9 Memory Devices & Chip Area

18-548/15-548 Memory System Architecture

Philip Koopman

September 30, 1998

Required Reading: **Understanding SRAM (App. Note)**
What's All This Flash Stuff? (App. Note)

Carnegie Mellon

Assignments

- **By next class read about multi-level caching:**
  - Cragon 2.6-2.7, 2.8-2.8.2, 2.8.4
  - Supplemental Reading:
    - Hennessy & Patterson 5.5
    - Jouppi paper, 1990 ISCA, pp. 364-373

- **Homework 5 due October 7**

- **Lab #3 due October 9**

Where Are We Now?

- **Where we've been:**
  - Cache data organization
  - Cache management policies
- **Where we're going today:**
  - Underlying SRAM technology
    - While we're at it -- DRAM & non-volatile memory technology too
  - Constraints on size & shape of cache arrays
  - Sector size & block size tradeoffs
- **Where we're going next:**
  - Multi-level caching
  - System-level effects (context switching, *etc.*)

Preview

- **How SRAM works**
  - Memory cells
  - Memory Arrays
- **How DRAM & flash memory work**
  - A brief aside -- preview for main memory lecture
- **On-chip cache area**
  - Approximate area calculation
- **Size tradeoffs in terms of area vs. data organization**
  - Sectors
  - Blocks
  - Associativity

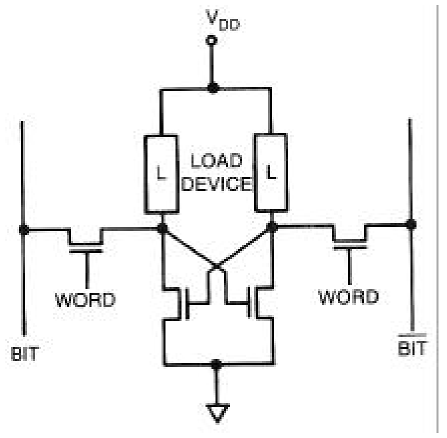
Physical Aspects of On-Chip Cache Memory

- **Cache memory arrays are physical devices, and have physical limitations**
  - Made from SRAM cells that form 2-dimensional geometric arrays
  - Have aspect ratio limits based on electrical loading
  - Have aspect ratio limits based on cache memory layout
- **Real cache design tradeoffs include:**
  - Changing cache parameters for minimal miss rate
  - Changing cache parameters so memory array fits on available chip area
    - Including both size & shape

HOW SRAM WORKS

SRAM Theory

- **Feedback of circuit retains state**
  - Bit lines read or write value
  - Word lines enable access to a row of cells
  - Load devices are resistors in "4T" cell designs

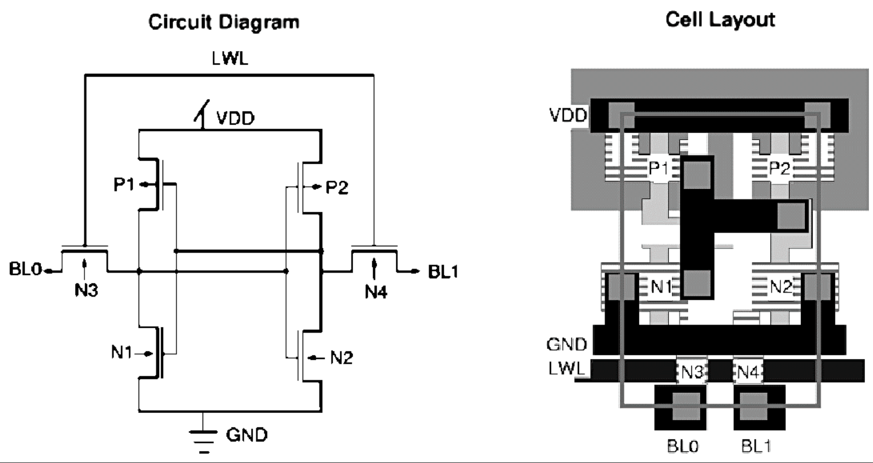


(Weste & Eshraghian, Figure 8.40)

