

Lecture 15

Memories

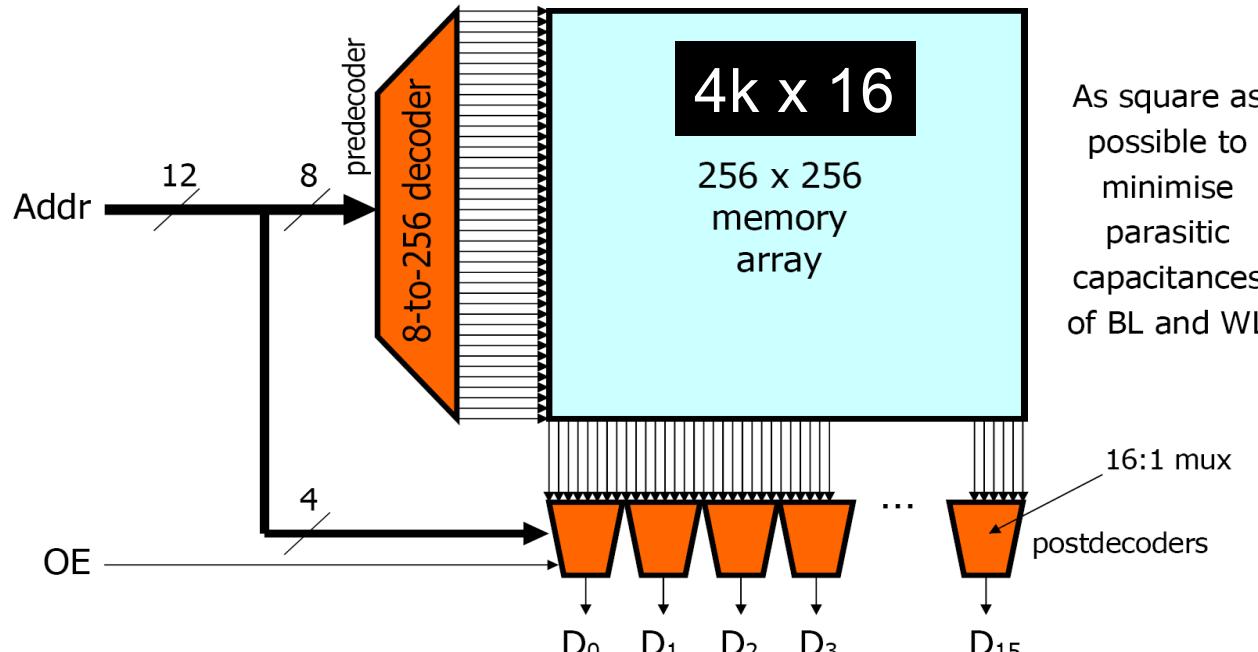
Peter Cheung
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/E2_CAS/
E-mail: p.cheung@imperial.ac.uk

Lecture Objectives

- ◆ Explain the sequence of events in reading from and writing to a static RAM
- ◆ Explain the structure and input/output signals of a static RAM
- ◆ How to design an address decoder
- ◆ Investigate the timing diagrams for a microprocessor when reading from or writing to memory
- ◆ Explain how the embedded memory in an FPGA can be used to implement memory blocks in a digital design

Simplified RAM Organization



A typical 8-bit microprocessor typically has

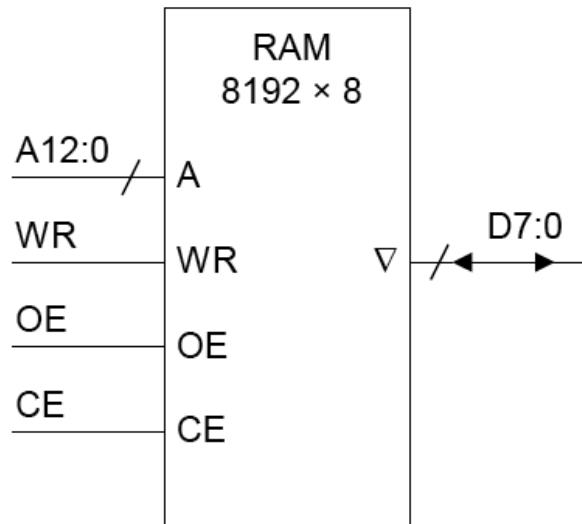
- ◆ A 16-bit address bus, A15:0
 - Can have up to $2^{16} = 65536$ memory locations
- ◆ An 8-bit data bus, D7:0 - Each data word in memory has $2^8 = 256$ possible values
- ◆ In the RAM shown above uses 12-bit address and 16-bit data, i.e. 4096 locations of 16-bits each
- ◆ These are arranged as 256 x 256 rows of memory cells. $4096 = 256$ rows x 16 columns as shown
- ◆ The address bus is therefore split into two components: 8-bit to specify which row, and 4-bit to select the correct column.

A typical memory has:

- ◆ N-bit address bus
 - This defines 2^N memory locations
- ◆ M-bit data bus
 - This defines the size of data word
- ◆ Control signals to define read/write cycles,
 - Asynchronous memory – no clock in control
 - Synchronous memory – every action synchronised to a clock

RAM: Read/Write Memory

8k × 8 Static RAM



CE	OE	WR	D0:7	Action
0	?	?	Hi Z	Disabled
1	0	0	Hi Z	Idle
1	1	0	Out	Read
1	?	1	In	Write

Hi Z = High impedance

Static RAM: Data stored in bistable latches

Dynamic RAM: Data stored in charged capacitors:
retained for only 2ms.
Less circuitry \Rightarrow denser \Rightarrow cheaper.

▼ Tri-state output: Low, High or Off (High Impedance).
Allows outputs from several chips to be connected;
Designer must ensure only one is enabled at a time.

CE Chip Enable: disabling chip cuts power by 80%.

OE Output Enable: Turns the tri-state outputs on/off.

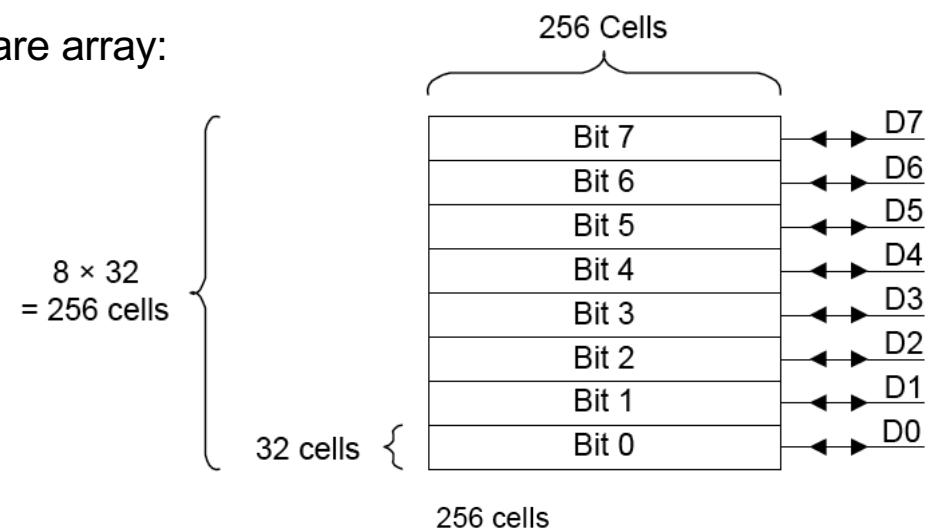
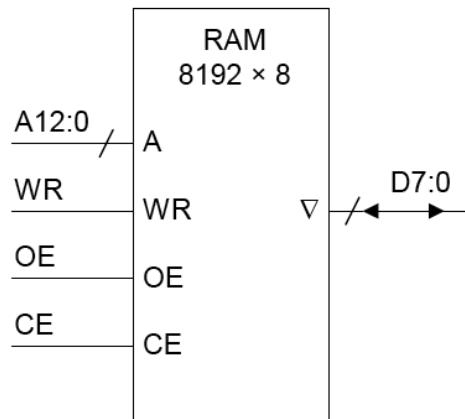
A12:0 Address: selects one of the 2^{13} 8-bit locations.

