

L7: Memory Basics and Timing



Acknowledgements:

Materials in this lecture are courtesy of the following sources and are used with permission.

Nathan Ickes

Rex Min

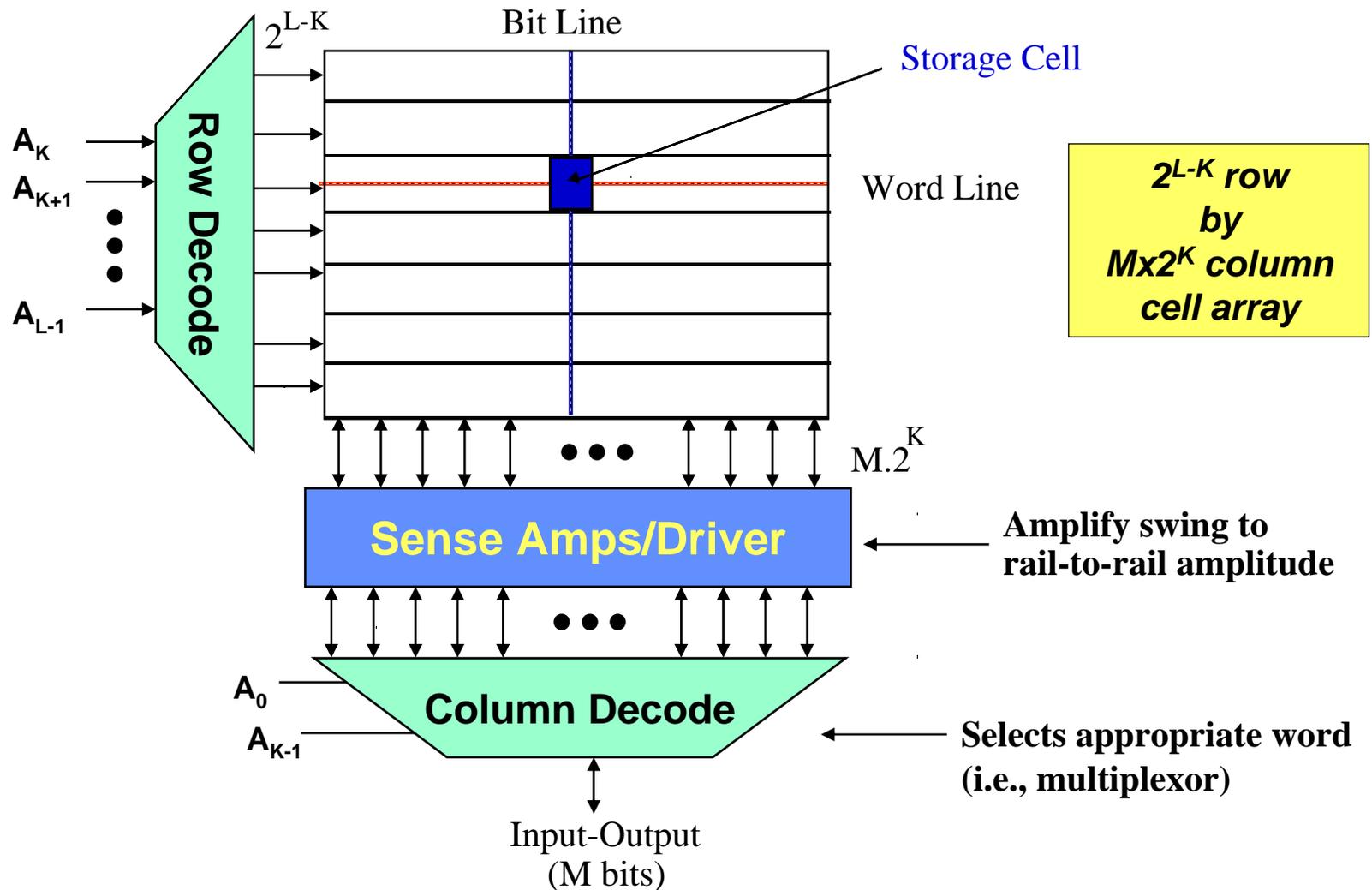
Yun Wu

J. Rabaey, A. Chandrakasan, B. Nikolic. *Digital Integrated Circuits: A Design Perspective*.
Prentice Hall/Pearson, 2003.

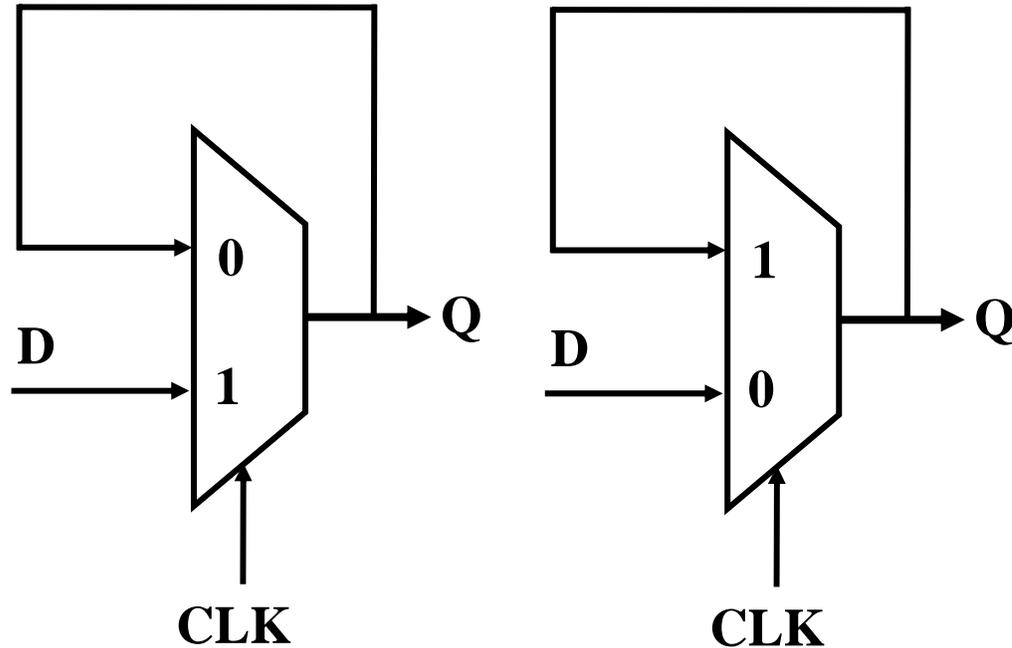
Read-Write Memory		Non-Volatile Read-Write Memory	Read-Only Memory (ROM)
Random Access	Non-Random Access	EPROM E ² PROM FLASH	Mask-Programmed
SRAM DRAM	FIFO LIFO		

Key Design Metrics:

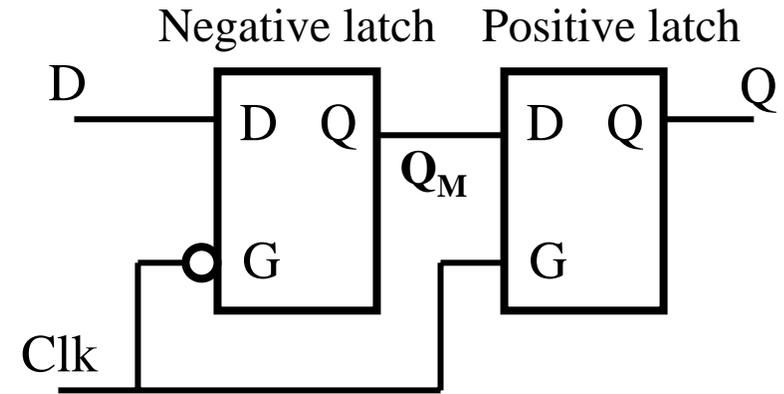
1. Memory Density (number of bits/ μm^2) and Size
2. Access Time (time to read or write) and Throughput
3. Power Dissipation



