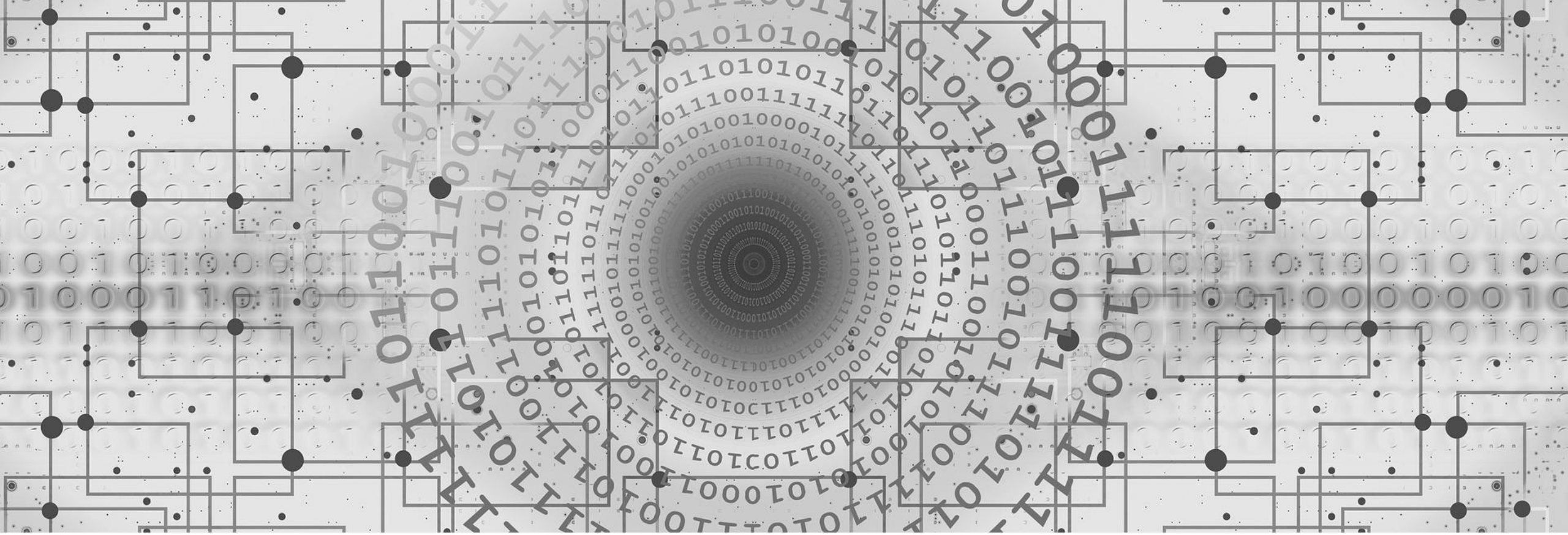


Lecture-15 : Packed/Unpacked Array, Memory Modeling

ECE-111

Vishal Karna

Summer 2025

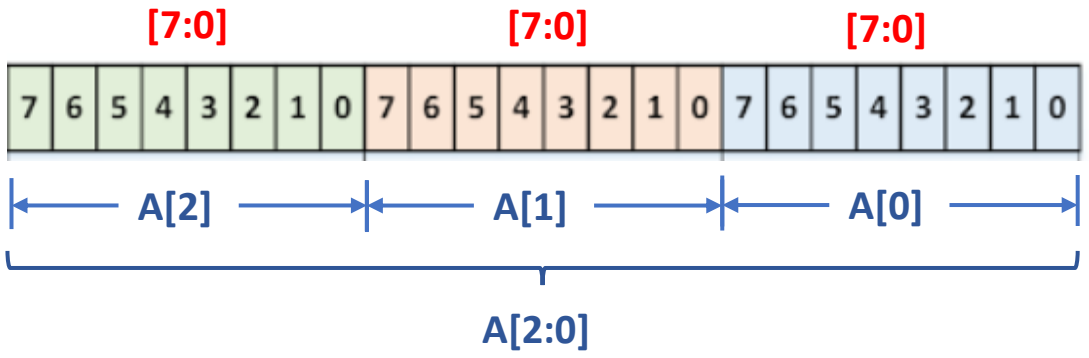
UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Packed and Unpacked Arrays

Packed and Unpacked Arrays

Packed Array

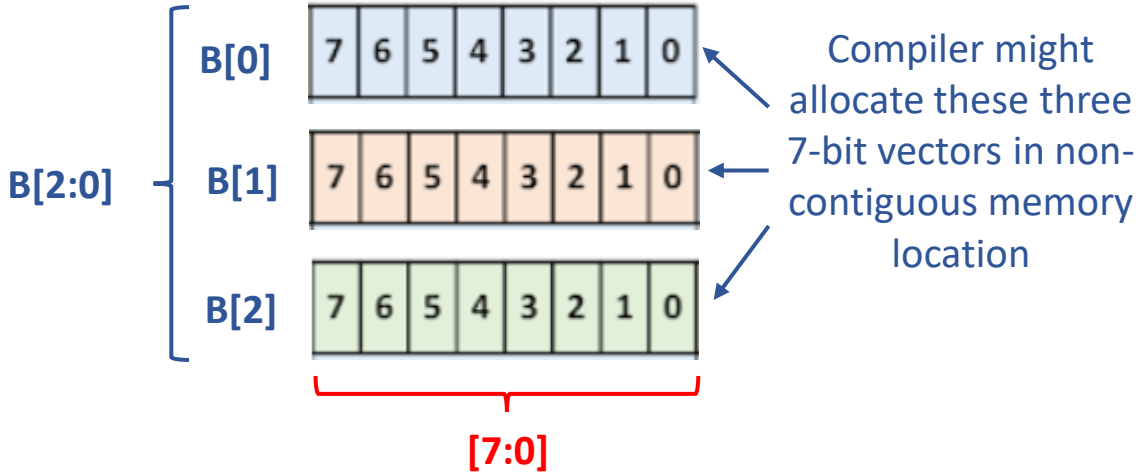
Example : bit[2:0][7:0] A



- Information **is** represented as a **contiguous** set of bits.
- Dimensions declared **before** variable name
- Can be constructed with :
  - of single bit data types (reg, logic, bit),
  - enumerated types
  - packed structures
  - packed arrays
- **Examples :**
  - bit [3:0] **C**; → 4 bit vector
  - logic [2:0] [7:0] **A**; → logic type 24 bit vector

Unpacked Array

Example : bit[7:0] B[2:0]