WR Write: stores new data in selected location

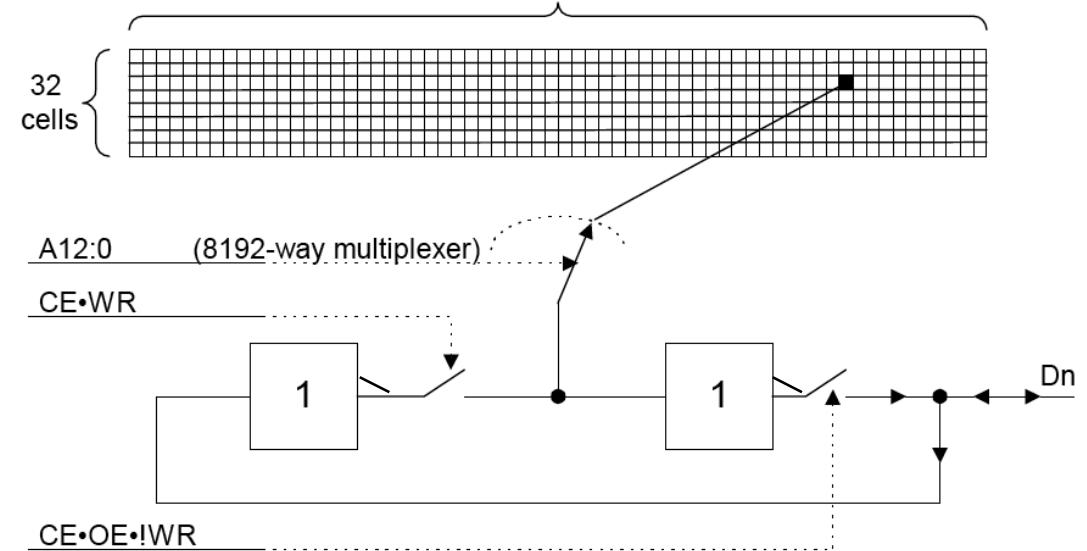
D7:0 Data in for write cycles or out for read cycles.

8k x 8 Static RAM

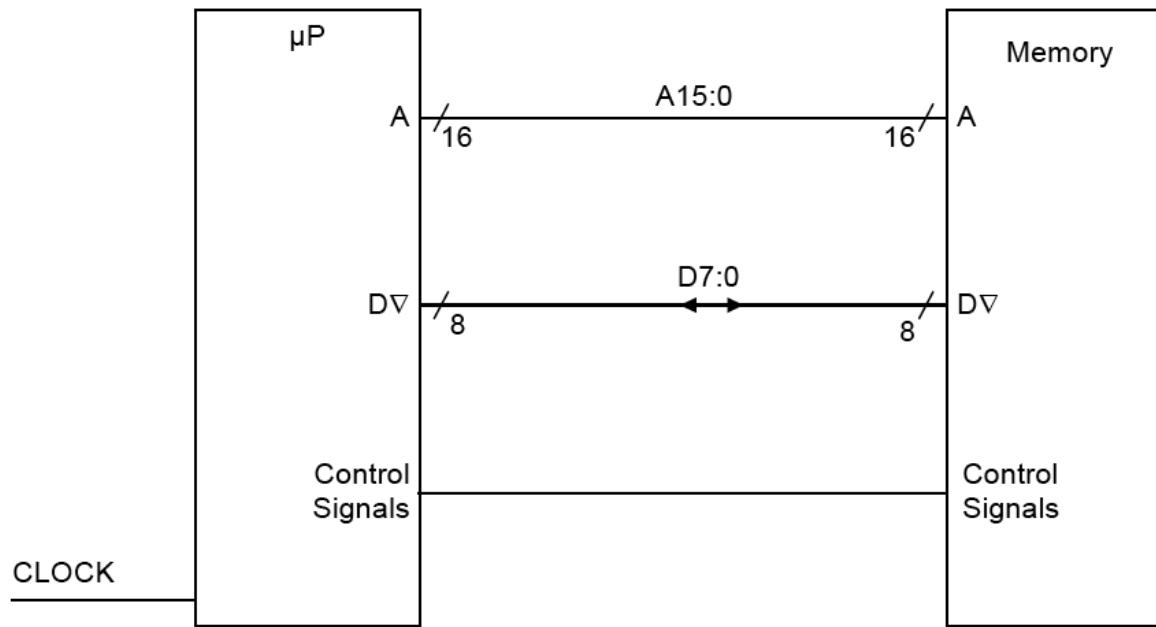
- ◆ The 64k memory cells are arranged in a square array:



- ◆ For each output bit, an 8192-way multiplexer selects one of the cells. The control signals, **OE**, **CE** and **WR** determine how it connects to the output pin via buffers:
- ◆ Occasionally DIN and DOUT are separate but \Rightarrow more pins

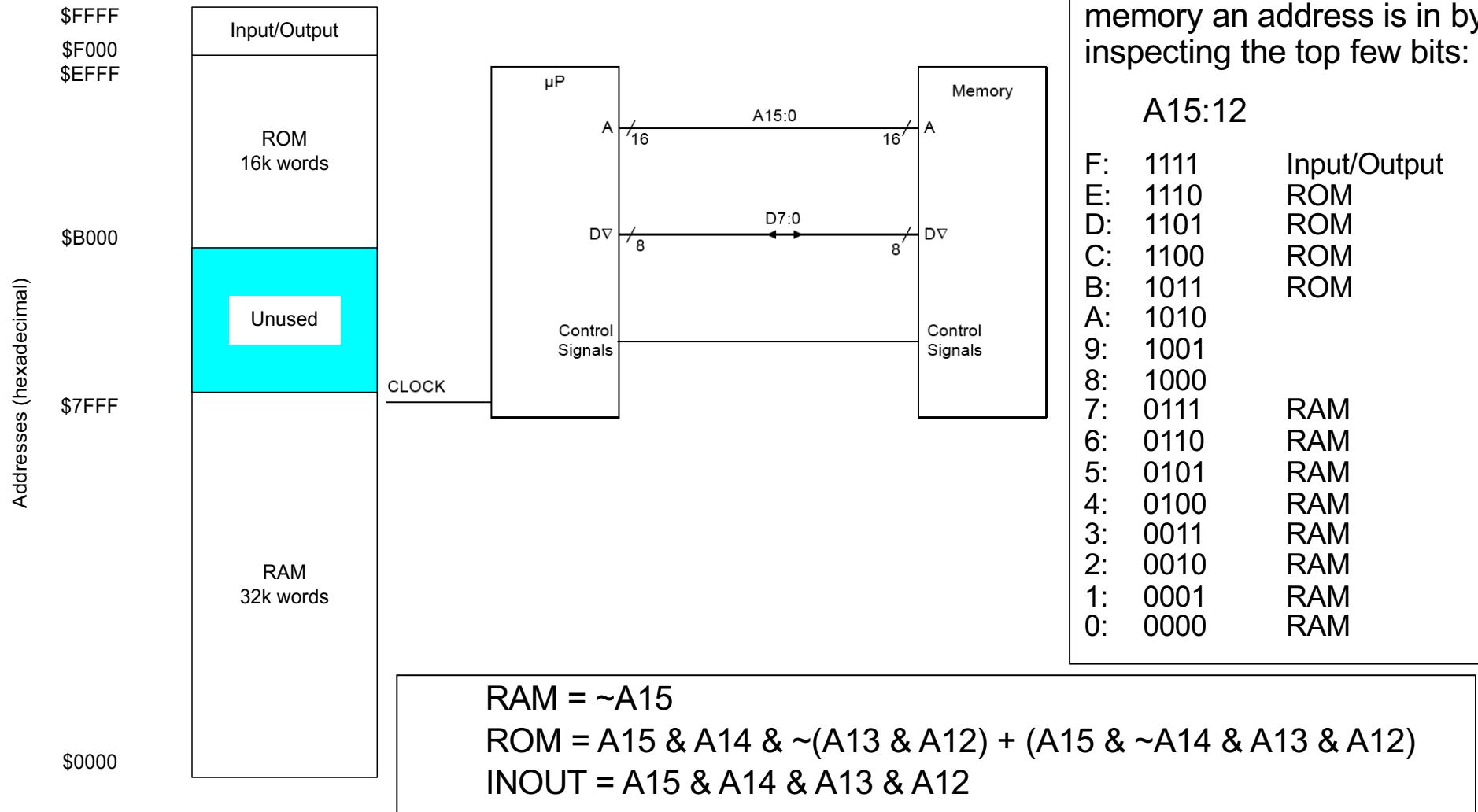


Microprocessor \leftrightarrow Memory Interface



- ◆ During each memory cycle:
- ◆ A15:0 selects one of 2^{16} possible memory locations
- ◆ D7:0 transfer one word (8 bits) of information either to the memory (write) or to the microprocessor (read).
- ◆ D7:0 connections to the microprocessor are tri-state (∇): they can be:
 - “logic 0”, “logic 1” or “high impedance” (inputs)
- ◆ The control signals tell the memory what to do and when to do it.

