ECEN 5623

Embedded Memories

Be Boulder.

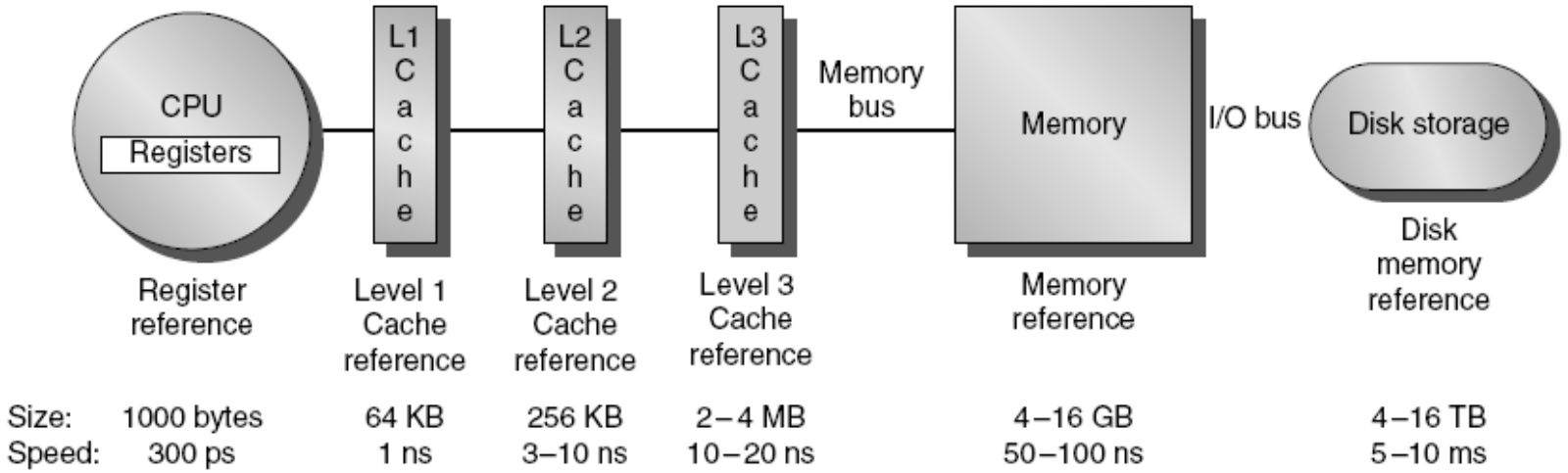
University of Colorado Boulder

Embedded Memory Systems

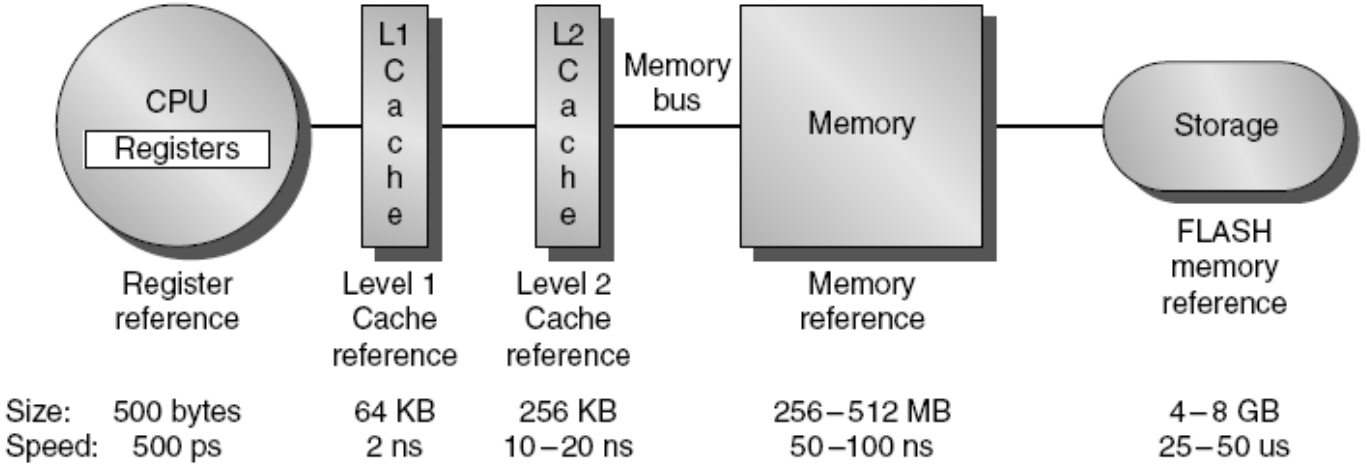
- Typical Memory Hierarchy On-Chip
  - L1 I-Cache
  - L1 D-Cache
  - L2 Unified Cache
  - SRAM (Tightly Coupled Memory Device)
- Typical Memory Hierarchy Off-Chip (External Devices)
  - L3 Unified Cache
  - SRAM/DRAM (DDR) - ECC
  - Flash – Mirrors, XOR Parity (Erasure) with Wear Leveling
  - EEPROM

