

Memory in SystemVerilog

Prof. Stephen A. Edwards

Columbia University

Spring 2020

Implementing Memory

Memory = Storage Element Array + Addressing

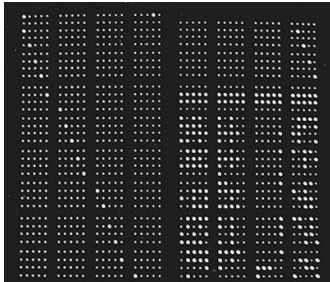
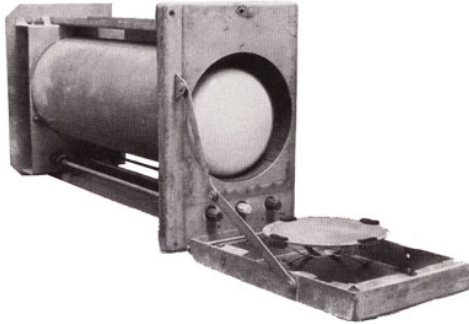
Bits are expensive

They should be dumb, cheap, small, and tightly packed

Bits are numerous

Can't just connect a long wire to each one

Williams Tube



Mercury acoustic delay line



Used in the EDASC,
1947.

32×17 bits

Selectron Tube



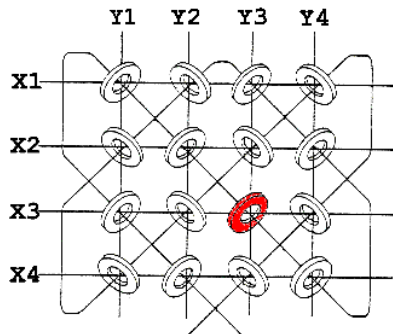
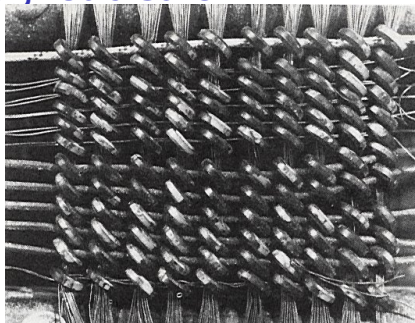
RCA, 1948.

2×128 bits

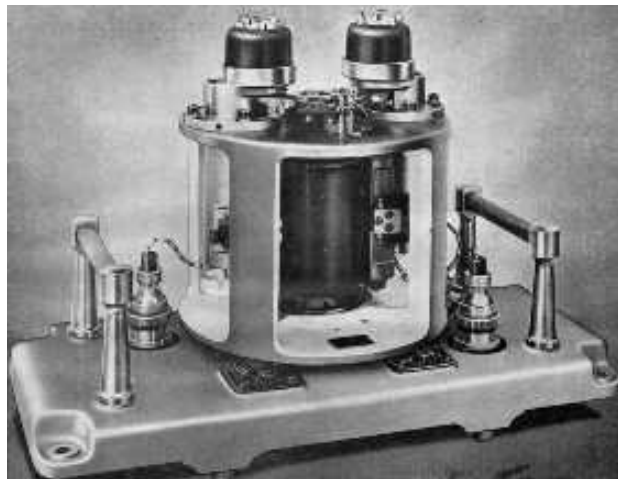
Four-dimensional addressing

A four-input AND gate at
each bit for selection

Magnetic Core











Magnetic Drum Memory

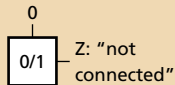


1950s & 60s. Secondary storage.

Modern Memory Choices

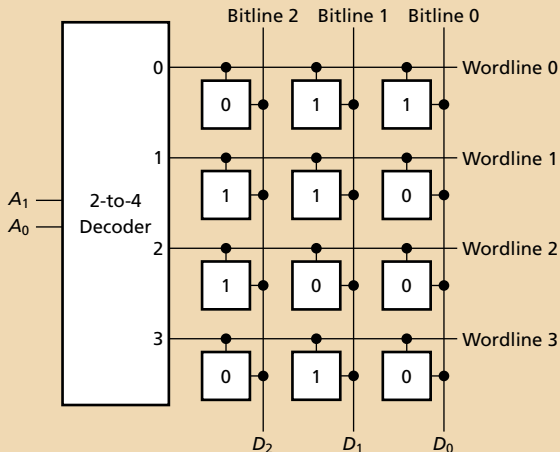
	Family	Programmed	Persistence
	Mask ROM	at fabrication	∞
	PROM	once	∞
	EPROM	1000s, UV	10 years
	FLASH	1000s, block	10 years
	EEPROM	1000s, byte	10 years
	NVRAM	∞	5 years
	SRAM	∞	while powered
	DRAM	∞	64 ms

