13. Memories and PLAs

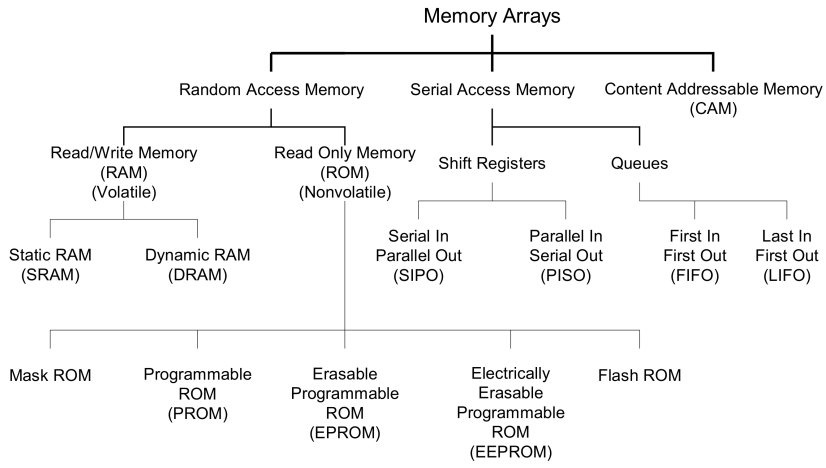
Jacob Abraham

Department of Electrical and Computer Engineering
The University of Texas at Austin

VLSI Design
Fall 2020

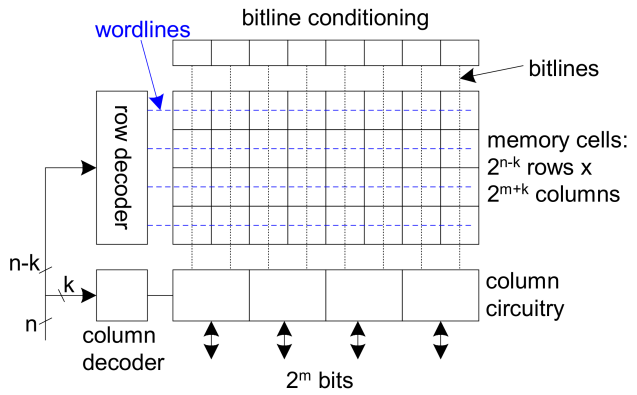
October 13, 2020

Taxonomy for Memory Arrays



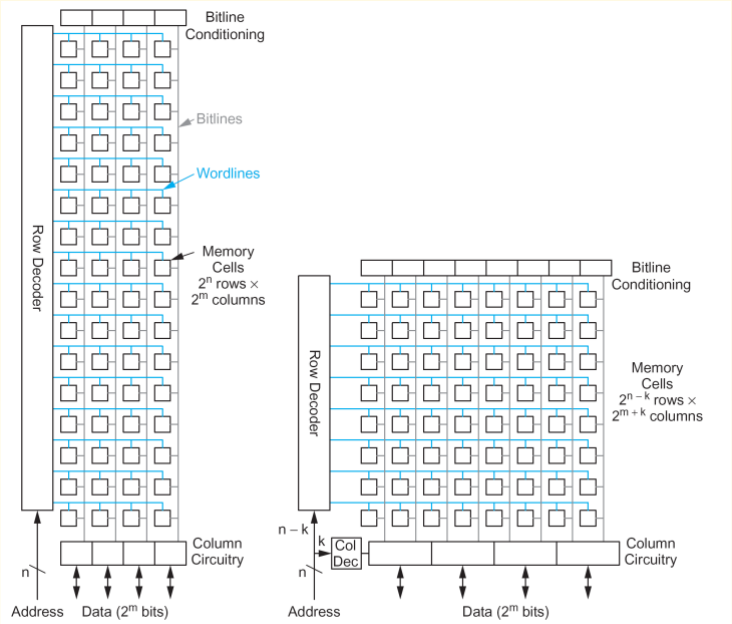
Array Architecture

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



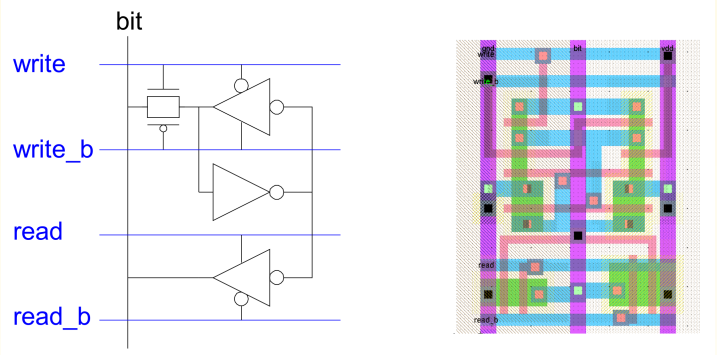
- Good regularity – easy to design
- Very high density if good cells are used

Different Memory Array Structures



12-Transistor SRAM Cell

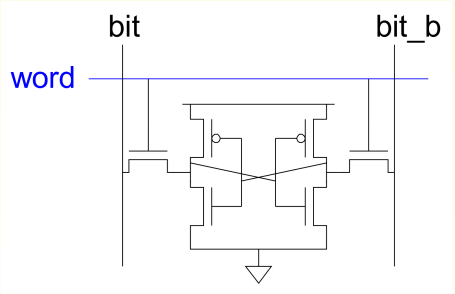
- Basic building block: SRAM Cell
  - Holds one bit of information, like a latch
  - Must be read and written
- 12-transistor (12T) SRAM cell
  - Use a simple latch connected to bitline
  - 46×75 λ unit cell



6-Transistor SRAM Cell

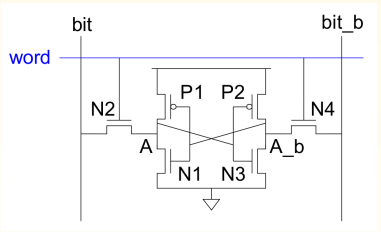
- Cell size accounts for most of array size
  - Reduce cell size at expense of complexity
- 6T SRAM Cell
  - Used in most commercial chips
  - Data stored in cross-coupled inverters

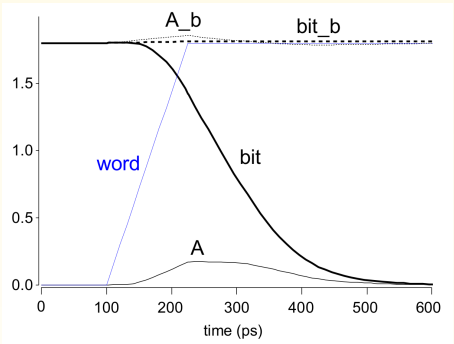
- Read:
  - Precharge bit, bit_b
  - Raise wordline
- Write
  - Drive data onto bit, bit_b
  - Raise wordline



