inst.eecs.berkeley.edu/~eecs151

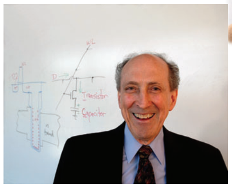
EECS151 : Introduction to Digital Design and ICs

Lecture 25 – Memories

Bora Nikolić

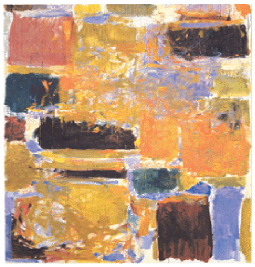
Robert Dennard

- Invented DRAM in 1968 at IBM
- Formulate Dennard's scaling: Maintain constant power density with improved frequency/performance
- The end of Dennard's scaling leads to the inability to further increase clock frequencies and multicore processors as an alternative way to improve system performance



Review

- Memory decoding is done hierarchically
  - Wire-limited in large arrays
  - Design by using the method of logical effort
- Larger memories can be built out of smaller blocks



Caches

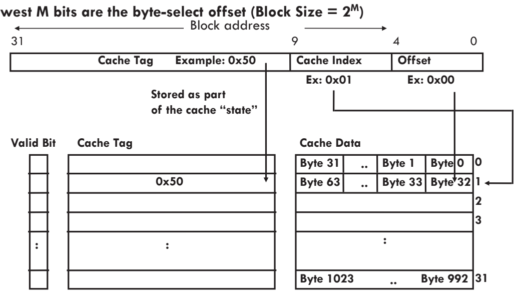
Caches (Review from 61C)

- **Two Different Types of Locality:**
  - **Temporal locality (Locality in time): If an item is referenced, it tends to be referenced again soon.**
  - **Spatial locality (Locality in space): If an item is referenced, items whose addresses are close by tend to be referenced soon.**
- **By taking advantage of the principle of locality:**
  - **Present the user with as much memory as is available in the cheapest technology.**
  - **Provide access at the speed offered by the fastest technology.**
- **DRAM is slow but cheap and dense:**
  - **Good choice for presenting the user with a BIG memory system**
- **SRAM is fast but expensive and not as dense:**
  - **Good choice for providing the user FAST access time.**

Example: 1 KB Direct Mapped Cache with 32-B Blocks

For a 2^N-byte cache:

- **The uppermost (32 - N) bits are always the Cache Tag**
- **The lowest M bits are the byte-select offset (Block Size = 2^M)**

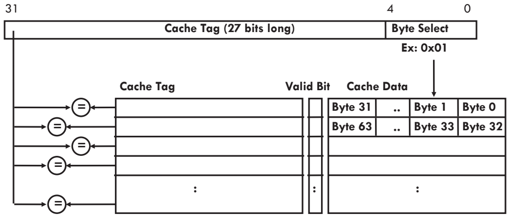


Fully Associative Cache

Fully Associative Cache

- **Ignore cache Index for now**
- **Compare the Cache Tags of all cache entries in parallel (expensive…)**
- **Example: Block Size = 32 B blocks, we need N 27-bit comparators**

By definition: Conflict Miss = 0 for a fully associative cache



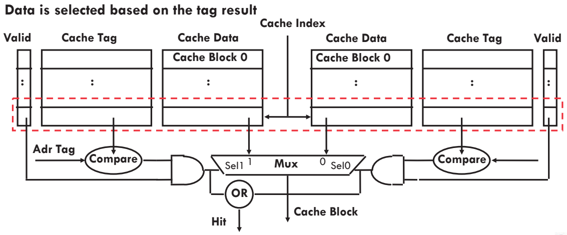
Set Associative Cache

N-way set associative: N entries for each Cache Index

- **N direct mapped caches operate in parallel**

Example: Two-way set associative cache

- **Cache Index selects a "set" from the cache**
- **The two tags in the set are compared to the input in parallel**
- **Data is selected based on the tag result**



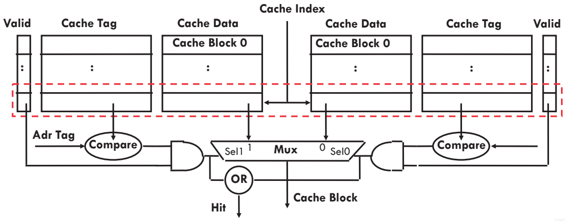
Disadvantage of Set Associative Cache

N-way Set Associative Cache versus Direct Mapped Cache:

- **N comparators vs. 1**
- **Extra MUX delay for the data**
- **Data comes AFTER Hit/Miss decision and set selection**

In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:

- **Possible to assume a hit and continue. Recover later if miss.**



Block Replacement Policy

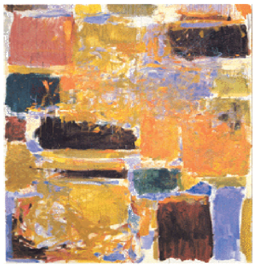
- Direct-Mapped Cache
  - index completely specifies position which position a block can go in on a miss
- N-Way Set Assoc
  - index specifies a set, but block can occupy any position within the set on a miss
- Fully Associative
  - block can be written into any position
- Question: if we have the choice, where should we write an incoming block?
  - If there's a valid bit off, write new block into first invalid.
  - If all are valid, pick a replacement policy
    - rule for which block gets "cached out" on a miss.

