# Memories & More

- Overview of Memories
- External Memories
  - SRAM (async, sync)
  - Flash
  - DRAM
- Memories in Verilog
- Memories on the FPGA

**EMAIL GIM teams!!!/teaming Issues**

# Memories: a practical primer

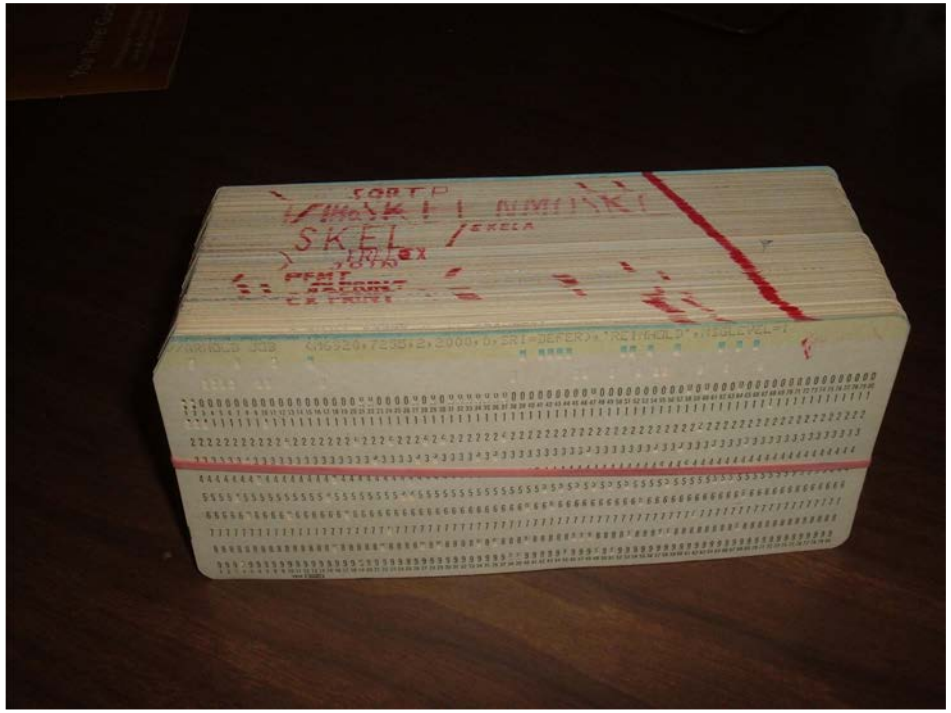
- The good news: huge selection of technologies
  - Small & faster vs. large & slower
  - Every year capacities go up and prices go down
  - Almost cost competitive with hard disks: high density, fast flash memories
    - Non-volatile, read/write, no moving parts! (robust, efficient)
- The bad news: perennial system bottleneck
  - Latencies (access time) haven't kept pace with cycle times
  - Separate technology from logic, so must communicate between silicon, so physical limitations (# of pins, R's and C's and L's) limit bandwidths
    - New hopes: capacitive interconnect, 3D IC's
  - Likely the limiting factor in cost & performance of many digital systems: designers spend a lot of time figuring out how to keep memories running at peak bandwidth
  - "It's the memory - just add more faster memory"

# How do we Electrically Remember Things?

- We can convey/transfer information with voltages that change over time
- How can we store information in an electrically accessible manner?
- Store in either:
  - Electric Field
  - Magnetic Field

# Mostly focus on rewritable

- Punched Cards have existed as electromechanical program storage since ~1800s
- We're mostly concerned with rewritable storage mechanisms today (cards were true ROMs)



*Computer program in punched card format*

https://en.wikipedia.org/wiki/Computer_programming_in_the_punched_card_era

# Electronic Memories in History

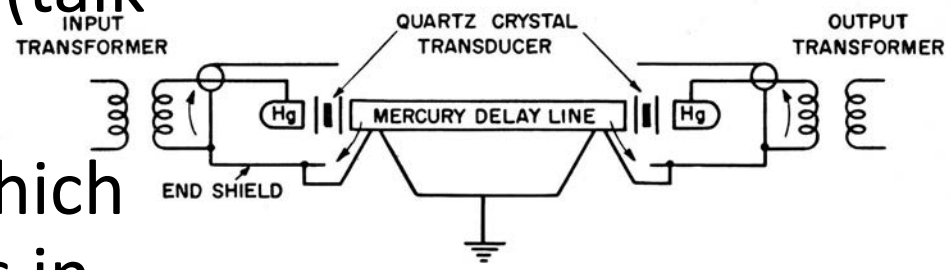
http://www.computerhistory.org/timeline/memory-storage/

- Drum Memory:
  - Information stored magnetically on large rotating metallic cylinder
  - Could read/write to it
- Did not require periodic refresh

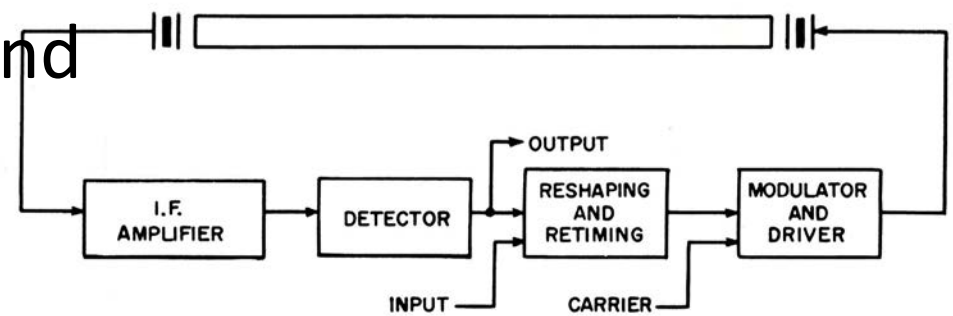
- Non-volatile (last after power cycles off)

# Delay Line Memory

- Early form of FIFO memory (talk about later)
- Generate a wave pattern which exists for a few milliseconds in mercury
- Recover on the other end and either reload or use
- Requires refresh circuitry
- Volatile (info lost soon after power cut)



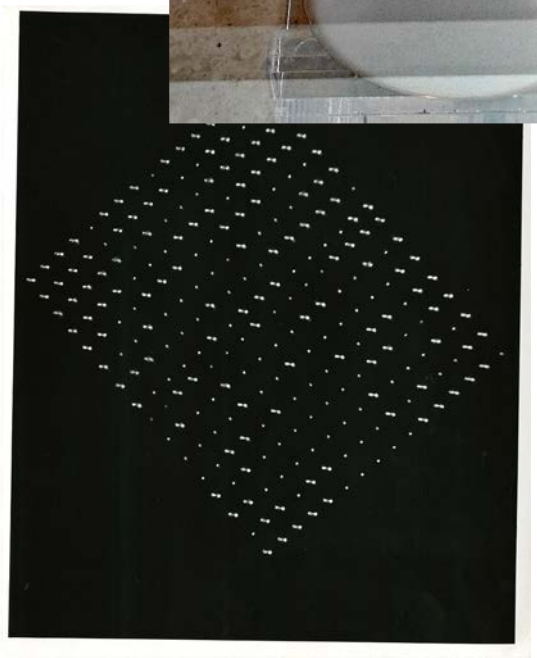

*Schematic diagram of circuit connections to the acoustic delay line used in NBS mercury memory.*




*Block diagram of the mercury memory system.*

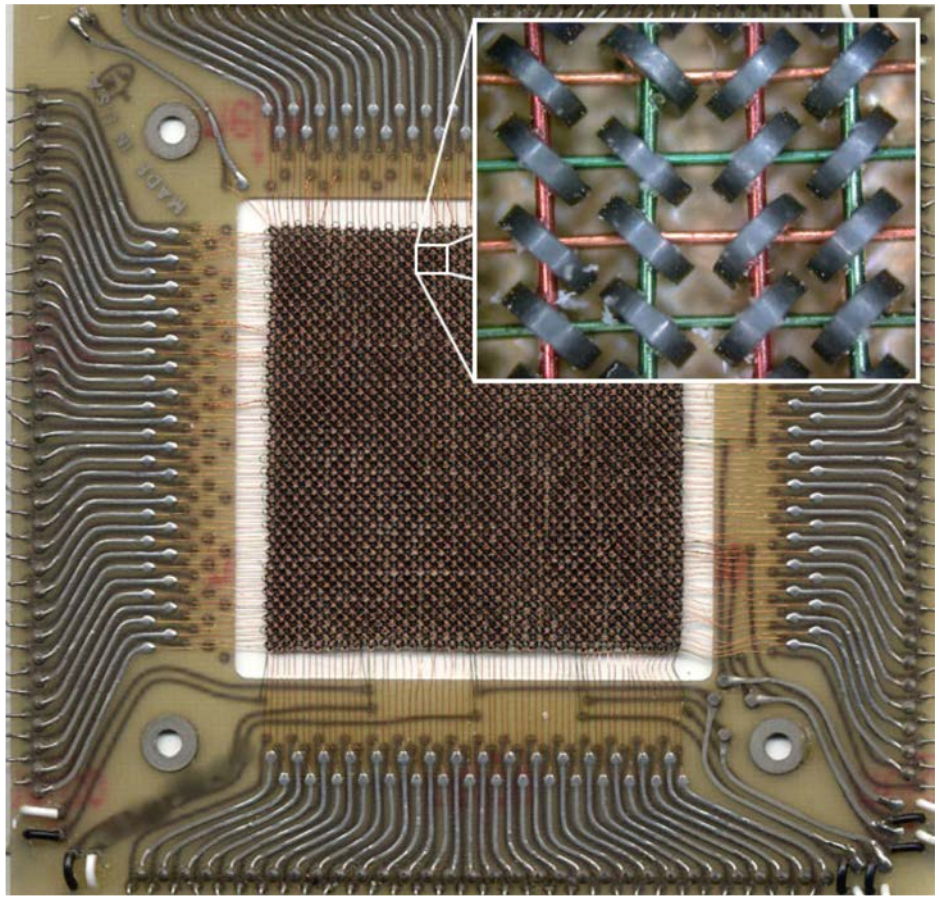
# William's Tube

- Take advantage of non-negligible decay time of phosphors on CRT to store data
- Project data image
- Little bit later (milliseconds) recover it .
- Either use it or reproject it for later use
- Requires periodic refresh (obv.)



# Core Memory

- MIT!
- Store 1's and 0's in the magnetic field of small torroids (magnetic cores)
- Where the term "core dump" comes from.
- Used up until mid 70's
- Few on display in fourth floor of 38
- Non volatile!



https://en.wikipedia.org/wiki/Magnetic-core_memory#/media/File:KL_Kernspeicher_Makro_1.jpg

# Memory Classification & Metrics

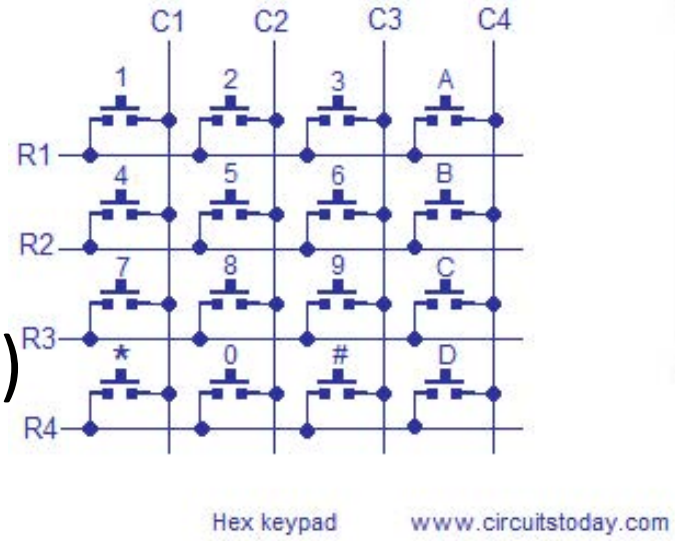
| Read-Write Memory | | Non-Volatile Read-Write Memory | Read-Only Memory |
|---|---|---|---|
| Random Access | Sequential Access | | |
| SRAM<br>DRAM | FIFO | EPROM<br>$E^2$PROM<br>FLASH | Mask-Programmed ROM |

Key Design Metrics:

1. Memory Density (number of bits/$mm^2$) and Size
2. Access Time (time to read or write) and Throughput
3. Power Dissipation

# Memory Array's (Inspiration in Switches)

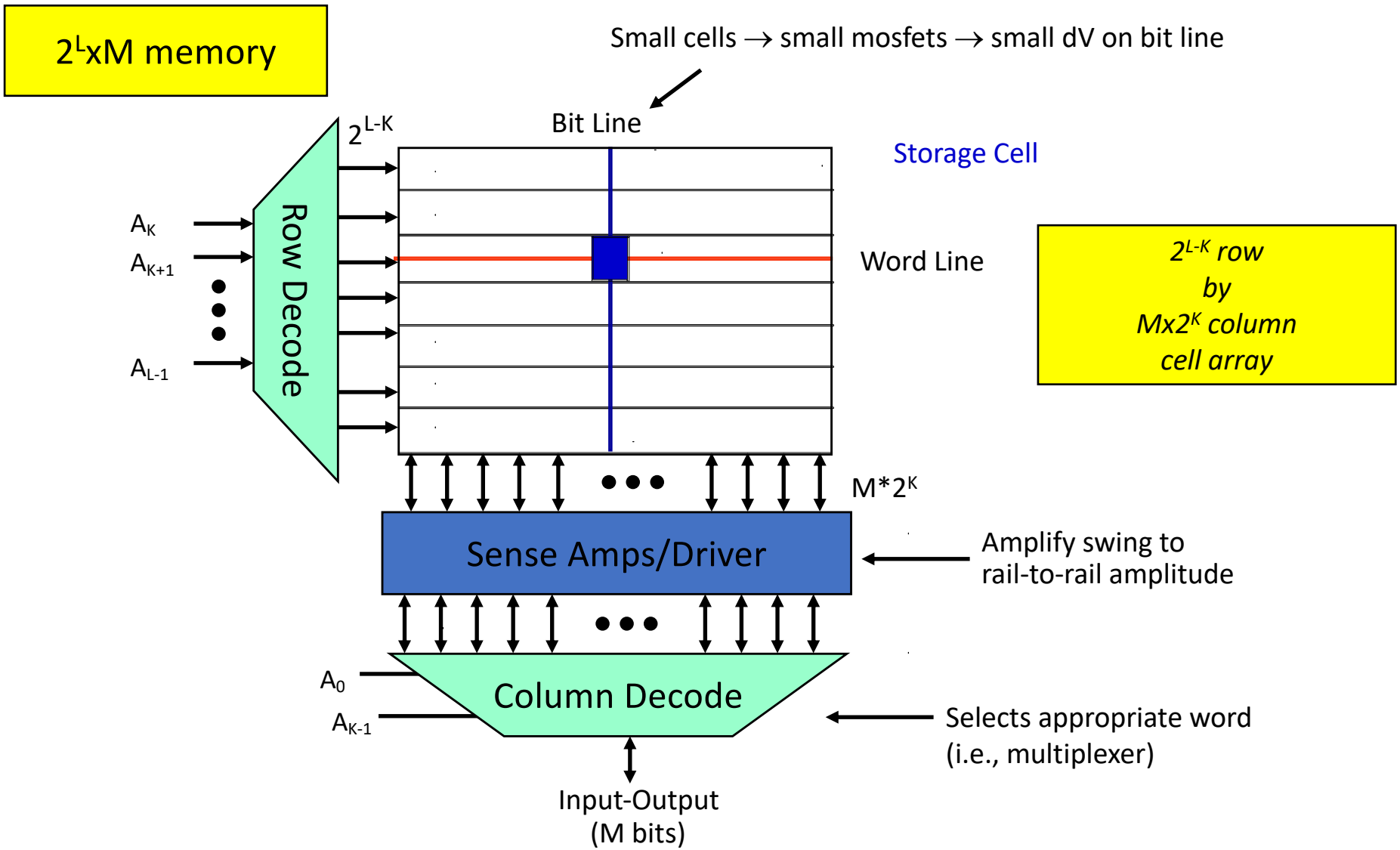
- If you have 16 switches, you can convey that using 16 independent wires (one-hot encoding)
- Alternatively if you assemble in an array/matrix, you can do with 8 wires (if you add some interfacing circuitry)
- Same situation in memory



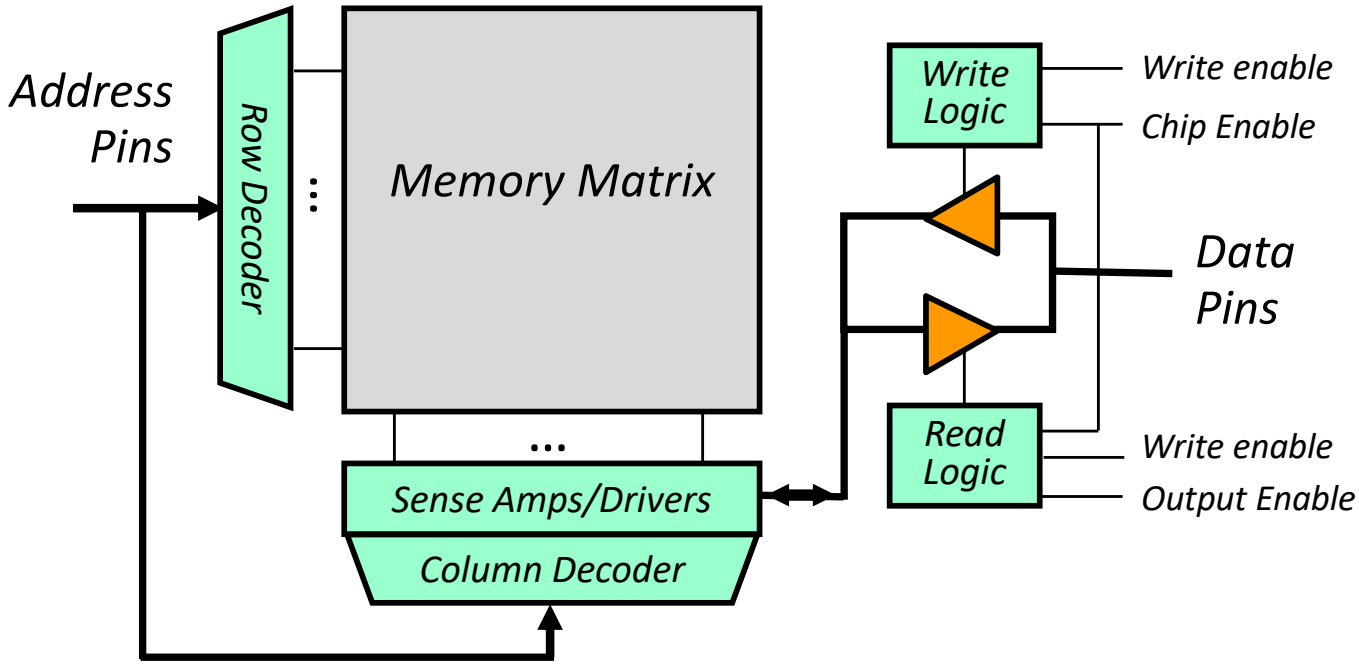


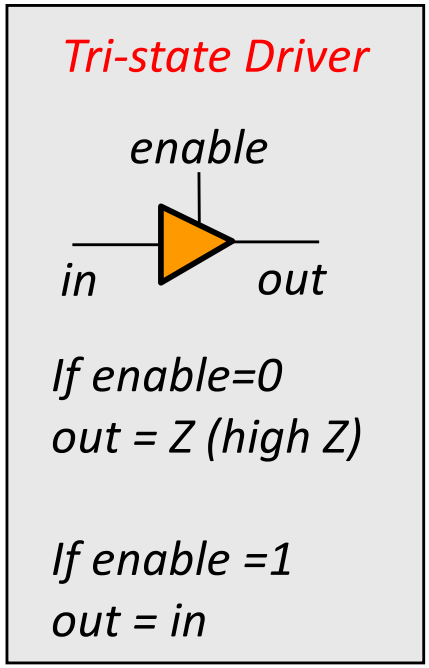
***With correct interfacing you can still think of this as a 16X1 array of switches!!! Even though it isn't***

# Memory Array Architecture



# Using External Memory Devices



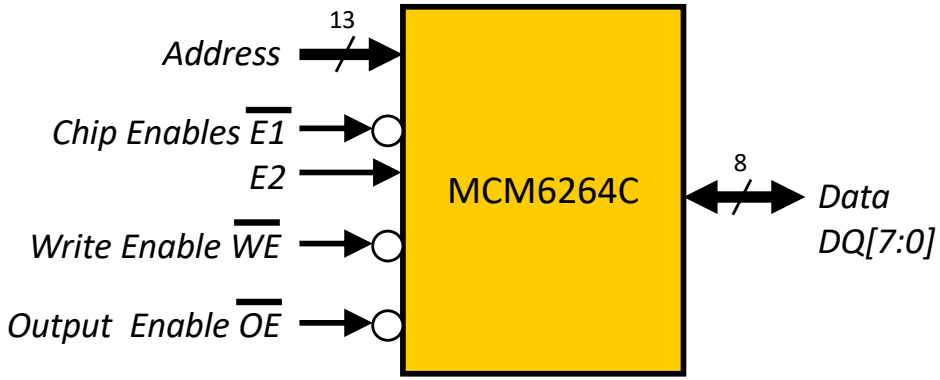

- Address pins drive row and column decoders
- Data pins are bidirectional: shared by reads and writes

Concept of "Data Bus"

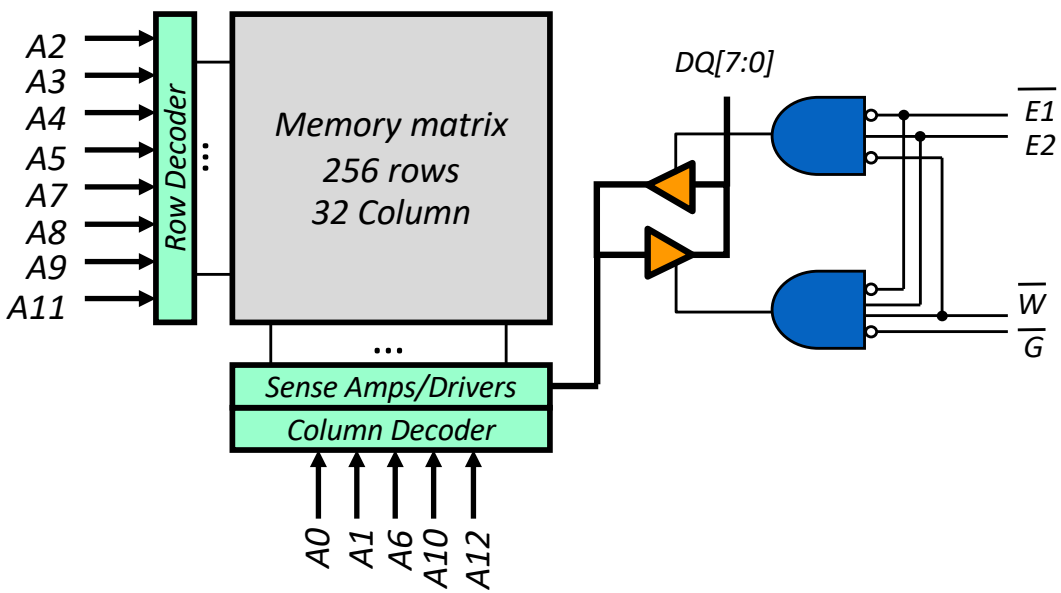
- Output Enable gates the chip's tristate driver
- Write Enable sets the memory's read/write mode
- Chip Enable/Chip Select acts as a "master switch"

# MCM6264C 8K x 8 Static RAM

*On the outside:*




*On the inside:*




Same (bidirectional) data bus used for reading and writing

Chip Enables ($\overline{E1}$ and E2)

- $\overline{E1}$ must be low and E2 must be high to enable the chip

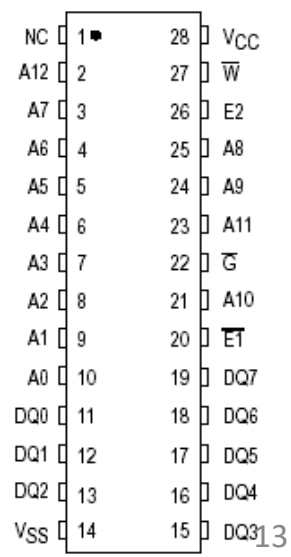
Write Enable ($\overline{WE}$)

- When low (and chip enabled), values on data bus are written to location selected by address bus

Output Enable ($\overline{OE}$ or $\overline{G}$)

