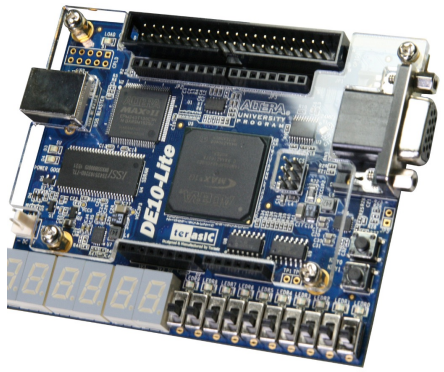
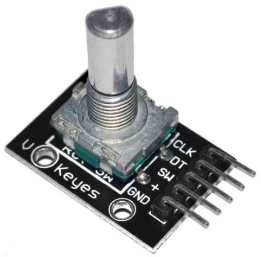
Digital Circuit Design

Li Bai

Return equipment

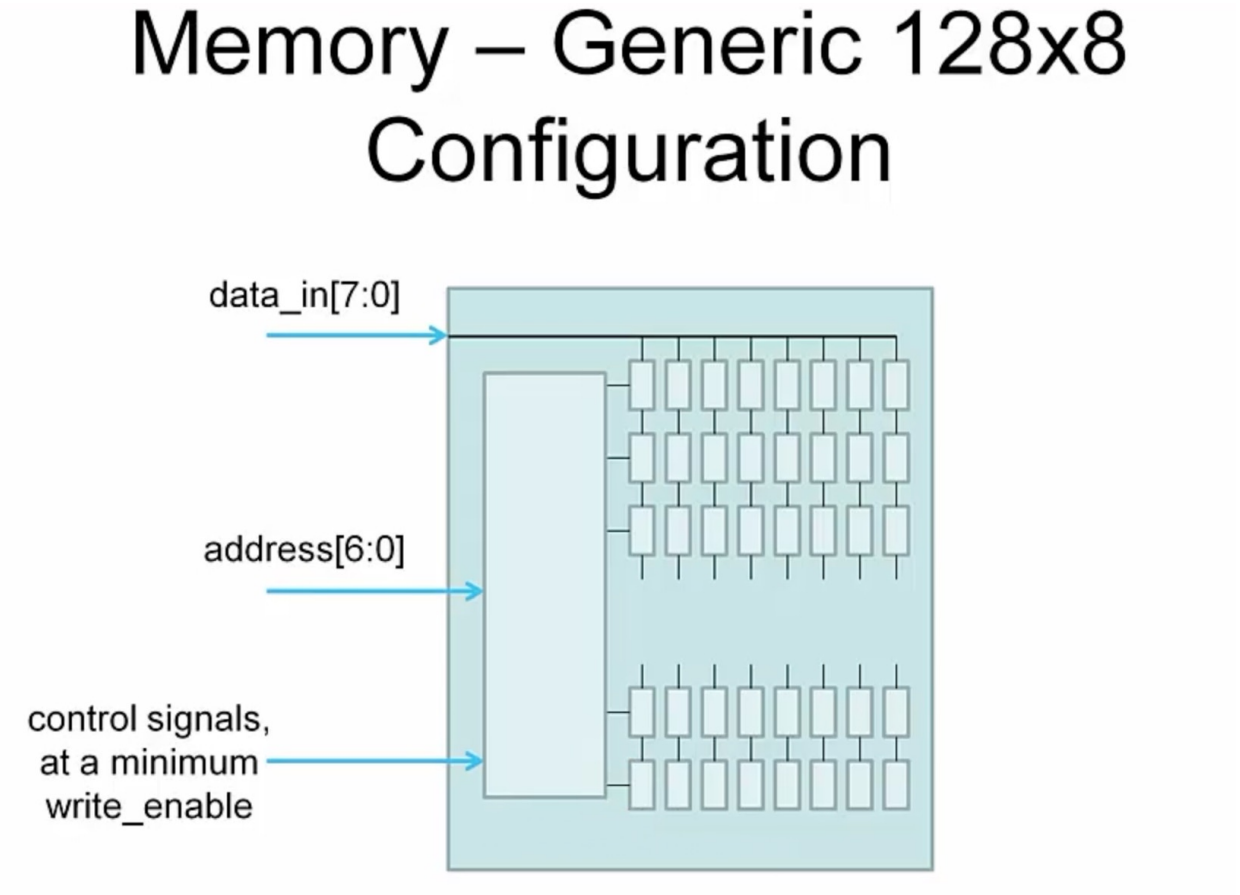
- DE10-Lite, Raspberry Pi 3B+, light sensor, and rotary encoder



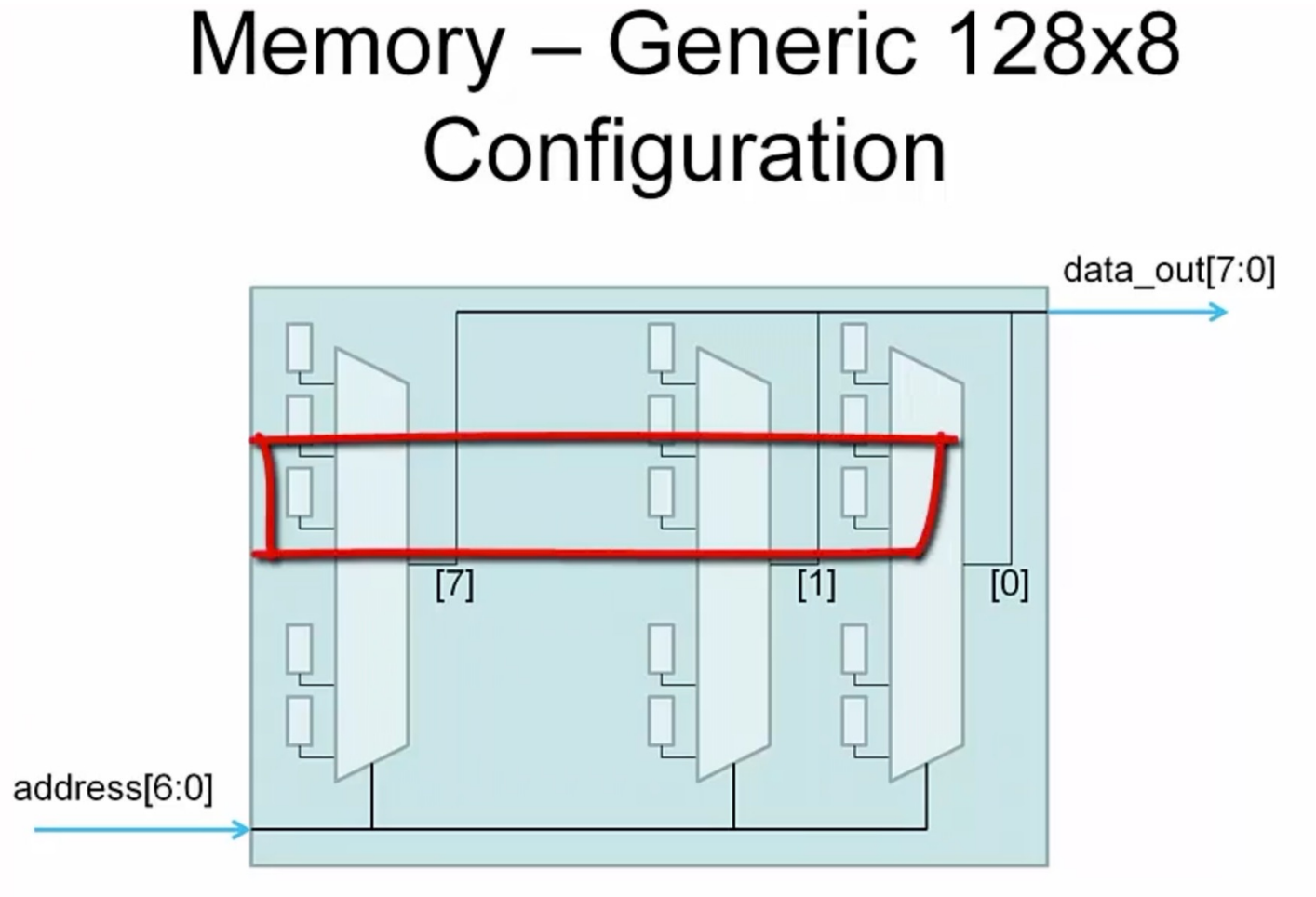


- Return days: May 6, 2024
- If you cannot return in person, please mail it back to Michael Caiozzo. Get an instruction from him. It is better to get a tracking number and send the number to him.