- Information **may or may not** be represented as a **contiguous** set of bits. **Looks like a RAM !!**
- Dimensions declared **after** variable name
- Can be constructed with any data type:
  - of single bit data types (reg, logic, bit),
  - enumerated types, user defined types
  - packed structures, packed arrays, unpacked array
- **Examples :**
  - logic [3:0] **D** [7:0][2:0]; → 2-dimension array of 4-bit vector
  - bit [7:0] **B** [2:0]; → 1-dimension array of 8-bit vector

Packed and Unpacked Arrays

Packed Array

- **Assigning values to packed arrays :**
  - Assigning constant value :
    - logic [7:0] P = 8'h24;
    - logic [1:0][3:0] Q = 8'h24;
  - Assigning the result of a replication operator
    - logic [1:0][3:0] Q = {2{4'b1001}};
  - Assigning the result of a concatenation operator
    - logic [1:0][3:0] Q = {4'h2, 4'h4};
- **Use packed arrays to model**
  - Vectors of 1-bit types, e.g., logic
  - Vectors where it is useful to access subfields

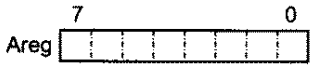
Unpacked Array

- **Assigning values to unpacked arrays :**
  - Assigning constant value :
    - logic R[7:0] = ' {0,0,1,0,0,1,0,0}; **// unpacked array**
    - logic [1:0] S[3:0] = 8'h24; **// mix of packed and unpacked array**
  - Assigning the result of a replication operator
    - logic [1:0] T[2:0] = ' {3{2'b11}}; **// mix of packed and unpacked array**
    - logic V[1:0][2:0] = ' {2{'{1,0,1}}}; **// unpacked array**
  - Assigning the result of a concatenation operator
    - logic [1:0] T[2:0] = ‘ {{2'b11}, {2'b10}, {2'b01}}; **// mix of packed and unpacked array**
    - logic V[1:0][2:0] = ' {'{1,0,1}, '{0,1,1}}; **// unpacked**
- **Use unpacked arrays to model**
  - Arrays accessed one element at a time, e.g., RAM
  - Arrays of byte, int, real, etc.

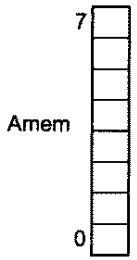
Packed and Unpacked Arrays

Declaration of Packed and Unpacked Array

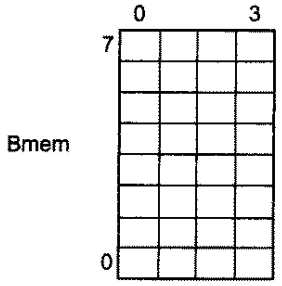
// An 8-bit vector
reg [7:0] Areg;



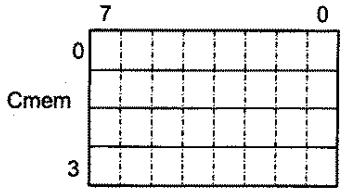
// A memory of 8 one-bit elements
reg Amem [7:0];



// 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];

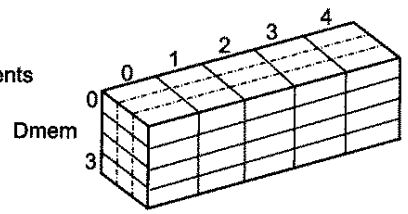
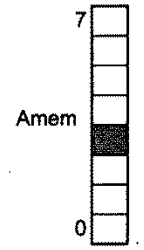


Figure 3.8 Array structures

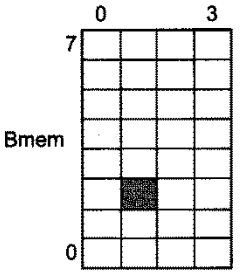
Indexing Packed and Unpacked Array

// declaration: **reg** [7:0] Areg;
Areg [7:5]

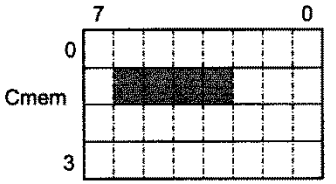
// declaration: **reg** Amem [7:0];
Amem [3]



// declaration: **reg** Bmem [7:0][0:3]
Bmem [2] [1]



// declaration: **reg**[7:0] Cmem [0:3]
Cmem [1] [6 -: 4]



// declaration: **reg** [2:0] Dmem [0:3][0:4]
Dmem [0] [2]

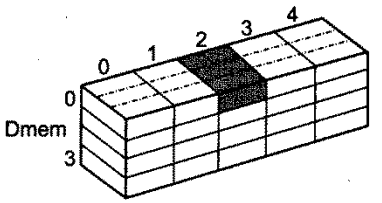


Figure 3.16 Array Addressing and Selection

Packed and Unpacked Arrays

Referencing packed arrays

A packed array can be referenced as a whole, as bit-selects, or as part-selects. Multidimensional packed arrays can also be referenced in slices. A slice is one or more contiguous dimensions of an array.

```
logic [3:0][7:0] data;  // 2-D packed array


wire [31:0] out = data;           // whole array
wire sign = data[3][7];           // bit-select
wire [3:0] nib = data [0][3:0];   // part-select
byte high_byte;
assign high_byte = data[3];       // 8-bit slice
logic [15:0] word;
assign word = data[1:0];          // 2 slices
```

Operations on packed arrays

any vector operation can be performed on packed arrays

Because packed arrays are stored as vectors, any legal operation that can be performed on a Verilog vector can also be performed on packed arrays. This includes being able to do bit-selects and part-selects from the packed array, concatenation operations, math operations, relational operations, bit-wise operations, and logical operations.

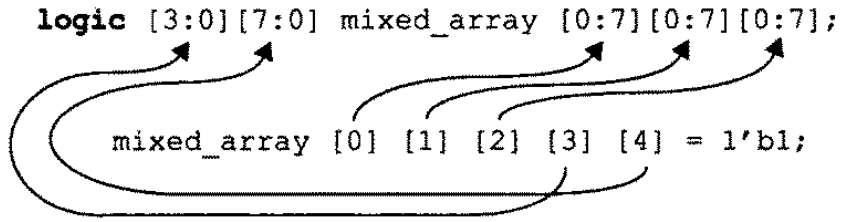
```
logic [3:0][15:0] a, b, result; // packed arrays
...
result = (a << 1) + b;
```

Indexing arrays of arrays

unpacked dimensions are indexed before packed dimensions

When indexing arrays of arrays, unpacked dimensions are referenced first, from the left-most dimension to the right-most dimension. Packed dimensions (vector fields) are referenced second, from the left-most dimension to the right-most dimension. Figure 5-5 illustrates the order in which dimensions are selected in a mixed packed and unpacked multi-dimensional array.

