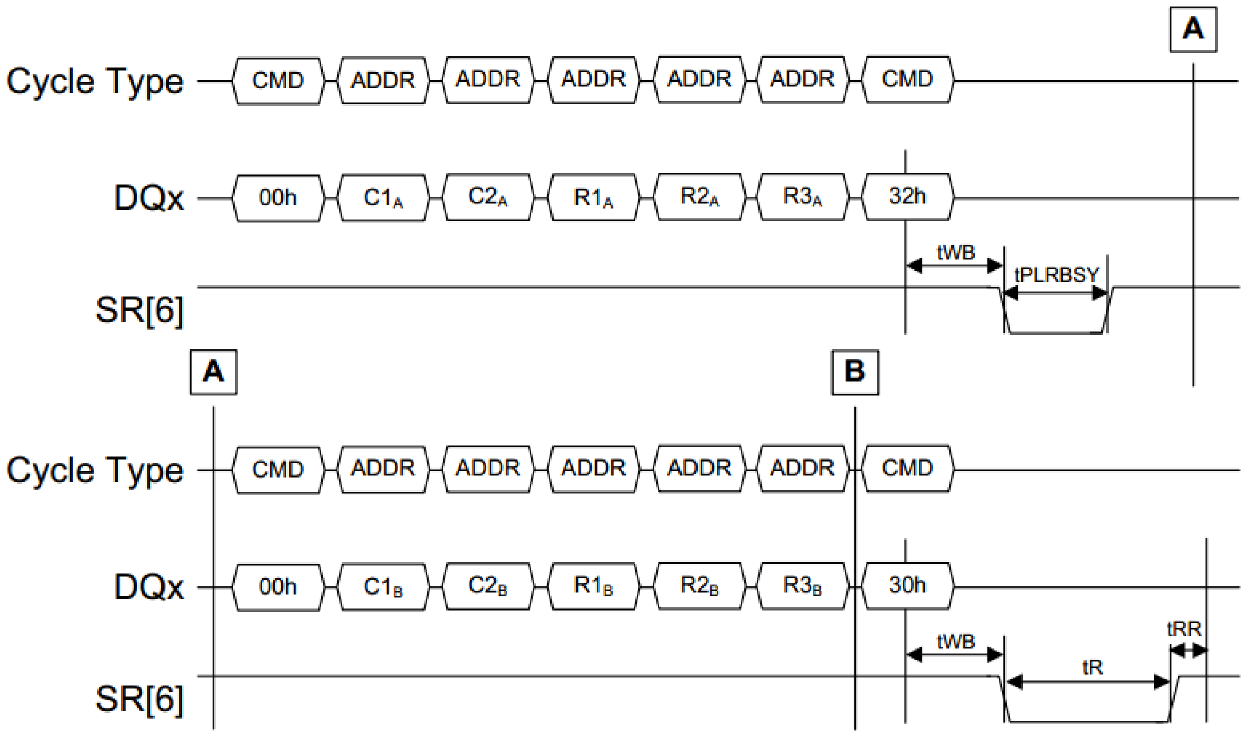


Figure 4.14: Read page Multi-Plane

4.3.5 Program Operations

Page is least addressable/programmable unit. In Program page operations data is move data from data registers or cache registers to NAND flash array. First data is sent from host and loaded into data registers or cache registers. Then thsi program operation starts. Pages are programmed or written sequentially from first page to 512th page in a block. Random page programming is prohibited within a block. As program page operation takes much time, multiple commands like Program page(80-10h), Program page multi-plane(80-11h) and Pragram page cache(80-15h) are used increasing throughput. These commands are issued when LUN is ready. During program operations, the LUN is busy. After completion of operations, LUN will be ready. This should be confirmed by checking status. RDY and ARDY must be '1'. And host must check FAIL bit to verify whether operation is success or not.

Program Page

In a single page program, starting command is 80h, then 5 address cycles are being sent. Then the data in cache register is sent to the issued page in particular block of LUN in NAND flash array according to the address. And after sending all data through DQ[7:0] operation is closed by issuing 10h. This is also final command issued in multi plane program page. Host must check status of operation by status operations.

Program Page Multi-Plane

Program page Multi plane operations(80-11h) is used for increasing system performance as by this command more than one page can moved from cache registers to different planes of NAND flash array. This is ended by either program page(80-10h) or program page cache(80-15h). First 80h is issued as starting command. Then 5 address cycles are sent. Then data cycles are sent through DQ[7:0]. After all the data cycles are being sent this operation is ended by issuing 11h. And next command may be program page cache or program page. All of the queued planes will move the data to the NAND array.

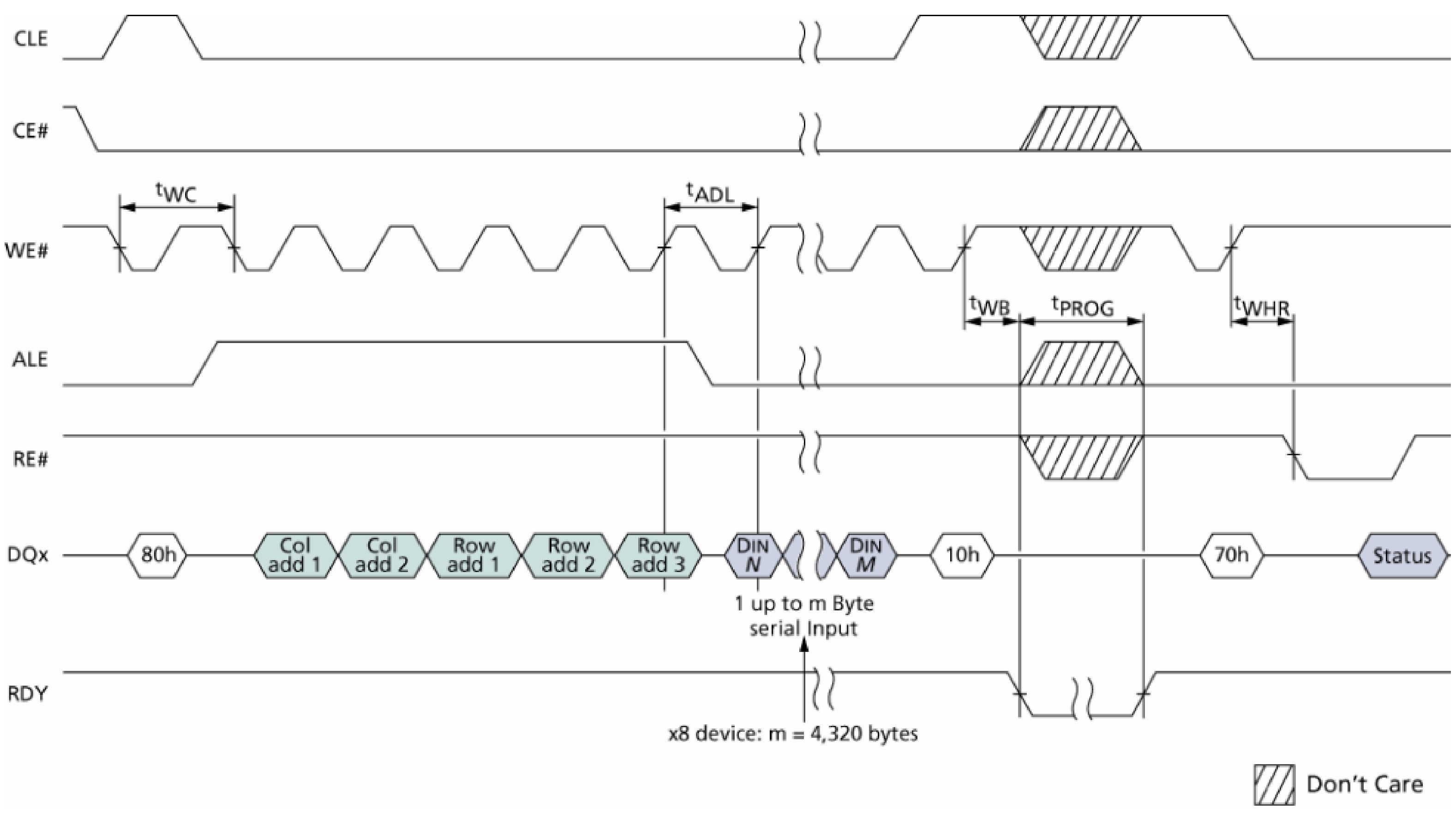


Figure 4.15: Program Page

Program Page Cache

Program page cache gives performance improvement over normal PROGRAM PAGE operations. Program page cache operation is a technique that enables host to input data into cache register and data register is used as a holding area which supplies data for programming the array. By this the cache register will be free fastly so that the next page operation can be loaded and done in parallel. With the program page cache command, the data register is used to maintain the data throughput. In a single page program, starting command is 80h, then 5 address cycles are being sent. Then the data from host is sent to cache register and then to data register to the issued page in particular block of LUN in NAND flash array according to the address. And after sending all data through DQ[7:0] operation

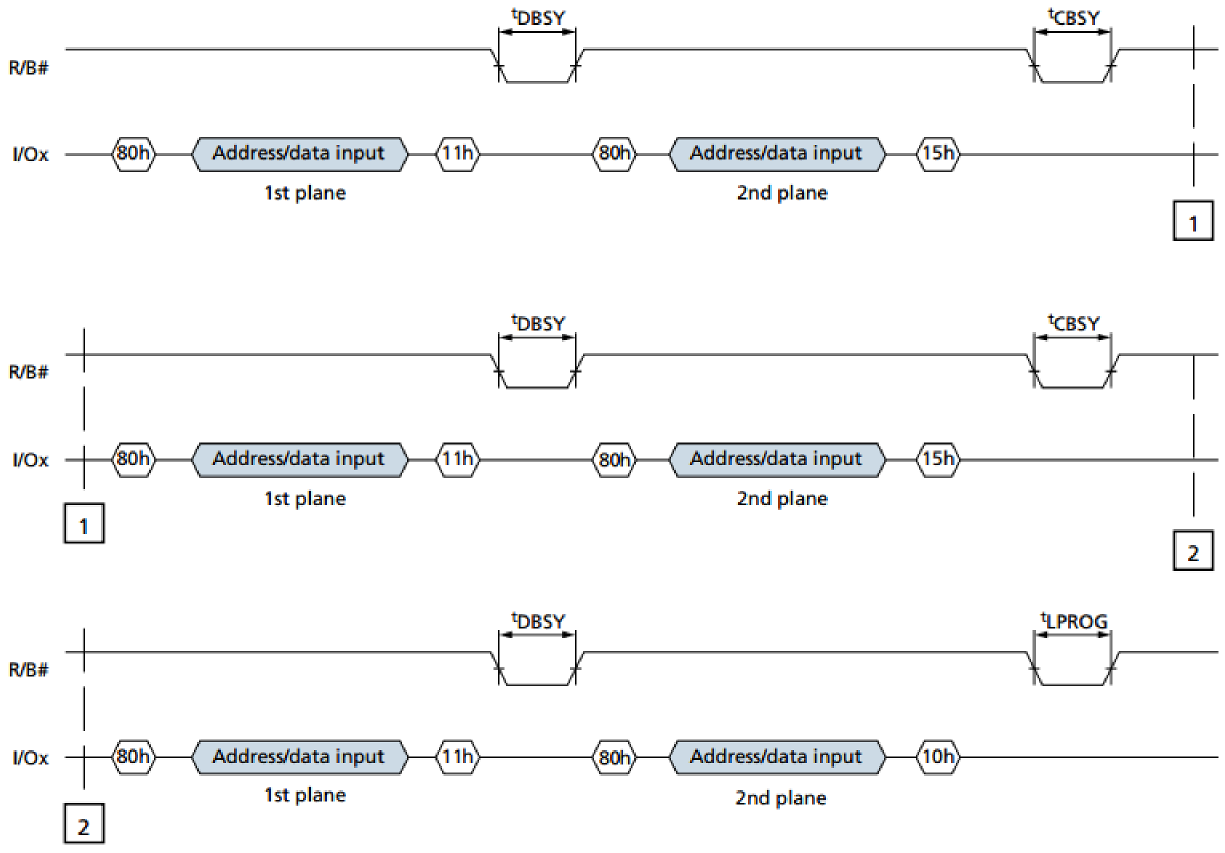


Figure 4.16: Program Page Multi Plane

is closed by issuing 10h. This is can also final command issued in multi plane program page. Host must check status of operation by status operations.

