Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
NAND Flash Architecture

- Employing cache and data registers
- Multiple planes (memory array)
- Multiple dies

Memory Array (Plane)
Package Organization

- Flash package can employ multiple Dies, each consisting of one or more flash memory islands, referred to as Plane
  - Planes in typical share multiple peripherals, which enable all the target pages connected to their wordlines/bitlines
  - Please review the bottom lines of flash memory operations at our first lecture
- Each plane employs a set of registers to cache/buffer the data brought by flash interface
Physical Plane and Die

• Each plane defines a different set of block addresses, and the pages/blocks on different planes operate in parallel
  • The operation type for the requests across different plane should be same
  • The page and plane addresses should be identical

• All individual dies can be simultaneously activated, but share datapath (described at the next page)
Flash Microarchitecture

- All data, commands and addresses are **multiplexed** onto same I/O pins and received by I/O control circuit.
- Each component of flash transactions is latched by:
  - An address register
  - A status register
  - A command register
  - Cache/data registers

![Flash Microarchitecture Diagram](image-url)
Flash Interface and Protocol

• Thus, to issue a read/write request through the multiplexed I/O pins, users (i.e., flash controllers) should obey the protocol that flash interface defines for all memory transactions
  • Basic operations
  • Cache mode operations
  • Multi-plane mode operations
  • Copyback, etc.
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Legacy Operation

• An I/O request splits into several operation stages (defined by flash interface protocols such as ONFi)
• Each stage should be appropriately handled by flash controller(s)
A Basic Write

• The first command is needed to indicate that this is a legacy page write
• The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order
• Data should be then transferred, and the second command initiates a write
• Once the write is performed, users need to send the last command to check up the status of target

[EXAMPLE]
A Basic Write

- The first command is needed to indicate that this is a legacy page write.
- The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order.
- Data should be then transferred, and the second command initiates a write.
- Once the write is performed, users need to send the last command to check up the status of target.
A Basic Write

• The first command is needed to indicate that this is a legacy page write
• The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order
• Data should be then transferred, and the second command initiates a write
• Once the write is performed, users need to send the last command to check up the status of target
A Basic Write

- The first command is needed to indicate that this is a legacy page write
- The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order
- Data should be then transferred, and the second commend initiates a write
- Once the write is performed, users need to send the last command to check up the status of target
A Basic Write

• The first command is needed to indicate that this is a legacy page write
• The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order
• Data should be then transferred, and the second command initiates a write
• Once the write is performed, users need to send the last command to check up the status of target
A Basic Write

• The first command is needed to indicate that this is a legacy page write
• The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order
• Data should be then transferred, and the second command initiates a write
• Once the write is performed, users need to send the last command to check up the status of target
A Basic Write

- The first command is needed to indicate that this is a legacy page write.
- The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order.
- Data should be then transferred, and the second command initiates a write.
- Once the write is performed, users need to send the last command to check up the status of target.

*Typically, memory writes take 300 us~6 ms based on the target address*
Intrinsic Latency Variation

• Fowler-Nordheim Tunneling
  • Making an electron channel
  • Voltage is applied over a certain threshold

• Incremental step pulse programming (ISPP)
Intrinsic Latency Variation

- Each step of ISPP needs different programming duration (latency)
- Latencies of the NAND flash memory fluctuate depending on the address of the pages in a block
A Basic Write

- The first command is needed to indicate that this is a legacy page write.
- The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order.
- Data should then be transferred, and the second command initiate a write.
- Once the write is performed, users need to send the last command to check up the status of target.

)[EXAMPLE]
A Basic Write

• The first command is needed to indicate that this is a legacy page write

• The column address (C-ADDR) and row address (R-ADDR) are shipped in a serial order

• Data should be then transferred, and the second commend initiate a write

• Once the write is performed, users need to send the last command to check up the status of target
A Basic Write

• Unlike other memory technologies, flash controller(s) should hold the data till the target return a success state.

• If it fails, the controller should retry the write till the target reports a success
  • This indicates that buffer or queue mechanisms within SSDs should appropriately manage the data based on the corresponding flash interface protocol that system employs
A Basic Read

- The first command and addresses follow the same sequence of a write
- The second command initiates a read
- A read occurs, and then the data comes out
A Basic Read

• The first command and addresses follow the same sequence of a write
• The second command initiates a read
• A read occurs, and then the data comes out
A Basic Read

- The first command and addresses follow the same sequence of a write
- The second command initiates a read
- A read occurs, and then the data comes out

[EXAMPLE]
A Basic Read

- The first command and addresses follow the same sequence of a write
- The second command initiates a read
- A read occurs, and then the data comes out
A Basic Read