Figure 5-5: Selection order for mixed packed/unpacked multi-dimensional array



Memory Modeling for FPGA

Memory Modeling

- **When modeling memory in SystemVerilog code, one can either use:**
  - Dedicated registers (flipflops) within each ALUT inside FPGA

    OR
  - Embedded memory IP's, such as Block Rams, available inside FPGA

- **Memory modeling using flipflops inside ALUT vs using Embedded Memory**
  - Memory modeled using flipflops inside ALUT reduces design performance and utilizes more area
  - Flipflops inside ALUT to model memory should be used when all embedded memory resources are used
  - Embedded memory IP's within FPGA are optimized for speed and area.
  - Embedded Memory IP is known as Block Ram
  - Synthesizer will generate messages whenever it infers embedded memory IP resources
  - Memory modeled using logic cells is know as Distributed Ram
  - Synthesizer tool should be instructed to use embedded memory for RAM modeling, otherwise it will use flipflops within ALUT's

Memory Modeling

- **Altera FPGA devices has various types of Memory IP cores**
  - Based on SystemVerilog memory modeling style and if auto option RAM replacement feature set in synthesizer settings, synthesizer will select appropriate embedded memory IP to meet speed, area, power targets
  - With Auto option, synthesizer will favor larger Block RAM's to fit entire memory inside single embed memory block.
    - **This gives the best performance and requires no logic elements (LEs) for glue logic !**

Table 1. Memory IP Cores and Their Features

Memory IP	Supported Memory Mode	Features
RAM: 1-PORT	Single-port RAM	• Non-simultaneous read and write operations from a single address. • Read enable port to specify the behavior of the RAM output ports during a write operation, to overwrite or retain existing value. • Supports freeze logic feature.
RAM: 2-PORT	Simple dual-port RAM	• Simultaneous one read and one write operations to different locations. • Supports error correction code (ECC). • Supports freeze logic feature.
	True dual-port RAM	• Simultaneous two reads. • Simultaneous two writes. • Simulatenous one read and one write at two different clock frequencies. • Supports freeze logic feature.
ROM: 1-PORT	Single-port ROM	• One port for read-only operations. • Initialization using a **.mif** or **.hex** file.
ROM: 2-PORT	Dual-port ROM	• Two ports for read-only operations. • Initialization using a **.mif** or **.hex** file.

Embedded Memory Blocks in Intel Altera FPGA Devices

Table 4. Embedded Memory Blocks in Intel FPGA Devices

Device Family	Memory Block Type					
	MLAB (640 bits)	M9K (9 Kbits)	M144K (144 Kbits)	M10K (10 Kbits)	M20K (20 Kbits)	Logic Cell (LC)
Arria® II GX	Yes	Yes	–	–	–	Yes
Arria II GZ	Yes	Yes	Yes	–	–	Yes
Arria V	Yes	–	–	Yes	–	Yes
Intel Arria 10	Yes	–	–	–	Yes	Yes
Cyclone® IV	–	Yes	–	–	–	Yes
Cyclone V	Yes	–	–	Yes	–	Yes
Intel Cyclone 10 LP	–	Yes	–	–	–	Yes
Intel Cyclone 10 GX	Yes	–	–	–	Yes	Yes
MAX® II	–	–	–	–	–	Yes
Intel MAX 10	–	Yes	–	–	–	Yes
Stratix IV	Yes	Yes	Yes	–	–	Yes
Stratix V	Yes	–	–	–	Yes	Yes

Note: To identify the type of memory block that the software selects to create your memory, refer to the Fitter report after compilation.

Single Port Distributed RAM with Asynchronous Read

```
// Single-port Distributed RAM with Asynchronous Read
module single_port_distributed_ram_model1 #(
  parameter DATA_WIDTH=4, //width of data bus
  parameter ADDR_WIDTH=4  //width of addresses buses)(
  input logic clk, // clock
  input logic wr_en, // '1' indicates write and '0' indicates read
  input logic[DATA_WIDTH-1:0] write_data, //data to be written to memory
  input logic[ADDR_WIDTH-1:0] addr, //address for write or read operation
  output logic[DATA_WIDTH-1:0] read_data //read data from memory);
  // Two dimensional memory array with 16 locations
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

  // Synchronous write
  always_ff@(posedge clk) begin
    if(wr_en) mem[addr]  <= write_data;
  end

  // Asynchronous read
  assign read_data = mem[addr];
endmodule:
```

addr	mem
0	4'b1011
1	4'b0001
2	4'b1001
...	....
15	4'b0011

Since address is not registered for read, synthesizer will not be able to map memory model to internal embedded block ram IP. Instead, will use ALUT's and dedicated logic registers and create distributed RAM

ysis & Synthesis Resource Usage Summary

<Filter>>

	Resource	Usage
1	Estimated ALUTs Used	36
1	-- Combinational ALUTs	36
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	64
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	36

Indicates distributed memory model was implemented

Synthesizer indicates that Block Ram IP was *not* inferred

```
276014 Found 1 instances of uninferred RAM logic
286030 Timing-Driven Synthesis is running
 16010 Generating hard_block partition "hard_block:auto_generated_inst"
 21057 Implemented 114 device resources after synthesis - the final resource count might be different
       Quartus Prime Analysis & Synthesis was successful. 0 errors, 2 warnings
```

Single Port Block RAM with Asynchronous Read

```
// Single-port Block RAM with Asynchronous Read
module single_port_block_ram_model2 #(
  parameter DATA_WIDTH=4, //width of data bus
  parameter ADDR_WIDTH=4  //width of addresses buses)(
  input logic clk, // clock
  input logic wr_en, // '1' indicates write and '0' indicates read
  input logic[DATA_WIDTH-1:0] write_data, //data to be written
  input logic[ADDR_WIDTH-1:0] addr, //address for write or read operation
  output logic[DATA_WIDTH-1:0] read_data //read data from memory);
  // Two dimensional memory array
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];
  logic[ADDR_WIDTH-1:0] read_addr_t;

 // Synchronous write
  always_ff@(posedge clk) begin
    if(wr_en) mem[addr]  <= write_data;
    read_addr_t  <= addr;
  end

  // Asynchronous read
  assign read_data = mem[read_addr_t];
endmodule
```

Since address is registered for read, Synthesizer will map memory model to internal embedded **block ram IP**.