Block Replacement Policy: LRU

- LRU (Least Recently Used)
  - Idea: cache out block which has been accessed (read or write) least recently
  - Pro: temporal locality ➔ recent past use implies likely future use: in fact, this is a very effective policy
  - Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires more complicated hardware and more time to keep track of this

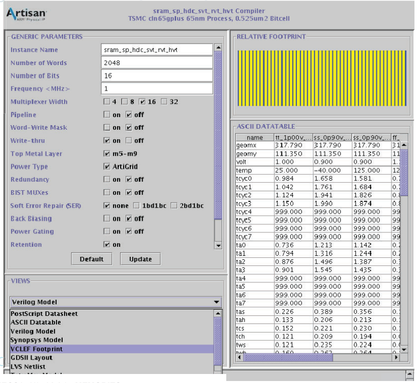
Administrivia

- Homework 10 posted on Friday, due 11/22
  - No homework during Thanksgiving
- Project checkpoints #2/#3 this week
- Next week:
  - Class lab and discussion on Monday
  - No classes/labs/discussions Tu and We



ASIC Memories

ASIC Memory Compilers



Artisan ARM Physical IP

sram_sp_hdc_svt_rvt_hvt Compiler
TSMC cln65gplus 65nm Process, 0.525um2 Bitcell

GENERIC PARAMETERS

Instance Name: sram_sp_hdc_svt_rvt_hvt
Number of Words: 2048
Number of Bits: 16
Frequency <MHz>: 1
Multiplexer Width: □ 4 □ 8 ☑ 16 □ 32
Pipeline: □ on ☑ off
Word-Write Mask: □ on ☑ off
Write-thru: ☑ on □ off
Top Metal Layer: ☑ m5-m9
Power Type: ☑ ArtiGrid
Redundancy: □ on ☑ off
BIST MUXes: □ on ☑ off
Soft Error Repair (SER): ☑ none □ 1bd1bc □ 2bd1bc
Back Biasing: □ on ☑ off
Power Gating: □ on ☑ off
Retention: ☑ on

VIEWS

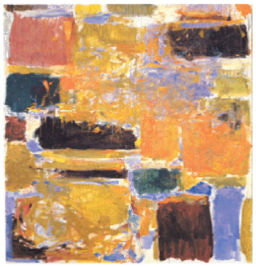
Verilog Model

PostScript Datasheet
ASCII Datatable
Verilog Model
Synopsys Model
VCLEF Footprint
GDSII Layout
LVS Netlist

ASCII DATATABLE

name	tt_1p00v_...	ss_0p90v_...	ss_0p90v_...	ff_
geomx	317.790	317.790	317.790	31
geomy	111.350	111.350	111.350	11
volt	1.000	0.900	0.900	1.
temp	25.000	-40.000	125.000	12
tcyc0	0.984	1.658	1.581	0.
tcyc1	1.042	1.761	1.684	0.
tcyc2	1.124	1.941	1.826	0.8
tcyc3	1.150	1.990	1.874	0.8
tcyc4	999.000	999.000	999.000	99
tcyc5	999.000	999.000	999.000	99
tcyc6	999.000	999.000	999.000	99
tcyc7	999.000	999.000	999.000	99
ta0	0.736	1.213	1.142	0.
ta1	0.794	1.316	1.244	0.
ta2	0.876	1.496	1.387	0.
ta3	0.901	1.545	1.435	0.
ta4	999.000	999.000	999.000	99
ta5	999.000	999.000	999.000	99
ta6	999.000	999.000	999.000	99
ta7	999.000	999.000	999.000	99
tas	0.226	0.389	0.356	0.
tah	0.133	0.206	0.213	0.
tcs	0.152	0.221	0.230	0.
tch	0.121	0.209	0.194	0.0
tws	0.121	0.235	0.224	0.0

- Memory compiler produces front-end views (similar to standard cells, but really large ones)



FPGA Memories

Verilog RAM Specification

```
//
// Single-Port RAM with Asynchronous Read
//
module ramBlock (clk, we, a, di, do);
    input  clk;
    input  we;          // write enable
    input  [19:0] a;      // address
    input  [7:0] di;    // data in
    output [7:0] do;    // data out
    reg    [7:0] ram [1048575:0];  // 8x1Meg
    always @(posedge clk) begin    // Sync write
        if (we)
            ram[a] <= di;
    assign do = ram[a];            // Async read
endmodule
```

What do the synthesis tools do with this?

Verilog Synthesis Notes (FPGAs)

- Block RAMS and LUT RAMS all exist as primitive library elements. However, it is much more convenient to **use inference.**
- Depending on how you write your Verilog, you will get either a collection of block RAMs, a collection of LUT RAMs, or a collection of flip-flops.
- The synthesizer uses size, and read style (sync versus async) to determine the best primitive type to use.
- It is possible to force mapping to a particular primitive by using synthesis directives. Ex: (* ram_style = "distributed" *) reg myReg;
- The synthesizer has limited capabilities (eg., it can combine primitives for more depth and width, but is limited on porting options). Be careful, as you might not get what you want.
- See **User Guide** for examples.
- CORE generator memory block has an extensive set of parameters for explicitly instantiated RAM blocks.

Inferring RAMs in Verilog (FPGA)

```
// 64X1 RAM implementation using distributed RAM

module ram64X1 (clk, we, d, addr, q);
input clk, we, d;
input [5:0] addr;
output q;

   reg [63:0] temp;
   always @ (posedge clk)
         if(we)
           temp[addr] <= d;
   assign q = temp[addr];

   endmodule
```

Verilog reg array used with "always @ (posedge ... infers memory array.