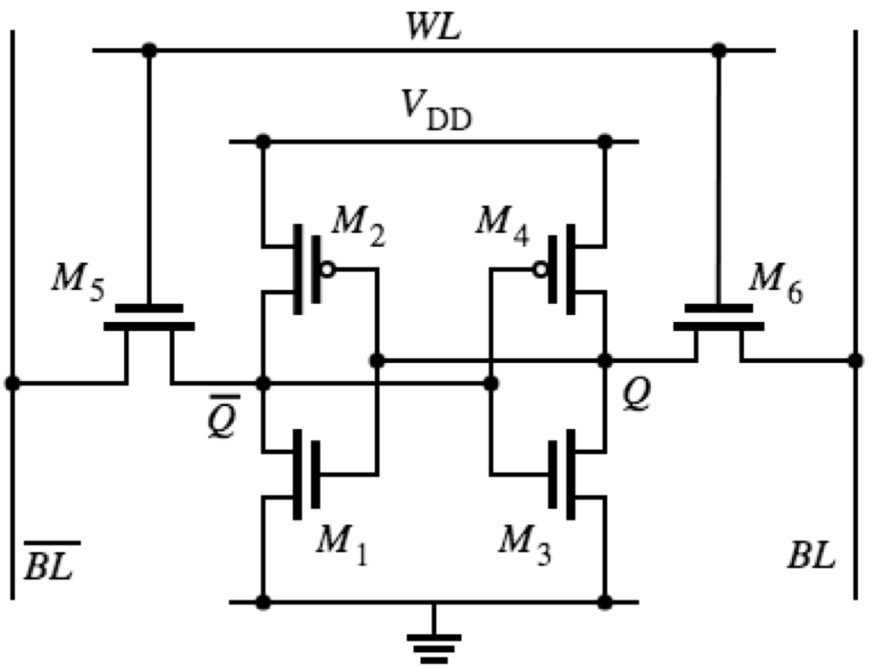
- When low (and chip is enabled), data bus is driven with value of selected memory location

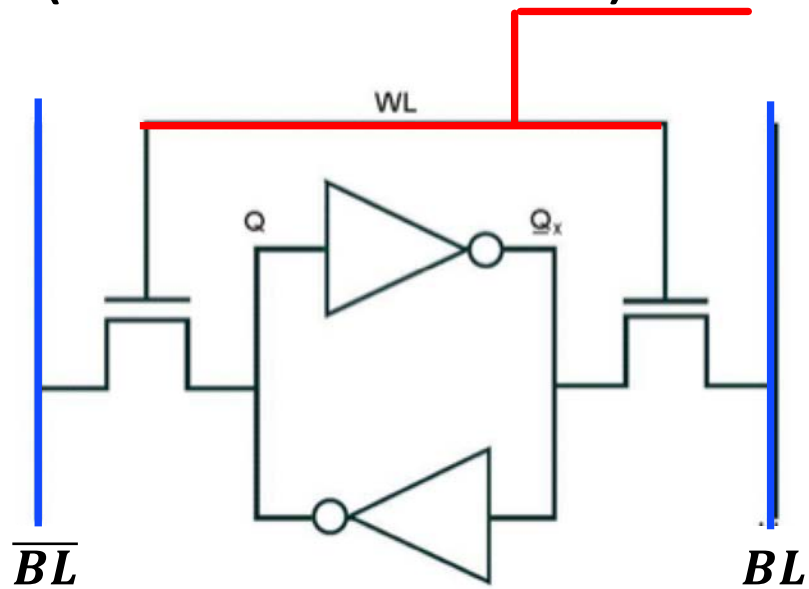
*Pinout*



# SRAM

# Static RAM (SRAM) Cell (The 6-T Cell)



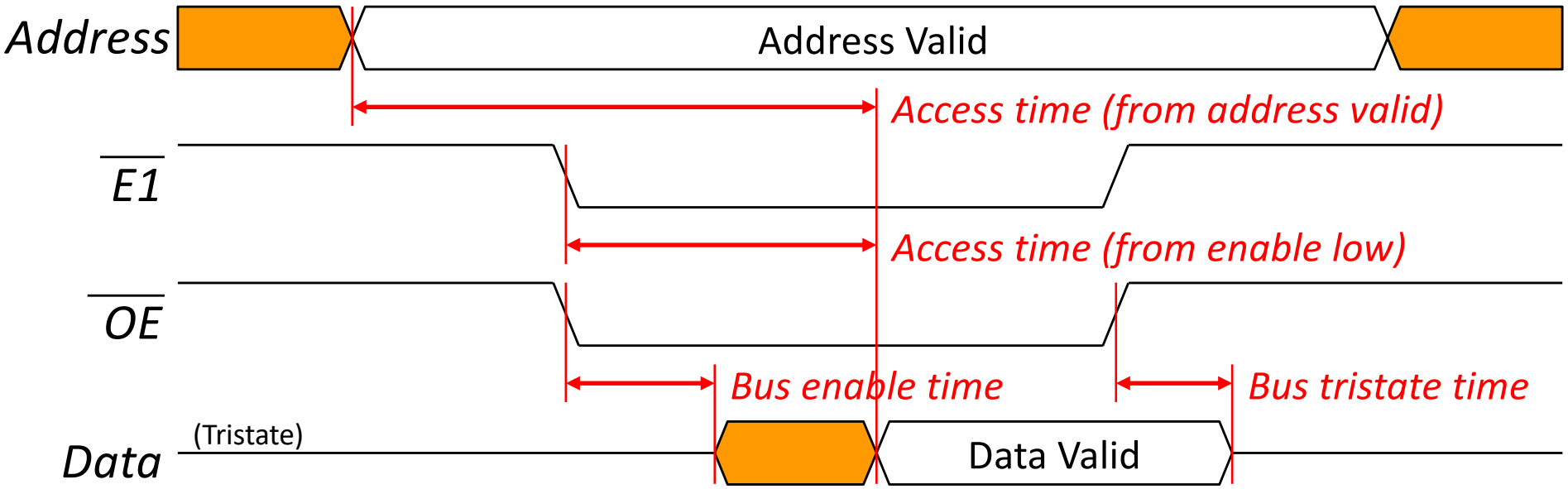




*Write:* Set BL, $\overline{BL}$ to (0,$V_{DD}$ ) or ($V_{DD}$,0) then enable WL (= $V_{DD}$)

*Read:* Disconnect drivers from BL and $\overline{BL}$, then enable WL (=$V_{DD}$). Sense a small change in BL or $\overline{BL}$

- State held by cross-coupled inverters (M1-M4)
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory

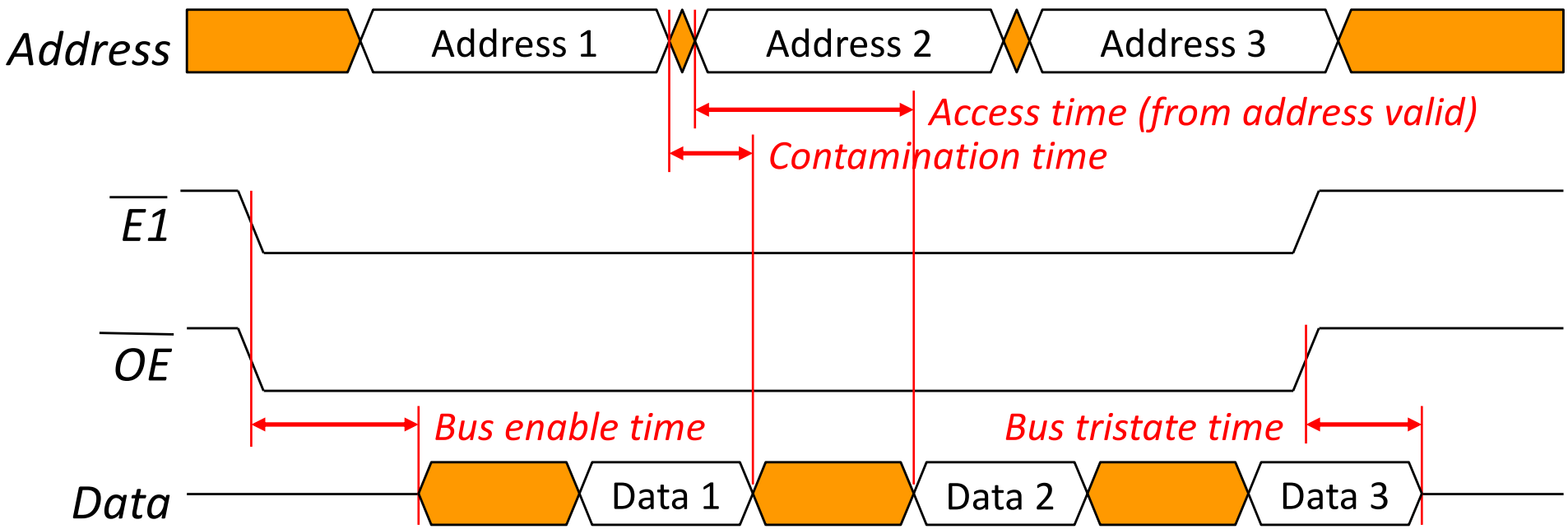
# Reading an Asynchronous SRAM




*E2 assumed high (enabled), $\overline{W}$ =1 (read mode)*

- Read cycle begins when all enable signals ($\overline{E1}$, E2, $\overline{OE}$) are active
- Data is valid after read access time
  - Access time is indicated by full part number: *MCM6264CP-12 → 12ns*
- Data bus is tristated shortly after $\overline{OE}$ or $\overline{E1}$ goes high

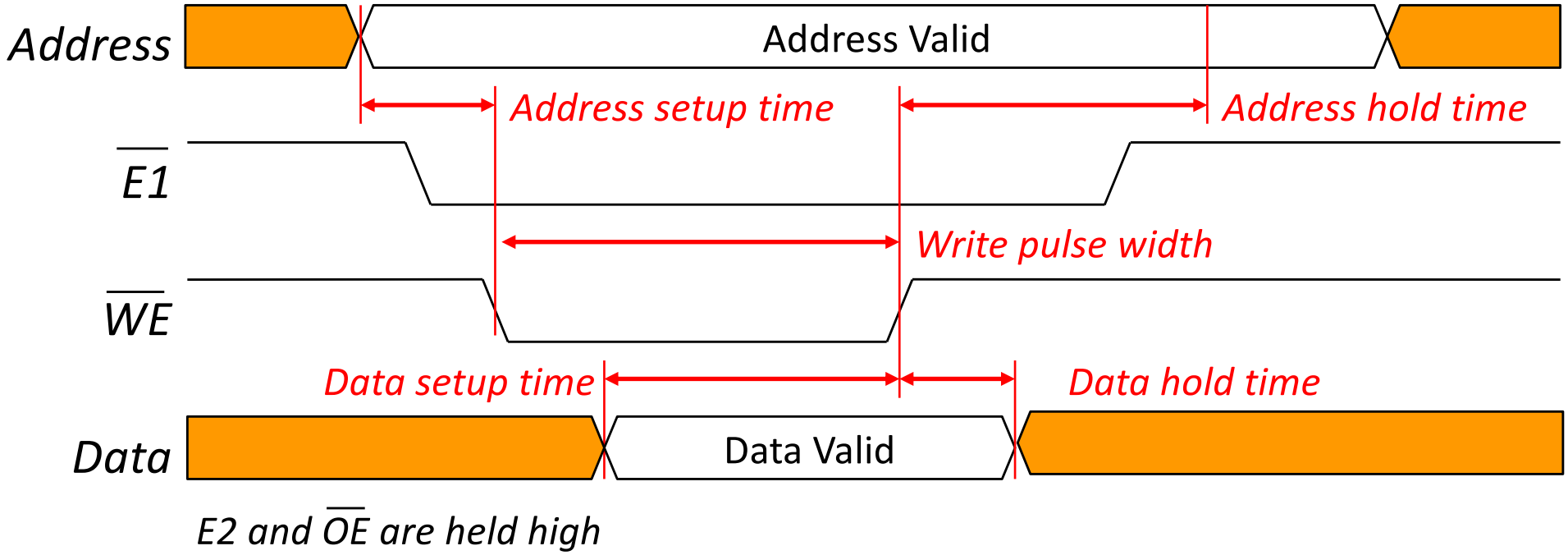
# Address Controlled Reads




*E2 assumed high (enabled), WE =$\overline{1}$ (read mode)*

- Can perform multiple reads without disabling chip
- Data bus follows address bus, after some delay

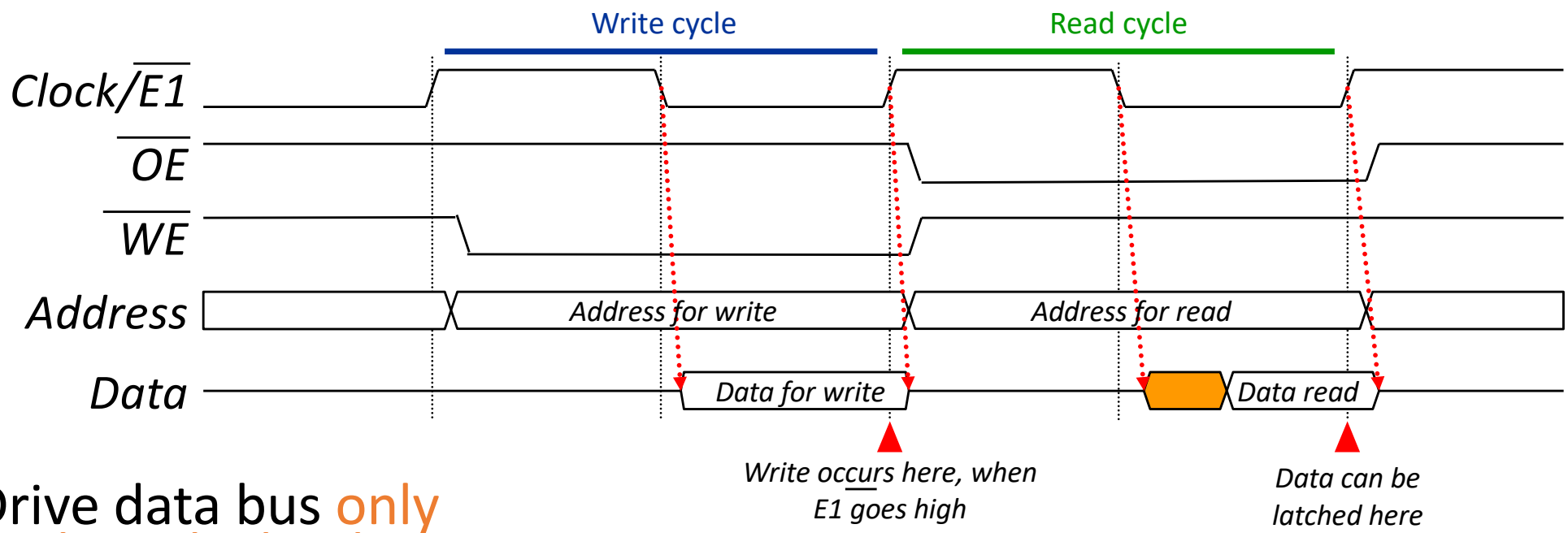
# Writing to Asynchronous SRAM




- Data latched when $\overline{WE}$ or $\overline{E1}$ goes high (or E2 goes low)
  - Data must be stable at this time
  - Address must be stable before $\overline{WE}$ goes low

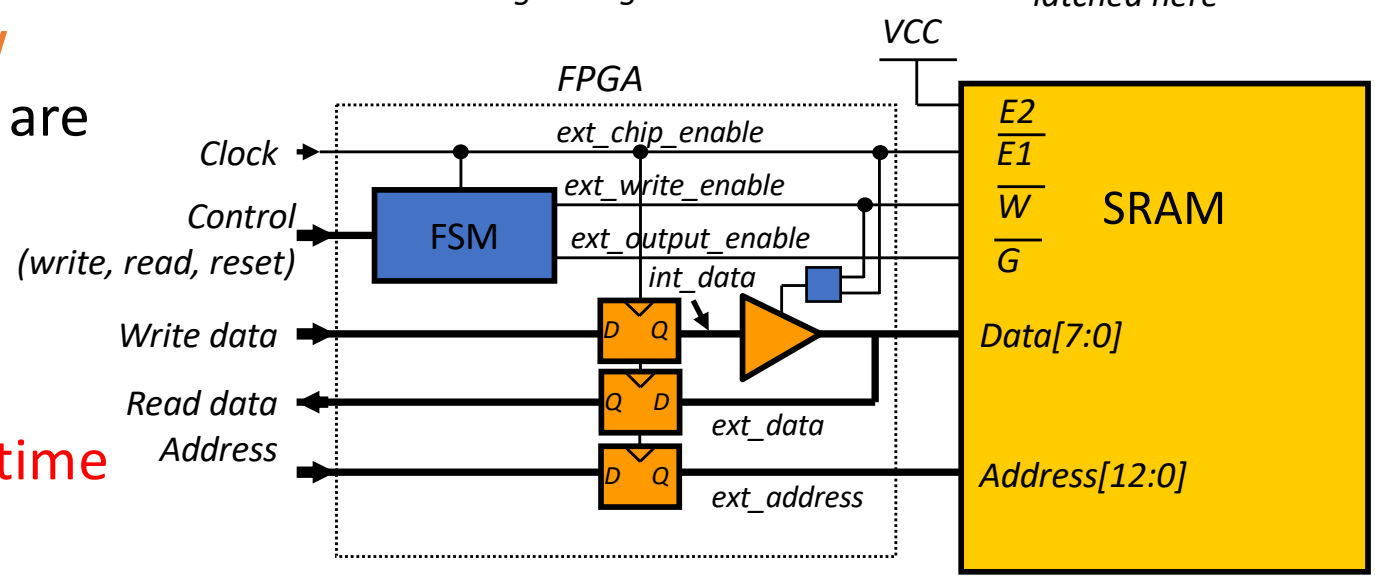
- Write waveforms are more important than read waveforms
  - Glitches to address can cause writes to random addresses!

# Sample Memory Interface Logic

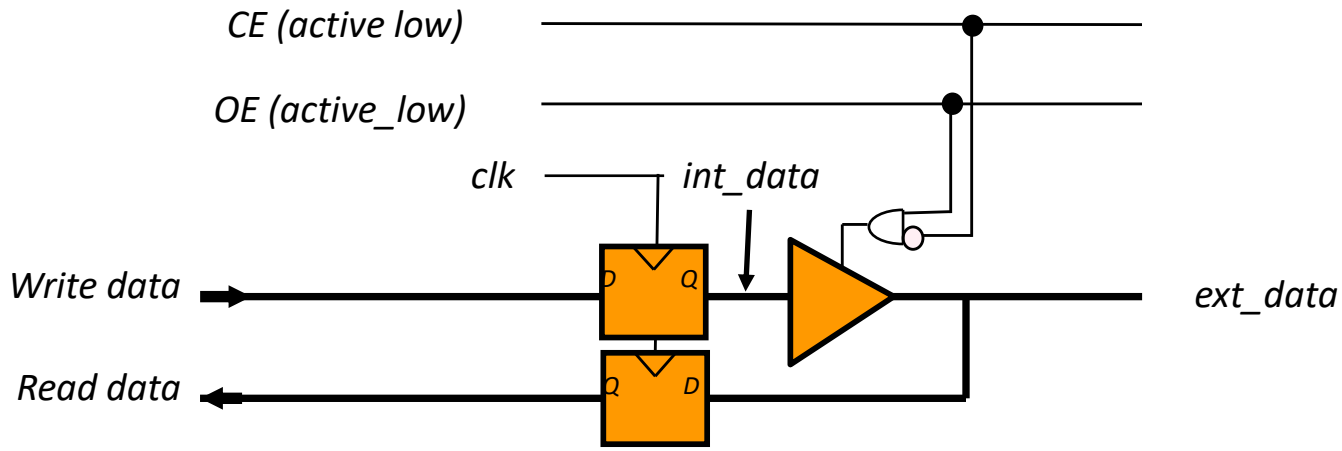



Drive data bus only when clock is low

- Ensures address are stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time



# Tristate Data Buses in Verilog

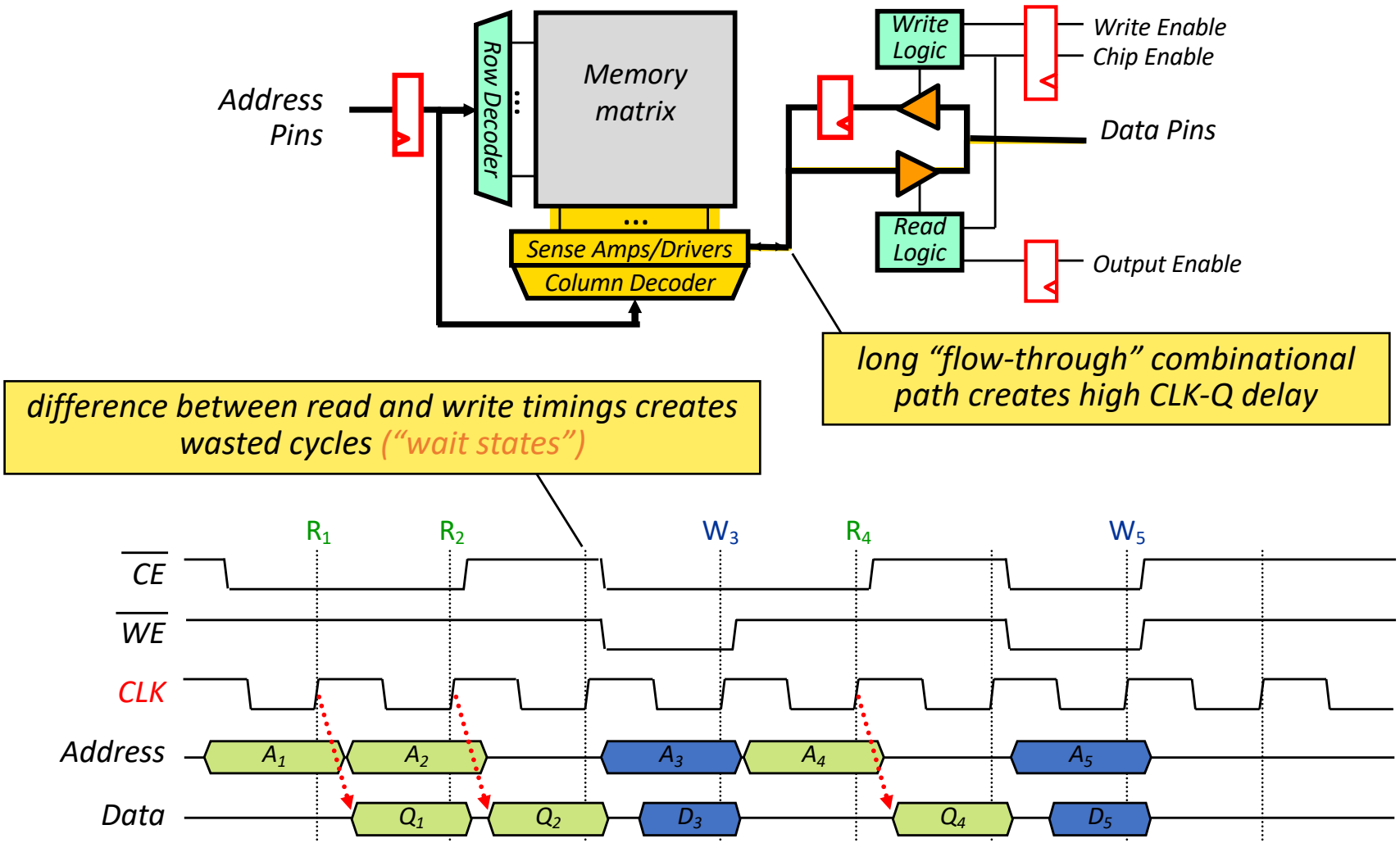



**The inout**

```
output CE,OE;  // these signals are active low
inout [7:0] ext_data;
reg [7:0] read_data,int_data
wire [7:0] write_data;

always @(posedge clk) begin
  int_data <= write_data;
  read_data <= ext_data;
end

// Use a tristate driver to set ext_data to a value
assign ext_data = (~CE & OE) ? int_data : 8'hZZ;
```

# Synchronous SRAM Memories

- Clocking provides input synchronization and encourages more reliable operation at high speeds