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated ALUTs Used	0
1	-- Combinational ALUTs	0
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	0
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	0

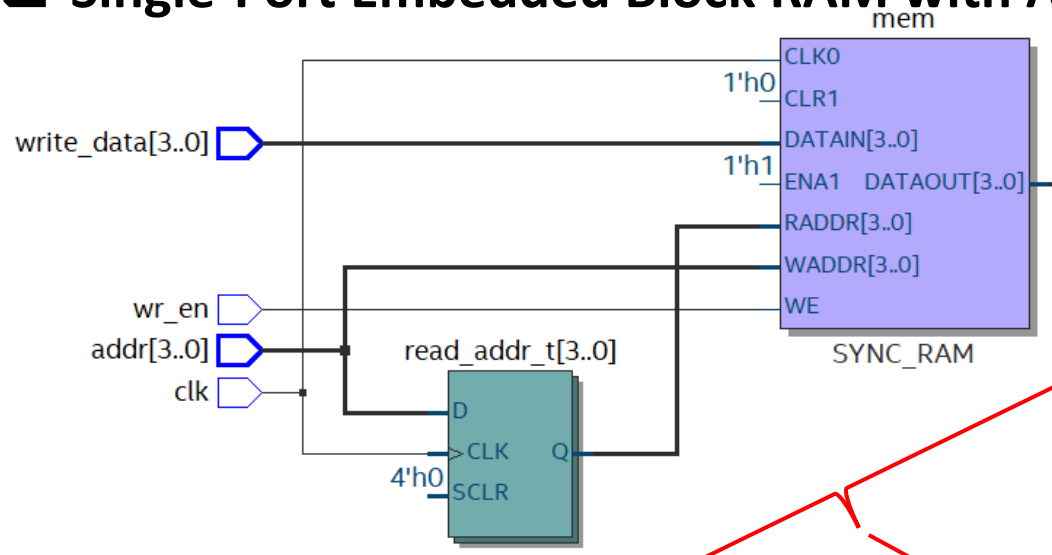
Dedicated logic register and ALUT count is 0 since Synthesizer mapped SystemVerilog code to embedded Block Ram

Message from synthesizer "1 megafunctions from design logic" indicates that Block Ram IP was inferred for given SystemVerilog model

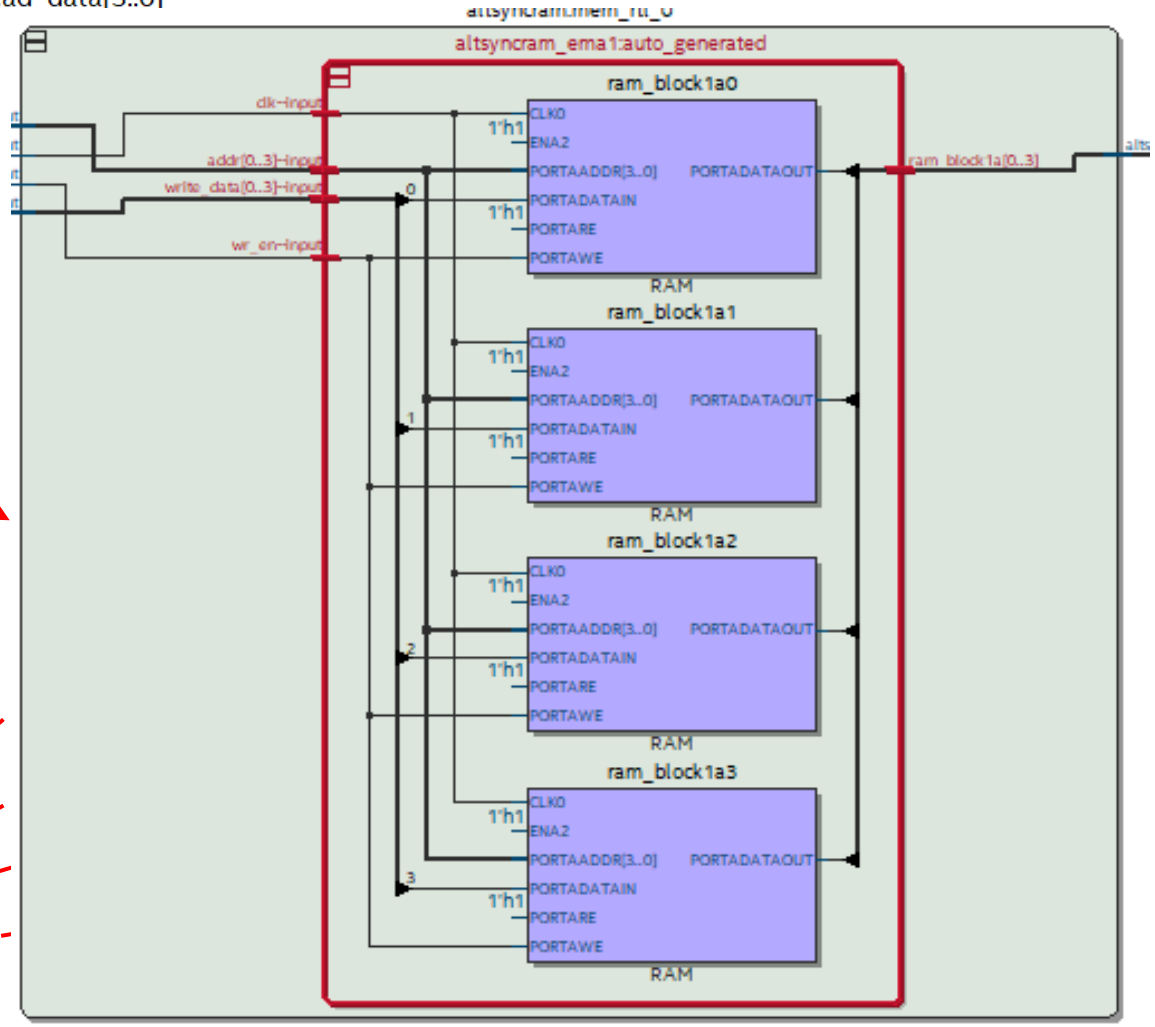
```
> 19000 Inferred 1 megafunctions from design logic
  12130 Elaborated megafunction instantiation "altsyncram:mem_rtl_0"
> 12133 Instantiated megafunction "altsyncram:mem_rtl_0" with the following parameter:
> 12021 Found 1 design units, including 1 entities, in source file db/altsyncram_ema1.tdf
```