SRAM Read

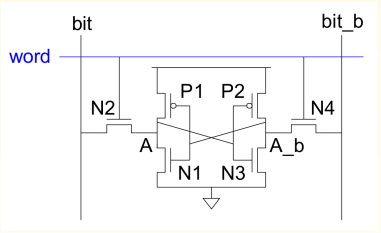
- Precharge both bitlines high
- Then turn on wordline
- One of two bitlines will be pulled down by the cell
- Example: A = 0, A_b = 1
  - Bit discharges, bit_b stays high
  - But A bumps up slightly
- **Read stability**
  - A must not flip
  - N1 >> N2

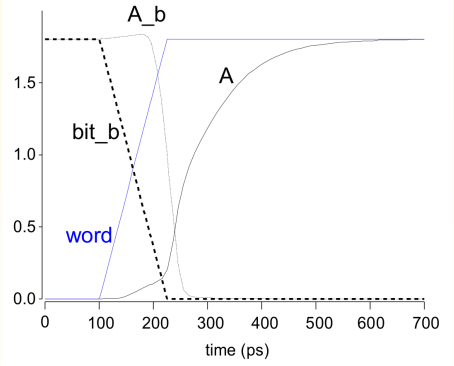




SRAM Write

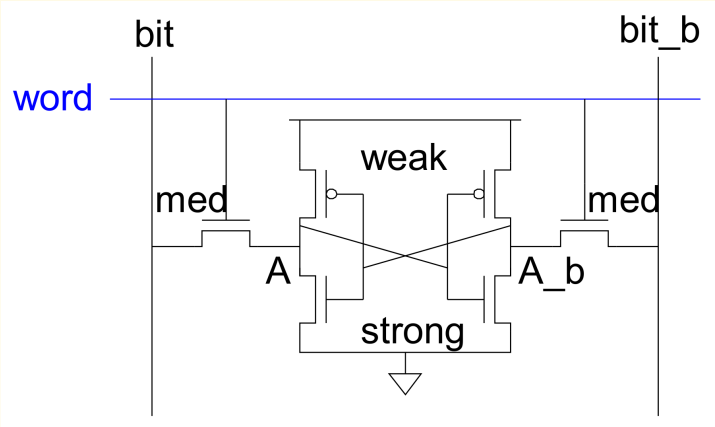
- Drive one bitline high, the other low
- Then turn on wordline
- Bitlines overpower cell with new value
- Example: A = 0, A_b = 1, bit = 1, bit_b = 0
  - Force A_b low, then A rises high
- **Writability**
  - Must overpower feedback inverter
  - N2 >> P1