Physical SRAM Cell Construction

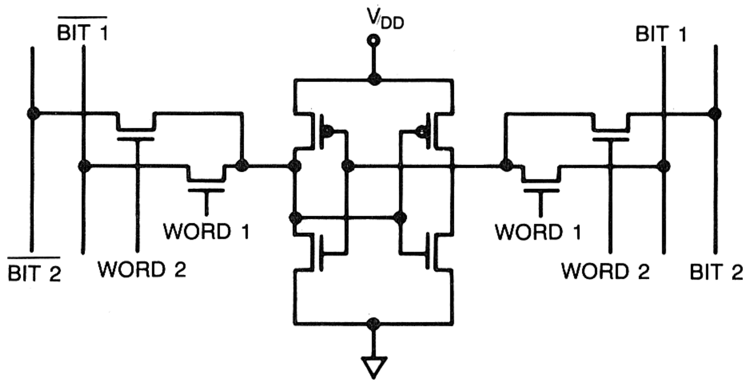
- **Uses "6T" cell design to reduce power consumption -- static CMOS**
  - In rough terms: about 60% the size of a register bit
  - Uses same process technology as CPU logic

IBM's 6-Transistor Memory Cell



Dual-Ported SRAM

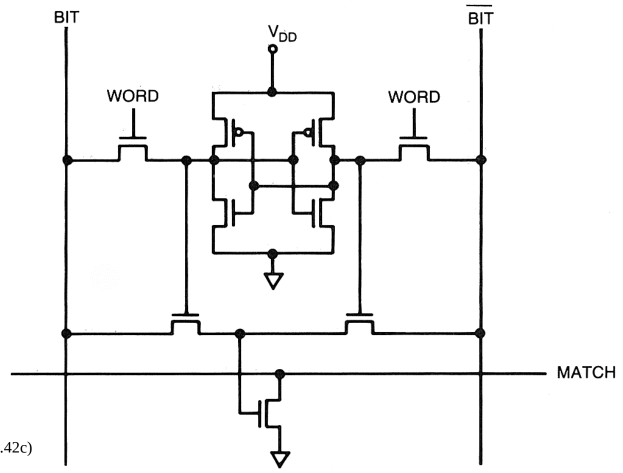
- **Permits two simultaneous accesses to memory array**
  - Uses two sets of Bit and Word lines, but only one set of cells
  - In rough terms: about 40% to 60% larger than single-ported SRAM
  - Used for simultaneous instruction & data accesses to unified cache, *etc.*



(Weste & Eshraghian, Figure 8.42b)

Content Addressable Memory

- **Detects match based on contents**
  - Can be read or written as normal SRAM
  - Additional logic to assert Match signal if contents match Bit values
  - Used for fully associative caches (*e.g.,* TLB lookup)
  - In rough terms: about 200% the size of a register bit



(Weste & Eshraghian, Figure 8.42c)

HOW DRAM & FLASH MEMORY WORK

DRAM Cells

- **DRAM optimized for small size, not speed**
  - Uses different process technology than SRAMs or CPUs
    - Integrated DRAM + CPU is not efficient

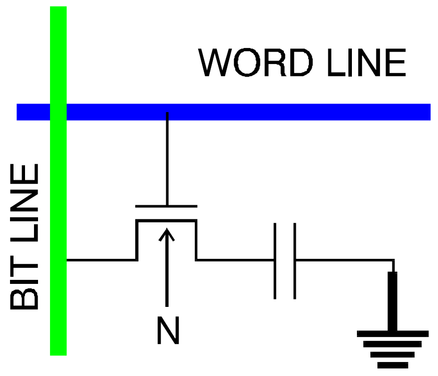
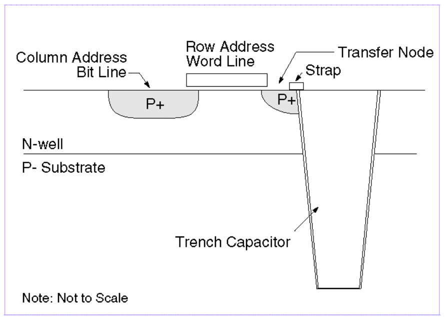


Figure 1: IBM Trench Capacitor Memory Cell



DRAM cell operation

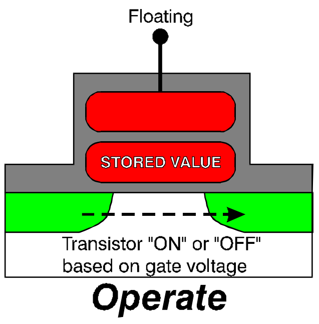
- **Transistor + capacitor**
  - Transistor used to access data
  - Capacitor used to store data value (for a while, anyway)
- **Data leaks from capacitor over time**
  - Cells must be continually "refreshed" to recharge capacitor, every few milliseconds
  - Reading cell discharges capacitor -- data must be re-written after reading (looks like a "refresh" operation)