Post Synthesis and Mapping Schematic

❑ Single-Port Embedded Block RAM with Asynchronous Read



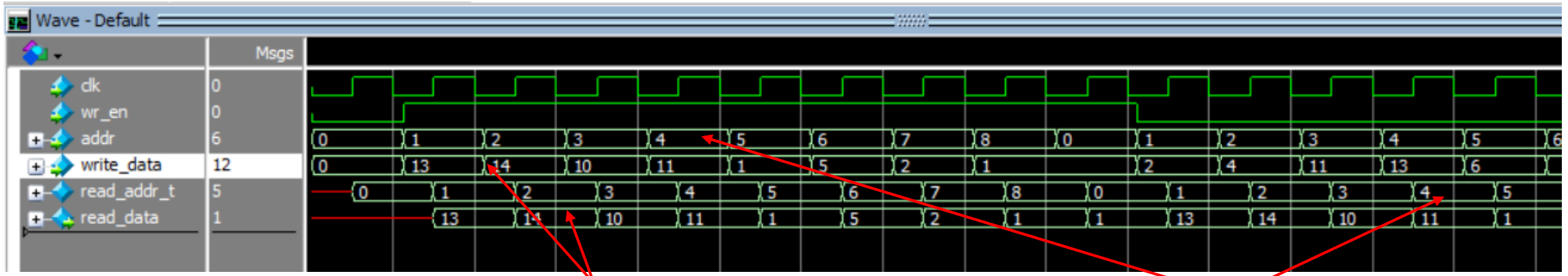
Embedded Block Ram already registers Read address hence post mapping netlist will not show externally placed register (flipflop) before read address port



Embedded Block Ram Cells inferred by Synthesizer

Simulation Result For Single-Port RAM with Asynchronous Read

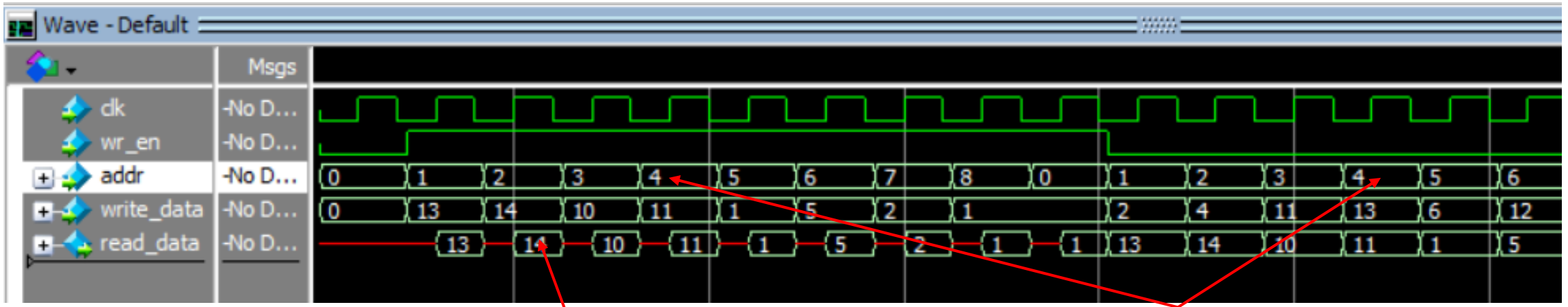
❑ Embedded Block RAM Simulation Result



upon each write to memory, read is automatically available in next clock cycle

During read, data returned is immediate and also last written data on this address
Read on addr=4, returns last value return which in 11

❑ Distributed RAM Simulation Result



upon each write to memory, read is automatically available however it is less than 1 cycle pulse since address changed

Read on addr=4, returns last value return which in 11

Single Port Distributed RAM & Block RAM with Synchronous Read

```
// Single-port Distributed RAM with Synchronous Read (Read Through)
module single_port_distributed_ram_model3 #(
  parameter DATA_WIDTH=4,
  parameter ADDR_WIDTH=4 )(
  input logic clk, rstn,  // added reset
  input logic wr_en,
  input logic[DATA_WIDTH-1:0] write_data,
  input logic[ADDR_WIDTH-1:0] addr,
  output logic[DATA_WIDTH-1:0] read_data);

 // Two-dimensional memory array
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

 // Synchronous write and read
  always_ff@(posedge clk) begin
    if(wr_en) mem[addr]  <= write_data;
    if(!rstn) read_data  <= 0;
    else read_data <= mem[addr];
  end
endmodule
```

Distributed Memory will have Synchronous read behavior

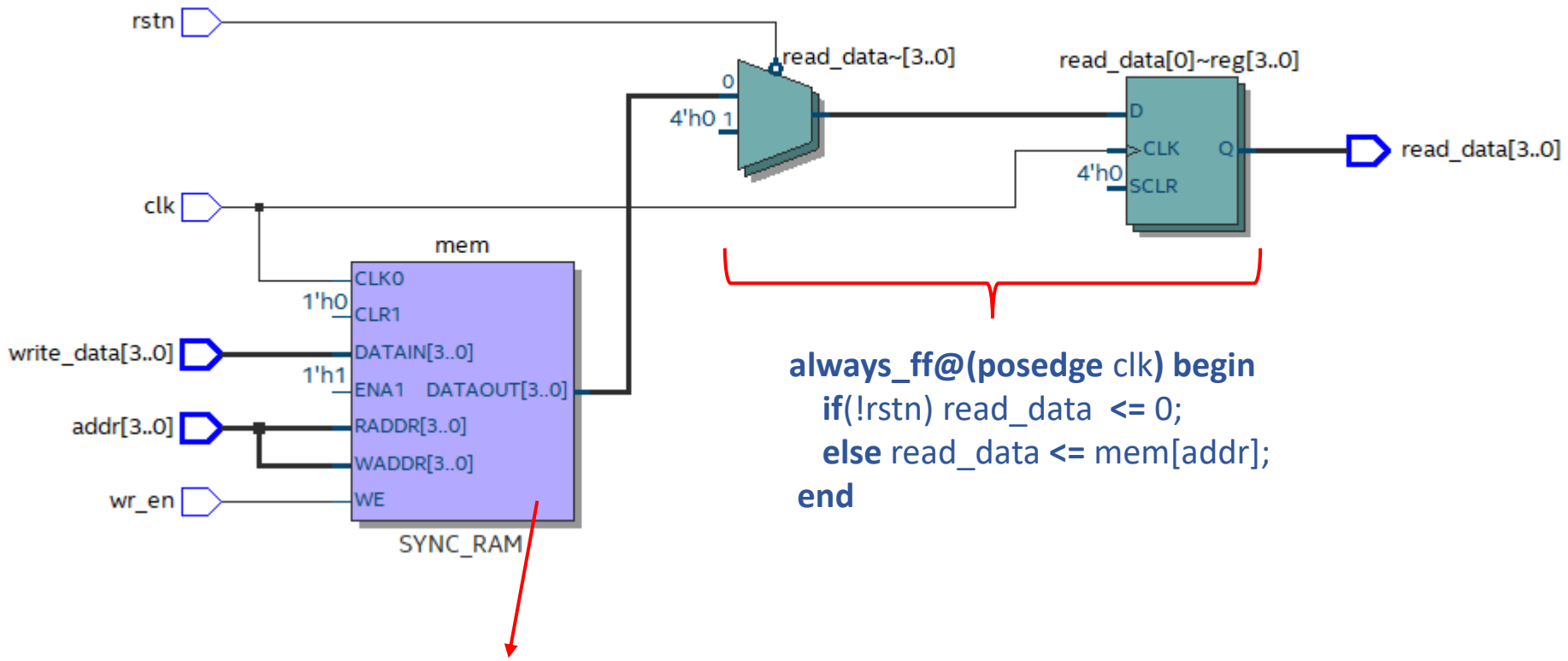
Memory model with reset for read data output are only mappable onto distributed RAM. No Embedded Block Ram inferred by Synthesizer

```
// Single-port Block RAM with Synchronous Read (Read Through)
module single_port_block_ram_model4 #(
  parameter DATA_WIDTH=4,
  parameter ADDR_WIDTH=4
)(
  input logic clk,
  input logic wr_en,
  input logic[DATA_WIDTH-1:0] write_data,
  input logic[ADDR_WIDTH-1:0] addr,
  output logic[DATA_WIDTH-1:0] read_data);

 // Two dimensional memory array
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

 // Synchronous write and read
  always_ff@(posedge clk) begin
    if(wr_en) mem[addr]  <= write_data;
    else read_data <= mem[addr];
  end
endmodule
```

