



L7: Memory Basics and Timing



Acknowledgement: Nathan Ickes

Some slides are derived from slides used in past terms of 6.111

L7 6.111 Fall 2003 – Introductory Digital Systems Laboratory



Memory Classification & Metrics

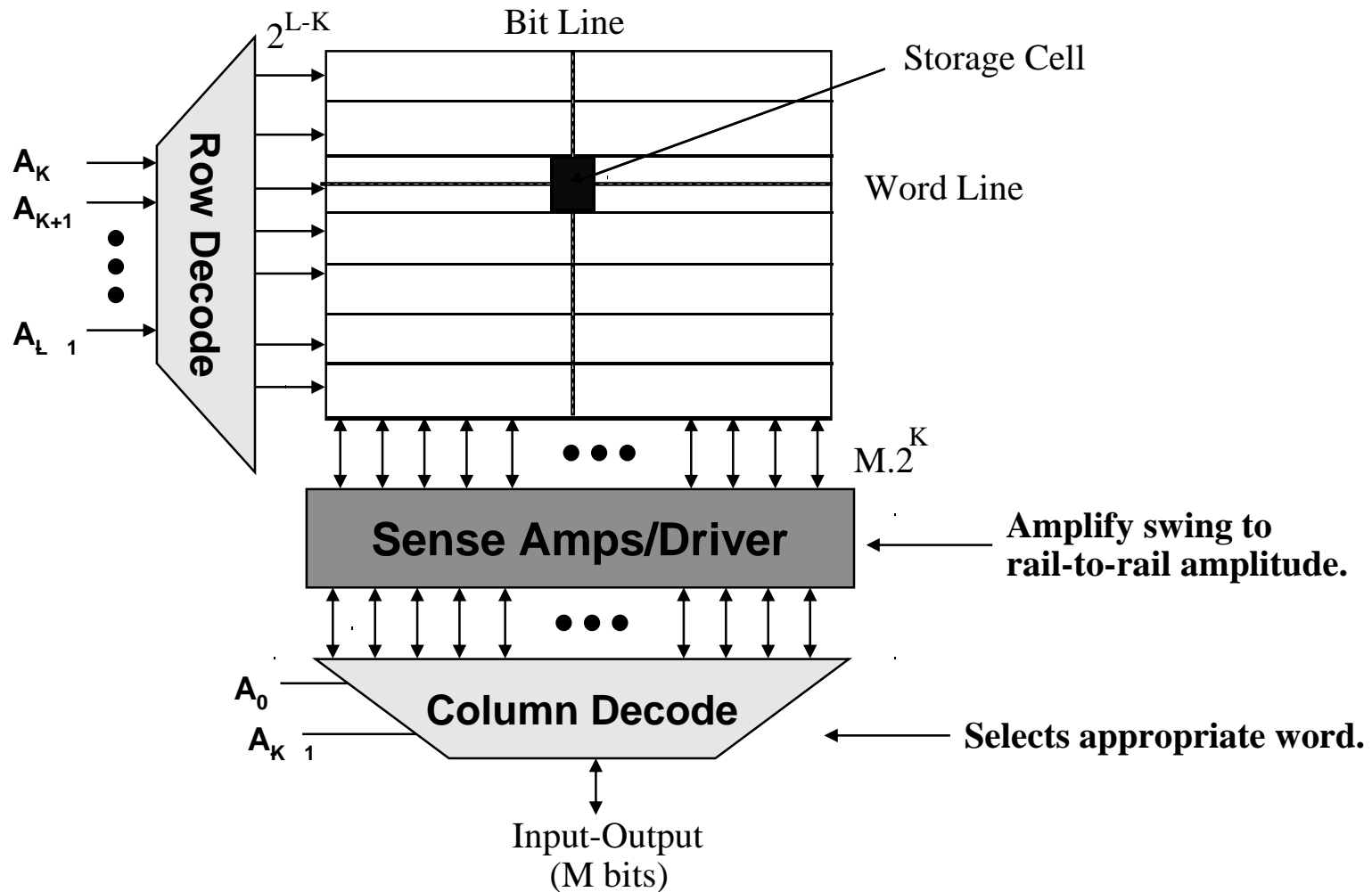


Read Write Memory		Non Volatile Read Write Memory	Read Only Memory
Random Access	Non Random Access	EPROM E ² PROM FLASH	Mask Programmed
SRAM DRAM	FIFO LIFO		