Positive Latch Negative Latch

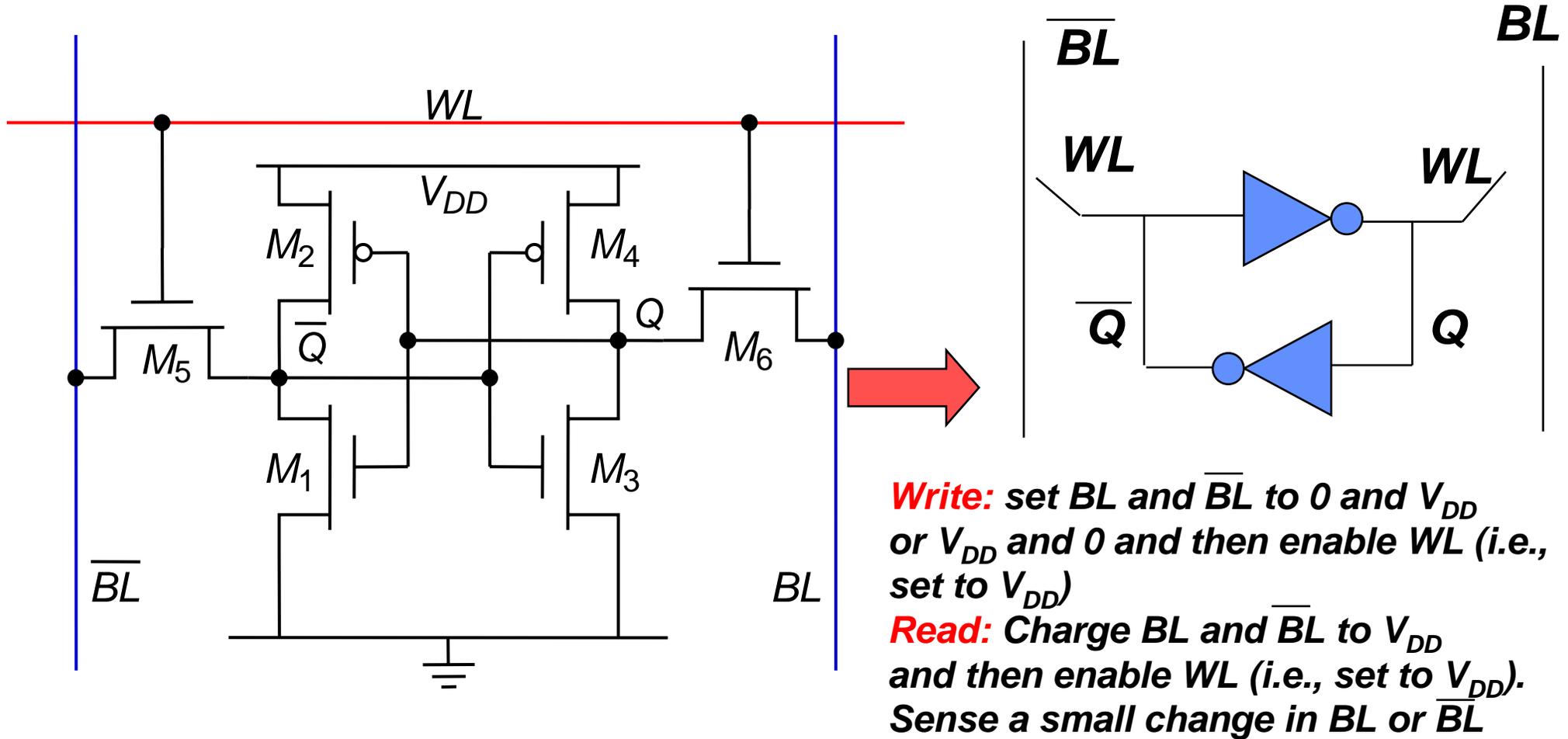


Register Memory

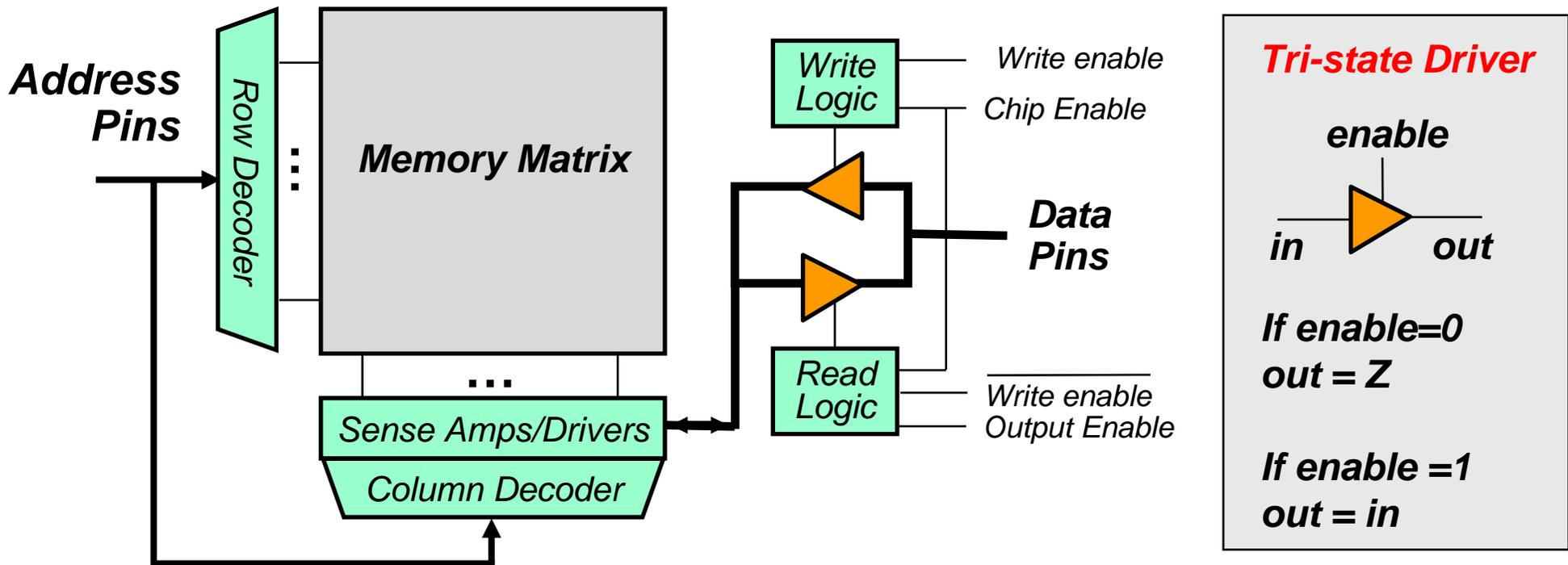


- *Works fine for small memory blocks (e.g., small register files)*
- *Inefficient in area for large memories – **density is the key metric in large memory circuits***

How do we minimize cell size?



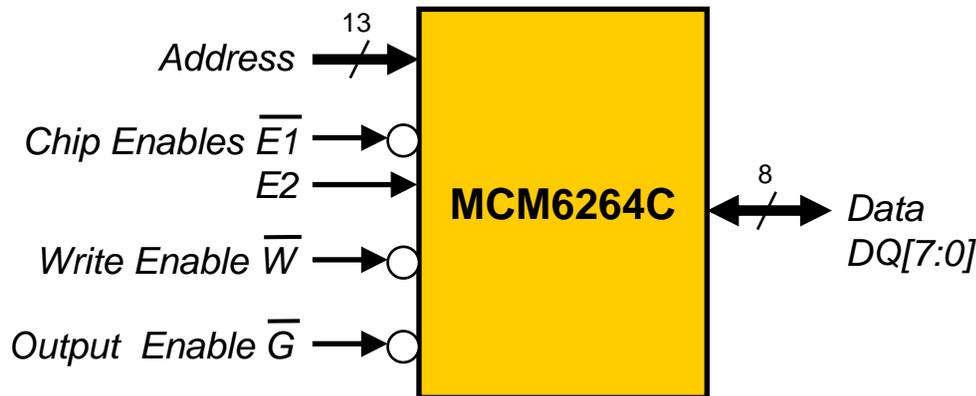
- **State held by cross-coupled inverters (M1-M4)**
- **Static Memory - retains state as long as power supply turned on**
- **Feedback must be overdriven to write into the memory**



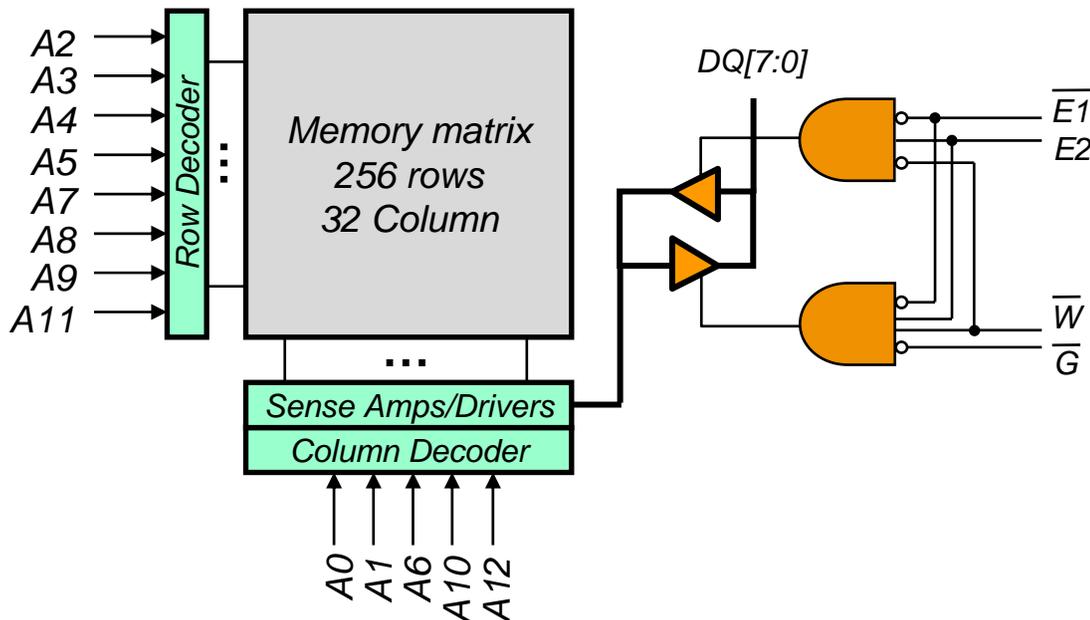
- **Address** pins drive row and column decoders
- **Data** pins are bidirectional and shared by reads and writes

- **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"

On the outside:



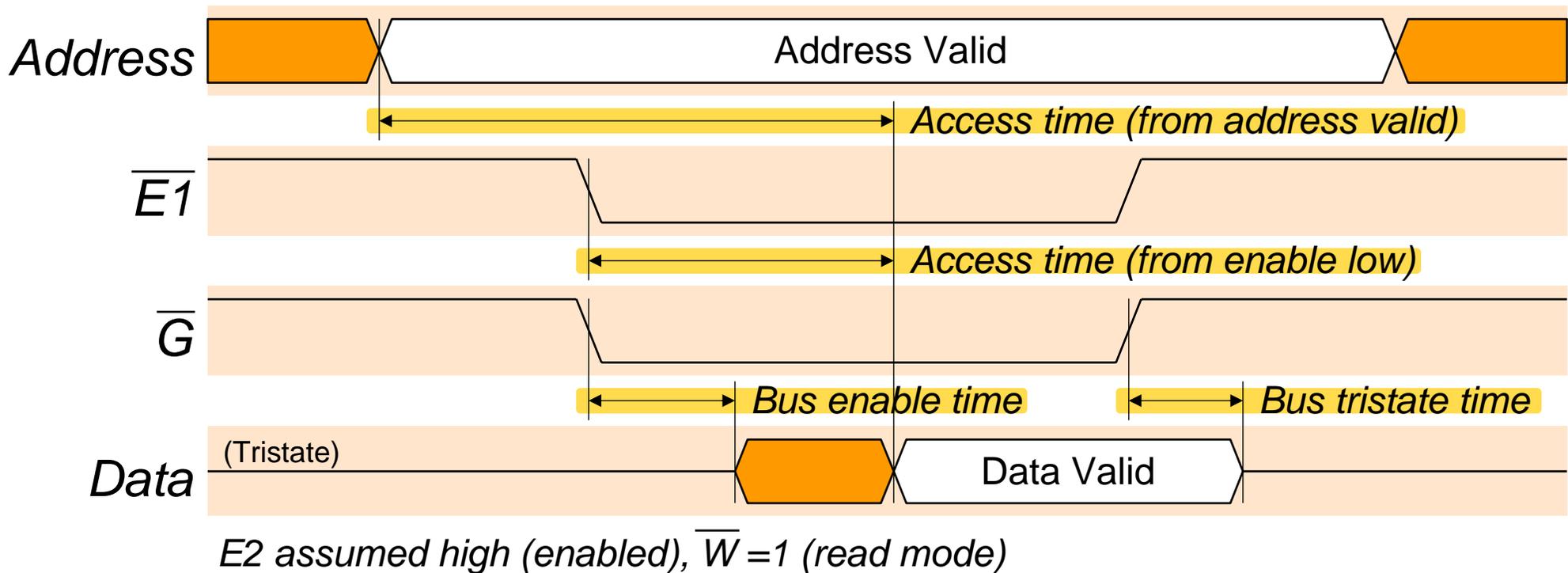
On the inside:



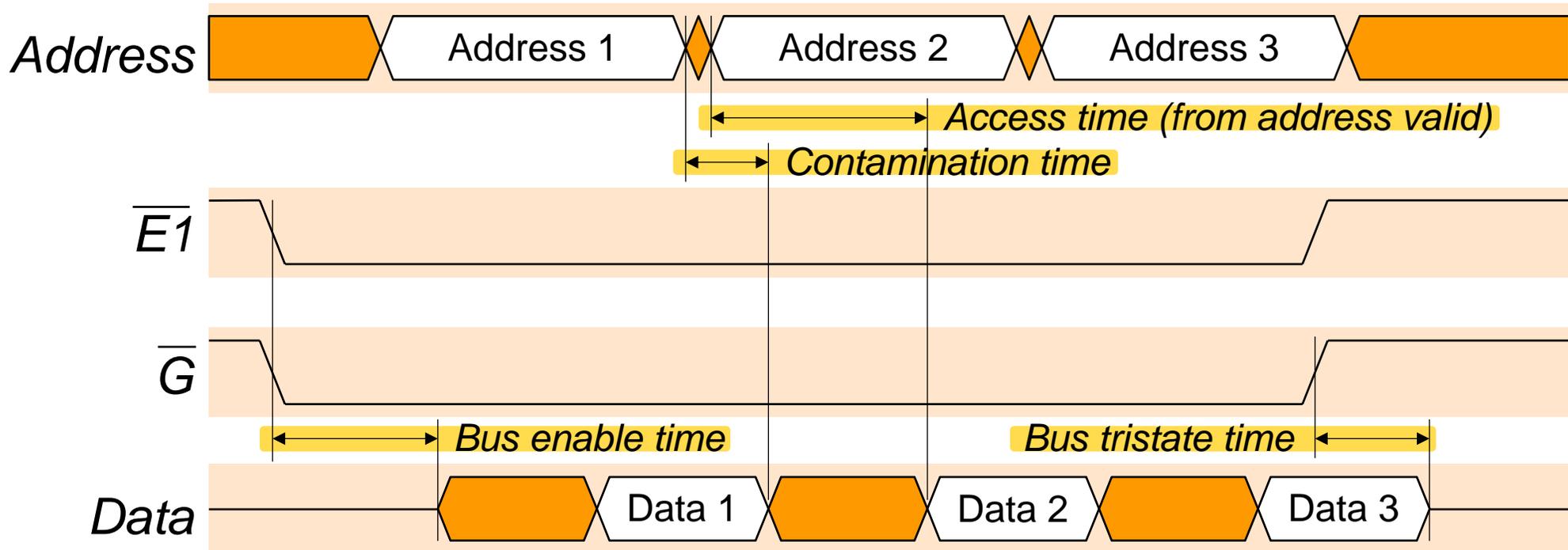
- Same (bidirectional) data bus used for reading and writing
- Chip Enables ($\overline{E1}$ and $E2$)
 - $E1$ must be low and $E2$ must be high to enable the chip
- Write Enable (\overline{W})
 - When low (and chip is enabled), the values on the data bus are written to the location selected by the address bus
- Output Enable (\overline{G})
 - When low (and chip is enabled with $W=0$), the data bus is driven with the value of the selected memory location

Pinout	NC	1	28	V_{CC}
	A12	2	27	\overline{W}
	A7	3	26	$E2$
	A6	4	25	A8
	A5	5	24	A9
	A4	6	23	A11
	A3	7	22	\overline{G}
	A2	8	21	A10
	A1	9	20	$\overline{E1}$
	A0	10	19	DQ7
	DQ0	11	18	DQ6
	DQ1	12	17	DQ5
	DQ2	13	16	DQ4
	V_{SS}	14	15	DQ3

(Image by MIT OCW.)

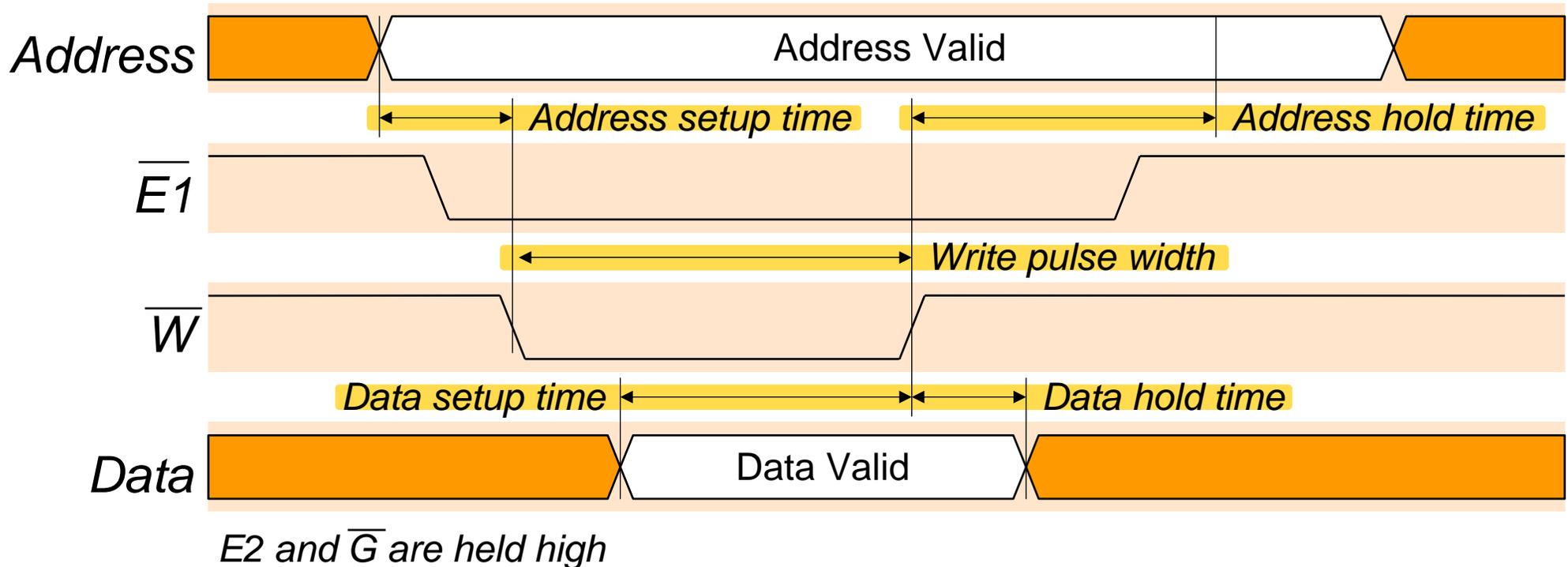


- Read cycle begins when all enable signals ($\overline{E1}$, $\overline{E2}$, \overline{G}) are active
- Data is valid after read access time
 - Access time is indicated by full part number: *MCM6264CP-12* → *12ns*
- Data bus is tristated shortly after \overline{G} or $\overline{E1}$ goes high

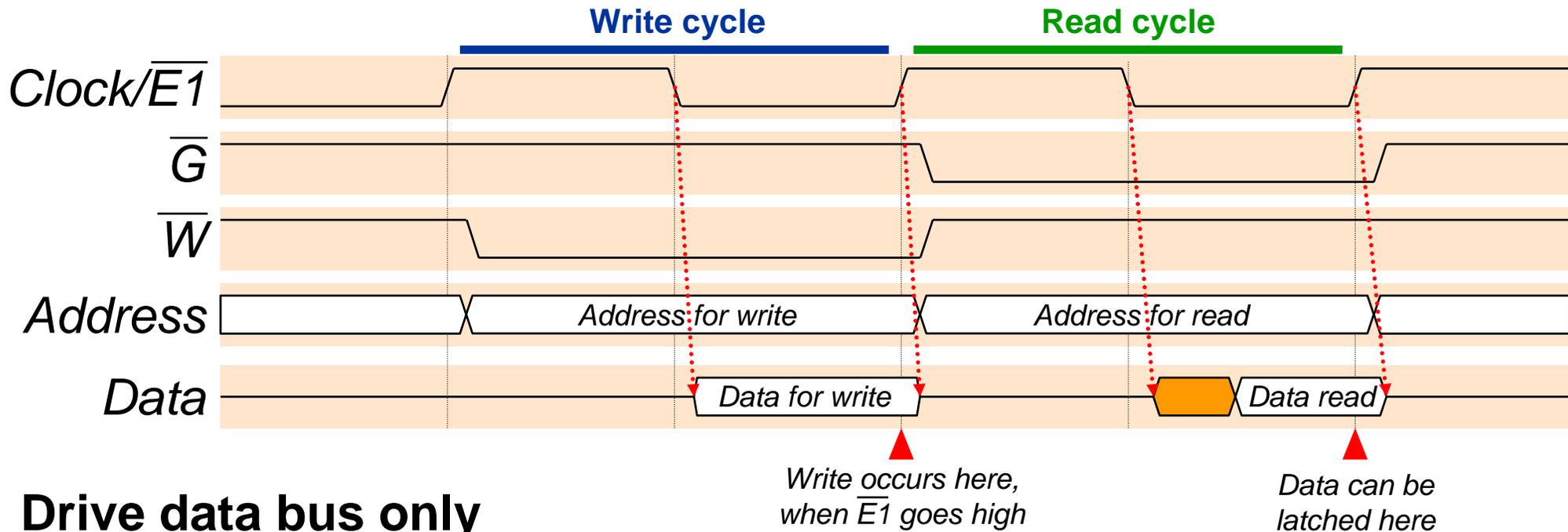


$E2$ assumed high (enabled), $\overline{W} = 1$ (read mode)