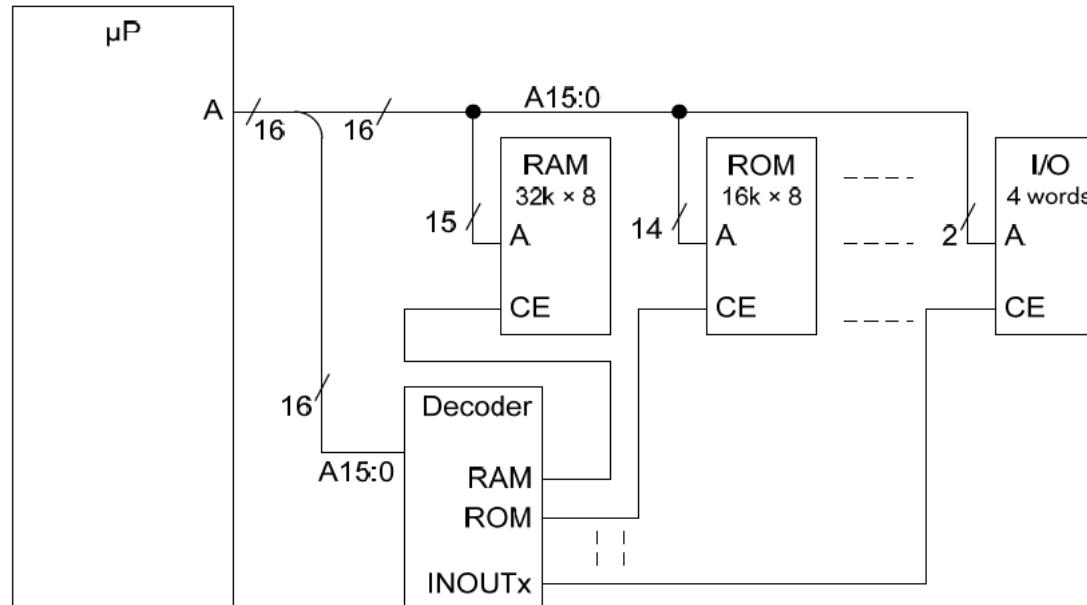
Microprocessor Memory Map



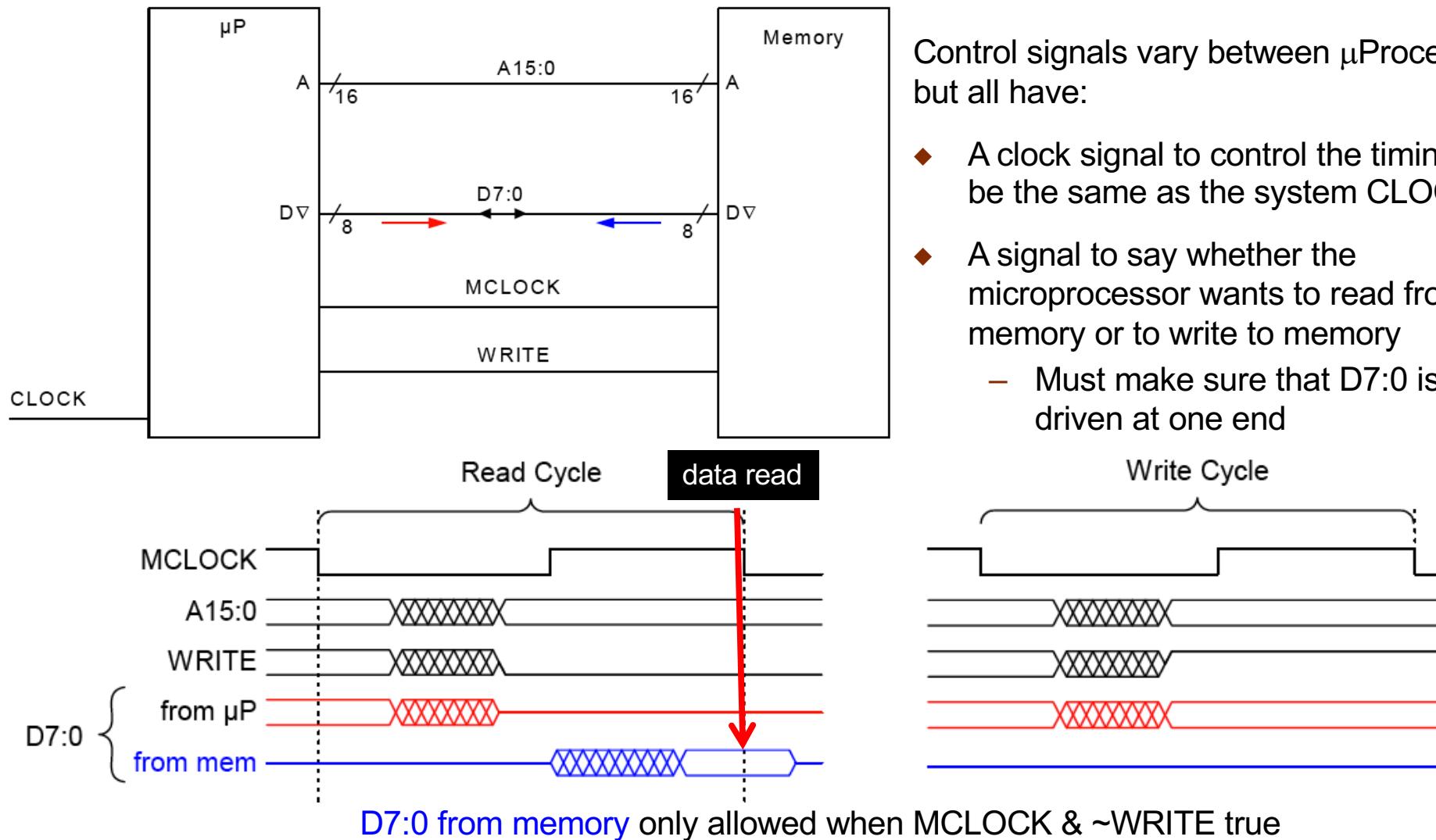
Memory Chip Selection

- ◆ Each memory circuit has a “chip enable” input (CE)
- ◆ The “Decoder” uses the top few address bits to decide which memory circuit should be enabled. Each one is enabled only for the correct address range:
 - RAM = $\sim A_{15}$
 - ROM = $A_{15} \& A_{14} \& \sim(A_{13} \& A_{12}) + (A_{15} \& \sim A_{14} \& A_{13} \& A_{12})$
 - INOUTx = **$A_{15} \& A_{14} \& A_{13} \& A_{12} \& \sim A_{11} \& A_{10} \& \sim A_9 \& A_8 \& \sim A_7 \& A_6 \& A_5 \& A_4 \& \sim A_3 \& A_2$**
- ◆ INOUTx responds to addresses: \$F574 to \$F577
other I/O circuits will have different addresses
- ◆ Low n address bits select one of 2^n locations within each memory circuit (value of n depends on memory size)

