Sharif University of Technology
Department of Computer Engineering

Digital System Design Course

Memory

Amin Foshati (/ˈfɒsˈhati/)

Spring Semester 2024

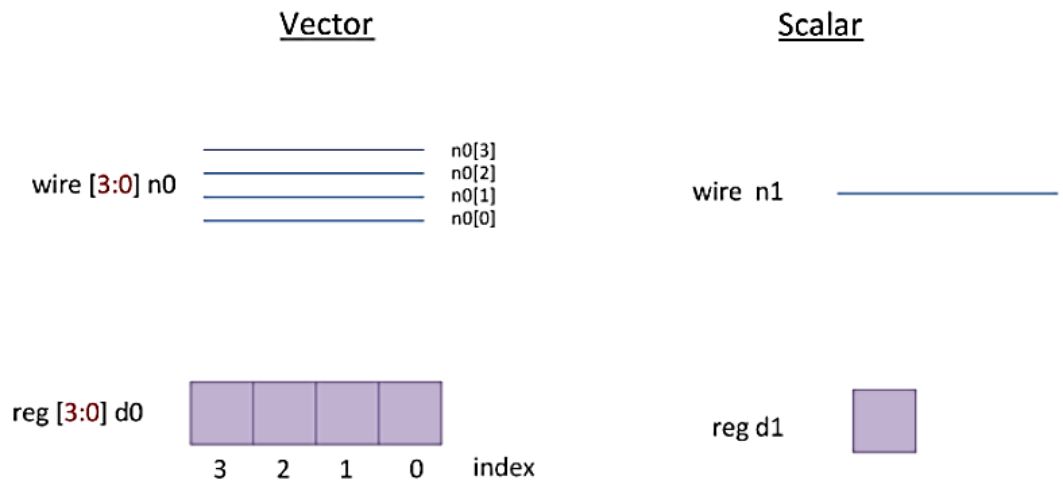
Vector

- Nets or reg data types can be declared as vectors (multiple bit widths).

  wire/reg [msb_index : lsb_index] identifier;

- Example

  wire a;
  wire [7:0] bus;
  wire [31:0] busA, busB, busC;
  reg clock;
  reg [0:40] virtual_addr;



- Access to parts of a vector

  bus[2:0] // Three least significant bits of vector bus
  busA[7] // bit # 7 of vector busA
  virtual_addr[0:1] // Two most significant bits of vector virtual_addr

Homework

Simulate the following code and pay attention to errors.

```
module example_vector;
  reg [15:0] data_1;
  reg [0:15] data_2;
  initial
  begin
            data_1 = 16'h23AF;
            data_2 = 16'h01BC;
            $display("data_1[7:0] = %h", data_1[7:0]);
            $display("data_1[8:15] = %h", data_1[8:15]);
            $display("data_2[0:7] = %h", data_2[0:7]);
            $display("data_2[15:8] = %h", data_2[15:8]);
  end
endmodule
```

Homework

Simulate the following code and pay attention to errors.

```
module example_vector;
  reg [15:0] data_1;
  reg [0:15] data_2;
  initial
  begin
            data_1 = 16'h23AF;
            data_2 = 16'h01BC;
            $display("data_1[7:0] = %h", data_1[7:0]);
            $display("data_1[8:15] = %h", data_1[8:15]);
            $display("data_2[0:7] = %h", data_2[0:7]);
            $display("data_2[15:8] = %h", data_2[15:8]);
  end
endmodule
```

```
** Error:
(vlog-3373) Range of part-select [8:15] into 'data_1' [15:0] is reversed.
** Error:
(vlog-3373) Range of part-select [15:8] into 'data_2' [0:15] is reversed.
```

Array

- In Verilog, we can create and use array.
  - useful for modeling memory.

```
<type> [size] <variable_name> <elements>;
```

```
type => reg, integer, time, real, wire
```

Example:

```
integer count[0:7]; // An array of 8 integer count variables
count[5] = 0; // Reset 6th element of the array of count variables
```

Array Example

- time chk_point[1:100]; // Array of 100 time checkpoint variables
- chk_point[100] = 0; // Reset 100th time check point value

- reg [4:0] port_id[0:7]; // Array of 8 port_ids; each port_id is 5 bits wide
- port_id[3] = 0; // Reset 3rd element (a 5-bit value) of port_id array.

