Verilog HDL:
Memory Design

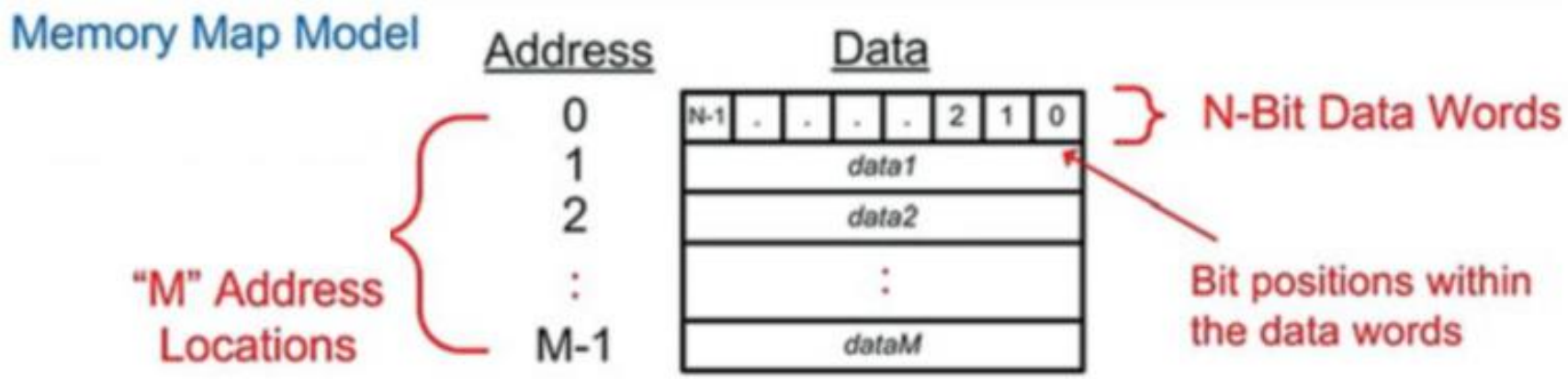
Pravin Zode

Outline

- Memory Model
- Memory Model (Aynchronous)
- Memory Model (Synchronous)

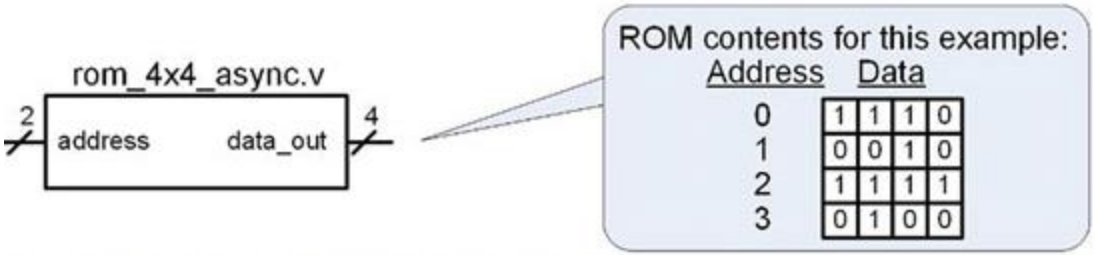
Introduction

- Memory size expressed as M × N (addresses × bits per word)
- Example: 16 × 8 memory → 16 addresses, each 8 bits
- Total capacity = 16 × 8 = 128 bits
- Number of addresses M = 2^n, where n = number of address lines



Memory Model (Asynchronous)

Asynchronous Memory , meaning that as soon as the address changes the data from the ROM will appear immediately