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

Memory Array Architecture

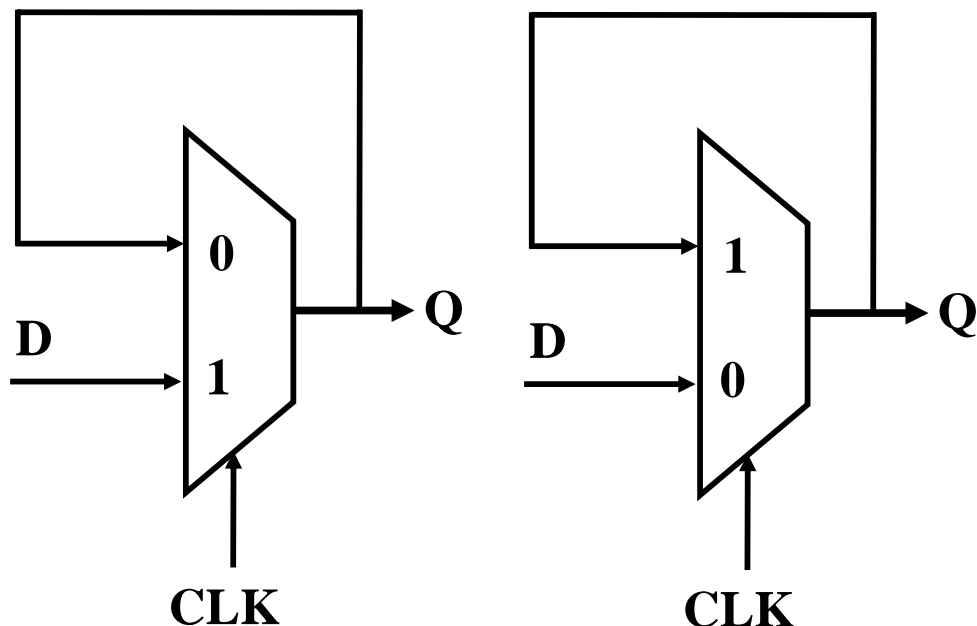




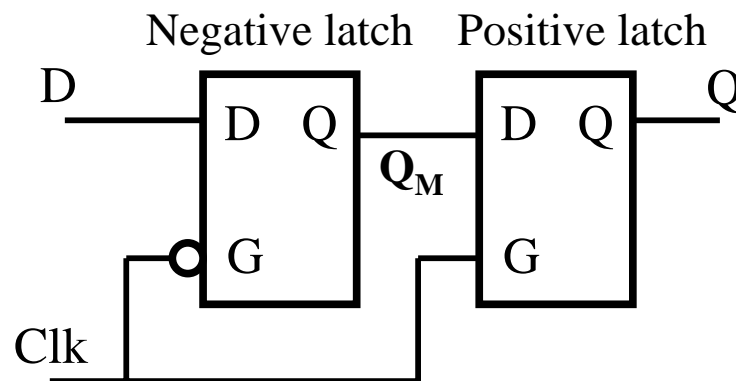
Latch and Register Based Memory



Positive Latch Negative Latch



Register Memory

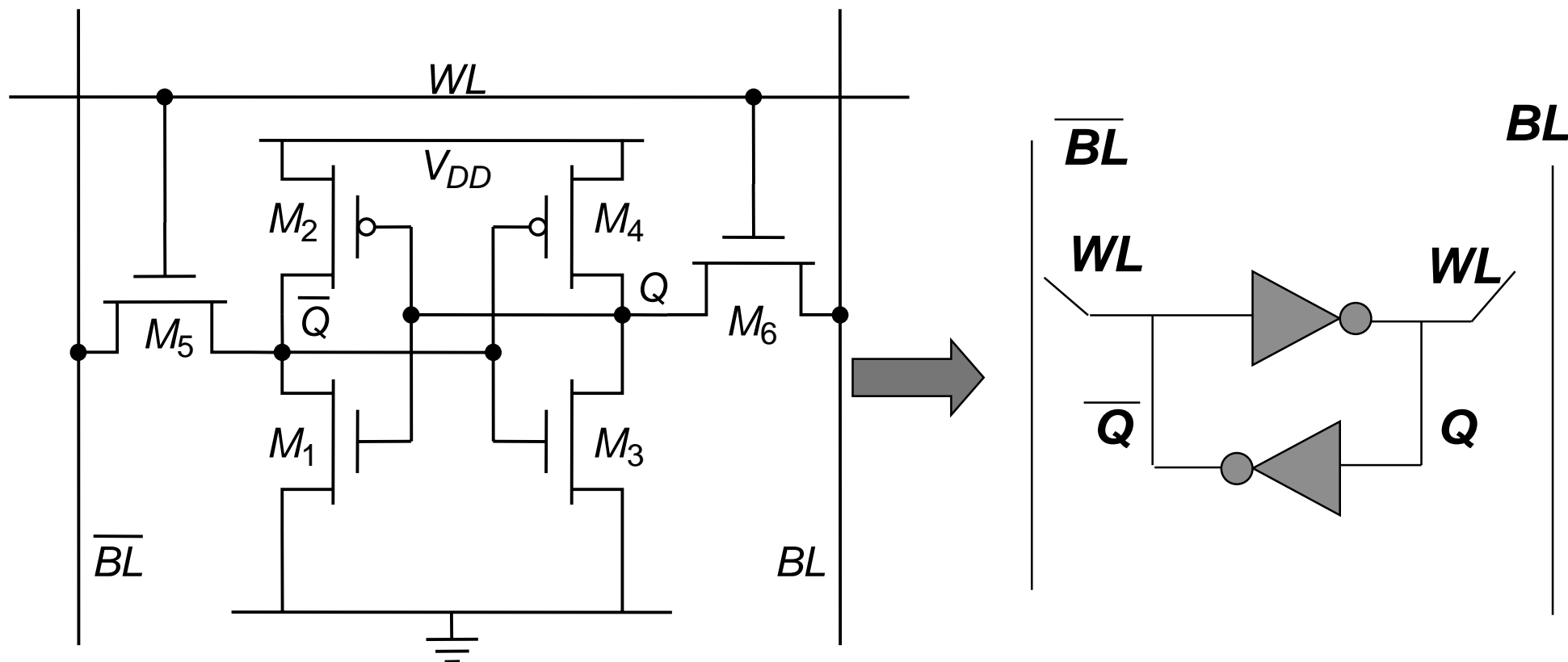


- *Works fine for small memory blocks (e.g., small register files).*
- *Inefficient in area for large memories*
- *Density is the key metric in large memory circuits.*

How do we minimize cell size?

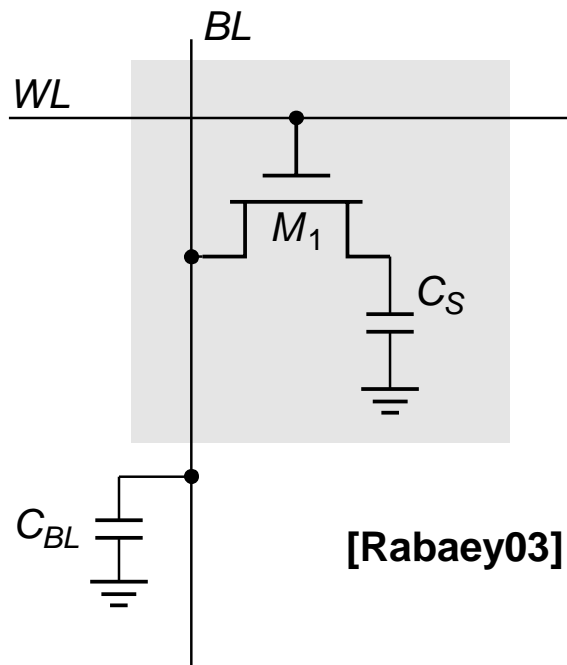


Static RAM (SRAM) Cell (The 6 T Cell)

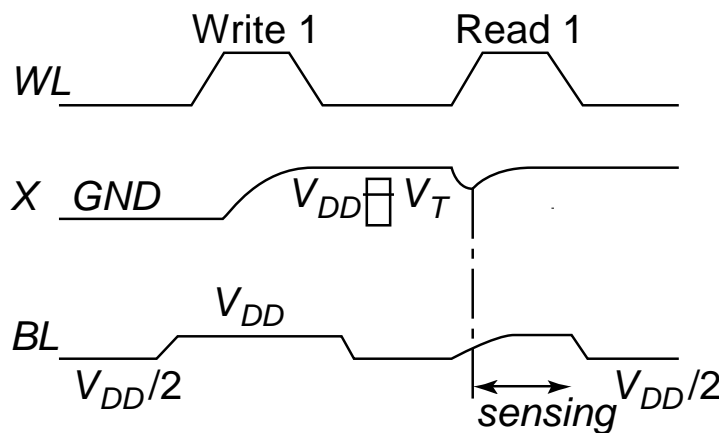


- State held by cross coupled inverters (M1 M4).
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory.

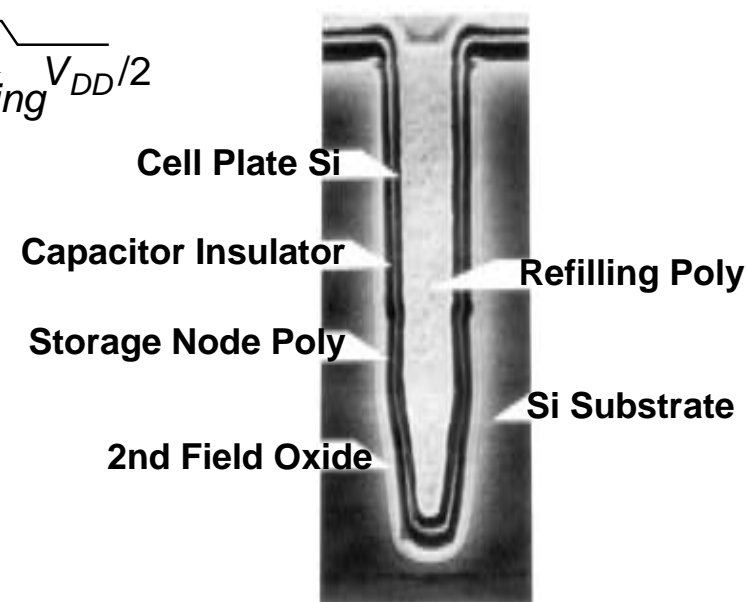
Dynamic RAM (DRAM) Cell



[Rabaey03]



DRAM uses Special Capacitor Structures.



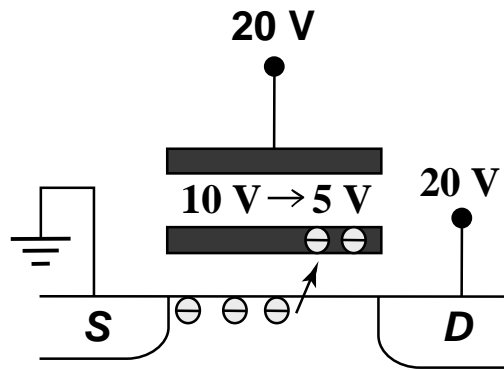
To Write: set Bit Line (BL) to 0 or V_{DD} & enable Word Line (WL) (i.e., set to V_{DD})

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

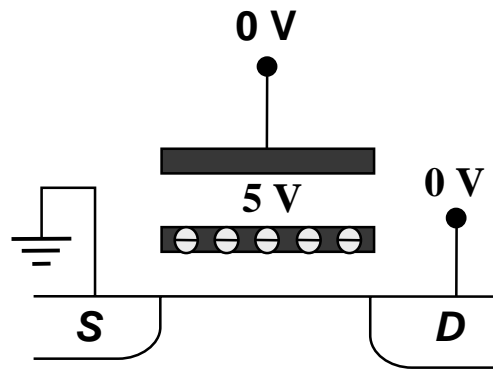
- DRAM relies on charge stored in a capacitor to hold state.
- Found in all high density memories (one bit/transistor)
- Must be “refreshed” or state will be lost – high overhead