Asynchronous (combinatorial) read infers LUT RAM

Dual-read-port LUT RAM (FPGA)

```
//
// Multiple-Port RAM Descriptions
//
module v_rams_17 (clk, we, wa, ra1, ra2, di, do1, do2);
    input  clk;
    input  we;
    input  [5:0] wa;
    input  [5:0] ra1;
    input  [5:0] ra2;
    input  [15:0] di;
    output [15:0] do1;
    output [15:0] do2;
    reg    [15:0] ram [63:0];
    always @(posedge clk)
    begin
        if (we)
            ram[wa] <= di;
    end
    assign do1 = ram[ra1];
    assign do2 = ram[ra2];
endmodule
```

Multiple reference to same array.

Block RAM Inference (FPGA)

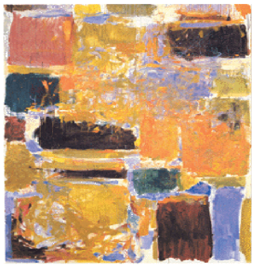
```
//
// Single-Port RAM with Synchronous Read
//
module v_rams_07 (clk, we, a, di, do);
    input  clk;
    input  we;
    input  [5:0] a;
    input  [15:0] di;
    output [15:0] do;
    reg    [15:0] ram [63:0];
    reg    [5:0] read_a;
    always @(posedge clk) begin
        if (we)
            ram[a] <= di;
        read_a <= a;
    end
    assign do = ram[read_a];
endmodule
```

Synchronous read (registered read address) infers Block RAM

FPGA Block RAM initialization (FPGA)

```
module RAMB4_S4 (data_out, ADDR, data_in, CLK, WE);
   output[3:0] data_out;
   input [2:0] ADDR;
   input [3:0] data_in;
   input CLK, WE;
   reg [3:0] mem [7:0];
   reg [3:0] read_addr;

   initial
     begin
       $readmemb("data.dat", mem);
     end

   always@(posedge CLK)
     read_addr <= ADDR;

   assign data_out = mem[read_addr];

   always @(posedge CLK)
     if (WE) mem[ADDR] = data_in;

   endmodule
```

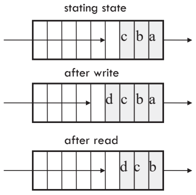
"data.dat" contains initial RAM contents, it gets put into the bitfile and loaded at configuration time.
(Remake bits to change contents)



FIFOs

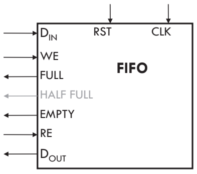
First-in-first-out (FIFO) Memory

- Used to implement *queues.*
- These find common use in processor and communication circuits.
- Generally, used to "decouple" actions of producer and consumer:



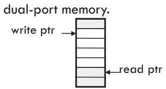
- Producer can perform many writes without consumer performing any reads (or vice versa). However, because of finite buffer size, on average, need equal number of reads and writes.
- Typical uses:
  - interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.
  - Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.

FIFO Interfaces

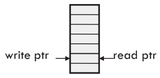


- After write or read operation, FULL and EMPTY indicate status of buffer.
- Used by external logic to control own reading from or writing to the buffer.
- FIFO resets to EMPTY state.
- HALF FULL (or other indicator of partial fullness) is optional.

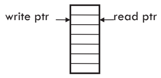
- Address pointers are used internally to keep next write position and next read position into a dual-port memory.



- If pointers equal after write ⇒ FULL:



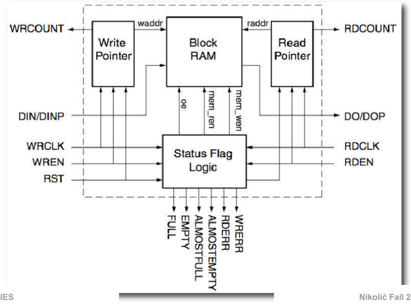
- If pointers equal after read ⇒ EMPTY:

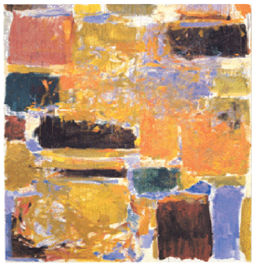


Note: Pointer incrementing is done "mod size-of-buffer"

Xilinx Virtex5 FIFOs

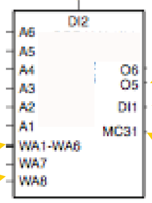
- Virtex5 BlockRAMS include dedicated circuits for FIFOs.
- Details in User Guide (ug190).
- Takes advantage of separate dual ports and independent ports clocks.





FPGA Memory Blocks

A SLICEM 6-LUT ...



Memory data input

Normal 6-LUT inputs.

Normal 5/6-LUT outputs.

Memory data input.

Memory write address

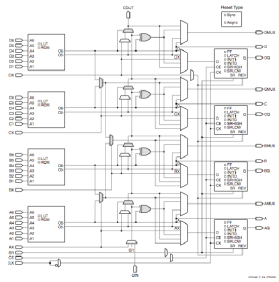
Control output for chaining LUTs to make larger memories.

Synchronous write / asychronous read

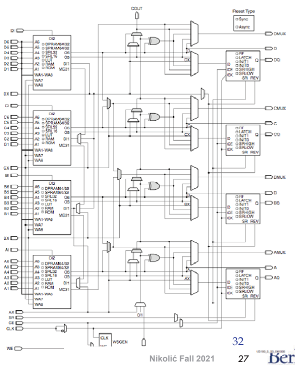
A 1.1 Mb distributed RAM can be made if all SLICEMs of an LX110T are used as RAM.

SLICEL vs SLICEM ...

SLICEL



SLICEM



SLICEM adds memory features to LUTs, + muxes.