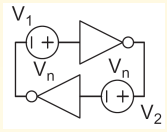


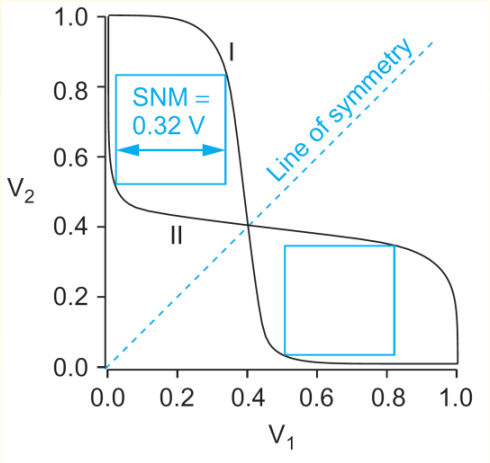
SRAM Sizing

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

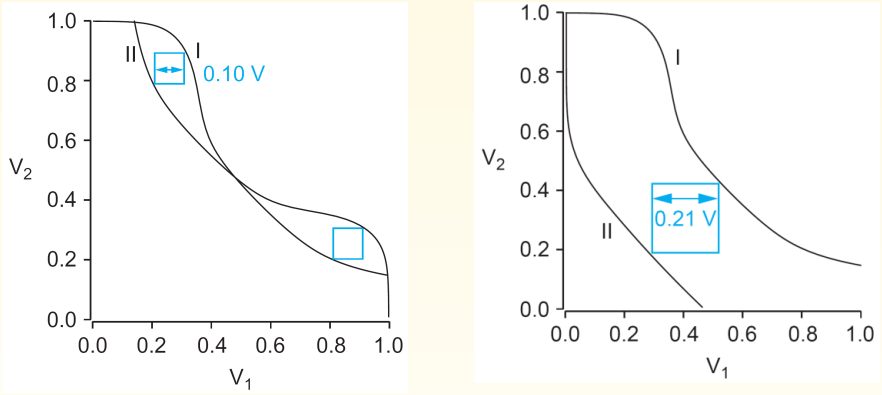


Hold Margin

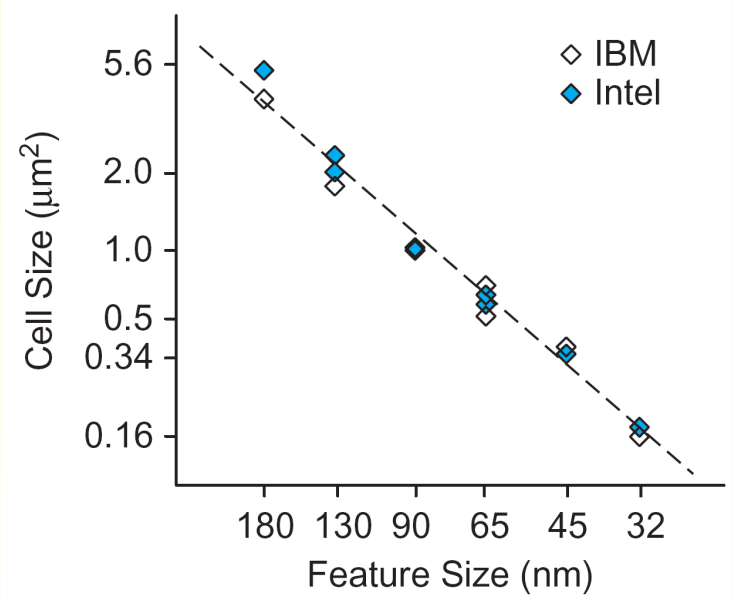




Read and Write Margins



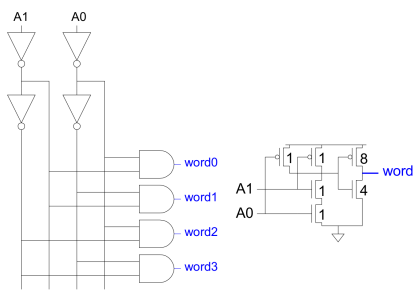
SRAM Cell Size vs. Feature Size



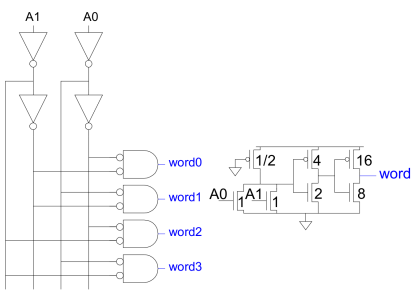
Decoders

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

Static CMOS

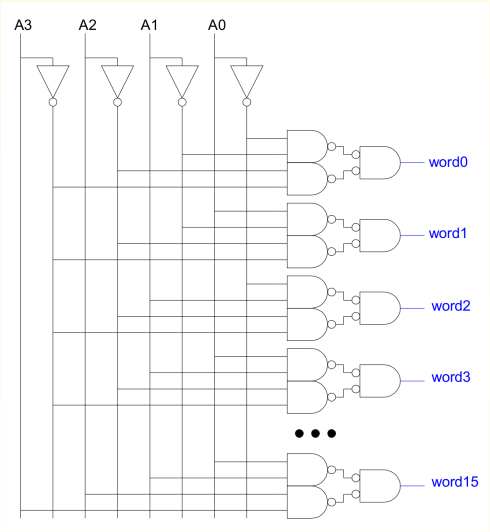


Pseudo-nMOS



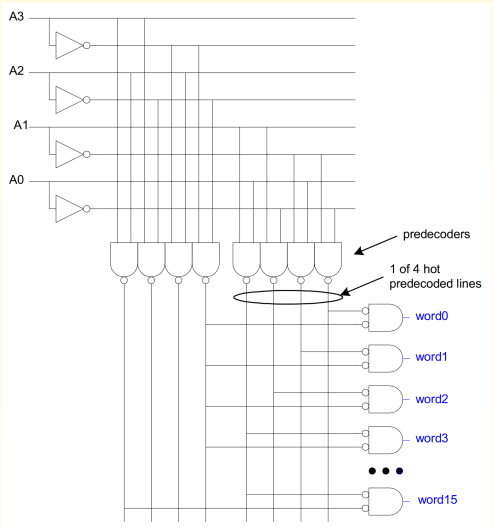
Large Decoders

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



Pre-Decoding

- Many of the gates are redundant
  - Factor out common gates into predecoder
  - Saves area
  - Same path effort

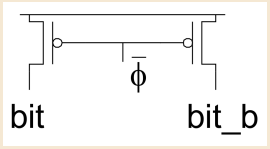


Column Circuitry

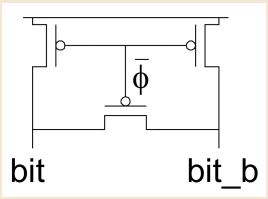
- Some circuitry is required for each column
  - Bitline conditioning
  - Sense amplifiers
  - Column multiplexing

Bitline Conditioning

- Precharge bitlines high before reads



- Equalize bitlines to minimize voltage difference when using sense amplifiers

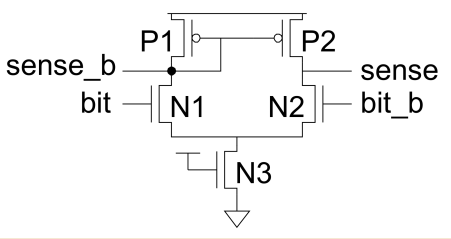


Sense Amplifiers

- Bitlines have many cells attached
  - Example, 32-kbit SRAM has 256 rows x 128 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 a small voltage swing (reduce ΔV)

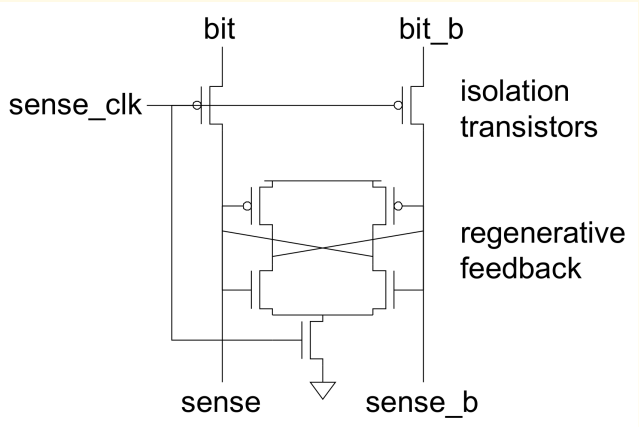
Example: Differential Pair Amplifier

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



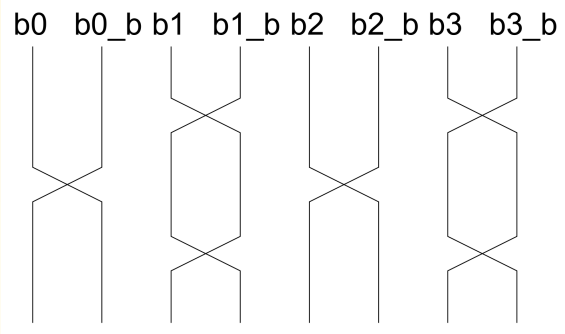
Clocked Sense Amplifier

- Clocked sense amp saves power
- Requires sense_clk after enough bitline swing
- Isolation transistors cut off large bitline capacitance



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

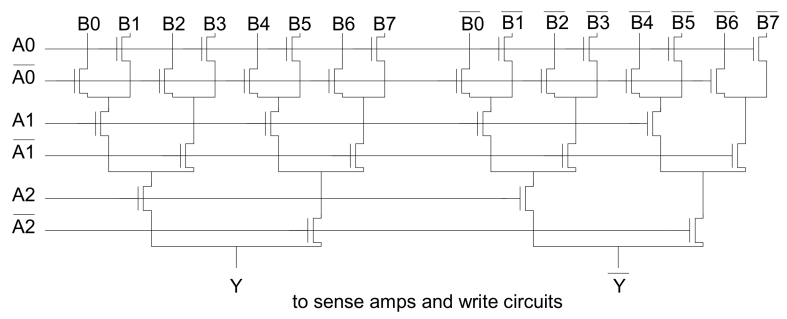


Column Multiplexing

- Recall that array may be folded for good aspect ratio
- Example: 2K word x 16 array folded into 256 rows $\times$ 128 columns
  - Must select 16 output bits from the 128 columns
  - Requires 16 8:1 column multiplexers