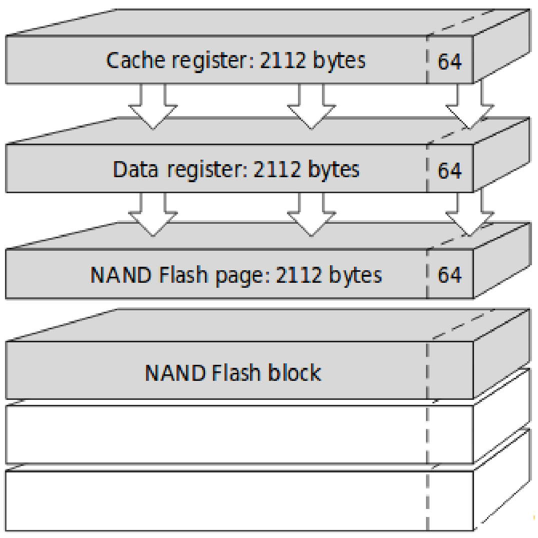

Cache register: 2112 bytes
64
Data register: 2112 bytes
64
NAND Flash page: 2112 bytes
64
NAND Flash block

Figure 4.17: Architecture of program page Cache operation

4.3.6 Erase Operations

This is row addressable operation. Only 3 address cycles of row address is given. Block is least erasable unit. A complete block is erased i.e., 512 pages or 2048KB is erased totally. After erasing the data will be all ’1’.

BLOCK ERASE

The Block erase(60-D0h) operation erases a full block. To issue a BLOCK ERASE operation, use the WE signal to clock in the ERASE when issuing starting command(60h) with CLE enabling. In Next clock three row address cycles are issued, keeping ALE high. These three address cycles include block address, page address and LUN address. The page address portion i.e., 9 lower order bits is ignored, and only the block address portion is used. After the address is issued completely, the end command D0h, which is sent in with WE while CLE is high. Then device goes busy. When the device completes erase operation, it is ready for another command. READ STATUS command can be issued at any time, even when the device is busy during the ERASE operation.

Multi plane Block Erase

The ERASE BLOCK MULTI-PLANE command queues a block in the specified plane to be erased. This command can be issued multiple times. Everytime new plane address is specified and the new plane also queued. Finally Block erase operation is given like read and program multi plane operations. In multi plane block erase, starting command is 60h, then 3 address cycles of row address are being sent. Then end command D1 is issued. Host must check status of operation

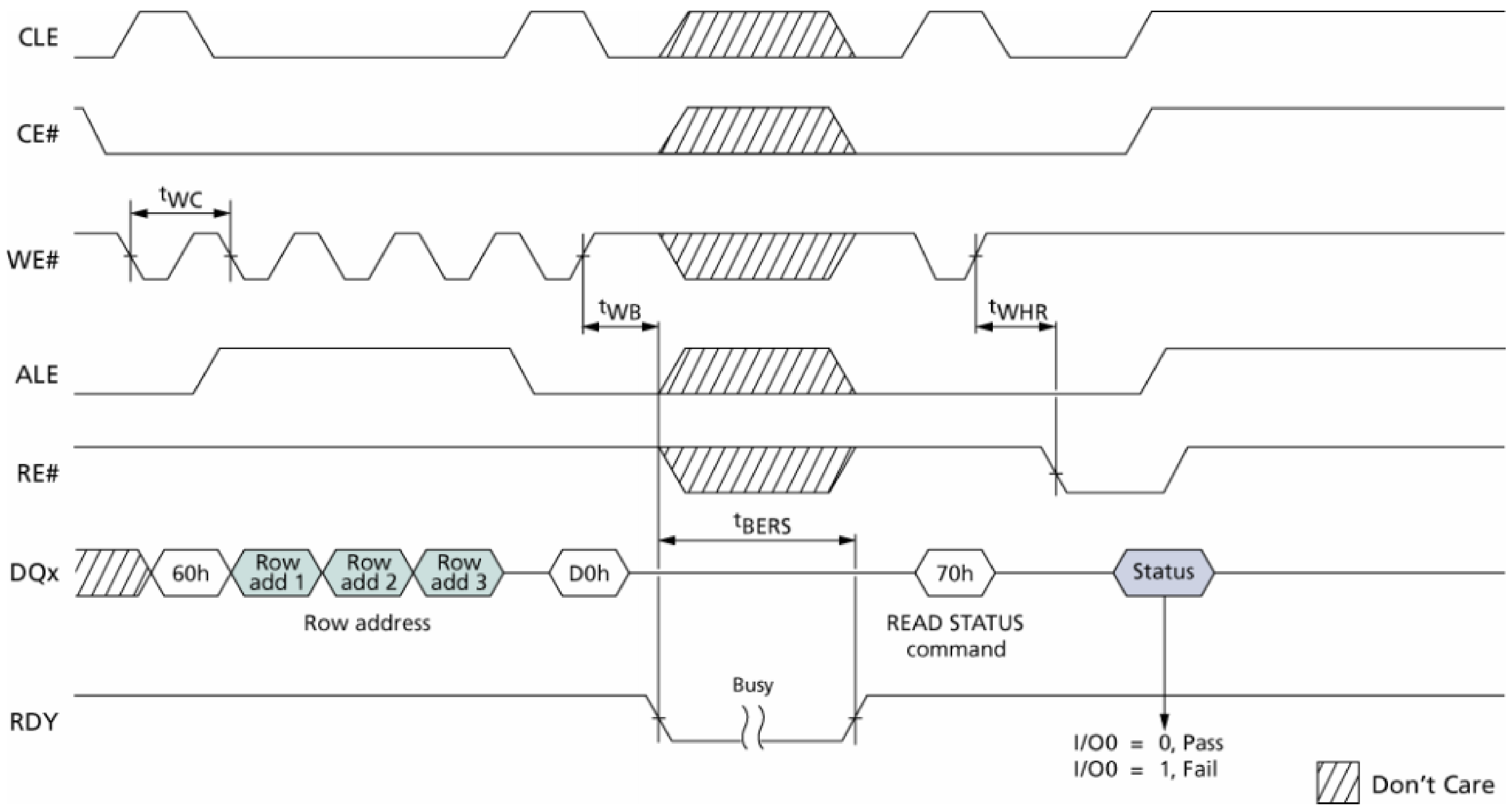


Figure 4.18: Block Erase Operation

by status operations for verifing last operation sucess or fail.

CHAPTER 5

Interleaving Commands

Although this implementation of NAND flash memory is faster than typical magnetic hard drives, there are some limitations. Using multi plane operations in a better way increases the performance of NAND flash devices. With multi plane commands PROGRAM, READ and ERASE can performance is increased.

5.1 Multi plane interleaving

Only one page of PROGRAM(READ) can be done at a time in a block. So, for PROGRAM(READ) next page, we have to wait until previous operation is done. This is not efficient way of using multi plane oprations. For example if we want to PROGRAM 4 pages, 2nd page operation must wait until 1st page completes, 3rd page must wait until 2nd page completes it operation and 4th should wait for 3rd. So, this increase latency. This consecutive page operations can divied between adjacent planes in same LUN.

In the above method consecutive page operations are done in such a way that all 4 pages are written into Block0 of plane0 in LUN0 as page0, page1, page2, page3 if we give starting address as 0. In case of multi plane operations 1st page is written into block0 of plane0 and second page is written into another plane.

But there are some restictions in implementing multiple page operations. 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. And page address must also be same. taking these into consideration, 4 pages are programmed in such a way that 1st page falls as page0 of block0 plane0 in LUN0. 2nd page falls as page0 of block1 plane1 in LUN0. 3rd page falls as page0 of block0 plane0 in LUN1. 4th page falls as page0 of block1 plane1 in LUN1. All 4 pages with same page address are written such that no two consecutive pages falls on same block. Combining these 4 blocks creates a 'Mega block' having 4 blocks.

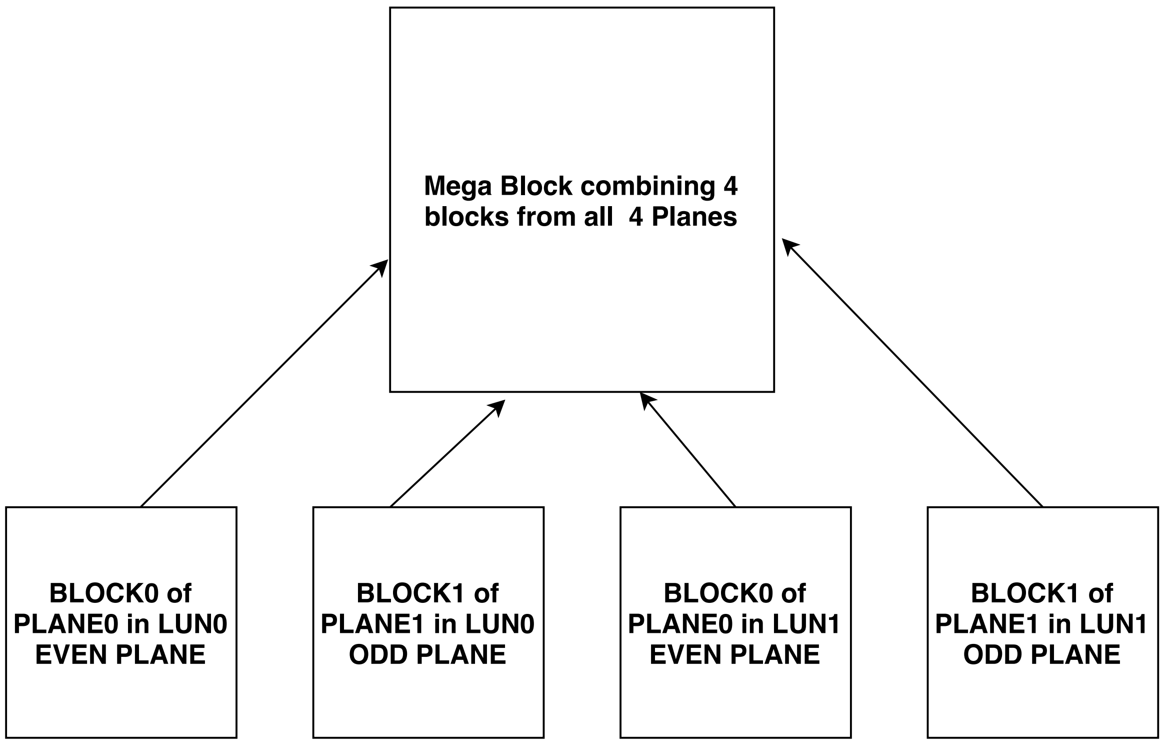

Mega Block combining 4 blocks from all 4 Planes
BLOCK0 of PLANE0 in LUN0 EVEN PLANE
BLOCK1 of PLANE1 in LUN0 ODD PLANE
BLOCK0 of PLANE0 in LUN1 EVEN PLANE
BLOCK1 of PLANE1 in LUN1 ODD PLANE

Figure 5.1: Mega Block

5.2 Mega Block

Mega block has 2 blocks from LUN0 and two from LUN1. In LUN0 Block B and Block B+1 will be used. And in LUN1 block B and block B+1 are used. Refer to

Table 5.1: Physical addressing

LUN0		**LUN1**	
PLANE0	**PLANE1**	**PLANE0**	**PLANE1**
BLOCK0	**BLOCK0**	**BLOCK0**	**BLOCK0**
000000(0)	000100(4)	100000(32)	100100(36)
000001(1)	000101(5)	100001(33)	100101(37)
000010(2)	000110(6)	100010(34)	100110(38)
000011(3)	000111(7)	100011(35)	100111(39)
BLOCK1	**BLOCK1**	**BLOCK1**	**BLOCK1**
001000(8)	001100(12)	101000(40)	101100(44)
001001(9)	001101(13)	101001(41)	101101(45)
001010(10)	001110(14)	101010(42)	101110(46)
001011(11)	001111(15)	101011(43)	101111(47)
BLOCK2	**BLOCK2**	**BLOCK2**	**BLOCK2**
010000(16)	010100(20)	110000(48)	110100(52)
010001(17)	010101(21)	110001(49)	110101(53)
010010(18)	010110(22)	110010(50)	110110(54)
010011(19)	010111(23)	110011(51)	110111(55)
BLOCK3	**BLOCK3**	**BLOCK3**	**BLOCK3**
011000(24)	011100(28)	111000(56)	111100(60)
011001(25)	011101(29)	111001(57)	111101(61)
011010(26)	011110(30)	111010(58)	111110(62)
011011(27)	011111(31)	111011(59)	111111(63)

Figure 5.1. So, 4 consecutive logical pages are present as 4 consecutive pages in one mega block but physically they are in 4 different blocks. This megablock is from our view only but physically there will be normal memory organization. The physical addressing is shown in table 5.1. So proper addressing must be done.

5.2.1 Logical address to Physical address

Consider an example of a NAND flash having 2 LUNs, 2 planes per LUN, 4 blocks per plane and 4 pages per block. So, 1 bit for LUN, 1 bit for plane, 2 bits for block and 2 bits for page addressing. Here not considering column address as that is beyond this implementation. Page is the least addressable in this implementation.