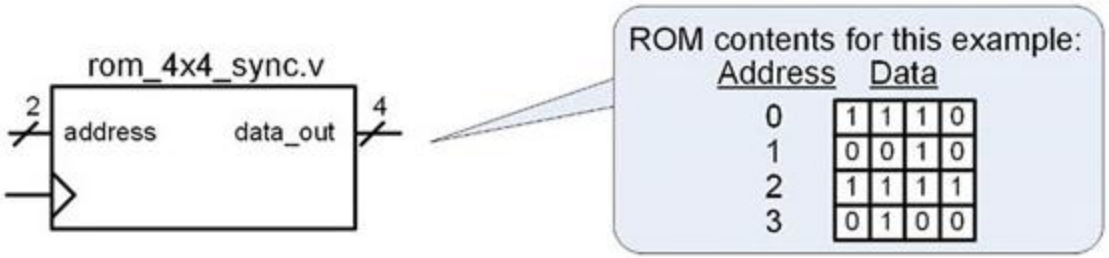

```
module rom_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address);

  always @ (address)
    case (address)
      0       : data_out = 4'b1110;
      1       : data_out = 4'b0010;
      2       : data_out = 4'b1111;
      3       : data_out = 4'b0100;
      default : data_out = 4'bXXXX;
    endcase

endmodule
```

```
module rom_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address);

  reg[3:0] ROM[0:3];

  initial
    begin
      ROM[0] = 4'b1110;
      ROM[1] = 4'b0010;
      ROM[2] = 4'b1111;
      ROM[3] = 4'b0100;
    end
  always @ (address)
    data_out = ROM[address];
endmodule
```

← An MxN array is declared.

Memory Model (Synchronous)

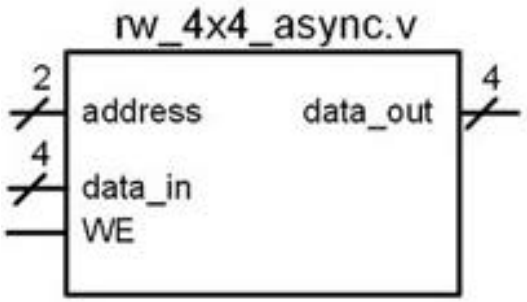
Synchronous ROM, a clock edge is used to trigger the procedural block that updates data_out. A sensitivity list is used that contains the clock to trigger the assignment




```
module rom_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        Clock);

  always @ (posedge Clock)
    case (address)
      0       : data_out = 4'b1110;
      1       : data_out = 4'b0010;
      2       : data_out = 4'b1111;
      3       : data_out = 4'b0100;
      default : data_out = 4'bXXXX;
    endcase
endmodule
```

```
module rom_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        Clock);
  reg[3:0] ROM[0:3];

  initial
    begin
      ROM[0] = 4'b1110;
      ROM[1] = 4'b0010;
      ROM[2] = 4'b1111;
      ROM[3] = 4'b0100;
    end

  always @ (posedge Clock)
    data_out = ROM[address];
endmodule
```

Asynchronous Read/Write Memory



Contents to be written:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
module rw_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        WE,
    input  wire [3:0]  data_in);

  reg[3:0] RW[0:3];

  always @ (address or WE or data_in)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

An MxN array is declared called "RW".

This provides the asynchronous behavior.

Writing to the RW array when WE=1.

Reading from the RW array when WE=0.