Example Distributed RAM (LUT RAM)

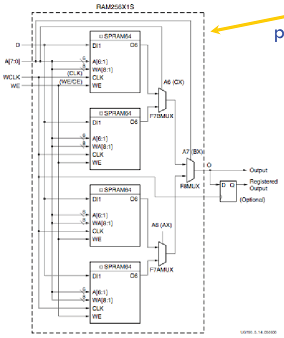
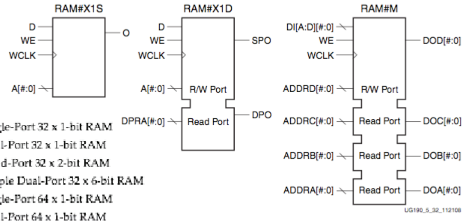


Figure 5-14: Distributed RAM (RAM256X1S)

Example configuration: Single-port 256b x 1, registered output.

Distributed RAM Primitives

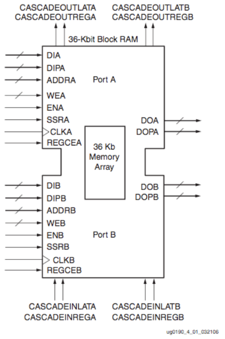


- Single-Port 32 x 1-bit RAM
- Dual-Port 32 x 1-bit RAM
- Quad-Port 32 x 2-bit RAM
- Simple Dual-Port 32 x 6-bit RAM
- Single-Port 64 x 1-bit RAM
- Dual-Port 64 x 1-bit RAM
- Quad-Port 64 x 1-bit RAM
- Simple Dual-Port 64 x 3-bit RAM
- Single-Port 128 x 1-bit RAM
- Dual-Port 128 x 1-bit RAM
- Single-Port 256 x 1-bit RAM

All are built from a single slice or less.

Remember, though, that the SLICEM LUT is naturally only 1 read and 1 write port.

Block RAM Overview



- 36K bits of data total, can be configured as:
  - 2 independent 18Kb RAMs, or one 36Kb RAM.
- Each 36Kb block RAM can be configured as:
  - 64Kx1 (when cascaded with an adjacent 36Kb block RAM), 32Kx1, 16Kx2, 8Kx4, 4Kx9, 2Kx18, or 1Kx36 memory.
- Each 18Kb block RAM can be configured as:
  - 16Kx1, 8Kx2, 4Kx4, 2Kx9, or 1Kx18 memory.
- Write and Read are synchronous operations.
- The two ports are symmetrical and totally independent (can have different clocks), sharing only the stored data.
- Each port can be configured in one of the available widths, independent of the other port. The read port width can be different from the write port width for each port.
- The memory content can be initialized or cleared by the configuration bitstream.

UltraRAM Blocks

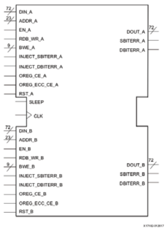
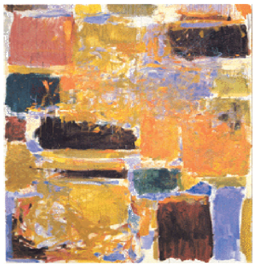


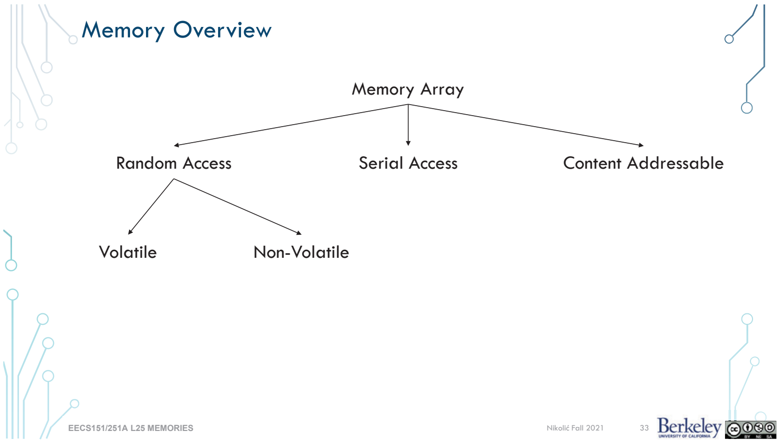
Figure 2-1: UltraRAM URAM288_BASE Primitive

Table 2-1: **Block RAM and UltraRAM Comparison**

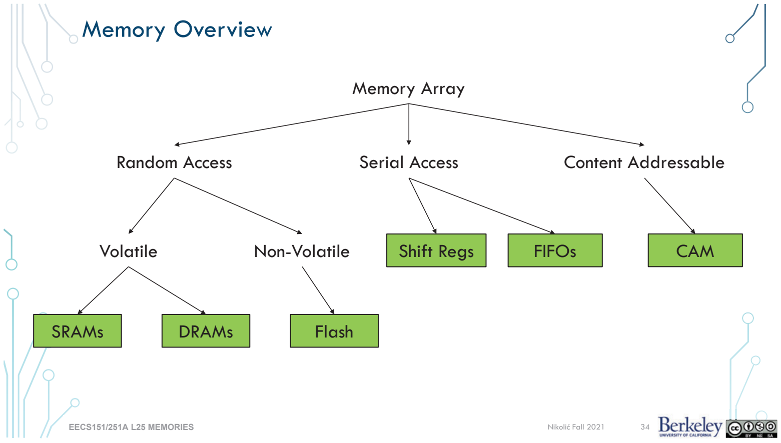
Feature	Block RAM	UltraRAM
Clocking	Two clocks	Single clock
Built-in FIFO	Yes	No
Data width	Configurable (1, 2, 4, 9, 18, 36, 72)	Fixed (72-bits)
Modes	SDP and TDP	Two ports, each can independently read or write (a superset of SDP)
ECC	64-bit SECDED Supported in 64-bit SDP only (one ECC decoder for port A and one ECC encoder for port B)	64-bit SECDED One set of complete ECC logic for each port to enable independent ECC operations (ECC encoder and decoder for both ports)
Cascade	• Cascade output only (input cascade implemented via logic resources) • Cascade within a single clock region	• Cascade both input and output (with global address decoding) • Cascade across clock regions in a column • Cascade across several columns with minimal logic resources
Power savings	One mode via manual signal assertion	One mode via manual signal assertion