How memory write works



How memory read works



Memory of 1024 bits

- What other configurations might designers consider when *packaging up* 1024 bits?
  - 1024 x 1
  - 256 x 4
  - 128 x 8
  - 64 x 16

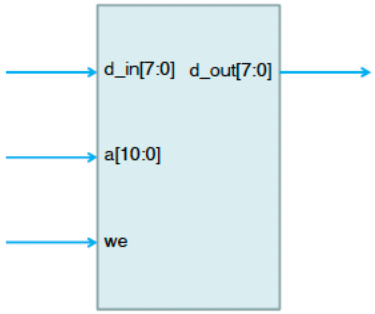
- All have the same number of storage bits, just configured differently.

Size

Memory Size Notation

- Kilo (K): 2^{10} = 1,024
- Mega (M): 2^{20} = 1,024x1,024 = 1,048,576
- Giga (G): 2^{30} = 1,073,741,824
- Tera (T): 2^{40} = 1,099,511,627,776

Base Design

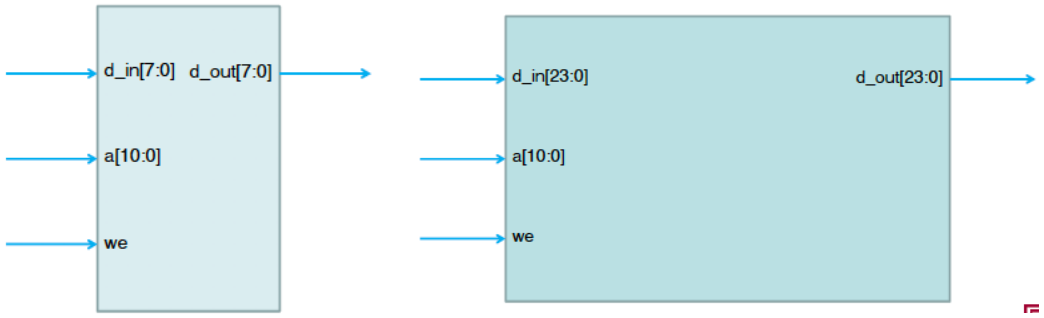



```
// base design generic 2K x 8
module basic_2k_8 (output wire [7:0] d_out,
input wire [7:0] d_in, input wire [10:0] a, input wire we);
...
endmodule
```

Memory expansion

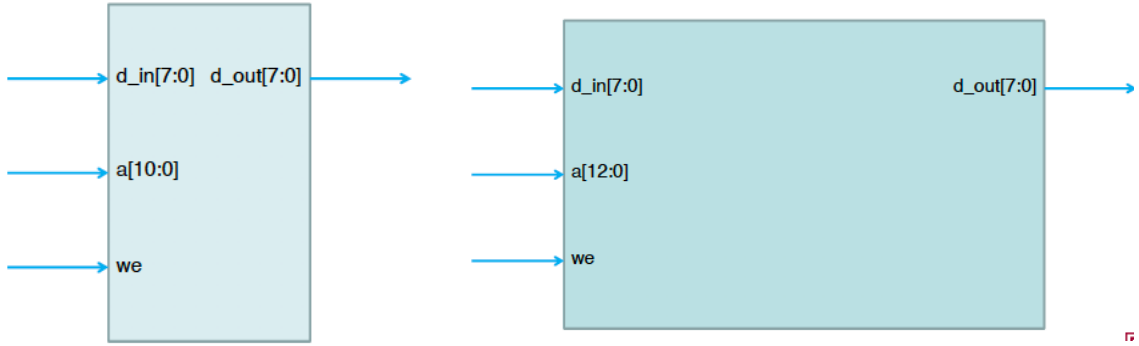
Expand data bit

- Base design and target design
- 2K x 8 (b) ➔ 2K x 24 (t)

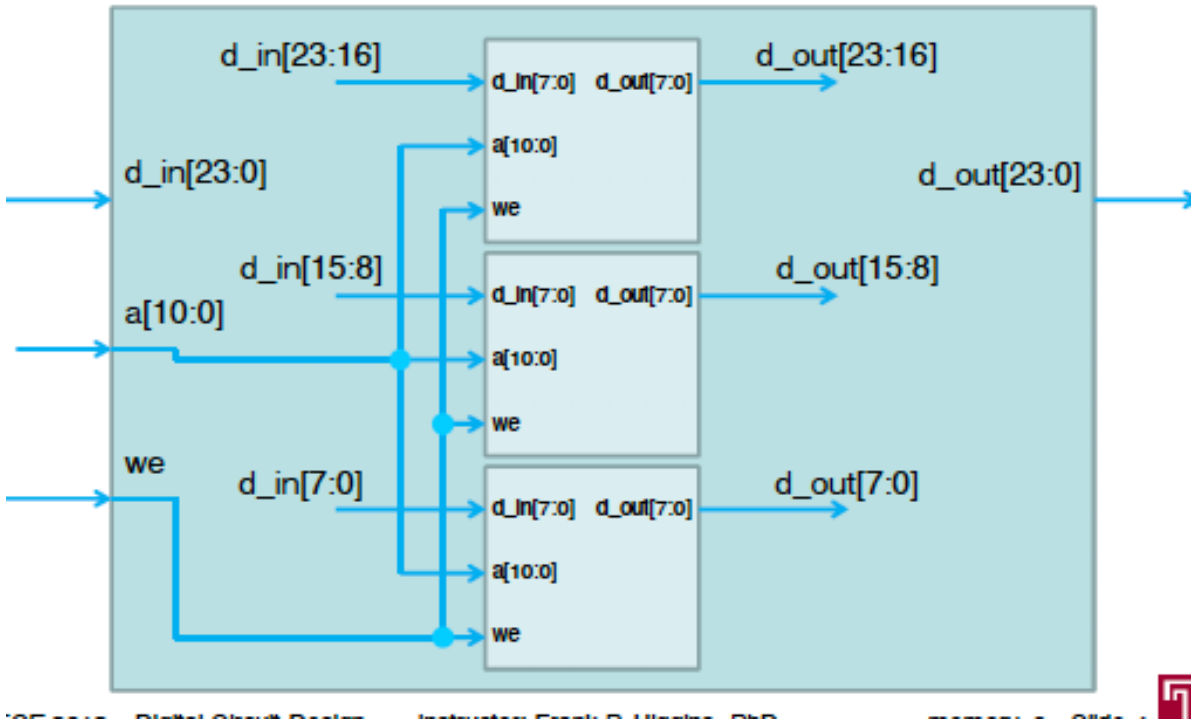


Expand address bit

- Target and base design
- 2K x 8 (b) ➔ 8K x 8 (t)