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

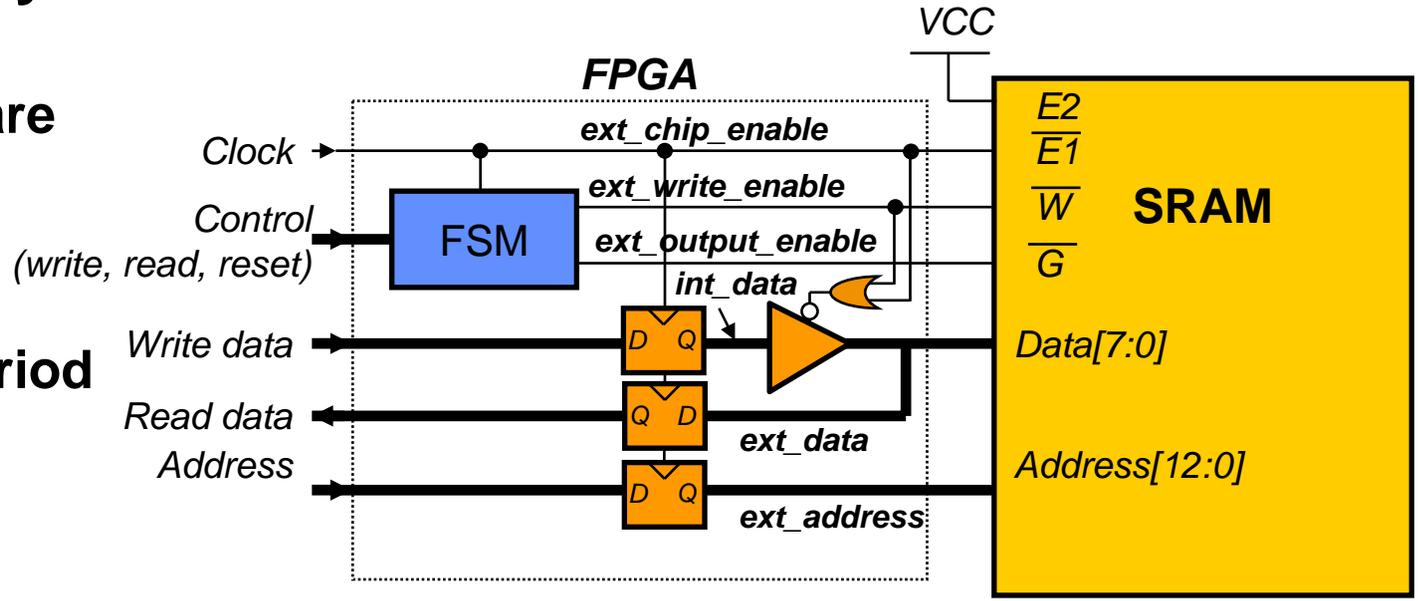


- Data latched when \overline{W} or $\overline{E1}$ goes high (or E2 goes low)
 - Data must be stable at this time
 - Address must be stable before \overline{W} goes low
- Write waveforms are more important than read waveforms
 - Glitches to address can cause writes to random addresses!

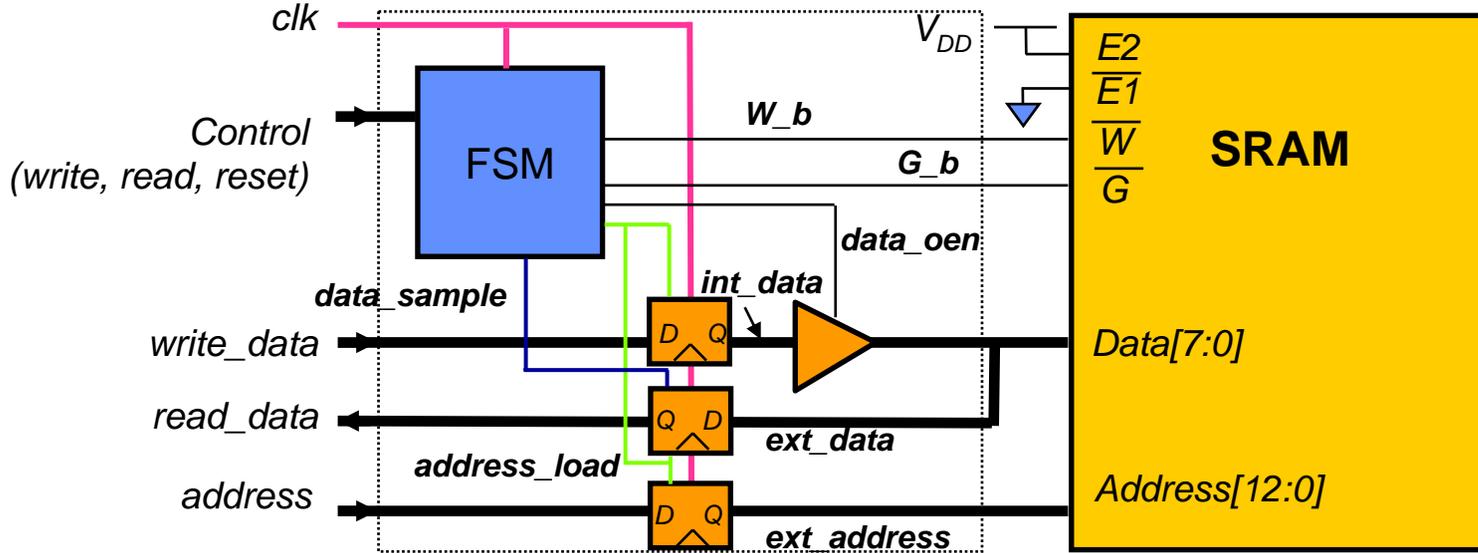


■ **Drive data bus only when clock is low**

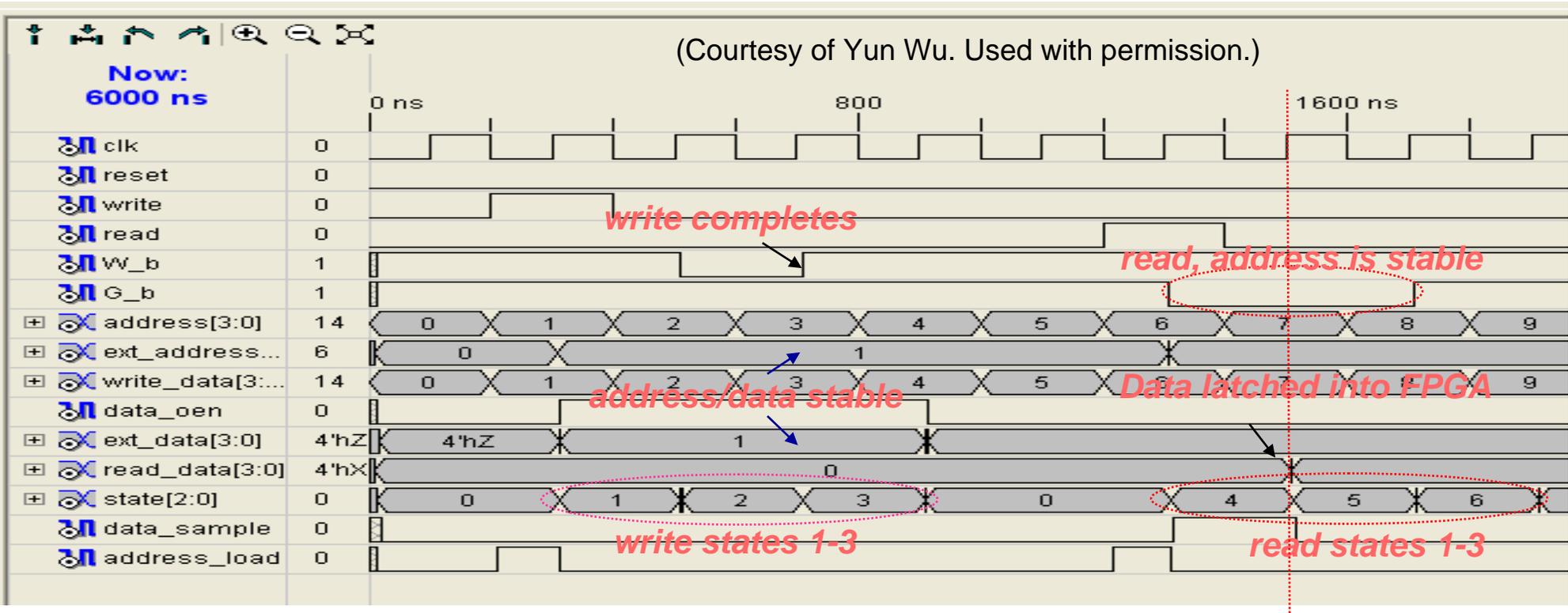
- Ensures address are stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time

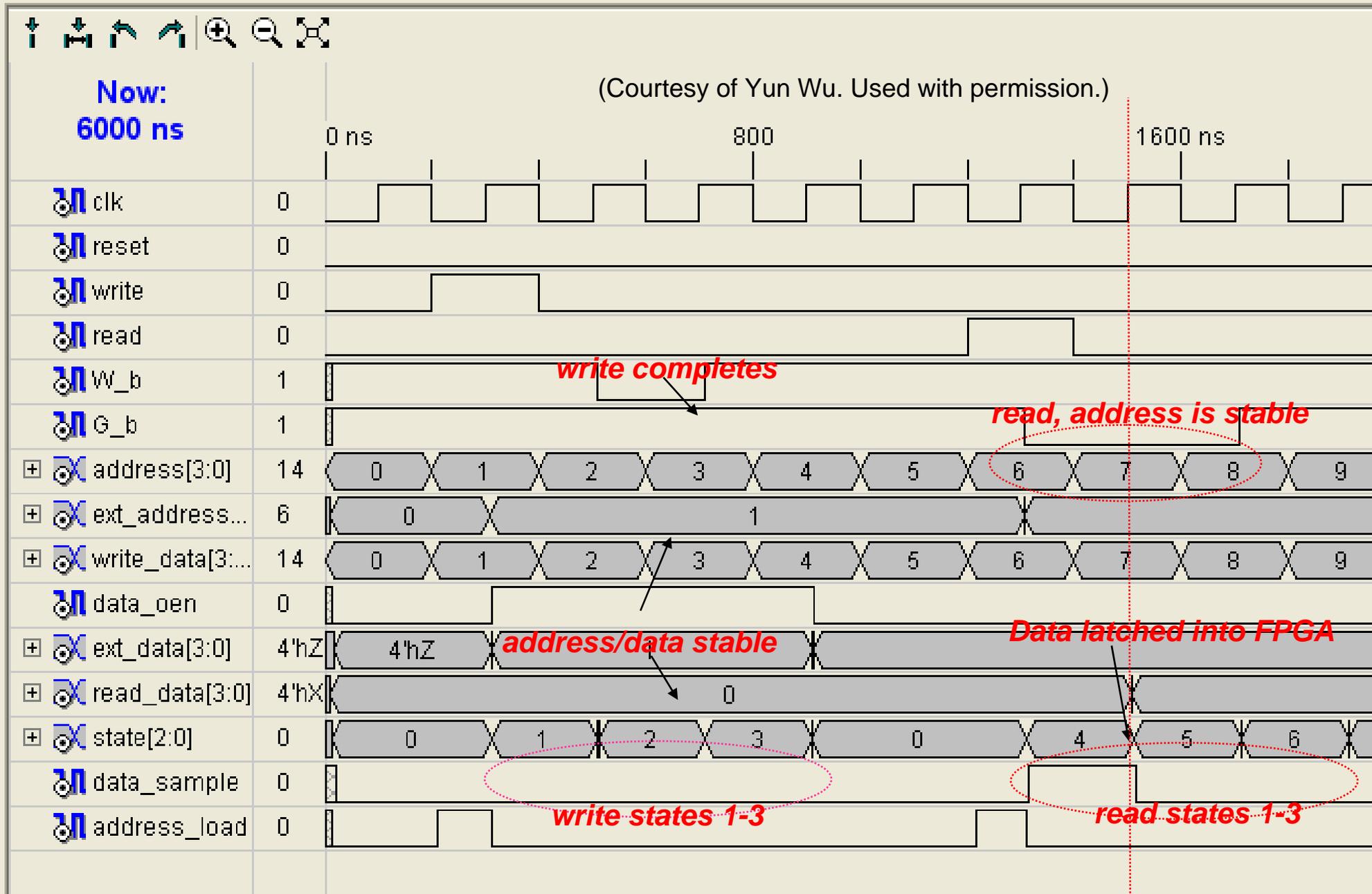


Multi-Cycle Read/Write (less aggressive, recommended timing)