© Sam Siewert

Memory Hierarchy



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

ECC Memory

- Error Detecting and Correcting Memory
  - Typically Uses Extended Parity Encoding of Data
    - Derived from Hamming Encoding
    - Requires Additional Bits for Every Word
    - E.g. 32 Bit Data Word with 16 Bit Parity Bit Word
  - Detects and Corrects Single Bit Errors (SBEs)
  - Detects, But Does Not Correct Double-Bit Errors (MBEs)
  - Syndrome Encodes Error and Correction for SBEs
    - Check Bits are Calculated Just Like Parity Bits Using Sub-Fields in the Data Word
    - Location of Errant SBE bit Provided by Check Bit Syndrome

© Sam Siewert

ECC

- Error Correction Circuitry (ECC)
  - Detects and Automatically Corrects SBEs on Read from Memory
  - **SECDED – Single Error Correction, Double Error Detection**
  - May Not Correct Actual Memory Location (Software Write-back)
  - Typically Raises an Interrupt on SBE (Can be Masked)
    - Allows Firmware to Correct Location with a Write-Back
    - Location of Error Stored in Status Register
  - Raises Exception on DOUBLE MBE (No Masking – Fatal Error)
    - System Halts and Recovery is Required
    - If System Were to Continue Execution, then What?
- Each Memory in Hierarchy Must Have Extended Parity for Reliability
  - E.g. On-Chip SRAM, L1, L2 Cache
  - As SoC Integration Levels Increase, Potential for Soft-Errors Increases
  - E.g. CPPC is Being Applied to write-back caches

© Sam Siewert

Error Correction Coding

Objective: create code sets by adding bits to the message words in such a way as to increase the minimum code distance of the code set by adding redundancy.

Minimum distance of a code set = the minimum number of bits that are different between any two codewords in the set.

Syndromes = values calculated from the received codeword using finite field arithmetic and knowledge of the code generating polynomials. Syndromes are used to determine the number of errors and the error locations.

Error Correction Coding

Error Correction: In binary codes, errors are corrected by inverting the bits identified as errors.

Rate of the error correcting code = the message length divided by the block or code length of each codeword.

An example of a Forward Error Correcting (FEC) code is the Hamming code, a linear block code.

Hamming Code Algorithm

The following general algorithm generates a single-error correcting (SEC) code for any number of bits.

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.

2. Write the bit numbers in binary: 1, 10, 11, 100, 101, etc.

3. All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits: 1, 2, 4, 8, etc. (1, 10, 100, 1000)

4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.

5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

    1. Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
    2. Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
    3. Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.
    4. Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.
    5. In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

Hamming Code Algorithm

If all parity bits are correct, there is no error.