Bit expansion




```
// basic, generic 2K x 8
module basic_2k_8 (output wire [7:0] d_out,
input wire [7:0] d_in, input wire [10:0] a, input wire we);
...
endmodule

 // target, 2K x 24
 module target_2k_24 (output wire [23:0] d_out,
 input wire [23:0] d_in, input wire [10:0] a, input wire we);

   basic_2k_8 m1 (.d_in(d_in[7:0]), .a(a[10:0]), .we(we), .d_out(d_out[7:0]));
   basic_2k_8 m2(.d_in(d_in[15:8]), .a(a[10:0])), .we(we), .d_out(d_out[15:8]))

    basic_2k_8 m3 (.d_in(d_in[23:16]), .a(a[10:0]), .we(we), .d_out(d_out[23:16]));
 endmodule
```

How 2k is mapped in 8k memory block

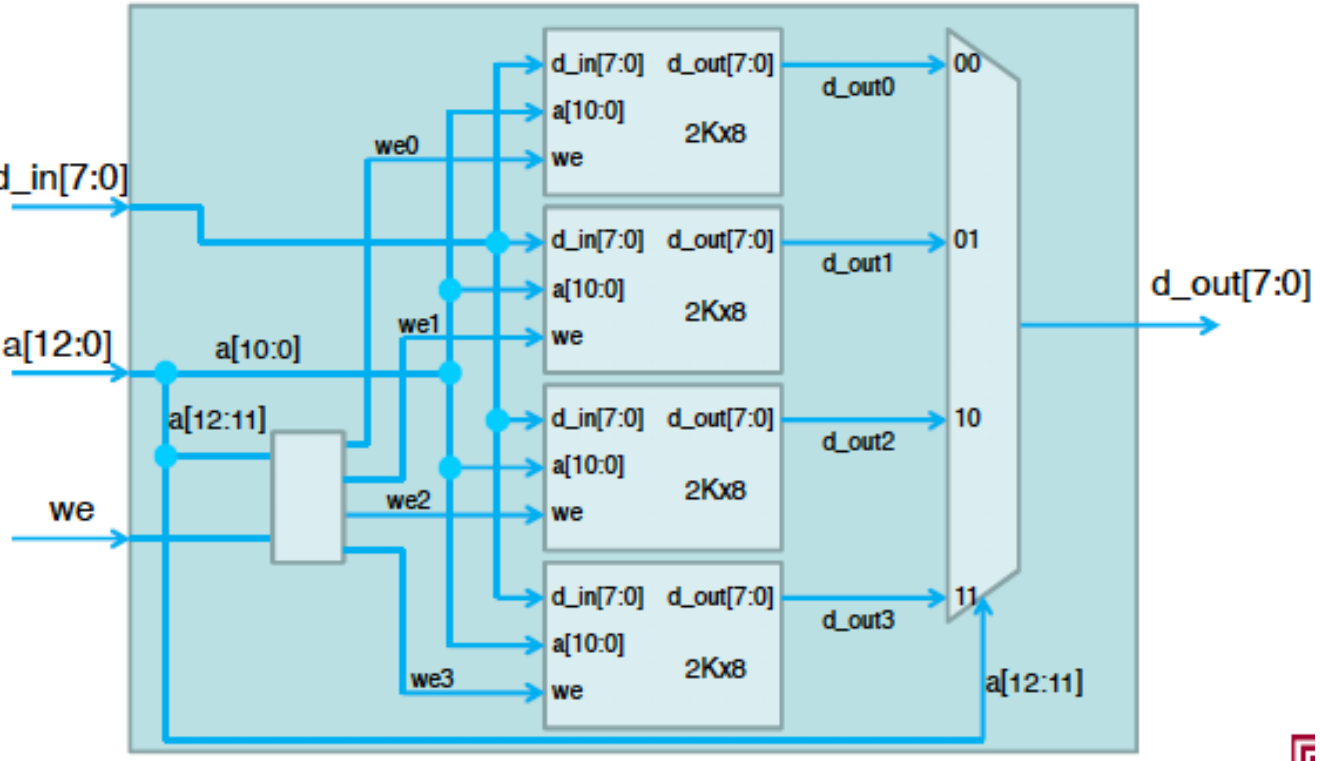
Target design, memory: 8K x 8

Binary Address	Decimal Address
0000000000000	0
0000000000001	1
...	...
0011111111111	2047
0100000000000	2048
0100000000001	2049
...	...
0111111111111	4095
1000000000000	4096
1000000000001	4097
...	...
1011111111111	6143
1100000000000	6144
1100000000001	6145
...	...
1111111111111	8191

Target design, memory: 8K x 8

Binary Address	Decimal Address
0000000000000	0
0000000000001	1
...	...
0011111111111	2047
0100000000000	2048
0100000000001	2049
...	...
0111111111111	4095
1000000000000	4096
1000000000001	4097
...	...
1011111111111	6143
1100000000000	6144
1100000000001	6145
...	...
1111111111111	8191

Address expansion




```
// basic, generic 2K x 8
module basic_2k_8 (output wire [7:0] d_out,
input wire [7:0] d_in, input wire [10:0] a, input wire we);
...
endmodule
```

