Outline

- Memory Arrays
- SRAM Architecture
  - SRAM Cell
  - Decoders
  - Column Circuitry
  - Multiple Ports
- Serial Access Memories
Memory Arrays

- Random Access Memory
  - Read/Write Memory (RAM) (Volatile)
    - Static RAM (SRAM)
    - Dynamic RAM (DRAM)
  - Read Only Memory (ROM) (Nonvolatile)
    - Mask ROM
    - Programmable ROM (PROM)
    - Erasable Programmable ROM (EPROM)
    - Electrically Erasable Programmable ROM (EEPROM)
- Serial Access Memory
  - Shift Registers
  - Queues
    - Serial In Parallel Out (SIPO)
    - Parallel In Serial Out (PISO)
    - First In First Out (FIFO)
    - Last In First Out (LIFO)
- Content Addressable Memory (CAM)
- Shift Registers
- Queues
Array Architecture

- $2^n$ words of $2^m$ bits each
- If $n \gg m$, fold by $2^k$ into fewer rows of more columns

- Good regularity – easy to design
- Very high density if good cells are used
12T SRAM Cell

- Basic building block: SRAM Cell
  - Holds one bit of information, like a latch
  - Must be read and written

- 12-transistor (12T) SRAM cell
  - Use a simple latch connected to bitline
  - 46 x 75 $\lambda$ unit cell
6T SRAM Cell

- Cell size accounts for most of array size
  - Reduced cell size is worth peripheral complexity
- 6T SRAM Cell
  - Used in most commercial chips
  - Data stored in cross-coupled inverters
- Read:
  - Precharge bit, bit\_b
  - Raise wordline
- Write:
  - Drive data onto bit, bit\_b
  - Raise wordline
SRAM Read

- Precharge both bitlines high
- Then turn on wordline
- One of the two bitlines will be pulled down by the cell
- Ex: A = 0, A_b = 1
  - bit discharges, bit_b stays high
  - But A bumps up slightly
- **Read stability**
  - A must not flip
  - N1 >> N2
SRAM Write

• Drive one bitline high, the other low
• Then turn on wordline
• Bitlines overpower cell with new value
  • Ex: A = 0, A\_b = 1, bit = 1, bit\_b = 0
    • Force A\_b low, then A rises high

Writability
  • Must overpower feedback inverter
  • N2 >> P1
SRAM Sizing

- High bitlines must not overpower inverters during reads
- But low bitlines must write new value into cell
SRAM Column Example

Read

Write

Bitline Conditioning

More Cells

SRAM Cell

word_q1

bit_v1f

bit_b_v1f

out_v1r

out_b_v1r

φ₁

φ₂

word_q1

bit_v1f

out_v1r

write_q1

φ₁

φ₂

data_s1

H

H
SRAM Column Example

Read

Write

More Cells

More Cells

Bitline Conditioning

Bitline Conditioning

SRAM Cell

SRAM Cell

word_q1

word_q1

bit_v1f

bit_v1f

out_v1r

out_v1r

write_q1

data_s1

φ
φ

φ
φ
SRAM Layout

- Cell size is critical: $26 \times 45 \lambda$ (even smaller in industry)
- Tile cells sharing $V_{\text{DD}}$, GND, bitline contacts
Thin Cell

• In nanometer CMOS
  • Avoid bends in polysilicon and diffusion
  • Orient all transistors in one direction

• Lithographically friendly or thin cell layout fixes this
  • Also reduces length and capacitance of bitlines
Commercial SRAMs

- Five generations of Intel SRAM cell micrographs
  - Transition to thin cell at 65 nm
  - Steady scaling of cell area
Decoders

- $n:2^n$ decoder consists of $2^n$ n-input AND gates
  - One needed for each row of memory
  - Build AND from NAND or NOR gates
  - Need complementary address bits

Static CMOS

- Diagram of a static CMOS decoder for 4 words (A0, A1).
- Each word has a dedicated AND gate.

Pseudo-nMOS

- Diagram of a pseudo-nMOS decoder for 16 words (A0, A1).
- Uses inverters and NAND gates to select the correct word.
Decoder Layout

- Decoders must be pitch-matched to SRAM cell
  - Requires very skinny gates

![Diagram of decoder layout]
Large Decoders

- For $n > 4$, NAND gates become slow
  - Break large gates into multiple smaller gates
Large Decoders

- For $n > 4$, NAND gates become slow
  - Break large gates into multiple smaller gates
- Note that many first-stage gates are redundant

```
A3 A2 A1 A0

All take \(/A3\cdot/A2\)
```
Predecoding

• Many of these gates are redundant
  • Factor out common gates into predecoder
  • Saves area
  • Same path effort
Column Circuitry

• Some circuitry is required for each column
  • Bitline conditioning
  • Sense amplifiers
  • Column multiplexing
Bitline Conditioning

- Precharge bitlines high before reads

- Equalize bitlines to minimize voltage difference when using sense amplifiers
Sense Amplifiers

- Bitlines have many cells attached
  - Ex: 32-kbit SRAM has 128 rows x 256 cols
  - 128 cells on each bitline
- $t_{pd} \propto (C/I) \Delta V$
  - Even with shared diffusion contacts, 64C of diffusion capacitance (big C)
  - Discharged slowly through small transistors (small I)
- Sense amplifiers are triggered on small voltage swing (reduce $\Delta V$ to 100-300 mV)
Differential Pair Amp

- Differential pair requires no clock
- But always dissipates static power

![Differential Pair Amplifier Diagram]

- \text{P1} and \text{P2}
- \text{N1} and \text{N2}
- \text{N3}
- \text{sense_b}
- \text{bit}
- \text{bit_b}
Clocked Sense Amp