Otherwise, the sum of the positions of the erroneous parity bits identifies the erroneous bit. For example, if the parity bits in positions 1, 2 and 8 indicate an error, then bit 1+2+8=11 is in error. If only one parity bit indicates an error, the parity bit itself is in error.

As you can see, if you have m parity bits, it can cover bits from 1 up to 2^m-1. If we subtract out the parity bits, we are left with 2^m-m-1 bits we can use for the data. As m varies, we get all the possible Hamming codes:

Parity bits	Total bits	Data bits	Name	Rate
2	3	1	Hamming(3,1)	1/3 ≈ 0.333
3	7	4	Hamming(7,4)	4/7 ≈ 0.571
4	15	11	Hamming(15,11)	11/15 ≈ 0.733
5	31	26	Hamming(31,26)	26/31 ≈ 0.839

What are the 2 code words for a Hamming (3,1) code?

A. 100, 101

B. 100, 011

C. 111, 000

D. 010, 101

When poll is active, respond at **pollev.com/timscherr391**

Text **TIMSCHERR391** to **37607** once to join

What are the 2 code words for a Hamming (3,1) code?

A 100, 101

B 100, 011

C 111, 000

D 010, 101

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What are the 2 code words for a Hamming (3,1) code?

Derivation of Hamming (3,1) code

For even parity:

Bit position: 1, 10, 11

Bit type: p1, p2, d

For **even** parity

if d = 0 then p1 must be 0 to maintain even parity, p2 is 0 for same reason
resulting codeword is 0, 0, 0

if d = 1 then p1 must be 1 to maintain even parity for that coverage
p2 is 1 for same reason
resulting codeword is 1, 1, 1

For **odd** parity

if d = 0 then p1 must be 1 to maintain odd parity, p2 is 1 for same reason
resulting codeword is 1, 1, 0

if d = 1 then p1 must be 0 to maintain even parity for that coverage
p2 is 0 for same reason
resulting codeword is 0, 0, 1

Hamming Encoding Example

- SBE Detection and Correction Including Parity Bits
- MBE Detection Only
- Syndrome Encodes Correction or MBE As Follows
  - If Check-Bits = 0 AND pW = 0 => NO ERRORS
  - If Check-Bits != 0 AND pW = 1 => SBE, CAN CORRECT
  - If Check-Bits != 0 AND pW = 0 => MBE DETECTED
  - If Check-Bits = 0 AND pW = 1 => pW ERROR, CAN CORRECT
- For 32-Bit Words, 39/32 Encoding
  - Information Rate = 82.05%
  - pW is Parity of the Encoded Word and Detects MBEs
  - pW Errors Can Also Be Detected and Corrected for SBEs

Hamming Example

(pW flip – Simple Parity Error, No ED error!)

		0	1	2	3	4	5	6	7	8	9	10	11	12
		pW	**p01**	**p02**	**d01**	**p03**	**d02**	**d03**	**d04**	**p04**	**d05**	**d06**	**d07**	**d08**
bit	**D**	X	X	X	1	X	1	0	0	X	0	1	0	0
1	**p01**		0		1		1		0		0		0	
2	**p02**			0	1			0	0			1	0	
4	**p03**					1	1	0	0					0
8	**p04**									1	0	1	0	0
16	**p05**													
32	**p06**													
	ED	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**

		0	1	2	3	4	5	6	7	8	9	10	11	12
SYN		**pW**	**p01**	**p02**	**d01**	**p03**	**d02**	**d03**	**d04**	**p04**	**d05**	**d06**	**d07**	**d08**
0	**ED**	0	0	0	1	1	1	0	0	1	0	1	0	0
c01	**0**		0		1		1		0		0		0	
c02	**0**			0	1			0	0			1	0	
c03	**0**					1	1	0	0					0
c04	**0**									1	0	1	0	0
c05	**X**													
c06	**X**													
pW2	1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**CD**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**

SYN indicates no error
(pW flip only possibility)

pW2 != pW
(indicating parity error)

`Check-Bits == 0 AND pW != pW2  => pW ERROR`

© Sam Siewert

Hamming Example

(pW flip – Simple Parity Error, No ED error!)