- port_id = 0; // Illegal syntax - Attempt to write the entire array

Array vs. Vector (Verilog)

```
module test;
  reg [0:9] x;
  reg [0:9] y;

  initial
  begin
    x[6]=1'b1;
    x[7]=1'b0;

    y[6:7]=x[6:7];  //it is OK
  end

endmodule
```

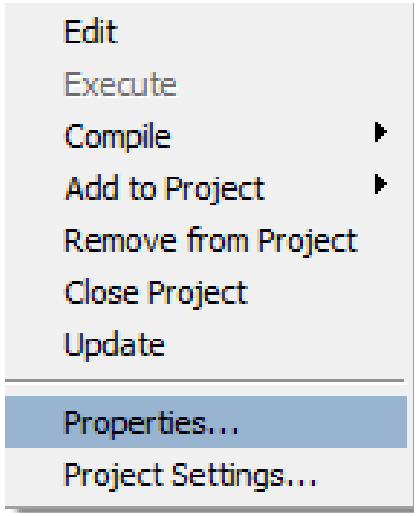
```
module test;
  integer x[0:9];
  integer y[0:9];

  initial
  begin
    x[7]=13;
    x[6]=10;

    y[6:7]=x[6:7];  //illegal
  end

endmodule
```

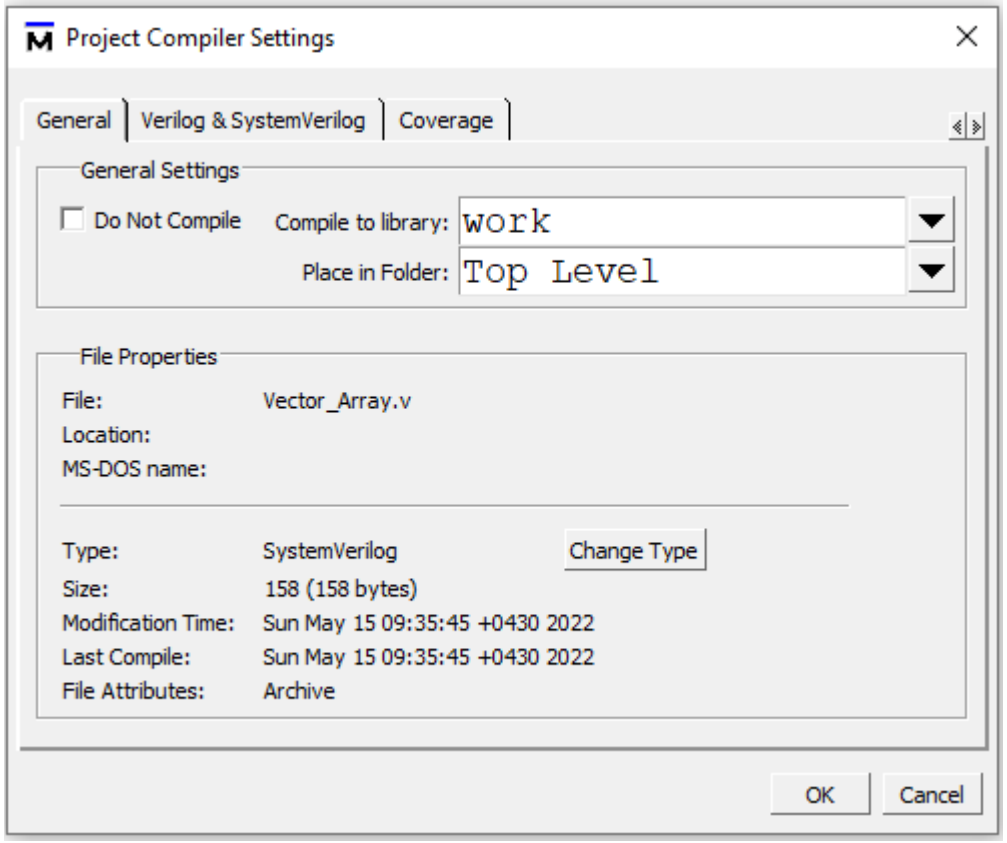
Array vs. Vector (Verilog)

```
module test;
  reg [0:9] x;
  reg [0:9] y;

  initial
  begin
    x[6]=1'b1;
    x[7]=1'b0;

    y[6:7]=x[6:7];  //it is OK
  end

endmodule
```

```
module test;
  integer x[0:9];
  integer y[0:9];

  initial
  begin
    x[7]=13;
    x[6]=10;

    y[6:7]=x[6:7];  //illegal
  end

endmodule
```