Implementing ROMs

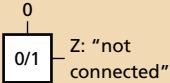


Add. Data

00	011
01	110
10	100
11	010

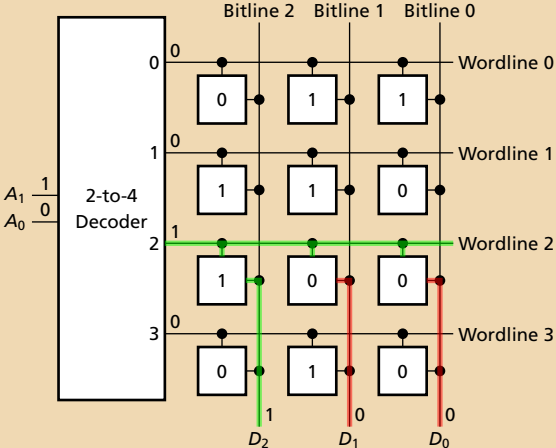


Implementing ROMs

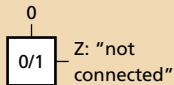


Add. Data

00	011
01	110
10	100
11	010

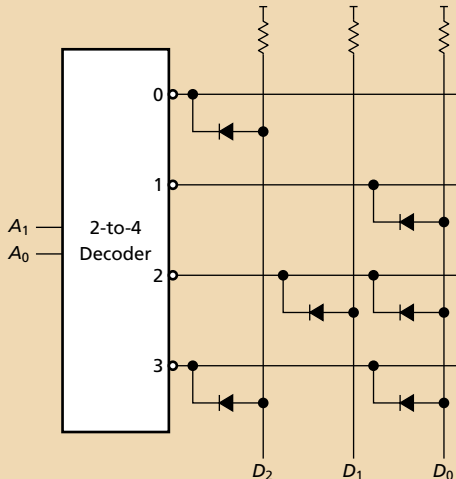


Implementing ROMs

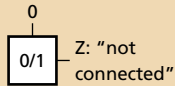


Add. Data

00	011
01	110
10	100
11	010

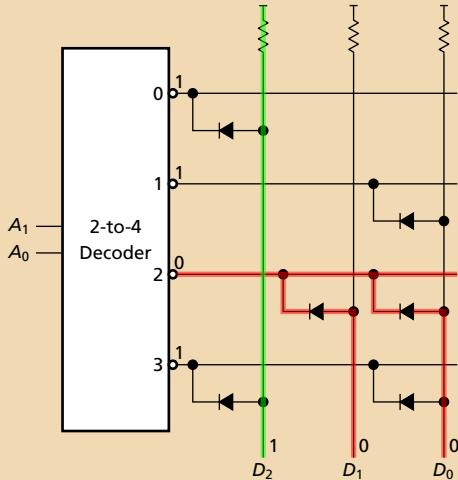


Implementing ROMs

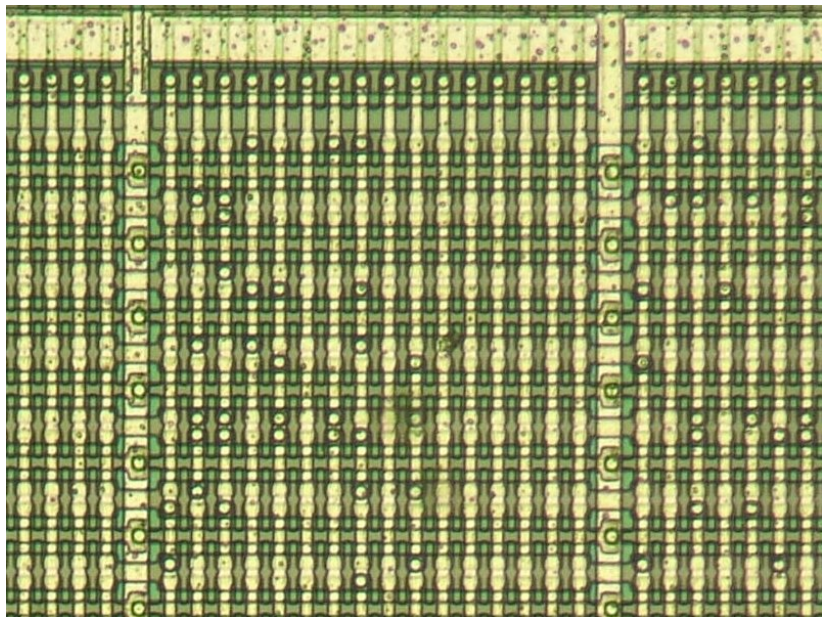


Add. Data

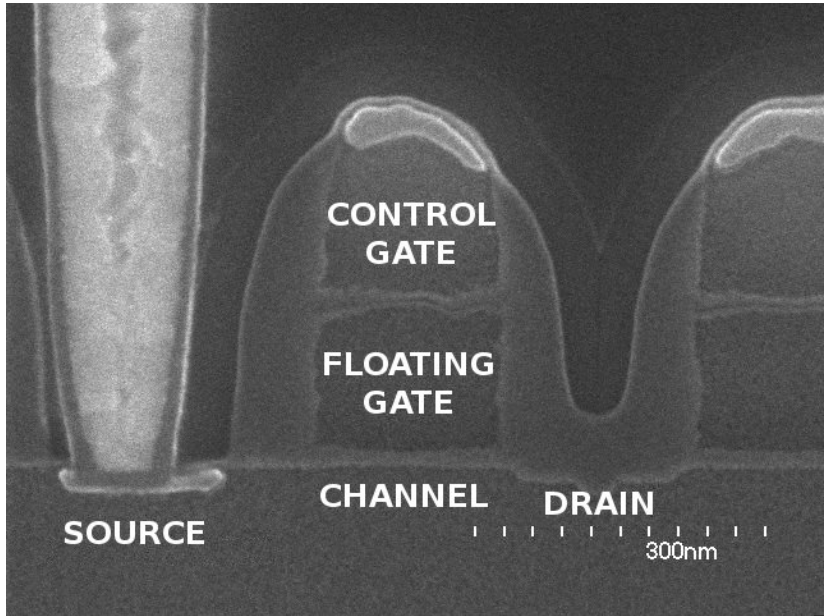
00	011
01	110
10	100
11	010



Mask ROM Die Photo

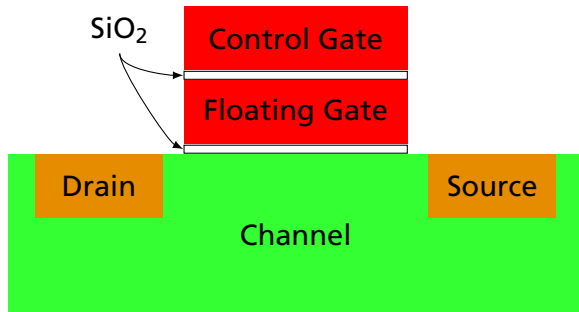


A Floating Gate MOSFET



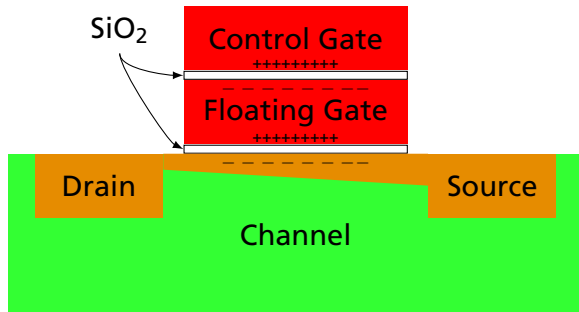
Cross section of a NOR FLASH transistor. Kawai et al., ISSCC 2008 (Renesas)