For creating mega block, consider block 'B' of plane0 and block 'B=1' of plane1

Table 5.2: Logical addressing

LUN0		**LUN1**	
PLANE0	**PLANE1**	**PLANE0**	**PLANE1**
BLOCK0	**BLOCK0**	**BLOCK0**	**BLOCK0**
000000(0)	010001(17)	000010(2)	010011(19)
000100(4)	010101(21)	000110(6)	010111(23)
001000(8)	011001(25)	001010(10)	011011(27)
001100(12)	011101(29)	001110(14)	011111(31)
BLOCK1	**BLOCK1**	**BLOCK1**	**BLOCK1**
010000(16)	000001(1)	010010(18)	000011(3)
010100(20)	000101(5)	010110(22)	000111(7)
011000(24)	001001(9)	011010(26)	001011(11)
011100(28)	001101(13)	011110(30)	001111(15)
BLOCK2	**BLOCK2**	**BLOCK2**	**BLOCK2**
100000(32)	110001(49)	100010(34)	110011(51)
100100(36)	110101(53)	100110(38)	110111(55)
101000(40)	111001(57)	101010(42)	111011(59)
101100(44)	111101(61)	101110(46)	111111(63)
BLOCK3	**BLOCK3**	**BLOCK3**	**BLOCK3**
110000(48)	100001(33)	110010(50)	100011(35)
110100(52)	100101(37)	110110(54)	100111(39)
111000(56)	101001(41)	111010(58)	101011(43)
111100(60)	101101(45)	111110(62)	101111(47)

in LUN0 and block 'B' of plane0 and block 'B+1' of plane1 in LUN1. This satisfies the conditons of multi plane addressing we discussed in before chapter.

Logical address to physical address is obtained from table 5.2 and 5.1. By this logical addressing no two adjacent blocks will not be accessed simultaneously in multi plane operations.

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:

$PhysicalAddr[5] = LogicalAddr[1]$

If PW is page width, BW is block width(including Plane width), LW is LUN width in binary. Then general mapping is shown below,

Page Address Map:

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

Block Address Map:

PhysicalAddr [PW] = LogicalAddr[0]

PhysicalAddr [PW + 1] = LogicalAddr[0] $\oplus LogicalAddr[PW + LW + 1]$

$PhysicalAddr[(BW + PW - 1) : (PW + 2)] = LogicalAddr[(BW + PW) : (PW + LW + 2)]$

LUN Address Map:

$PhysicalAddr[LW + BW + PW1] = LogicalAddr[1]$

In this thesis discussion, Logical address is given as input in testbench and by using above mapping it is converted to physical address for performing operations in a mega block. Taking the values of PW = 9, BW = 12(including plane width) and LW = 1 of this implementation,

Page Address Map:

PhysicalAddr [8 : 0] = LogicalAddr [10 : 2]

Block Address Map:

PhysicalAddr [9] = LogicalAddr[0]

PhysicalAddr [10] = LogicalAddr[0] $\oplus LogicalAddr[11]$

$PhysicalAddr[20:11] = LogicalAddr[21:12]$

LUN Address Map:

$PhysicalAddr[21] = LogicalAddr[1]$

The conversion of physical address to logical address is shown below with the above values.

LogicalAddr[0] = PhysicalAddr [9]

LogicalAddr [1] = PhysicalAddr [21]

LogicalAddr [10 : 2] = PhysicalAddr [8 : 0]

LogicalAddr [11] = PhysicalAddr [9] $\oplus PhysicalAddr[10]$

$LogicalAddr[21:12] = PhysicalAddr[20:11]$

CHAPTER 6

Design, Implementation and Simulation results

6.1 Introduction

In the previous chapters, ONFI commands have been described. This chapter will discuss about the designing and implementation of NAND flash power-ON commands and operational requests such as write, read and erase. Finally simulation results are shown.

6.2 Design of NAND Flash Controller

In full implementation of memory subsystem, host will request NVMe for some operation like read, write, erase etc through PCIe. This NVMe provides the operational requests which needs to be translated ONFI commands by Nand flash controller. The NVMe gives the number of pages, address for reading data from Nand flash cells. For write number of pages, address and data needs to be sent. For erase, block address. But this thesis deals with Nand flash controller only. So, in place of NVMe the operational requests are sent from testbench. Nand flash controller has been designed in such a way that it can write, read and erase data in Nand flash chips.

6.2.1 Overview of Full flow of commands

If any operational request such as write or read is sent from testbench(NVMe), Nand flash controller needs to wait for power ON. After power ON NFC needs to send RESET command(FFh) to Nand flash memory. Any operation must not be started before RESET as it is used for getting the target into a known state. To abort any command sequences which are in progress RESET can be used. This RESET operation must be monitored till it completes and this is can be known by Ready/Busy signal.

After target is free, SET FEATURE(EFh) command needs to be sent for setting operating timing mode of Nand flash device.

Then Nand flash memory has to be scanned for bad blocks. All the bad blocks needs to tabulated which can be used from not doing any operations in those blocks.

After the bad block table is done. Status request needs to sent for confirming whether operation is completed or not. If target is free, operational request such as read, write and erase can be given from testbench(NVMe). After completing the operation, responses are sent to testbench(NVMe).

6.3 Implementation of Nand Flash Controller

Detailed state diagrams of commands and operations such as RESET, READ, WRITE and ERASE have been discussed in this section. These state machines are implemented using Bluespec System Verilog. BSV provides higher level of abstraction than verilog. BSV has default clock. A rule in BSV is an atomic block and

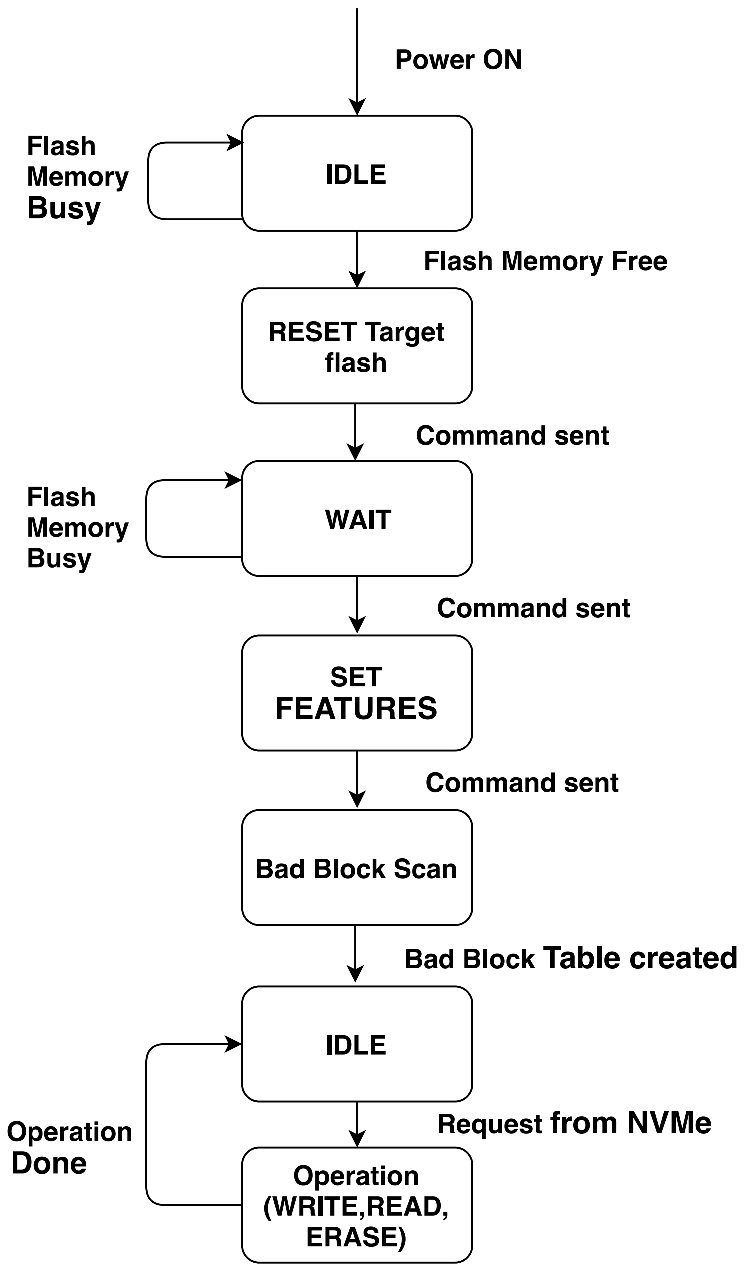


Figure 6.1: Full flow of NAND flash Controller

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. State machines are discussed in the order mentioned in fig 6.1.

6.3.1 RESET

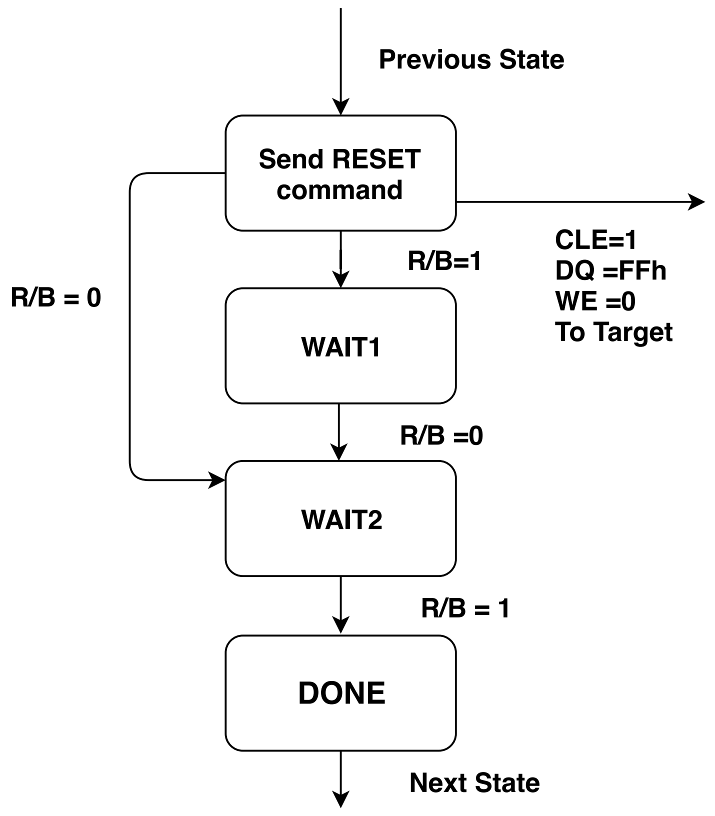


Figure 6.2: Reset

This is the first command issued immediately after power ON. When Nand Flash memory is free after power ON, RESET command(FFh) is sent into both the chips. Ready/Busy signal is observed for knowning whether NAND flash is free or not. Refer fig. 6.2.

6.3.2 Bad Block Management

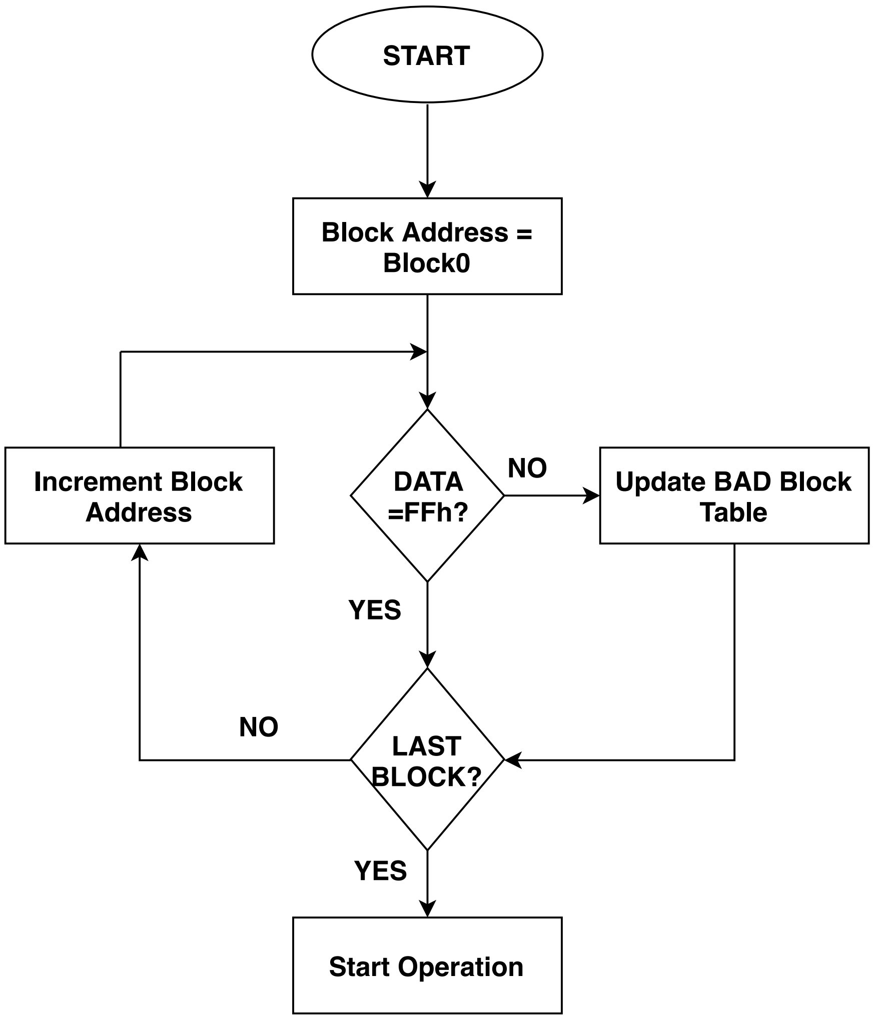


Figure 6.3: Bad block

After RESET is done, SET FEATURES command is issued for setting timing mode. Then Bad block management is done. There may be chance that initial bad blocks that are marked by flash vendor, could be erased, destroyed or used by a user. For avoiding this location of bad blocks need to be known which eliminates the chance of programming in that bad blocks. So, initially before any operation, bad block scan is done and the bad blocks which are located are tabulated. In future operations program or erase must be avoided in these bad blocks.