Asynchronous Read/Write Memory

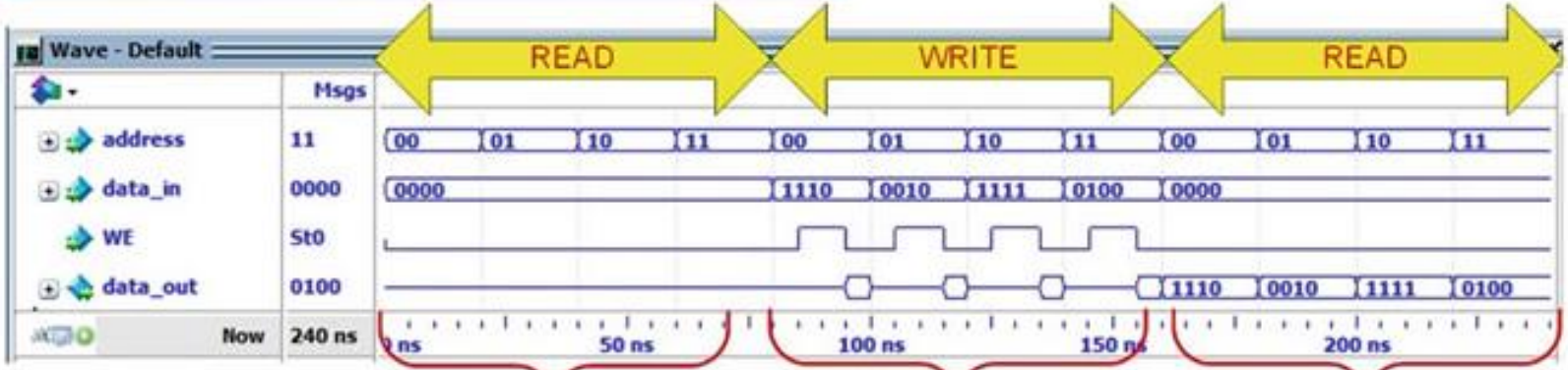
```
module rw_4x4_async
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        WE,
    input  wire [3:0]  data_in);

  reg[3:0] RW[0:3];

  always @ (address or WE or data_in)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

An MxN array is declared called "RW".

This provides the asynchronous behavior.

Writing to the RW array when WE=1.

Reading from the RW array when WE=0.

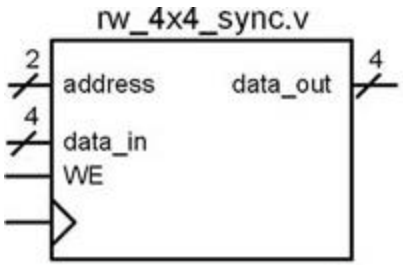


On start-up, the memory is empty so the reads from the four addresses yield "uninitialized".

Data is then written to the four addresses.

When reads are performed again, the data that was written appears.

Synchronous Read/Write Memory



Contents to be written:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

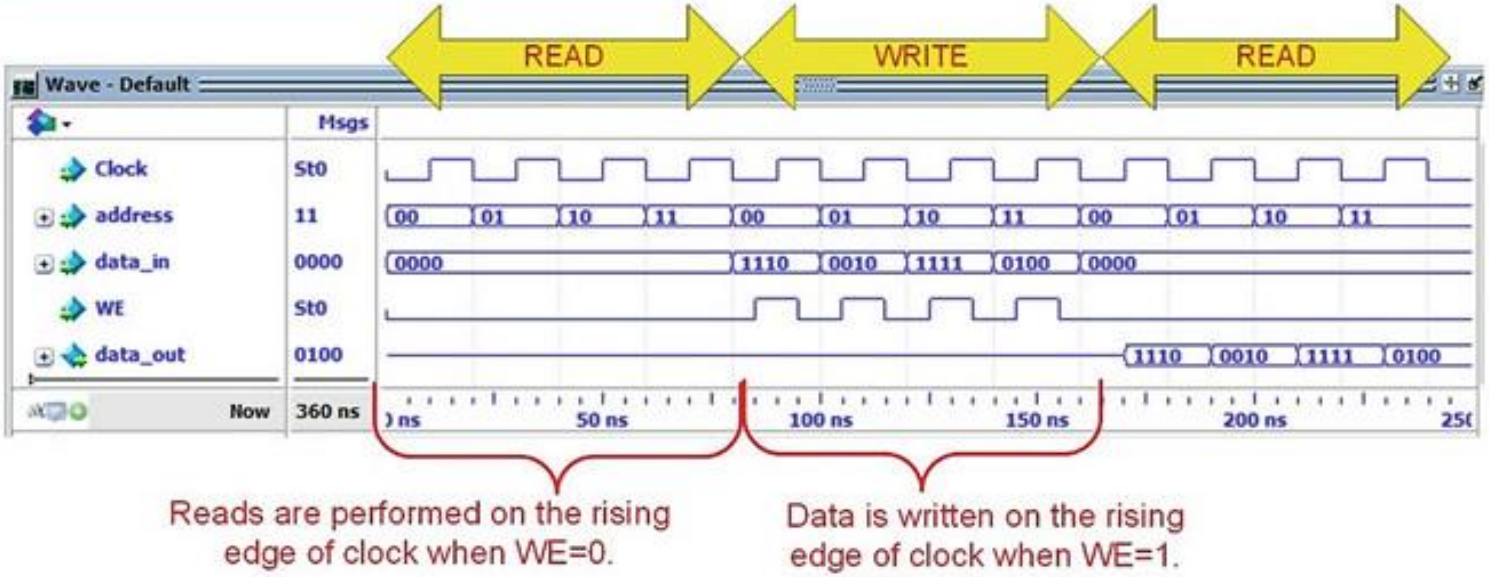
```
module rw_4x4_sync
   (output reg  [3:0]  data_out,
    input  wire [1:0]  address,
    input  wire        WE,
    input  wire [3:0]  data_in,
    input  wire        Clock);