• The first command and addresses follow the same sequence of a write
• The second command initiates a read
• A read occurs, and then the data comes out

Typically, it takes 25 us and 50 us for LSB pages and MSB pages, respectively
A Basic Read

- The first command and addresses follow the same sequence of a write
- The second command initiates a read
- A read occurs, and then the data comes out

A byte or word is brought by each cycle
(data out for a page requires a few thousand cycles!)
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Cache Operation

- The cache mode operations leverage internal registers in an attempt to hide performance overheads imposed by data movements
Cache Mode (Reads)

- A sequential cache operation reads a next sequential page by using a data register while the data of the previous page is in transferring from a plane to a cache register.
- A cache command should be issued just **before the data out** occurs.
- Random caching is also available by placing the target address in stead of the sequential command.
Cache Mode (Reads)

Flash-level Controller

ARRAY (PLANE)

CACHE REG

DATA REG

CMD REG

ADDR REG

Control logic

Decoder

I/O control

Cycle type

Command

Address

CMD

DOUT

Page Address

ADDR REG

Page M

Page M+1

RDY

RDY

RDY

RDY

RDY

RDY
Cache Mode (Reads)
Cache Mode (Reads)
Cache Mode (Reads)

Diagram showing the flow of data and control signals in a flash memory controller. The diagram illustrates the interaction between different registers and the array (plane). The components labeled include:

- I/O control
- ADDR REG
- CMD REG
- Control logic
- Decoder
- CACHE REG
- DATA REG
- ARRAY (PLANE)

The diagram also shows a timeline indicating the timing of commands and data transfers.
No address is required (for reading out a target)
The second tR is invisible (as it is overlapped with data transfers)
Cache Mode (Reads)
Cache Mode (Reads)

The selection of a register is controlled by control logic (invisible to flash controllers).
Cache Mode (Reads)

31h indicates that there will be a subsequent read

3Fh indicates that this data out will be the last
Cache Mode (Writes)

- Hide long latency of memory program time
  - Enable flash controller(s) to transfer data to a cache register while the target is in programming at some extent
  - Internal control logic copies the data from the cache register to the data register
    - No programming operation is involved here, which can be done as quickly as a register copy
  - The logic internally moves the target data sitting on the data register to a specified block and page addresses in the target array of selected die
    - In the meantime, the flash controller is able to transfer data to the cache register (or another one)
Cache Mode (Writes)
Cache Mode (Writes)

During this time, the status of the target chip (RB) is busy.
Cache Mode (Writes)

A flash controller can program data as many as it wants through 15h
Cache Mode (Writes)

Unlike the cache mode on reads, caching the writes is insufficient to hide long latency of the flash programming.

Normal program page command (10h) can be used for the last
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Multi-plane Mode Operation

• Two (or more) different pages can be served in parallel
• For some specific flash technologies, the operation should be composed by a same page offset (in a block) and a same die address, but different plane addresses (plane addressing rule)
Multi-plane Mode

• Before sending a basic operation command, flash controller(s) needs a request to transfer data from a target plane to its cache register.

• Each new plane address is specified, the corresponding request is queued for transferring data.

• The address of the final plane should be brought by a basic read or write command.
  • The internal flash control logic then serves all queued plane requests at a time (multi-plane operation).
Multi-plane Mode

![Diagram of multi-plane mode](image-url)
Multi-plane Mode
Multi-plane Mode
Multi-plane Mode

Diagram showing the connection between I/O, R/B# signals, and the data flow through the CACHE REG, DATA REG, and ARRAY (PLANE) components.
Multi-plane Mode
Multi-plane Mode
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations
• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Copyback Operations

- This can save space and cycles to copy data
- The source and destination page address should be located in the same die

No data movement through NAND interface
Copyback Operations

• Block management (like a garbage collection) and wear-leveling mechanisms require to move data from a block to another block through an external interface

• Copyback operations allow to transfer data within a plane from one page to another using the cache register
  • Note that it cannot be performed across different planes (even within a same die)

• AKA Internal Data Movement Mode
Copyback Operations

Why do we need this data transfer in a copyback?