This Memory model will infer embedded Block Ram with Synchronous Read

Post Synthesis and Mapping Schematic

❑ Single-Port Distributed Block RAM with Asynchronous Read (Read Through)

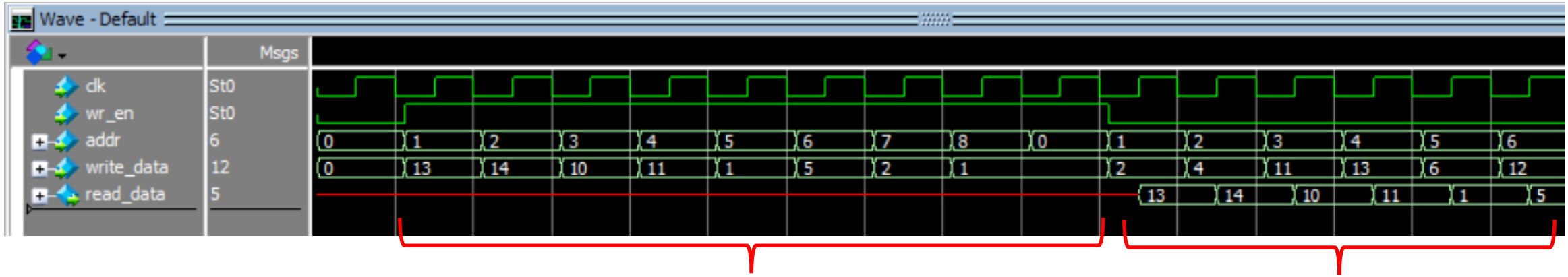



```
always_ff@(posedge clk) begin
   if(!rstn) read_data  <= 0;
   else read_data <= mem[addr];
end
```

Synthesizer will use logic cells, ALUTs and dedicated registers to implement memory model since embedded block RAM IP does not support reset for read data output

Simulation Result For Single Port RAM with Synchronous Read

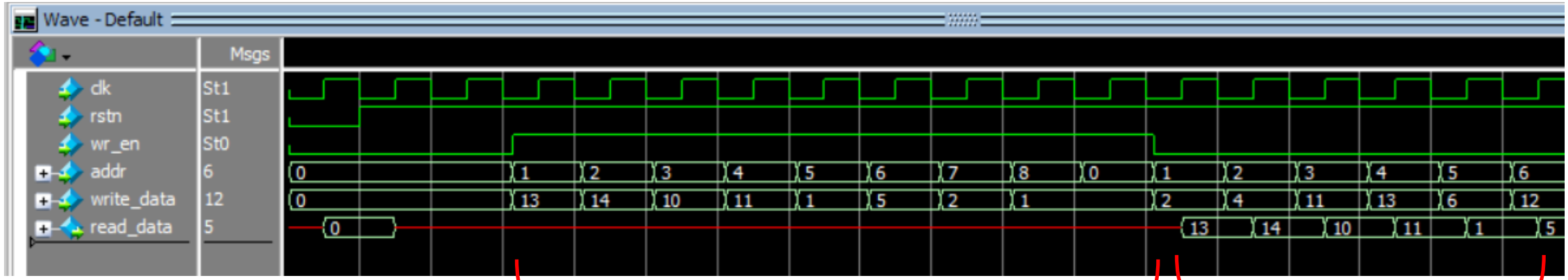
❑ Embedded Block RAM Simulation Result



During write operation no read data available. Only data written to memory

During read operation, read data is sync to clock & available for 1 full clock and no write data committed to memory

❑ Distributed RAM Simulation Result



Same behavior has Block RAM. During write operation no read data available. Only data written to memory

Same behavior as for Block RAM. Read data sync to clock.