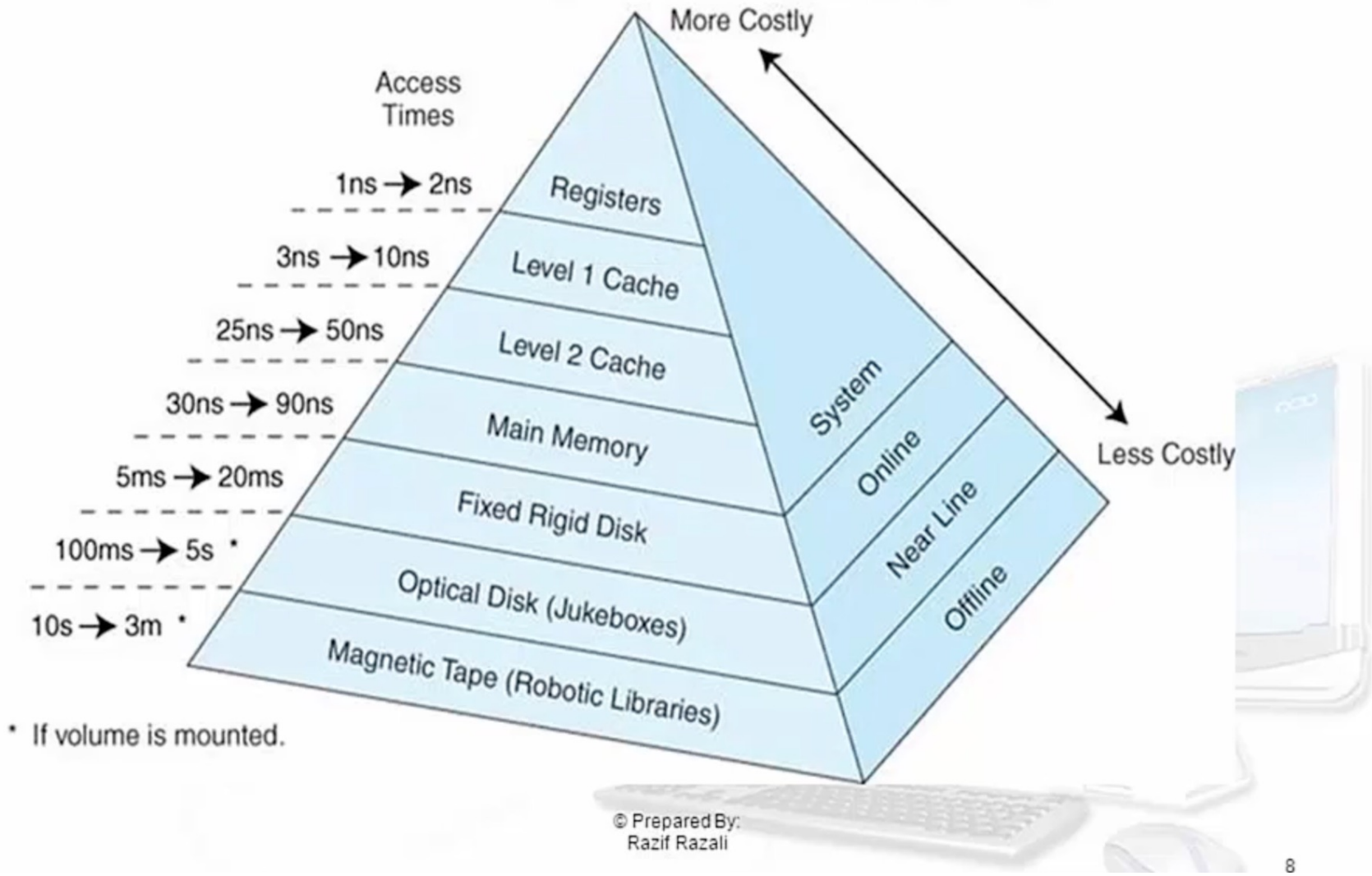
Memory Types

- RAM
- DRAM
- SRAM
- SSRAM
- ASRAM
- ROM
- PROM
- EPROM
- FLASH
- HD
- SSD
- CD/DVD

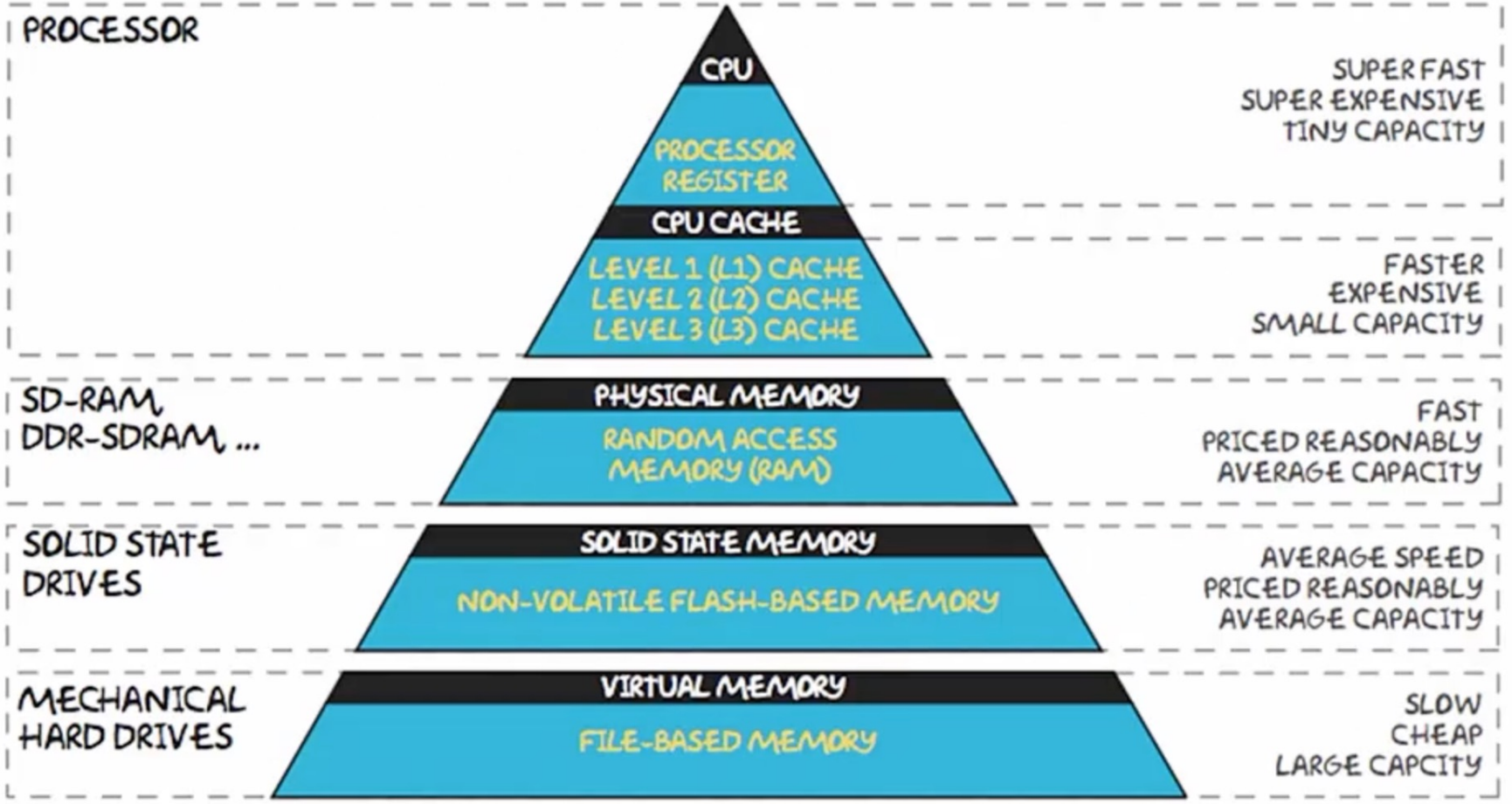
Memory Charaterstics

- Cost – dollars, area, etc.
- Power
- Access time – how fast can you access a word
- Speed – how fast can you access sequential words
- Volatile – does the memory retain its data when power is removed?

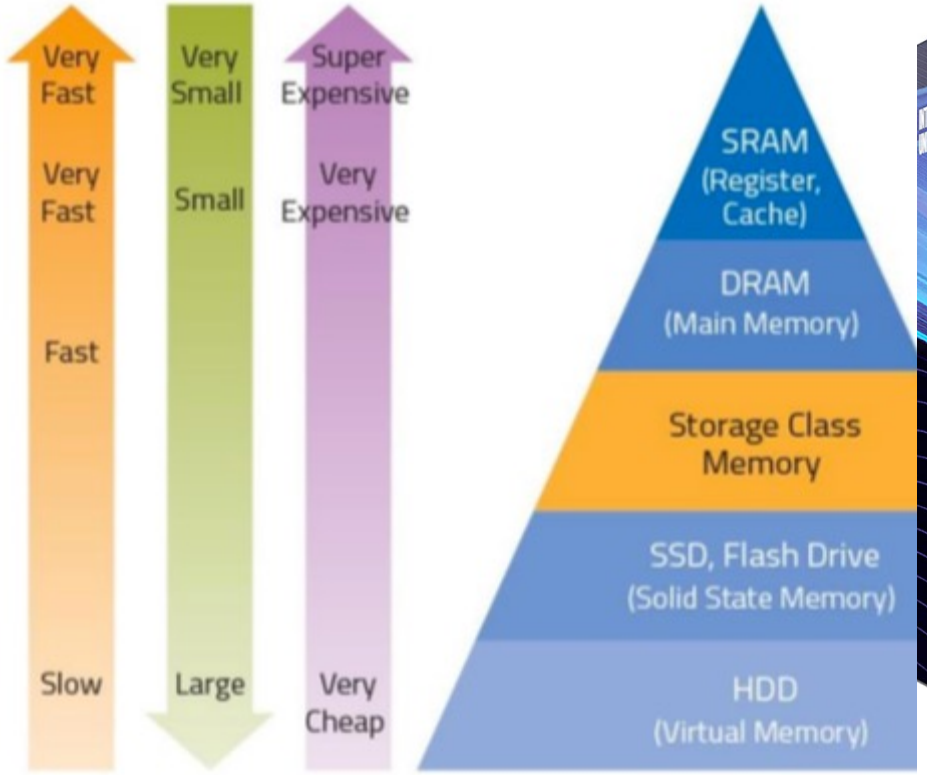
Memory hierarchy pyramid



THE MEMORY HIERARCHY



Memory hierarchy



amazon.com/Intel-i9-9900K-Desktop-Processor-Unlocked/dp/B005404P9I

Roll over image to zoom in

Pay $33.33/month for 12 months, interest-free with your Amazon Prime Rewards Visa Card

Not eligible for Amazon Prime. Available with free Prime shipping from other sellers on Amazon.