Floating Gate n-channel MOSFET



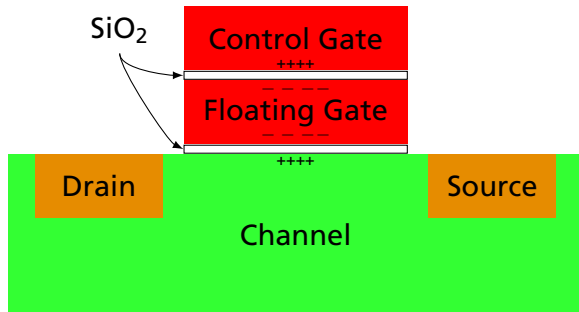
Floating gate uncharged; Control gate at 0V: Off

Floating Gate n-channel MOSFET



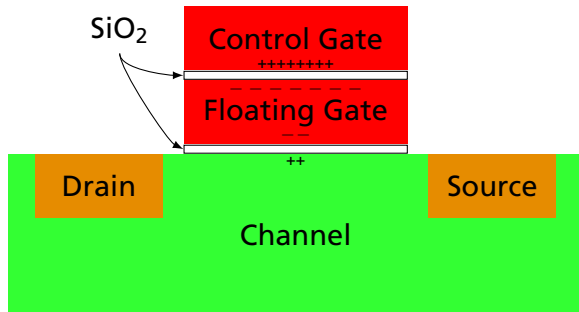
Floating gate uncharged; Control gate positive: On

Floating Gate n-channel MOSFET



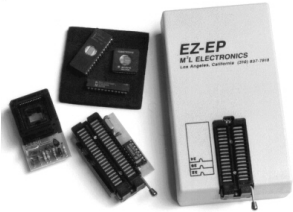
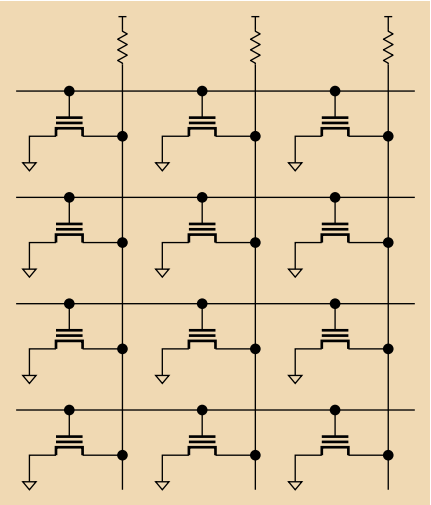
Floating gate negative; Control gate at 0V: Off

Floating Gate n-channel MOSFET

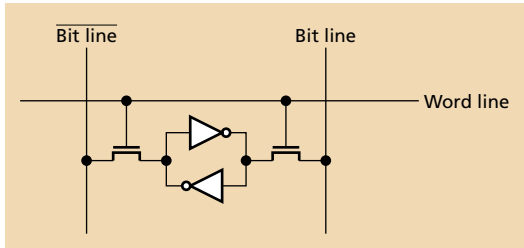


Floating gate negative; Control gate positive: Off

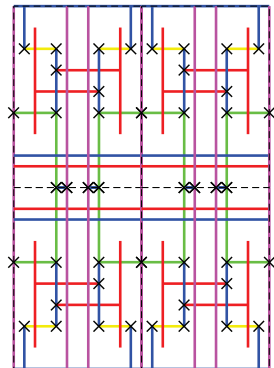
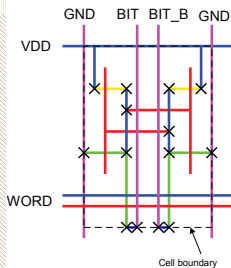
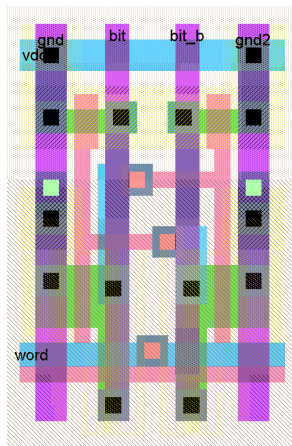
EPROMs and FLASH use Floating-Gate MOSFETs