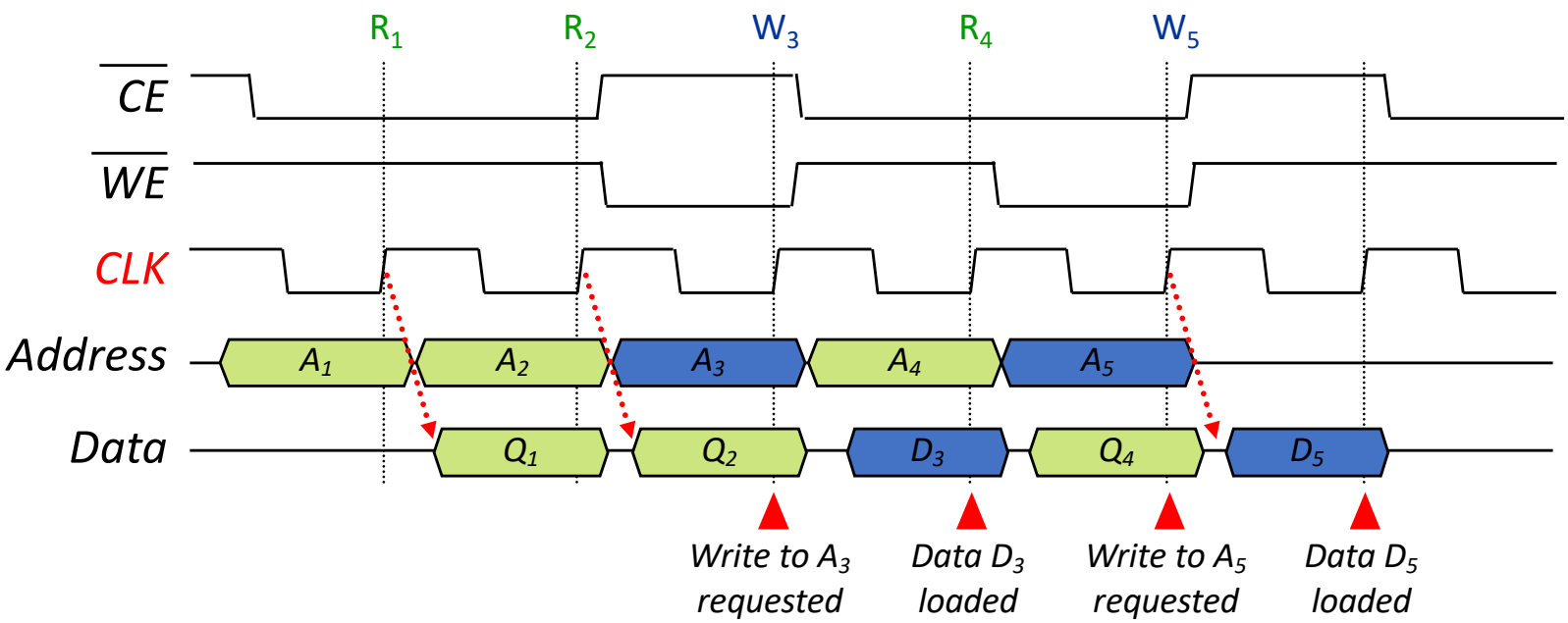


# ZBT Eliminates the Wait State

- The wait state occurs because:
  - On a read, data is available *after* the clock edge
  - On a write, data is set up *before* the clock edge

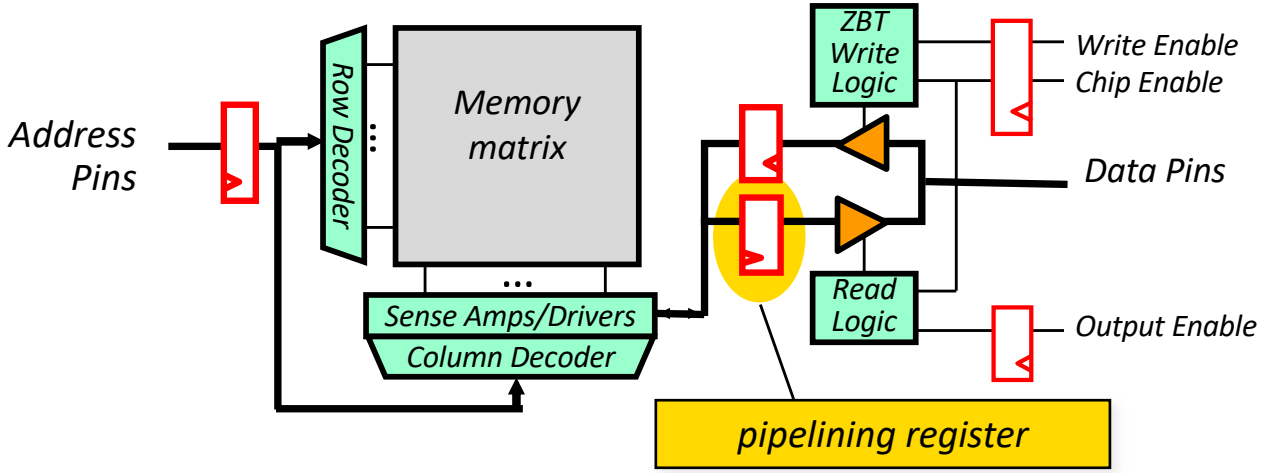
But we have to wait for bus to clear out read data ☹

- ZBT ("zero bus turnaround") memories change the rules for writes
  - On a write, data is set up after the clock edge (so that it is read on the following edge)
  - Result: no wait states, higher memory throughput

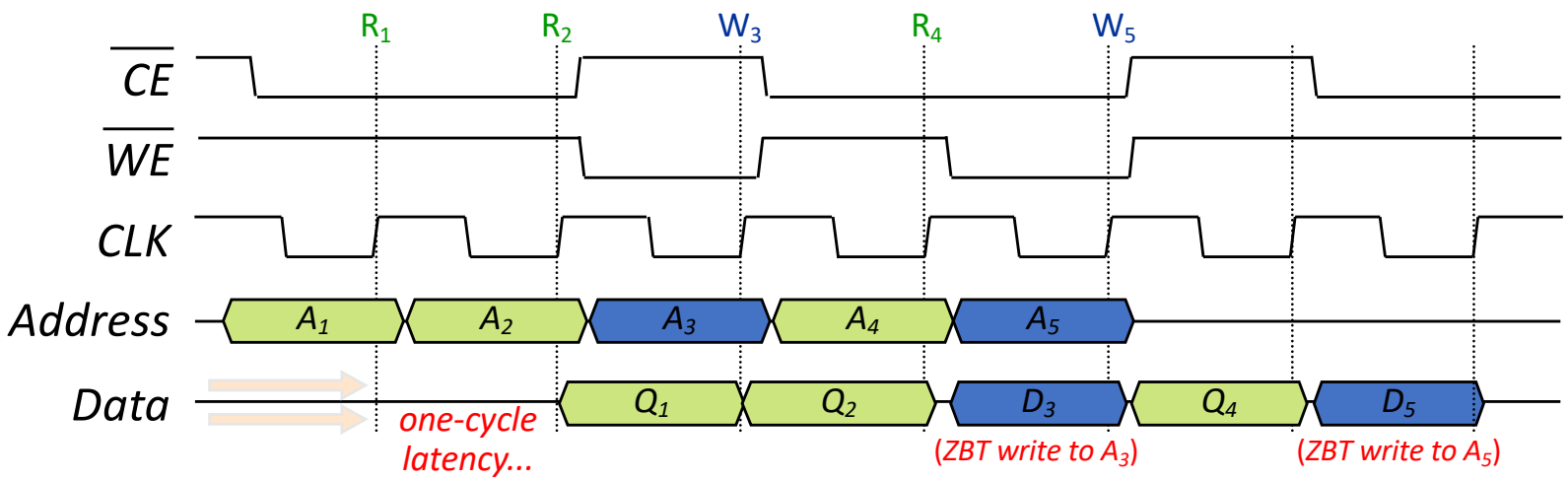


# Pipelining Allows Faster CLK

- Pipeline the memory by registering its output
  - Good: Greatly reduces CLK-Q delay, allows higher clock (**more throughput**)
  - Bad: Introduces an extra cycle before data is available (**more latency**)

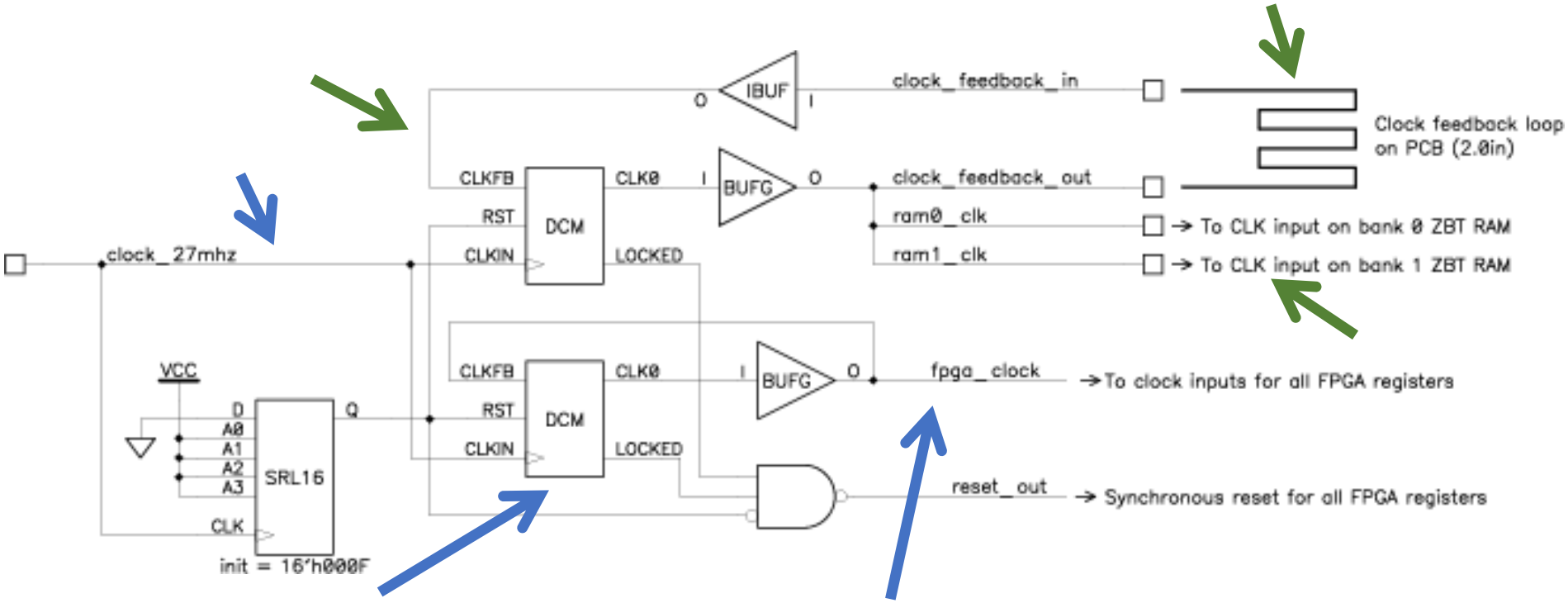



As an example, see the CY7C147X ZBT Synchronous SRAM



# Labkit ZBT interface

The upper DCM is used to generate the de-skewed clock for the external ZBT memories. The feedback loop for this DCM includes a 2.0 inch long trace on the labkit PCB and matches in distance all of the PCB traces from the FPGA to the ZBT memories. The propagation delay from the output of the upper DCM back to its CLKFB input should be almost exactly the same as the propagation delay from the DCM output to the ZBT memories.



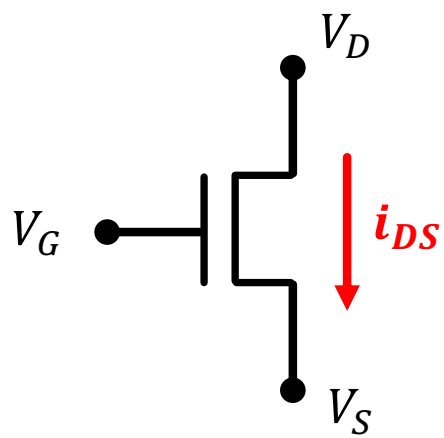

The lower DCM is used to ensure that the fpga_clock signal, which clocks all of the FPGA flip-flops, is in phase with the reference clock (clock_27mhz).

# EPROM Families

- Includes EPROM, EEPROM, Flash memory, (and SSDs)
- Utilize **<u>Floating Gates</u>**
- Different from SRAM!
- Instead of ~6 transistors per bit, you can do about 1!
- Acts just like SRAM from outside but Non-Volatile and writes are much slower than reads
- Invented by Dov Frohman while at Intel ~1970ish


*An early EPROM.*
*You'd program electrically and then shine UV onto it to erase it…don't use these anymore*

# Quick Review on MOSFETs




In sub-threshold mode:

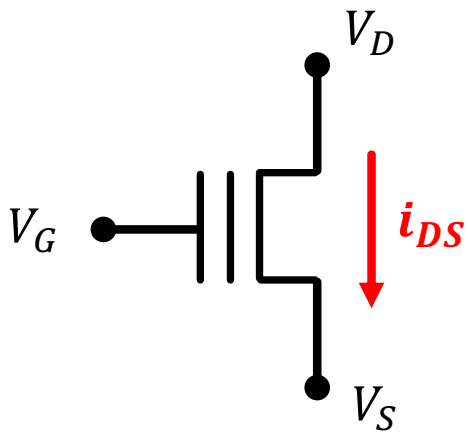
$$i_{DS} = K\left((V_{GS} - V_T)V_{DS} - \frac{V_{DS}^2}{2}\right)$$

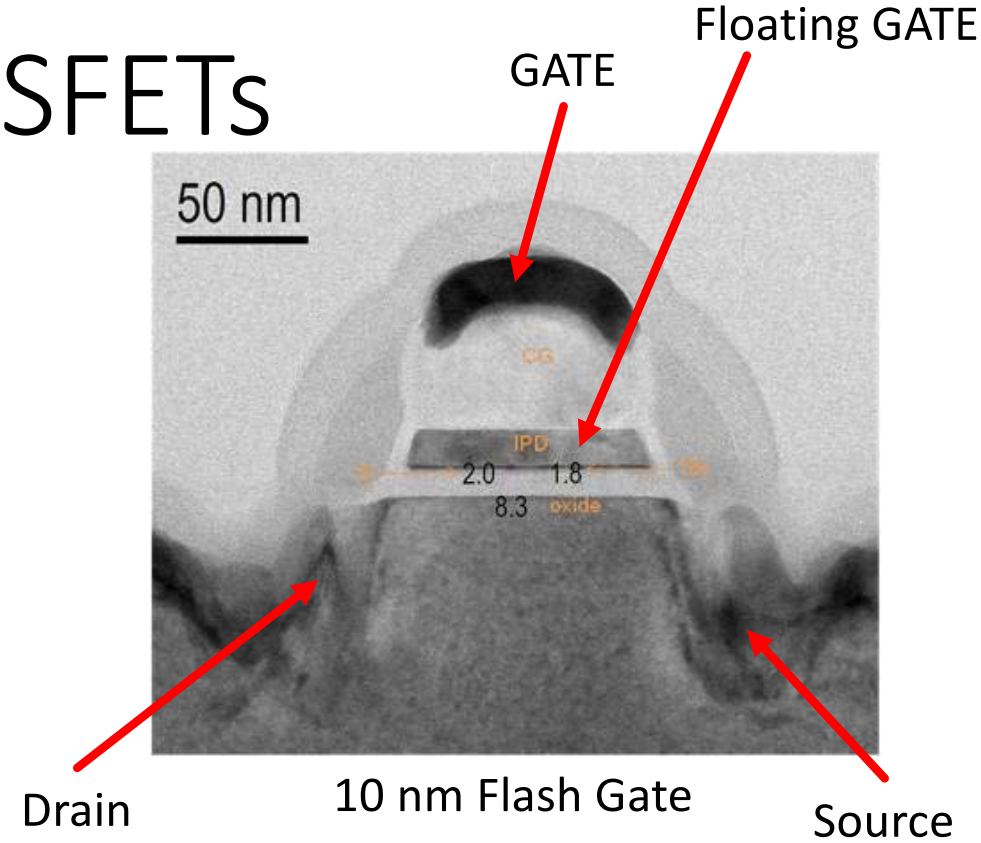
Above Threshold (saturation)

$$i_{DS} = K(V_{GS} - V_T)^2$$

- Basically:
  - If $V_G$ is > $V_T$ you conduct (are ”on”)
  - If $V_G$ is < $V_T$ you do not conduct (are ”off”)
- Traditionally $V_T$ is a function of doping, transistor dimensions, etc…
- BUT!....

# Floating Gate MOSFETs




Presence or absence of carriers on floating gate affects the threshold voltage of MOSFET

- Default ("binary 1")…Threshold voltage is lower $V_{TL}$
- Programmed bit ("binary 0")…threshold voltage is higher $V_{TH}$

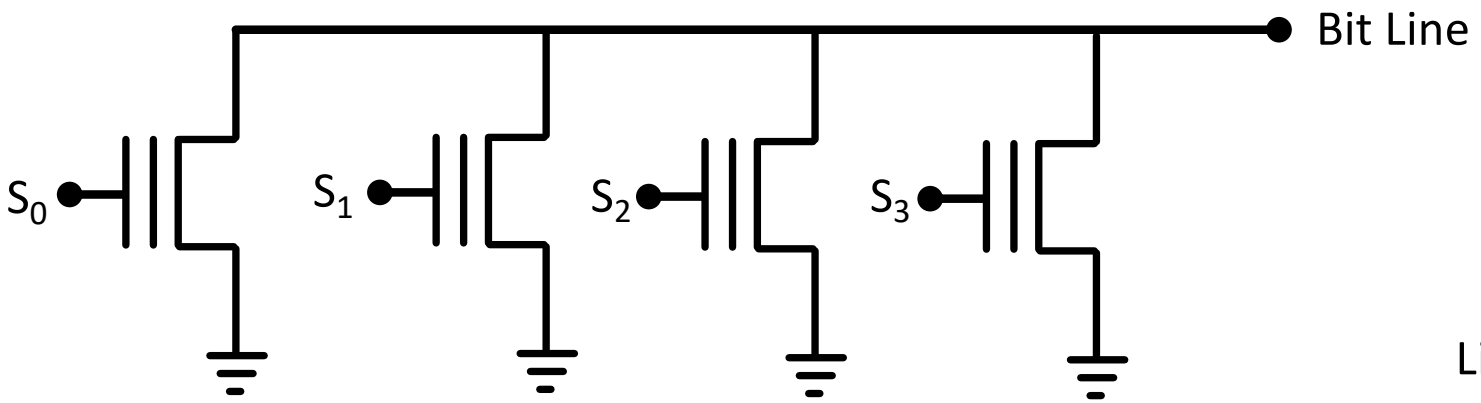
*https://www.electronicsweekly.com/news/research-news/device-rd/iedm-hybrid-floating-gate-scales-flash-to-10nm-2012-12/*

# Hot Carrier Injection/Tunneling to Program/Reprogram

- To add or remove electrons to the floating gate you use a quantum tunneling phenomenon
- High voltage (~12V over 100's of Angstroms) is used to force electrons to tunnel into floating gate… the term "hot" refers to high energies on electrons.
- A similar process is used in reverse to tunnel them out again
- This is a potentially destructive process and will eventually destroy the device. Flash therefore has limits of ~ several 100,000's of program/erase cycles
- Mitigate issues by wear-leveling (try to spread out usage across all of device…like rotating tires on a car)

# NOR Flash

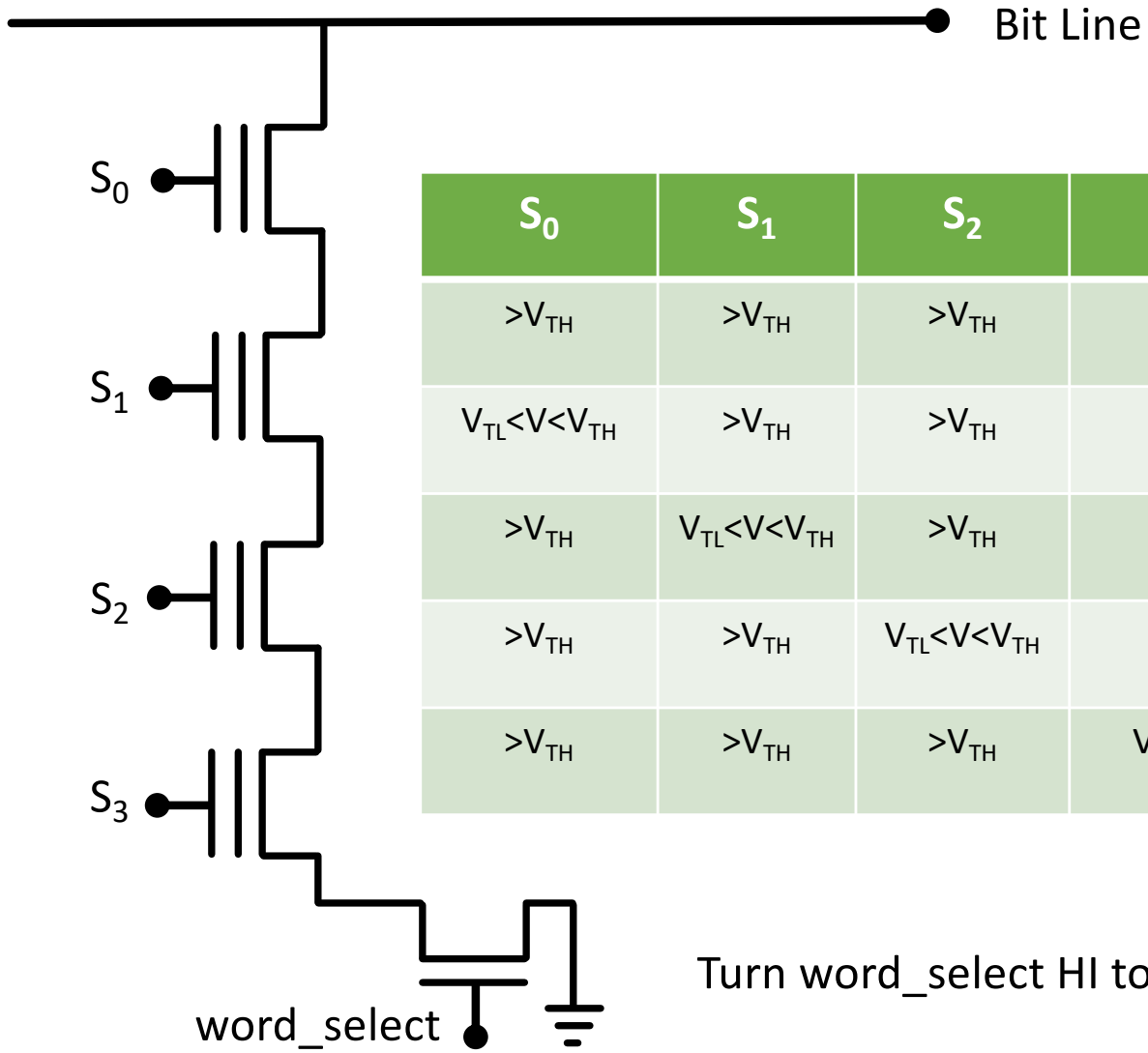
1 on the select bits means voltage greater than $V_{TL}$ but less than $V_{TH}$ (high and low thresholds)




| $S_0$ | $S_1$ | $S_2$ | $S_3$ | Out |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 if $B_0$==1, 1 if $B_0$==0 |
| 0 | 1 | 0 | 0 | 0 if $B_1$==1, 1 if $B_1$==0 |
| 0 | 0 | 1 | 0 | 0 if $B_2$==1, 1 if $B_2$==0 |
| 0 | 0 | 0 | 1 | 0 if $B_3$==1, 1 if $B_3$==0 |

Like a **NOR** gate since when all bits are 1, bit line goes low if any input is turned high…hence called **NOR Flash**

# NAND Flash




Like a **NAND** gate since when all bits are 1, bit line goes low if tested input is also 1…hence called **NAND Flash**

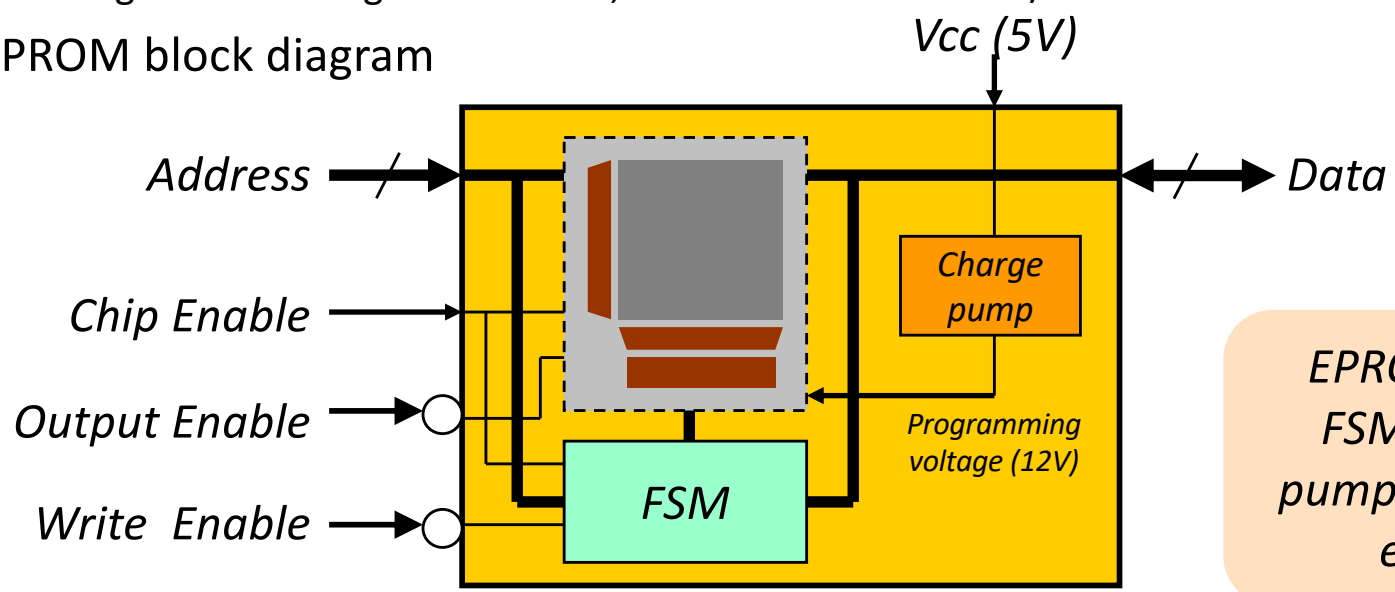
| $S_0$ | $S_1$ | $S_2$ | $S_3$ | Out |
|---|---|---|---|---|
| $>V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | 0 |
| $V_{TL}<V<V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | 0 if $B_0$==1, 1 if $B_0$==0 |
| $>V_{TH}$ | $V_{TL}<V<V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | 0 if $B_1$==1, 1 if $B_1$==0 |
| $>V_{TH}$ | $>V_{TH}$ | $V_{TL}<V<V_{TH}$ | $>V_{TH}$ | 0 if $B_2$==1, 1 if $B_2$==0 |
| $>V_{TH}$ | $>V_{TH}$ | $>V_{TH}$ | $V_{TL}<V<V_{TH}$ | 0 if $B_3$==1, 1 if $B_3$==0 |

Turn word_select HI to enable whole word

# Interacting with Flash and (E)EPROM

- Reading from flash or (E)EPROM is the same as reading from SRAM
- Vpp: input for programming voltage (12V)
  - EPROM: Vpp is supplied by programming machine
  - Modern flash/EEPROM devices generate 12V using an on-chip charge pump
- EPROM lacks a write enable (unless you work for missile defense, you'll not interact with EPROM)
  - Not in-system programmable (must use a special programming machine)
- For **flash** and **EEPROM**, write sequence is controlled by an internal FSM
  - Writes to device are used to send signals to the FSM
  - Although the same signals are used, one can't write to flash/EEPROM in the same manner as SRAM

Flash/EEPROM block diagram




*EPROM omits FSM, charge pump, and write enable*

# Flash Memory

- Flash memory uses NOR or NAND flash.
  - NAND cells connected in series like resembling NAND gate.
  - NAND requires 60% of the area compared to NOR. NAND used in flash drives.
  - Endurance: 100,000 – 300,000 p/e cycles
  - Life cycle extended through wear –leveling: mapping of physical blocks changes over time.