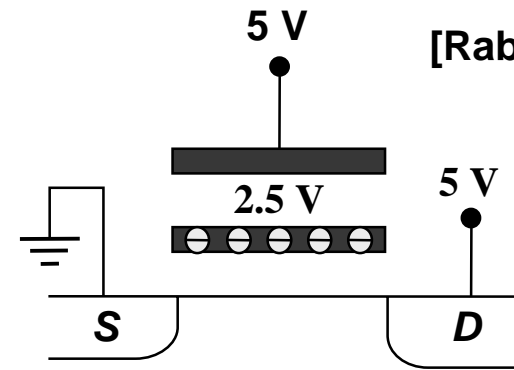
EPRM Cell – The Floating Gate Transistor



Avalanche injection



Removing programming voltage leaves charge trapped

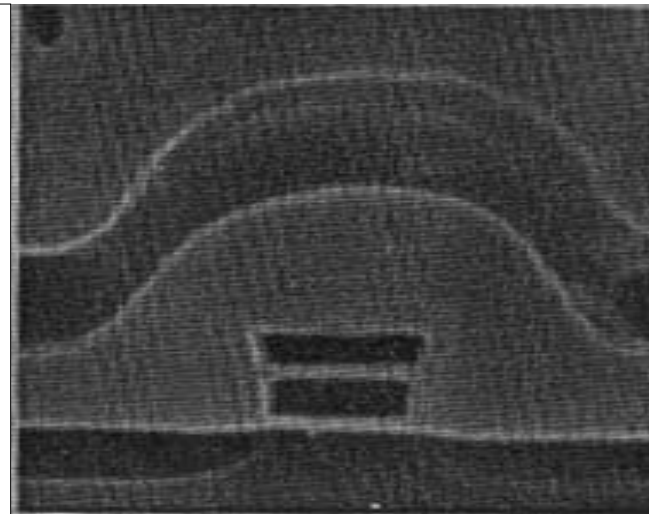


Programming results in higher V_T .

[Rabaey03]

EPROM Cell

Courtesy Intel



This non-volatile memory retains state when supply is turned off.

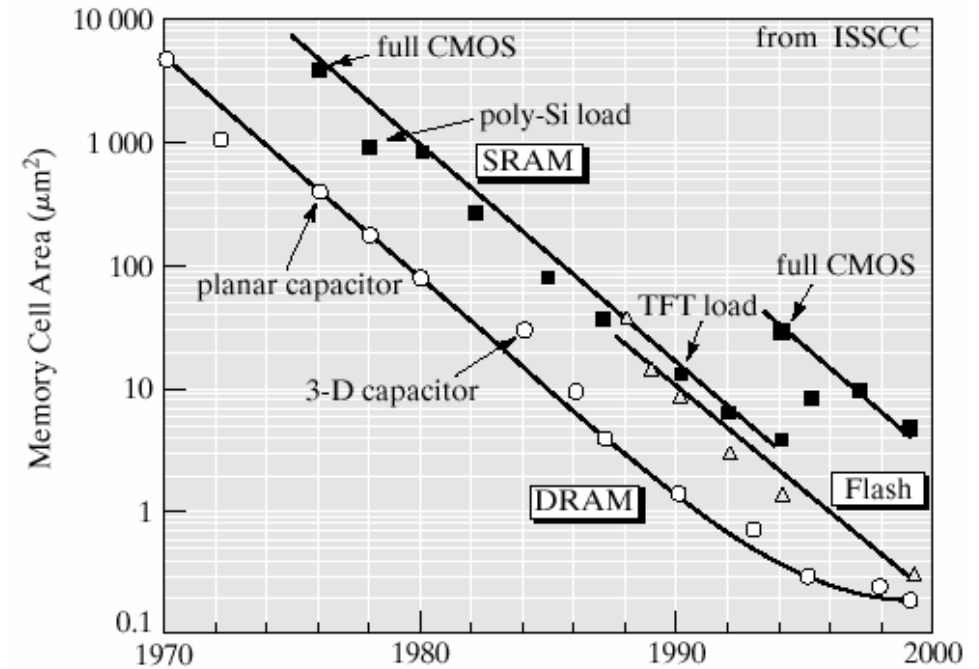
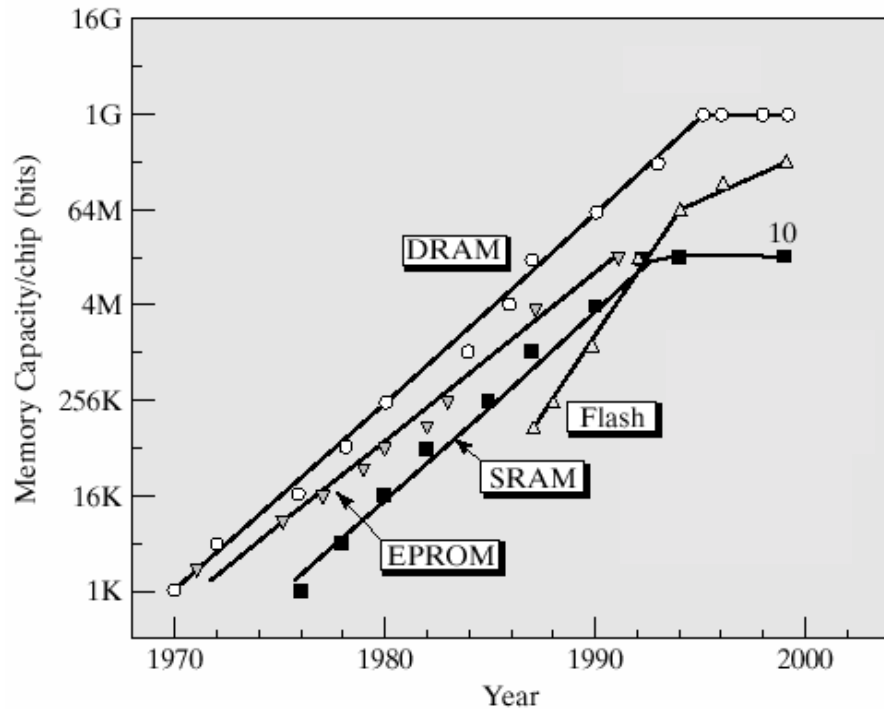


EEPROM / FLASH



- Like EPROM, both EEPROM and FLASH technology use floating gate transistors.
 - The primary difference is that EEPROM and FLASH allow electrical erasing while EPROM is UV erasable.
- EPROM and FLASH use FAMOS (Floating gate Avalanche MOS) storage transistors.
 - The floating gate of the FAMOS transistors gets its electrons by avalanche (hot electron) injection.
 - EPROM is erased by UV (~35 minutes).
 - FLASH is erased by tunneling.
- EEPROM uses FLOTOX (Floating gate Tunnel Oxide storage transistors).
 - The floating gate of an EEPROM gets its electrons by Fowler Nordheim tunneling.
- EEPROM and FLASH are erased by sections.
 - not individual bits so they are not like SRAMS

Density/Size Memory Trends



From [Itoh01]



Key Messages on Memory Devices



■ 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 PC board integration.

■ Device details

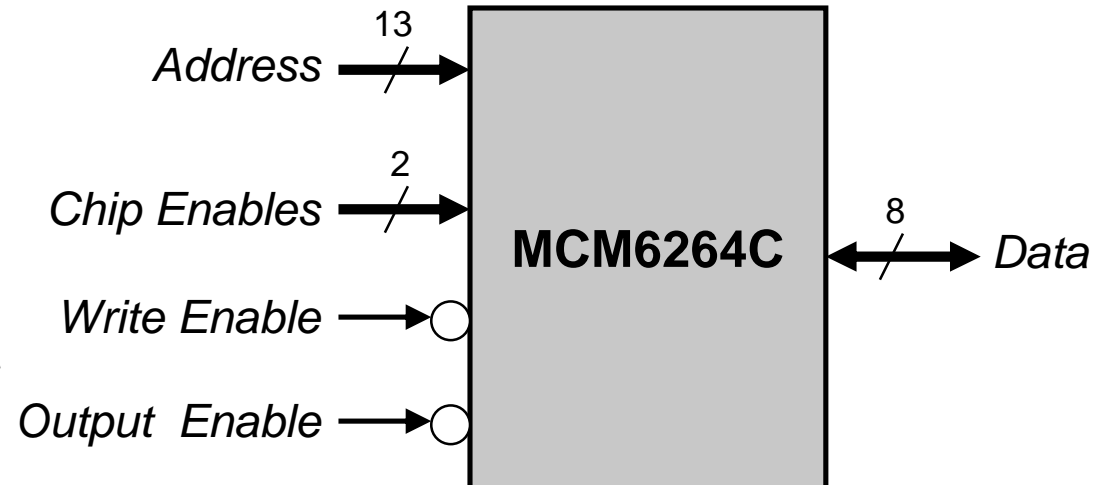
- Don't worry about them, wait until 6.012, 6.371 or 6.374.