		0	1	2	3	4	5	6	7	8	9	10	11	12
		pW	**p01**	**p02**	**d01**	**p03**	**d02**	**d03**	**d04**	**p04**	**d05**	**d06**	**d07**	**d08**
bit	**D**	X	X	X	1	X	1	0	0	X	0	1	0	0
1	**p01**		0		1		1		0		0		0	
2	**p02**			0	1			0	0			1	0	
4	**p03**					1	1	0	0					0
8	**p04**									1	0	1	0	0
16	**p05**													
32	**p06**													
	ED	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**

		0	1	2	3	4	5	6	7	8	9	10	11	12
SYN		**pW**	**p01**	**p02**	**d01**	**p03**	**d02**	**d03**	**d04**	**p04**	**d05**	**d06**	**d07**	**d08**
0	**ED**	0	0	0	1	1	1	0	0	1	0	1	0	0
c01	**0**		0		1		1		0		0		0	
c02	**0**			0	1			0	0			1	0	
c03	**0**					1	1	0	0					0
c04	**0**									1	0	1	0	0
c05	**X**													
c06	**X**													
pW2	1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
0	**CD**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**

SYN indicates no error
(pW flip only possibility)

pW2 != pW
(indicating parity error)

`Check-Bits == 0 AND pW != pW2  => pW ERROR`

© Sam Siewert

Errors in book

The files on the DVD for "Figures" are in error, but I have uploaded the up-to-date correct figures here:

http://mercury.pr.erau.edu/~siewerts/cec450/documents/Figures/

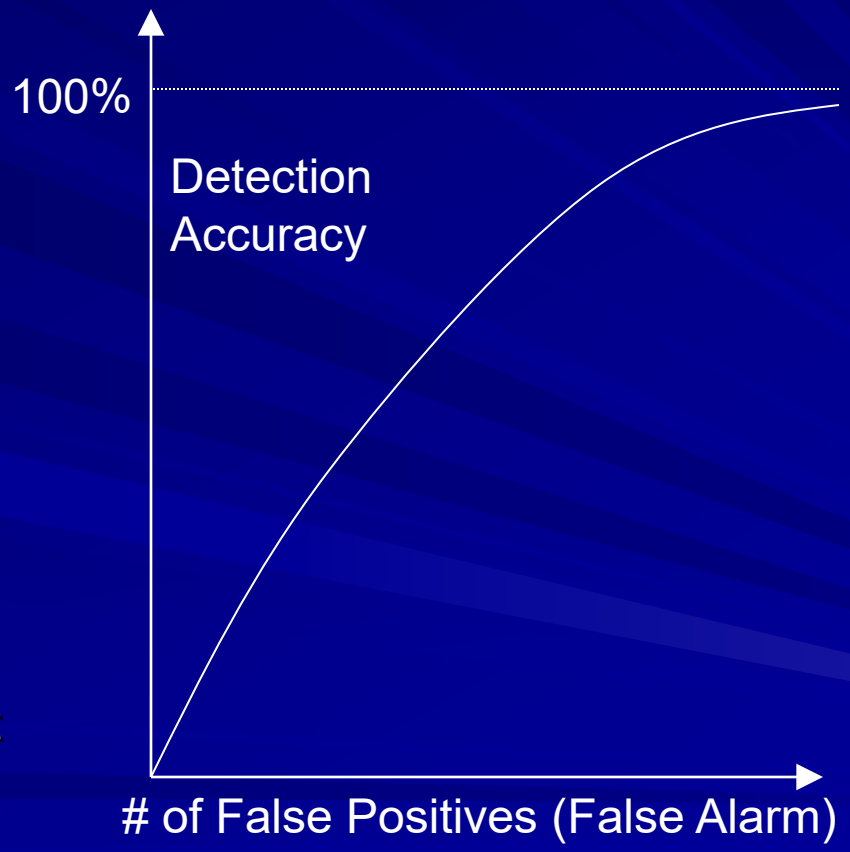
It looks clean otherwise – Figures in Chapter 17 for basic circuits used somehow developed arrow-heads – sort of confusing since the circuits should all just be simple schematics, but I corrected in downloadable figures.