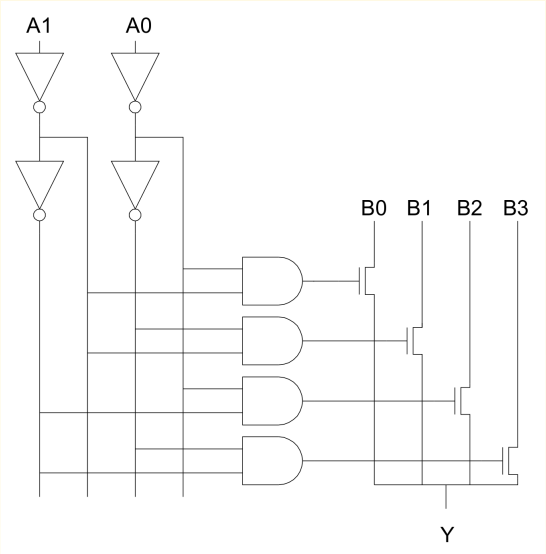
Tree Decoder Multiplexer

- 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

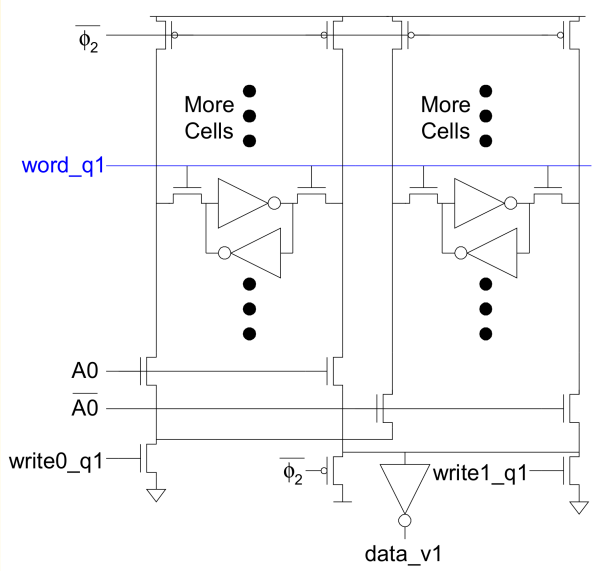


Single Pass-Gate Multiplexer

- Eliminate series transistors with separate decoder



Example:2-Way Muxed SRAM

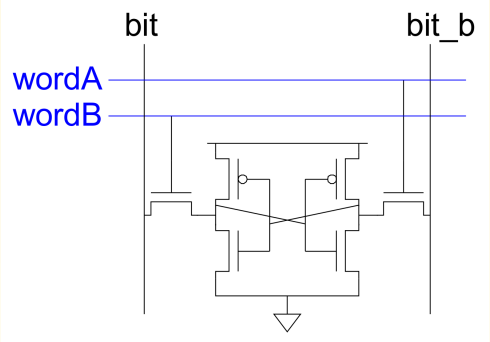


Multiple Ports

- We have considered single-ported SRAM
  - One read or one write on each cycle
- Multiported SRAMs 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



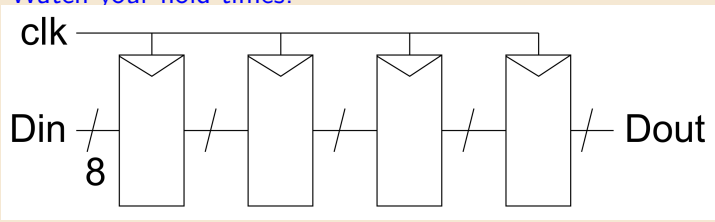
- Do two reads and one write by time multiplexing
  - Read during ph1, write during ph2

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 (Stacks))
- Some of these used in circuitry for communications

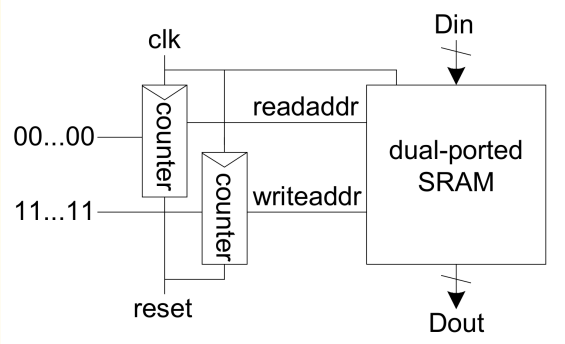
Shift Registers Store and Delay Data

- Simple design: cascade of registers
- Watch your hold times!



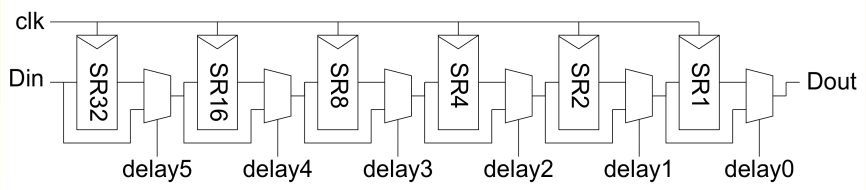
Denser Shift Registers

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



Tapped Delay Line

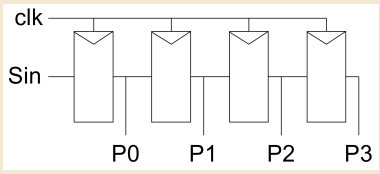
- Shifter register with a programmable number of stages
- Set number of stages with delay controls to mux
  - Example, 0 – 63 stages of delay



Serial/Parallel Conversion

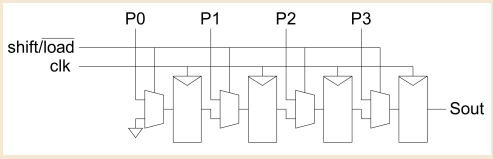
Serial In Parallel Out

- 1-bit shift register reads in serial data
  - After N steps, presents N-bit parallel output



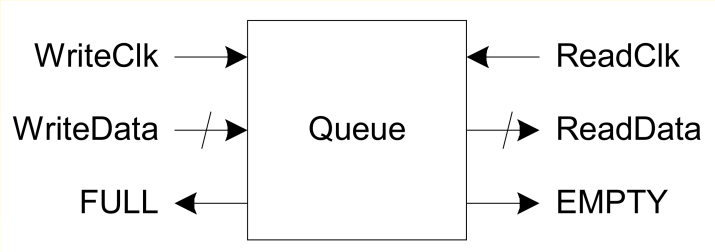
Parallel In Serial Out

- Load all N bits in parallel when shift = 0
  - Then shift one bit out per cycle



Queues

- Queues allow data to be read and written at different rates
- Read, Write each use their own clock, data
- Queue indicates whether it is full or empty
- Build with SRAM and read/write counters (pointer

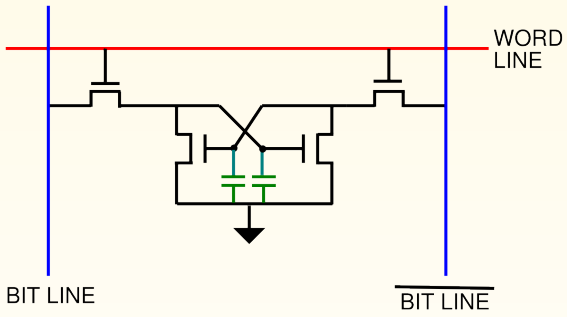


FIFO, LIFO Queues

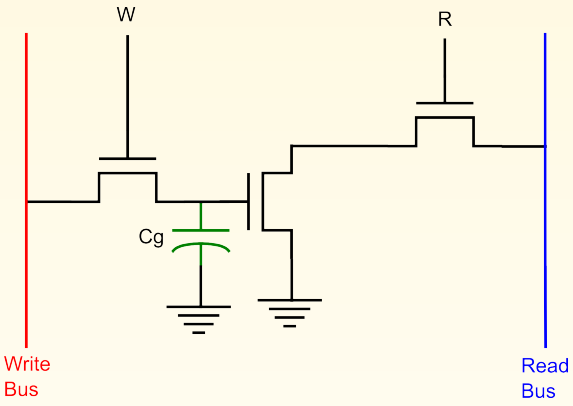
- First In First Out (FIFO)
  - Initialize read and write pointers to first element
  - Queue is EMPTY
  - On write, increment write pointer
  - If write almost catches read, Queue is FULL
  - On read, increment read pointer
- Last In First Out (FIFO)
  - Also called a stack
  - Use a single stack pointer for read and write