Set Filetype to SystemVerilog





Array vs. Vector (SystemVerilog)

```
module test;
  reg [0:9] x;
  reg [0:9] y;

  initial
  begin
    x[6]=1'b1;
    x[7]=1'b0;

    y[6:7]=x[6:7];  //it is OK
  end

endmodule
```

```
module test;
  integer x[0:9];
  integer y[0:9];

  initial
  begin
    x[7]=13;
    x[6]=10;

    y[6:7]=x[6:7];  //it is OK
  end

endmodule
```

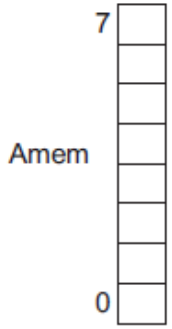
Multidimensional Array

- In the Verilog 1995 standard,
  - it is only possible to create one dimensional arrays
- In the Verilog 2001 standard,
  - Multi-dimensional arrays can also be declared with any number of dimensions.
- Example:
  - integer matrix[4:0][0:255]; // Two dimensional array of integers
  - matrix[1][0] = 33559; // Set value of element indexed by [1][0] to 33559
  - reg [63:0] array_4d [15:0][7:0][7:0][255:0]; //Four dimensional array
  - array_4d[0][0][0][0][15:0] = 0; //Clear bits 15:0 of the register accessed by indices [0][0][0][0]
  - matrix [1] = 0; // Illegal syntax

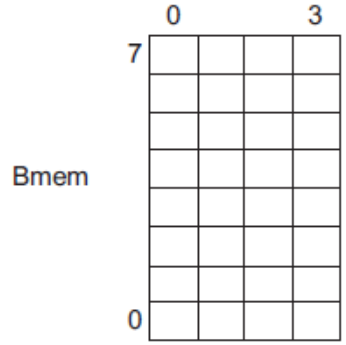
Multidimensional Array (Continue)

```
// An 8-bit vector
reg [7:0] Areg;
```

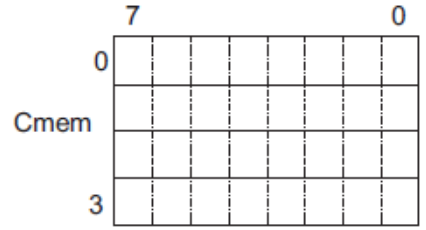

```
// A memory of 8 one-bit elements
reg Amem [7:0];
```



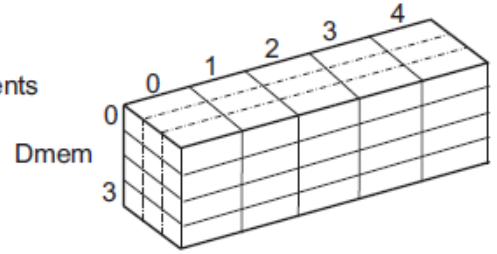

```
// A two-dimensional memory of one-bit elements
reg Bmem [7:0] [0:3];
```




```
// A memory of four 8-bit words
reg [7:0] Cmem[0:3];
```




```
// A two-dimensional memory of 3-bit elements
reg[2:0] Dmem [0:3] [0:4];
```



Memory

Memory

- Memories are modeled in Verilog simply as a one-dimensional array of registers.
- Example:
  - reg mem1bit[0:1023]; // Memory mem1bit with 1K 1-bit words
  - reg [7:0] membyte[0:1023]; // Memory membyte with 1K 8-bit words(bytes)
  - membyte[511] // Fetches 1 byte word whose address is 511

Initializing Memories

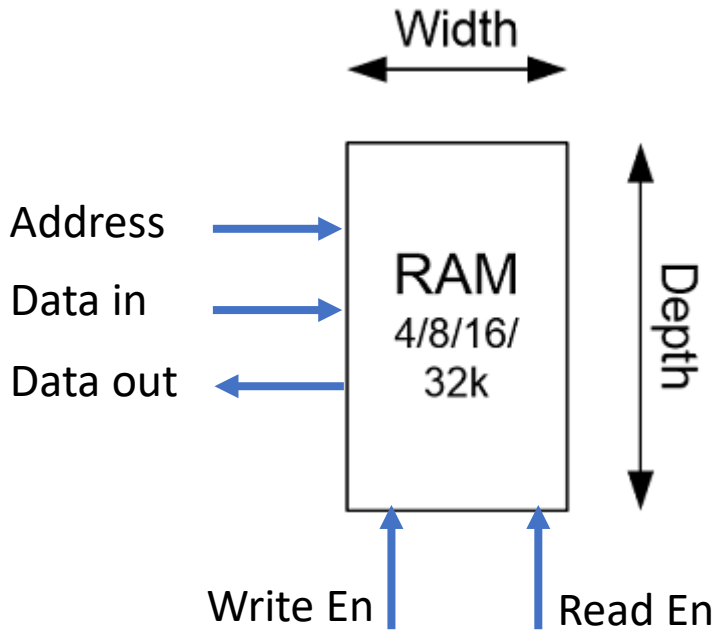
- System Tasks
  - $readmemb
    - for binary representation of memory content
  - $readmemh
    - for hex representation of memory content
- Example:
  - $readmemb("file_name",mem_array,start_addr,stop_addr);