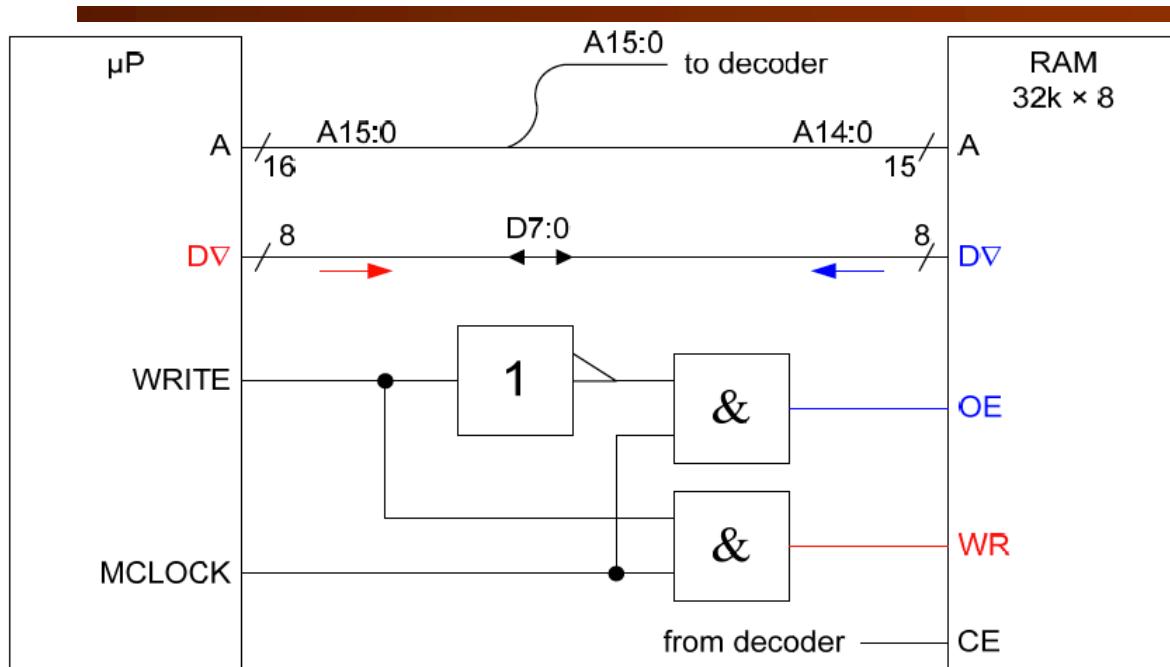
Addr Range	Usage
\$F578 - \$FFFF	Not used
\$F574 - \$F577	INOUTx
\$F000 - \$F573	Not used
\$B000 - \$EFFF	16k ROM
\$8000 - \$AFFF	Not used
\$0000 - \$7FFF	32k RAM



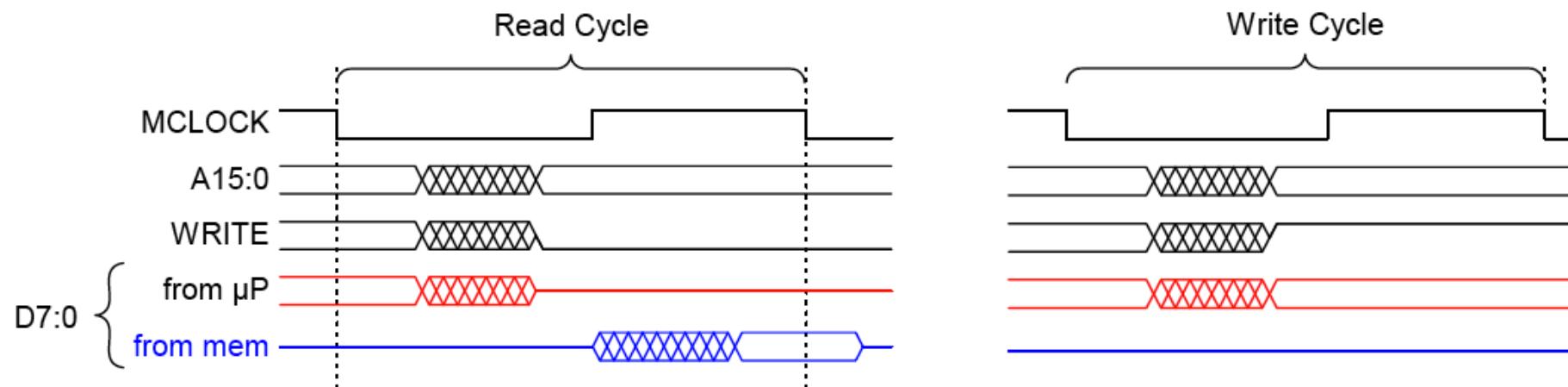
Memory Interface Control Signals



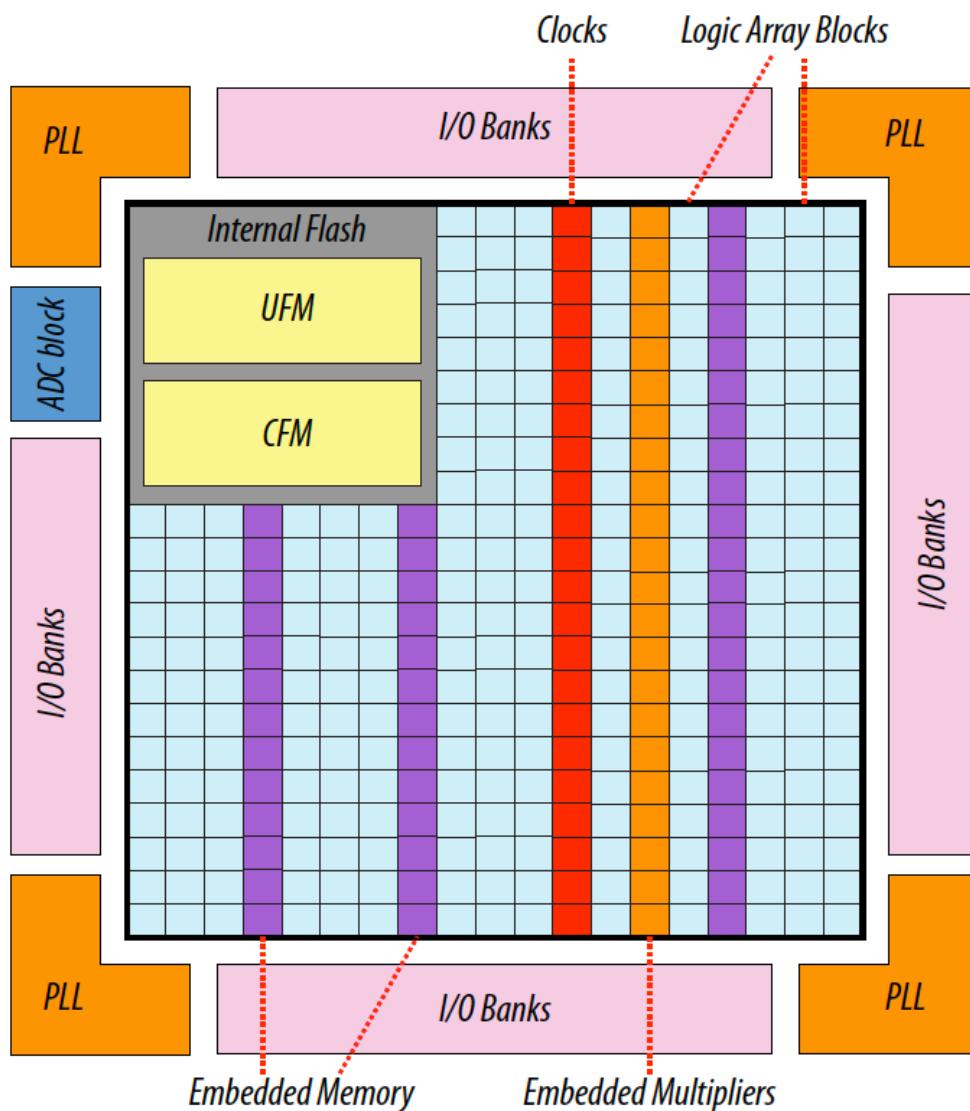
Memory Circuit Control Signals



- ◆ **Output enable:** $OE = MCLOCK \& \sim WRITE$ turns on the $D7:0$ output from the memory
- ◆ **Write enable:** $WE = MCLOCK \& WRITE$ writes new information into the selected memory location with data coming from microprocessor
- ◆ **Chip enable:** comes from the decoder and makes sure the memory only responds to the correct addresses



Max 10 FPGA – Embedded blocks



- ❖ MAX10 device: 10M50DAF484C7G
- ❖ 50,000 Logic Elements (4-LUT + FF)
- ❖ 182 M9K embedded memory blocks
- ❖ 5,888 kbits user flash memory
- ❖ 144 hard multiplier (18 x 18)
- ❖ 4 PLL (for clock generation)