The host controller need to scan all the blocks from block 0 to last block using page read command and check the 1st byte in the spare area of the last page. If the read data is not FFh, the block is interpreted as an invalid block and added to bad block table. These factory marked bad blocks must not be erased or programmed. The host controller must be able to recognize the initial invalid block information and to create a corresponding bad block table for managing block avoiding program or erase in a bad block. For avoiding any erase or program operation on bad blocks, bad block algorithm is implemented immediately after SET FEATURES as shown in 6.3.

Bad block are scanned w.r.t mega block concept. 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. For bad block scan, the first spare byte of last page in every block is read and equal to FFh then bit 1 is stored and if not equal to FFh then bit 1 is stored. This is continued until we reach last block in both chips. At end bad block table with bad mega blocks is formed.

6.3.3 READ STATUS operation

Status of a LUN must be known for executing next commands if they are free. The STATUS commans(70h or 78h) are sent and status is known by reading the data I/O signal. For reading the status, RE signal is toggled from high to low. The difference between 70h and 78h is there will be no address cycles in 70h command. State diagram is shown in fig. 6.4

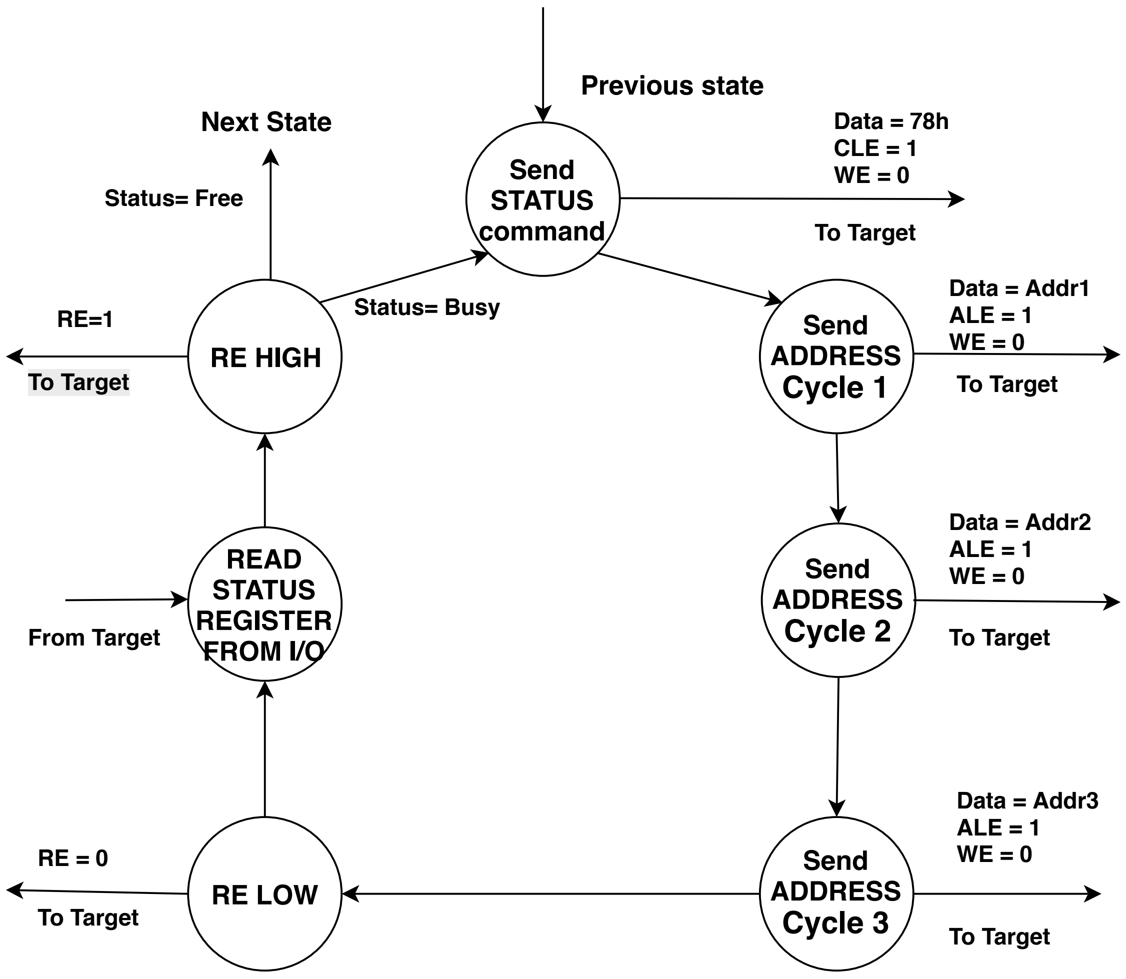


Figure 6.4: Read Status state machine

6.3.4 PROGRAM operation

In Program is higher latency operation compared to read. So, use of multi plane operations and cache operations make job simple. For a write request, starting address, pagelength(number of pages) and data is being sent from testbench(NVMe) to Nand flash controller. As AXI4 bus is used only starting address is enough as next address is incremented automatically if we give awburst as '01'. The data will then be buffered by the NFC and the buffer size is one-page size.

In programming operations all three PROGRAM PAGE(80h-10h), PROGRAM

PAGE MULTI PLANE(80h-11h) and PROGRAM PAGE CACHE(80h-15h) have same starting command, same number of address cycles but different end commands. Data is sent after sending all 5 address cycles. A full page of data is sent at a time i.e., data will be sent until column address has reached the final value in page and data is sent at falling edge of WE signal. After sending 4096Bytes of data ECC data must be sent so as to store it in the spare area of 224Bytes. So a total of 4320Bytes are sent per page. After sending all the data, end command is issued according to type of program page operation. See the flow in fig. 6.5.

Before starting PROGRAM operation, status of LUN needs to be checked. From status register, present LUN status whether it is free or busy is known and previous operation is success or failure can also be known. Logical which is given by testbench(NVMe) needs to be converted to physical address as per mega block architecture. Total of 4096 Bytes excluding spare data is sent by AXI4 bus. The data width is 64. So, totally 512 AXI requests are sent. To send the data into flash, it has to be split into single bytes starting from LSB. Few cases of program page will be discussed based on length.

Length=1: After getting write request and address from testbench(NVMe), status of LUN is read. If free then PROGRAM PAGE is executed as shown in 6.5 with end command as 10h. After this operation again status of LUN is checked to know the operation is completed and successful.

Length=2: For writing 2 pages, there are two cases. Both the pages fall on smae LUN i.e., address of first page is on even plane or both pages fall on different LUN i.e., address of first page is on odd plane. In first case i.e., even plane address case, first status of LUN is checked and if it free, then data is sent with PROGRAM PAGE MULTI PLANE(11h) command. And next page is sent through PROGRAM PAGE

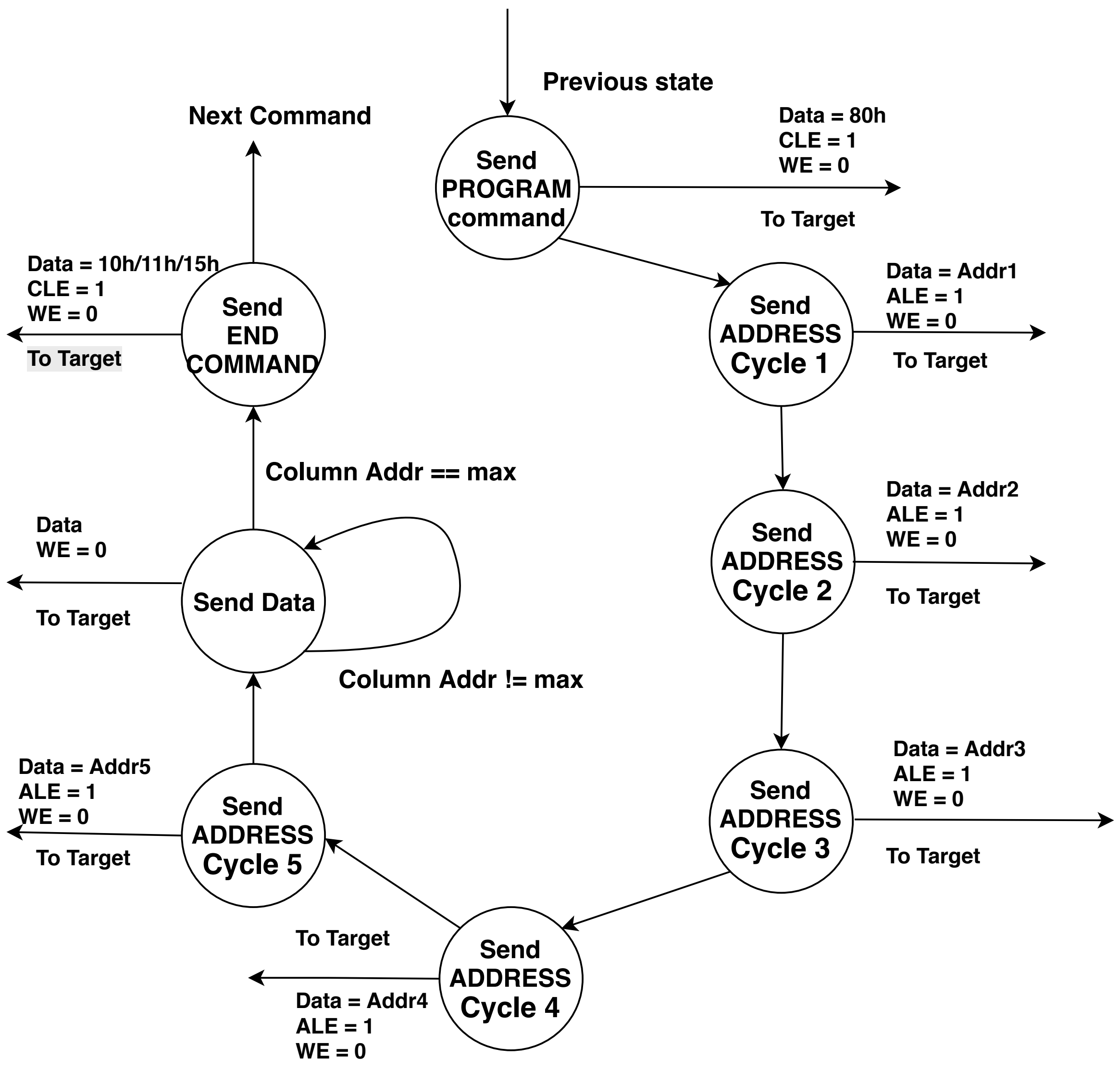


Figure 6.5: WRITE State Machine

command. Status is checked to confirm completion of operation and operation status whether it is done or fail. If any one page has failed to program, the fail acknowledgment is interrupted else a success is interrupted, which completes the operation.