Static Random-Access Memory Cell

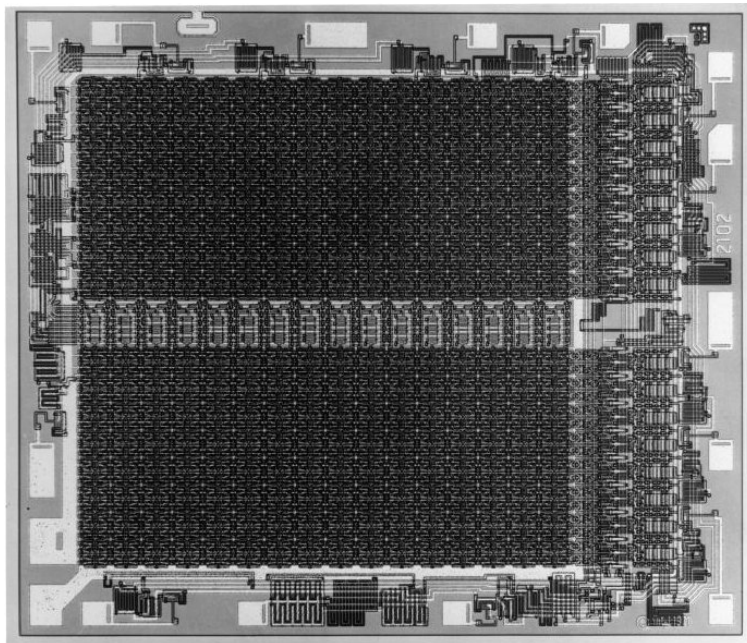


Layout of a 6T SRAM Cell

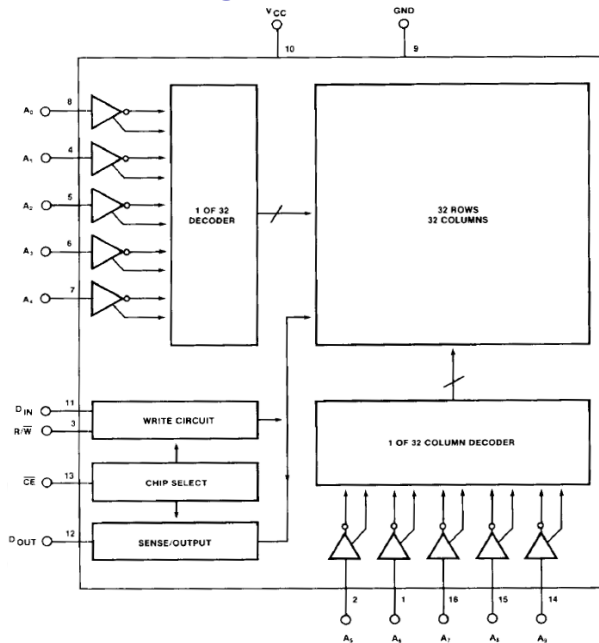


Weste and Harris. *Introduction to CMOS VLSI Design*. Addison-Wesley, 2010.

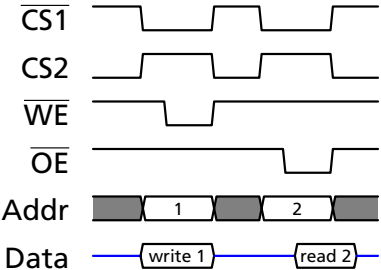
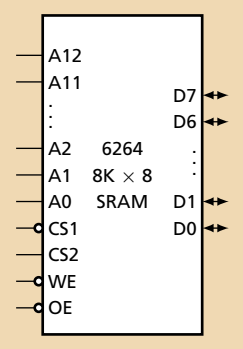
Intel's 2102 SRAM, 1024 × 1 bit, 1972