MAX 10 Embedded Memory

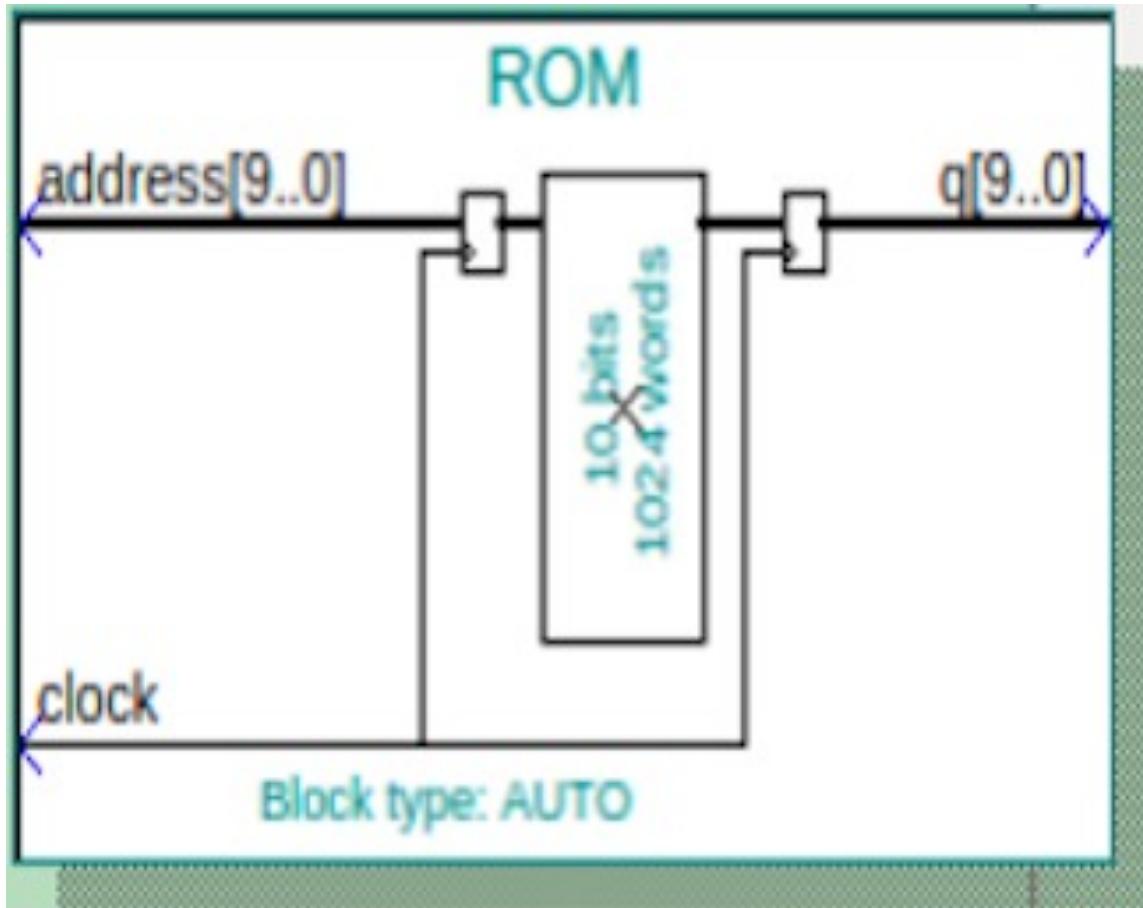
- ◆ Each 9kbit memory block (M9K) can be configured with different data width from 1 bit to 36 bit wide
- ◆ It also has multiple operating modes (which is user configurable), of which we will focus on the following only: 1-port ROM, FIFO, 2-port RAM

- Single-port
- Simple dual-port
- True dual-port
- Shift-register
- ROM
- FIFO

Configuration	M9K Block
Depth × width	8192 × 1
	4096 × 2
	2048 × 4
	1024 × 8
	1024 × 9
	512 × 16
	512 × 18
	256 × 32
	256 × 36

Initialization of ROM Contents (1k x 8)

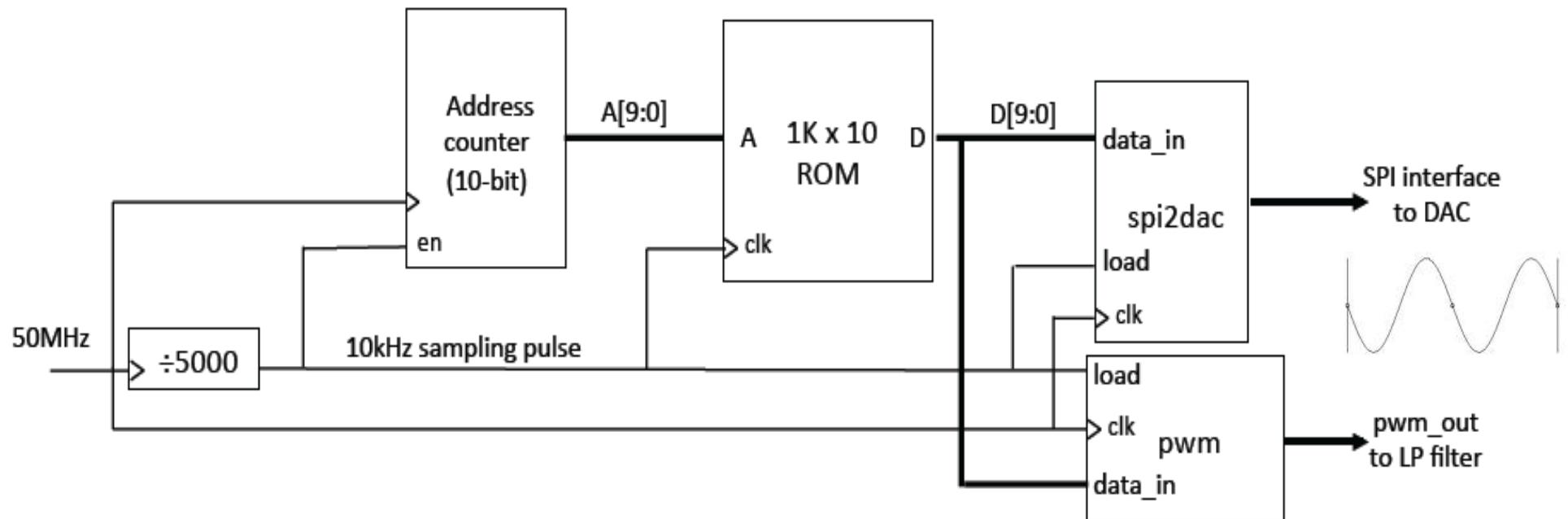
- ◆ Create ROM and initialize its content in a .mif file:



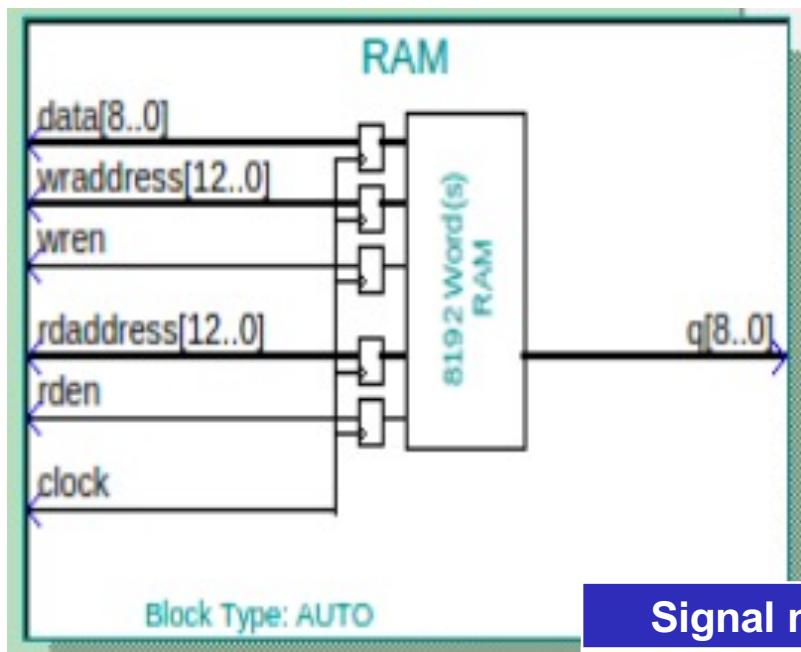
```
-- ROM Initialization file
WIDTH = 10;
DEPTH = 1024;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN
  0 : 200;
  1 : 203;
  2 : 206;
  3 : 209;
  4 : 20C;
  5 : 20F;
  6 : 212;
  7 : 215;
  8 : 219;
  9 : 21C;
  A : 21F;
```