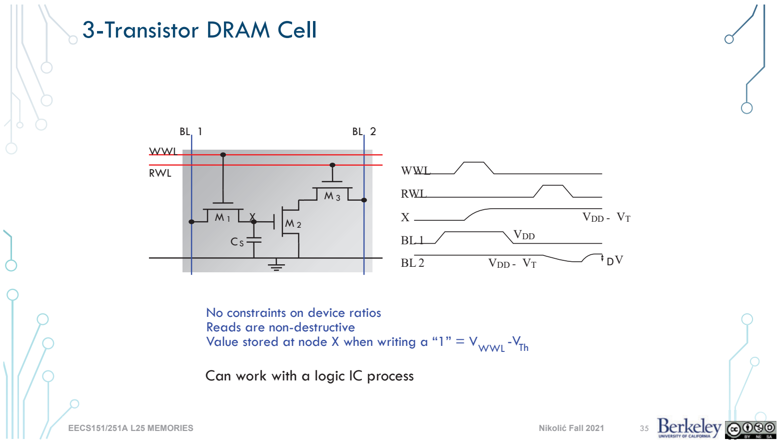
DRAM



Memory Overview



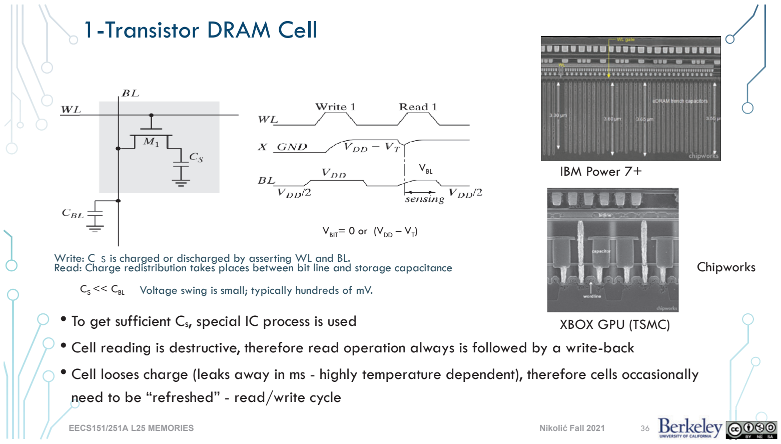
Memory Overview



3-Transistor DRAM Cell

No constraints on device ratios
Reads are non-destructive
Value stored at node X when writing a “1” = V_{WWL} -V_{Th}

Can work with a logic IC process



1-Transistor DRAM Cell

V_{BIT}= 0 or ($V_{DD} - V_T$)

Write: C S is charged or discharged by asserting WL and BL.
Read: Charge redistribution takes places between bit line and storage capacitance

$C_S << C_{BL}$ Voltage swing is small; typically hundreds of mV.

IBM Power 7+

XBOX GPU (TSMC)

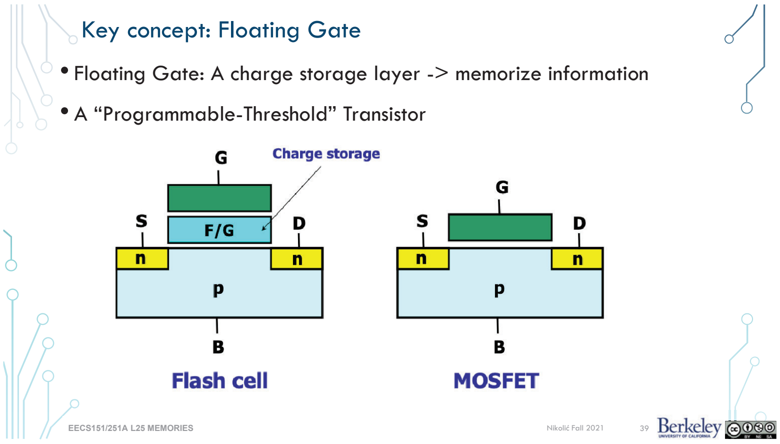
Chipworks

- To get sufficient C_s, special IC process is used
- Cell reading is destructive, therefore read operation always is followed by a write-back
- Cell looses charge (leaks away in ms - highly temperature dependent), therefore cells occasionally need to be “refreshed” - read/write cycle