*Flash is slow, cache to RAM for fast read speed*

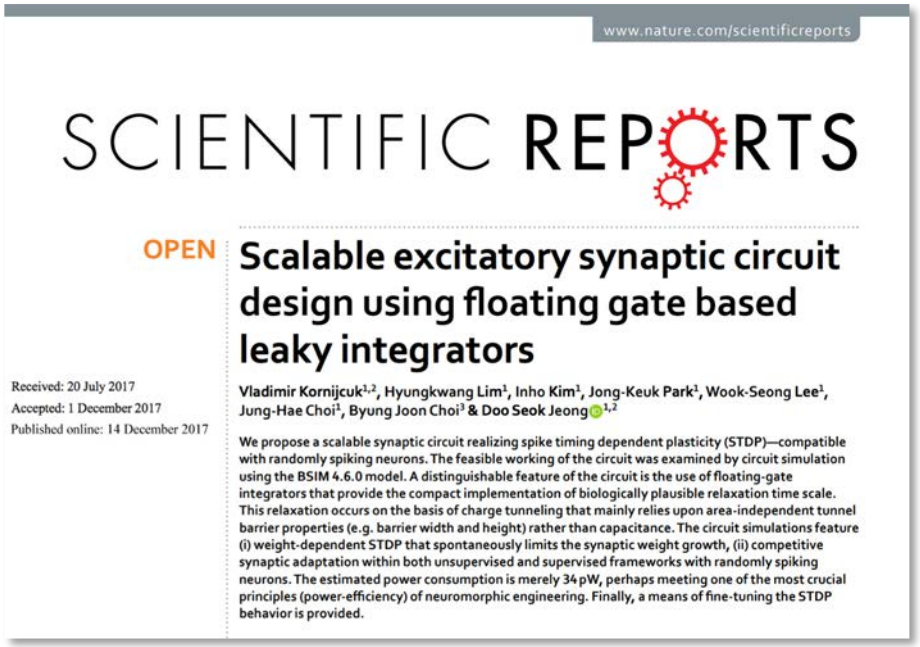
- Flash memory limitations
  - Can be read or written byte a time
  - Can only be erased block at a time
  - Erasure sets bits to 1.
  - Location can be re-written if the new bit is zero.

- Labkit has 128Mbits of memory in 1Mbit blocks.
  - 3 Volt Intel StrataFlash® Memory (28F128J3A)
  - 100,000 min erase cycle per block
  - Block erasures takes one second
  - 15 minutes to write entire flash ROM

http://www.embeddedintel.com/special_features.php?article=124

# Floating Gates

- Some neat recent work using floating gates and their adjustable threshold capabilities
- Result is ability to adjust/teach a single transistor when to fire based on input signals!



www.nature.com/scientificreports

SCIENTIFIC REPORTS

OPEN

**Scalable excitatory synaptic circuit design using floating gate based leaky integrators**

Received: 20 July 2017
Accepted: 1 December 2017
Published online: 14 December 2017

Vladimir Kornijcuk[1,2], Hyungkwang Lim[1], Inho Kim[1], Jong-Keuk Park[1], Wook-Seong Lee[1], Jung-Hae Choi[1], Byung Joon Choi[3] & Doo Seok Jeong[1,2]

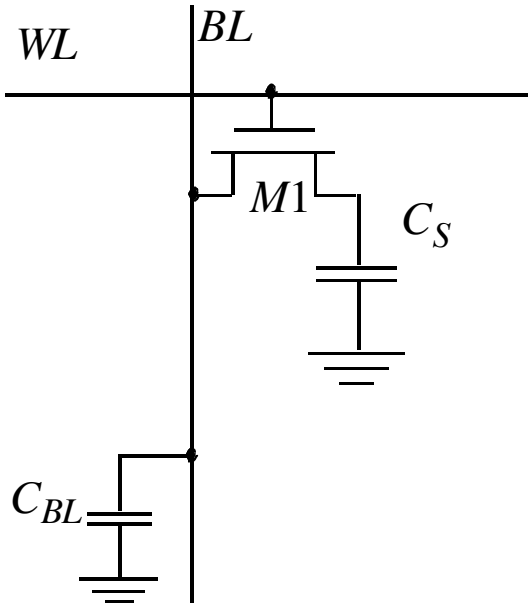
We propose a scalable synaptic circuit realizing spike timing dependent plasticity (STDP)—compatible with randomly spiking neurons. The feasible working of the circuit was examined by circuit simulation using the BSIM 4.6.0 model. A distinguishable feature of the circuit is the use of floating-gate integrators that provide the compact implementation of biologically plausible relaxation time scale. This relaxation occurs on the basis of charge tunneling that mainly relies upon area-independent tunnel barrier properties (e.g. barrier width and height) rather than capacitance. The circuit simulations feature (i) weight-dependent STDP that spontaneously limits the synaptic weight growth, (ii) competitive synaptic adaptation within both unsupervised and supervised frameworks with randomly spiking neurons. The estimated power consumption is merely 34 pW, perhaps meeting one of the most crucial principles (power-efficiency) of neuromorphic engineering. Finally, a means of fine-tuning the STDP behavior is provided.

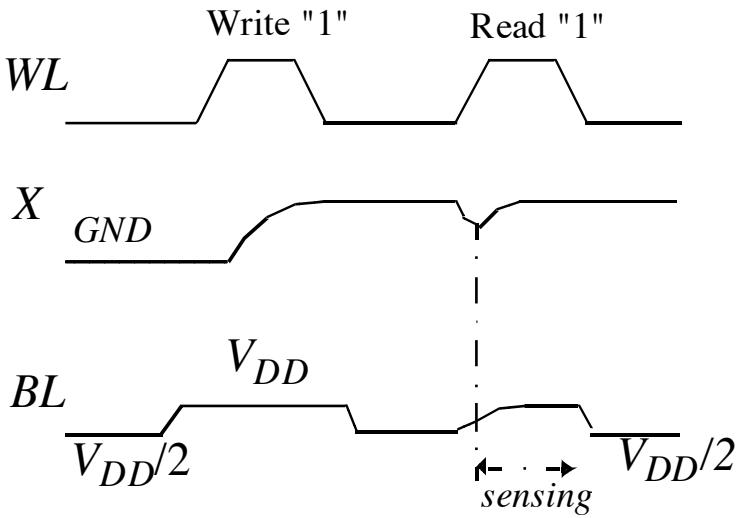
*Floating Gate neural-like implementations, 2017*

# DRAM

- Dynamic Random Access Memory!
- Single transistor and capacitor per bit (capacitor does the storage)
- Capacitors decay rather quickly (especially since DRAM capacitors are about 10 femtoFarads)
- Can be made extremely dense and therefore economical
- Are fast-ish:
  - SRAM will have access time of down to 10ns
  - **DRAM** will have access time from 50-150ns
  - EEPROM/Flash way slower (esp for writes)

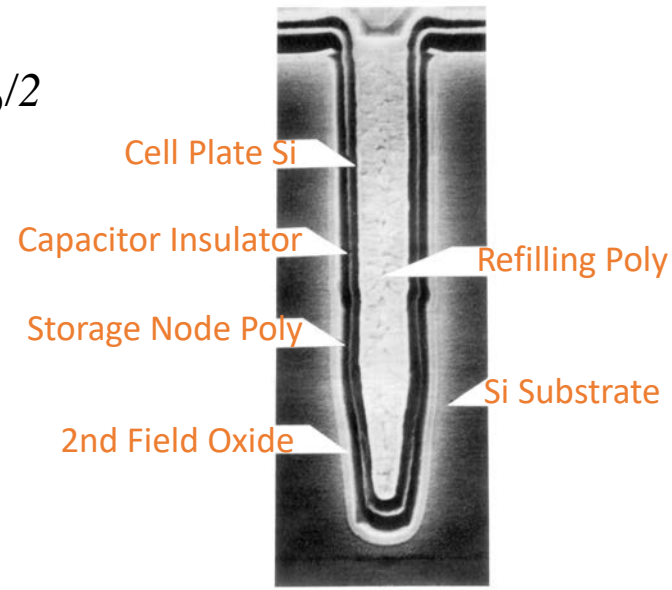
# Dynamic RAM (DRAM) Cell




[Rabaey03]

DRAM uses Special Capacitor Structures

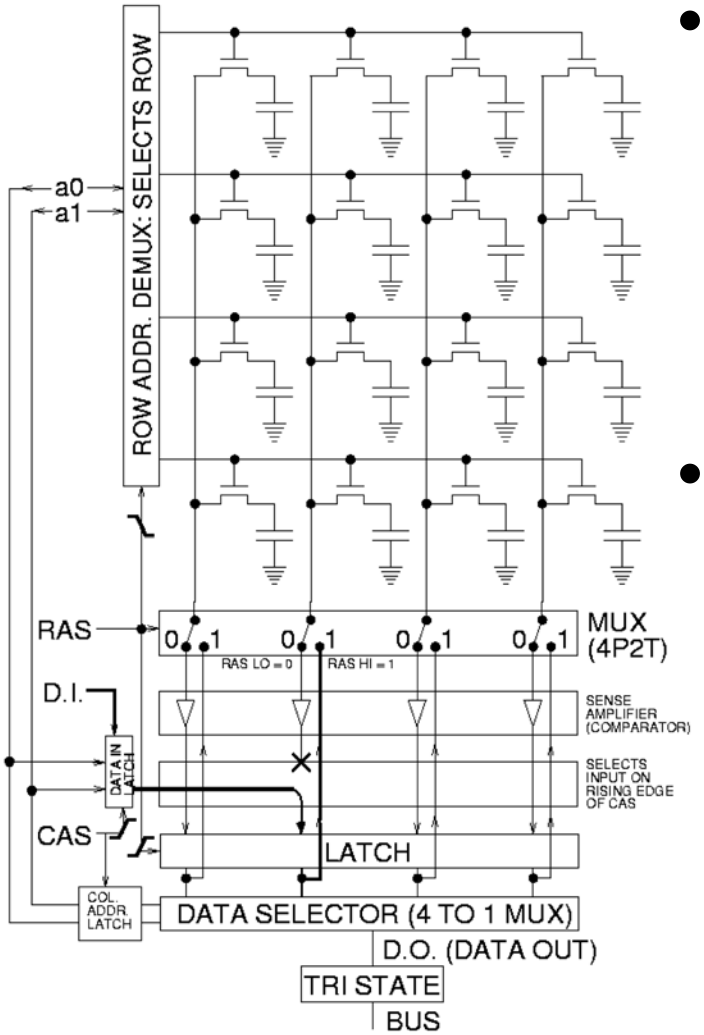



To Write: set Bit Line (BL) to 0 or $V_{DD}$
& enable Word Line (WL) (i.e., set to $V_{DD}$ )

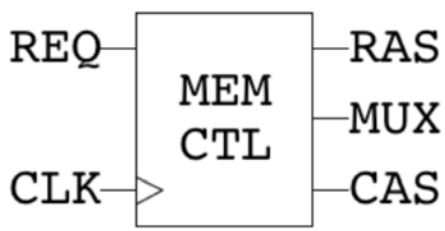
To Read: set Bit Line (BL) to $V_{DD}$ /2
& enable Word Line (i.e., set it to $V_{DD}$ )

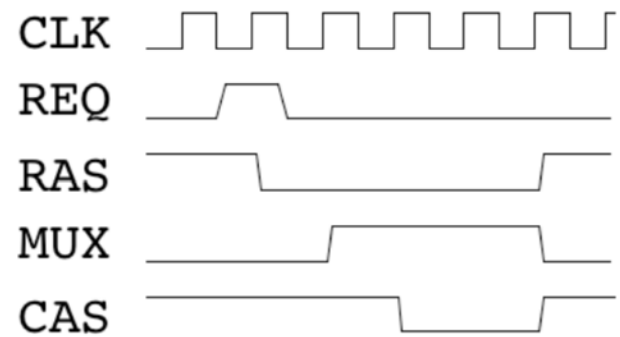
- DRAM relies on charge stored in a capacitor to hold state
- Found in all high density memories (one bit/transistor)
- Must be "refreshed" or state will be lost – high overhead

# DRAM Memory and Controller

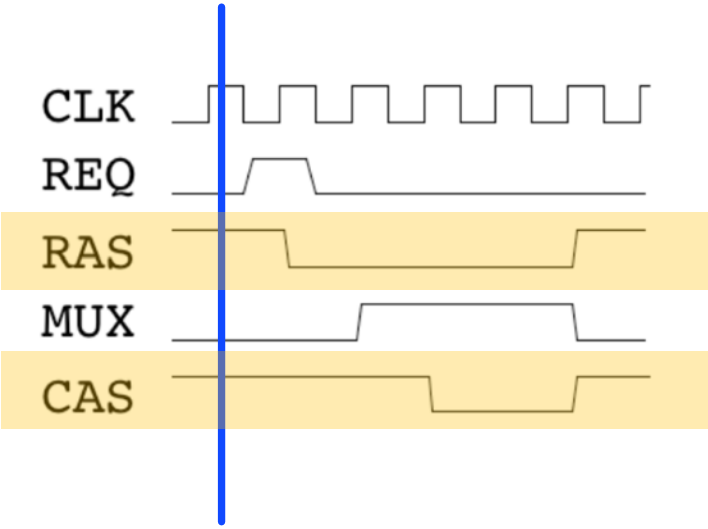



- Reading is destructive!
  - Data stored on small capacitor
  - To read it we must bleed the capacitor off
  - Therefore need to refresh
- Need to refresh even when not reading (every 100 ms)