This is optional
Copyback Operations

Cycle type: Command, Address, Address, Address, Address, Address, Command

DQ[7:0]: 00h, C1, C2, R1, R2, R3, 35h

RDY

DATA REG

ARRAY (PLANE)

Target

Cycle type: Command, Address, Address, Address, Address, Address, Command

DQ[7:0]: 85h, C1, C2, R1, R2, R3, 10h

RDY
Copyback Operations

DATA REG

ARRAY (LANE) Target

No data transfer
Outline

- Flash Microarchitecture
- Basic Operations (Legacy)
- Cache Mode Operations
- Multi-plane Mode Operations
- Copy-back Operations
- Erase Operations
- Evaluation Studies
  - Simulation-based studies
  - Real implementation (FGPA) and evaluation
Erase Operations

• The command sequence for erasing a block is not different with basic operations, but the page address is ignored (column)

• It also supports multi-plane mode for erasing multiple blocks in parallels

There are only three cycles for the target block address
Combination

• All NAND flash memory operations are used in any combination

• Scheduling NAND flash command is an important design parameter to determine system performance
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations

• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Evaluation Studies

NANDFlashSim

A Cycle-accurate Microarchitecture-level NAND Flash Memory System Simulation Framework

Overview

NANDFlashSim is a flash simulation model, decoupled from specific flash firmware and supports detailed NAND flash transactions with cycle accuracy. This low-level simulation framework can enable research on the NAND flash memory system itself as well as many NAND flash-based devices. The simulation model of NANDFlashSim is validated with an hardware prototype (MBS), and currently we have successfully integrated it with a very-large scale NAND flash-based storage system and evaluated a thousand thousand NANDFlashSim instances on NERSC Hopper and Carver supercomputers. NANDFlashSim is originally designed and implemented as a library, which is a part of the cycle-level simulation models – Hardware/software co-simulation model forexascale of National Energy Research Scientific Computing Center.

Lawrence Berkeley National Laboratory (CoDE) and a many NVRAM-based SSD platform (FLASHWOD) of UT Dallas and PennState. For better understanding NANDFlashSim usages, we provide a sample system with NANDFlashSim, but please note that such sample system doesn’t contain system clock itself.

If you decide to use NANDFlashSim in your own research, please cite our MSST paper. For convenience, here is the BibTex information.

Why NANDFlashSim?

• Cycle-level simulation model
• http://nfs.camelab.org
High-level View

- Employ a command set architecture and an individual state machine
- Host clock domain and NAND flash clock domain are separated.
- All internal entries (controller, register, die, etc.) are updated at every cycle
Command Set Architecture

• Multi-stage operations
  • Stages are defined by common operations
  • CLE, ALE, TIR, TIN, TOR, TON, etc...

• Command chains
  • Declare the command sequences based on ONFi

[Diagram showing the process flow with stages and chains, labeled 'FETCH', 'BUILD', 'STAGE', 'CHAIN', 'ARBI.', 'EXE.', and 'REL.', with nodes labeled 'Die', 'Plane', and 'NAND I/O Bus'.]
State Machine

• With the command chains, NANDFlashSim’s state machine
  • regulates an appropriate sequence of command sets
  • checks a violation of command management
• Each die has its own state machine with an independent clock
Performance of Multiple Planes

• **WRITE:** Write performance is significantly enhanced as the number of plane increases
  - Cell activities (TIN) can be executed in parallel

• **READ:** Data movement (TOR) is a dominant factor in determining bandwidth

![Graph showing write performance (single die)]

Larger transfer sizes couldn’t take a full advantage of multiple plane operations

![Graph showing read performance (single die)]
Performance of Multiple Dies

• Similar to the multi-plane architecture, write performance is improved by increasing the number of dies

• Multiple die architecture provides a little worse performance than multi-plane architecture
Multi-plane VS Multi-die

• Under disk-friendly workload
  • The performance of die interleaving is 54.5% better than that of multi-plane operations on average
  • Interleaved-die operations (i.e., die interleaving) have less restrictions for addressing
Breakdown of Cycles

- While most cycles of writes are used for the NAND flash memory operation itself, reads consume at least 50.5% of the total time for their data movements (i.e., data transfers).
Page Migration Test (for GC)

Energy consumption characteristics are mostly related to # of activate components.
Varying Page Sizes

[Graphs showing the relationship between page size and bandwidth for different cache configurations: Legacy, Cache, Two-plane(2x), and 2x Cache.]
Outline

• Flash Microarchitecture
• Basic Operations (Legacy)
• Cache Mode Operations
• Multi-plane Mode Operations
• Copy-back Operations
• Erase Operations

• Evaluation Studies
  • Simulation-based studies
  • Real implementation (FGPA) and evaluation
Open-license hardware platform is available

- Provide all necessary information from the ground up:
  - PCB design/drawing
  - FPGA source code and design files
  - Host application and manual

- [http://opennvm.camelab.org](http://opennvm.camelab.org)
Host: Conventional personal computer
FPGA: Diligent Nexys 3 (based on Xilinx Spartan 6)
Daughterboard: connects NVM pins to FPGA pins
Power Board: measures average current
Software Support

Host (Python)
- Control Application
  - Control Scripts
  - Log Collector
- Log Parser

FPGA
- Control Core
  - Main Loop
- UART
  - RX: Configure by scripts
  - TX: Report results
- Memory Controller
Hardware Implementation

Actual hardware implementation based on architectural design
NAND Flash stores data based on cell $V_{th}$ level, which is determined by the presence (or absence) of electron trapped in the floating gate.