BTW, I improved the Hamming code and ECC example - http://mercury.pr.erau.edu/~siewerts/cec450/code/hamming2/

It had a bug – I fixed – and now it prints out some nice tracing info.

Error Detection Theory

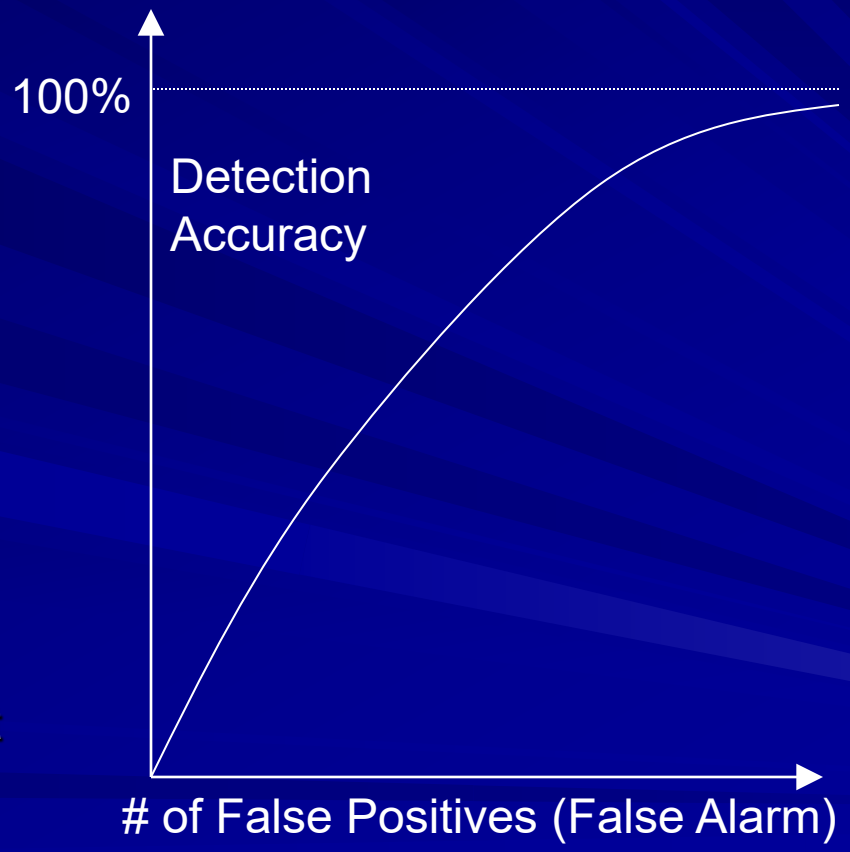
- Perfect Detector?
  - False Positives
  - False Negatives
  - Perfect Detector May be Too Costly or Impossible
- Imperfect Detector Performance
  - Number of False Positives
  - Number of False Negatives
- Detector Receiver Operator Curve
  - Plot of Number of False Positives vs. Accuracy of Correct Detection of Real Errors
  - Detection Accuracy = Correct Positives / Total Events
  - Incorrect Negative – Missed Event
  - Incorrect Positive – False Alarm
  - False Alarms Can Impact System Performance



© Sam Siewert

Error Detection Theory

- Perfect Detector?
  - False Positives
  - False Negatives
  - Perfect Detector May be Too Costly or Impossible
- Imperfect Detector Performance
  - Number of False Positives
  - Number of False Negatives
- Detector Receiver Operator Curve
  - Plot of Number of False Positives vs. Accuracy of Correct Detection of Real Errors
  - Detection Accuracy = Correct Positives / Total Events
  - Incorrect Negative – Missed Event
  - Incorrect Positive – False Alarm
  - False Alarms Can Impact System Performance