MCM6264C 8k x 8 Static RAM



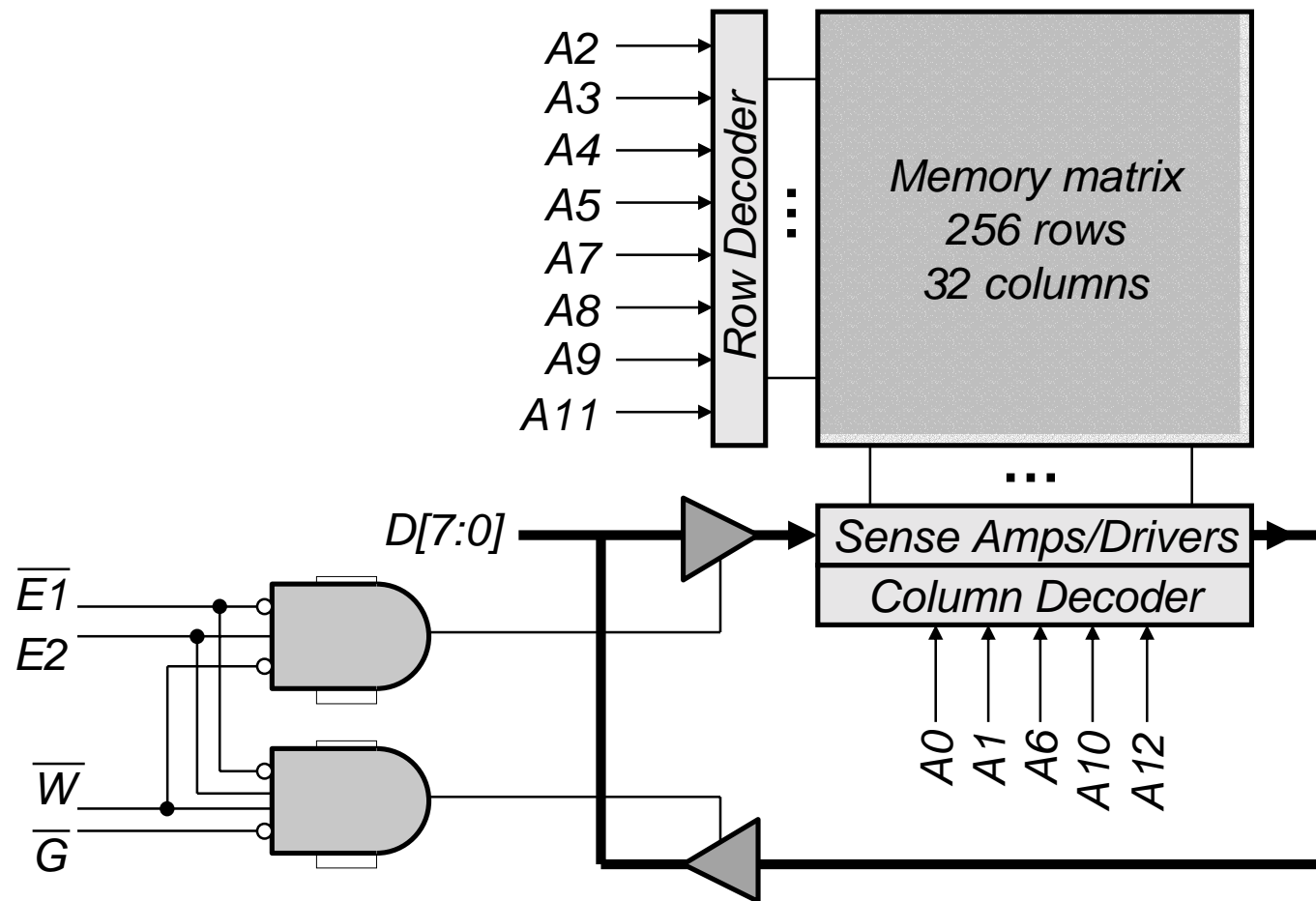
- Same (bidirectional) data bus used for reading and writing
- Chip Enables ($\overline{E1}$ and $E2$)
 - $\overline{E1}$ must be low and $E2$ must be high to enable the chip.
- Write Enable (\overline{W})
 - When low (and chip is enabled), the values on the data bus are written to the location selected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