(Courtesy of Yun Wu. Used with permission.)





```

module memtest (clk, reset, G_b, W_b, address,
  ext_address, write_data, read_data, ext_data, read,
  write, state, data_oen, address_load, data_sample);
  input clk, reset, read, write;
  output G_b, W_b;
  output [12:0] ext_address;
  reg [12:0] ext_address;
  input [12:0] address;
  input [7:0] write_data;
  output [7:0] read_data;
  reg [7:0] read_data;
  inout [7:0] ext_data;
  reg [7:0] int_data;
  output [2:0] state;
  reg [2:0] state, next;
  output data_oen, address_load, data_sample;
  reg G_b, W_b, G_b_int, W_b_int, address_load,
  data_oen, data_oen_int, data_sample;

  wire [7:0] ext_data;
  parameter IDLE = 0;
  parameter write1 = 1;
  parameter write2 = 2;
  parameter write3 = 3;
  parameter read1 = 4;
  parameter read2 = 5;
  parameter read3 = 6;

```

1/4

```

  assign ext_data = data_oen ? int_data : 8'hz;

```

```

  // Sequential always block for state assignment

```

```

  always @ (posedge clk)

```

```

    begin

```

```

      if (!reset) state <= IDLE;

```

```

      else state <= next;

```

```

      G_b <= G_b_int;

```

```

      W_b <= W_b_int;

```

```

      data_oen <= data_oen_int;

```

```

      if (address_load) ext_address <= address;

```

```

      if (data_sample) read_data <= ext_data;

```

```

      if (address_load) int_data <= write_data;

```

```

    end

```

```

  // note that address_load and data_sample are not

```

```

  // registered signals

```

2/4

```
// Combinational always block for next-state
```

```
// computation
```

```
always @ (state or read or write) begin
```

```
    W_b_int = 1;
```

```
    G_b_int = 1;
```

```
    address_load = 0;
```

```
    data_oen_int = 0;
```

```
    data_sample = 0;
```

```
    case (state)
```

```
        IDLE:    if (write) begin
```

```
            next = write1;
```

```
            address_load = 1;
```

```
            data_oen_int = 1;
```

```
        end
```

```
        else if (read) begin
```

```
            next = read1;
```

```
            address_load = 1;
```

```
            G_b_int = 0;
```

```
        end
```

```
        else next = IDLE;
```

```
    write1: begin
```

```
        next = write2;
```

```
        W_b_int = 0;
```

```
        data_oen_int = 1;
```

```
    end
```

**Setup the
Default values**

3/4

```
    write2: begin
```

```
        next = write3;
```

```
        data_oen_int = 1;
```

```
    end
```

```
    write3: begin
```

```
        next = IDLE;
```

```
        data_oen_int = 0;
```

```
    end
```

```
    read1: begin
```

```
        next = read2;
```

```
        G_b_int = 0;
```

```
        data_sample = 1;
```

```
    end
```

```
    read2: begin
```

```
        next = read3;
```

```
    end
```

```
    read3: begin
```

```
        next = IDLE;
```

```
    end
```

```
        default: next = IDLE;
```

```
    endcase
```

```
end
```

```
endmodule
```

4/4

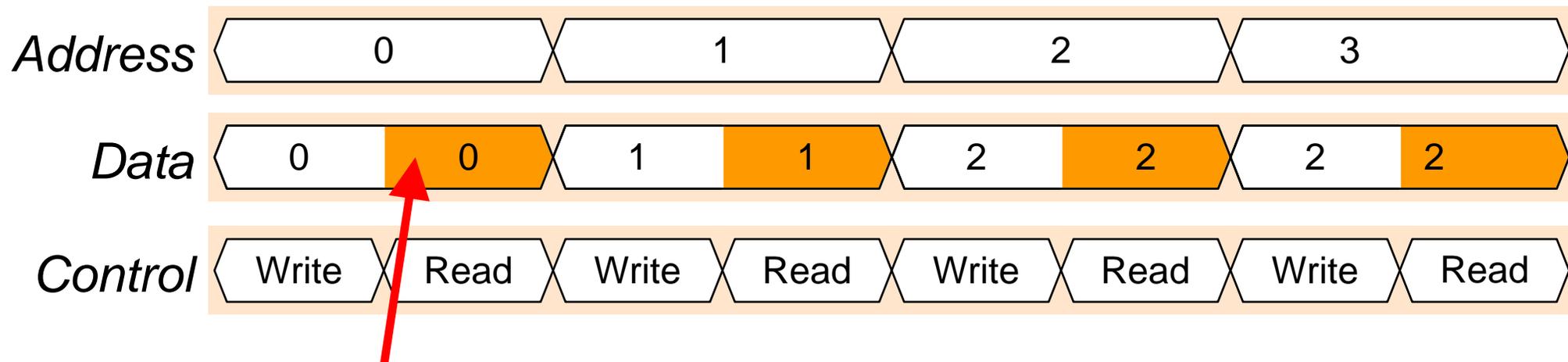
- **Common device problems**
 - **Bad locations: rare for individual locations to be bad**
 - **Slow (out-of-spec) timing(s): access incorrect data or violates setup/hold**
 - **Catastrophic device failure: e.g., ESD**
 - **Missing wire-bonds/devices (!): possible with automated assembly**
 - **Transient Failures: Alpha particles, power supply glitch**
- **Common board problems**
 - **Stuck-at-Faults: a pin shorted to V_{DD} or GND**
 - **Open Circuit Fault: connections unintentionally left out**
 - **Open or shorted address wires: causes data to be written to incorrect locations**
 - **Open or shorted control wires: generally renders memory completely inoperable**
- **Approach**
 - **Device problems generally affect the entire chip, almost any test will detect them**
 - **Writing (and reading back) many different data patterns can detect data bus problems**
 - **Writing unique data to every location and then reading it back can detect address bus problems**

- **An idea that almost works**

1. Write 0 to location 0
2. Read location 0, compare value read with 0
3. Write 1 to location 1
4. Read location 1, compare value read with 1
5. ...

- **What is the problem?**

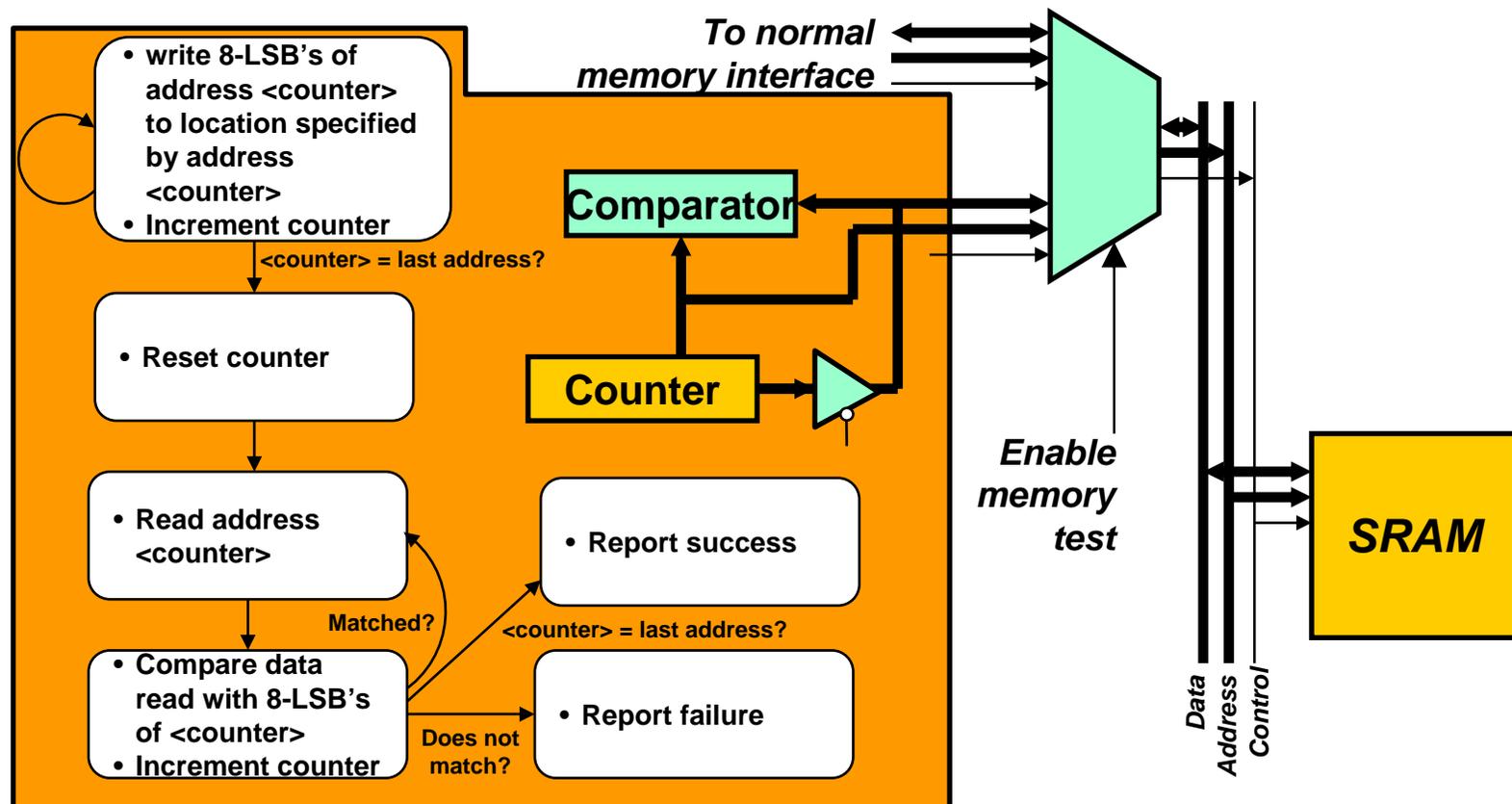
- Suppose the memory was missing (or output enable was disconnected)