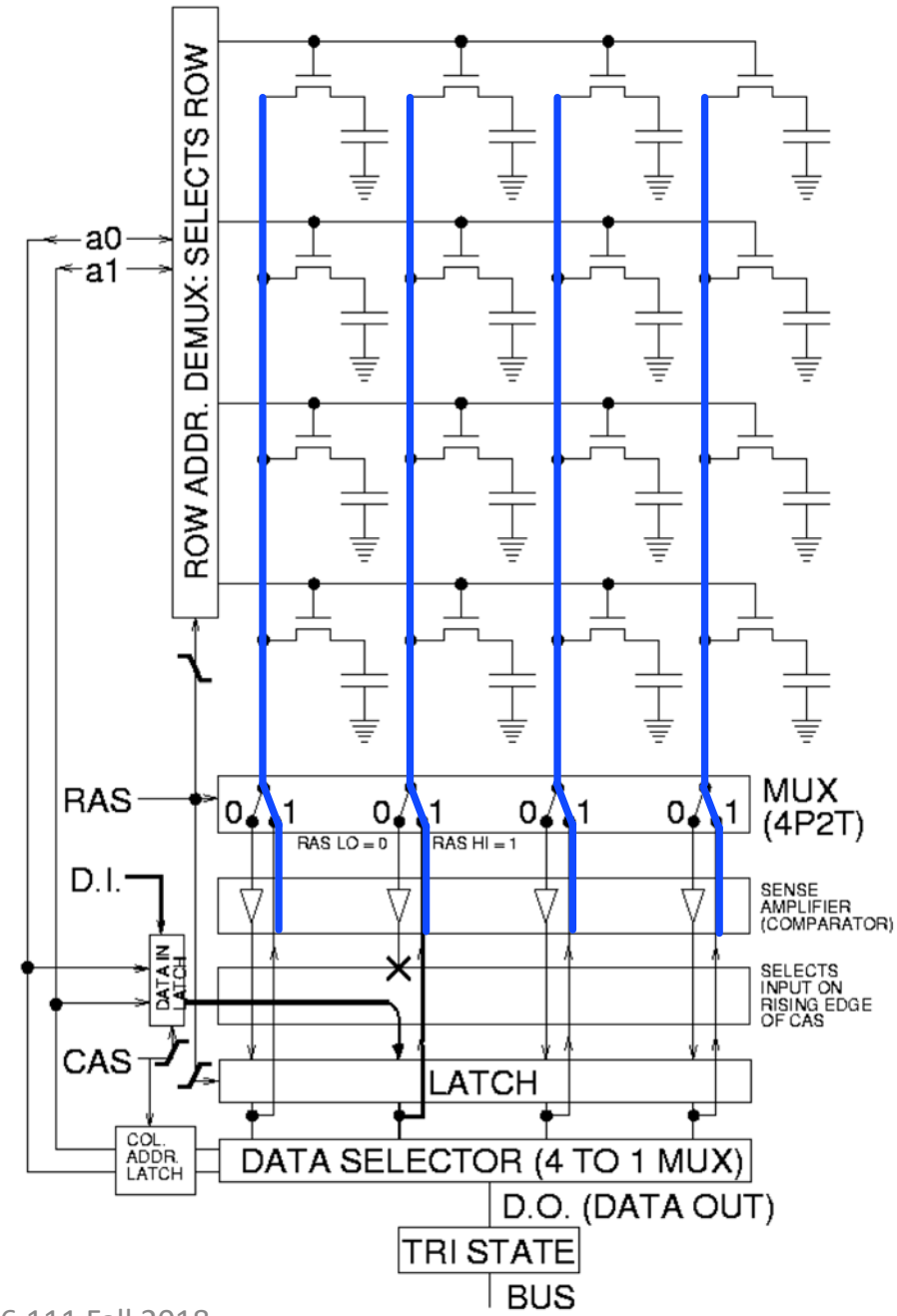


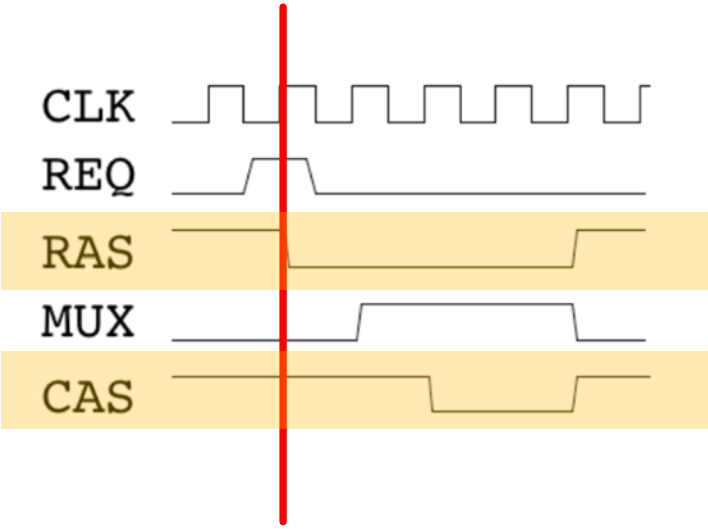
# *Asynchronous DRAM*



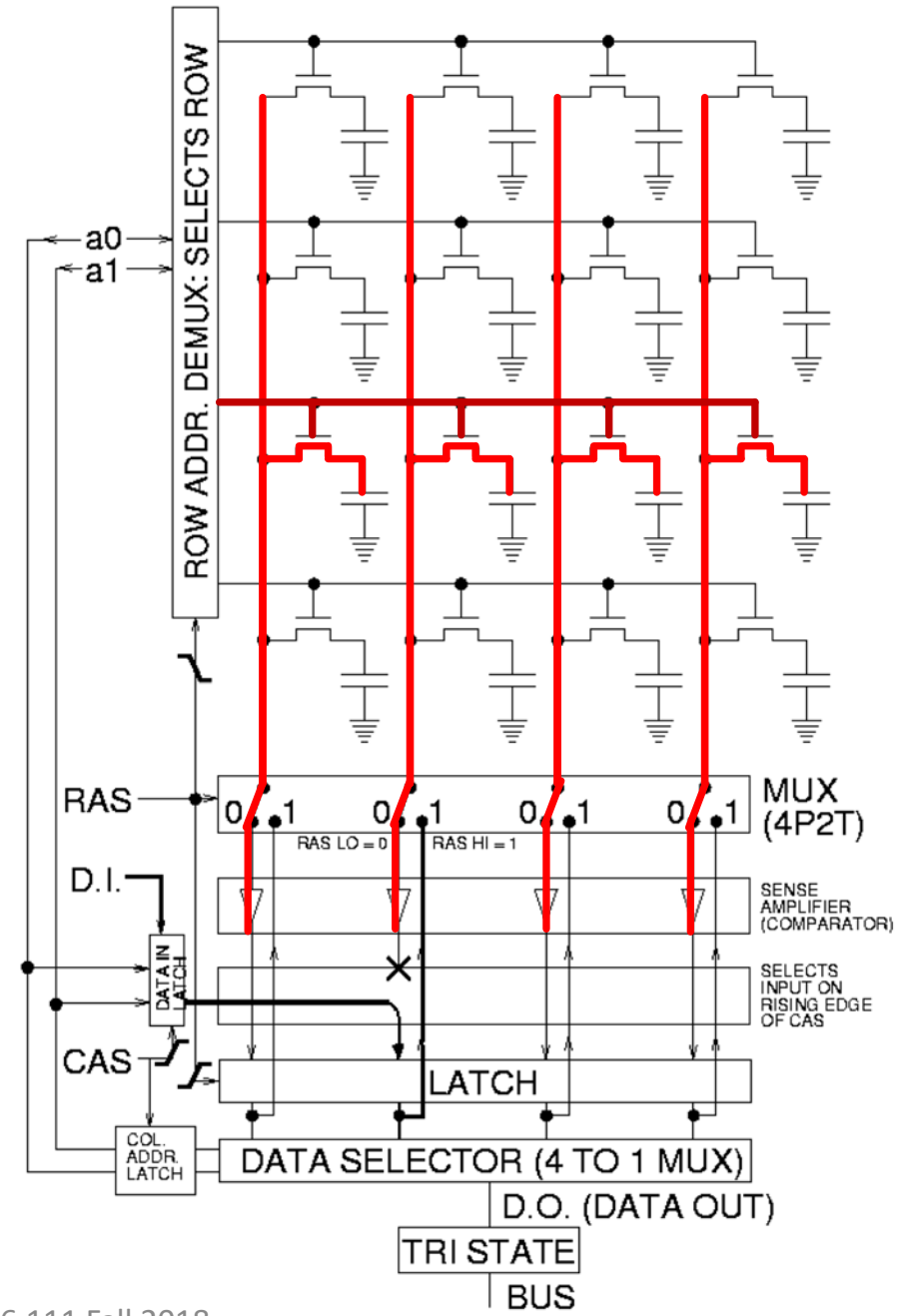

1. Column lines start charged to mid-voltageCharge up column lines to “mid-voltage” (REQ=1, RAS=1)



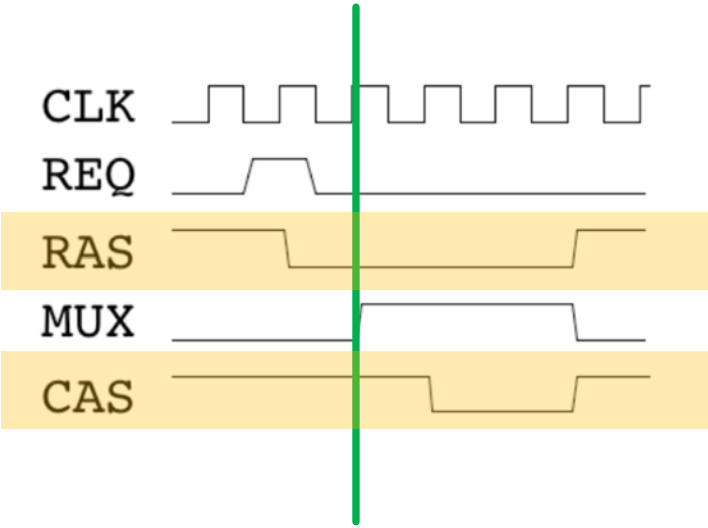
# *Asynchronous DRAM*



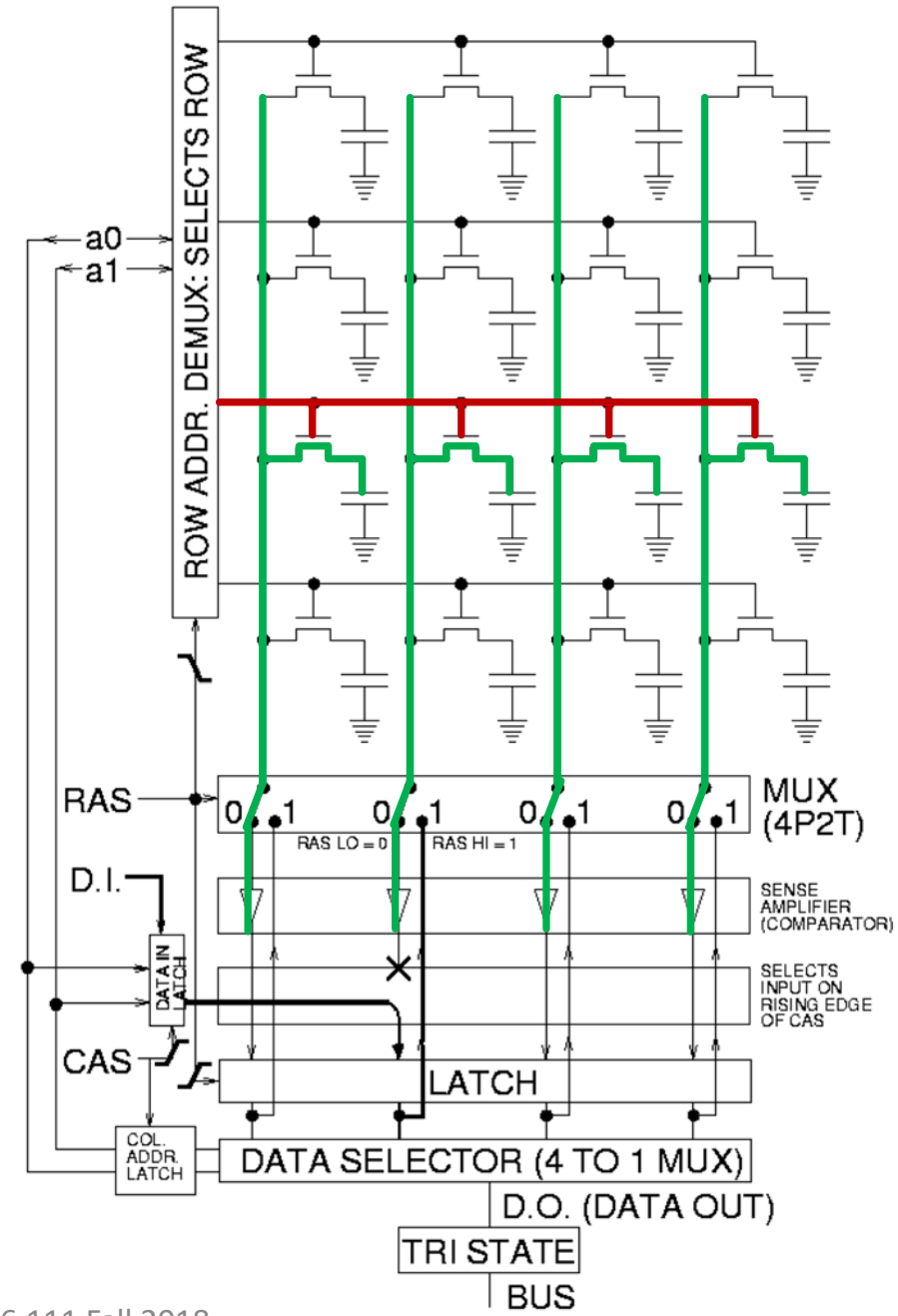

1. Connect to sense amplifiers
2. Select Row
3. Discharge caps onto column lines
4. Sense voltage dips or rises for 1's and 0's
5. Use positive feedback to charge bit lines up to true 1 or 0
6. Cache remembered 1's and 0's



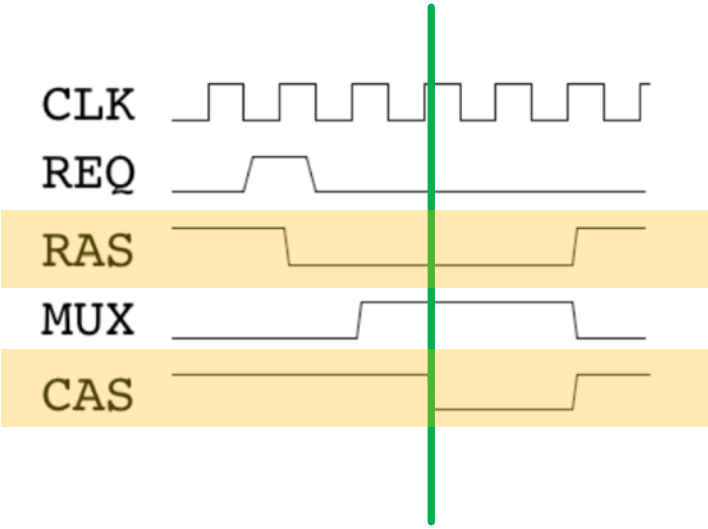
# *Asynchronous DRAM*



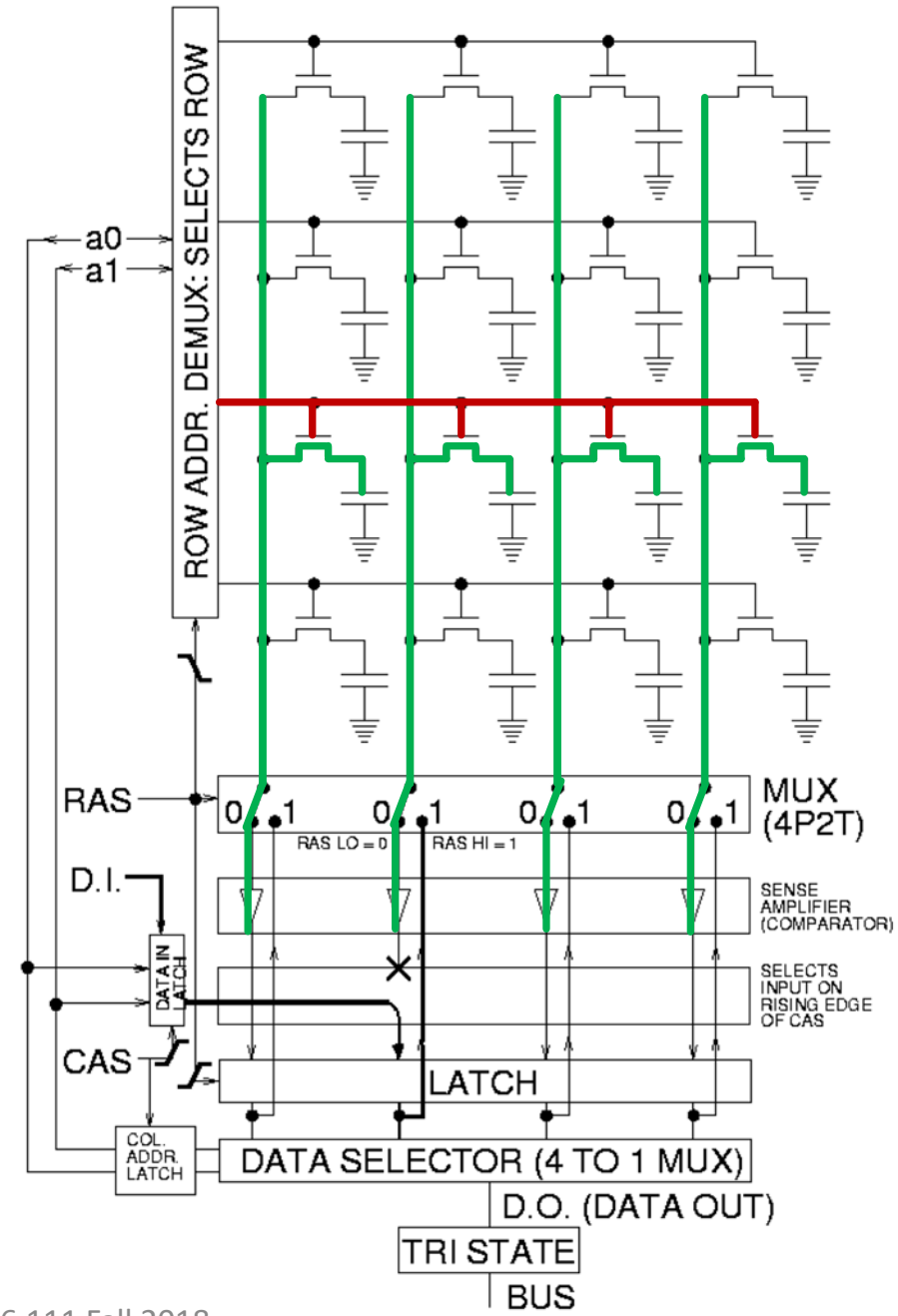

1. Read/Write by adjusting some input/output selector



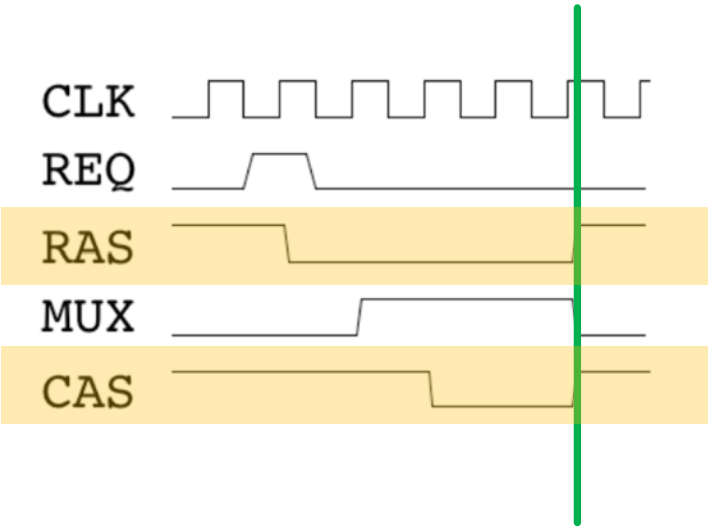
# *Asynchronous DRAM*



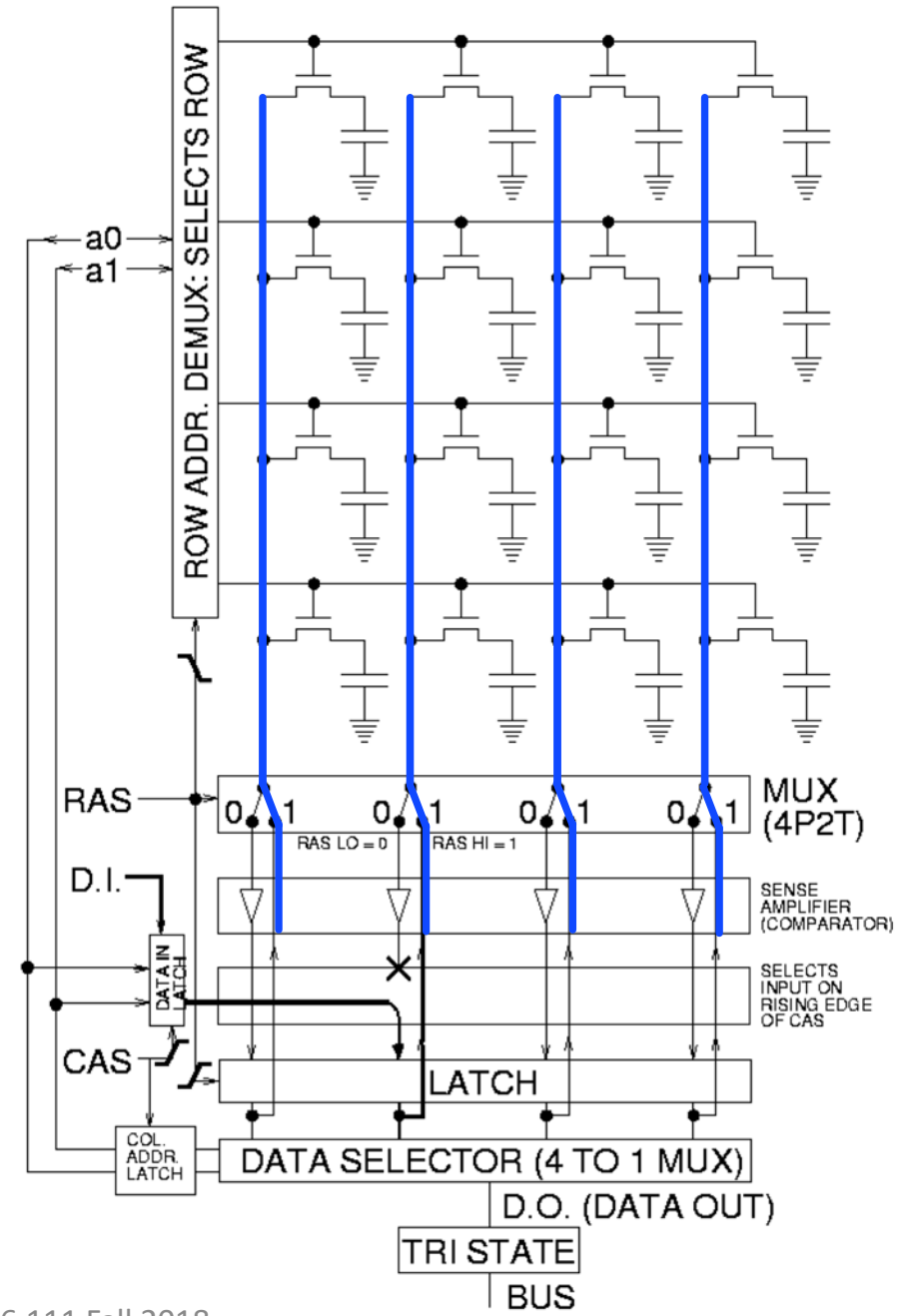

1. Select which Column to route out



# *Asynchronous DRAM*

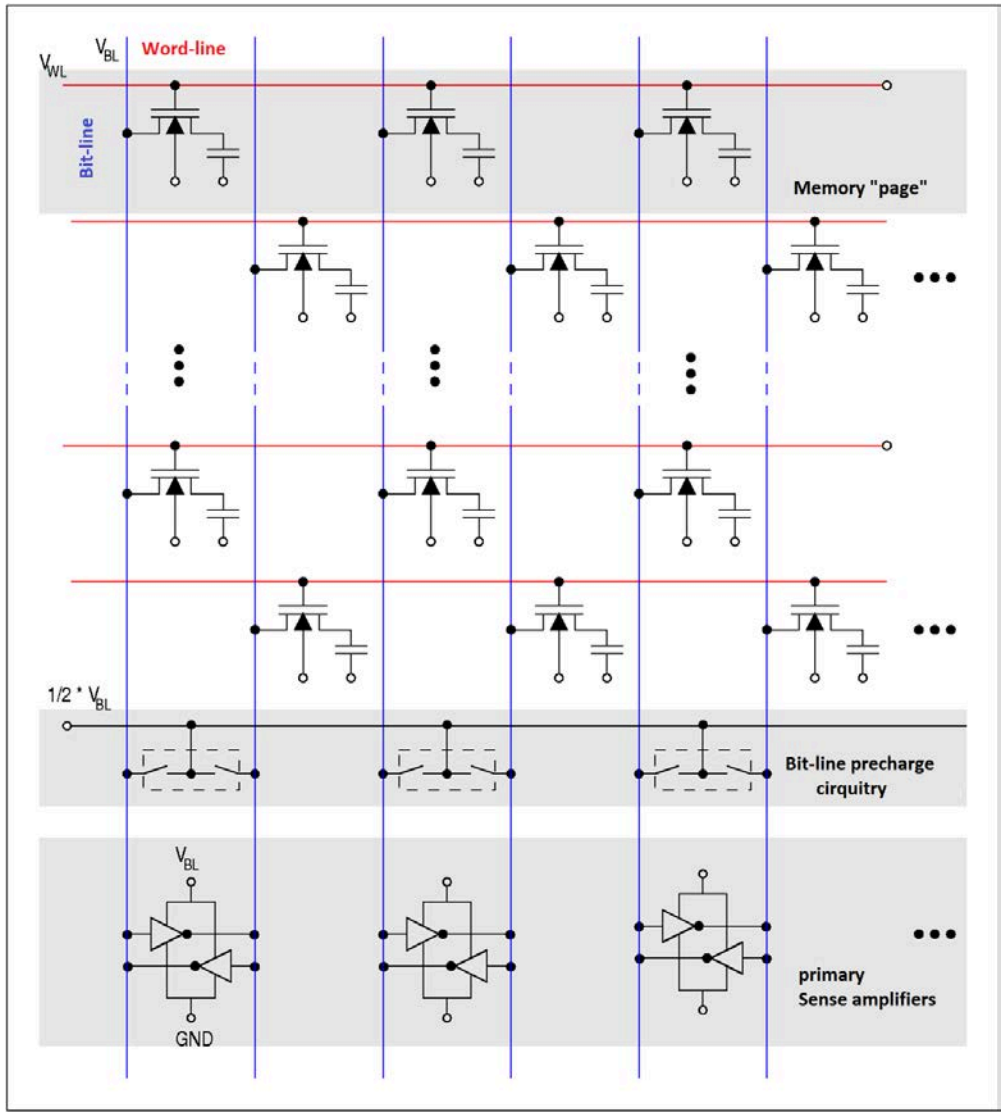



1. Back to beginning (just about). Need to recharge lines

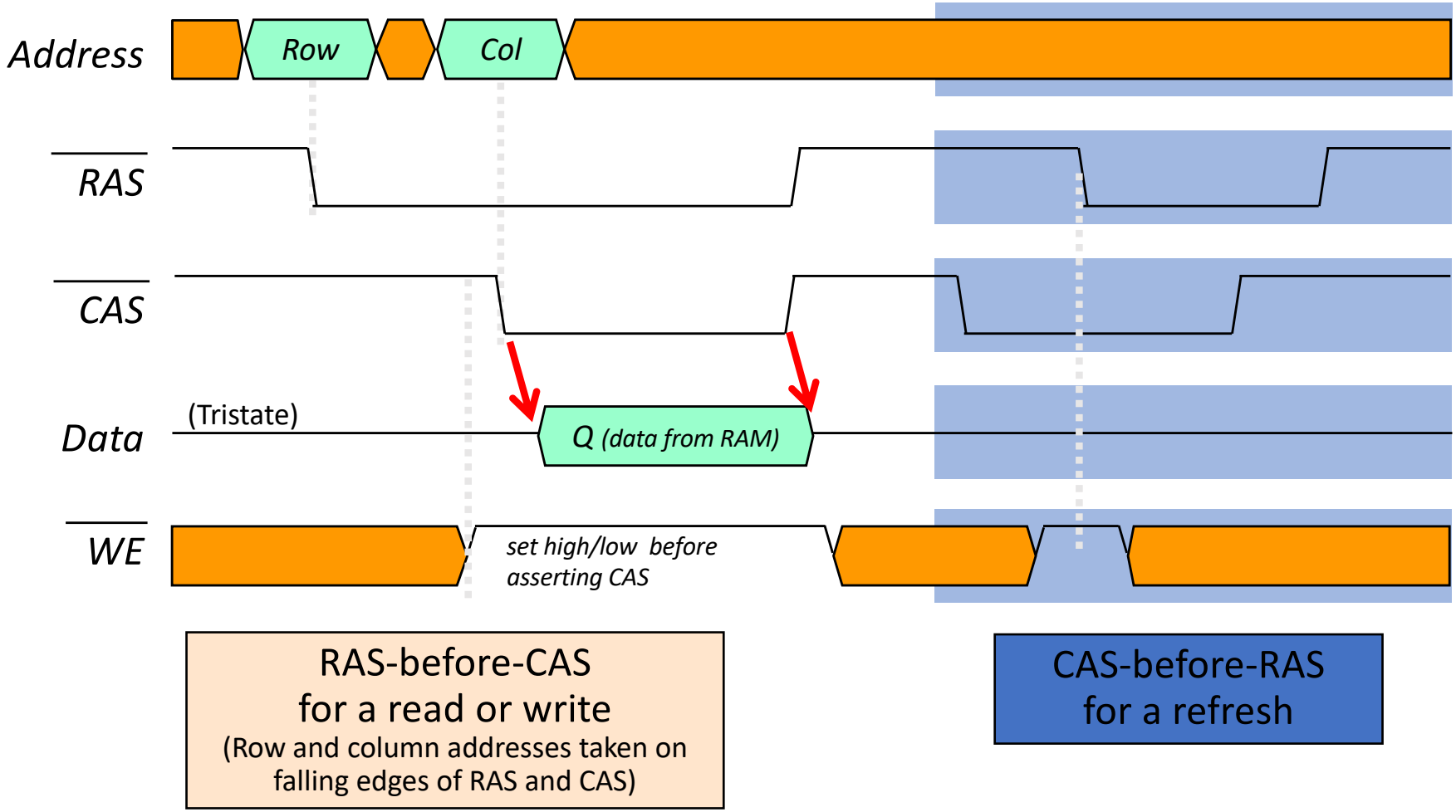


# DRAM Cells are Staggered Physically

- The sense amplifiers use two parallel bit lines (one active and one for reference) to detect the slight perturbation when you discharge the capacitor

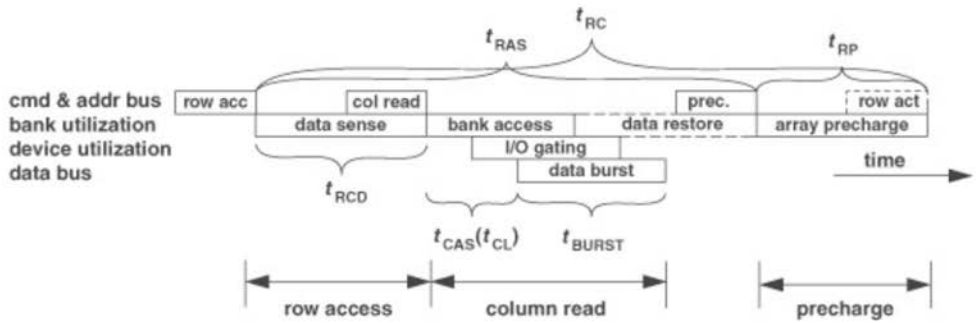


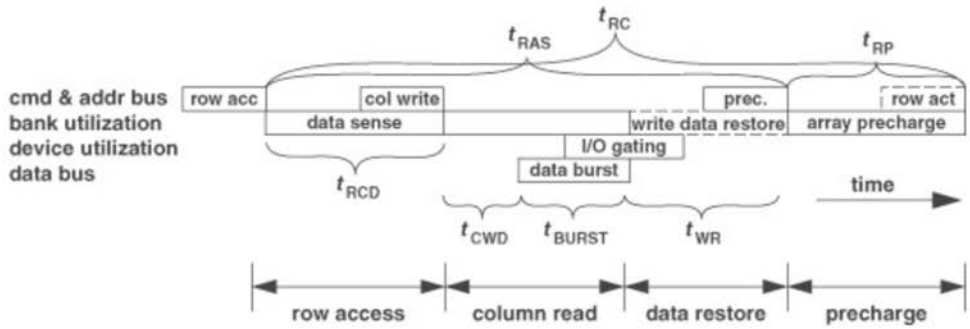
# Asynchronous DRAM Operation




- Clever manipulation of RAS and CAS after reads/writes provide more efficient modes: early-write, read-write, hidden-refresh, etc.
  (See datasheets for details)

# Read and Write Sequences




**Note: % of time data bus bandwidth is utilized**

Even though we can run DRAM very fast, because of all the maintenance involved in it (we have to clean up after ourselves every time we do something), there's a lot of downtime on the data bus. Compare that to the SRAM/ZBT from earlier!