Flash

Flash Memory

- Non-volatile memory

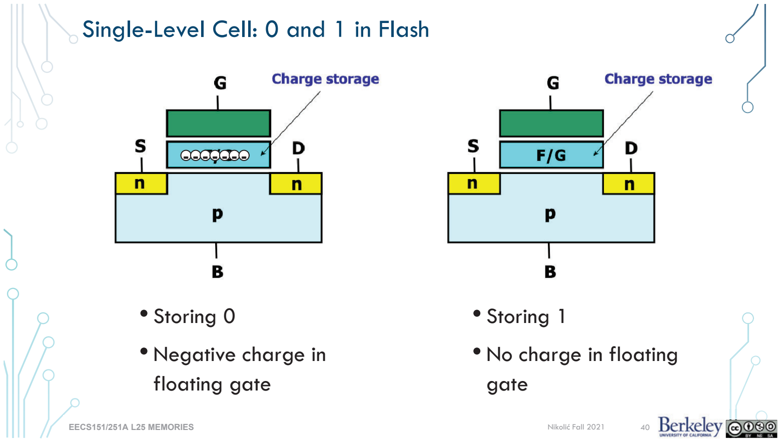


Key concept: Floating Gate

- Floating Gate: A charge storage layer -> memorize information
- A “Programmable-Threshold” Transistor

Flash cell

MOSFET

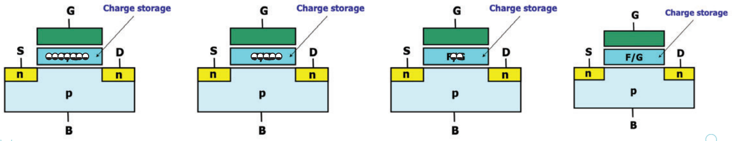


Single-Level Cell: 0 and 1 in Flash

- Storing 0
- Negative charge in floating gate

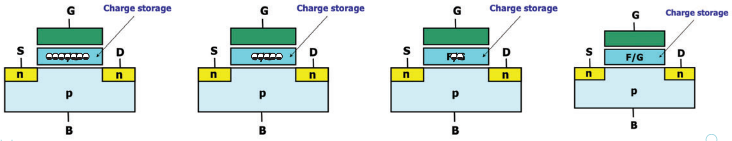
- Storing 1
- No charge in floating gate

Multi-Level Cell



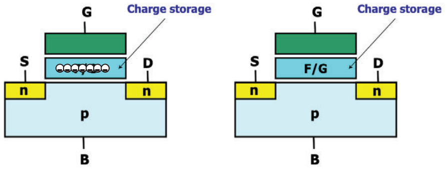
Multi-Level Cell

- Higher density
- Errors more likely



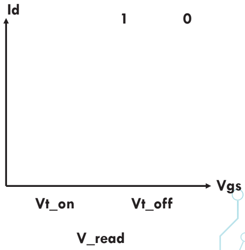
Read a Flash cell

- Floating gate change the threshold voltage of a cell
- Read the cell value by sensing the current



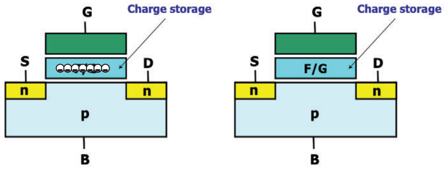
Storing 0
OFF State

Storing 1
ON State



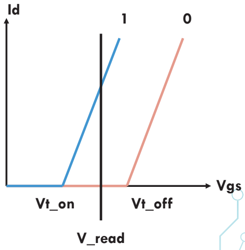
Read a Flash cell

- Floating gate changes the threshold voltage of a cell
- Read the cell value by sensing the current



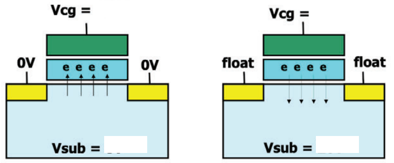
Storing 0
OFF State

Storing 1
ON State



Write a Flash cell

- Write 0: program; Write 1: erase
- Must be erased (store 1) before reprogrammed.
- Endurance: ~100K erase-program cycles

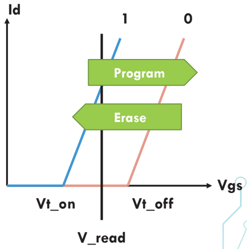


Program
F-N Tunneling

Off cell
(Solid-0)

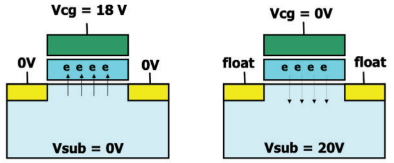
Erase
F-N Tunneling

On cell
(Solid-1)



Write a Flash cell

- Write 0: program; Write 1: erase
- Must be erased (store 1) before reprogrammed.
- Endurance: ~100K erase-program cycles

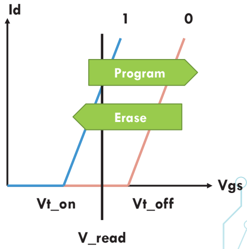


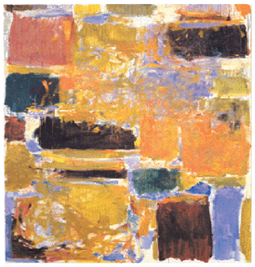
Program
F-N Tunneling

Off cell
(Solid-0)

Erase
F-N Tunneling

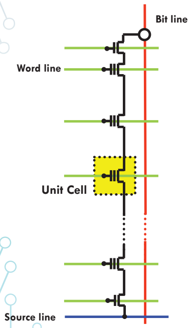
On cell
(Solid-1)