4-Transistor Dynamic RAM Cell

- Remove the two p-channel transistors from the static RAM cell to get a four-transistor dynamic RAM cell
- Data stored as charge on gate capacitors (complementary nodes)
- Data must be refreshed regularly
- Dynamic cells must be designed very carefully

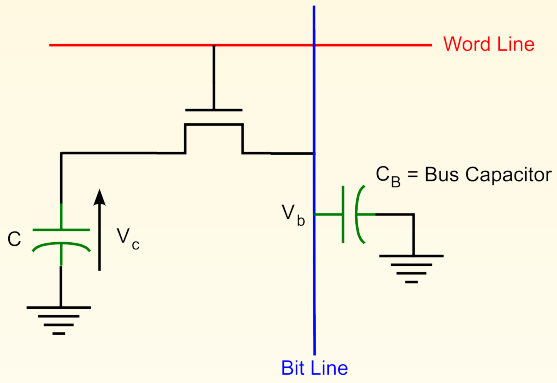


3-Transistor Dynamic RAM Cell



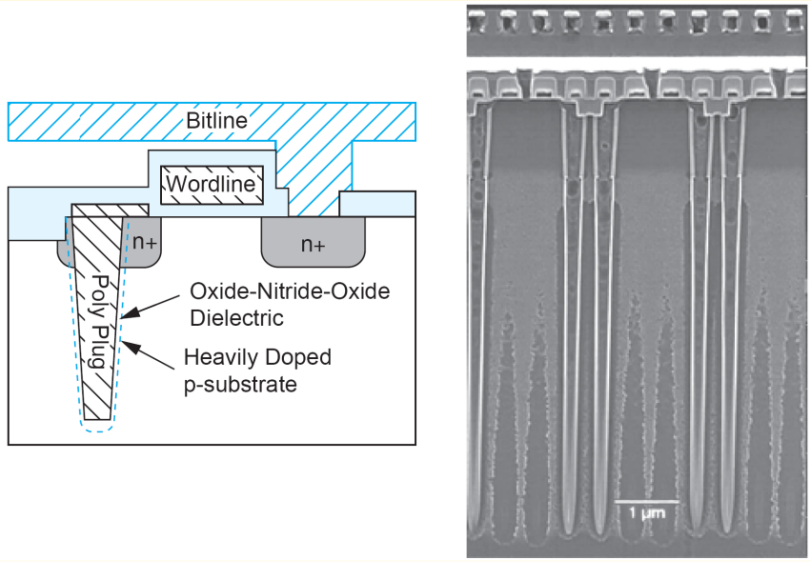
- Data stored on the gate of a transistor
- Need two additional transistors, one for write and the other for read control

1-Transistor Dynamic RAM Cell



- Cannot get any smaller than this: data stored on a (trench) capacitor C, need a transistor to control data
- Bit line normally precharged to $1/2$ VDD (need a well-designed sense amplifier)
- Value of capacitances must be chosen very carefully; voltages on stored bit and bit-line affected by charge sharing

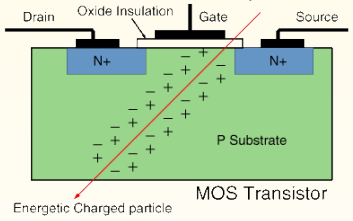
Trench Capacitor

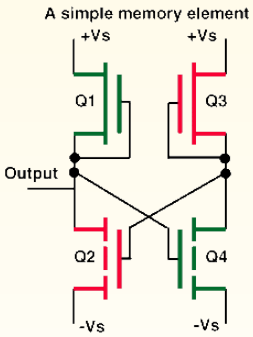


Single-Event Upsets

- High-energy particle produces electron-hole pairs in substrate; when collected at source and drain, will cause current pulse
  - Cosmic Radiation
- A "bit-flip" can occur in the memory cell due to the charge generated by the particle – called a "single-event upset"
- Seen in spacecraft electronics in the past, now in computers on the ground

Source: Aerospace Corporation
Interaction of a Cosmic Ray and Silicon





Embedded Memory IP (Intellectual Property)

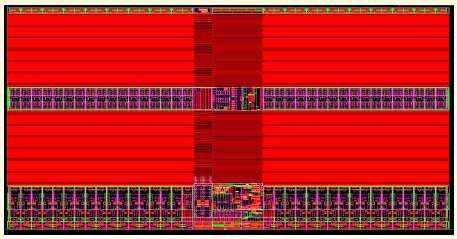
Example, from ARM

- Single Port SRAM
- Dual Port SRAM
- Single Port Register file
- Two Port Register file
- Via and Diffusion Programmable ROM

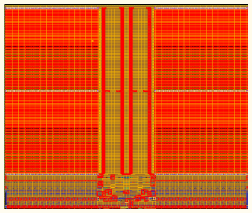
Memory Compilers

- Automatically generate memory structures
- High density, high speed and low power SRAMs
- Over 15 different foundries and 65 process variants from 28nm to 250nm

ARM Embedded Memory IP



SRAM IP

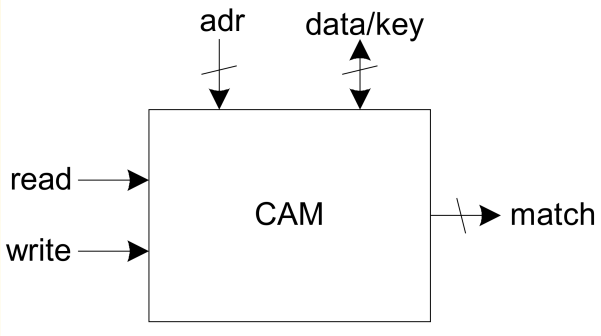


Register File IP

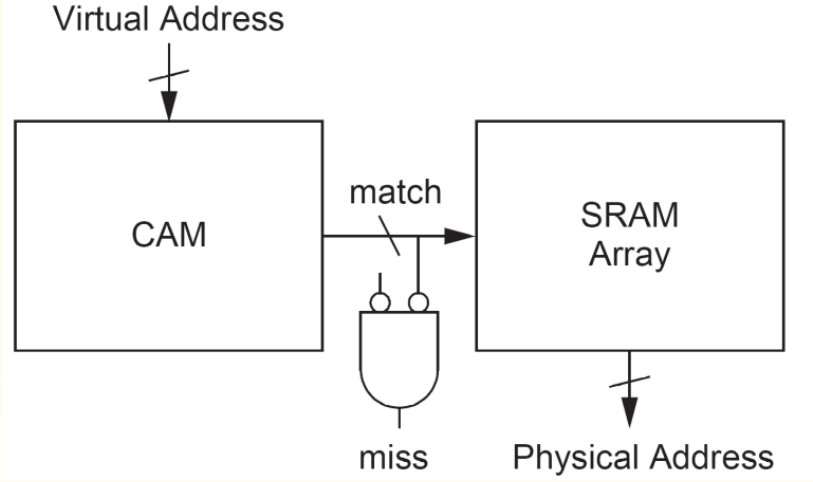
Source: ARM (http://www.arm.com/products/physical-ip/embedded-memory-ip/index.php)