Style: **9900K**

9900K
$399.97

Processor w/ motherboard
- -

Brand	Intel
CPU Manufacturer	Intel
CPU Model	Core i9
Processor Speed	5 GHz
Processor Socket	LGA 1151

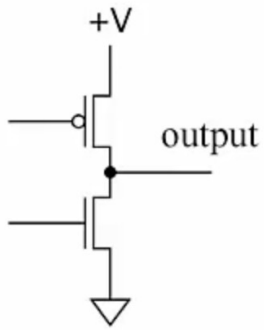
About this item

- 8 Cores / 16 Threads
- 3.60 GHz up to 5.00 GHz / 16 MB Cache
- Compatible only with Motherboards based on Intel 300 Series Chipsets
- Intel Optane Memory Supported
- Intel UHD Graphics 630

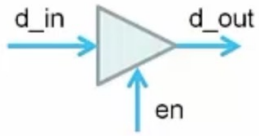
Bus

Tristate Driver

Transistor view



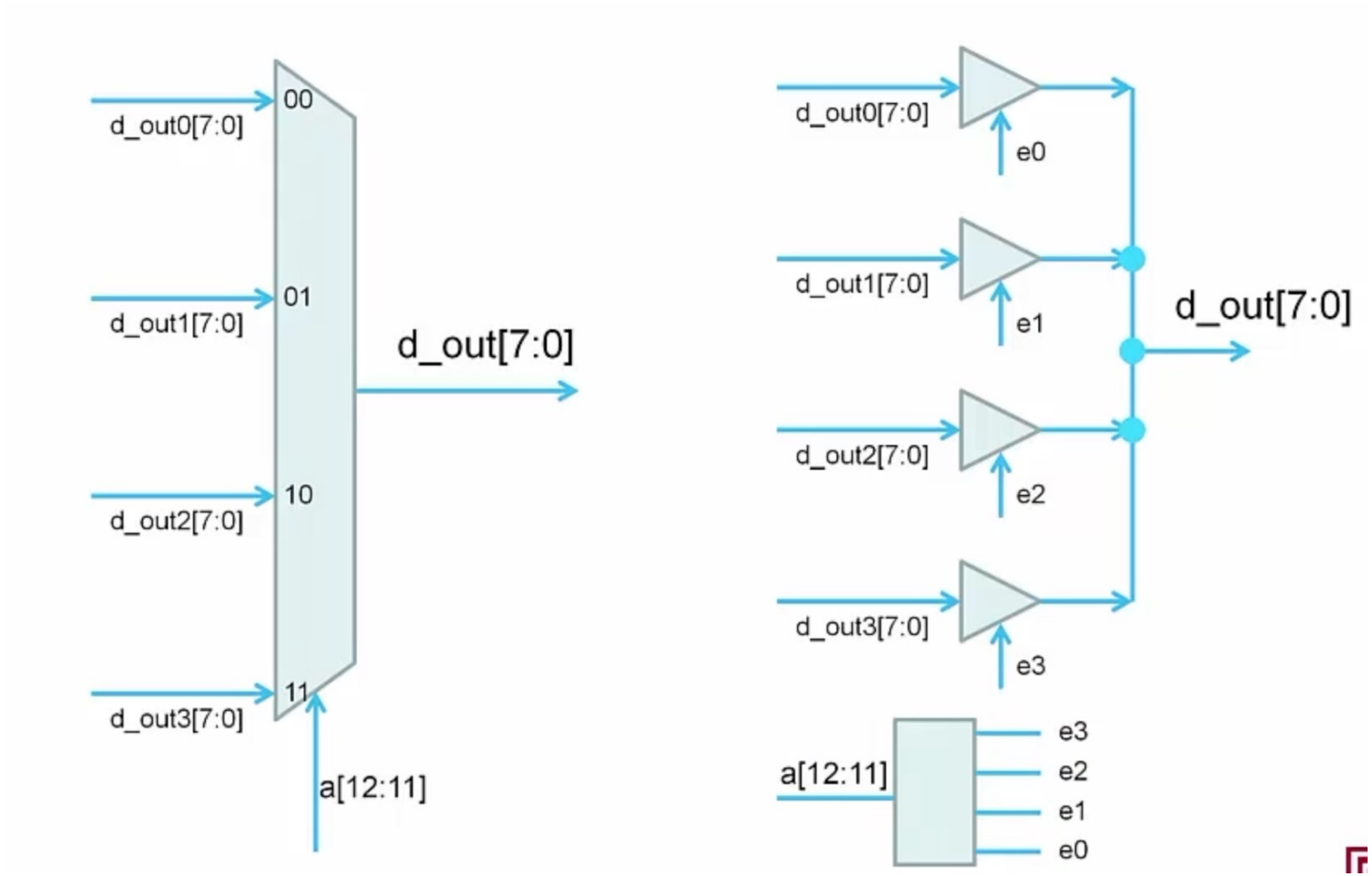
Schematic view



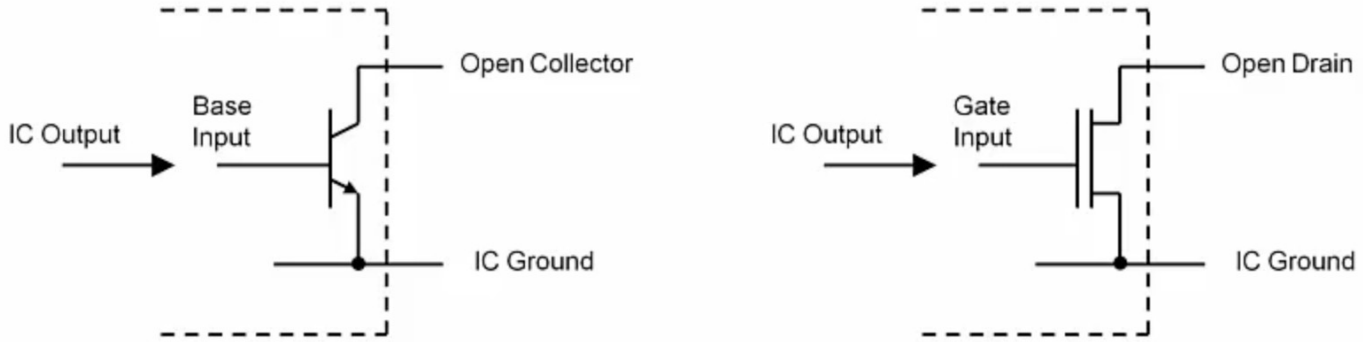

```
assign d_out = en ? d_in : 1'bz;
```

Transistor view		Schematic view		
condition	output	d_in	en	d_out
both mosfets off	Z	x	0	Z
lower mosfet on/upper off	0	0	1	0
upper mosfet on/lower off	1	1	1	1