- SLC stores 1 bit/cell, MLC usually 2 bits/cell, and TLC 3 bits/cell.
- In TLC NAND flash, CSB and MSB take more programming steps compared to LSB, and therefore are expected to have higher latency, particularly for writes.

In this work, we evaluated Micron’s MT29F64G08, which is a 64 Gb TLC NAND flash product.
Characterization Parameters

Latency:
  Access Latency  
  + Operation Latency

Power: Current × Voltage

Energy: Power × Latency

Reliability:
  ✓ Endurance
  ✓ Retention
  ➞ BER (Bit Error Rate)

Write: 00110101 ...
Read: 00100101 ...

\[ BER = \frac{\text{error bits}}{\text{total bits}} \]
NAND Flash structure and Address Scrambling

(a) Plane, block and page structure.

(b) Page address scrambling.
Static Analysis: Latency vs. Page and Operation Type

- Read latency is significantly shorter than Write latency, as expected.
- Read latency of MSB pages is longer than CSB and LSB pages by 42% and 85%, respectively.
- Similar trend for writes, the latency of LSB pages is shorter than the CSB and MSB pages by about 60% and 90%, respectively.

Write performance of TLC NAND for MSB pages can be worse than high-performance rotatory HDDs!

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Read Average</th>
<th>Read Max</th>
<th>Write Average</th>
<th>Write Max</th>
<th>Erase</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB</td>
<td>0.058</td>
<td>0.129</td>
<td>0.54</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>CSB</td>
<td>0.075</td>
<td>0.128</td>
<td>2.122</td>
<td>3.027</td>
<td>2.57 -11.14</td>
</tr>
<tr>
<td>MSB</td>
<td>0.107</td>
<td>0.122</td>
<td><strong>4.814</strong></td>
<td>7.116</td>
<td></td>
</tr>
<tr>
<td>Server HDD [1, 22]</td>
<td>2.0</td>
<td>4.2</td>
<td>2.0</td>
<td>4.2</td>
<td>(15000 RPM)</td>
</tr>
</tbody>
</table>
**Dynamic Analysis: Latency vs. P/E Cycles**

- **Read Latency**: Experience no significant change in latency with increased P/E cycles.

- **Write Latency**: Shows a logarithmic decline for higher P/E cycles. Specifically, MSB is affected most, showing around 18% decrease in latency after 4K P/E cycles.

- **Erase Latency**: Shows a linear increase for increasing number of P/E cycles.

Variation in Read, Write and Erase latency for P/E cycles ranging from 0 to 4000.
Write and Erase operations consume more than **2X power** compared to Read operations.

Write are about **30X slower** than reads, and TLC flash can consume up to **60X more energy** for writing, compared to reading the same amount of data.

- BER deteriorates for higher P/E cycles. Each page type shows different degradation rate, and the **MSB pages suffer significantly more**.
- The **CSB pages demonstrate a unstable BER range**, while the other page types show a **stable BER**.
Write Once Memory (WOM) Code

- A mechanism for minimizing block erase operations and improve endurance in flash memory
  - When a block needs to be erased, it is virtually marked and treated as erased
  - Wear-leveling algorithm decides when the block must be physically erased

<table>
<thead>
<tr>
<th>2-Bit Data</th>
<th>1st Write Code</th>
<th>2nd Write Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>01</td>
<td>001</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>011</td>
</tr>
</tbody>
</table>

Without WOM code:

1st Write: 10 → Erase → 1st Write: 01

With WOM code:

1st Write: 010 → Erase → 1st Write: 110

- WOM generates two 3-bit code for each of the four 2-bit data value
- 1st write code writes data for the first time, while the 2nd write code is used to overwrite a data value with one of the three other data
Error Impact of using WOM Code

- Impact of WOM related updating is significantly higher for CSB, and particularly MSB pages
  - Updating 1 bit in the MSB pages can make its BER increase to as high as 0.48
  - Updating a bit in a LSB page results in a BER of 0.12

Interestingly, updating the LSB bit also causes the BER of CSB and MSB pages to increase by 0.34%, and 1.57%, respectively
References

• NANDFlashSim: High-Fidelity, Microarchitecture-Aware NAND Flash Memory Simulation, TOS

• Internal Parallelism of Flash Memory-Based Solid-State Drive, TOS

• Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity, ICS

• OpenNVM: An Open-Sourced FPGA-based NVM Controller for Low Level Memory Characterization, ICCD

• Exploring Design Challenges in Getting Solid State Drives Closer to CPU, TC