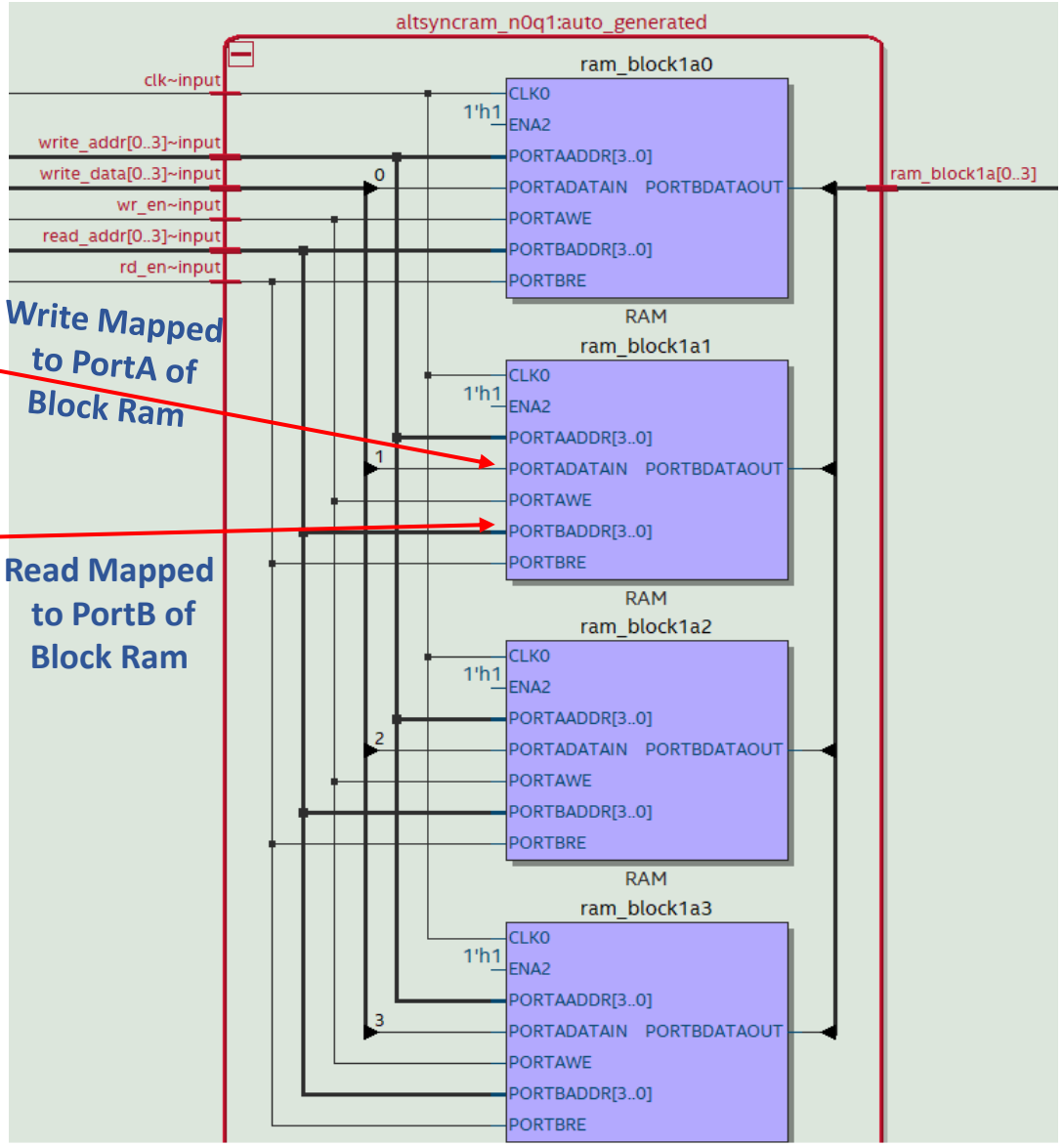
Simple Dual Port Single Clock RAM with Simultaneous Read and Write

```
// Simple Dual-port Single Clock Block RAM with Synchronous Read
module simple_dual_port_block_ram_model5 #(
  parameter DATA_WIDTH=4, parameter ADDR_WIDTH=4)
(
  input logic clk,
  input logic wr_en,
  input logic[DATA_WIDTH-1:0] write_data,
  input logic[ADDR_WIDTH-1:0] write_addr,

  input logic rd_en,
  input logic[ADDR_WIDTH-1:0] read_addr,
  output logic[DATA_WIDTH-1:0] read_data);

  // Two dimensional memory array
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

  // Synchronous write and read
  always_ff@(posedge clk) begin
    if(wr_en) mem[write_addr]  <= write_data;
    if(rd_en) read_data <= mem[read_addr];
  end
endmodule
```

Write data, address, enable ports

Read data, address, enable ports

Simultaneous write and read to different memory location. Write and read operations in same clock domain

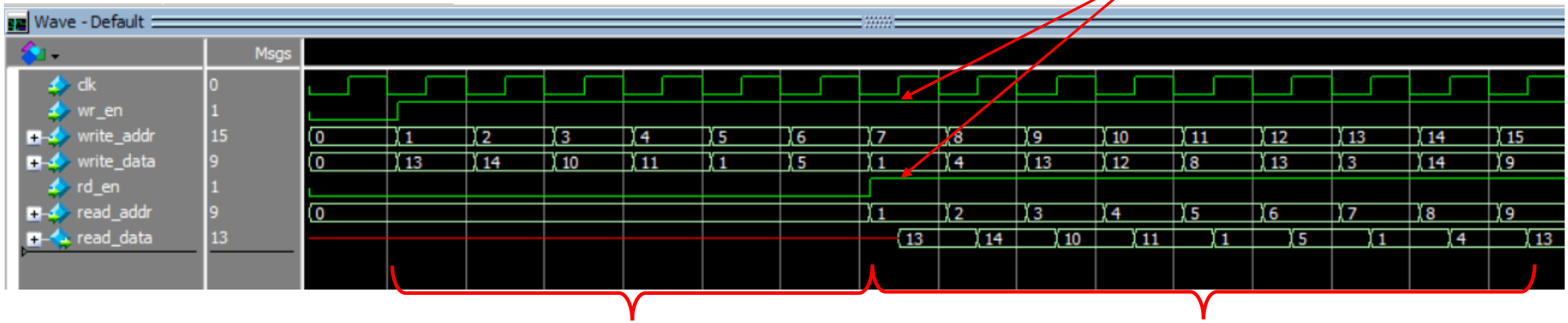


Note: In case of same write and read address, model can return older data on read_data port before writing new data to memory location

Simple Dual Port Single Clock RAM with Simultaneous Read and Write

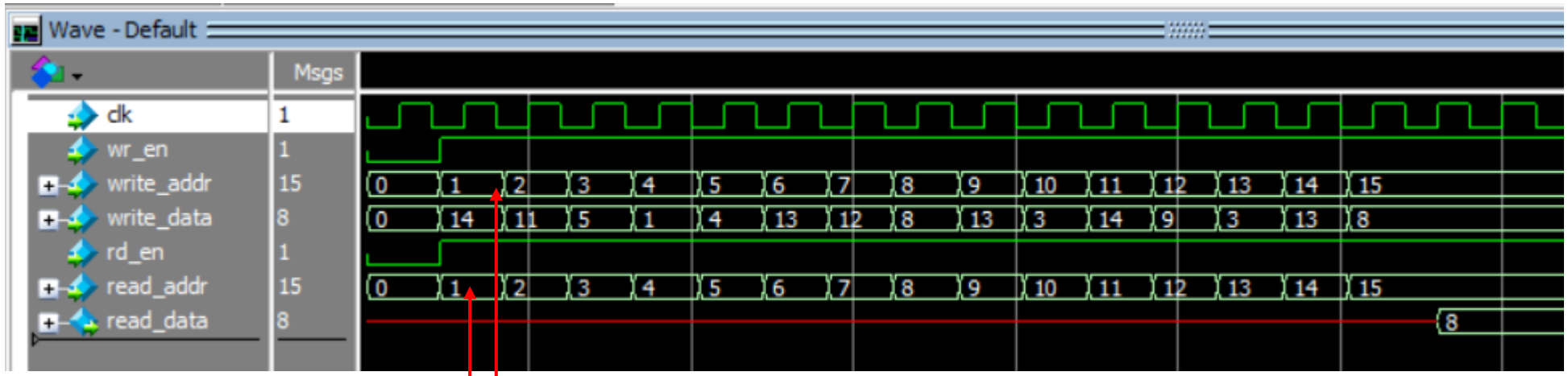
❑ Embedded Block RAM Simulation Result

Both Write and Read Enables are set to '1'
Simultaneous read and write operation to RAM on different addresses



Only write operation to RAM during this period.