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

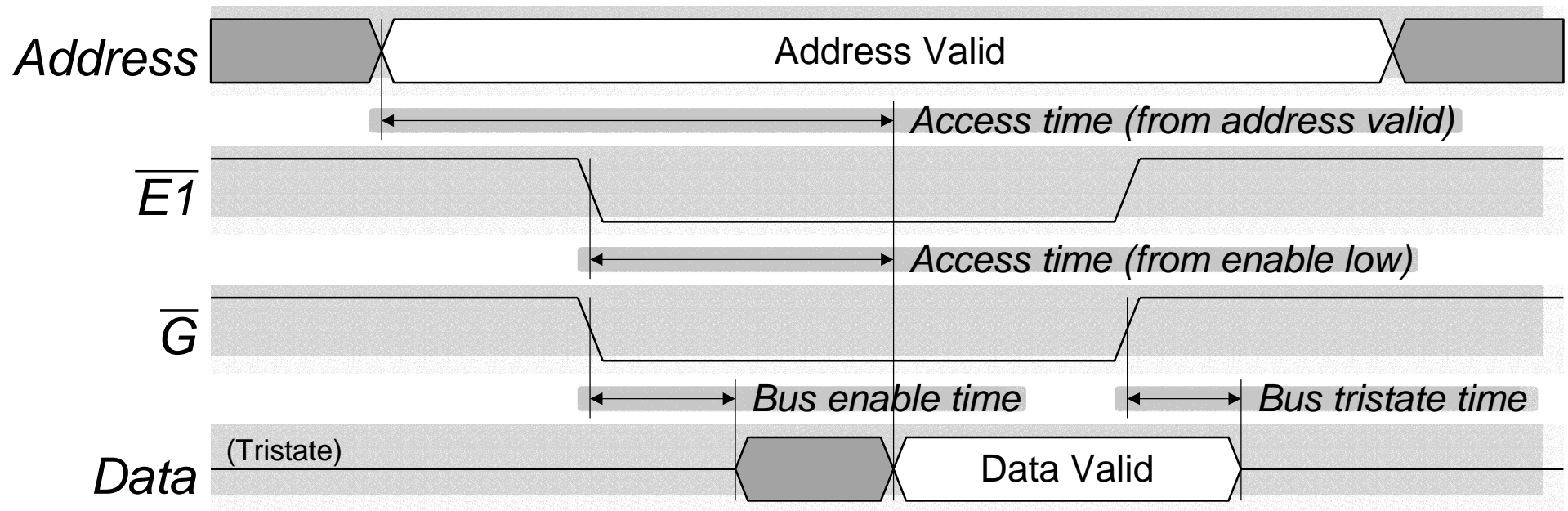


Inside the '6264C





Reading From SRAM

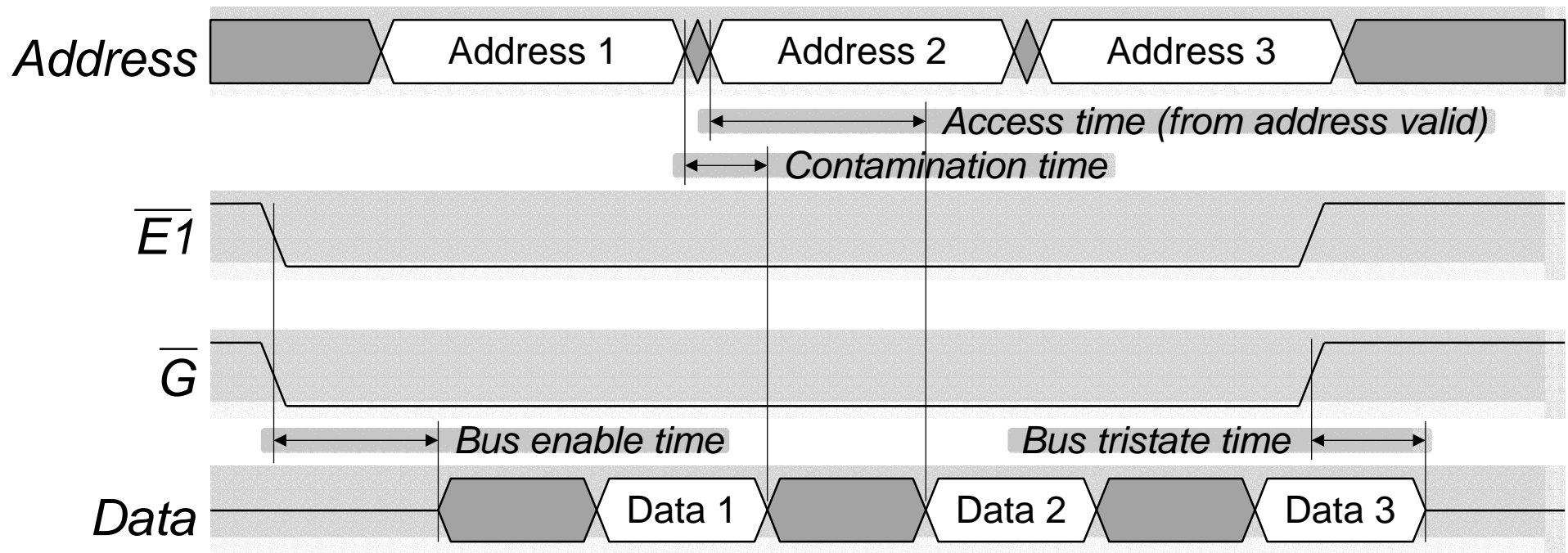


$E2$ assumed high (enabled), $\overline{W} = 1$ (read mode)

- Read cycle begins when all enable signals ($\overline{E1}$, $E2$, \overline{G}) are active.
- Data is valid after read access time.
 - Access time is indicated by full part number: *MCM6264CP* $t_2 \rightarrow 12ns$.
- Data bus is tristated shortly after \overline{G} or $\overline{E1}$ goes high.



Address Controlled Reads

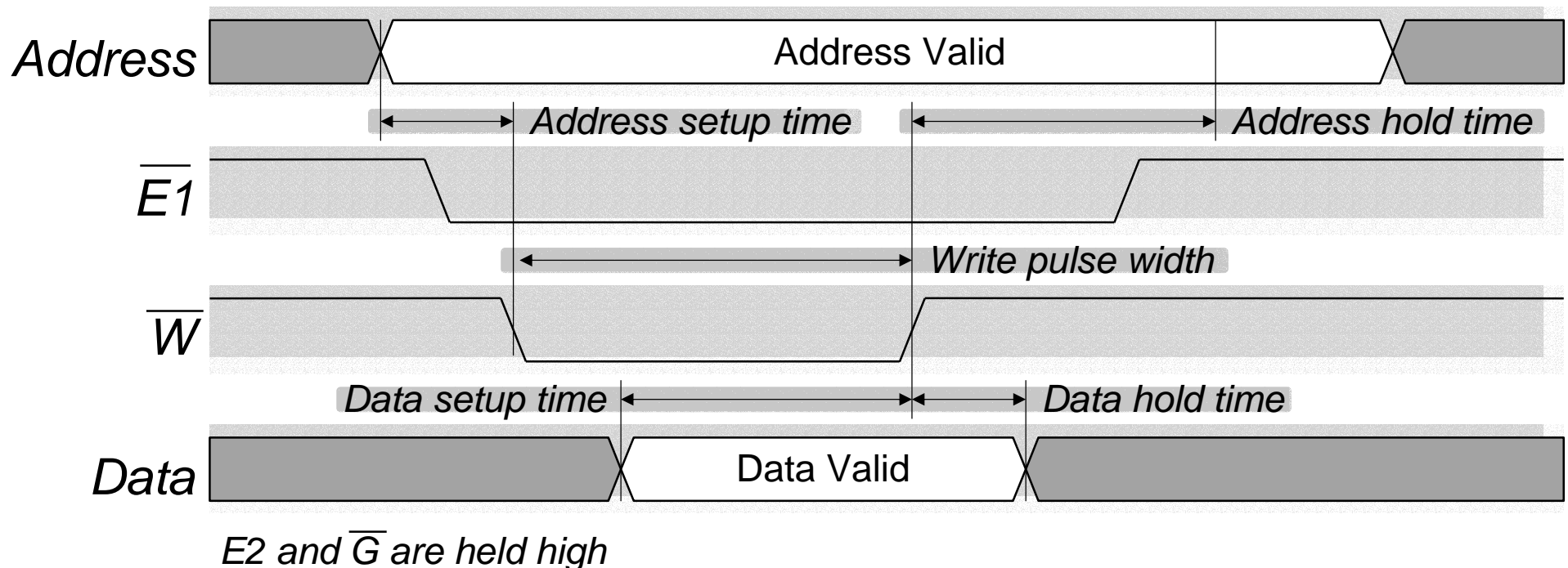


$E2$ assumed high (enabled), $\overline{W} = 1$ (read mode)

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



Writing to SRAM



- 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 can cause writes to random addresses!