Non-Volatile Memory

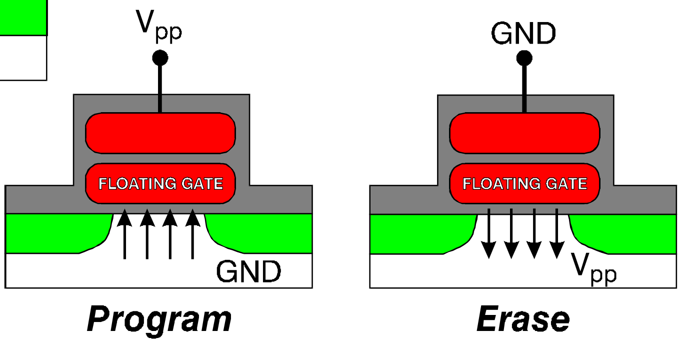
- **Non-volatile memory retains information with power turned off**
  - Battery-backed SRAM -- battery provides standby power
  - Masked ROM -- metalization contains information
  - PROM -- fuses blown during programming
  - EPROM -- gates charged during programming; UV light to erase
  - EEPROM -- EPROM, but can use high voltage to erase
  - Flash memory -- Evolution of EEPROM, only one transistor per cell
- **Non-volatile memory historically used in the memory hierarchy**
  - "Control store" for microcoded instruction implementation
  - Program memory for embedded microcontrollers
  - Boot memory for larger processors (instead of front-panel/bootstrap loader)

Flash Memory Operation

- **Flash memory holds data on a floating transistor gate**
  - Gate voltage turns transistor on or off for reading data
  - Degradation of oxide limits cycles
    - 100K cycles for NOR flash
    - 1M cycles for NAND flash



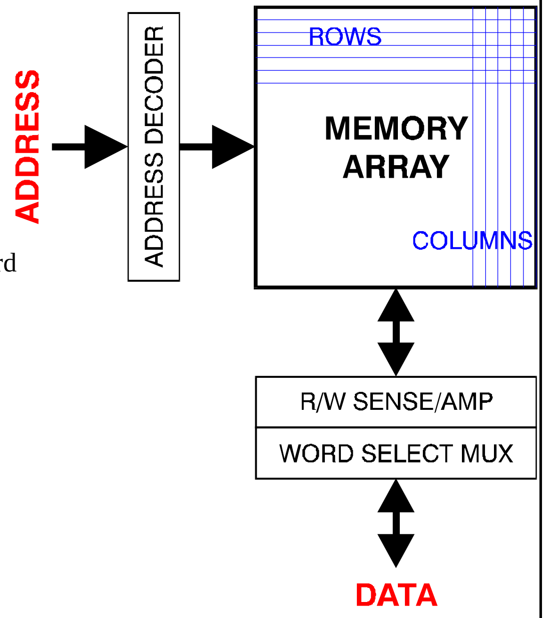
NAND Flash PROM Operation



MEMORY ARRAY CONSTRUCTION

Memory Array Geometry

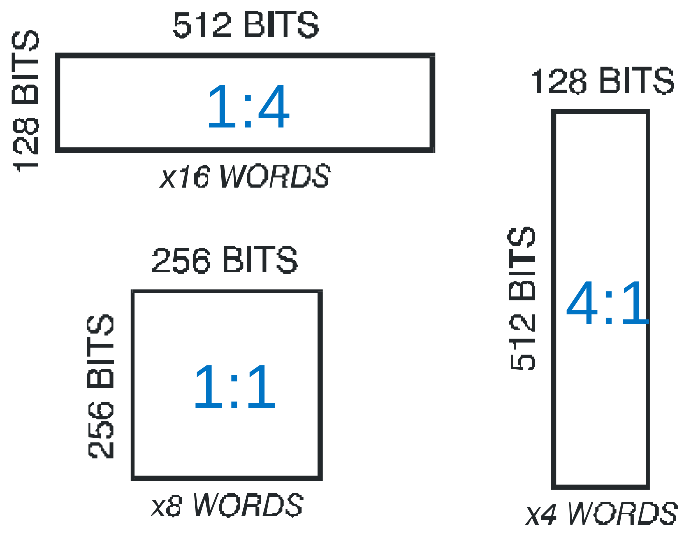
- **2-D array composed of identical memory cells**
  - Address decoder is a de-mux that selects one row
  - Sense amps detect and amplify memory cell value
  - Word select takes a subset of columns that have the byte/word of interest