© Sam Siewert

NV RAM Devices

- Traditional Devices: PLD, ROM, EPROM, EEPROM
- Emergent Devices: NAND, NOR, and SFDC Flash
  - Multi-Megabyte devices with erasable sectors often 128 Kbytes in size
    - E.g. 2048 128K Sectors for 256 Mbyte Device
    - 8-bit/16-bit Width Devices Ganged for Wider Data Buses – e.g. 4x8-bit
  - Each block has erase lifetime typically 100,000 cycles
  - Read/Write time is a known number of CPU cycles (e.g. 6 cycles)
  - Erase time still varies (erase to all F's or all 0's in order to write)
- Boot and raw data application of flash similar to EEPROM
- Flash Filesystems are emergent use of flash
  - MINIMIZE SECTOR ERASES - Mapping of existing file-system block management strategy onto flash physical blocks
  - EVENLY DISTRIBUTE SECTOR ERASES - Goal is to provide wear leveling by evenly distributing minimum number of device sector erases over whole flash device or devices

© Sam Siewert

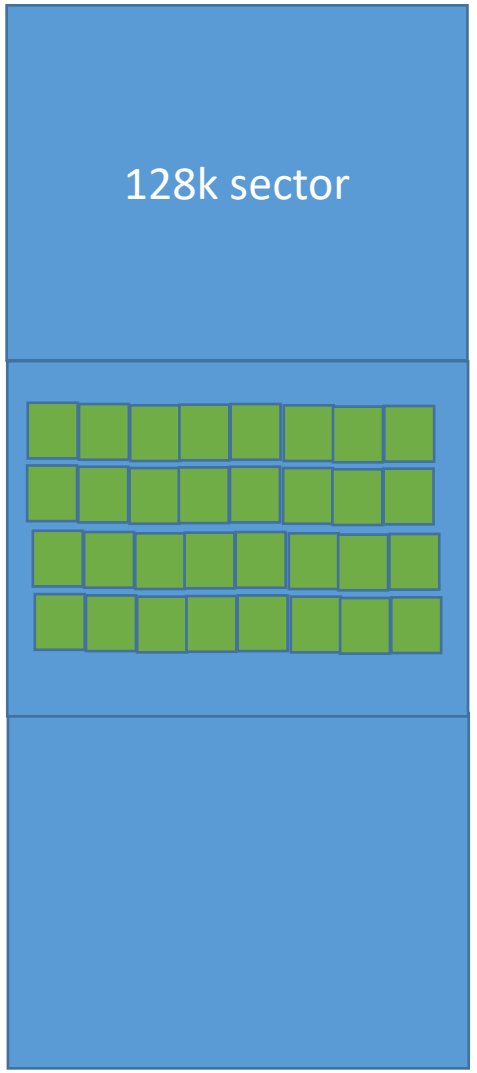
FS LB->PB Mapping and Wear Leveling

- File-systems Manage Logical Blocks of Data (1-8 Kbytes)
  - FS LBs May be Cached in Working Memory
    - Blocks Loaded into Memory on Read
    - Blocks Evicted on Reads and Written Back to Flash if Modified
    - Modified Blocks in Cache Periodically Synchronized with Flash
      - Avoid Data Loss Due to Unexpected System Reset
      - Commanded Prior to System Shutdown
  - Re-Mapped LBs Cause Invalidation of PB Locations, New LBA
  - FS LBs Can be Re-Mapped to Flash Sectors and Physical Blocks on Write-Backs
    - If PBs Exist that Have Never been Written, No Sector Erase Required – Remap Updated Blocks, Invalidate Old
    - If no Free PBs Exist, Find Sector(s) with Sufficient Invalid Blocks to Accommodate Write-Back Updates – Remap Updated Blocks, Invalidate Old
  - Block Management Requires Super-Block and Block Pointers
    - E.g. Ext2fs, UFS Super-Block and Inodes, DOS FS FAT