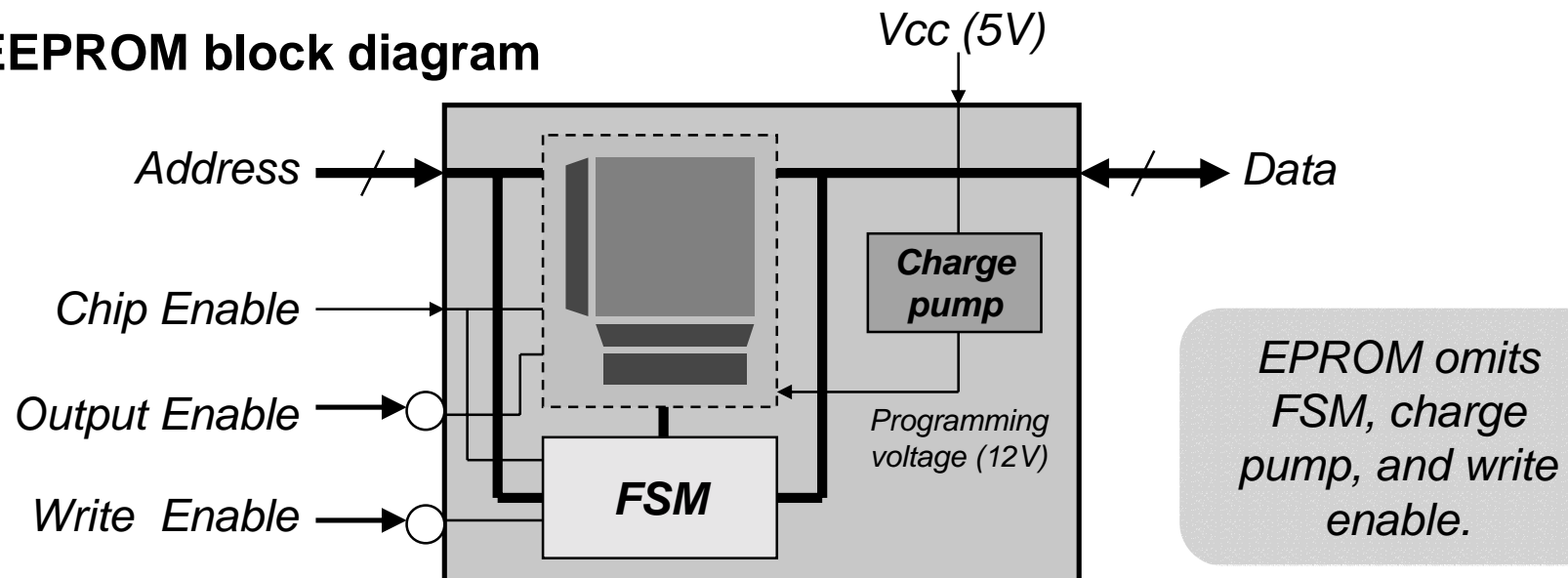


Flash and (E)EPROM



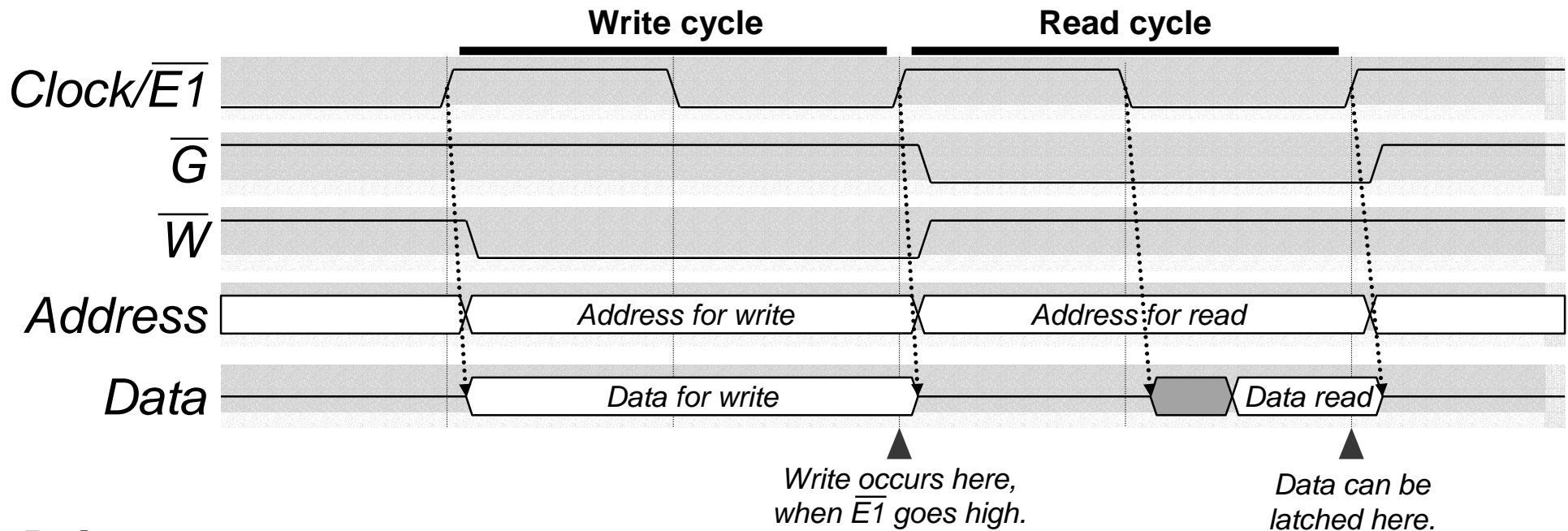
- Reading from flash or (E)EPROM is the same as reading from SRAM.
- V_{pp} : input for programming voltage (12V)
 - EPROM: V_{pp} is supplied by programming machine.
 - Modern flash/EEPROM devices generate 12V using an on chip charge pump.
- EPROM lacks a write enable.
 - Not in system programmable (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



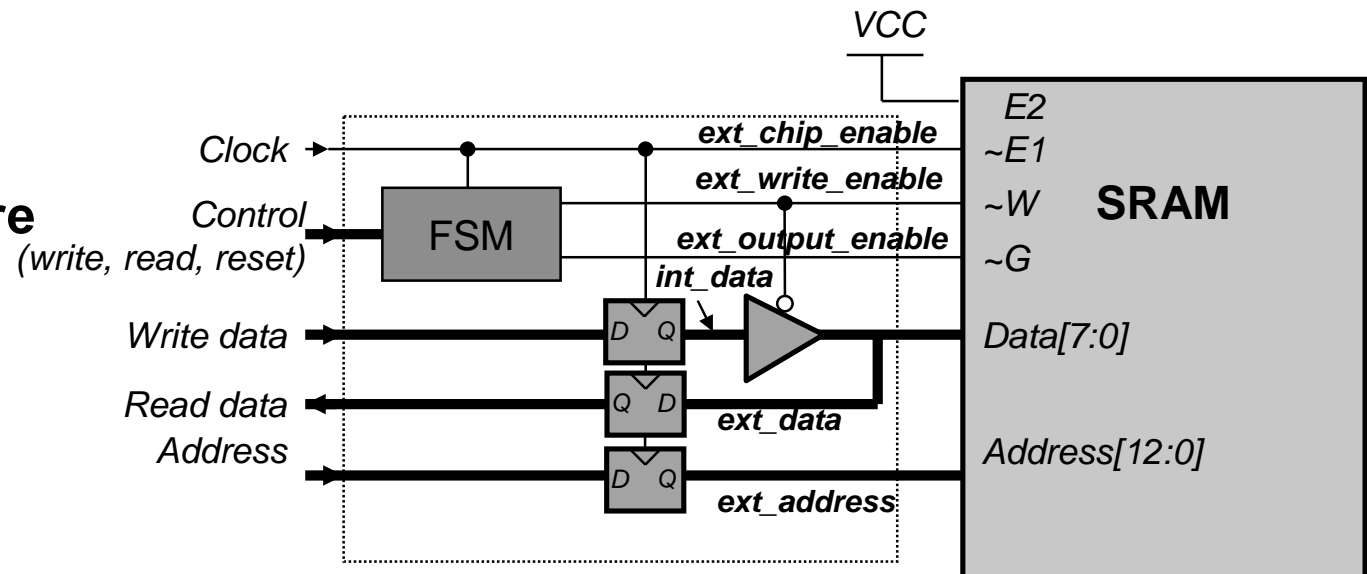


Sample Memory Interface Logic



■ Drive memory enable with clock

- Ensures data and memory busses are stable for writes
- Minimum clock period is twice memory access time.





Toy Memory Interface in VHDL (I)



```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity mem_int is  
  port (clock : in std_logic;  
        reset : in std_logic;  
        write : in std_logic;  
        read : in std_logic;  
        address : in std_logic_vector(12 downto 0);  
        write_data : in std_logic_vector(7 downto 0);  
        read_data : out std_logic_vector(7 downto 0);  
        ext_chip_enable : out std_logic;  
        ext_write_enable : out std_logic;  
        ext_output_enable : out std_logic;  
        ext_address : out std_logic_vector(12 downto 0);  
        ext_data : inout std_logic_vector(7 downto 0));  
end mem_int;
```

```
architecture behavioral of mem_int is
```

```
  signal int_data : std_logic_vector(7 downto 0);
```

```
(cont. on next viewgraph)
```