*https://pubweb.eng.utah.edu/~cs7810/pres/dram-cs7810-protocolx2.pdf*

# Many Flavors of DRAM

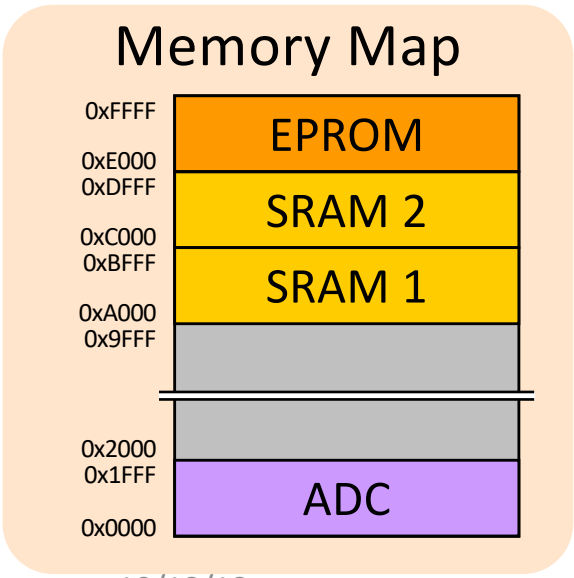
- DRAM (Asynchronous)
- SDRAM (Synchronous DRAM)
- Double-Data Rate DRAM (DDR SDRAM)
  - Double/Quadruple pump your data (work at integer multiples of your clock)

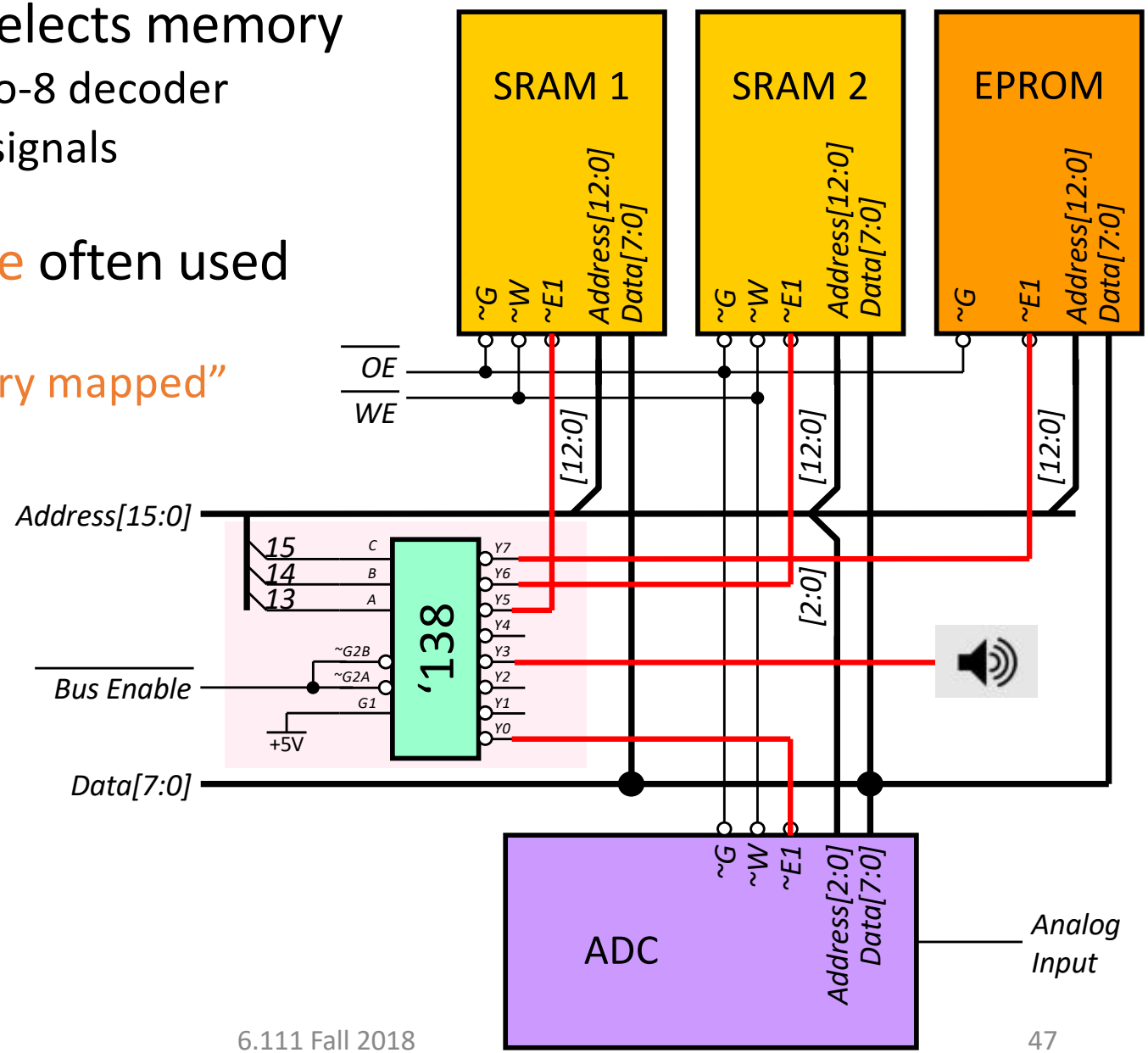
# Memory Devices: Helpful Knowledge

- SRAM vs. DRAM
  - SRAM holds state as long as power supply is turned on. DRAM must be "refreshed" – results in more complicated control
  - DRAM has much higher density, but requires special capacitor technology so usually separate chip since separate fab needed
  - FPGA usually implemented in a standard digital process technology and uses SRAM technology
- Non-Volatile Memory
  - Fast Read, but very slow write (EPROM must be removed from the system for programming!)
  - Holds state even if the power supply is turned off
  - Flash memory is slow, microsecond read, much longer writes
- Memory Internals
  - Has quite a bit of analog circuits internally -- pay particular attention to noise and PCB board integration
- Device details
  - Don't worry about them, wait until 6.012 or 6.374

# Addressing with Memory Maps

- Address decoder selects memory
  - Example: '138 3-to-8 decoder
  - Produces enable signals
- SRAM-like interface often used for peripherals
  - Known as "memory mapped" peripherals



# Memory

- control signals such as *Write Enable* should be registered
- a multi-cycle read/write is safer from a timing perspective than the single cycle read/write approach
- it is a bad idea to enable two tri-states driving the bus at the same time
- an SRAM does not need to be “refreshed” while a DRAM requires refresh (sometimes take care of on chip, though not always).
- an EPROM/EEPROM/FLASH cell can hold its state even if the power supply is turned off
- a synchronous memory can result in higher throughput

# Memories in Verilog

- `reg bit;  // a single register`
- `reg [31:0] word;  // a 32-bit register`
- `reg [31:0] array[15:0];  // 16 32-bit regs`
- `reg [31:0] array_2d[31:0][15:0];`
  `// 2 dimensional 32-bit array`

- `wire [31:0] read_data,write_data;`
  `wire [3:0] index;`

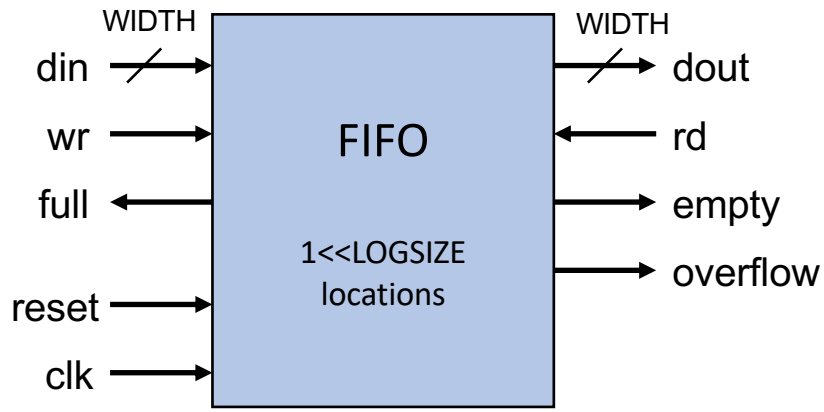
```
// combinational (asynch) read
assign read_data = array[index];

// clocked (synchronous) write
always @(posedge clock)
    array[index] <= write_data;
```

# Multi-port Memories (aka register files)

```
reg [31:0] regfile[30:0]; // 31 32-bit words

// Beta register file: 2 read ports, 1 write
wire [4:0] ra1,ra2,wa;
wire [31:0] rd1,rd2,wd;

assign ra1 = inst[20:16];
assign ra2 = ra2sel ? inst[25:21] : inst[15:11];
assign wa = wasel ? 5'd30 : inst[25:21];

// read ports
assign rd1 = (ra1 == 5’d31) ? 32’d0 : regfile[ra1];
assign rd2 = (ra2 == 5’d31) ? 32’d0 : regfile[ra2];
// write port
always @(posedge clk)
  if (werf) regfile[wa] <= wd;
```

# FIFOs

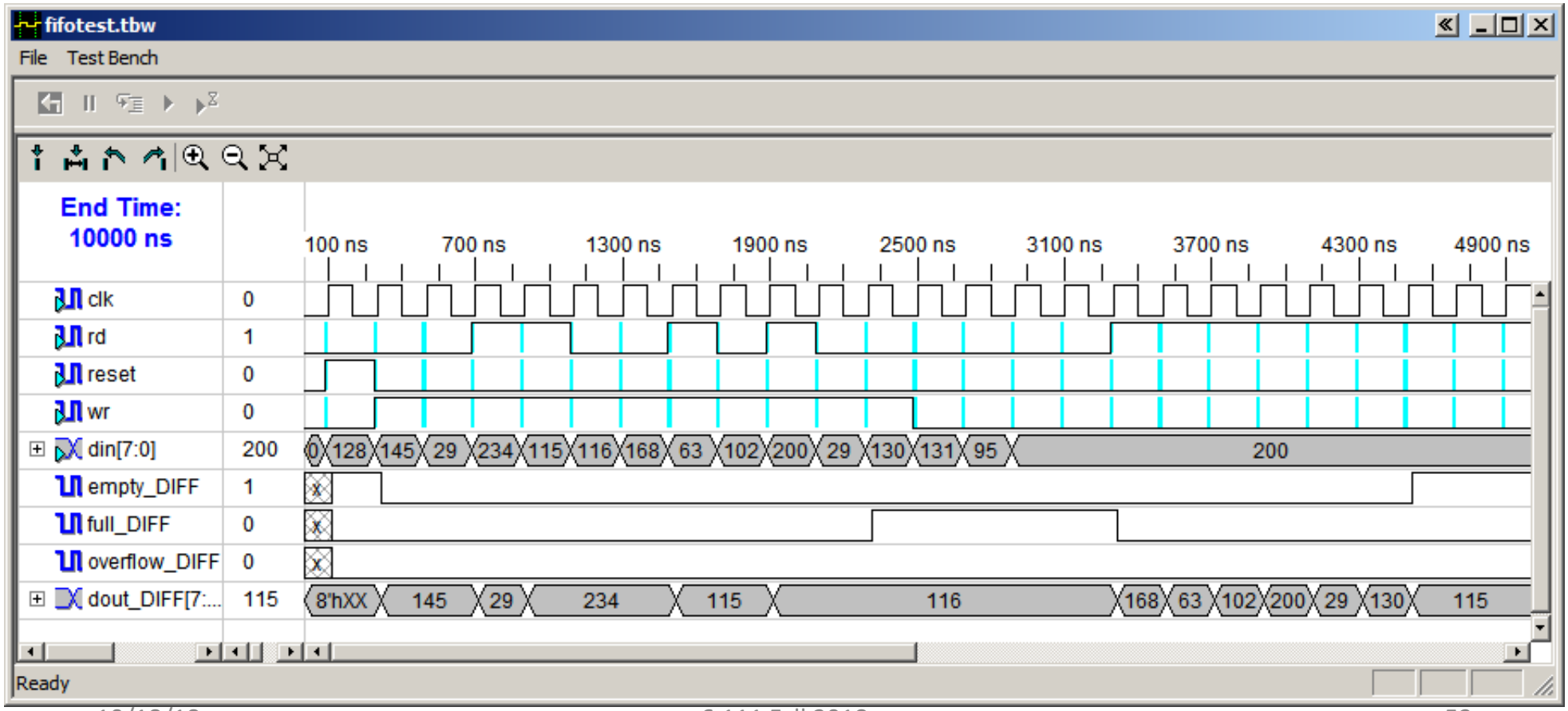
Code included with slides on site!




```
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
//    LOGSIZE  (parameter) FIFO has 1<<LOGSIZE elements
//    WIDTH    (parameter) each element has WIDTH bits
// Ports:
//    clk      (input) all actions triggered on rising edge
//    reset    (input) synchronously empties fifo
//    din      (input, WIDTH bits) data to be stored
//    wr       (input) when asserted, store new data
//    full     (output) asserted when FIFO is full
//    dout     (output, WIDTH bits) data read from FIFO
//    rd       (input) when asserted, removes first element
//    empty    (output) asserted when fifo is empty
//    overflow (output) asserted when WR but no room, cleared on next RD

module fifo #(parameter LOGSIZE = 2,   // default size is 4 elements
                        WIDTH = 4)     // default width is 4 bits
            (input clk,reset,wr,rd, input [WIDTH-1:0] din,
             output full,empty,overflow, output [WIDTH-1:0] dout);
…
endmodule
```

# FIFOs in action

```
// make a fifo with 8 8-bit locations
fifo f8x8 #(.LOGSIZE(3),.WIDTH(8))
          (.clk(clk),.reset(reset),
           .wr(wr),.din(din),.full(full),
           .rd(rd),.dout(dout),.empty(empty),
           .overflow(overflow));
```

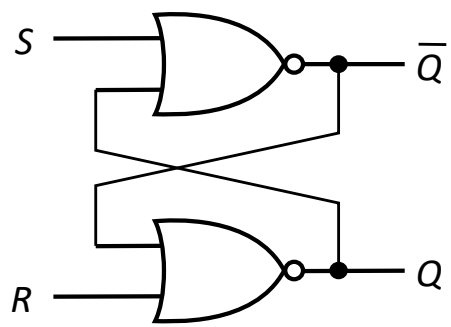


# FPGA memory implementation

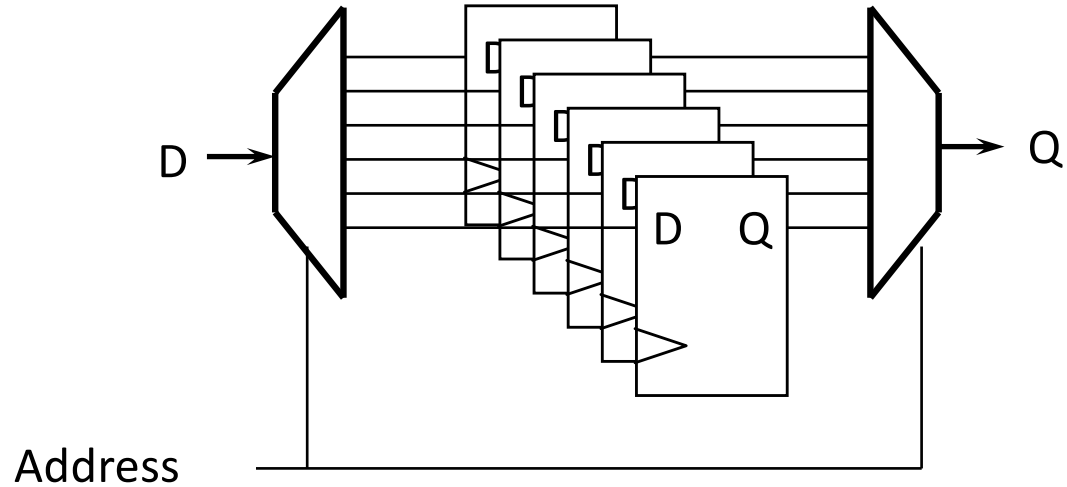
- Regular registers in logic blocks
  - Awful use of resources, but convenient & fast if small
  - SLOOOOOW to build sometimes. omg
- [Xilinx Vertex II] use the LUTs:
  - Single port: 16x(1,2,4,8), 32x(1,2,4,8), 64x(1,2), 128x1
  - Dual port (1 R/W, 1R): 16x1, 32x1, 64x1
  - Can fake extra read ports by cloning memory: all clones are written with the same addr/data, but each clone can have a different read address
- [Xilinx Vertex II] use block ram:
  - 18K bits: 16Kx1, 8Kx2, 4Kx4
    with parity: 2Kx(8+1), 1Kx(16+2), 512x(32+4)
  - Single or dual port
  - Pipelined (clocked) operations
  - Labkit XCV2V6000: 144 BRAMs, 2952K bits total

# Static RAMs: Latch Based Memory

## Set Reset Flip Flop



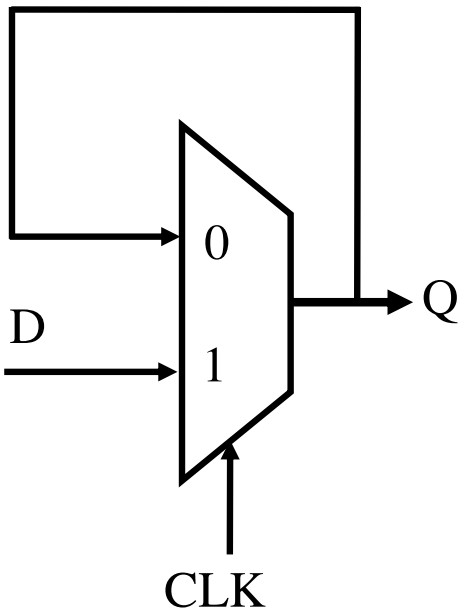

## Register Memory



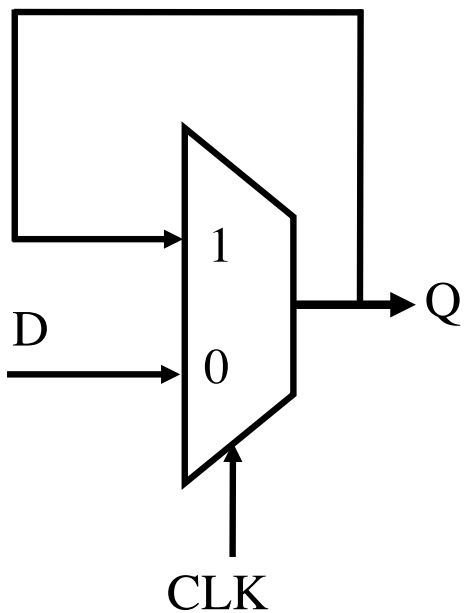

- Works fine for small memory blocks (e.g., small register files)
- Inefficient in area for large memories
- Density is the key metric in large memory circuits

# Latch and Register Based Memory

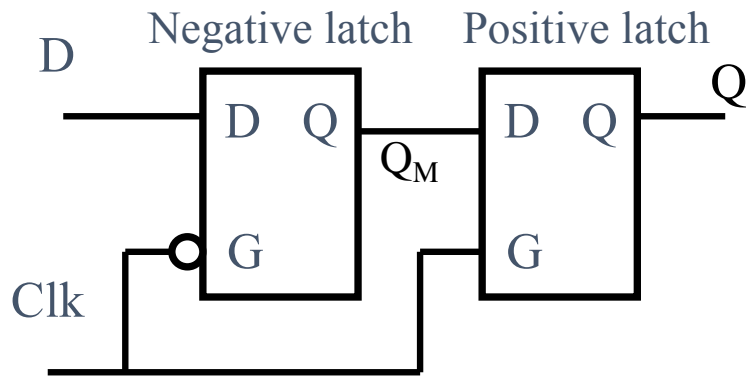
## Positive Latch




## Negative Latch



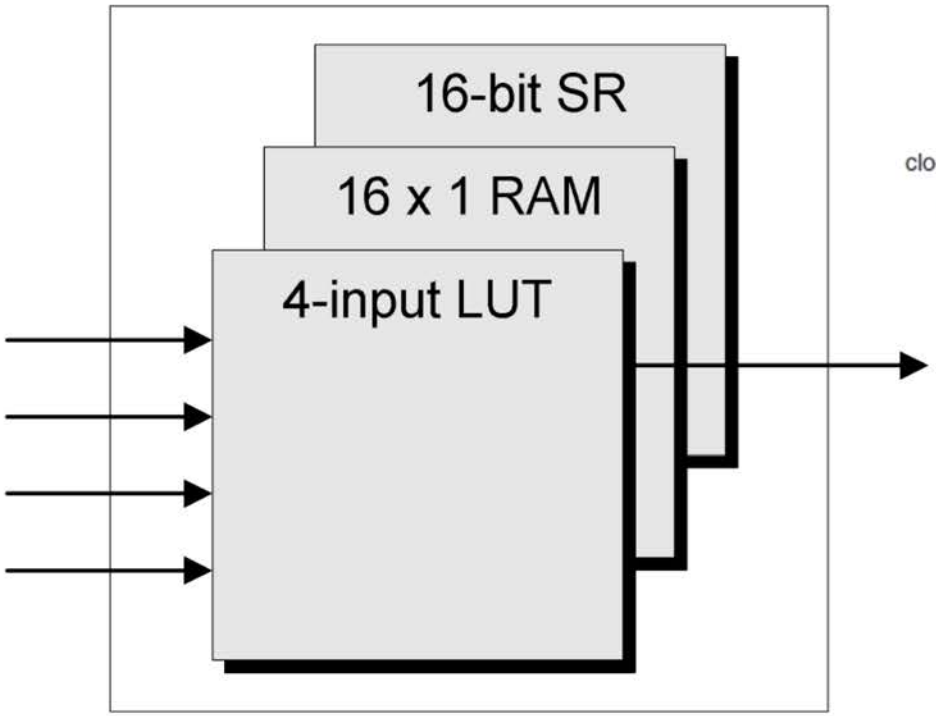

## Register Memory

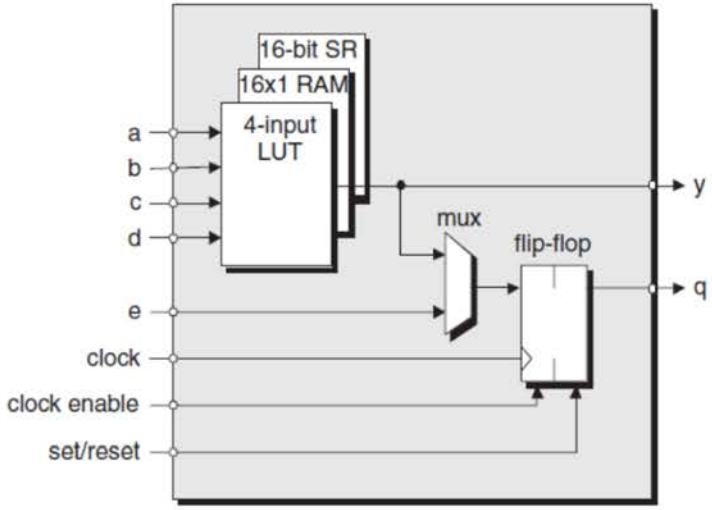



- Alternative view

# Xilinx Multipurpose LUT

On Nexys 4 DDR, you have about 14,000 Logic slices, each which has four 6 input LUTs and 8 Flip flops