Multiplexer vs Tristate

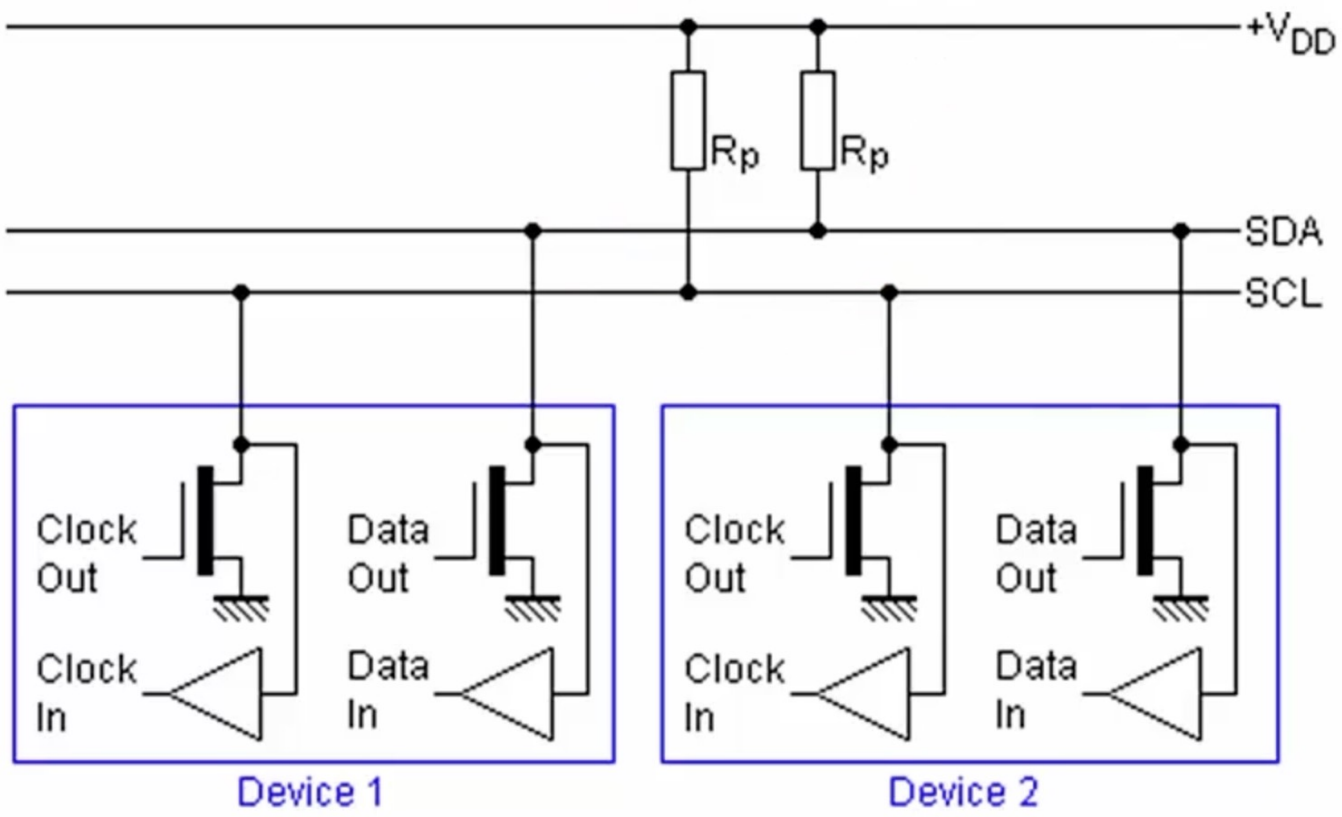


Open Collector and Open Drain



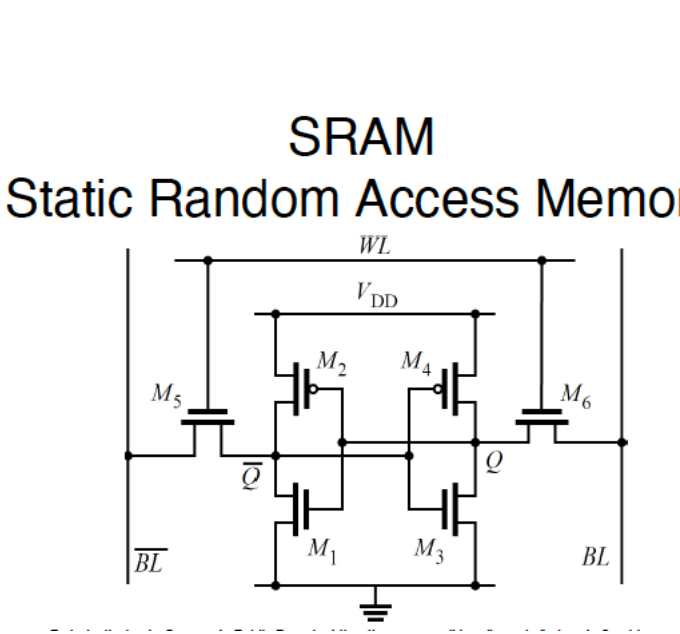
- Signal is inverted
- Need to connect a pull up resistor to output
- Disadvantage – power

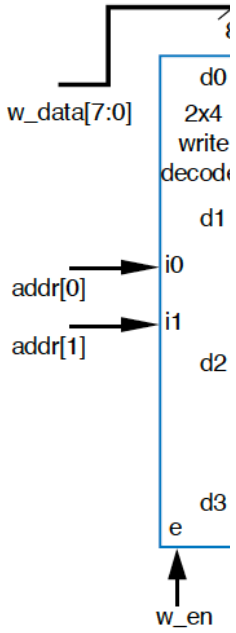
Physical Implementation of I2C bus



http://www.lammertbies.nl/comm/info/I2C-bus.html

SRAM





General Purpose Working Registers

7	0	Addr.	
R0		0x00	
R1		0x01	
R2		0x02	
...			
R13		0x0D	
R14		0x0E	
R15		0x0F	
R16		0x10	
R17		0x11	
...			
R26		0x1A	X-register Low Byte
R27		0x1B	X-register High Byte
R28		0x1C	Y-register Low Byte
R29		0x1D	Y-register High Byte
R30		0x1E	Z-register Low Byte
R31		0x1F	Z-register High Byte

– cells made from d flip flops
– i/o data ports separate and unidirectional
– only write enable control signal

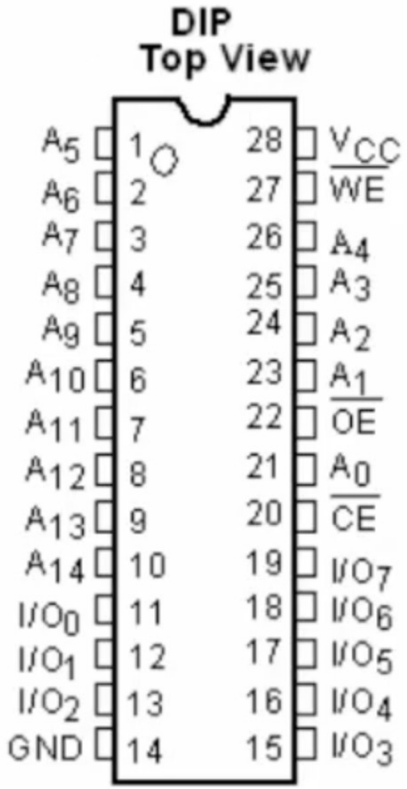
register

SRAM

- Two implementations
  - asynchronous: independent of a clock; data in and out are controlled by address translation
  - synchronous: all timings are controlled by a clock edge; address, data in and out are synchronous with a clock