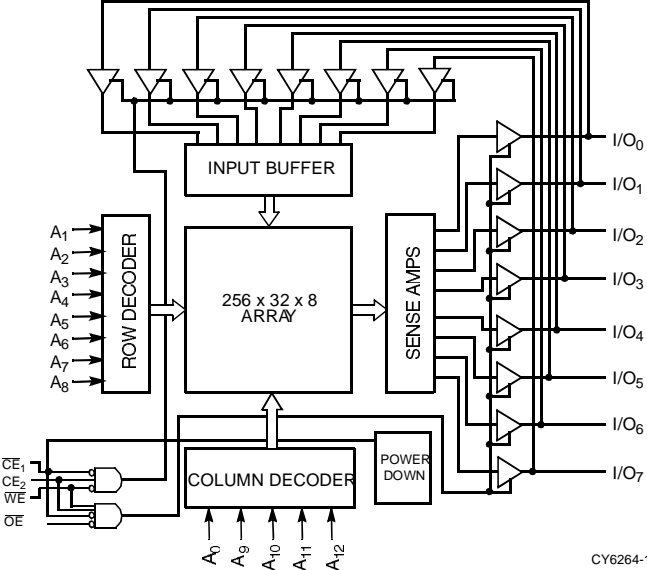
2102 Block Diagram



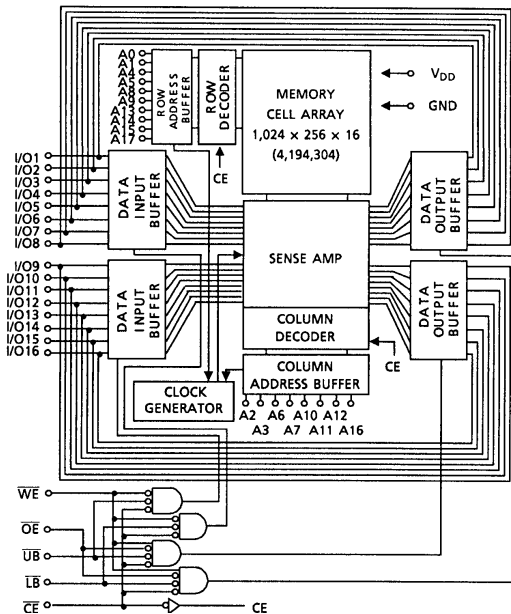
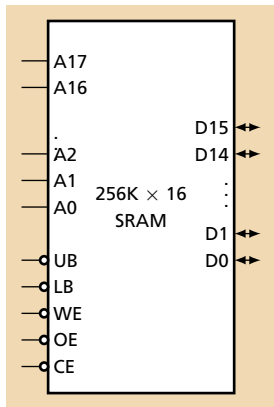
SRAM Timing



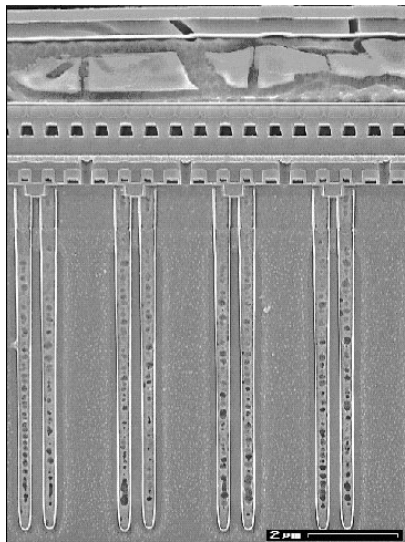
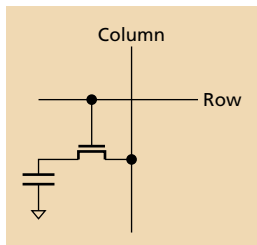
6264 SRAM Block Diagram



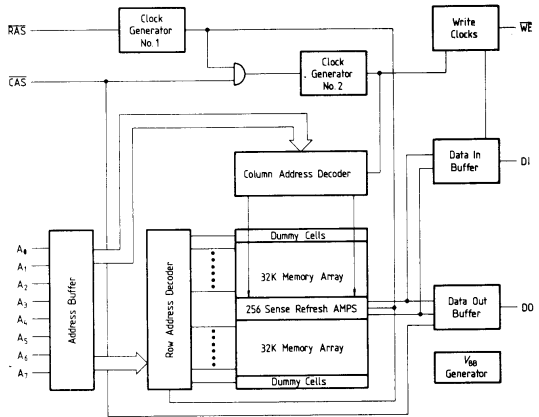
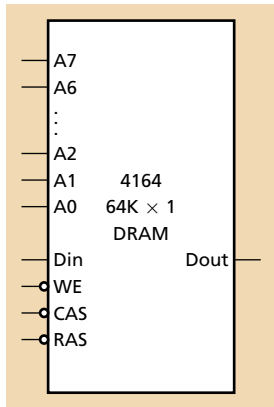
Toshiba TC55V16256J 256K × 16