Sinewave Generation

- ◆ Generate any waveform or function $y = F(x)$ using table lookup
- ◆ Phase counter increment phase whenever *step* goes high
- ◆ ROM stores one cycle of sinewave to produce $F(x)$
- ◆ Digital-to-Analogue convert and the PWM DAC generate the analogue outputs on L & R channels



Dual-port RAM (8k x 9)

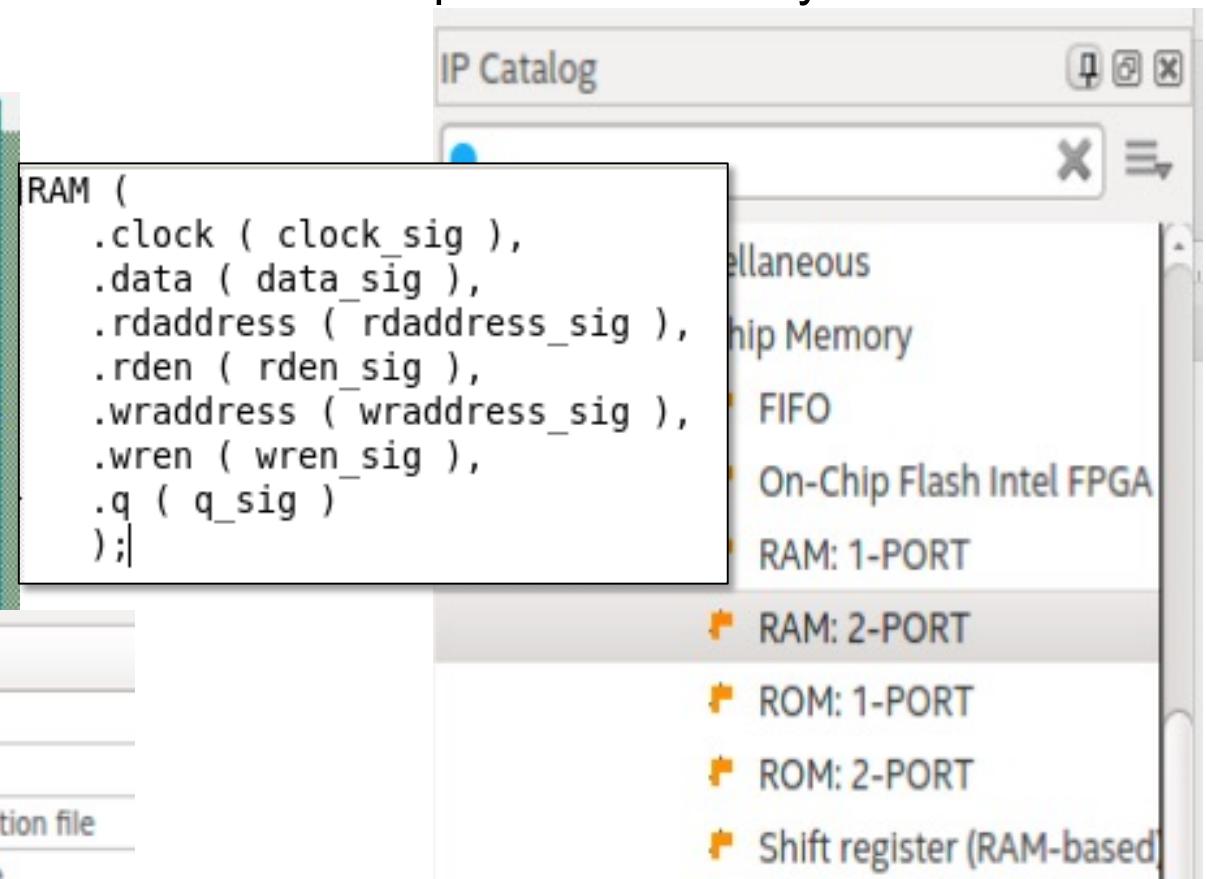
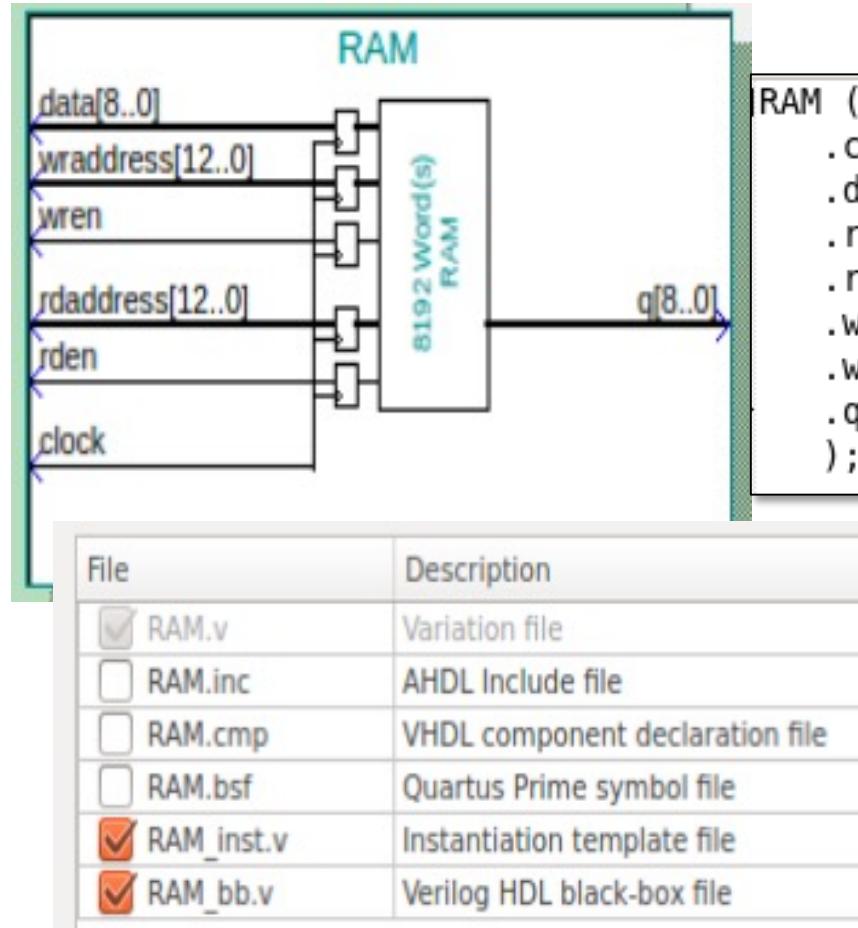


- ◆ Limited data width (1 to 9, 16, 18, 32, 36 ...)
- ◆ Depth up to 65,536 (4-bit to 16-bit address)
- ◆ **Simple dual-port: 1 read port & 1 write port**
- ◆ True dual-port: both ports can read/write

Signal name	meaning
data[]	Write data port
address[]	Read/write address port
q[]	Read data port
wren	Write enable
rden	Read enable
clock	Clock signal to control both read & write

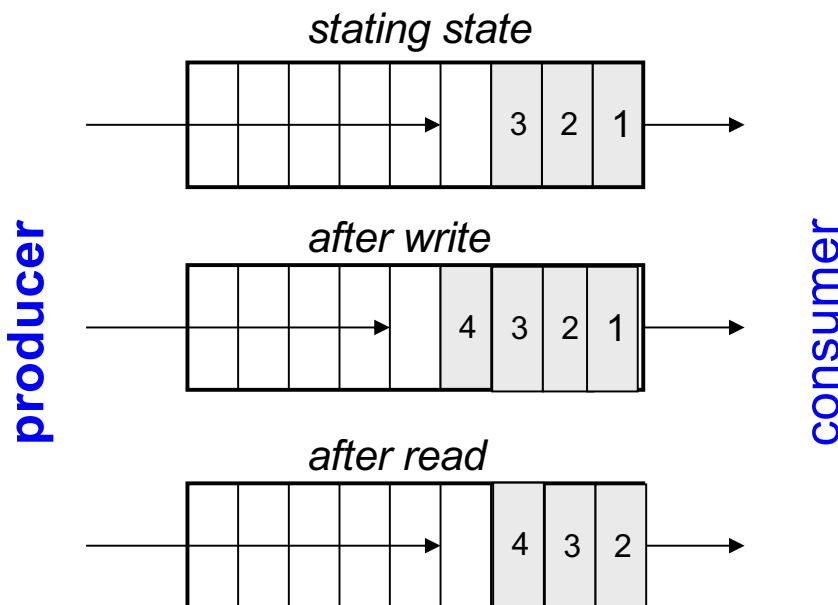
How to use M9K memory block? (8k x 9)