  reg[3:0] RW[0:3];

  always @ (posedge Clock)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

Reads and writes only occur on the rising edge of the clock.

Synchronous Read/Write Memory

```
module rw_4x4_sync
    (output reg  [3:0]  data_out,
     input  wire [1:0]  address,
     input  wire        WE,
     input  wire [3:0]  data_in,
     input  wire        Clock);

  reg[3:0] RW[0:3];

  always @ (posedge Clock)
    if (WE)
      RW[address] = data_in;
    else
      data_out = RW[address];

endmodule
```

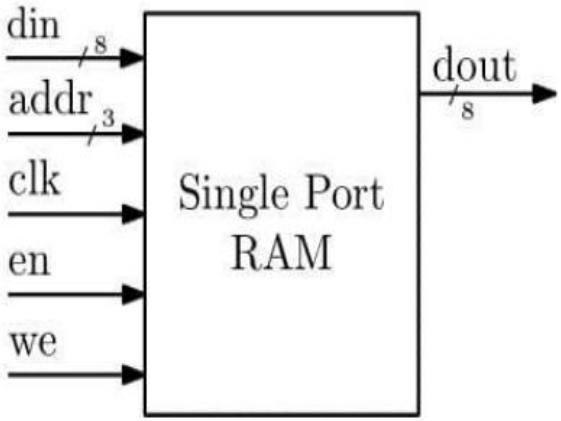
Reads and writes only occur on the rising edge of the clock.



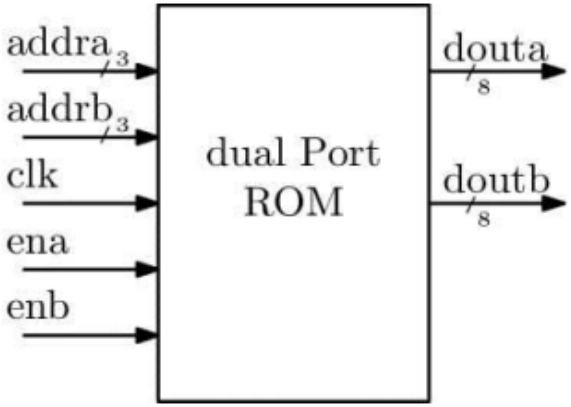
Reads are performed on the rising edge of clock when WE=0.

Data is written on the rising edge of clock when WE=1.