Both write and read operation to RAM during this period.



In case of simultaneous write and read operation to same address, RAM will prioritize write operation over read. And there will be no data returned for read when write is being performed.

Simple Dual Port Dual Clock RAM with Simultaneous Read and Write

```
// Simple Dual-port Dual Clock Block RAM with Synchronous Read
module simple_dual_port_block_ram_model6 #(
 parameter DATA_WIDTH=4, parameter ADDR_WIDTH=4)(

 input logic wr_clk, wr_en,
 input logic[DATA_WIDTH-1:0] write_data,
 input logic[ADDR_WIDTH-1:0] write_addr,

 input logic rd_en, rd_clk,
 input logic[ADDR_WIDTH-1:0] read_addr,
 output logic[DATA_WIDTH-1:0] read_data);

  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

  always_ff@(posedge wr_clk) begin
   if(wr_en) mem[write_addr]  <= write_data;
  end

  always_ff@(posedge rd_clk) begin
    if(rd_en) read_data <= mem[read_addr];
  end
endmodule
```

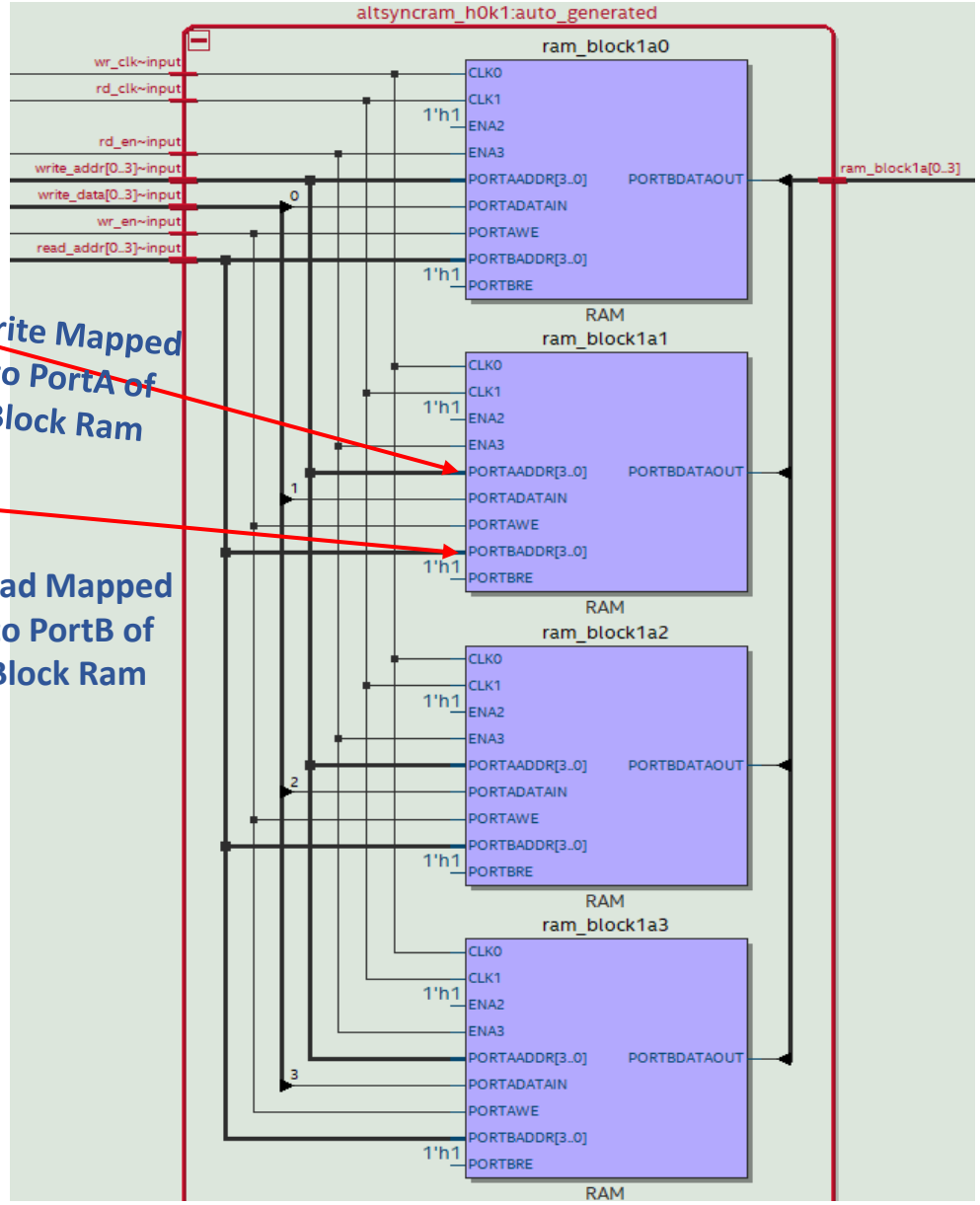
Write clk, data, address, enable ports

Read clk, data, address, enable ports

Simultaneous write and read to different memory location. Write and read operations using different clocks (wr_clk and rd_clk)

Write Mapped to PortA of Block Ram

Read Mapped to PortB of Block Ram



True Dual Port Dual Clock RAM

```
//True Dual-port Block RAM with Dual Clock
module true_dual_port_block_ram_model7 #(
  parameter DATA_WIDTH=4, parameter ADDR_WIDTH=4)
(
  input logic[DATA_WIDTH-1:0] data_a, data_b,
  input logic[ADDR_WIDTH-1:0] addr_a, addr_b,
  input logic we_a, we_b, rd_a, rd_b, clk_a, clk_b,
  output logic[DATA_WIDTH-1:0] q_a, q_b);

  // Declare the RAM variable
  logic[DATA_WIDTH-1:0] mem[2**ADDR_WIDTH-1:0];

  // Port A
  always_ff@(posedge clk_a) begin
    if(we_a) mem[addr_a] <= data_a;
    else if(rd_a) q_a <= mem[addr_a];
  end

  // Port B
  always_ff@(posedge clk_b) begin
    if(we_b) mem[addr_b] <= data_b;
    else if(rd_b) q_b <= mem[addr_b];
  end
endmodule
```

Dual PortA and PortB

- Simultaneous write and read from different ports
- Simultaneous two reads from different ports
- Simultaneous two writes from different ports