- ◆ Use IP Catalog manager tool in Quartus to produce memory of the correct configuration:



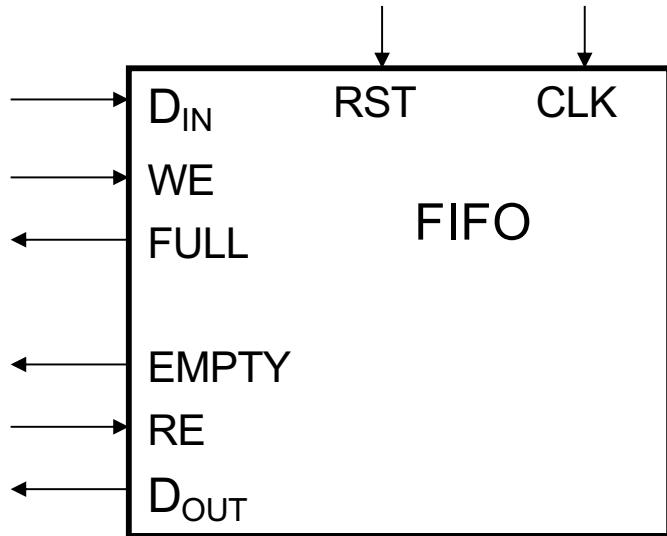
First-in-first-out (FIFO) Memory

- ◆ Used to implement *queues*.
- ◆ These find common use in computers and communication circuits.
- ◆ Generally, used for rate matching data producer and consumer:



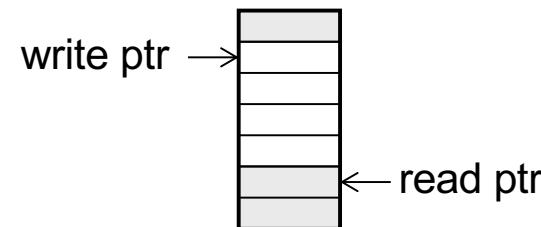
- ◆ Producer can perform many writes without consumer performing any reads (or vice versa). However, because of finite buffer size, on average, need equal number of reads and writes.
- ◆ Typical uses:
 - interfacing I/O devices. Example network interface. Data bursts from network, then processor bursts to memory buffer (or reads one word at a time from interface). Operations not synchronized.
 - Example: Audio output. Processor produces output samples in bursts (during process swap-in time). Audio DAC clocks it out at constant sample rate.

FIFO Interfaces

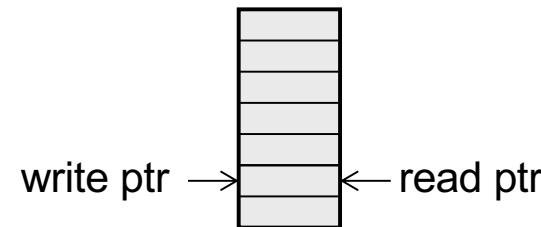


- ◆ After write or read operation, FULL and EMPTY indicate status of buffer.
- ◆ Used by external logic to control own reading from or writing to the buffer.
- ◆ FIFO resets to EMPTY state.

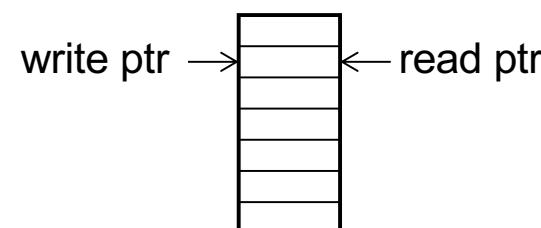
- ◆ Address pointers are used internally to keep next write position and next read position into a dual-port memory.



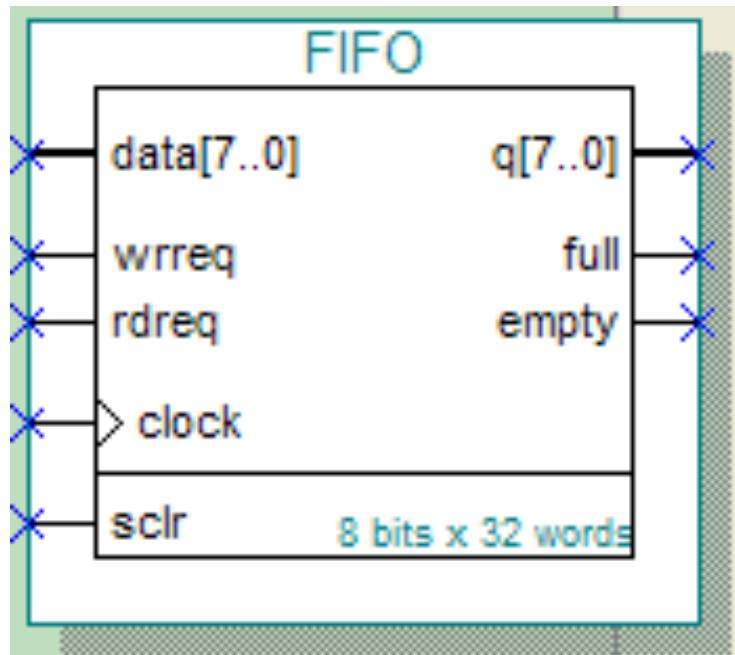
- ◆ If pointers equal after write \Rightarrow FULL:



- ◆ If pointers equal after read \Rightarrow EMPTY:



M9K Memory as FIFO (8-bit x 32 word)



```
module FIFO (
    clock,
    data,
    rdreq,
    sclr,
    wrreq,
    empty,
    full,
    q);

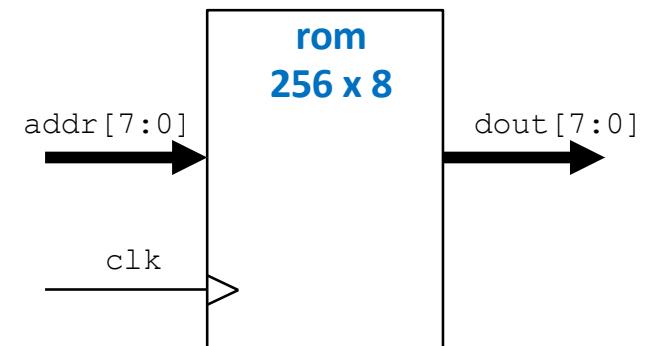
    input      clock;
    input      [7:0] data;
    input      rdreq;
    input      sclr;
    input      wrreq;
    output     empty;
    output     full;
    output      [7:0] q;

endmodule
```

System Verilog specification of 256 x 8 ROM

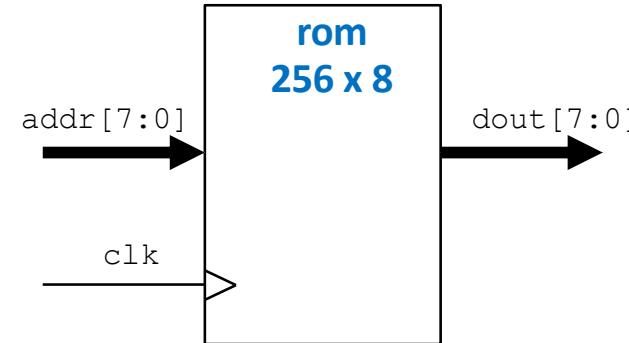
```
1 module rom #(  
2     parameter ADDRESS_WIDTH = 8,  
3             DATA_WIDTH = 8  
4 )(  
5     input logic                 clk,  
6     input logic [ADDRESS_WIDTH-1:0] addr,  
7     output logic [DATA_WIDTH-1:0]   dout  
8 );  
9  
10 logic [DATA_WIDTH-1:0] rom_array [2**ADDRESS_WIDTH-1:0];  
11  
12 initial begin  
13     $display("Loading rom.");  
14     $readmemh("sinerom.mem", rom_array);  
15 end;  
16  
17 always_ff @(posedge clk)  
18     // output is synchronous  
19     dout <= rom_array [addr];  
20  
21 endmodule
```

Verilator gives a warning unless you add an extra line!!!!!