Toy Memory Interface in VHDL (II)



```
begin
ext_chip_enable <= clock;
ext_data <= int_data when ext_write_enable = '0' else (others => 'Z');

process (clock, reset)
begin
  if reset = '0' then
    ext_write_enable <= '1';
    ext_output_enable <= '1';
  elseif clock'event and clock = '1' then
    ext_address <= address;
    read_data <= ext_data;
    int_data <= write_data;
    if write = '1' then
      ext_write_enable <= '0';
    elsif read = '1' then
      ext_output_enable <= '0';
    else
      ext_write_enable <= '1';
      ext_output_enable <= '1';
    end if;
  end if;
end process;

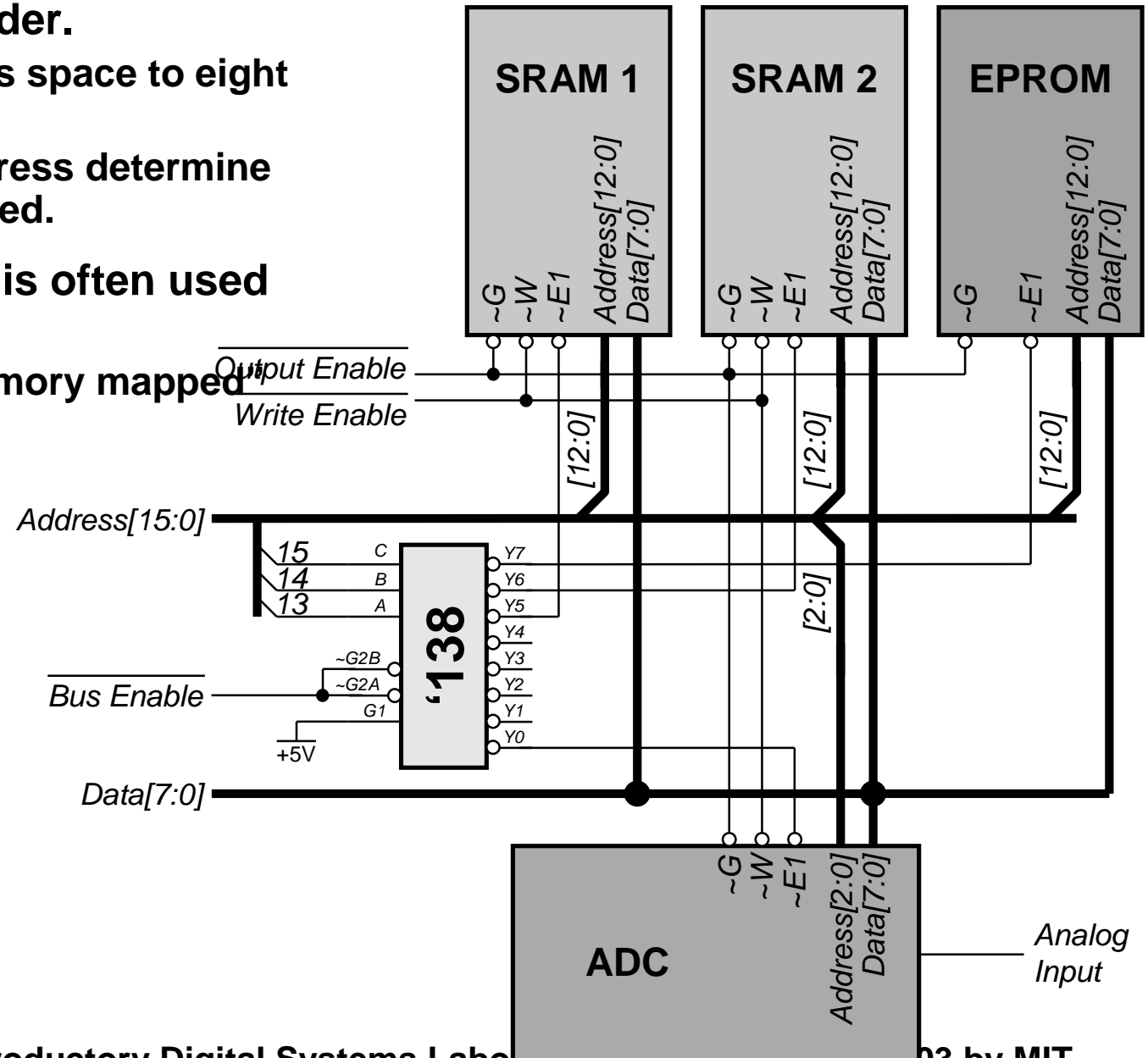
end behavioral;
```



Creating Larger Memories



- '138 is a 3 to 8 decoder.
 - Maps 16 bit address space to eight 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"



Memory Map

0xFFFF	EPROM
0xE000 0xDFFF	
0xC000 0xBFFF	SRAM 2
0xA000 0x9FFF	SRAM 1
0x2000 0x1FFF	ADC
0x0000	



Testing Memories



■ Common device problems

- **Bad locations:** rare for individual locations to be bad
- **Slow (out of spec) timings:** causes intermittent failures
- **Catastrophic device failure:** e.g., ESD
- **Missing wire bonds/devices (!):** possible with automated assembly
- **Transient failures:** Alpha particles, power supply glitch

■ Common circuit problems

- **Stuck at faults:** a pin shorted to VDD 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



Testing Memory



- **Since 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.**



Testing Memory

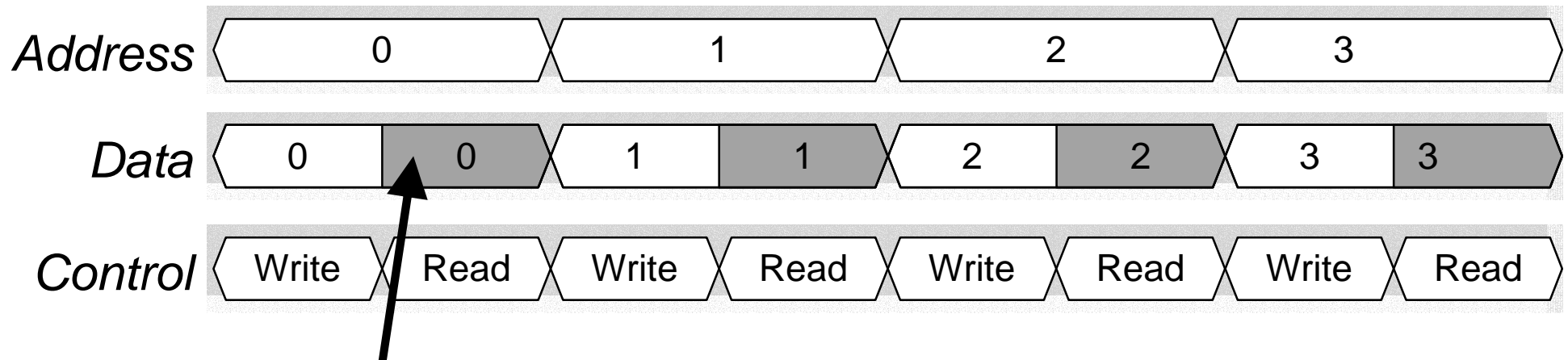


■ An idea that almost works

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!