© Sam Siewert

FLASH Wear-leveling

Typical Pages are 4k Bytes



FS LB->PB Mapping and Wear Leveling

- File-systems Manage Logical Blocks of Data (1-8 Kbytes)
  - FS LBs May be Cached in Working Memory
    - Blocks Loaded into Memory on Read
    - Blocks Evicted on Reads and Written Back to Flash if Modified
    - Modified Blocks in Cache Periodically Synchronized with Flash
      - Avoid Data Loss Due to Unexpected System Reset
      - Commanded Prior to System Shutdown
  - Re-Mapped LBs Cause Invalidation of PB Locations, New LBA
  - FS LBs Can be Re-Mapped to Flash Sectors and Physical Blocks on Write-Backs
    - If PBs Exist that Have Never been Written, No Sector Erase Required – Remap Updated Blocks, Invalidate Old
    - If no Free PBs Exist, Find Sector(s) with Sufficient Invalid Blocks to Accommodate Write-Back Updates – Remap Updated Blocks, Invalidate Old
  - Block Management Requires Super-Block and Block Pointers
    - E.g. Ext2fs, UFS Super-Block and Inodes, DOS FS FAT

© Sam Siewert

NV Memory

Why use NAND FLASH instead of NOR FLASH?

A. More Reliable
B. Less Expensive
C. Higher Density
D. All of the above
E. B & C only

What is the advantage of SLC NAND FLASH over MLC NAND FLASH?

A. More Reliable
B. Less Expensive
C. Higher Density
D. All of the above
E. B & C only

When poll is active, respond at **pollev.com/timscherr391**

Text **TIMSCHERR391** to **37607** once to join

Why use NAND FLASH instead of NOR FLASH?

A More Reliable

B Less Expensive

C Higher Density

D All of the above

E B & C only

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/timscherr391**

Text **TIMSCHERR391** to **37607** once to join

What is the advantage of SLC NAND FLASH over MLC NAND FLASH?

A More Reliable

B Less Expensive

C Higher Density

D All of the above

E B & C only

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

FS LB to PB Mapping Example

- Min Sector Erases = Ceiling(update-blks / blks-per-sector)
- Max Sector Erases = update-blks
- Working Buffer for One Full Sector Read, Erase, Modify
- Perfect Mapping
  - Each FS LB Cache Sync/Write-back Causes Min Sector Erases
  - Sectors Erased Are Distributed Evenly over Flash Device(s)
- Example
  - File-system
    - 2 Files
    - File A = LB0, LB1
    - File B = LB2, LB3
    - FS LB Cache for 2 FS Blocks
    - All Blocks Read are Modified (Un-Modified Less Interesting)
  - Flash System
    - 1 Flash Device
    - 2 Sectors
    - Each Sector Has Capacity for Exactly 4 FS LBs – 4 PBs

© Sam Siewert

FS LB to PB Mapping Example

#1 – All blocks FREE
#2 – Erase S0 & S1, **Write LB 0, 1, 2, 3**
#3 – Read LB 0, 2, Modify, **Write LB 0, 2**
#4 – Read LB 1, 3, Modify, **Write LB 1, 3**
#5 – Read LB 0, 2, Modify and Cache
#6 – Buffer LB 0, 1, 2, Erase S0
#7 – Write-back **LB 0, 1, 2** to S0

11 Writes, 3 Sector Erases

Write Amplification = 11 / 10 = 1.1

#0 – Start State from End State Above
#1 - Read LB 1, 3, Modify and Cache
#2 – Erase S1
#3 – **Write-back LB 1, 3** to S1
#4 – Read LB 0, 2, Modify, **Write LB 0, 2**
#5 – Read LB 1, 3, Modify and Cache
#6 – Erase S0
#7 – **Write-back LB 1, 3**