Initialization of the ROM

```
12 initial begin
13     $display("Loading rom.");
14     $readmemh("sinerom.mem", rom_array);
15 end;
```



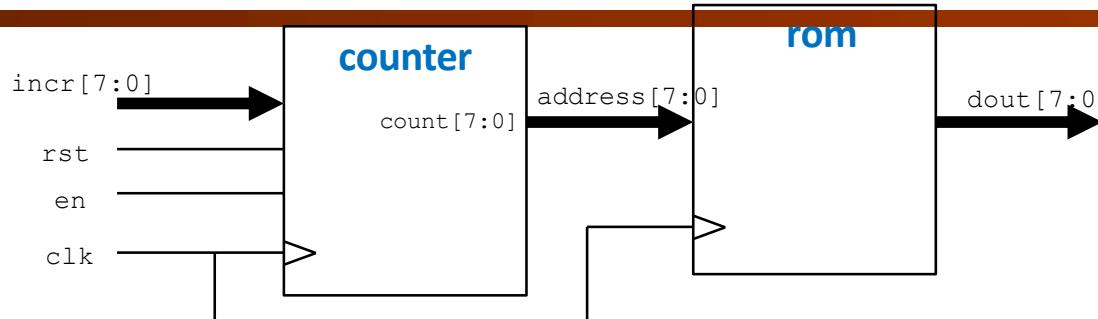
sinegen.py

```
1 import math
2 import string
3 f = open("sinerom.mem", "w")
4 for i in range(256):
5     v = int(math.cos(2*3.1416*i/256)*127+127)
6     if (i+1)%16 == 0:
7         s = "{hex:2X}\n"
8     else:
9         s = "{hex:2X} "
10    f.write(s.format(hex=v))
11
12 f.close()
```

sinerom.mem

1	FE	FD	FD	FD	FD	FC	FC	FB	FA	FA	F9	F8	F7	F6	F5	
2	F4	F3	F1	F0	EF	ED	EB	EA	E8	E6	E5	E3	E1	DF	DD	DA
3	D8	D6	D4	D1	CF	CD	CA	C8	C5	C2	C0	BD	BA	B8	B5	B2
4	AF	AC	A9	A6	A3	A0	9D	9A	97	94	91	8E	8B	88	85	82
5	7E	7B	78	75	72	6F	6C	69	66	63	60	5D	5A	57	54	51
6	4E	4B	48	45	43	40	3D	3B	38	35	33	30	2E	2C	29	27
7	25	23	20	1E	1C	1A	18	17	15	13	12	10	E	D	C	A
8	9	8	7	6	5	4	3	3	2	1	1	0	0	0	0	0
9	0	0	0	0	0	0	1	1	2	3	3	4	5	6	7	8
10	9	A	C	D	E	10	12	13	15	17	18	1A	1C	1E	20	23
11	25	27	29	2C	2E	30	33	35	38	3B	3D	40	43	45	48	4B
12	4E	51	54	57	5A	5D	60	63	66	69	6C	6F	72	75	78	7B
13	7F	82	85	88	8B	8E	91	94	97	9A	9D	A0	A3	A6	A9	AC
14	AF	B2	B5	B8	BA	BD	C0	C2	C5	C8	CA	CD	CF	D1	D4	D6
15	D8	DA	DD	DF	E1	E3	E5	E6	E8	EA	EB	ED	EF	F0	F1	F3
16	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD						

Simple Sinewave Generator



```
1 module sinegen #(
2     parameter A_WIDTH = 8,
3     parameter D_WIDTH = 8
4 )(
5     // interface signals
6     input logic          clk,      // clock
7     input logic          rst,      // reset
8     input logic          en,       // enable
9     input logic [D_WIDTH-1:0] incr,    // increment for addr counter
10    output logic [D_WIDTH-1:0] dout;   // output data
11 );
12
13    logic [A_WIDTH-1:0] address; // interconnect wire
14
15    counter addrCounter (
16        .clk (clk),
17        .rst (rst),
18        .en (en),
19        .incr (incr),
20        .count (address)
21    );
22
23    rom sineRom (
24        .clk (clk),
25        .addr (address),
26        .dout (dout)
27    );
28
29 endmodule
```

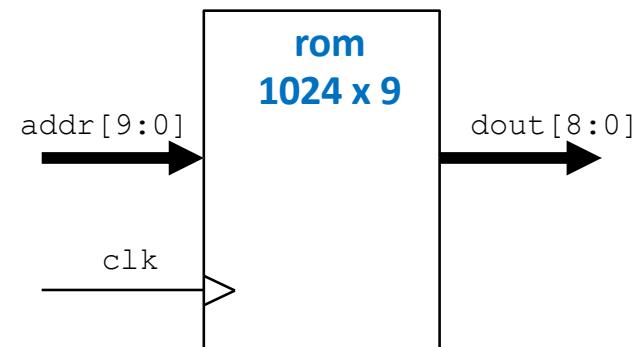
Instantiate counter module called addrCounter

external signal name

Internal signal name

Parameterised ROM:

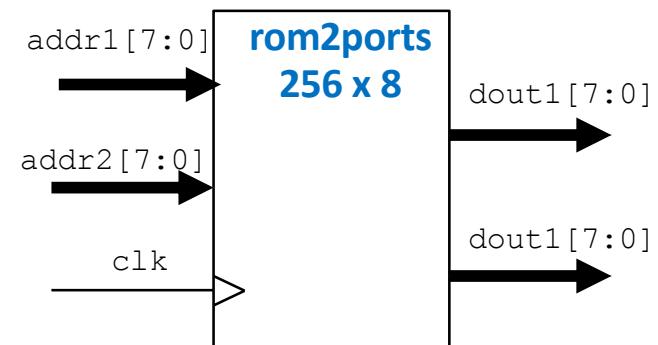
```
1 module rom #(                                counter
2     parameter ADDRESS_WIDTH = 8,
3             DATA_WIDTH = 8
4 )(
5     input logic                      clk,
6     input logic [ADDRESS_WIDTH-1:0] addr,
7     output logic [DATA_WIDTH-1:0]   dout
8 );
9
10 logic [DATA_WIDTH-1:0] rom_array [2**ADDRESS_WIDTH-1:0];
11
12 initial begin
13     $display("Loading rom.");
14     $readmemh("sinerom.mem", rom_array);
15 end;
16
17 always_ff @(posedge clk)
18     // output is synchronous
19     dout <= rom_array [addr];
20
21 endmodule
22
```



```
rom #(10, 9) sineRom_1024x9 (.....)
```

Dual-port ROM

```
1 module rom2ports #(  
2     parameter ADDRESS_WIDTH = 8,  
3     parameter DATA_WIDTH = 8  
4 )(  
5     input logic                      clk,  
6     input logic [ADDRESS_WIDTH-1:0]    addr1,  
7     input logic [ADDRESS_WIDTH-1:0]    addr2,  
8     output logic [DATA_WIDTH-1:0]     dout1,  
9     output logic [DATA_WIDTH-1:0]     dout2  
10 );  
11  
12 logic [DATA_WIDTH-1:0] rom_array [2**ADDRESS_WIDTH-1:0];  
13  
14 initial begin  
15     $display("Loading rom.");  
16     $readmemh("sinerom.mem", rom_array);  
17 end;  
18  
19 always_ff @(posedge clk) begin  
20     // output is synchronous  
21     dout1 <= rom_array [addr1];  
22     dout2 <= rom_array [addr2];  
23 end  
24  
25 endmodule
```



Dual-port RAM

```
1 module ram2ports #(  
2     parameter ADDRESS_WIDTH = 8,  
3     parameter DATA_WIDTH = 8  
4 )(  
5     input logic clk,  
6     input logic wr_en,  
7     input logic rd_en,  
8     input logic [ADDRESS_WIDTH-1:0] wr_addr,  
9     input logic [ADDRESS_WIDTH-1:0] rd_addr,  
10    input logic [DATA_WIDTH-1:0] din,  
11    output logic [DATA_WIDTH-1:0] dout  
12 );  
13  
14 logic [DATA_WIDTH-1:0] ram_array [2**ADDRESS_WIDTH-1:0];  
15  
16 always_ff @(posedge clk) begin  
17     if (wr_en == 1'b1)  
18         ram_array[wr_addr] <= din;  
19     if (rd_en == 1'b1)  
20         // output is synchronous  
21         dout <= ram_array [rd_addr];  
22 end  
23 endmodule
```