For second case, both the pages use PROGRAM page commands only but will fall in different LUN's. After reading the status of starting address i.e., odd plane

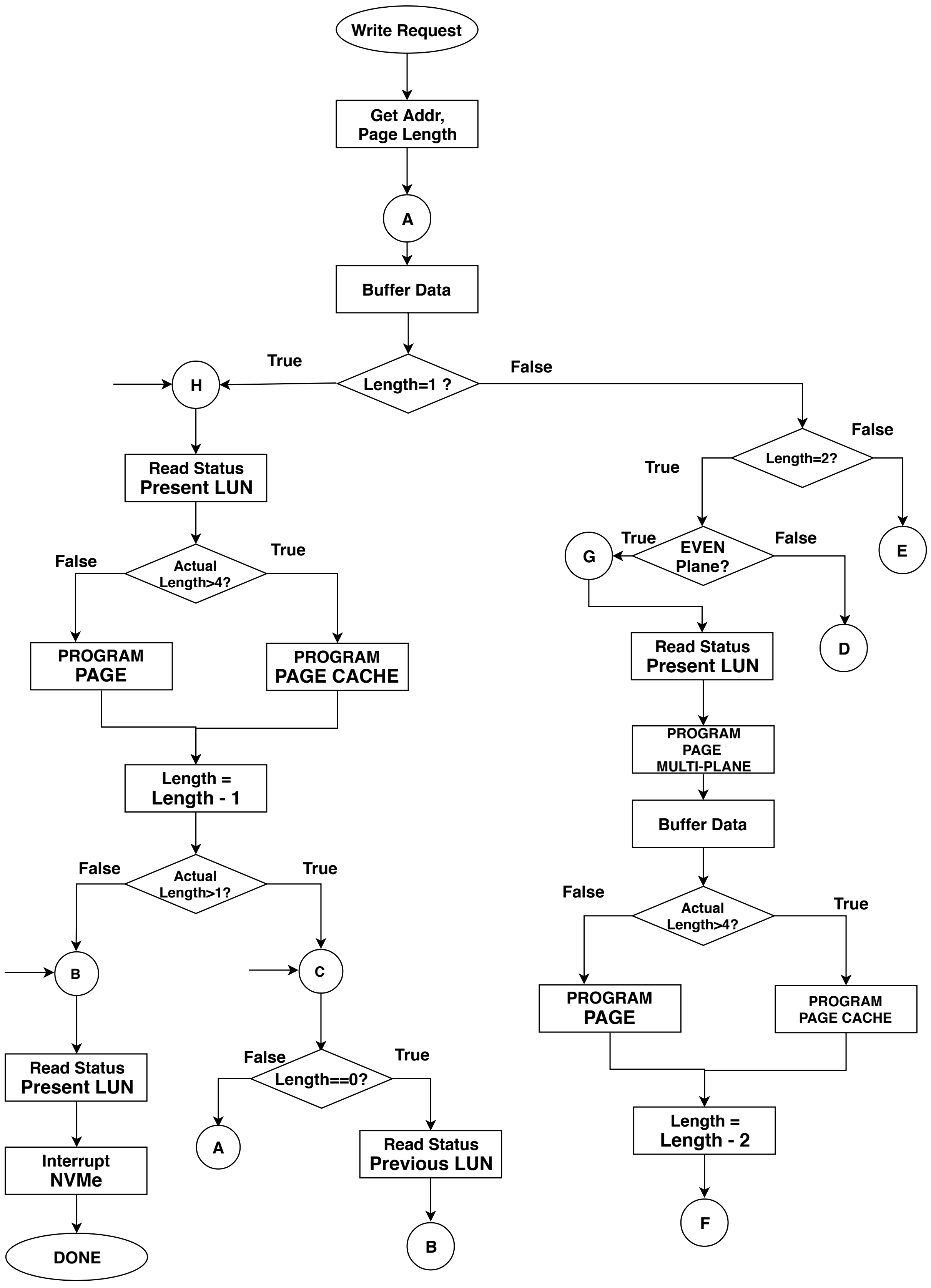


Figure 6.6: Flow of Program operation

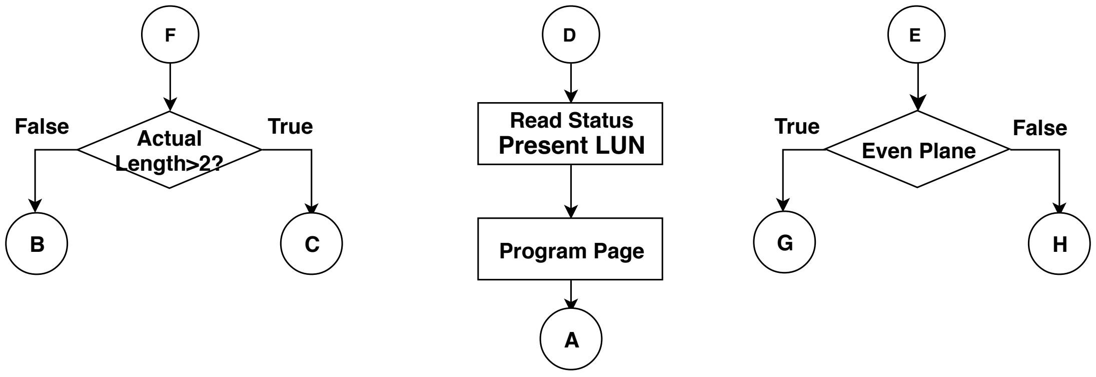


Figure 6.7: Flow of Program operation

LUN status, PROGRAM page operation is done. Then status of second(present) LUN is read and PROGRAM page operation is done. Finally after completion of both operations, status of both(present and previous) the LUN's are checked and should also check whether fail or pass. The interrupt and response is sent to testbench(NVMe).

Length=3: There are two possibilities. First address in even plane then first two pages in that LUN same as first case of length=2. One PROGRAM page and other PROGRAM page Multi-plane. And third page will fall in even plane of other LUN same like length=1.

In second case starting address may fall in odd plane. Then first page is PROGRAM page as of length=1. Remaining two pages will fall in even and odd planes of other LUN. This is same as first case of length=2.

After completing operation status is checked for both the LUN's and responses are sent to testbench(NVMe).

Length=4: For 4 pages programming, one page will fall in each block of mega block. If request falls in even plane or odd plane of one LUN. If even plane LUN, then two pages in two planes of one LUN and two pages in two planes of other

LUN. These both are as first case of length=2. Checking the status of both the LUN's is compulsory.

In second case, if starting is odd plane, the sequence of execution will be like this: Check status of present LUN, PROGRAM page operation in this LUN, check status of second LUN, PROGRAM PAGE MULTI PLANE opeartion of two pages in second LUN, check status of first LUN again, last page is PROGRAM PAGE operation in first LUN. Finally we have to check the status of both the LUN's whether operations are successful or failed.

Length greater than 4 pages: In this case atleast one LUN is guaranteed to have consecutive page program operations within a physical block(not Mega block). Hence PROGRAM PAGE CACHE is used instead of PROGRAM PAGE command for all writes of length greater than 4. Remaining data flow is same.

Proper timing has to be maintained especially multi plane operations. After sending one page the host must wait until Ready/Busy signal becomes free to send next page data. Once the whole operation is done Ready/Busy signal will assert status and returns whether operation is fail or success.

6.3.5 READ operation

The READ operation commands are used to read data from the target(NAND flash) into the corresponding plane registers. In read operations, cache commands will not be used as reading a page from flash memory into respective cache register is not a high latent operation as program operation. Adding cache commands will not contribute much since pages have to be read out from cache before any other opration starts on LUN. This time is comparable to reading from target plane to cache register. For reading from NAND flash memory, starting address and pagelength(number of pages) are required. Before sending these parameters, the address must be converted from logical address which is given in testbench to physical address. After requesting for data, the data is arrived from flash in terms of Bytes and this data is sent to testbench(NVMe) in 64 bits.

READ PAGE and READ PAGE MULTI PLANE have same states except end command. Starting command is 00h, then 5 address cycles and end command 30h for READ PAGE and 32h for READ PAGE MULTI PLANE. After end command the RE signal is toggled and for every toggle a Byte of data is received through I/O signal. Toggling read enable until maximum column address will give all the valid data. In READ MODE command(00h), previous state is status read state and present state command is 00h issued and toggling of read enable to read the data. The SELECT CACHE REGISTER command is same as READ MODE except that this need not be preceded by a status operation and can be used to read the data from the specified cache register. State diagram is shown in 6.8. Before sending the read commands, status of LUN's must be checked.

Length=1: First status is checked and then READ PAGE(00h-30h) command is sent. Once end command is sent, READ MODE command is sent for reading data from cache register. The interrupt is provided and data can be read by testbench(NVMe).

Length=2: There are two cases. If starting address falls in even plane then two pages are to read from same LUN. So, first status of LUN is checked and then READ PAGE MULTI-PLANE is issued and then READ PAGE command is issued for second page. The status of LUN is checked and if it is free READ MODE(00h) command is issued for reading first page and the data is buffered and sent to the

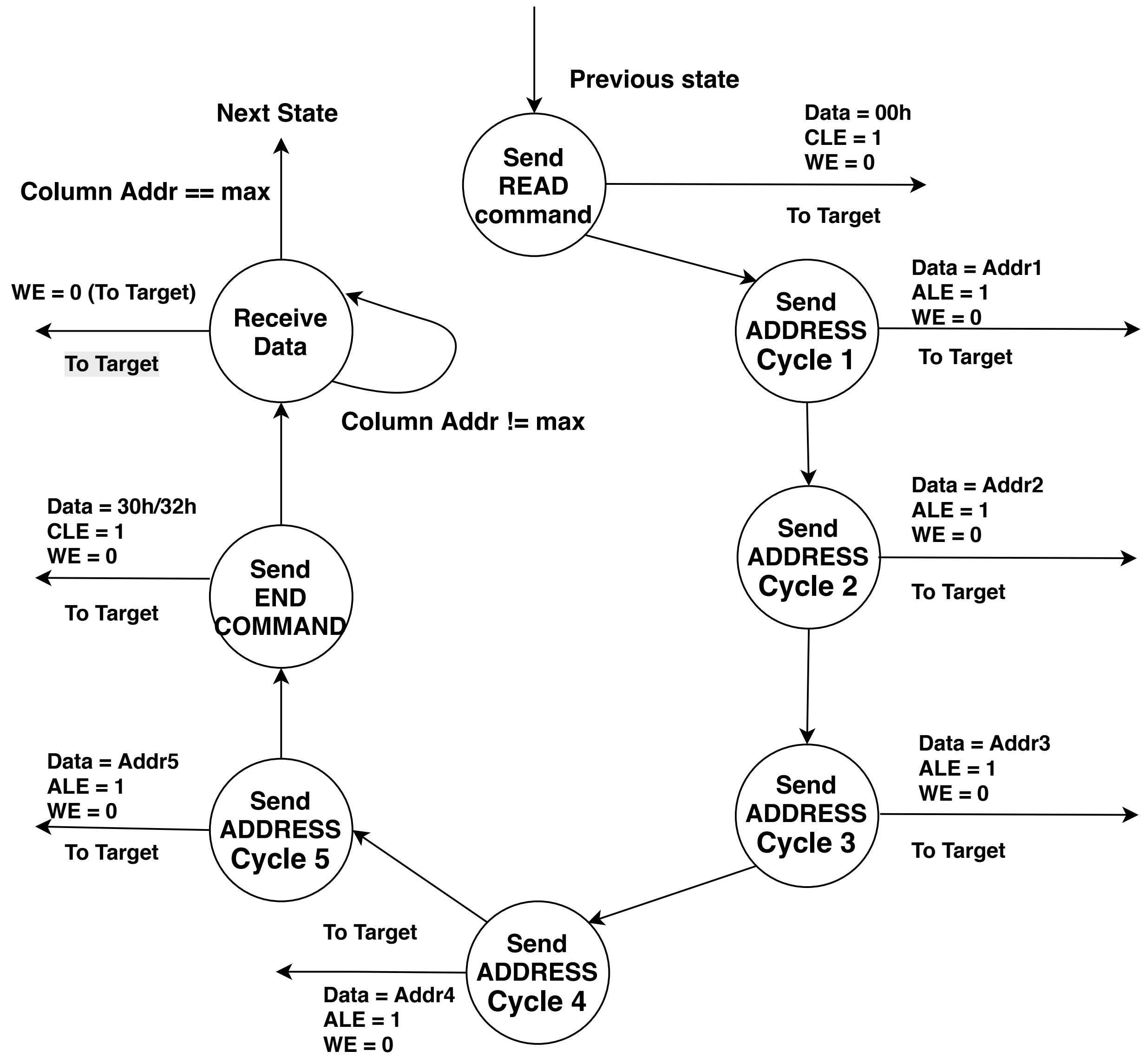


Figure 6.8: Read State Diagram

testbench(NVMe). After reading data, SELECT CACHE REGISTER command is issued for reading second page, data is buffered and sent to testbench(NVMe).

In second case, starting address is odd plane. So 2 READ PAGE commands are issued. READ command is sent after checking status of first LUN. Same for second LUN. The status monitoring is shifted to first LUN and once complete data is read out using READ MODE. Then status of second LUN is also checked and if it is free, data is read out again through READ MODE. After that interrupt NVMe reads the data.

Length=3: Here also two cases. If starting address falls in even plane then two page request and one page request are sent. First status of two LUN's are checked. After that both the requests are issued and status is checked for the LUN in which two page read is happening. Once this is done the data is read from the first cache register(first page 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. Then status of second LUN is checked for third page and when free data is read out. In second case first page falls on odd plane and remaining two pages fall in same LUN. This is done same as before.

Length=4: In first case, if starting address falls in even plane then 2 two page read reqquests. Status is read out, if LUN is free then two page request is sent.