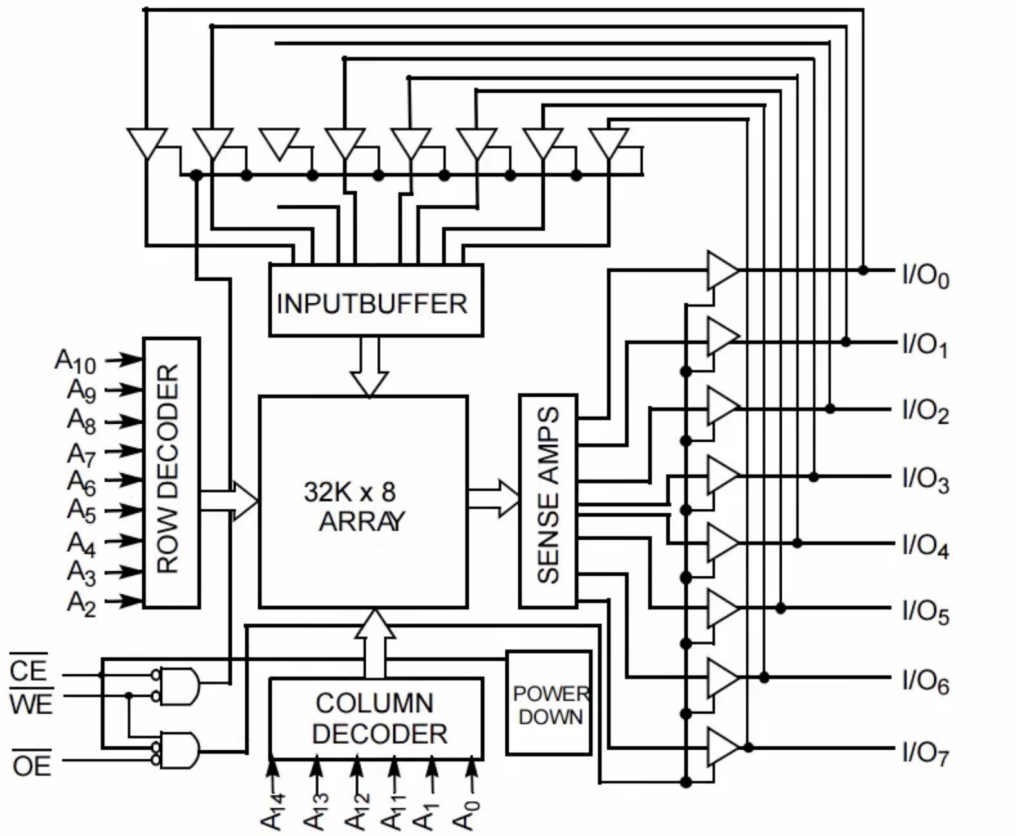
CY62256N

256-Kbit (32 K × 8) Static RAM



SRAM – schematics

Logic Block Diagram



Timing Diagram for read operation

Figure 5. Read Cycle No. 1 [13, 14]

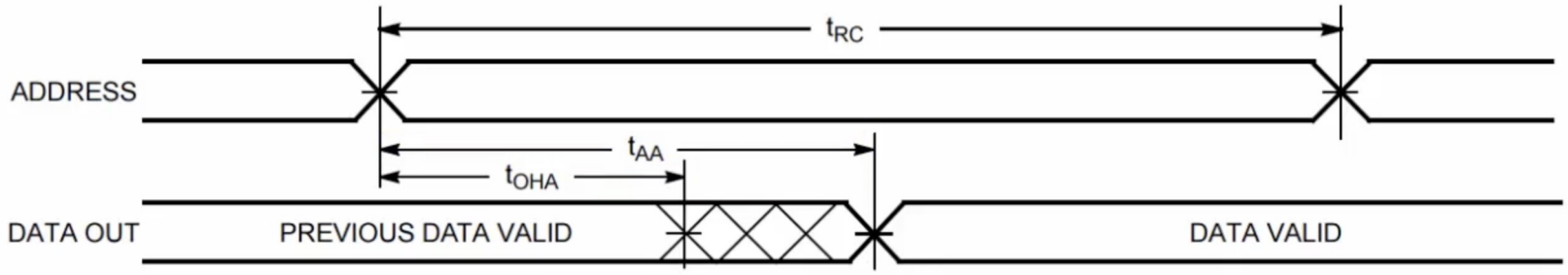
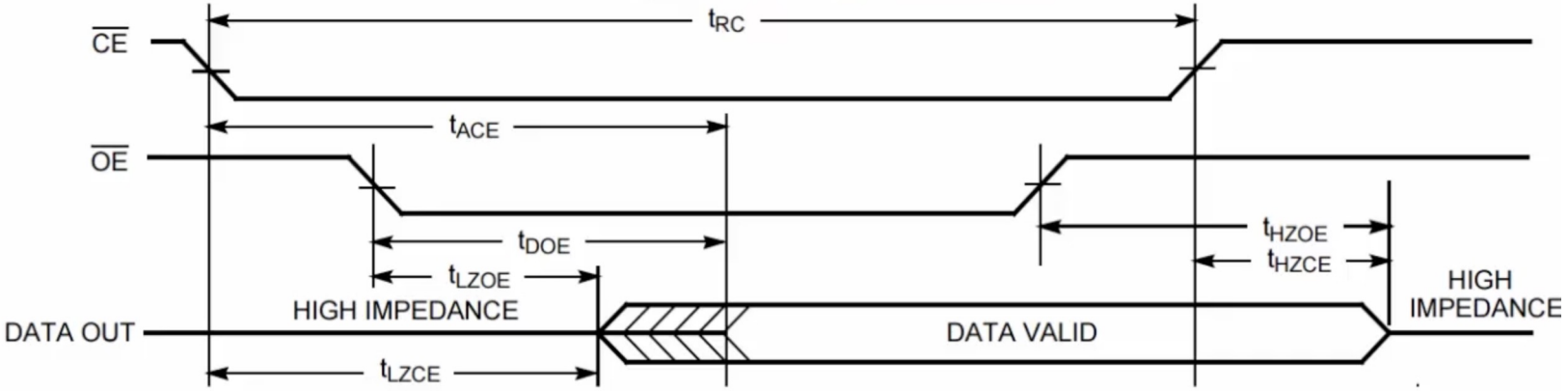
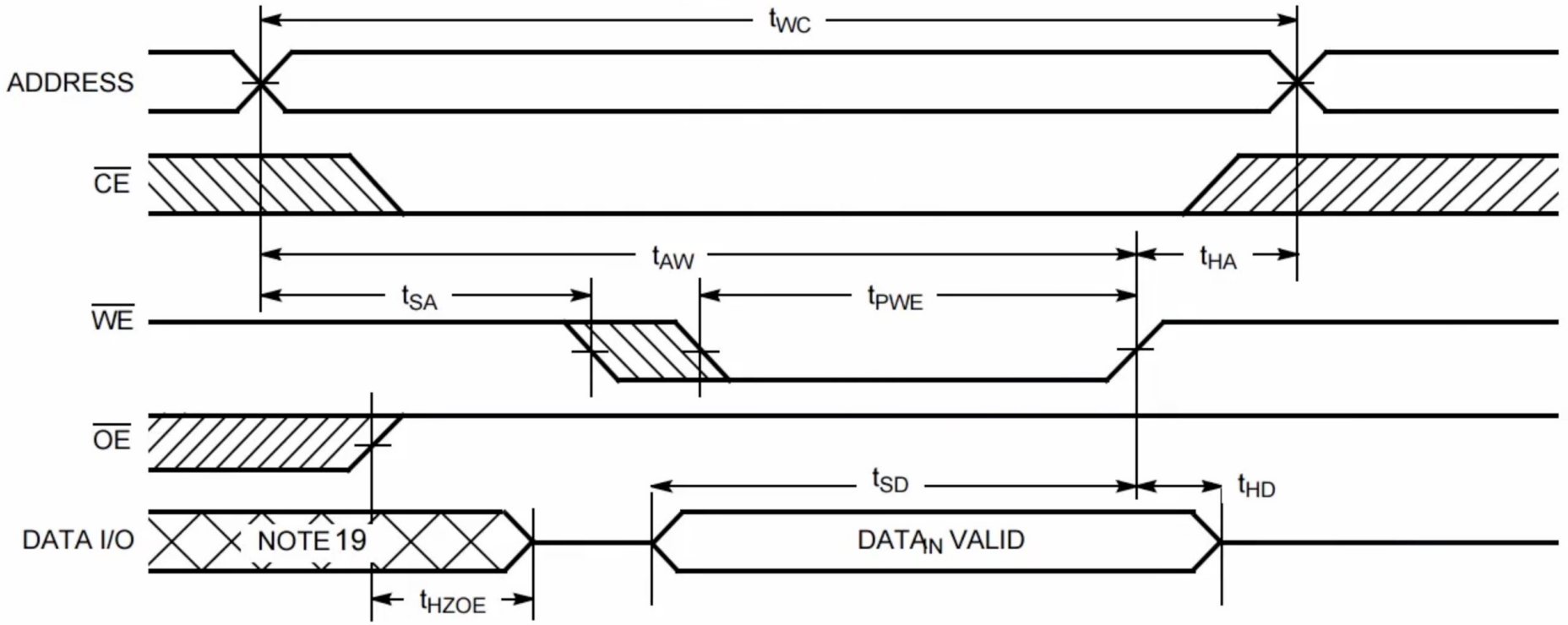