content of memory file:

```
1100_1100   // This is first address i.e 8'h00
1010_1010   // This is second address i.e 8’h01
```

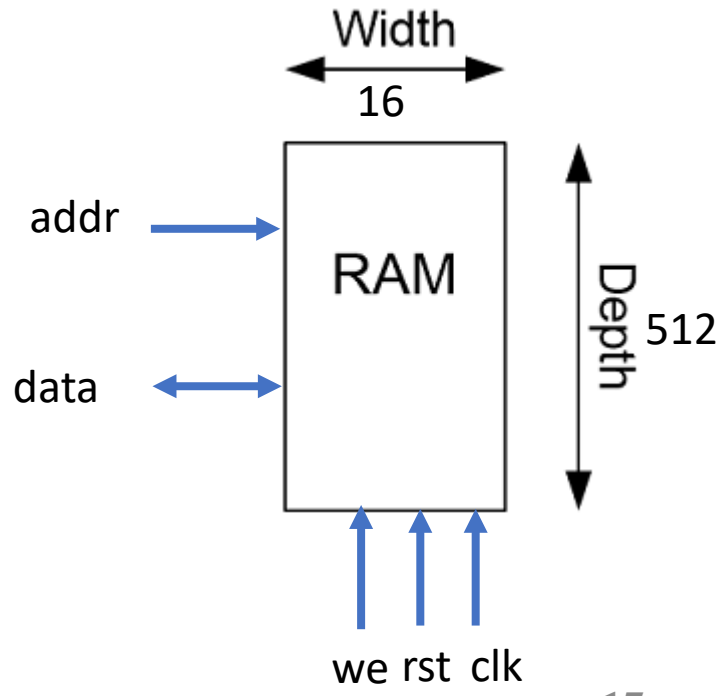
RAM

- RAM is a kind of random-access memory that allows multiple reads or writes at or near the same time.
  - single-ported RAM
    - Allow one access at a time
  - RAM Width
  - RAM Depth



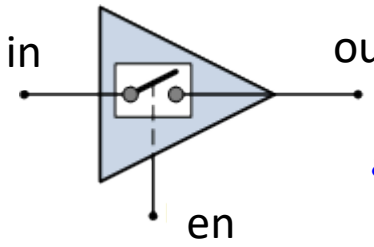
RAM Example

```
module mem(input [8:0] addr, input we, rst, clk, inout [15:0] data);

        reg [15:0] mem[0:511];

        …

endmodule
```

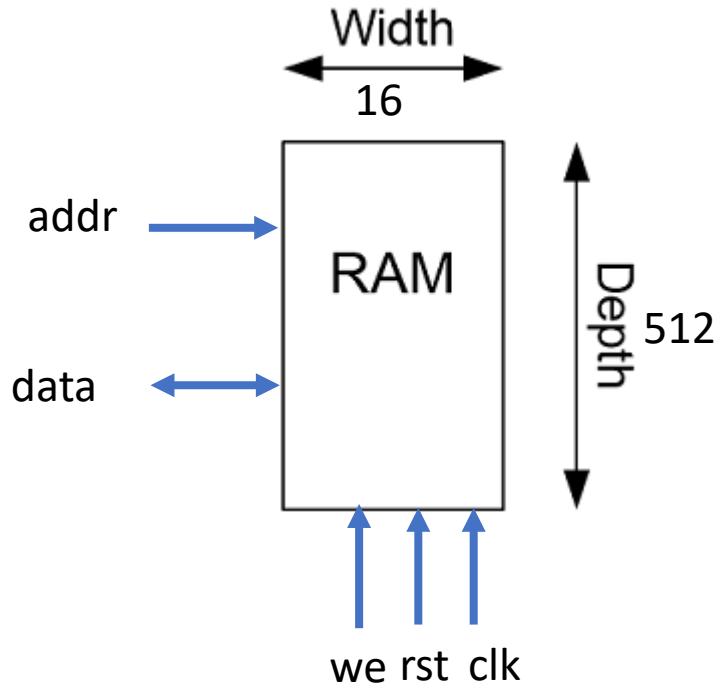


RAM Example

```
module mem(input [8:0] addr, input we, rst, clk, inout [15:0] data);

        reg [15:0] mem[0:511];

        …

endmodule
```