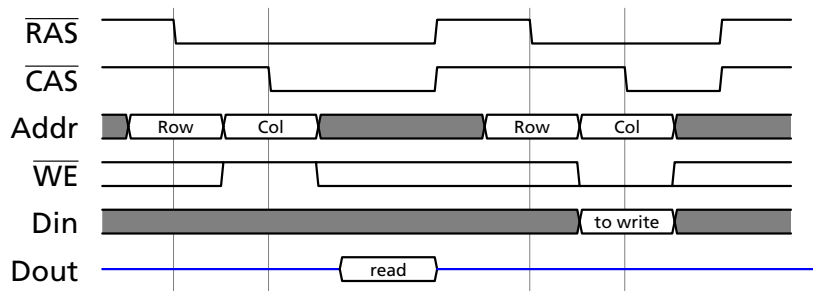
Dynamic RAM Cell



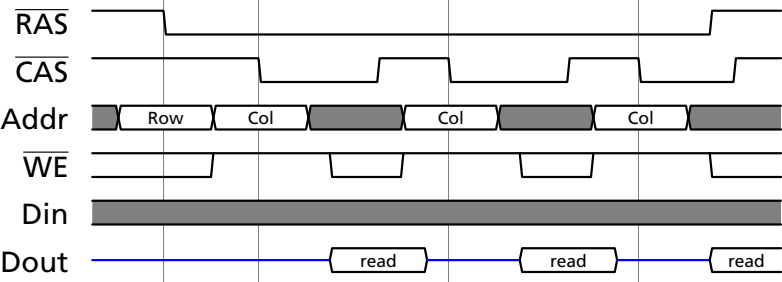
Ancient (c. 1982) DRAM: 4164 64K × 1



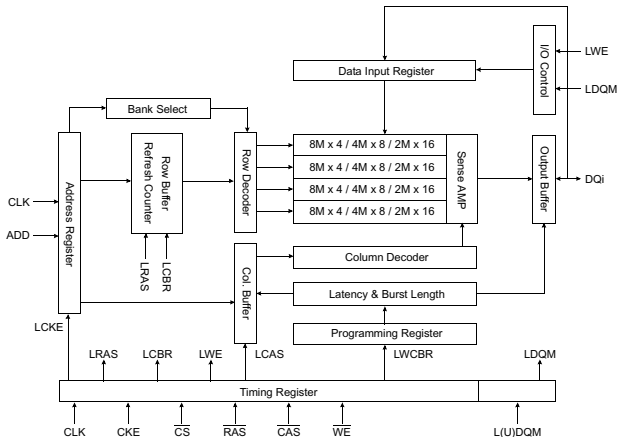
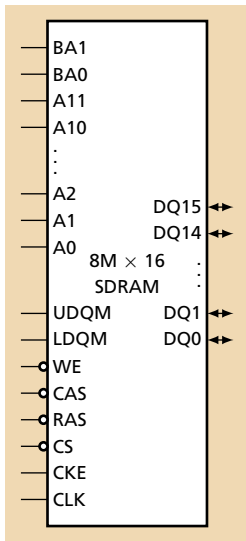
Basic DRAM read and write cycles



Page Mode DRAM read cycle



Samsung 8M × 16 SDRAM

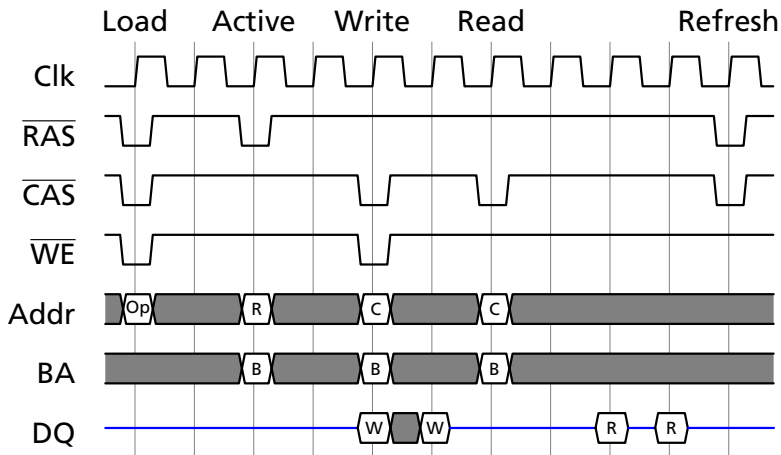


SDRAM: Control Signals

$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	Action
1	1	1	NOP
0	0	0	Load mode register
0	1	1	Active (select row)
1	0	1	Read (select column, start burst)
1	0	0	Write (select column, start burst)
1	1	0	Terminate Burst
0	1	0	Precharge (deselect row)
0	0	1	Auto Refresh

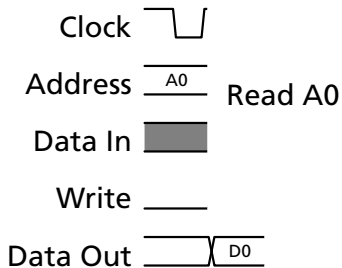
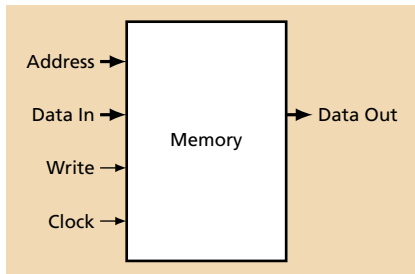
Mode register: selects 1/2/4/8-word bursts, CAS latency, burst on write

SDRAM: Timing with 2-word bursts

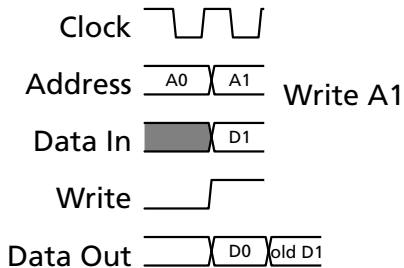
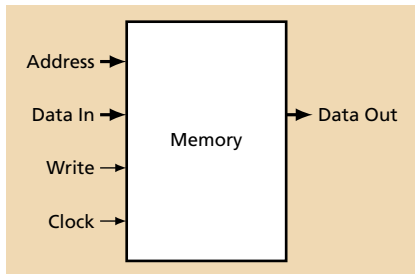


Using Memory in SystemVerilog

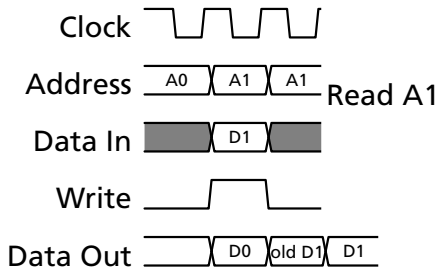
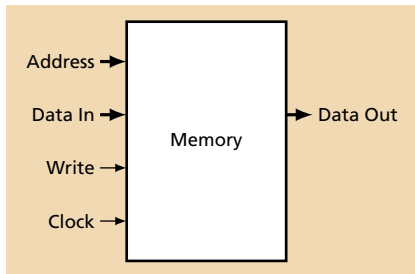
Synchronous SRAM