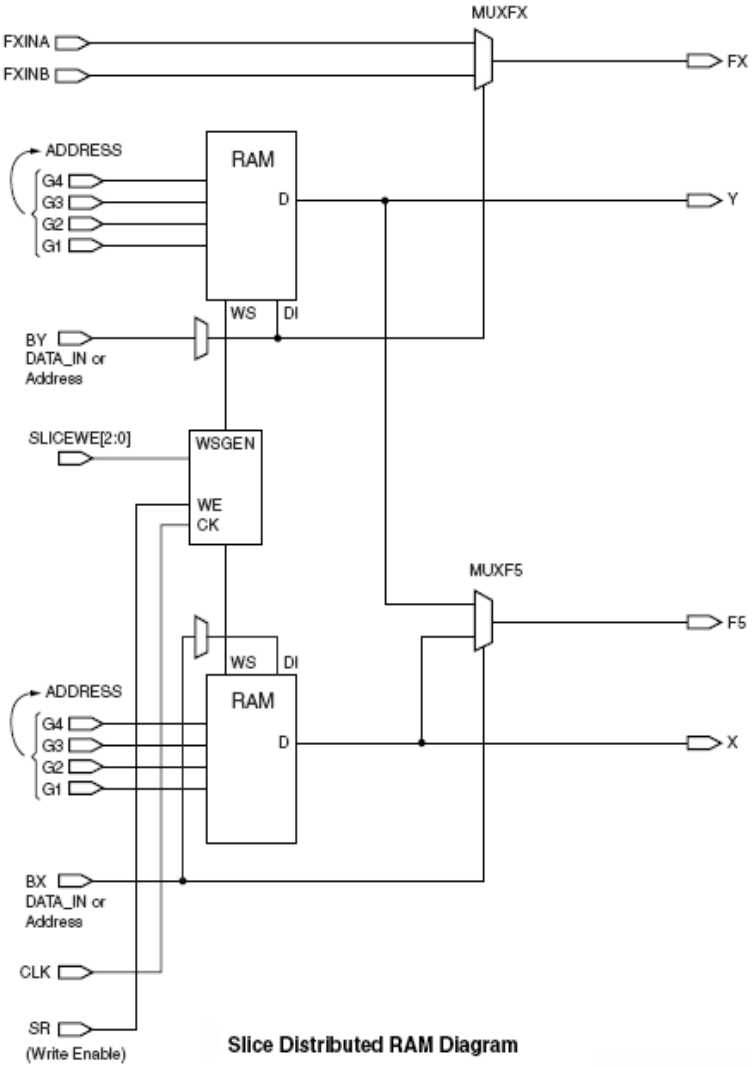


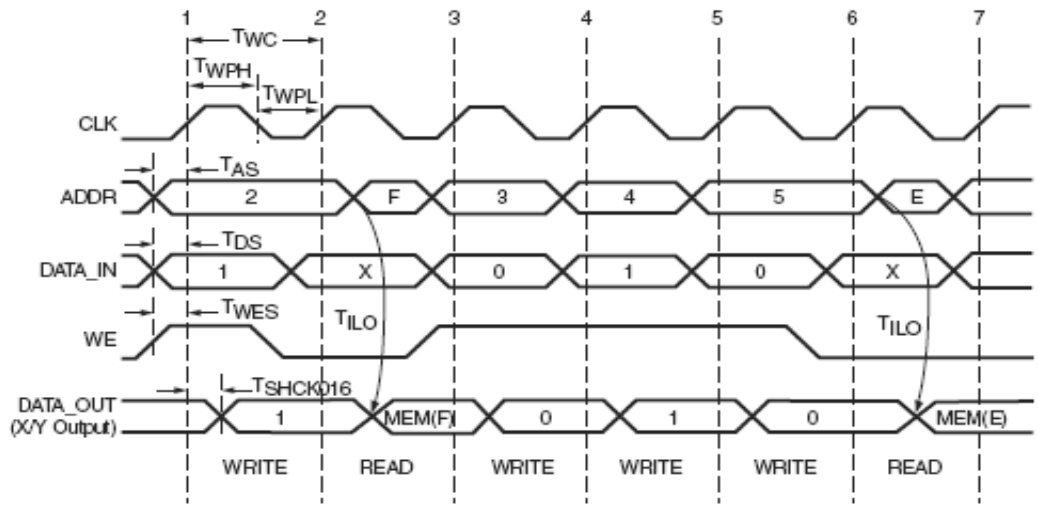

Simplified logic cell (LC)

The Design Warrior's Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043

# LUT-based RAMs




Slice Distributed RAM Diagram

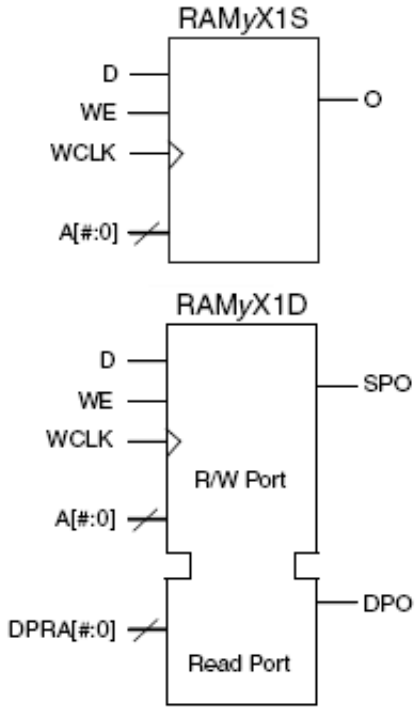



Slice Distributed RAM Timing Diagram

**CLB Distributed RAM Switching Characteristics**

| Description | Symbol | Speed Grade | | | Units |
|---|---|---|---|---|---|
| | | -6 | -5 | -4 | |
| **Sequential Delays** | | | | | |
| Clock CLK to X/Y outputs (WE active) in 16 x 1 mode | $T_{SHCKO16}$ | 1.63 | 1.79 | 2.05 | ns, Max |
| Clock CLK to X/Y outputs (WE active) in 32 x 1 mode | $T_{SHCKO32}$ | 1.97 | 2.17 | 2.49 | ns, Max |
| Clock CLK to F5 output | $T_{SHCKOF5}$ | 1.77 | 1.94 | 2.23 | ns, Max |
| **Setup and Hold Times Before/After Clock CLK** | | | | | |
| BX/BY data inputs (DIN) | $T_{DS}/T_{DH}$ | 0.53/–0.09 | 0.58/–0.10 | 0.67/–0.11 | ns, Min |
| F/G address inputs | $T_{AS}/T_{AH}$ | 0.40/ 0.00 | 0.44/ 0.00 | 0.50/ 0.00 | ns, Min |
| SR input (WS) | $T_{WES}/T_{WEH}$ | 0.42/–0.01 | 0.46/–0.01 | 0.53/–0.01 | ns, Min |
| **Clock CLK** | | | | | |
| Minimum Pulse Width, High | $T_{WPH}$ | 0.57 | 0.63 | 0.72 | ns, Min |
| Minimum Pulse Width, Low | $T_{WPL}$ | 0.57 | 0.63 | 0.72 | ns, Min |
| Minimum clock period to meet address write cycle time | $T_{WC}$ | 1.14 | 1.25 | 1.44 | ns, Min |
| **Combinatorial Delays** | | | | | |
| 4-input function: F/G inputs to X/Y outputs | $T_{ILO}$ | 0.35 | 0.39 | 0.44 | ns, Max |

# LUT-based RAM Modules




**Single-Port and Dual-Port Distributed SelectRAM Primitive**

**Single-Port and Dual-Port Distributed SelectRAM**

| Primitive | RAM Size | Type | Address Inputs |
|---|---|---|---|
| RAM16X1S | 16 bits | single-port | A3, A2, A1, A0 |
| RAM32X1S | 32 bits | single-port | A4, A3, A2, A1, A0 |
| RAM64X1S | 64 bits | single-port | A5, A4, A3, A2, A1, A0 |
| RAM128X1S | 128 bits | single-port | A6, A5, A4, A3, A2, A1, A0 |
| RAM16X1D | 16 bits | dual-port | A3, A2, A1, A0 |
| RAM32X1D | 32 bits | dual-port | A4, A3, A2, A1, A0 |
| RAM64X1D | 64 bits | dual-port | A5, A4, A3, A2, A1, A0 |

**Wider Library Primitives**

| Primitive | RAM Size | Data Inputs | Address Inputs | Data Outputs |
|---|---|---|---|---|
| RAM16x2S | 16 x 2-bit | D1, D0 | A3, A2, A1, A0 | O1, O0 |
| RAM32X2S | 32 x 2-bit | D1, D0 | A4, A3, A2, A1, A0 | O1, O0 |
| RAM64X2S | 64 x 2-bit | D1, D0 | A5, A4, A3, A2, A1, A0 | O1, O0 |
| RAM16X4S | 16 x 4-bit | D3, D2, D1, D0 | A3, A2, A1, A0 | O3, O2, O1, O0 |
| RAM32X4S | 32 x 4-bit | D3, D2, D1, D0 | A4,A3, A2, A1, A0 | O3, O2, O1, O0 |
| RAM16X8S | 16 x 8-bit | D <7:0> | A3, A2, A1, A0 | O <7:0> |
| RAM32X8S | 32 x 8-bit | D <7:0> | A4,A3, A2, A1, A0 | O <7:0> |

```
// instantiate a LUT-based RAM module
RAM16X1S mymem #(.INIT(16'b0110_1111_0011_0101_1100))  // msb first
                (.D(din),.O(dout),.WE(we),.WCLK(clock_27mhz),
                 .A0(a[0]),.A1(a[1]),.A2(a[2]),.A3(a[3]));
```

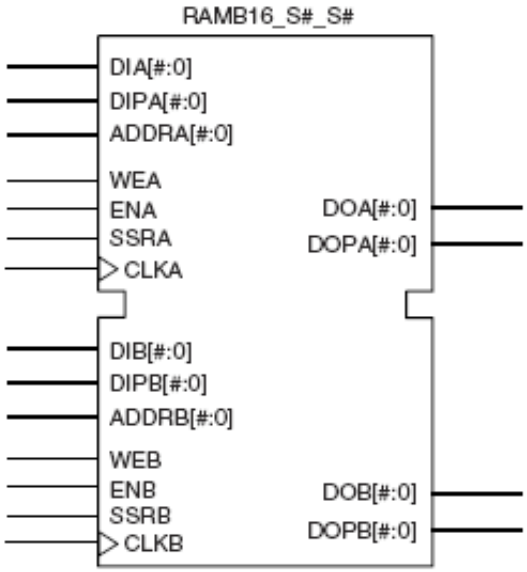
# Tools will often build these for you…

From Lab 2:

```
reg [7:0] segments;
always @ (switch[3:0]) begin
  case (switch[3:0])
  4'h0: segments[6:0] = 7'b0111111;
  4'h1: segments[6:0] = 7'b0000110;
  4'h2: segments[6:0] = 7'b1011011;
  4'h3: segments[6:0] = 7'b1001111;
  4'h4: segments[6:0] = 7'b1100110;
  4'h5: segments[6:0] = 7'b1101101;
  4'h6: segments[6:0] = 7'b1111101;
  4'h7: segments[6:0] = 7'b0000111;
  4'h8: segments[6:0] = 7'b1111111;
  4'h9: segments[6:0] = 7'b1100111;
  4'hA: segments[6:0] = 7'b1110111;
  4'hB: segments[6:0] = 7'b1111100;
  4'hC: segments[6:0] = 7'b1011000;
  4'hD: segments[6:0] = 7'b1011110;
  4'hE: segments[6:0] = 7'b1111001;
  4'hF: segments[6:0] = 7'b1110001;
  default: segments[6:0] = 7'b00000000;
  endcase
  segments[7] = 1'b0;  // decimal point
end
```

```
=============================================
*             HDL Synthesis                 *
=============================================

Synthesizing Unit <lab2_2>.
    Related source file is "../lab2_2.v".
    ...
    Found 16x7-bit ROM for signal <$n0000>.
    ...
    Summary:
            inferred   1 ROM(s).
        ...
Unit <lab2_2> synthesized.

=============================================
Timing constraint: Default path analysis
Total number of paths / destination ports: 28 / 7
-------------------------------------------------
Delay:               7.244ns (Levels of Logic = 3)
Source:            switch<3> (PAD)
Destination:       user1<0> (PAD)

Data Path: switch<3> to user1<0>
                     Gate     Net
Cell:in->out fanout Delay   Delay  Logical Name
----------------------------------  ------------
IBUF:I->O       7   0.825   1.102  switch_3_IBUF
LUT4:I0->O      1   0.439   0.517  Mrom__n0000_inst_lut4_01
OBUF:I->O           4.361          user1_0_OBUF
----------------------------------------
Total               7.244ns (5.625ns logic, 1.619ns route)
                            (77.7% logic, 22.3% route)
```

# Block Memories (BRAMs)

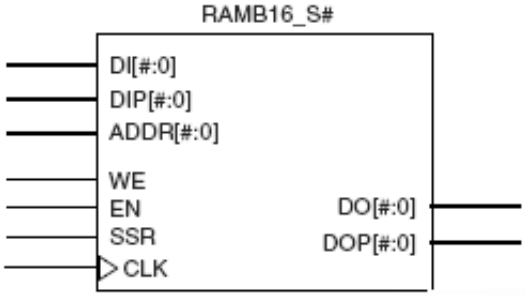



Dual-Port Block RAM Primitive

$(W_{DATA} + W_{PARITY})*(LOCATIONS) = 18K$ bits

1,2,4 16K,8K,4K,2K,1K,512

1
2
4
8
16
32




Single-Port Block RAM Primitive

**Dual-Port Block RAM Primitives**

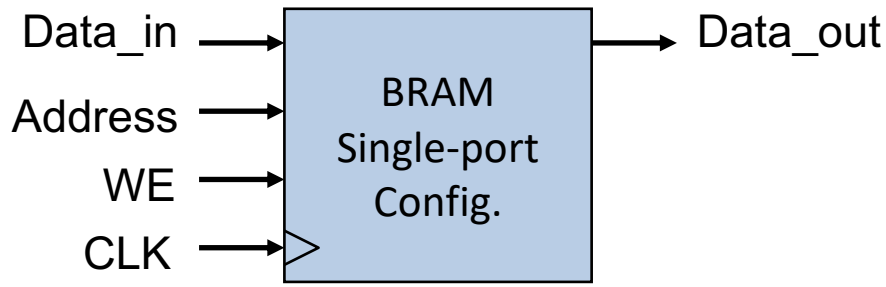
| Primitive | Port A Width | Port B Width |
|---|---|---|
| RAMB16_S1_S1 | 1 | 1 |
| RAMB16_S1_S2 | | 2 |
| RAMB16_S1_S4 | | 4 |
| RAMB16_S1_S9 | | (8+1) |
| RAMB16_S1_S18 | | (16+2) |
| RAMB16_S1_S36 | | (32+4) |
| RAMB16_S2_S2 | 2 | 2 |
| RAMB16_S2_S4 | | 4 |
| RAMB16_S2_S9 | | (8+1) |
| RAMB16_S2_S18 | | (16+2) |
| RAMB16_S2_S36 | | (32+4) |
| RAMB16_S4_S4 | 4 | 4 |
| RAMB16_S4_S9 | | (8+1) |
| RAMB16_S4_S18 | | (16+2) |
| RAMB16_S4_S36 | | (32+4) |
| RAMB16_S9_S9 | (8+1) | (8+1) |
| RAMB16_S9_S18 | | (16+2) |
| RAMB16_S9_S36 | | (32+4) |
| RAMB16_S18_S18 | (16+2) | (16+2) |
| RAMB16_S18_S36 | | (32+4) |
| RAMB16_S36_S36 | (32+4) | (32+4) |

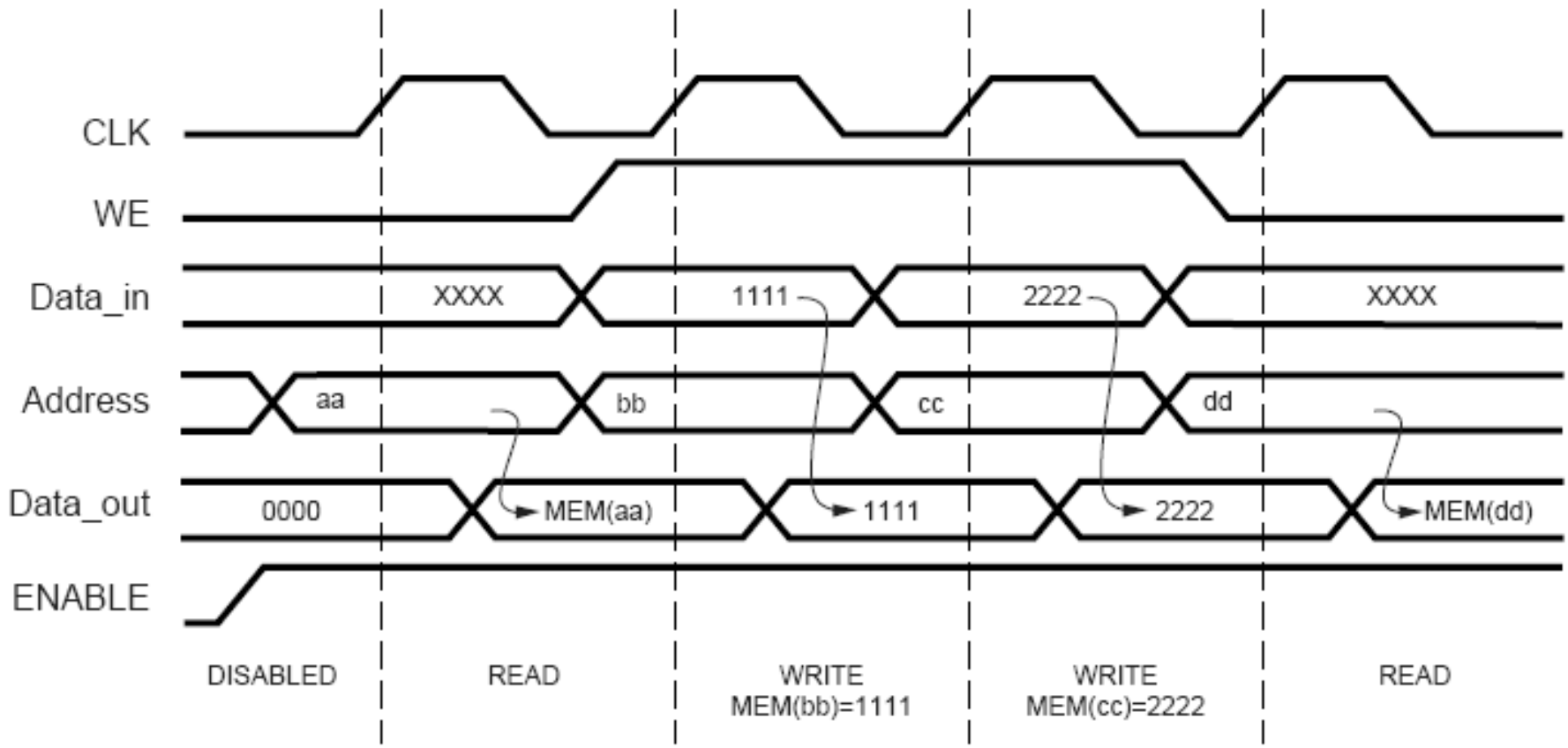
**Single-Port Block RAM Primitives**

| Primitive | Port Width |
|---|---|
| RAMB16_S1 | 1 |
| RAMB16_S2 | 2 |
| RAMB16_S4 | 4 |
| RAMB16_S9 | (8+1) |
| RAMB16_S18 | (16+2) |
| RAMB16_S36 | (32+4) |

# BRAM Operation

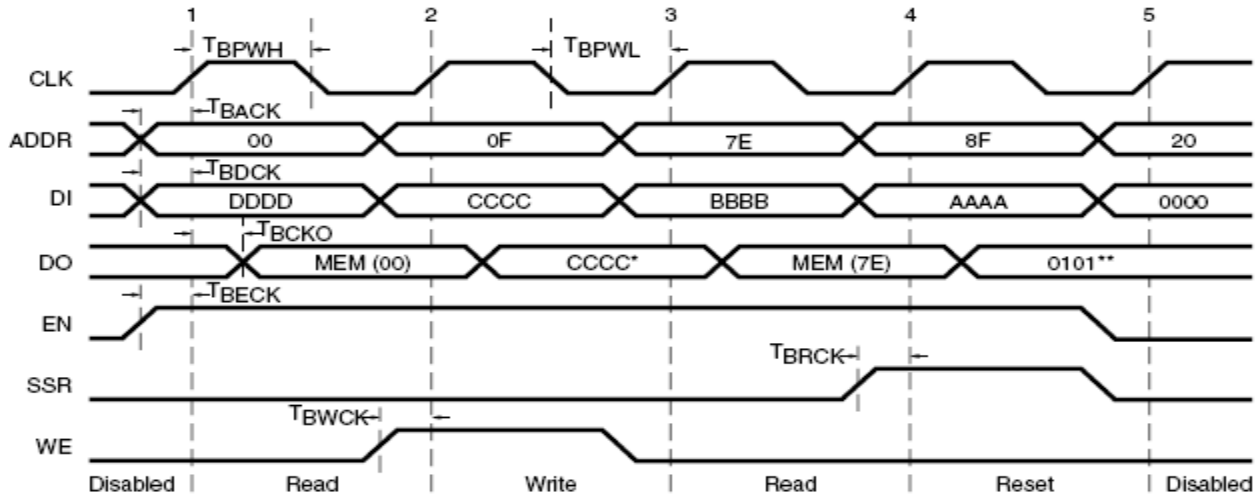
*Similar to what we did earlier with SRAM (BRAM is just a **B**lock of **SRAM** on the FPGA)*




Source: Xilinx App Note 463

# BRAM Timing




**Block SelectRAM Timing Diagram**

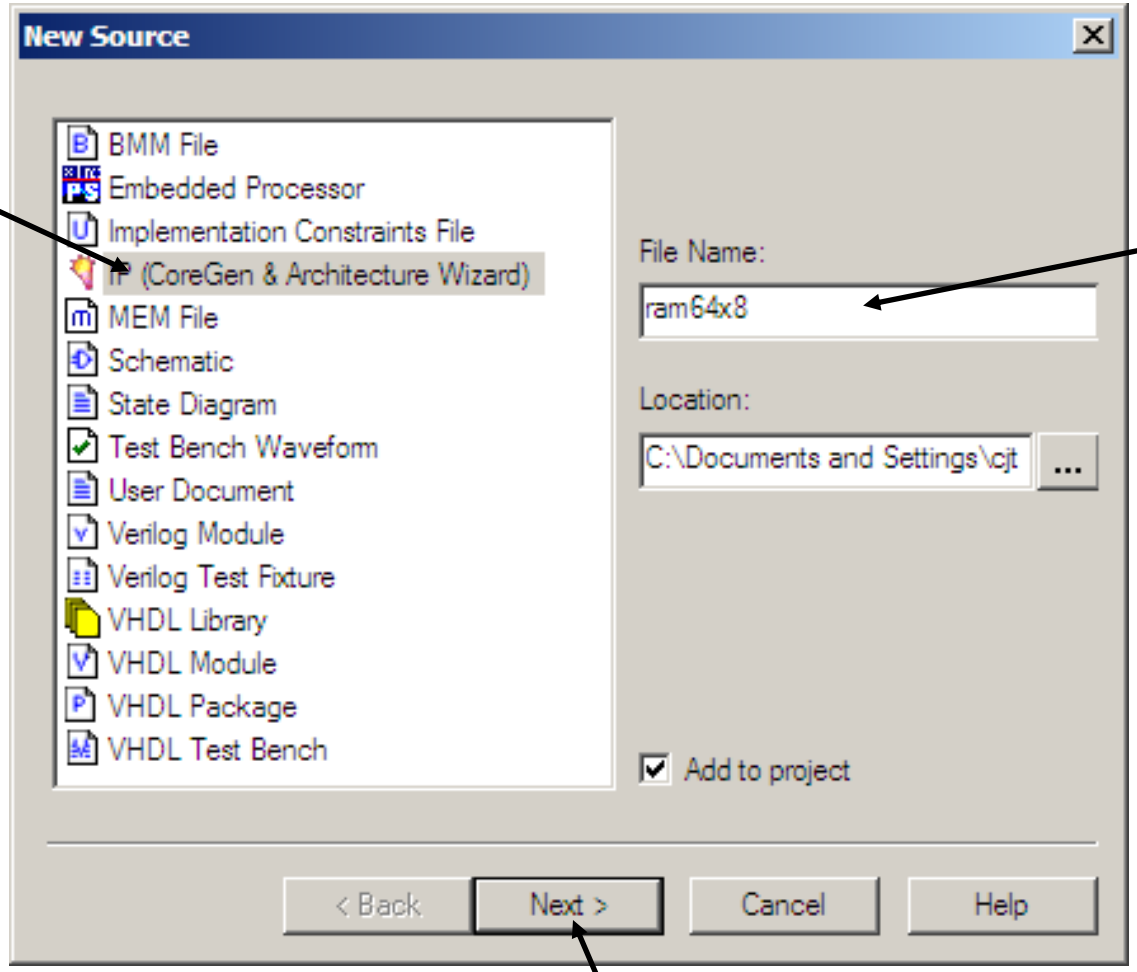
**Block SelectRAM Switching Characteristics**

| Description | Symbol | Speed Grade | | | Units |
|---|---|---|---|---|---|
| | | -6 | -5 | -4 | |
| **Sequential Delays** | | | | | |
| Clock CLK to DOUT output | $T_{BCKO}$ | 2.10 | 2.31 | 2.65 | ns, Max |
| **Setup and Hold Times Before Clock CLK** | | | | | |
| ADDR inputs | $T_{BACK}/T_{BCKA}$ | 0.29/ 0.00 | 0.32/ 0.00 | 0.36/ 0.00 | ns, Min |
| DIN inputs | $T_{BDCK}/T_{BCKD}$ | 0.29/ 0.00 | 0.32/ 0.00 | 0.36/ 0.00 | ns, Min |
| EN input | $T_{BECK}/T_{BCKE}$ | 0.95/–0.46 | 1.04/–0.50 | 1.20/–0.58 | ns, Min |
| RST input | $T_{BRCK}/T_{BCKR}$ | 1.31/–0.71 | 1.44/–0.78 | 1.65/–0.90 | ns, Min |
| WEN input | $T_{BWCK}/T_{BCKW}$ | 0.57/–0.19 | 0.63/–0.21 | 0.72/–0.25 | ns, Min |
| **Clock CLK** | | | | | |
| CLKA to CLKB setup time for different ports | $T_{BCCS}$ | 1.0 | 1.0 | 1.0 | ns, min |
| Minimum Pulse Width, High | $T_{BPWH}$ | 1.17 | 1.29 | 1.48 | ns, Min |
| Minimum Pulse Width, Low | $T_{BPWL}$ | 1.17 | 1.29 | 1.48 | ns, Min |