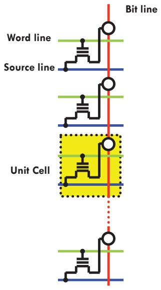


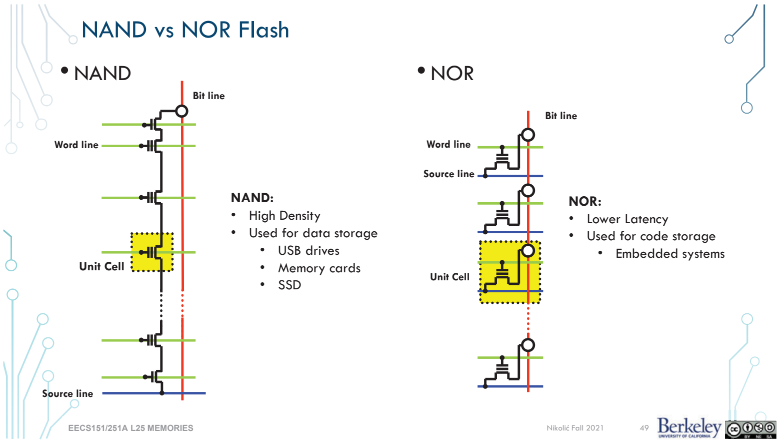


Flash Organization

NAND vs NOR Flash







NAND vs NOR Flash

- NAND

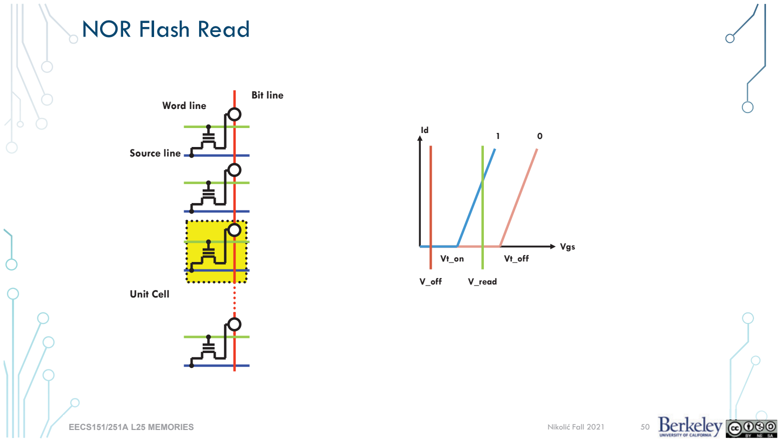
NAND:

- High Density
- Used for data storage
  - USB drives
  - Memory cards
  - SSD

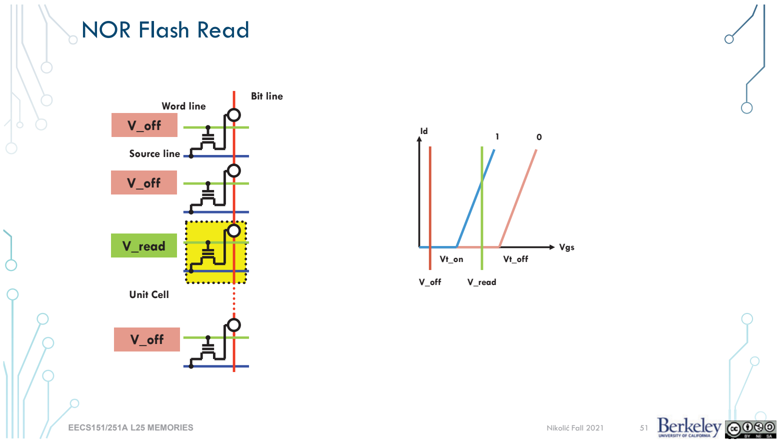
- NOR

NOR:

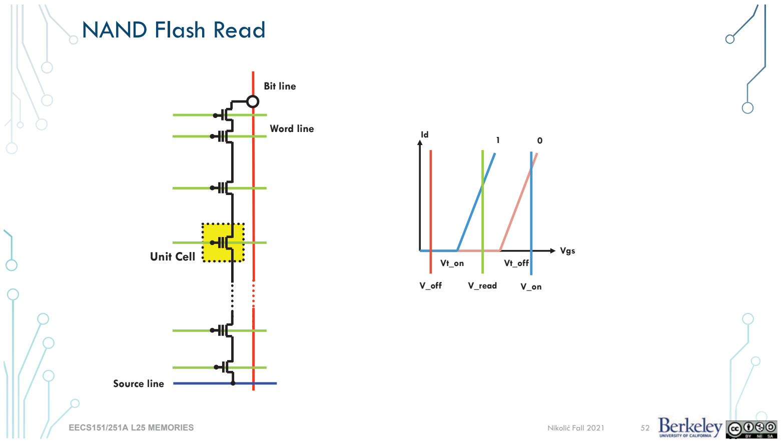
- Lower Latency
- Used for code storage
  - Embedded systems