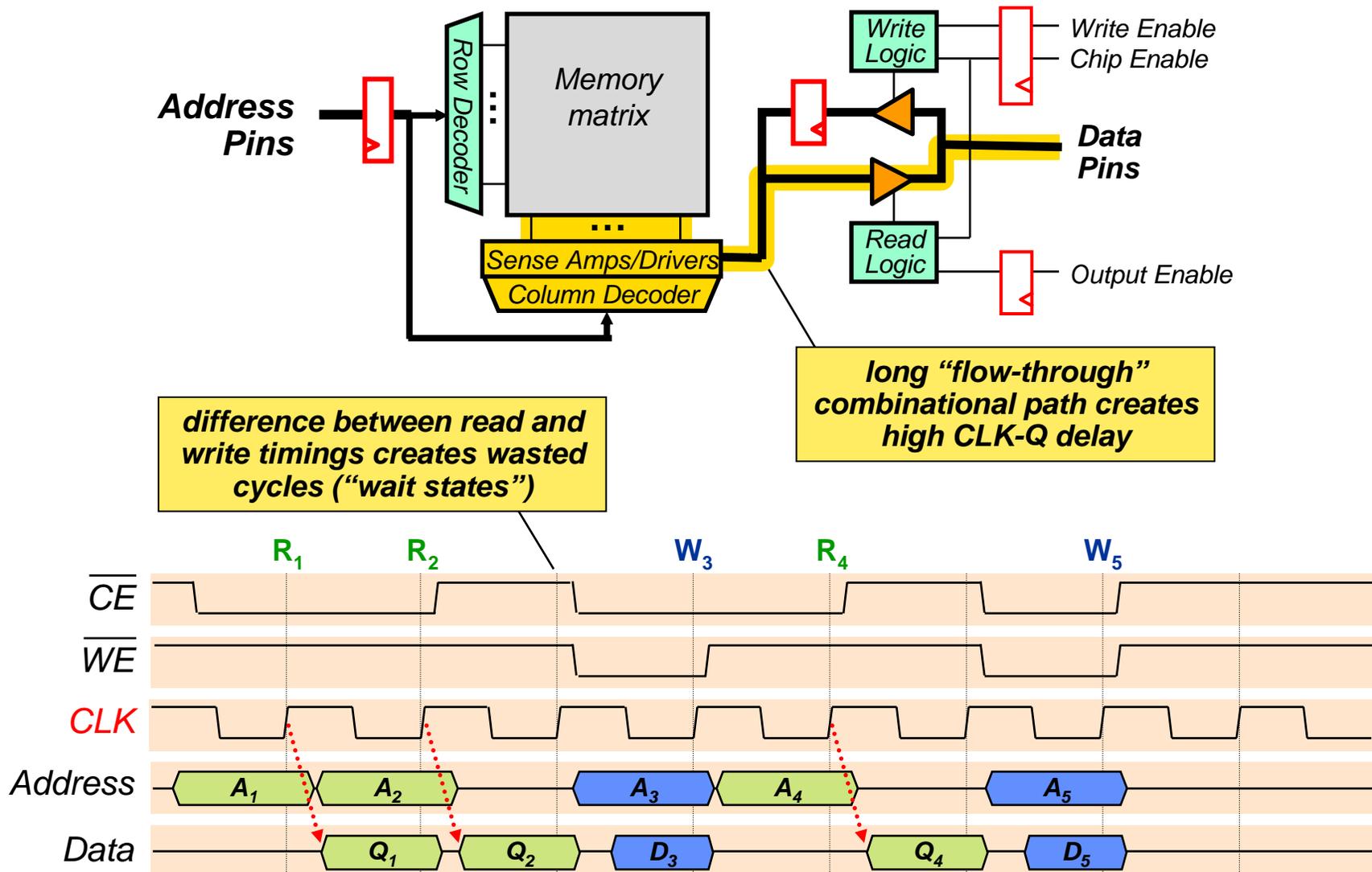
Data bus is undriven but wire capacitance briefly maintains the bus state: memory appears to be ok!

- Write to all locations, then read back all locations
 - Separates read/write to the same location with reads/writes of different data to different locations
 - (both data and address busses are changed between read and write to same location)

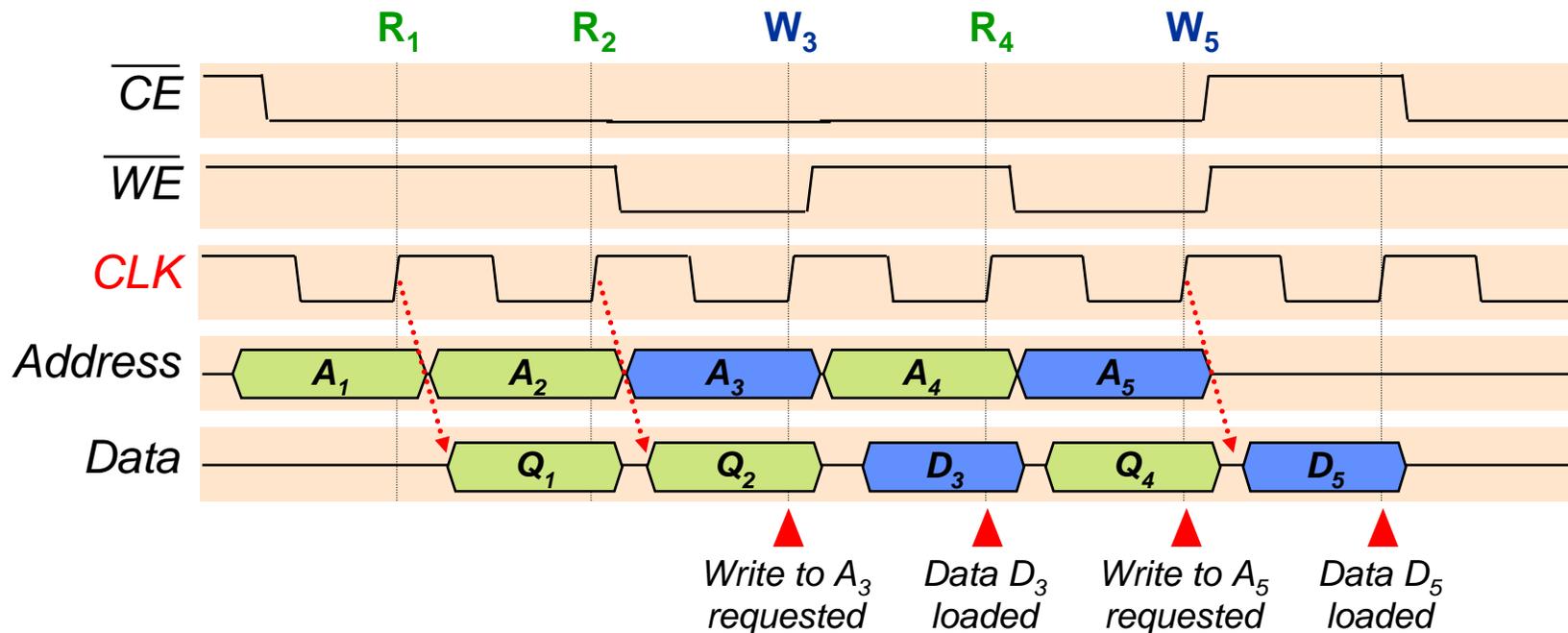
- Write 0 to address 0
- Write 1 to address 1
- ...
- Write $(n \bmod 256)$ to address n
- Read address 0, compare with 0
- Read address 1, compare with 1
- ...
- Read address n , compare with $(n \bmod 256)$



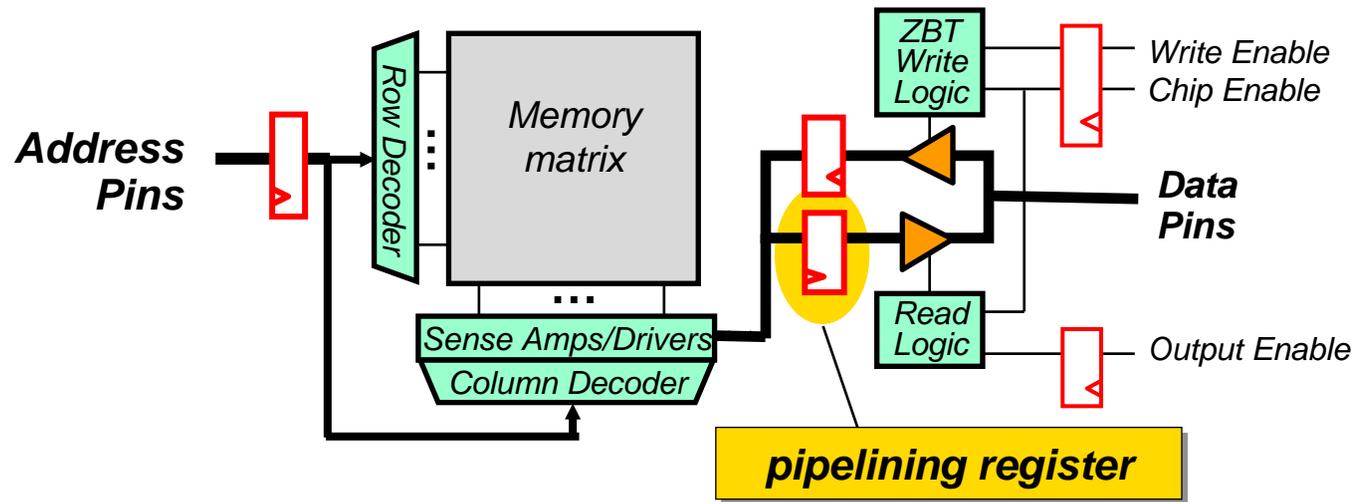
- **Clocking** provides input synchronization and encourages more reliable operation at high speeds