If first page falls in odd plane, first single page read request on first LUN, two pages read request on second LUN and last page read request on first LUN. First one page read request is executed on first LUN when it is free and status of other LUN is also checked.for sending two pageread request when it is free. The status of first LUN must be monitored and when ready data must be read out of cache register before issuing any more commands on this LUN. after this last page request is queued. Now switches to second LUN, read the status of the LUN that was handling the two page read. Once free, the data is read out. Two pages are read. After this LUN is switched again to first LUN and data is read out from cache register. NVMe reads out data then request completes. For pagelength more than 4, the request is split based on even or odd plane into a maximum of two page requests and data is read out after checking for status. Ready/Busy pin shows whether operation completed or not. after last request only it will become ready.

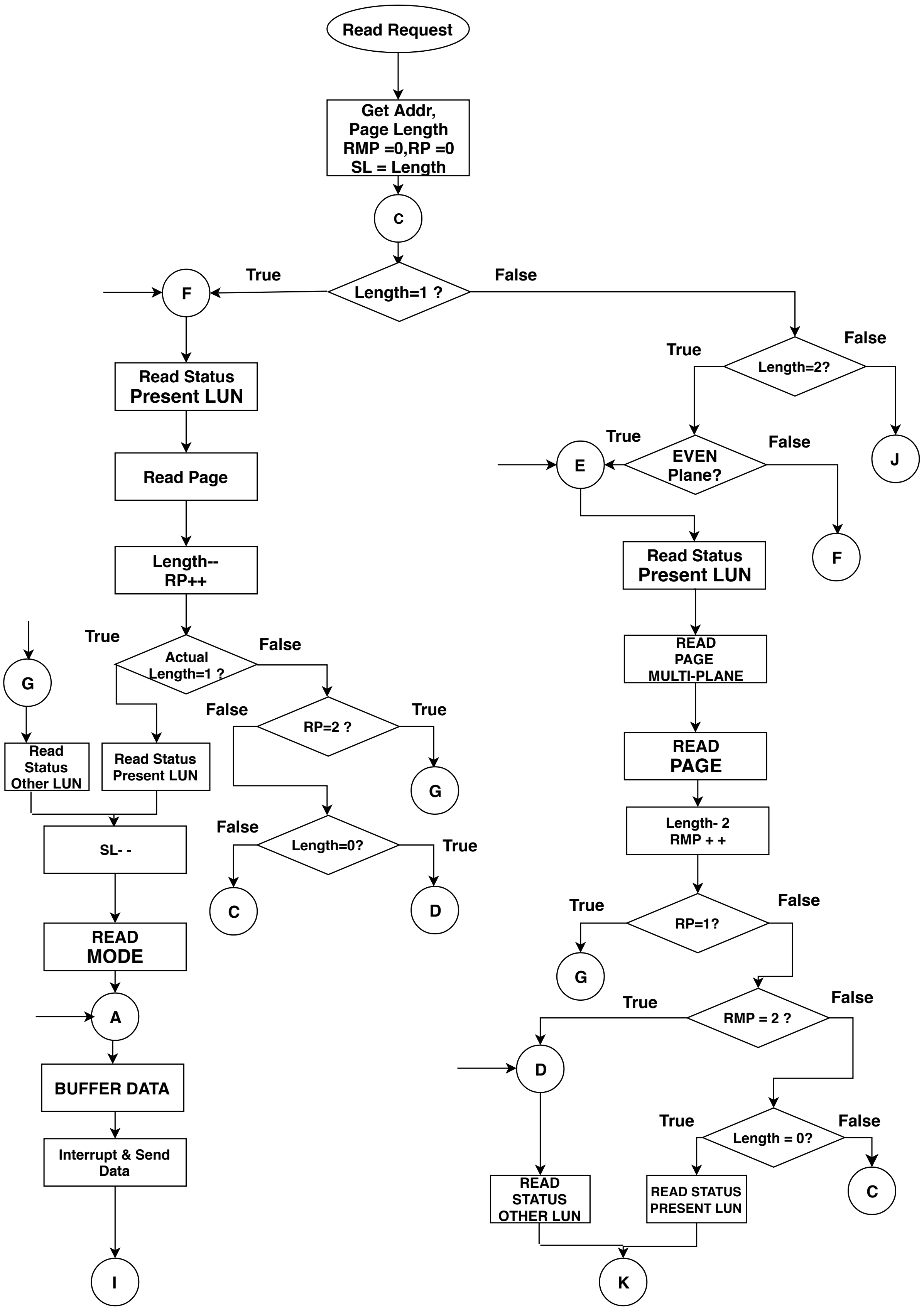


Figure 6.9: Flow of READ operation

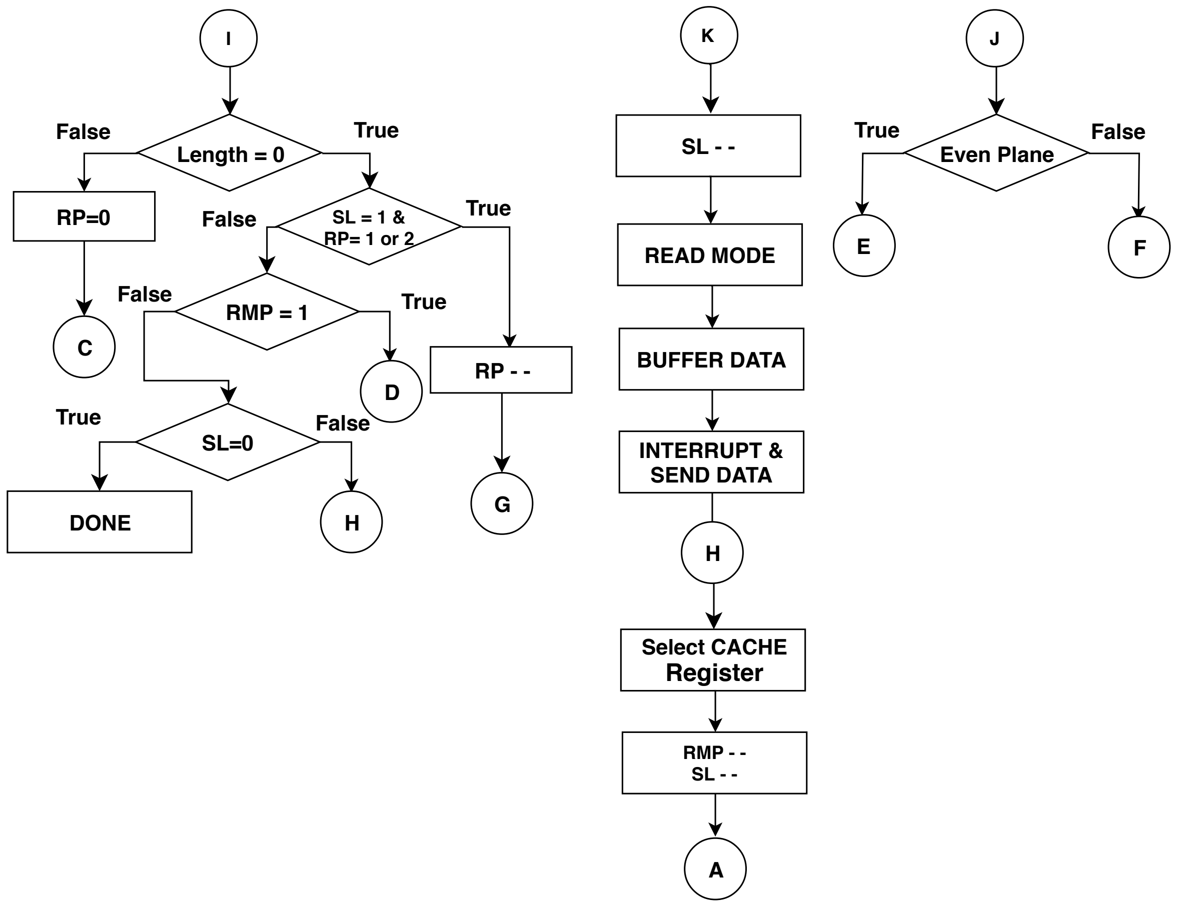


Figure 6.10: Flow of READ operation

6.3.6 ERASE Operation

Block erase needs only 2 address cycles as page address and column address is ignored. ERASE BLOCK(60-D0h) and ERASE BLOCK MULTI -PLANE(60-D1h) operations are same except end command. The block address is converted to logical address which is one block out of 4 blocks in a mega block. As mega block has 4 blocks, address of each LUN needs to be known and status is checked. Once block in LUN0 is free, BLOCK ERASE MULTI PLANE command is issued. Next the BLOCK ERASE command is issued for next block address. After this, status of next LUN is checked and same commands are sent with next two block addresses.

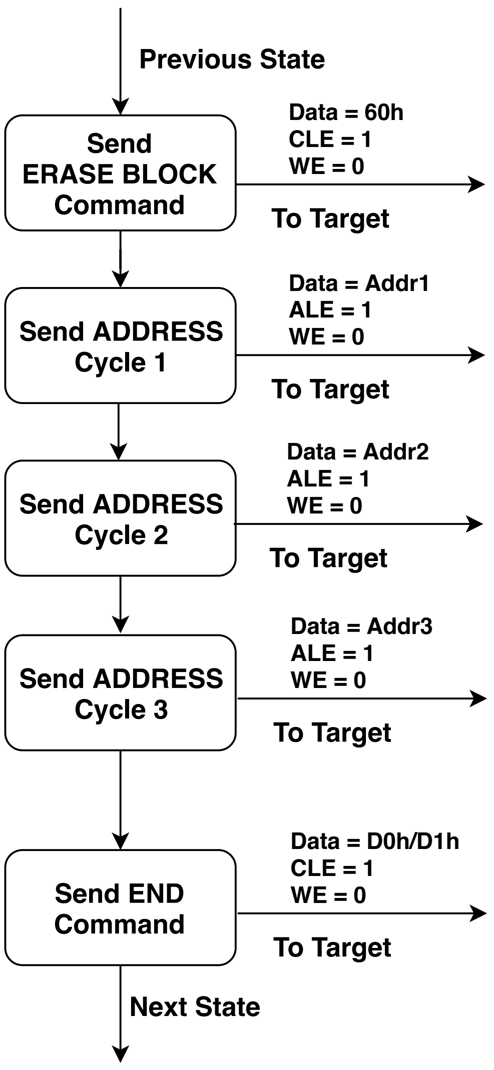


Figure 6.11: Erase State Diagram

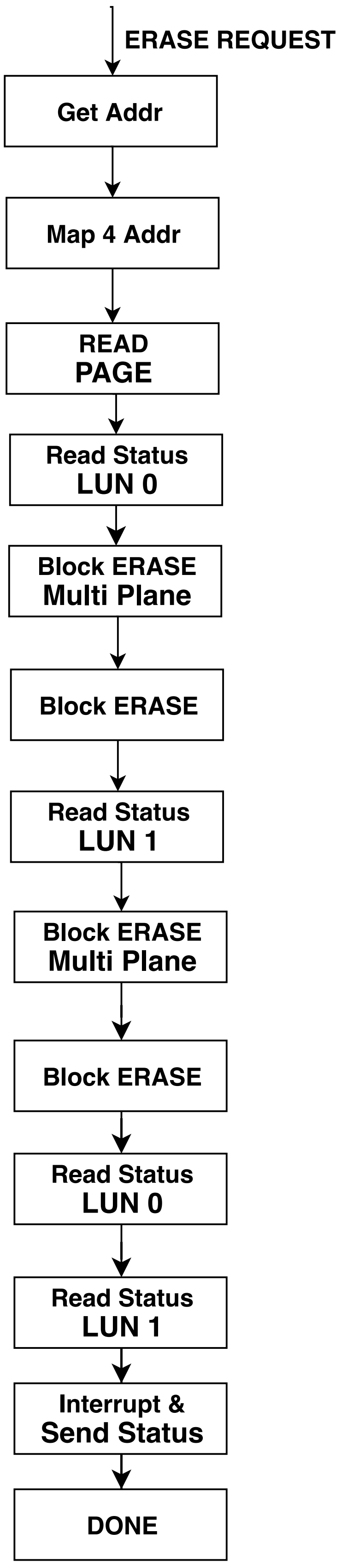


Figure 6.12: Flow of ERASE operation

6.4 Simulation results

Micron NAND flash device MT29F2G08AABWP have been used. This is of 2GB but this is extended to 16GB. The NAND flash memory target as shown in fig. 4.3. This target is instantiated twice for obtaining fig. 4.5. For the verification of the NFC, Bus Functional Model is used as per the requirements of this thesis implementation. This model is a superset of all supported Micron NAND devices. All the state machines mentioned have been implemeted in BSV and verified. In test bench LFSR(Linear Feedbak Shift Register) has been used for generating random 64 bit data. Along with sending data to NFC while programming, same data is being stored in BRAM for verification and comparision while reading. Simulation results will be seen in the same order as 6.1.

6.4.1 RESET Waveform

First command after power on.