- Clocked sense amp saves power
- Requires sense_clk after enough bitline swing
- Isolation transistors cut off large bitline capacitance after positive FB engaged to speed up settling on sense, sense_b

![Clocked Sense Amp Diagram]

- sense_clk
- isolation transistors
- regenerative feedback
Twisted Bitlines

- Sense amplifiers also amplify noise
  - Coupling noise is severe in modern processes
  - Try to couple equally onto bit and bit_b
  - Done by *twisting* bitlines

b0  b0_b  b1  b1_b  b2  b2_b  b3  b3_b
Column Multiplexing

• Recall that array may be folded for good aspect ratio
• Ex: 2 kword x 16 folded into 256 rows x 128 columns
  • Must select 16 output bits from the 128 columns
  • Requires 16 8:1 column multiplexers
Tree Decoder Mux

- Column mux can use pass transistors
  - Use nMOS only, precharge outputs
- One design is to use $k$ series transistors for $2^k$:1 mux
  - No external decoder logic needed

![Diagram of Tree Decoder Mux]

Y to sense amps and write circuits
Single Pass-Gate Mux

- Or eliminate series transistors with separate decoder
Ex: 2-way Muxed SRAM

More Cells

A0

write0_q1

word_q1

φ2

More Cells

A0

φ2

write1_q1

data_v1
Multiple Ports

- We have considered single-ported SRAM
  - One read or one write on each cycle
- Multiported SRAM are needed for register files
- Examples:
  - Multicycle MIPS must read two sources or write a result on some cycles
  - Pipelined MIPS must read two sources and write a third result each cycle
  - Superscalar MIPS must read and write many sources and results each cycle
Dual-Ported SRAM

- Simple dual-ported SRAM
  - Two independent single-ended reads
  - Or one differential write

- Do two reads and one write by time multiplexing
  - Read during ph1, write during ph2
Multi-Ported SRAM

- Adding more access transistors hurts read stability
- Multiported SRAM isolates reads from state node
- Single-ended bitlines save area
- Area estimate:
  - #WL tracks
  - #BL tracks
  - + 3 (due to int. wiring)
Cell Stability

- Cell should not be accidentally written by noise (hold margin) or during read (read margin) and must be writeable (write margin)
- Quantified by the hold margin, read margin, and write margin
- Indicates the amount of noise that can be tolerated before malfunction
- Static Margin: Robustness to DC source
- Dynamic Margin: Robustness to a transient source
  - Typically > Static Margin
Cell Stability

- Simulate static read margin.
  - Set $V_n=0$, Drive each side respectively and plot $V_1$ vs. $V_2$. Generates “Butterfly plot”

- Intersections are (metastable) points

- Static Noise Margin is length of largest square that fits in both sides
Noise > SNM Flips Stored Bit

• $V_n \sim 0.36 > 0.32$; Assume $V_1 = 0$, so $V_{1n} = 0.36(a_x)$
• $V_2 \sim 0.55(a_y)$, $V_{2n} = 0.16$ (b),
• $V_1 \sim 1.0$ Bit is flipped
Read Stability

- Static read margin – Want to avoid read upset (accidental write)
- \( \frac{W_{\text{pull-down}}}{W_{\text{access}}} = \text{Beta} \)
  - Larger improves read margin but increases area
- Lower VWL improves read margin, but slows read access
Write Stability

- Write Margin
- Want to ensure that there is only one stable state during write
- Test circuit is similar to read margin, but with one access transistor pulling low.
- Improved by higher $V_{WL}$, larger access transistor, weaker pull-up
SRAM Stability

- Mismatch adds variation to noise margin
- Roughly Gaussian, but some evidence of non-Gaussian long tails that can cause excess failures
- Higher VDD improves margins
  - $V_{\text{min}}$ = lowest reliable supply voltage: ~0.7V – 1.0V for 6T cell
- Dynamic noise margins allow for more aggressive scaling, but are somewhat difficult to model reliably
  - Ex: Slower access logic requires longer access time, worsens dynamic margin
Read Assist

• Techniques available to improve reading at low VDD
• Pulsed bit-lines to use dynamic margin (rather than static)
• Temporarily raise $V_{DD}$ during reads
• Set $V_{WL} < V_{DD}$ (hurts write margin)
Write Assist

- Allow reliable write with lower VDD
- $V_{BL} < 0$ – Extra current sink
- Raise VWL
- Float GND and/or VDD during writes
- Lower cell VDD (but not peripheral drive VDD) during writes
- Use 8T cell
Low-Power SRAM

- Use 8T cell to isolate against read upsets
- Improved read stability allows lower VDD.
- Also provides separate read and write ports
Leakage Reduction

- Static leakage can be a major component of power consumption for arrays that are usually off.
Large SRAMs

- Large SRAMs are split into subarrays for speed
- Adds peripheral circuits for each subarray, trades area for speed
- Ex: UltraSparc 512KB cache
  - 4 128 KB subarrays
  - Each have 16 8KB banks
  - 256 rows x 256 cols / bank
  - 60% subarray area efficiency
  - Also space for tags & control

[Shin05]
Serial Access Memories

- Serial access memories do not use an address
  - Shift Registers
  - Tapped Delay Lines
  - Serial In Parallel Out (SIPO)
  - Parallel In Serial Out (PISO)
  - Queues (FIFO, LIFO)
Shift Register

• *Shift registers* store and delay data
• Simple design: cascade of registers
  • Watch your hold times!
Denser Shift Registers

- Flip-flops aren’t very area-efficient
- For large shift registers, keep data in SRAM instead
- Move read/write pointers to RAM rather than data
  - Initialize read address to first entry, write to last
  - Increment address on each cycle