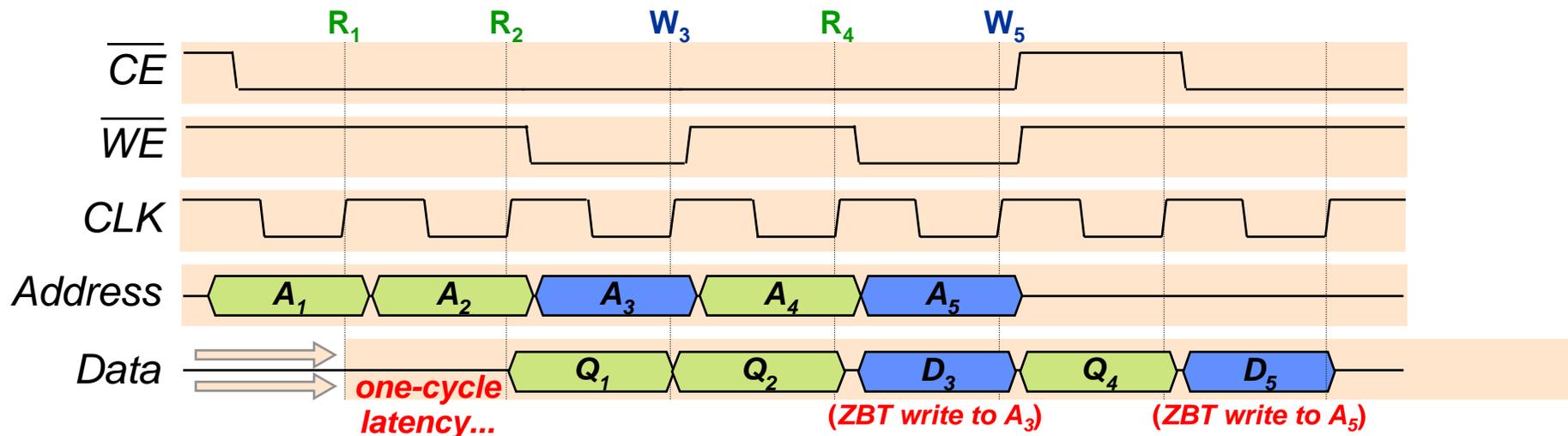
- The wait state occurs because:
 - On a read, data is available *after* the clock edge
 - On a write, data is set up *before* the clock edge
- ZBT (“zero bus turnaround”) memories **change the rules for writes**
 - On a write, data is set up **after** the clock edge (so that it is read on the following edge)
 - Result: no wait states, higher memory throughput



- 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



EPROM Cell – The Floating Gate Transistor

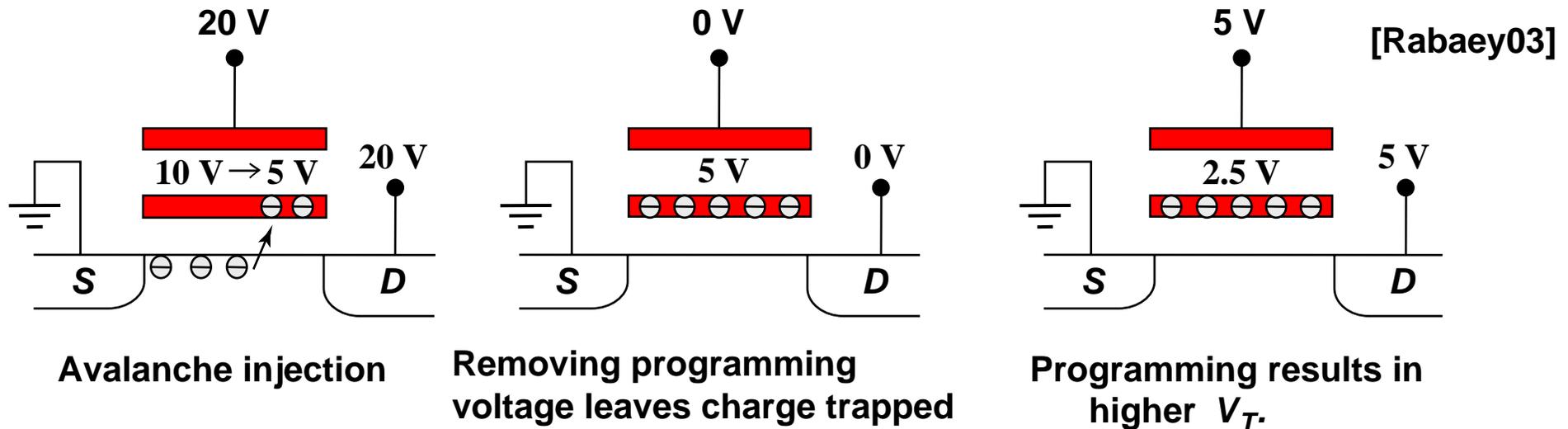


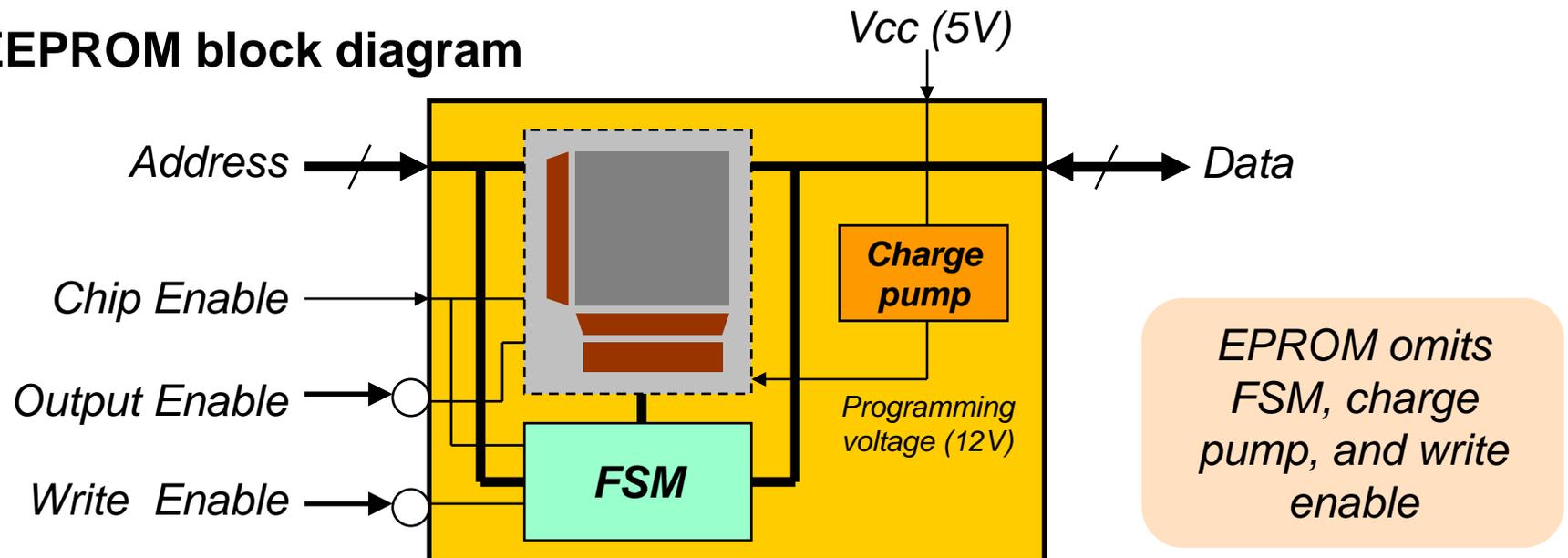
Image removed due to copyright restrictions.

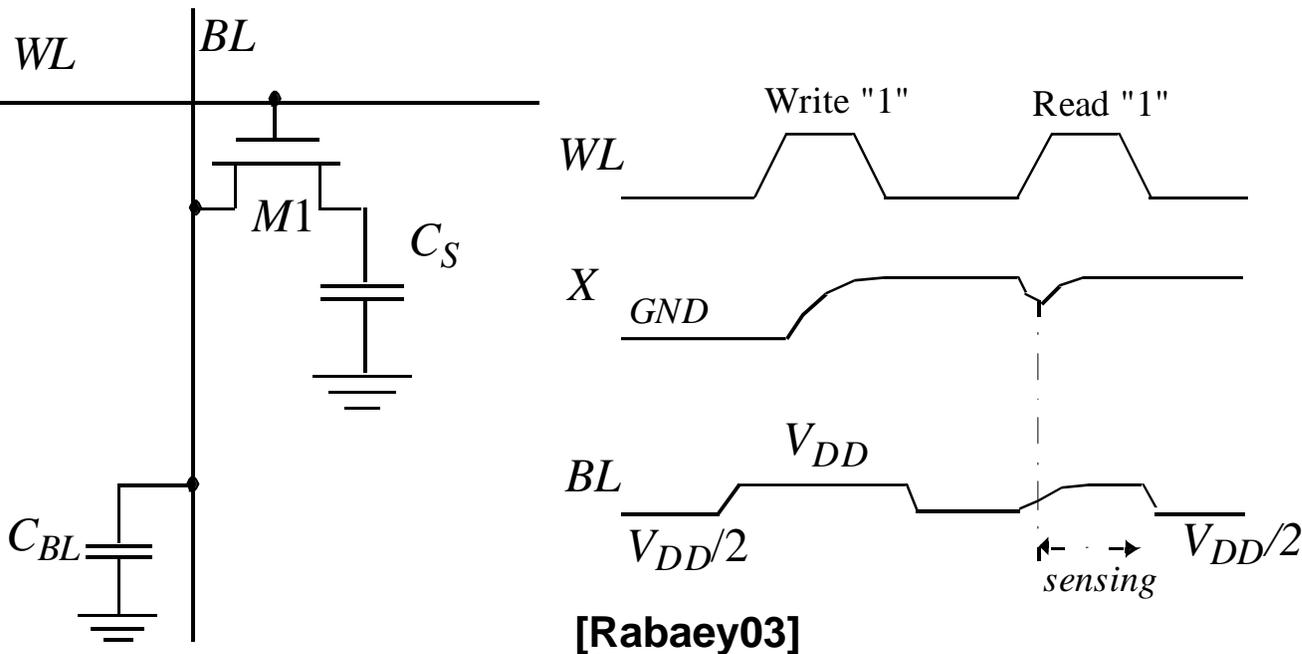
EPROM Cell

This is a non-volatile memory (retains state when supply turned off)

- 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
 - 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