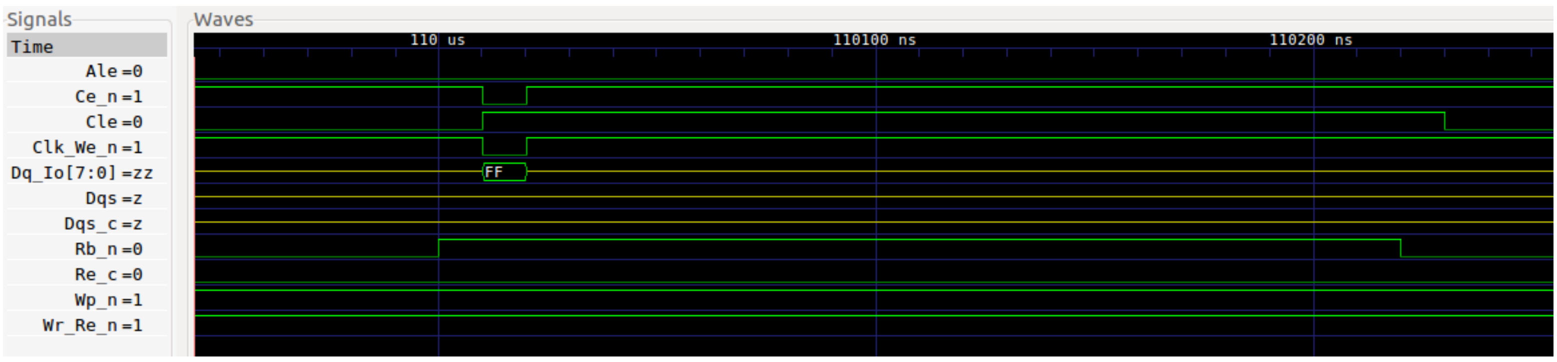


Figure 6.13: RESET operation

6.4.2 SET FEATURE Waveform

SET feature command waveform

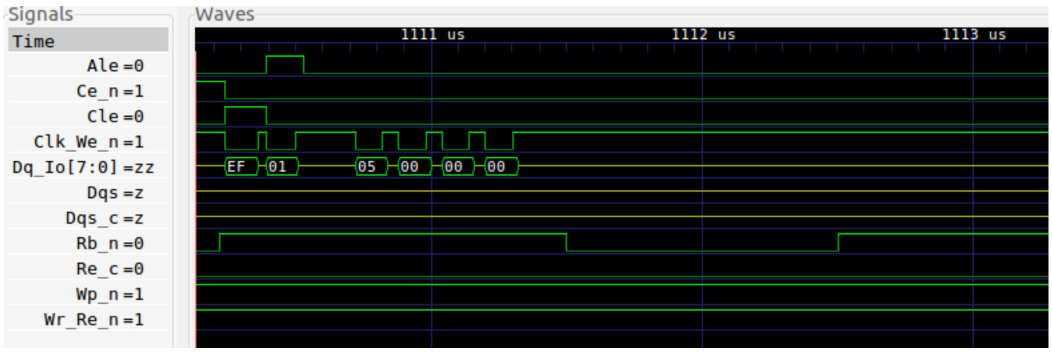


Figure 6.14: SET Feature operation

6.4.3 STATUS Waveform

STATUS check command waveform. 'hE0 means LUN is ready and previous operation successful. Observe RE signal while reading status.

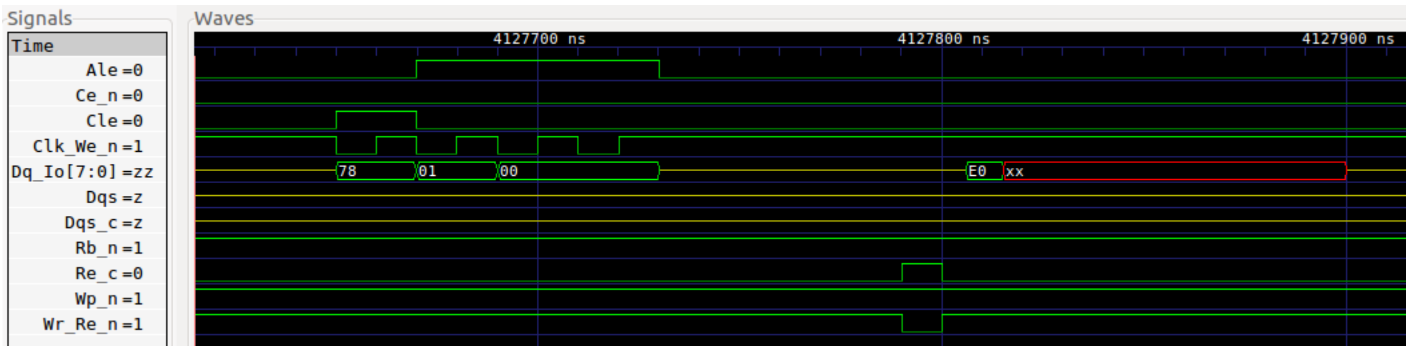


Figure 6.15: STATUS command operation

6.4.4 PROGRAM PAGE(80-10h)

Starting and end wave form is shown. In between data is being sent to memory.

When data is being sent WE signal is toggling.

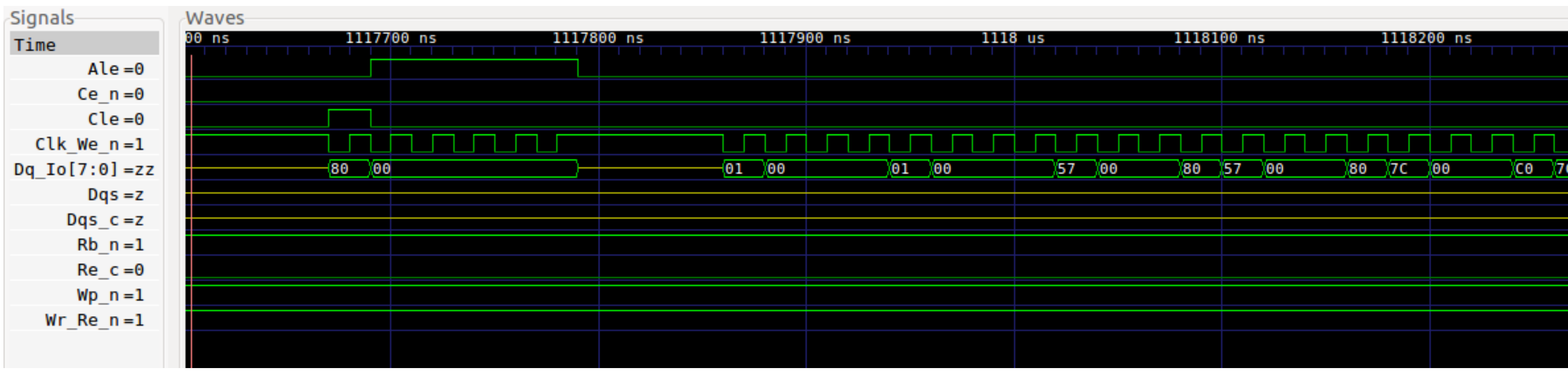


Figure 6.16: PROGARM PAGE Starting command, address and data

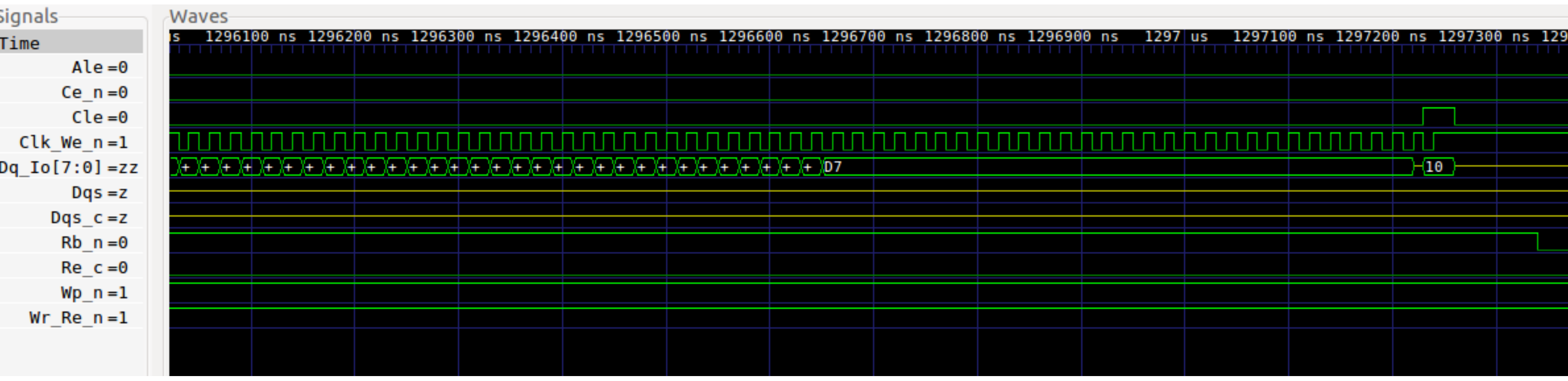


Figure 6.17: PROGARM PAGE final data, spare data and end command

6.4.5 PROGRAM PAGE MULTI-PLANE(80-11h)

Starting wave form is same as 6.16. The end command is shown below. Before ending full page data is being sent to memory.

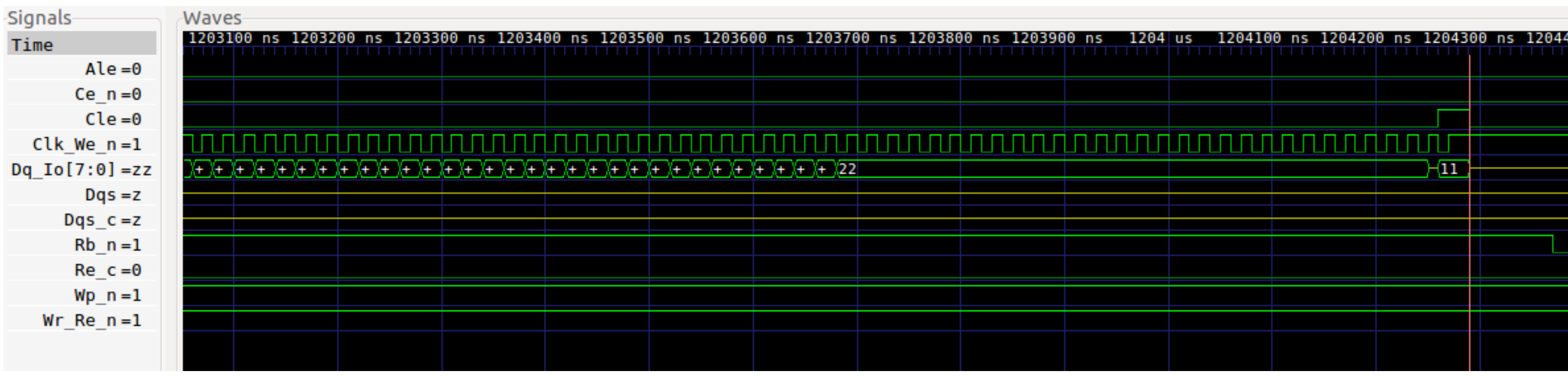

Signals
Waves
Time
Ale =0
Ce_n =0
Cle =0
Clk_We_n =1
Dq_Io[7:0] =zz
Dqs =z
Dqs_c =z
Rb_n =1
Re_c =0
Wp_n =1
Wr_Re_n =1
1203100 ns 1203200 ns 1203300 ns 1203400 ns 1203500 ns 1203600 ns 1203700 ns 1203800 ns 1203900 ns 1204 us 1204100 ns 1204200 ns 1204300 ns
22
11

Figure 6.18: PROGARM PAGE MULTI-PLANE final data, spare data and end command

6.4.6 PROGRAM PAGE CACHE(80-15h)

Starting wave form is same as 6.16. The end command is shown below. Before ending full page data is being sent to memory.