Memory Array Aspect Ratio

- **Aspect ratio (length:width)** **constrains** **data words/row**

  Example: 64-Kbit array (consider 32-bit data words, but not control/tag bits)



Sector Size Constrained By Memory Array

- **Integral number of sectors per memory row**
  - Normally a power of 2 to simplify address decoding
  - Memory array need not be a power of 2 wide (sectors may not be a power of 2 number of bits)

Memory Array Width (bits)	Permissable Sector Sizes	Aspect Ratio 8 KB Cache
64	32, 64	4:1
128	32, 64, 128	2:1
256	32, 64, 128, 256	1:1
512	32, 64, 128, 256, 512	1:2

Set Size Constrained By Memory Array

- **Want entire set in a single cache memory array row**
  - Row must hold data, tags, flags for all blocks in an entire set
  - Simultaneous tag access for set-associative compare
  - Simultaneous data access so it is ready to be selected at word demuxers
- **For highly associative caches can use tricks**
  - Keep tags+flags in a separate memory array with entire set in same row
  - Use address bits to split large blocks among memory rows

ON-CHIP CACHE AREA

Cache Area Factors

- **Area is sum of data and overhead**
  - Data bit area
  - Tag+flag bit area
  - Overhead area (decode logic, sense amps, drivers, mux/demux)

- **Many memories “want” to be square or near-square (*i.e.*, not skinny/flat)**
  - Balances area overhead for address decoder & sense/drive circuits
  - Balances capacitive load on row & column lines

- **For following models, assume an integral number of sets per row...**

Cache Area Model

- **Simple area model, normalized to cache memory bit cell size**
  - S = sets in cache
  - A = associativity (sectors / set)
  - X = blocks / sector
  - B = block size (bits), including flags
  - T = tag size (bits), including LRU counter (if any) for each sector
  - R = sets per row (determines aspect ratio)
  - V = width overhead (in fractional column widths) for sense amps/buffers
  - Z = height overhead (in fractional row heights) for decoder
    - (ignores n Log n addressing/muxing effects for length & width -- assume sense amps/drivers are the majority of the cost)
- **$W_{idth} = R * A * (T + B*X) + V$**
- **$H_{eight} = (S / R) + Z$**

$$Area = W * H = [R * A * (T + B * X) + V] * \left[\left(\frac{S}{R}\right) + Z\right]$$

Example for DTMR Baseline

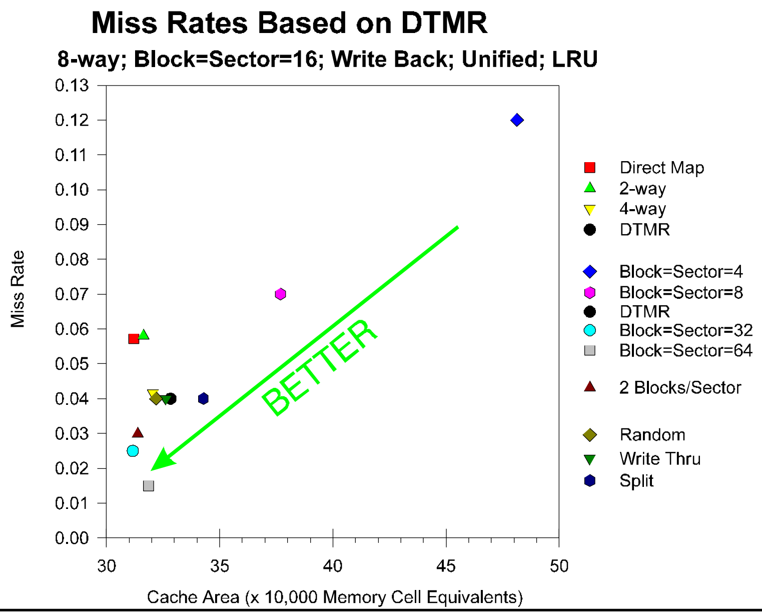
- **32 KB cache**
  - Write back; unified; assume 32-bit total virtual address size
  - S = <u>256</u> sets of 8 sectors * 16 data bytes/sector = 32KB in cache
    - » 8 address bits needed for 256 sets
  - A = <u>8</u> sectors per set
  - X = <u>1</u> block / sector
  - B = 16 bytes + 2 flags = <u>130 bit</u> block size (bits), including flags
    - » 4 address bits needed for 16 bytes
  - T = 32-4-8 = 20 address bits tag size (bits), + 3-bit LRU counter per sector = <u>23</u>
  - R = <u>1</u> set per row (can try with various integer powers of 2)
  - Assume V = Z = <u>10</u>; periphery overhead costs 10 row/column cell equivalents