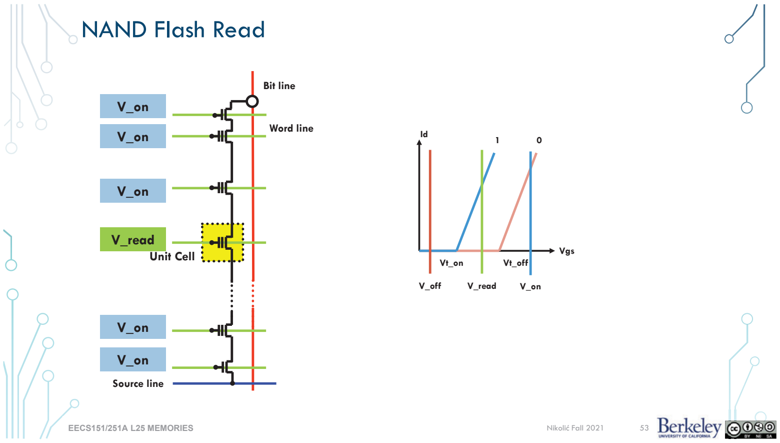
NOR Flash Read



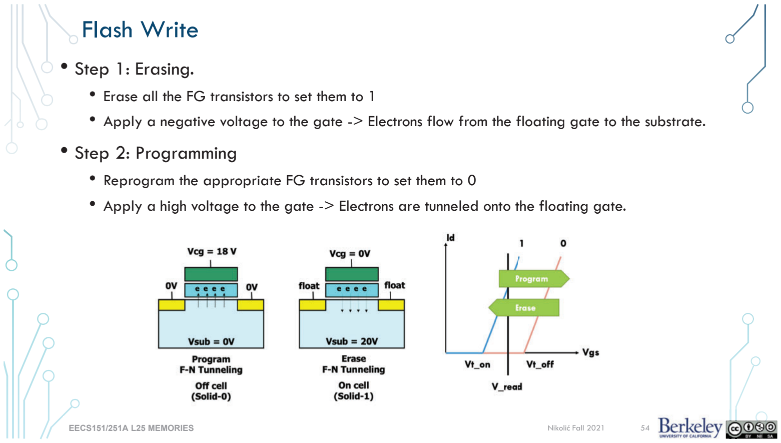
NOR Flash Read



NAND Flash Read

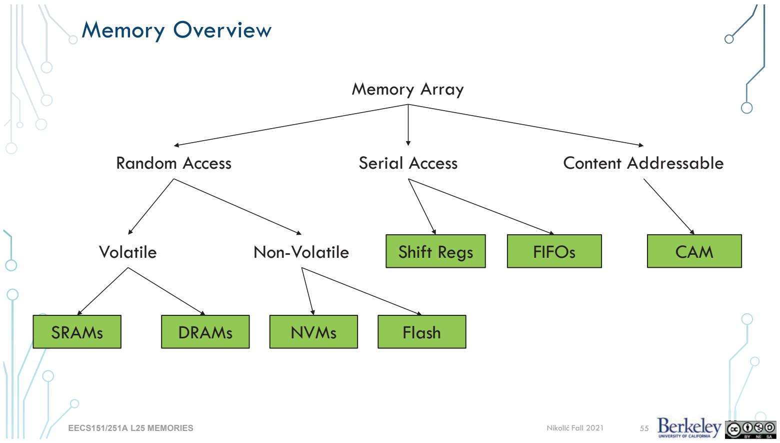


NAND Flash Read



Flash Write

- Step 1: Erasing.
  - Erase all the FG transistors to set them to 1
  - Apply a negative voltage to the gate -> Electrons flow from the floating gate to the substrate.
- Step 2: Programming
  - Reprogram the appropriate FG transistors to set them to 0
  - Apply a high voltage to the gate -> Electrons are tunneled onto the floating gate.

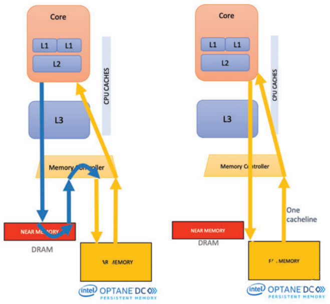


Memory Overview

Other Memories

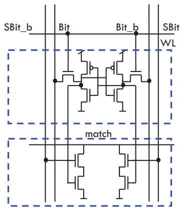
Non-Volatile Main Memory

- Intel Optane DC Persistent Memory
- Non-Volatile
- Storage based on resistance:
  - High resistivity : 0
  - Low Resistivity: 1
- High capacity:
  - 128, 256, 512 GB
- Modes:
  - Memory Mode
  - App-directed Mode



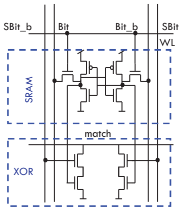
Content Addressable Memory

- Commonly used in translation lookaside buffers (TLBs).
- Matching asserts a matchline output for each world that contains a specified key

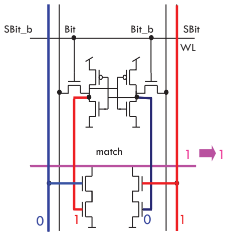


Content Addressable Memory

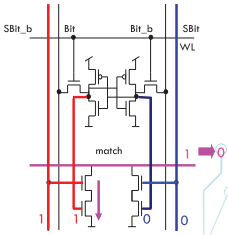
- Commonly used in translation lookaside buffers (TLBs).
- Matching asserts a matchline output for each word that contains a specified key



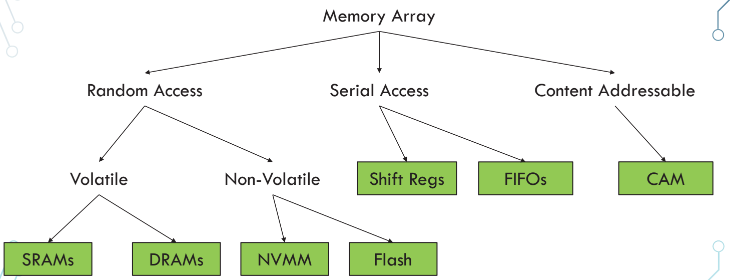
Match



Mismatch



Memory Overview



Memory Array

Random Access

Serial Access

Content Addressable

Volatile

Non-Volatile

Shift Regs

FIFOs

CAM

SRAMs

DRAMs

NVMM

Flash

Summary

- Multiple cache levels make memory appear both fast and big
- Direct mapped and set-associative cache
- Memory compilers generate SRAM blocks
- Several options for memory on FPGAs: Distributed, BlockRAM, UltraRAM
- Many more bits stored in DRAM and Flash
- Flash
  - Single-level vs multi-level
  - Read and Write Flash Cell
  - NAND vs NOR