6 Writes, 2 Sector Erases

Write Amplification = 17 / 16 = 1.0625

		#1 - Start	#2	#3	#4	#5	#6	#7
Sector Erased (S0, S1)		0,0	1,1	1,1	1,1	1,1	2,1	2,1
S1								
	PB7	FREE	FREE	FREE	LB3	LB3	LB3	LB3
	PB6	FREE	FREE	LB2	LB2	INVLD	INVLD	INVLD
	PB5	FREE	LB3	LB3	INVLD	INVLD	INVLD	INVLD
	PB4	FREE	LB2	INVLD	INVLD	INVLD	INVLD	INVLD
S0								
	PB3	FREE	FREE	FREE	LB1	LB1	FREE	LB1
	PB2	FREE	FREE	LB0	LB0	INVLD	FREE	FREE
	PB1	FREE	LB1	LB1	INVLD	INVLD	FREE	LB2
	PB0	FREE	LB0	INVLD	INVLD	INVLD	FREE	LB0
FS LBs Updated			0,1,2,3	0,2	1,3	0,2	0,2	0,2
FS LBs Cached						0,2	0,2	
Sector LBs Buffered							1	
		#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
Sectors Erased (S0, S1)	2,1	2,1	2,2	2,2	2,2	2,2	3,2	3,2
S1								
	LB3	INVLD	FREE	FREE	LB2	LB2	LB2	LB2
	INVLD	INVLD	FREE	FREE	LB0	LB0	LB0	LB0
	INVLD	INVLD	FREE	LB3	LB3	INVLD	INVLD	INVLD
	INVLD	INVLD	FREE	LB1	LB1	INVLD	INVLD	INVLD
S0								
	LB1	INVLD	INVLD	INVLD	INVLD	INVLD	FREE	FREE
	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE
	LB2	LB2	LB2	LB2	INVLD	INVLD	FREE	LB3
	LB0	LB0	LB0	LB0	INVLD	INVLD	FREE	LB1
FS LBs Updated	0,2	1,3	1,3	1,3	0,2	1,3	1,3	1,3
FS LBs Cached		1,3	1,3			1,3	1,3	
Sector LBs Buffered								

© Sam Siewert

Example File LB to PB Mapping

- Write Amplification – Ratio of Actual Writes / Required
- 1.0625 For our Example

- 2.5 Average Sector Erases for 16 Modifications

- Typical Flash File-system
  - 80% or More Capacity From Useable Blocks (in Sectors), Spares
  - Read, Write, Read-Modify-Write Workload %
  - 100,000+ Erases Before End of Life for Each Sector
  - 10x or More LB Updates Compared Erase Cycles With Leveling
  - Millions+ LB Updates Before End of Life With Leveling

- Low Write-Amplification and Long Total Lifetime

© Sam Siewert

Flash File-systems

- Wear Leveling and Minimizing Sector Erases – Key Concepts
  - Maximize full-capacity lifetime of device / file-system
  - Evenly distributed block erases so all wear out about the same time
  - Minimize erases by block mapping (erase only when required!)
    - Read, Modify-Block(s), Write for Sector Erases
    - FS block caches in memory – sync'd out to Flash as needed (Write-back)
    - move FS block in device block if possible before erasing
    - requires block FS with I-node or FAT for file block scatter/gather
- TFFS - software implementation of layered model
    - file-system
    - TFFS (wear leveling and FS logical-block to physical-block mapping)
    - MTD (mapping to device - read, write, erase, lock, unlock operations)
    - flash device
- Disk On Chip and Compact Flash
  - DOC - Hardware implementation of TFFS making device appear as IDE/ATA drive (fools filesystem software)
  - Compact Flash - Yet another hardware implementation that fools software into seeing flash device as a disk drive

© Sam Siewert