Figure 6. Read Cycle No. 2 [14, 15]

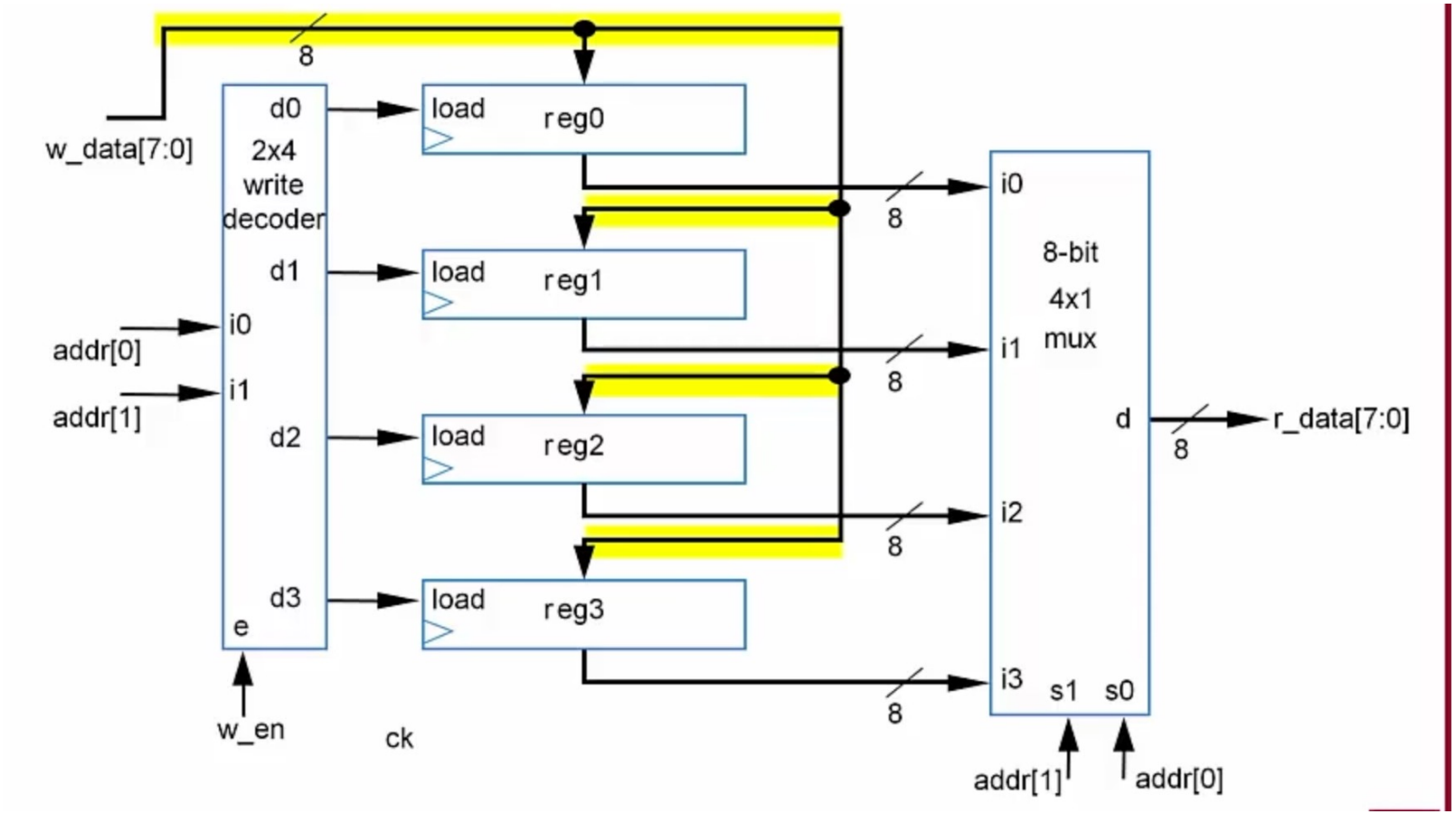


Timing Diagram for write Operation

Figure 7. Write Cycle No. 1 (WE Controlled) [16, 17, 18]



SRAM – 4x8 bit using tri-state



SRAM – 4x8-bit

```
module reg_file_4x8 (
  output [7:0] r_data, input ck,
  input w_en, input [1:0] addr,
  input [7:0] w_data);

  reg [7:0] q_regfile [0:3];

  // synchronous logic
  always @(posedge ck) begin
    if (w_en == 1) begin
      q_regfile[addr] <= w_data;
    end
  end
```

```
  // combinational block
  always @* begin
    // one way
    case (addr)
    2'b00: r_data = q_regfile[0];
    2'b01: r_data = q_regfile[1];
    2'b10: r_data = q_regfile[2];
    2'b11: r_data = q_regfile[3];
    endcase;

    // or shorthand
    r_data = q_regfile[addr];
  end

  // or just a simple assign
  assign r_data =
    q_regfile[addr];
endmodule
```

Skimpily as ...

```
module reg_file_4x8 (
  output [7:0] r_data, input ck,
  input w_en, input [1:0] addr,
  input [7:0] w_data);

  reg [7:0] q_regfile [0:3];

  always @(posedge ck) begin // synchronous logic
    if (w_en == 1) begin
      q_regfile[addr] <= w_data;
    end
  end

  assign r_data = q_regfile[addr];  // read port (combinational)

endmodule
```

Homework

3. Using the basic, generic 2K x 8 memory device below, use 4 of them to create a larger memory block that is 4K x 16. Draw a block diagram/schematic for your design, then a SystemVerilog design, using the target_4k_16 as the target memory module.

```
// basic, generic 2K x 8

module basic_2k_8 (output logic [7:0] d_out,

  input logic [7:0] d_in, input logic [10:0] a, input logic we);

...

endmodule: basic_2k_8


// target, 4K x 16

module target_4k_16 (output logic [15:0] d_out,

  input logic [15:0] d_in, input logic [11:0] a, input logic we);


endmodule: target_4k_16
```