Content Addressable Memories (CAMs)

- Extension of ordinary memory (e.g., SRAM)
  - Read and write memory as usual
  - Also match to see which words contain a key

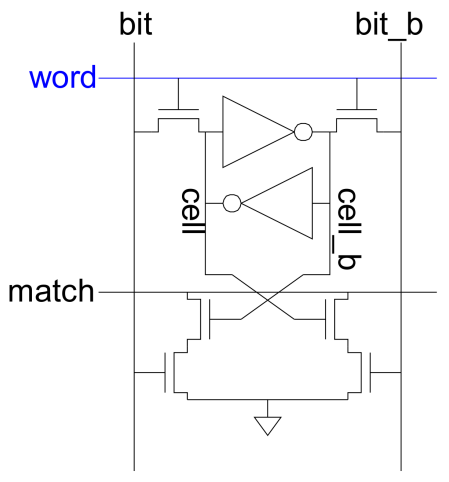


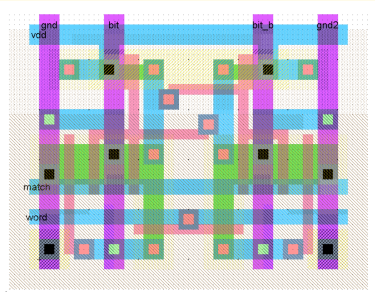
Translation Lookaside Buffer (TLB) Using CAM



10-Transistor CAM Cell

- Add four match transistors to 6T SRAM
  - 56×43 λ unit cell

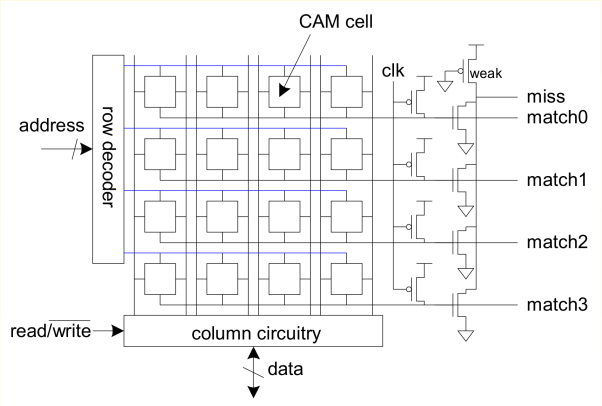




CAM Cell Operation

- Read and write like ordinary SRAM
- Additional "match" operation

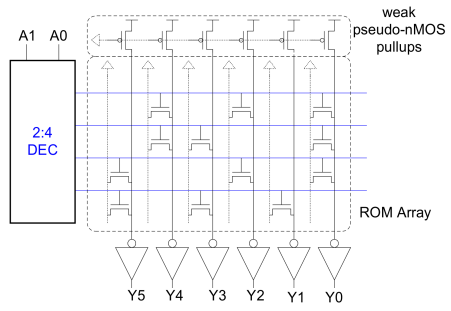
- For matching:
  - Leave wordline low
  - Precharge matchlines
  - Place key on bitlines
  - Matchlines evaluate
- Miss line
  - Pseudo-nMOS NOR of match lines
  - Goes high if no words match



Read-Only Memories (ROMs)

- Read-Only Memories are nonvolatile
  - Retain their contents when power is removed
- Mask-programmed ROMs use one transistor per bit
  - Presence or absence determines 1 or 0

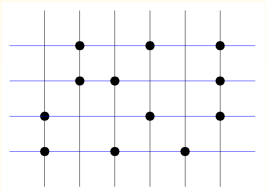
Example: 4-word × 6-bit ROM



Looks like 6 4-input pseudo-nMOS NORs

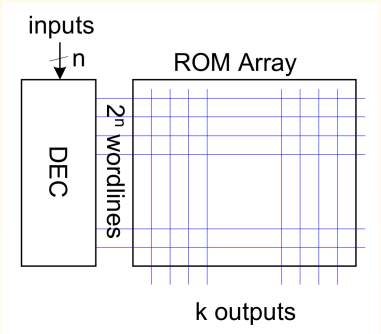
Can be represented by a "dot diagram"
Dots indicate 1s in ROM

```
Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
```

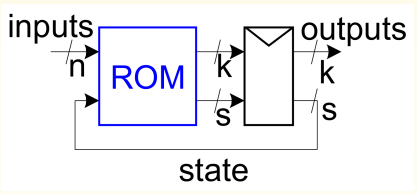


Building Logic with ROMs

- ROM as lookup table containing truth table
  - n inputs, k outputs requires 2^n words × k bits
  - Changing function is easy – reprogram ROM

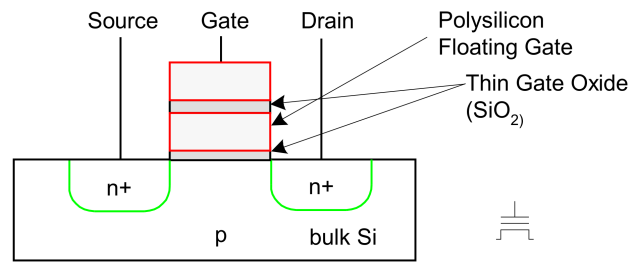


- Finite State Machine
  - n inputs, k outputs, s bits of state
  - Build with $2^{n+s} \times (k+s)$ bit ROM and $(k+s)$ bit register

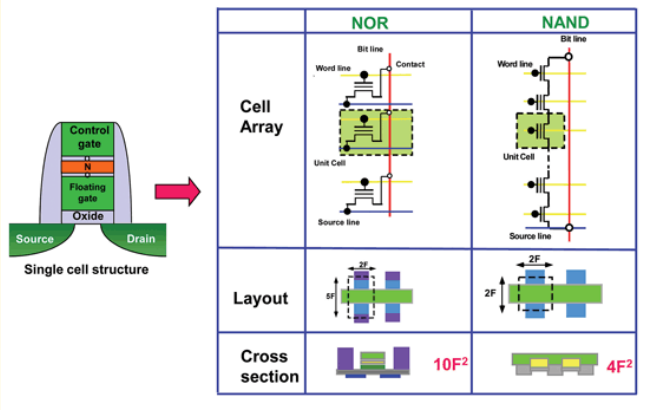


PROMs and EPROMs

- Programmable ROMs (PROMs)
  - Build array with transistors at every site
  - Burn out fuses to disable unwanted transistors
- Electrically Programmable ROMs (EPROMs)
  - Use floating gate to turn off unwanted transistors
  - EPROM, EEPROM, Flash