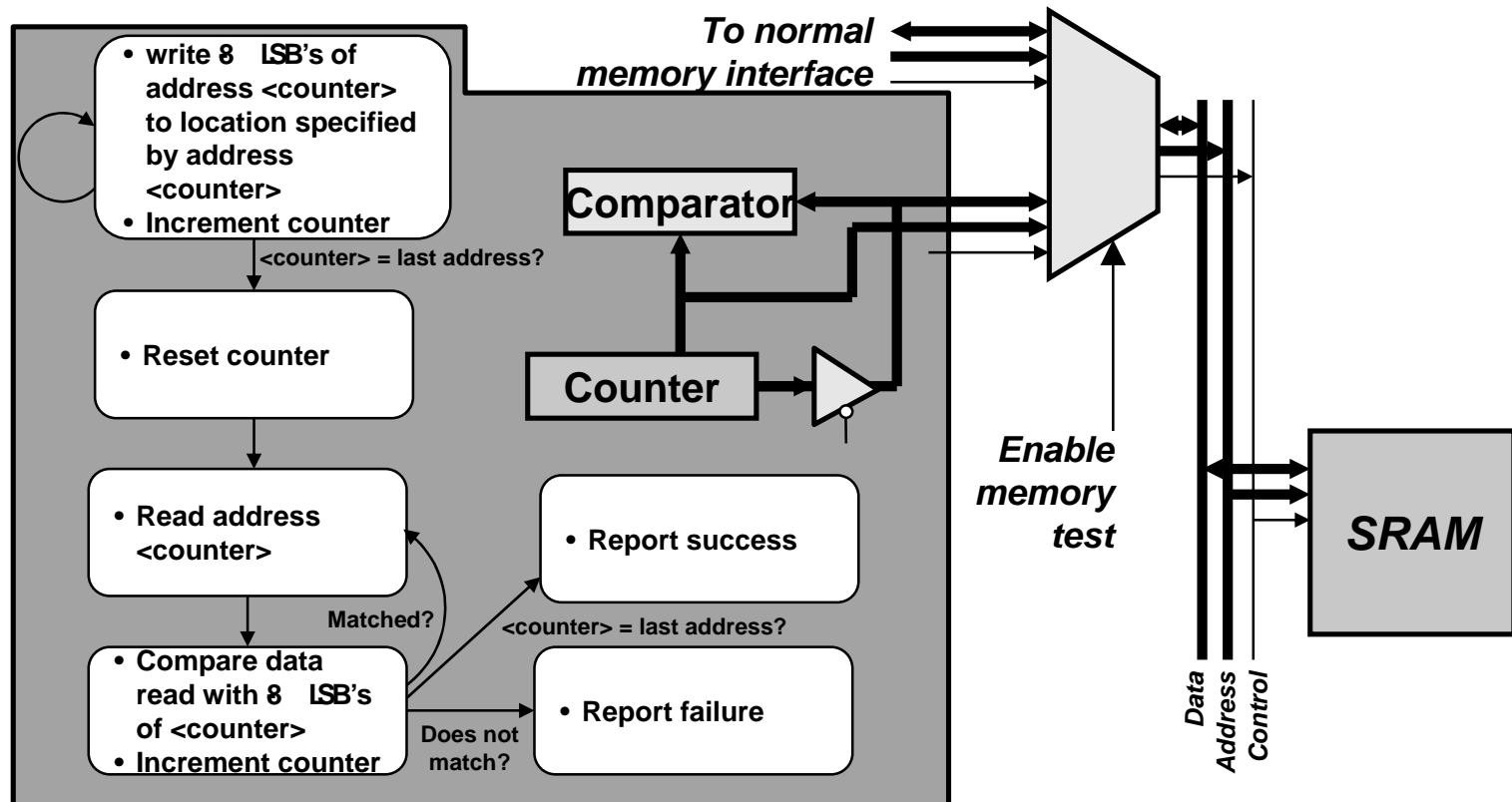


Testing Memory



- 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 ($n \bmod 256$).





Multi-Port Memories

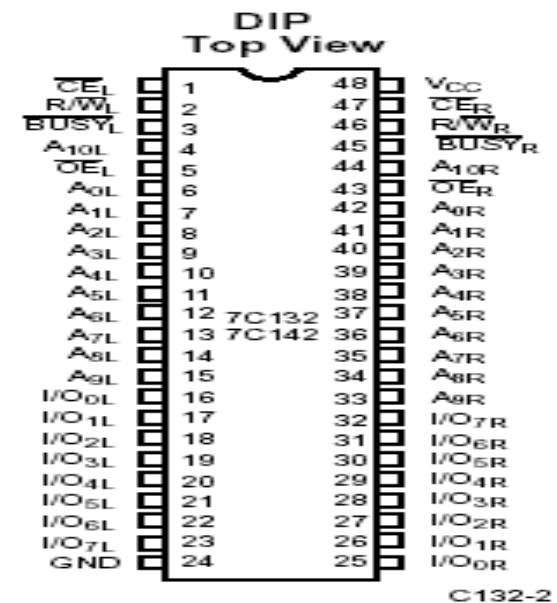
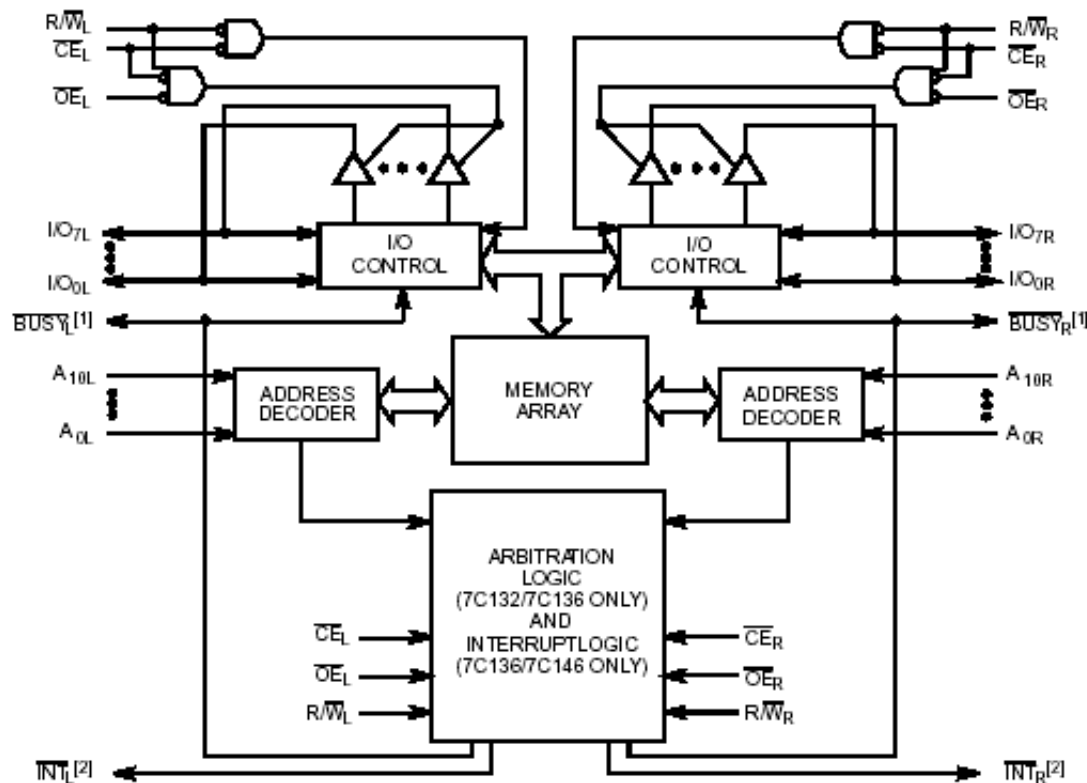


■ Applications

- Register files, FIFO, buffer, etc.

■ Cypress CY7C132 2kx8 dual port SRAM

- Two independent, asynchronous read/write ports
- Can access two different memory locations simultaneously
- $\overline{\text{Busy}}$ signal indicates if both ports attempt to access same location at the same time



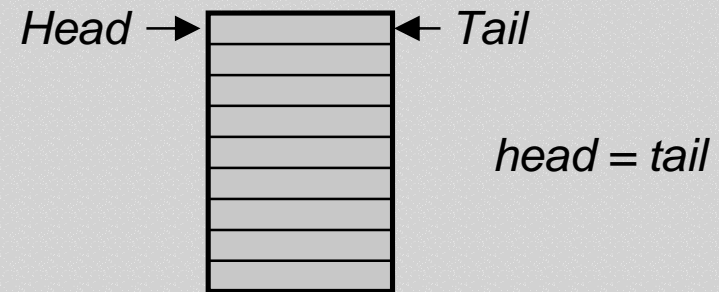


Building a FIFO

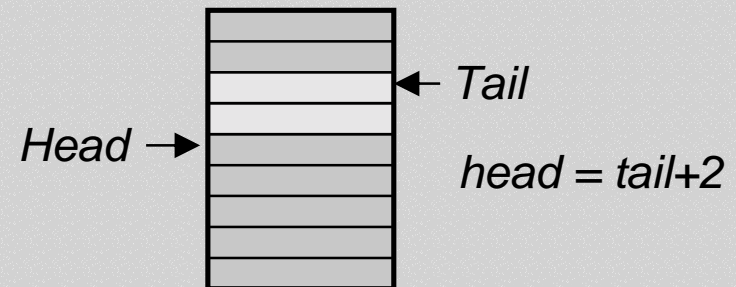


- **Head pointer:** points to next available location for writing
- **Tail pointer:** points to next available location for reading
- If head and tail pointers are the same, FIFO is empty.
- If $head+1 = tail$, FIFO is full.
- Pointers wrap around when they reach the end of the memory.

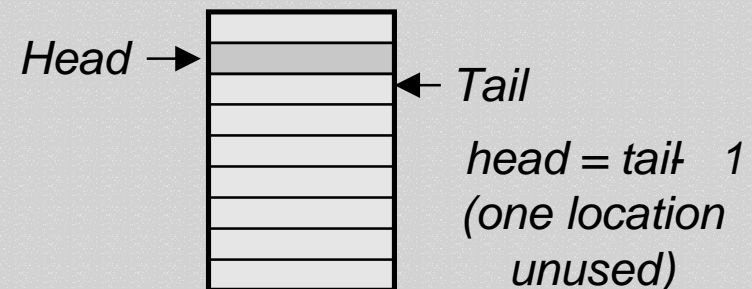
Empty



After 4 writes and 2 reads



Full





Building a FIFO