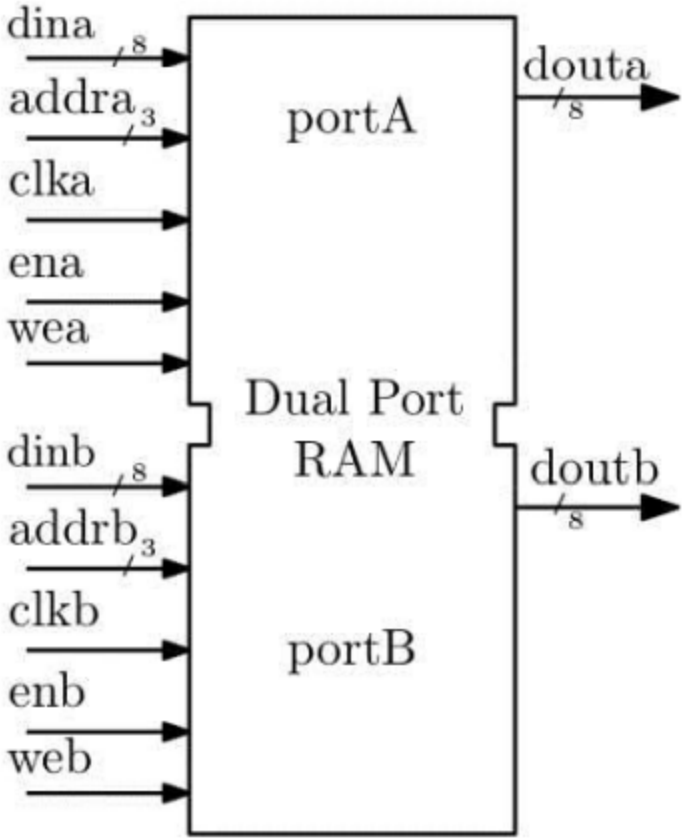
Memory Ports



Single Port RAM

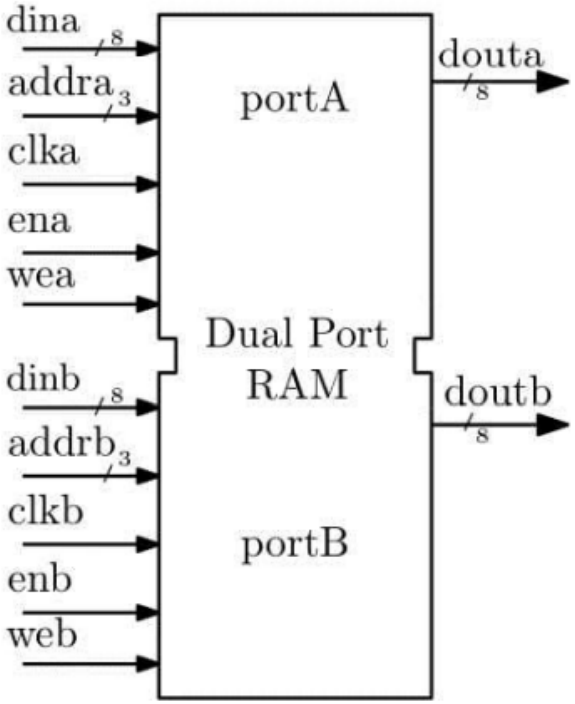


Dual Port ROM



Dual Port RAM

RAM Ports



modes	ena	wea	enb	web	portA	portB
1	1	1	1	1	write	write
2	1	1	1	0	write	read
3	1	0	1	1	read	write
4	1	0	1	0	read	read

- Mode 1, both ports can write data simultaneously, but not to the same memory location
- Mode 2 and 3, one port writes while the other reads, enabling concurrent read/write operations at different addresses
- In Mode 3, both ports can read data at the same time, even from the same address location

Port description

Port Name	Size	Description	Direction
Wr_en	1-bit	Write enable (active high)	Input
Rd_en	1-bit	Read enable (active high)	Input
Wr_addr	4-bit	Write address (16 locations)	Input
Rd_addr	4-bit	Read address (16 locations)	Input
Data_in	32-bit	Data to write	Input
Data_out	32-bit	Data read from memory	Output
Rst	1-bit	Active low asynchronous reset	Input
Clk	1-bit	Clock signal (operations on negative edge)	Input

Summary

- Asynchronous memory responds immediately
- Synchronous memory responds on clock edge
- Verilog models ROM with initialized arrays or case statements
- R/W memory with uninitialized arrays

Thank you !

Happy Learning