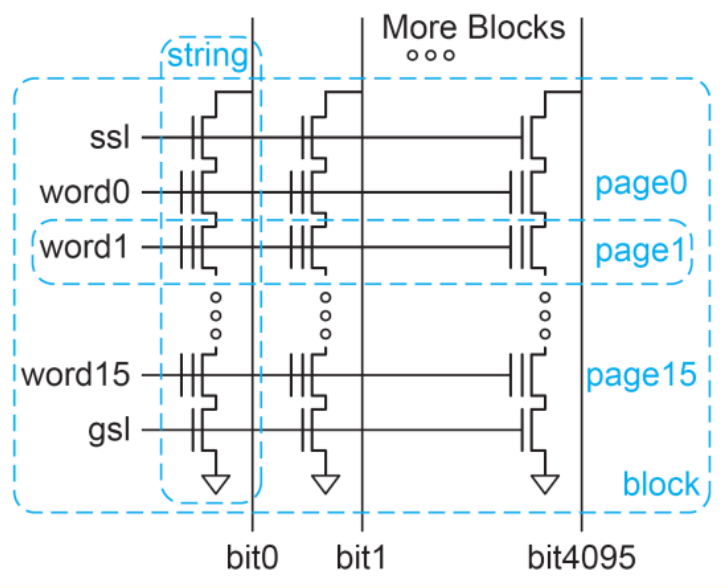


Flash Memory

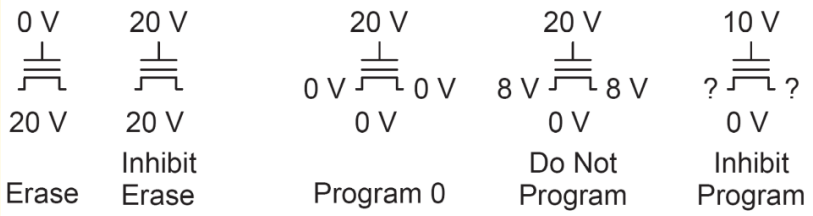


Source: www.ctimes.com.tw

NAND Flash String



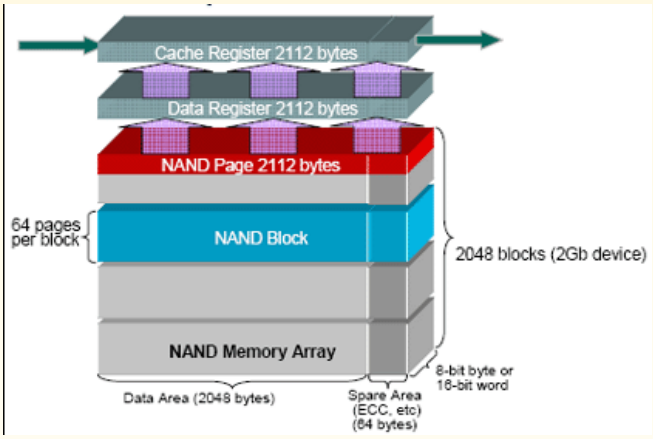
Erase and Program Operations



Issues: Retention Time, Endurance

"Wear leveling" to increase endurance

Flash Memory Architecture



Source: EE Times

Programmable Logic Arrays (PLAs)

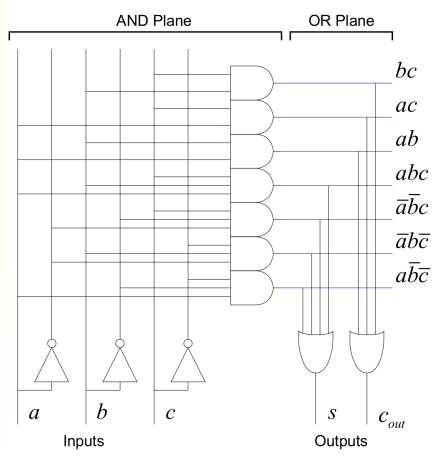
A Programmable Logic Array performs any function in sum-of-products form

- Literals: inputs and complements
- Products terms: AND of literals
- Outputs: OR of Product terms

Example: Full Adder

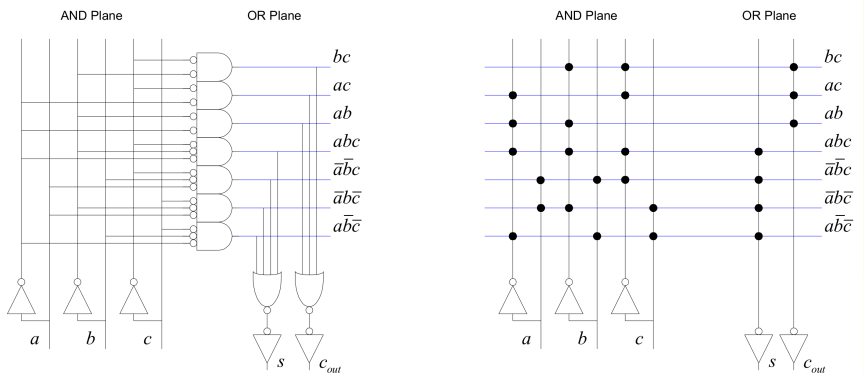
$s = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc$

$c = ab + bc + ac$

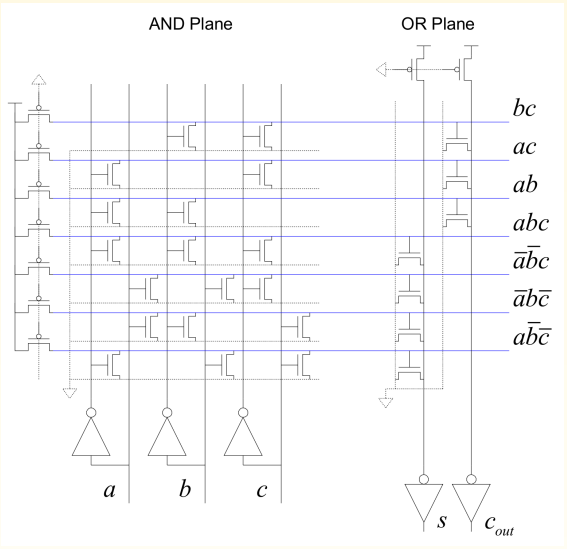


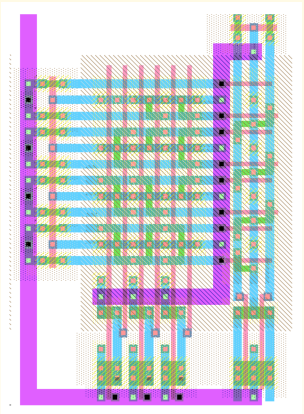
NOR-NOR PLAs

- ANDs and ORs not very efficient in CMOS
- Dynamic or Pseudo-nMOS NORs very efficient
- Use DeMorgan's Law to convert to all NORs