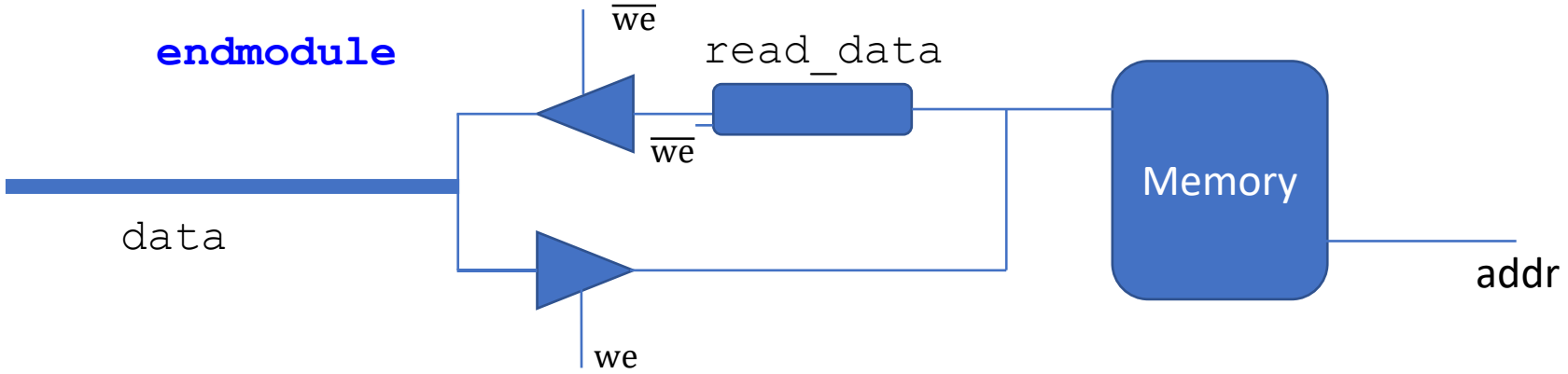


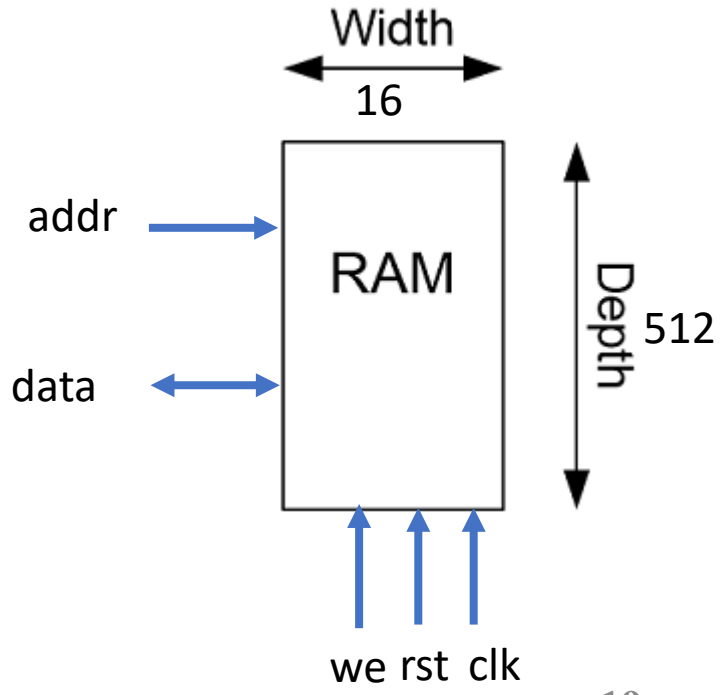

```
assign out = en ? in: 1'bz;
```



RAM Example

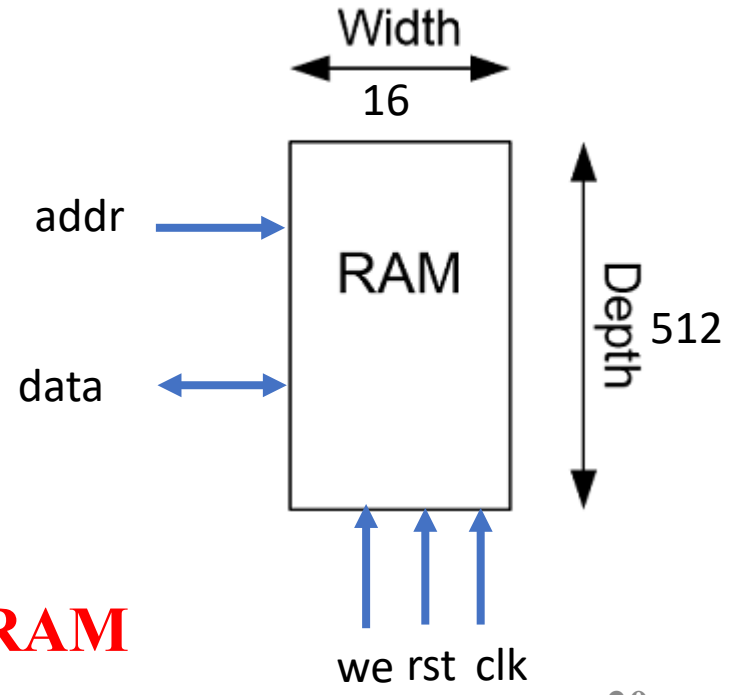
```
module mem(input [8:0] addr, input we, rst, clk, inout [15:0] data);
        reg [15:0] mem[0:511];
        reg [15:0] read_data;

        assign data = !we ? read_data : 16'bz;
        always @(posedge clk) begin
                if (we) mem[addr] <= data;
                …
        end

endmodule
```





RAM Example

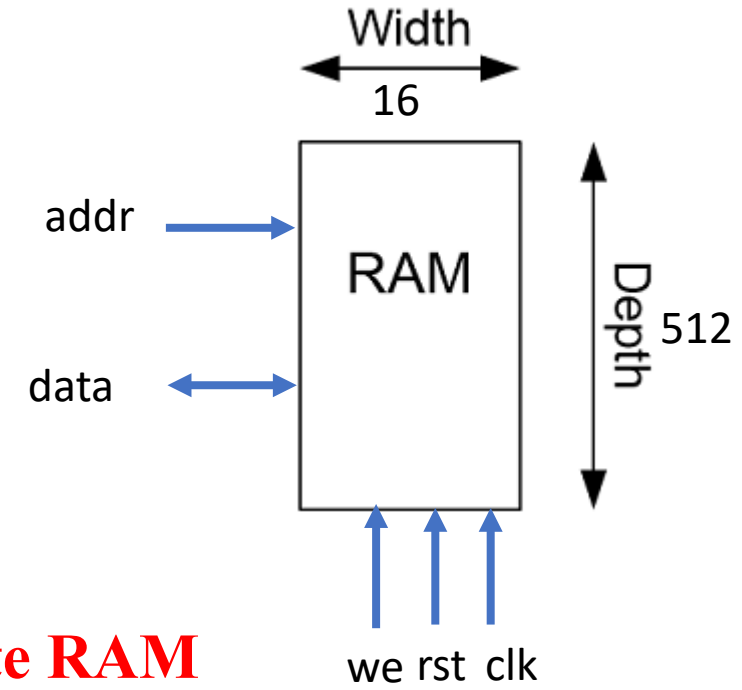
```
module mem(input [8:0] addr, input we, rst, clk, inout [15:0] data);

        reg [15:0] mem[0:511];
        reg [15:0] read_data;

        assign data = !we ? read_data : 16'bz;

        always @(posedge clk) begin
                if (rst) read_data <= {16{1'b0}};
                else if (we) mem[addr] <= data;
                else read_data <= mem[addr];
        end

endmodule
```



Single-Port Synchronous Read & Write RAM

RAM Example (Parameter)

```
module mem
#(parameter WIDTH = 16, parameter ADDR_WIDTH = 9)
(input [ADDR_WIDTH-1:0] addr, input we, rst, clk, inout [WIDTH-1:0] data);

        reg [WIDTH-1:0] mem[0:2**ADDR_WIDTH - 1];
        reg [WIDTH-1:0] read_data;

        assign data = !we ? read_data : 16'bz;

        always @(posedge clk) begin
                if (rst) read_data <= {16{1'b0}};
                else if (we) mem[addr] <= data;
                else read_data <= mem[addr];
        end

endmodule
```



Single-Port Synchronous Read & Write RAM

Homework

- Develop a single-port asynchronous read RAM with single-port asynchronous read/write.