# Using BRAMs (eg, a 64Kx8 RAM)

- From menus: Project → New Source…

Select "IP"



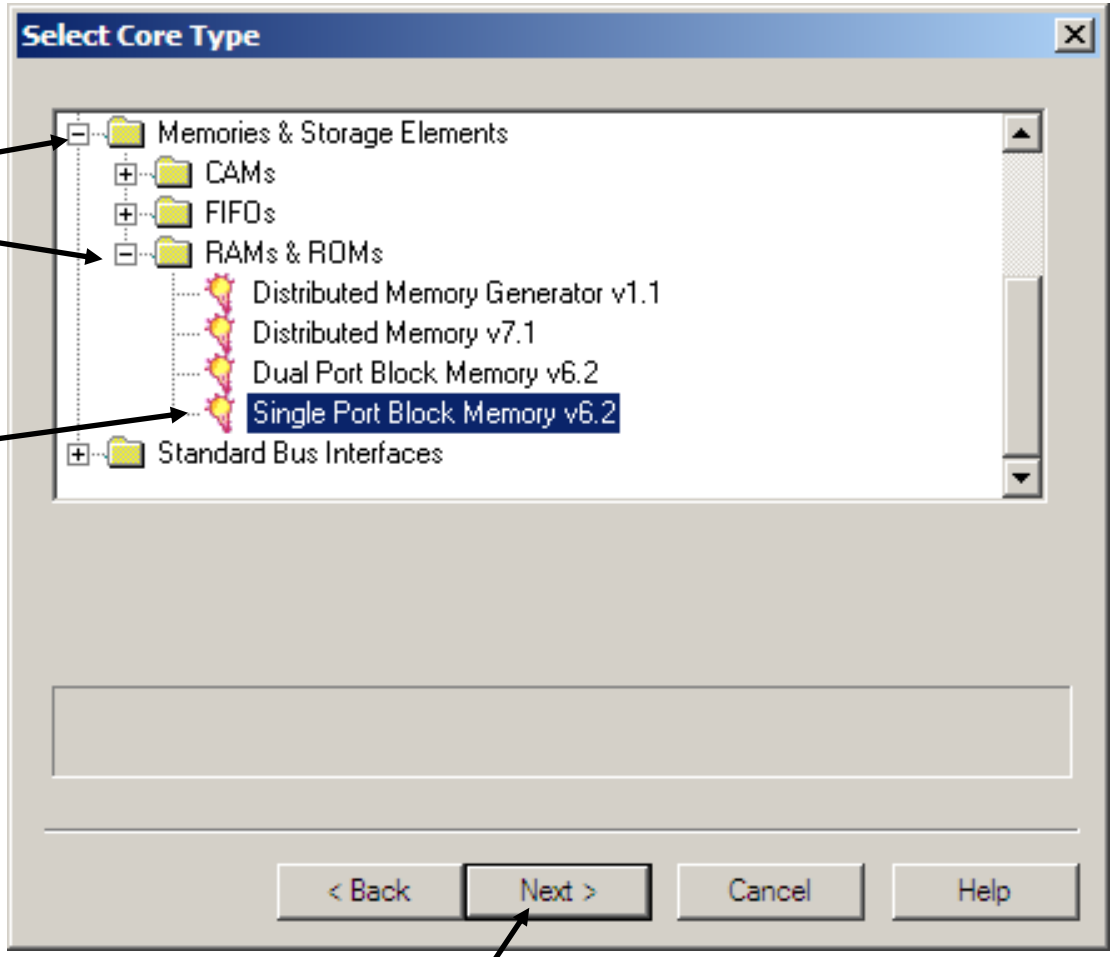

Fill in name

Click "Next" when done…

# BRAM Example

Click open folders

Select “Single Port Block Memory”




Click “Next” and then “Finish” on next window

# BRAM Example

Fill in name (again?!)

Select RAM vs ROM

Fill in width & depth

Usually “Read After Write” is what you want

Click “Next” …

Single Port Block Memory

Parameters | Core Overview | Contact | Web Links

LogiCORE

Single Port Block Memory

Component Name: ram64x8

Port Configuration

- Read And Write
- Read Only

Memory Size

Width: 8 — Valid Range 1..256

Depth: 65536 — Valid Range: 2..524288

Write Mode

- Read After Write
- Read Before Write
- No Read On Write

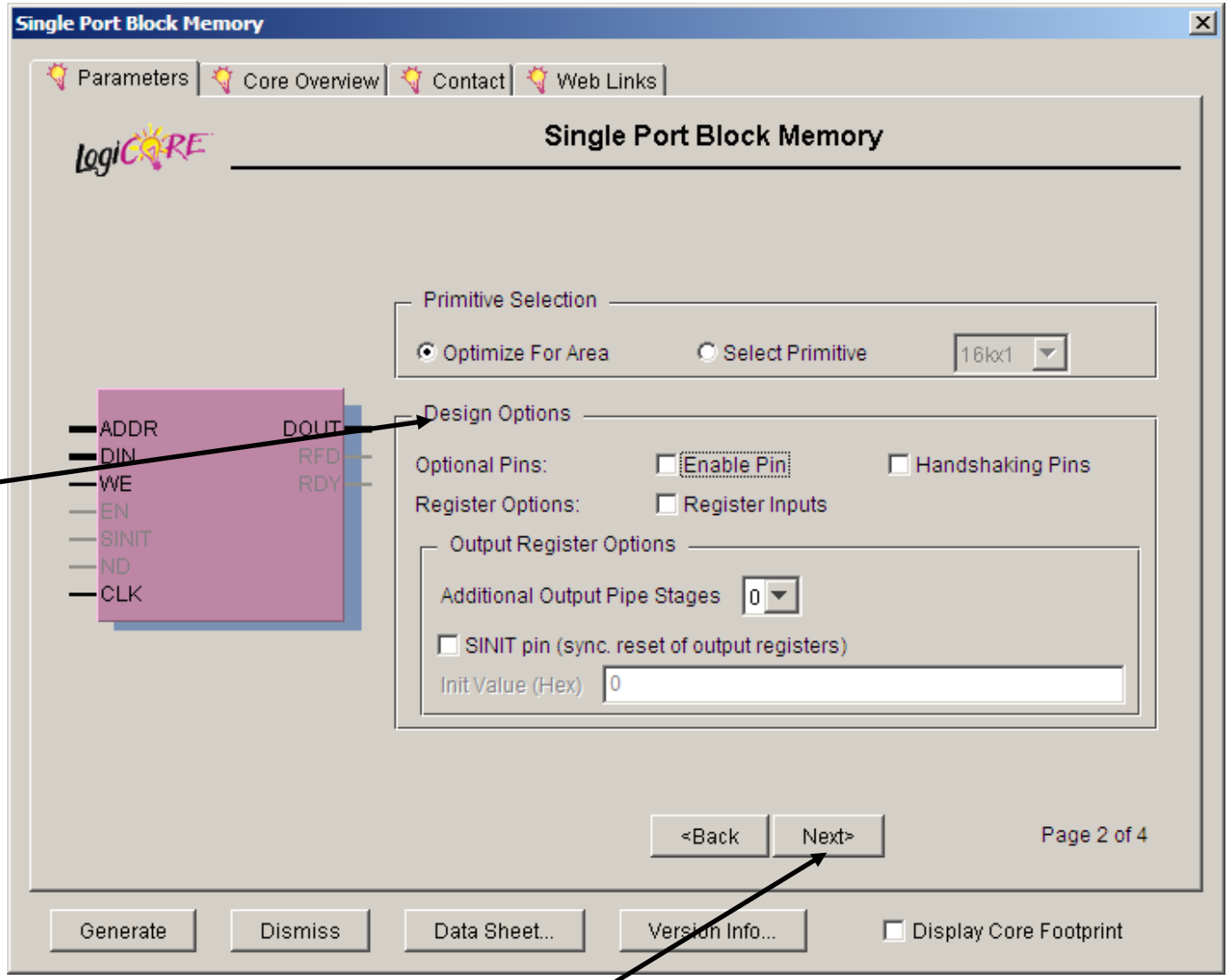
ADDR, DIN, WE, EN, SINIT, ND, CLK, DOUT, RFD, RDY

<Back | Next> | Page 1 of 4

Generate | Dismiss | Data Sheet... | Version Info... | Display Core Footprint

# BRAM Example

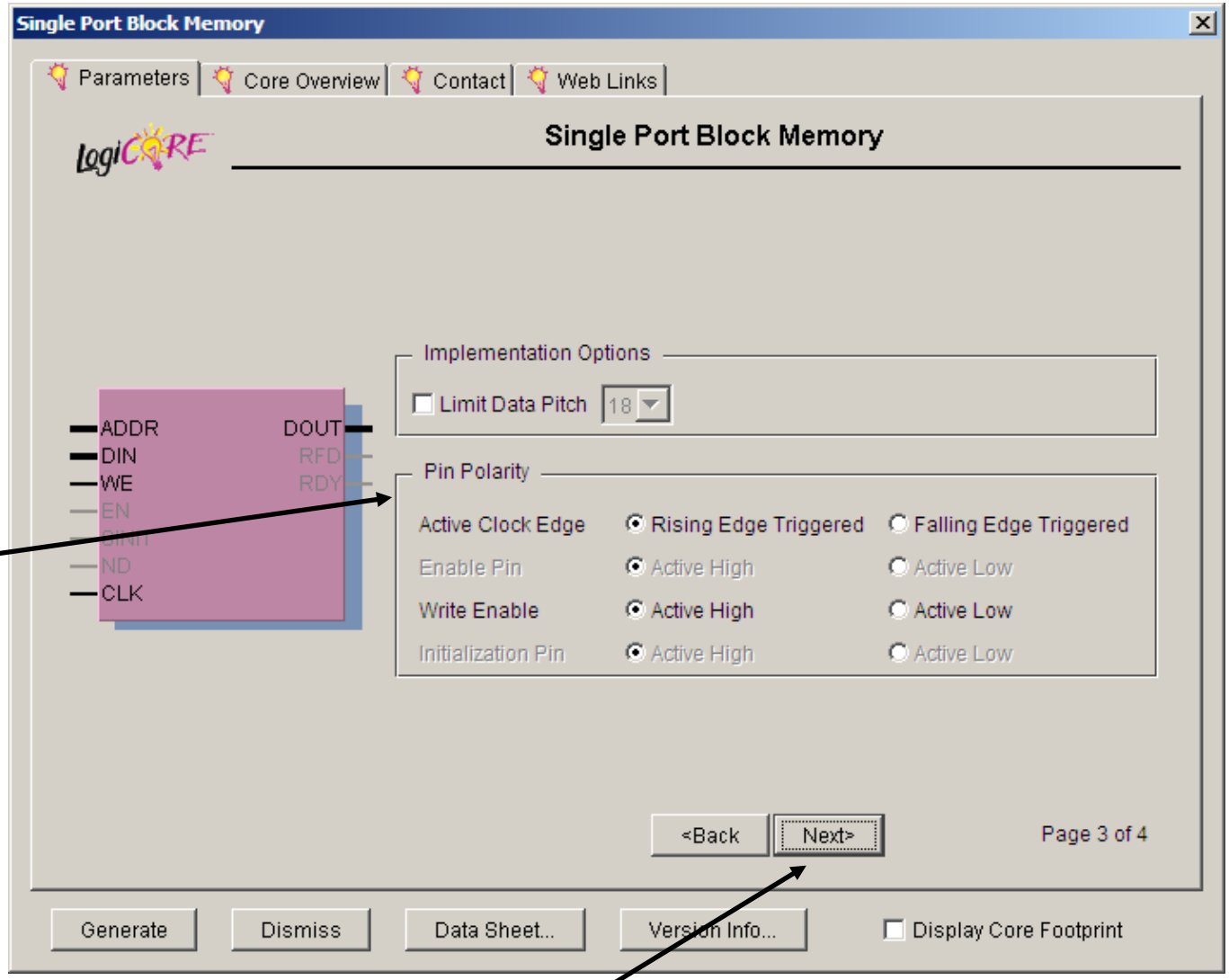
Can add extra control pins, but usually not




Click “Next” …

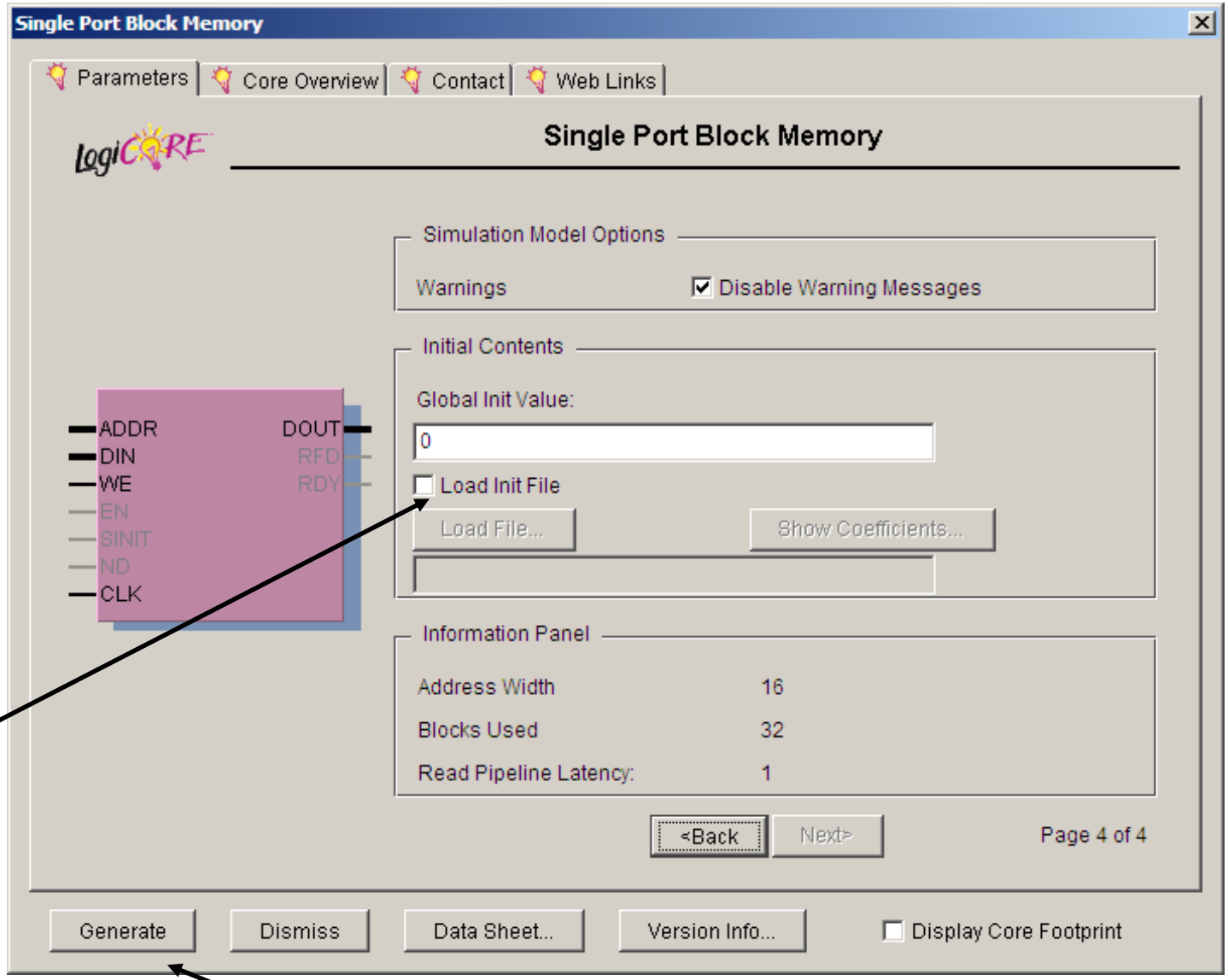
# BRAM Example

Select polarity of control pins; active high default is usually just fine




Click “Next” …

# BRAM Example




Click to name a .coe file that specifies initial contents (eg, for a ROM)

Click "Generate" to complete

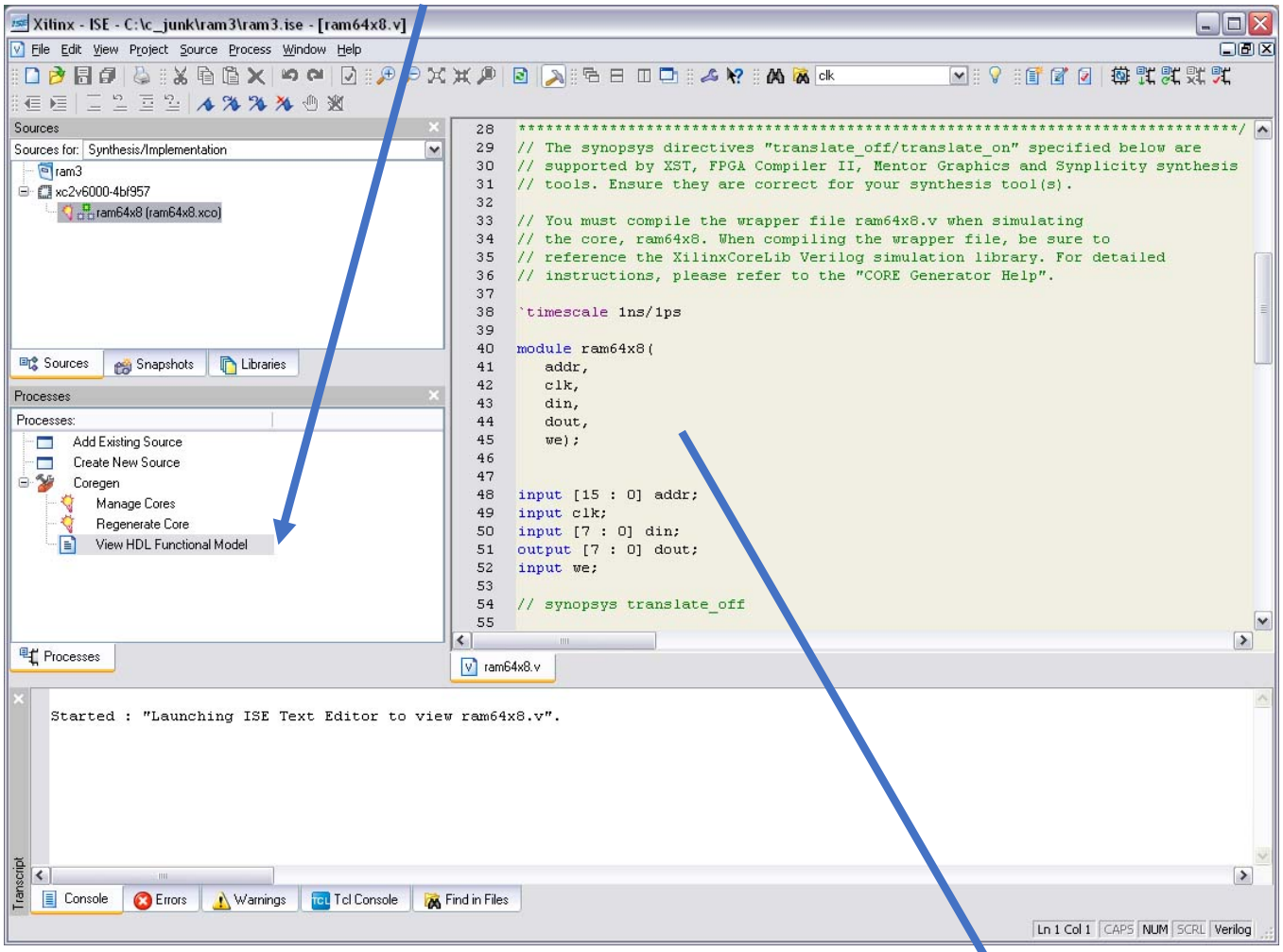
# .coe file format

```
memory_initialization_radix=2;
memory_initialization_vector=

00000000,
00111110,
01100011,
00000011,
00000011,
00011110,
00000011,
00000011,
01100011,
00111110,
00000000,
00000000,
```

Memory contents with location 0 first, then location 1, etc. You can specify input radix, in this example we're using binary. MSB is on the left, LSB on the right. Unspecified locations (if memory has more locations than given in .coe file) are set to 0.

# Using result in your Verilog

- Look at generated Verilog for module defintion (click on “View HDL Functional Model” under Coregen):




- Use to instantiate instances in your code:
  `ram64x8 foo(.addr(addr),.clk(clk),.we(we),.din(din),.dout(dout));`

# Labkit Memory

- Regular registers in logic blocks
  - Operates at system clock speed, expensive (CLB utilization)
  - Configuration set by Verilog design (eg FIFO, single/dual port, etc)
- FPGA Distributed memory
  - Operates at system clock speed
  - Uses LUTs (16 bits) for implementation, expensive (CLB utilization)
  - Requires significant routing for implementation
  - Configured using CoreGen
  - Theoretical maximum: 1Mbit
- FPGA block ram:
  - Implemented with (18 kbit) dedicated memory blocks distributed throughout the FPGA
  - Pipelined (clocked) operations
  - Labkit XCV2V6000: 144 BRAMs, 2952K bits total
- ZBT SRAM
  - two synchronous, 512k x 36 ZBT SRAM chips
  - Operates up to 167MHz
- Flash memory
  - 128Mbits with 100,000 minimum erase cycle per block
  - Slow read access, even slower write access time!
  - Must cache to ZBT or BRAM for video display

# Nexys4 DDR Memory

- Regular registers in logic blocks
  - Operates at system clock speed, expensive (CLB utilization)
  - Configuration set by Verilog design (eg FIFO, single/dual port, etc)
- FPGA Distributed memory (avoid if possible)
  - Operates at system clock speed
  - Uses LUTs (16 bits) for implementation, expensive (CLB utilization)
  - Requires significant routing for implementation
  - Configured using IP
  - Theoretical maximum: 1Mbit
- FPGA block ram:
  - 4,860K bits total
- DDR2 SDRAM
  - 128MiB (Megabytes)
  - Requires MIG (Memory Interface Generator) Wizard
- Flash memory
  - 16MiB
  - Slow read access, even slower write access time!
- microSD port
  - Tested with 2GB (Windows 7, FPGA)