DRAM uses Special Capacitor Structures

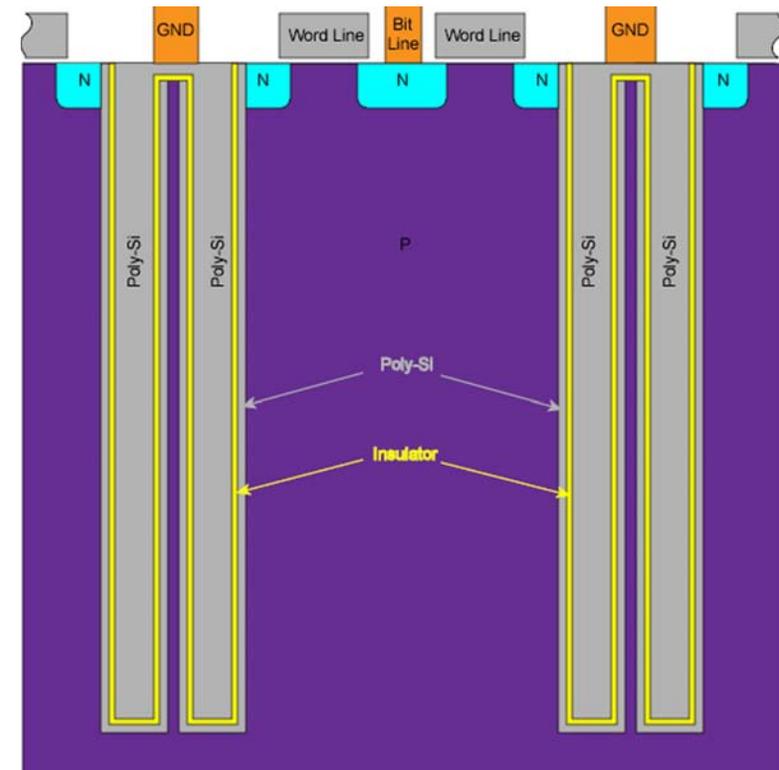
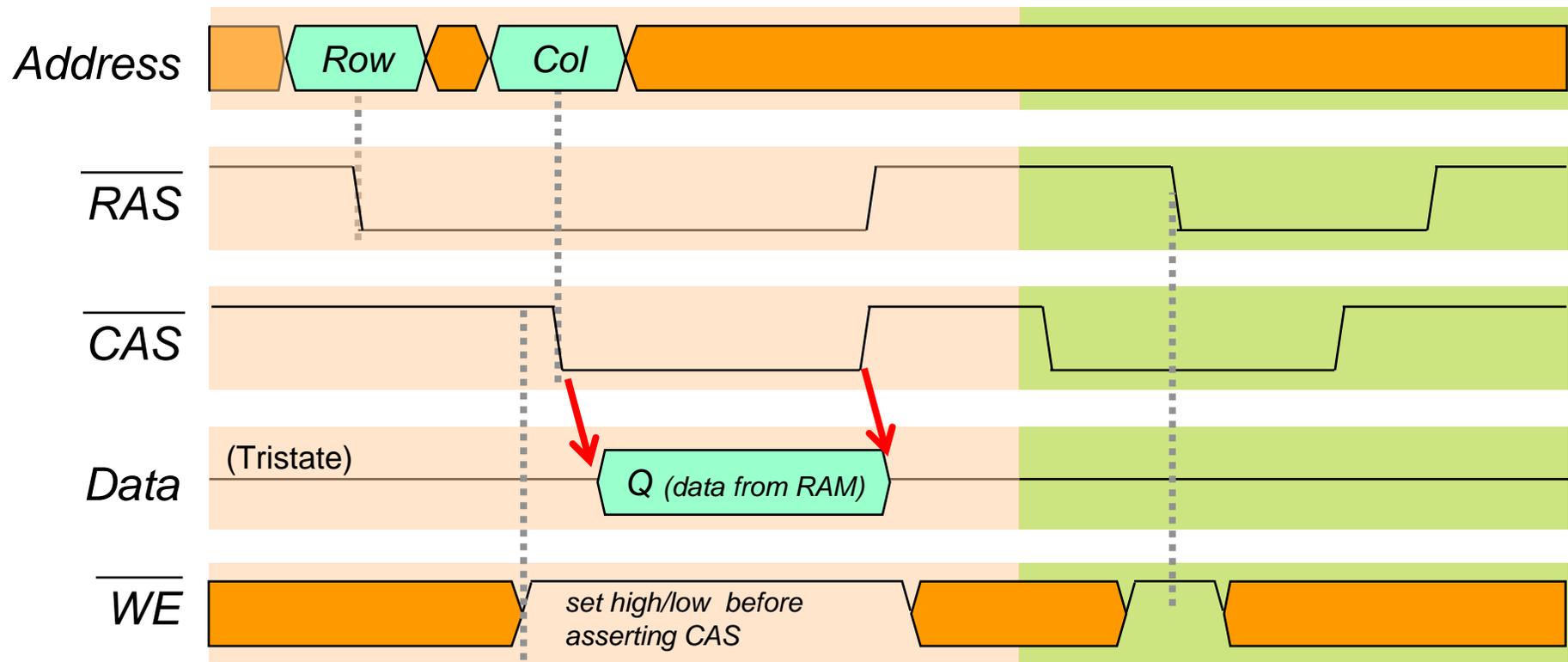


Image by Wikipedia user Cyferz.

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

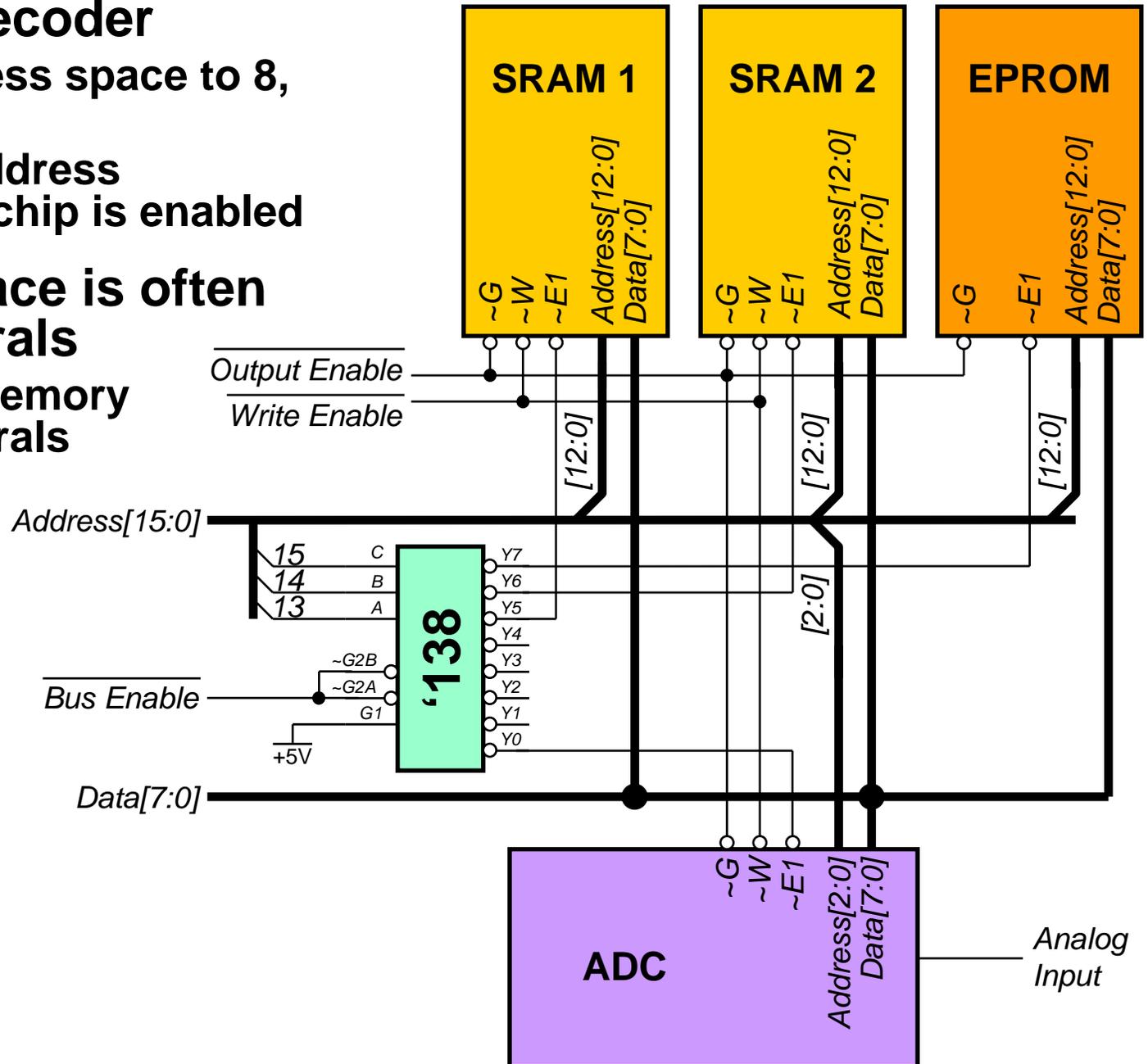
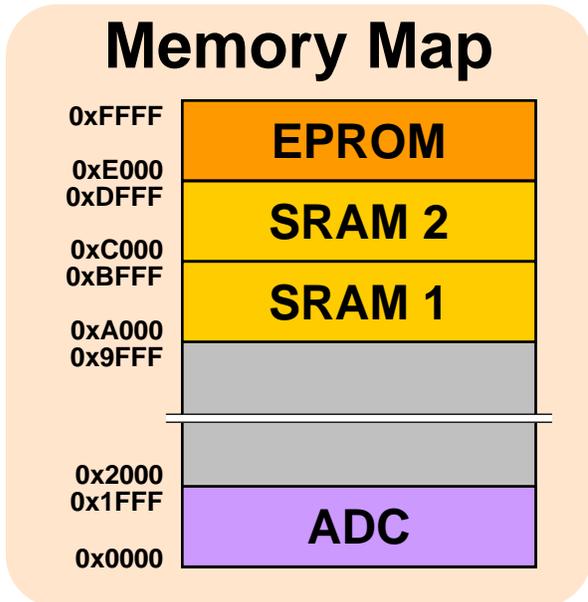


**RAS-before-CAS
for a read or write**
(Row and column addresses taken on falling edges of RAS and CAS)

**CAS-before-RAS
for a refresh**

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

- '138 is a 3-to-8 decoder
 - Maps 16-bit address space to 8, 13-bit segments
 - Upper 3-bits of address determine which chip is enabled
- SRAM-like interface is often used for peripherals
 - Referred to as “memory mapped” peripherals



■ 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.
- 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

■ 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

- 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 does
- 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