- **$W_{idth} = R * A * (T + B*X) + V =$ <u>1 * 8 * (23 + 130*1) + 10 = 1234</u>**
- **$H_{eight} = (S / R) + Z =$ <u>(256/1) + 10 = 266</u>**

- **Area = W * H = <u>328,244</u> memory cell equivalents**

Example Area Tradeoff

- 32KB DTMR, assume 32 bit address, V=Z=10 units

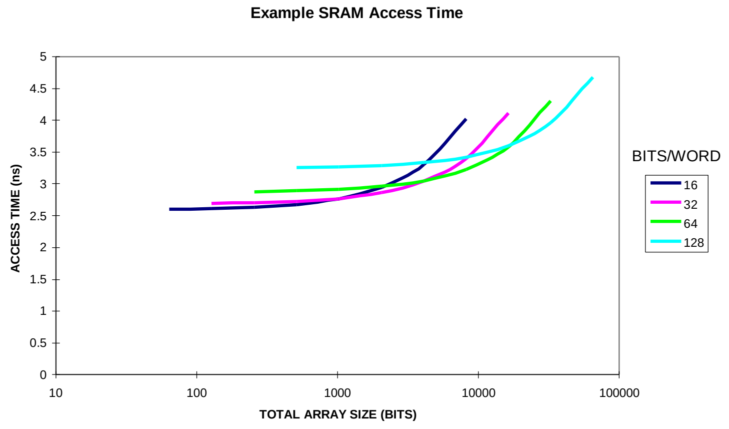


Interpreting DTMR Area Example

- **DTMR "wants" a short, fat aspect ratio**
  - DTMR is (approximated by) 8-way set associative
  - 8-way set associative; 32 KB cache; 16 bytes/sector
    - 256 sets of 1224 bits; 1 set per row; aspect ratio ~ 1 : 4.8
  - 2-way & 4-way set associative have aspect ratios closer to square
- **Block size increasing is good to a point**
  - Reduces tag overhead until aspect ratio overhead dominates at 64-byte block
  - This doesn't account for traffic ratio increases
- **4-way set associativity or direct mapped look interesting**
  - BUT: any less than 8-way set associative requires virtual memory pages > 4 KB (12 bits used in cache addressing means 12 untranslated bits available)
- **Policy changes & split cache make little size difference**
  - Policy affects few bits
  - Split cache mostly adds a second decoder overhead, but buys doubled bandwidth

Array & Cell Designs Affect Speed

- **Speed a function of:**
  - Speed of address decoding, clock ramp-up time, etc.
  - Capacitive loading of word lines (proportional to array width)
  - Capacitive loading of bit lines (proportional to array height)
  - Speed of sense amps (and loading of array outputs)



On-Chip Cache Geometry Tradeoffs

- **Associativity**
  - Low associativity helps aspect ratio for smaller (L1-sized) caches
  - Low associativity decreases tag size and LRU counter size
  - High associativity helps ease virtual memory page size constraint
- **Sector size**
  - Small sectors help aspect ratio if highly associative
  - Large sectors reduce tag overhead
  - Multiple blocks per sector benefit from reduced tag overhead per block
- **Split Cache**
  - Unified cache may have lower overhead cost (don’t need 2 address decoders)
  - Split cache relieves virtual memory page size pressure by 1 bit
- **Policy choices have less, but non-zero, area cost**
  - LRU requires counters; random needs only one LFSR for whole cache
  - Write back requires dirty bit

REVIEW

Review

- **Memory technology selection involves tradeoffs**
  - Speed+bandwidth vs. size
  - Volatility/power/lifetime
- **Cache memory array area involves:**
  - Aspect ratio limits
  - Sectors
  - Blocks
  - Associativity

Key Concepts

- **Latency**
  - "Square" memory array minimizes access latency
- **Bandwidth**
  - Wider memory arrays offer potentially higher access bandwidth, but may be slow or unwieldy
- **Concurrency**
  - Multiple sectors per cache row provides parallel lookup of tags and data
- **Balance**
  - Memory array aspect ratio and overhead requires balancing all the cache data organization parameters