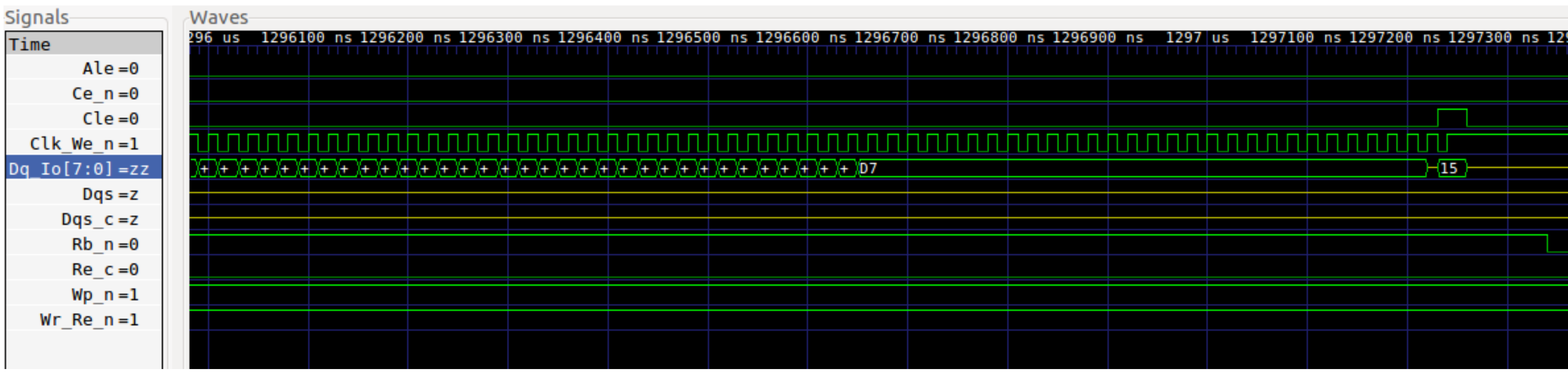

Signals
Waves
Time
Ale =0
Ce_n =0
Cle =0
Clk_We_n =1
Dq_Io[7:0] =zz
Dqs =z
Dqs_c =z
Rb_n =0
Re_c =0
Wp_n =1
Wr_Re_n =1
1296100 ns 1296200 ns 1296300 ns 1296400 ns 1296500 ns 1296600 ns 1296700 ns 1296800 ns 1296900 ns 1297 us 1297100 ns 1297200 ns 1297300 ns
D7
15

Figure 6.19: PROGARM PAGE CACHE final data, spare data and end command

6.4.7 READ PAGE(00-30h)

Starting wave form end command is shown below. Before ending full page data is being sent from memory.

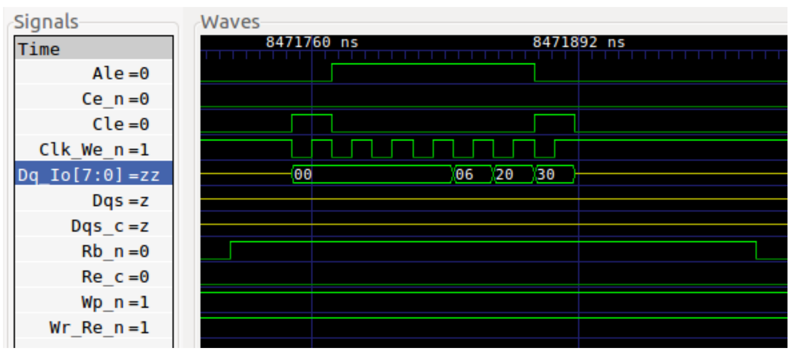


Figure 6.20: READ PAGE request

6.4.8 READ PAGE MULTI-PLANE(00-32h)

Starting wave form end command is shown below. Before ending full page data is being sent from memory.

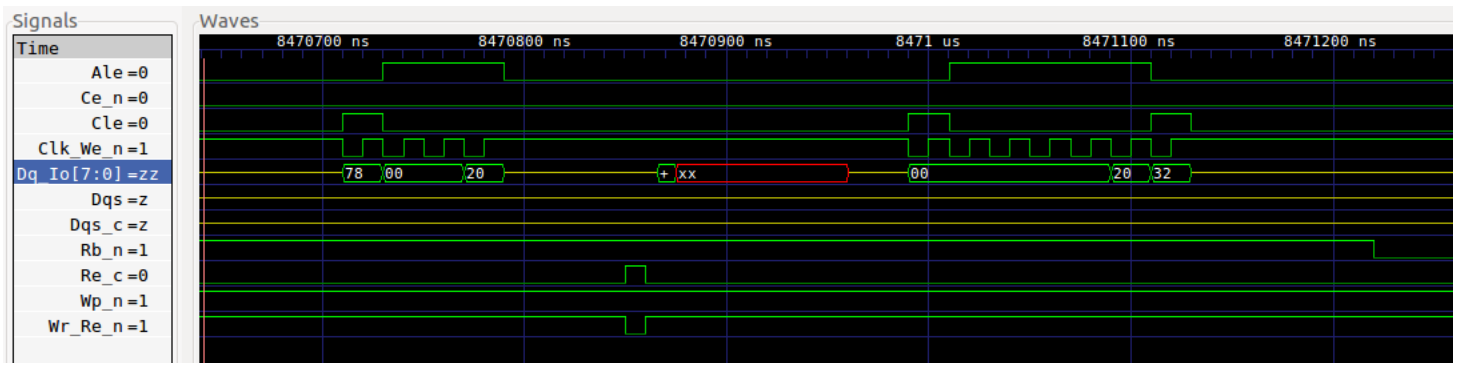


Figure 6.21: READ PAGE MULTI-PLANE request

6.4.9 READ MODE(00h)

After sending READ PAGE/READ PAGE MULTI-PLANE request, then status is checked and then controller into READ MODE for reading data. Observe that RE signal is toggling.

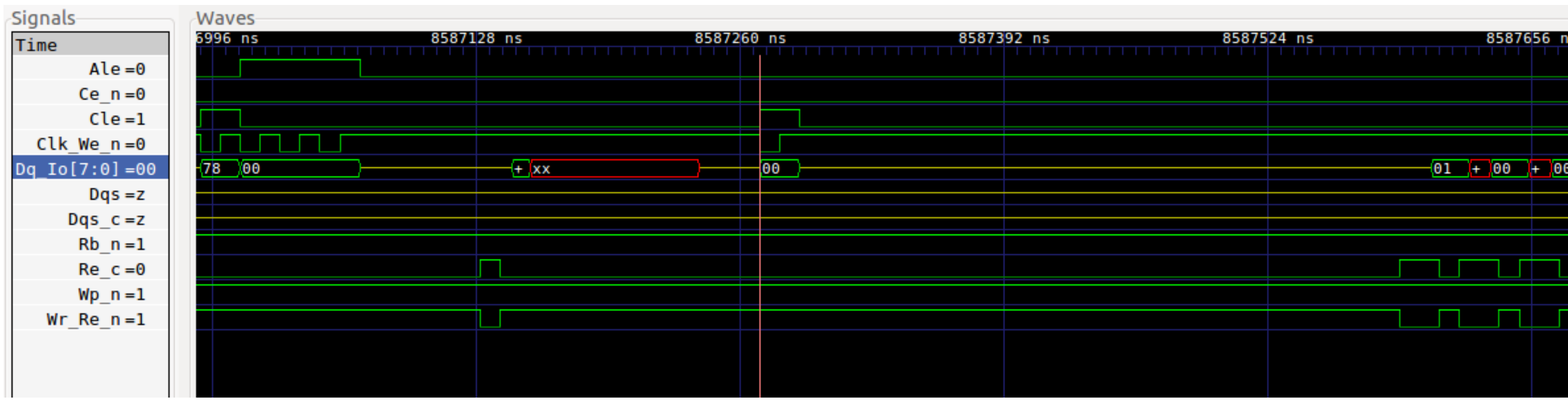


Figure 6.22: READ MODE request

6.4.10 ERASE BLOCK(06-D0h)

After checking status, ERASE BLOCK request is sent. It can be verified by reading at the same address. Observe that only 3 address cycles have been sent.

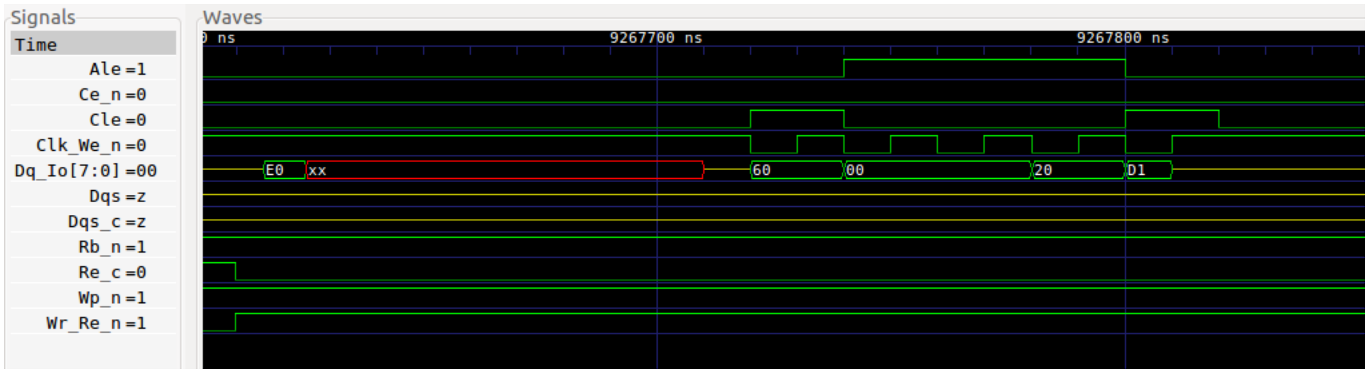


Figure 6.23: ERASE BLOCK REQUEST

6.4.11 ERASE BLOCK MULTI- PLANE(06-D2h)

After checking status, ERASE BLOCK MULTI-PLANE request is sent. It can be verified by reading at the same address. Observe that only 3 address cycles have been sent.

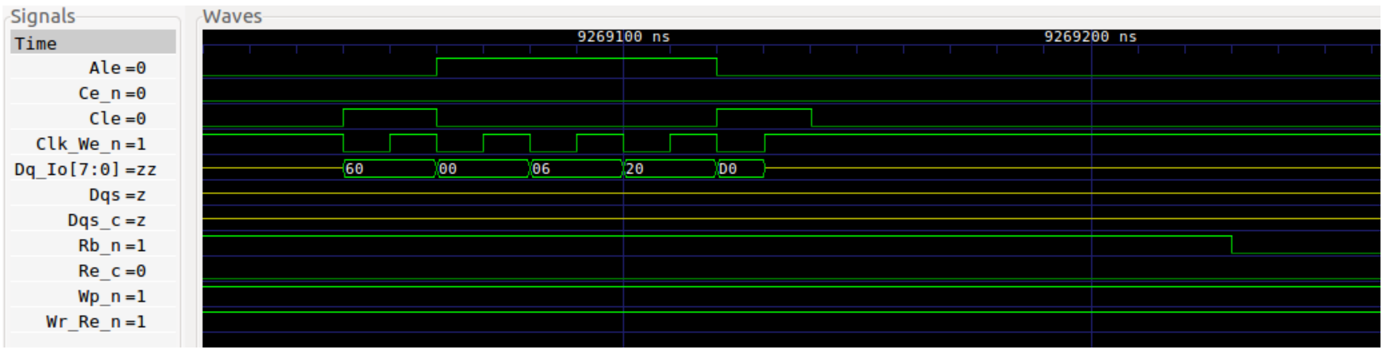


Figure 6.24: ERASE BLOCK MULTI- PLANE REQUEST

6.5 Conclusion

Program, read and erase have been implemented as above. Different test cases have been tested and verified. Some of the test cases verified are:

- WRITE N pages, READ N pages and comparing data. Where N is 1 to 2047 i.e., from 1 page to one mega block.
- Continuation of above case is to ERASE full mega block. Then READ data and data should be all 1's means FFFF in hexadecimal.
- WRITE, READ and ERASE a random block.
- Above all cases are with starting from even plane and odd plane also.
- Above all cases are verified with starting address as random page i.e., first page of a block.

CHAPTER 7

CONCLUSION AND FUTURE WORK

All three basic operations write, read and erase are implemented using ONFI 4.0 and tested well through simulation. Multi-plane operations operations also implemented and tested. This thesis focus is only on NFC and NAND flash memory. Requests are given from testbench to NFC and operations have been performed and verified using BFM model. In test bench LFSR is used for generating random data. Data has been stored in BRAM while sending data to NFC. So, when a READ happens BRAM data is compared with the data from flash memory. If data is same then both write and read are successful, Same for ERASE.

7.1 Future Work

Full flow from host to NAND flash chips has to be done. So NVMe and NFC interface must be done for full design of memory system. In this implementation only asynchronous interface has been completed. Still Synchronous interface must be added. Present SDR will improve to DDR which achieves greater data rate than the preceding single data rate. ECC must be added in spare data(224Bytes) which is present at end of every page.

REFERENCES

[1] **Bluespec-Inc,** *Bluespec System Verilog Refeerence Guide.* Bluespec Inc, 2014.

[2] **Bluespec-Inc,** *Bluespec System Verilog BSV by Examples.* Bluespec Inc, 2010.

[3] AMBA AXI and ACE Protocol Specification by ARM, 2003-2011.

[4] Solid State Drives 101 EBook. www.cactus-tech.com/landing/ssd-101-ebook

[5] ONFI Workgroup. Open Nand Flash Interface Specification. Revision 4.0, February 2014. www.onfi.org.

[6] Micron's x8 High Speed NAND FLash Memory of MT29H series.

[7] Micron technology Inc. TN2919 NAND Flash 101: An Introduction to NAND Flash and How to Design It In to Your Next Product

[8] Micron technology Inc. x8 NAND Flash Memory MT29F series.