Synchronous SRAM



Synchronous SRAM



Memory Is Fundamentally a Bottleneck



Plenty of bits, but

You can only see a small window each clock cycle

Using memory = scheduling memory accesses

Software hides this from you: sequential programs naturally schedule accesses

You must schedule memory accesses in a hardware design

Modeling Synchronous Memory in SystemVerilog

```
module memory(  
  input logic      clk      ,  
  input logic      write    ,  
  input logic [3:0] address ,  
  input logic [7:0] data_in ,  
  output logic [7:0] data_out);  
  
  logic [7:0] mem [15:0];  
  
  always_ff @(posedge clk)  
  begin  
    if (write)  
      mem[address] <= data_in;  
    data_out <= mem[address];  
  end  
  
endmodule
```

Write enable

4-bit address

8-bit input bus

8-bit output bus

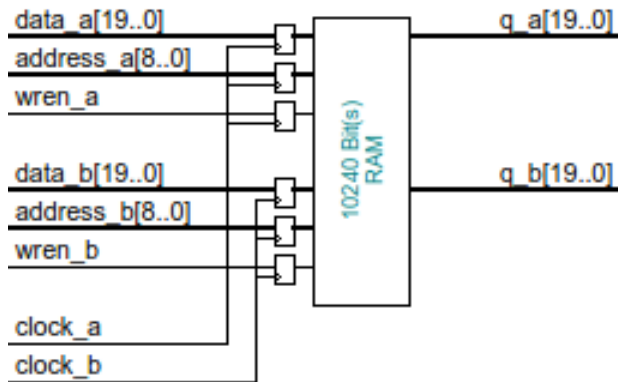
The memory array: 16 8-bit bytes

Clocked

Write to array when asked

Always read (old) value from array

M10K Blocks in the Cyclone V



10 kilobits (10240 bits) per block

Dual ported: two addresses, write enable signals

Data busses can be 1–20 bits wide

Our Cyclone 5CSEMA5 has 397 of these blocks = 496 KB

Memory in Quartus: the Megafunction Wizard

Which megafunction would you like to customize? Which device family will you be using? Cyclone V

Select a megafunction from the list below

Which type of output file do you want to create?

What name do you want for the output file?

Output files will be generated using the classic file structure

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).


Your current user library directories are:

Cancel < Back Next > Finish

Detailed description of the dialog box content:

- Which megafunction would you like to customize?** Select a megafunction from the list below
- Which device family will you be using?** Cyclone V
- Which type of output file do you want to create?**
 - AHDL
 - VHDL
 - Verilog HDL
- What name do you want for the output file?** /home/sedwards/svn/classes/2014/4840/dummy/memory
- Output files will be generated using the classic file structure**
 - Return to this page for another create operation
- Note:** To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).
- Your current user library directories are:** (Empty text box)

Memory: Single- or Dual-Ported



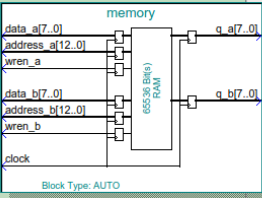
RAM: 2-PORT

[About](#) [Documentation](#)

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > Output1 > Output2 > Mem Init >

memory



Block Type: AUTO

Currently selected device family:

Match project/default

How will you be using the dual port RAM?

With one read port and one write port


With two read/write ports

How do you want to specify the memory size?

As a number of words

As a number of bits

Memory: Select Port Widths

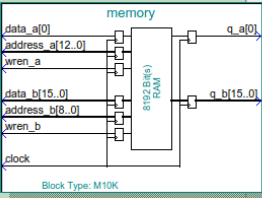


RAM: 2-PORT

[About](#) [Documentation](#)

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > Regs/Clkens/Aclrs > Output1 > Output2 > Mem Init >



Block Type: M10K

How many bits of memory?

Use different data widths on different ports

Read/Write Ports

How wide should the 'q_a' output bus be?

How wide should the 'data_a' input bus be?

How wide should the 'q_b' output bus be?


Note: You could enter arbitrary values for width and depth

What should the memory block type be?

Auto MLAB M10K M144K LCs

Set the maximum block depth to words

Memory: One or Two Clocks

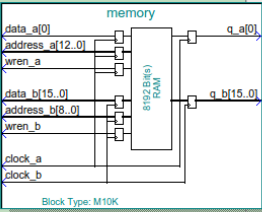


RAM: 2-PORT

[About](#) [Documentation](#)

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > **Clks/Rd, Byte En** > Regs/Clkens/Aclrs > Output2 > Mem Init >



Block Type: M10K

What clocking method do you want to use?

- Single clock
- Dual clock: use separate 'read' and 'write' clocks
- Dual clock: use separate 'input' and 'output' clocks
- No clock (fully asynchronous)
- Dual clock: use separate clocks for A and B ports

Create 'rden_a' and 'rden_b' read enable signals


Byte Enable Ports

- Create byte enable for port A
- Create byte enable for port B

What is the width of a byte for byte enables? bits

Enable error checking and correcting (ECC) to check and correct single bit errors and detect double errors

Memory: Output Ports Need Not Be Registered

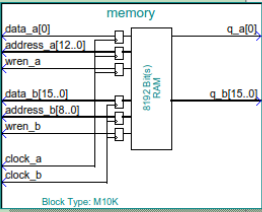


RAM: 2-PORT

[About](#) [Documentation](#)

1 Parameter Settings | 2 EDA | 3 Summary

General > Widths/Blk Type > Clks/Rd, Byte En > **Regs/Clkens/Aclrs** > Output2 > Mem Init >



Block Type: M10K

Which ports should be registered?