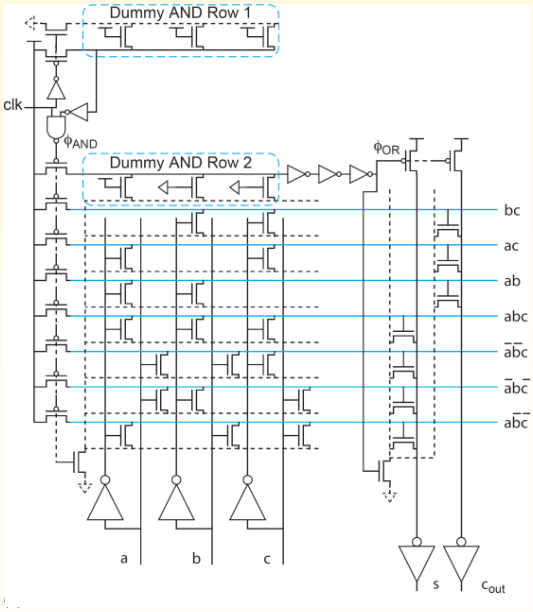


PLA Schematic and Layout





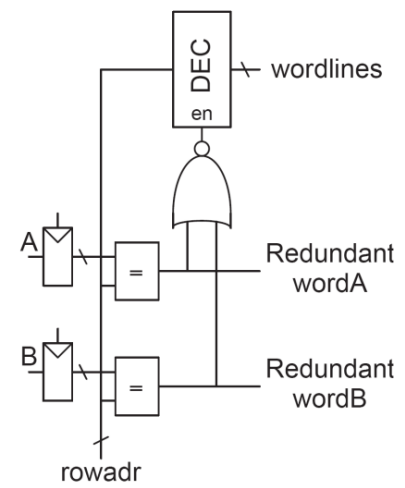
Dynamic PLA



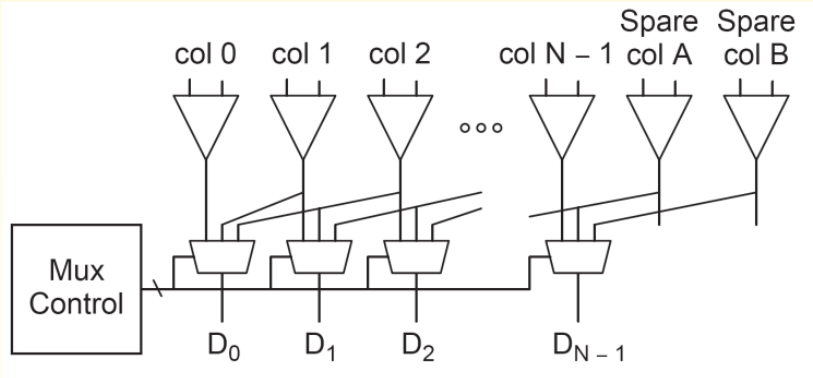
PLAs versus ROMs

- The OR plane of the PLA is like the ROM array
- The AND plane of the PLA is like the ROM decoder
- PLAs are more flexible than ROMs
  - No need to have 2^n rows for n inputs
  - Only generate the product terms that are needed
  - Take advantage of logic simplification

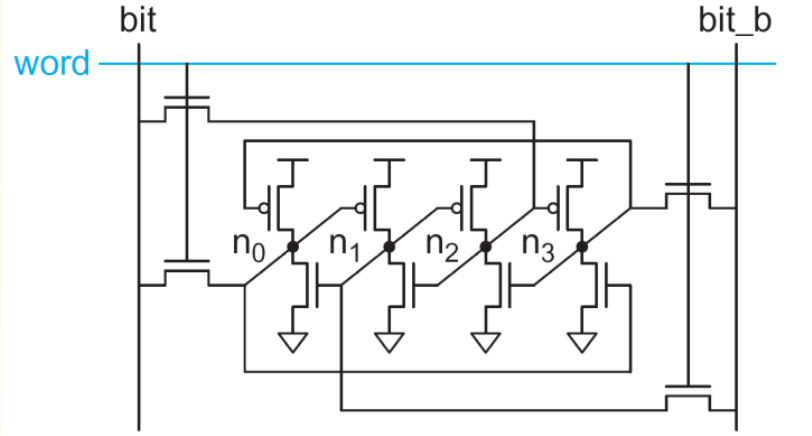
Row Redundancy to Tolerate Failures



Column Redundancy to Tolerate Failures



Radiation-Hardened SRAM Cell

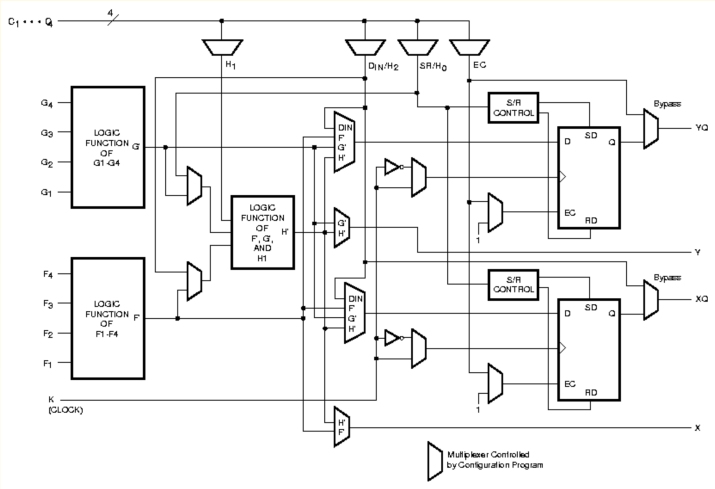


Alternative Solution: Use Error Correcting Codes

Field Programmable Logic Arrays (FPGAs)

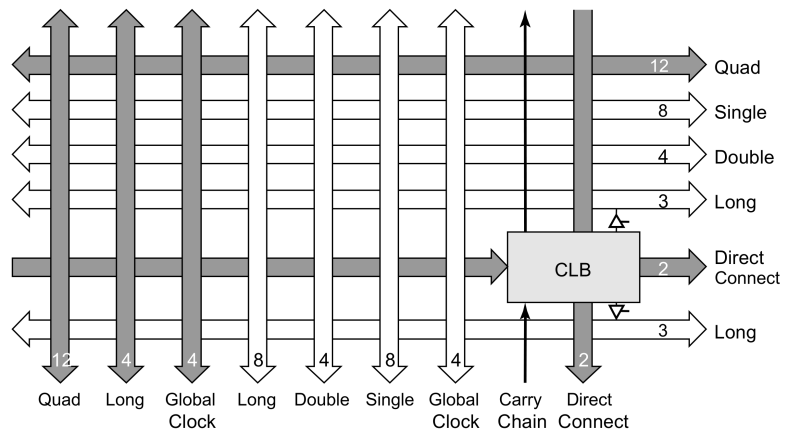
Lookup Table based logic cell

Xilinx 4000 Series

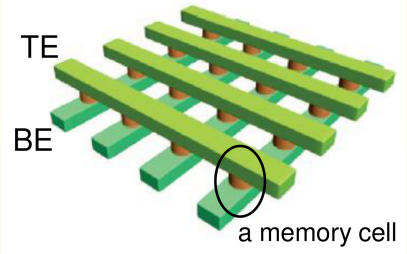


Xilinx 4000 Interconnect Architecture

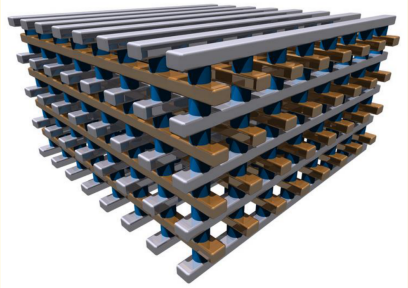
Source: Xilinx



Resistive RAM (RRAM)



2-D RRAM



3-D cross-point

Source: Crossbar, Inc.