- Write input ports
'data_a', 'waddress_a', and 'wren_a'
- Read input ports
'rdaddress' and 'rden'
- Read output port(s)
'q_a' and 'q_b'

Create one clock enable signal for each clock signal

Use different clock enables for registers

Create an 'aclr' asynchronous clear for the registered ports

[More Options...](#)

[More Options...](#)

[More Options...](#)

Memory: Wizard-Generated Verilog Module

This generates the following SystemVerilog module:

```
module memory (
    input logic [12:0] address_a, // Port A:
    input logic      clock_a,    // 8192 1-bit words
    input logic [0:0] data_a,
    input logic      wren_a,    // Write enable
    output logic [0:0] q_a,

    // Port B:
    input logic [8:0] address_b, // 512 16-bit words
    input logic      clock_b,
    input logic [15:0] data_b,
    input logic      wren_b,    // Write enable
    output logic [15:0] q_b);
```

Instantiate like any module; Quartus treats specially

Two Ways to Ask for Memory

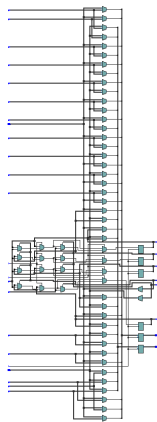
1. Use the Megafunction Wizard
 - + Warns you in advance about resource usage
 - Awkward to change
2. Let Quartus infer memory from your code
 - + Better integrated with your code
 - Easy to inadvertently ask for garbage

The Perils of Memory Inference

```
module twoport(  
    input logic clk,  
    input logic [8:0] aa, ab,  
    input logic [19:0] da, db,  
    input logic wa, wb,  
    output logic [19:0] qa, qb);  
  
    logic [19:0] mem [511:0];  
  
    always_ff @(posedge clk) begin  
        if (wa) mem[aa] <= da;  
        qa <= mem[aa];  
        if (wb) mem[ab] <= db;  
        qb <= mem[ab];  
    end  
  
endmodule
```

Failure: Exploded!

Synthesized to an 854-page
schematic with 10280
registers (no M10K blocks)
Page 1 looked like this:



The Perils of Memory Inference

Failure

Still didn't work:

*RAM logic "mem" is
uninferred due to
unsupported
read-during-write behavior*

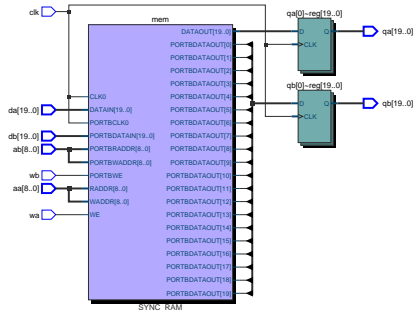
```
module twoport2(  
    input logic clk,  
    input logic [8:0] aa, ab,  
    input logic [19:0] da, db,  
    input logic wa, wb,  
    output logic [19:0] qa, qb);  
  
    logic [19:0] mem [511:0];  
  
    always_ff @(posedge clk) begin  
        if (wa) mem[aa] <= da;  
        qa <= mem[aa];  
    end  
  
    always_ff @(posedge clk) begin  
        if (wb) mem[ab] <= db;  
        qb <= mem[ab];  
    end  
  
endmodule
```

The Perils of Memory Inference

```
module twoport3(  
    input logic clk,  
    input logic [8:0] aa, ab,  
    input logic [19:0] da, db,  
    input logic wa, wb,  
    output logic [19:0] qa, qb);  
  
    logic [19:0] mem [511:0];  
  
    always_ff @(posedge clk) begin  
        if (wa) begin  
            mem[aa] <= da;  
            qa <= da;  
        end else qa <= mem[aa];  
    end  
  
    always_ff @(posedge clk) begin  
        if (wb) begin  
            mem[ab] <= db;  
            qb <= db;  
        end else qb <= mem[ab];  
    end  
  
endmodule
```

Finally!

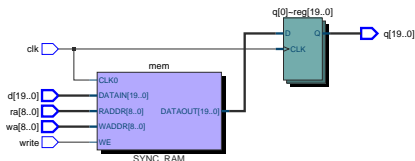
Took this structure from a
template: Edit→Insert
Template→Verilog HDL→Full
Designs→RAMs and
ROMs→True Dual-Port RAM
(single clock)



The Perils of Memory Inference

```
module twoport4(  
    input logic clk,  
    input logic [8:0] ra, wa,  
    input logic write,  
    input logic [19:0] d,  
    output logic [19:0] q);  
  
    logic [19:0] mem [511:0];  
  
    always_ff @(posedge clk) begin  
        if (write) mem[wa] <= d;  
        q <= mem[ra];  
    end  
  
endmodule
```

Also works: separate read
and write addresses



Conclusion:

Inference is fine for single
port or one read and one
write port.

Use the Megafuction Wizard
for anything else.