Hardware Modeling [VU] (191.011)
– WS25 –
RAM in VHDL (for FPGAs)

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 16:03 (f8a58e9)

Introduction

- On-chip RAM is a fundamental building block in digital design

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower
- Where do we need RAM in a design?

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower
- Where do we need RAM in a design?
  - Buffers inside or in between modules (often in the form of FIFO buffers)

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower
- Where do we need RAM in a design?
  - Buffers inside or in between modules (often in the form of FIFO buffers)
  - Caches for frequently accessed data in some external memory

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower
- Where do we need RAM in a design?
  - Buffers inside or in between modules (often in the form of FIFO buffers)
  - Caches for frequently accessed data in some external memory
  - Large look-up tables

Introduction

- On-chip RAM is a fundamental building block in digital design
- Fast, low-latency, flexible data storage
- Compared to flip-flops/latches, RAM ...
  - uses an address-based access scheme
  - is more compact in terms of area and, hence, more power efficient
  - allows for higher-capacity memory
  - is slightly slower
- Where do we need RAM in a design?
  - Buffers inside or in between modules (often in the form of FIFO buffers)
  - Caches for frequently accessed data in some external memory
  - Large look-up tables
  - Program and data memory for processors to store instructions and data

Introduction (cont'd)

- RAM capacity (C)

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$
- Memory access port types

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$
- Memory access port types
  - read
  - write
  - read/write

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$
- Memory access port types
  - read
  - write
  - read/write
- Multiple independent ports are possible

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$
- Memory access port types
  - read
  - write
  - read/write
- Multiple independent ports are possible
  - Simple dual-port RAM: one read and one write port

Introduction (cont'd)

- RAM capacity (C)
  - Data width (W): size of each data element stored per address
  - Depth (D): number of addressable memory locations
    - usually defined by the address width A
    - $\Rightarrow D = 2^A$
  - $C = W \times D$
- Memory access port types
  - read
  - write
  - read/write
- Multiple independent ports are possible
  - Simple dual-port RAM: one read and one write port
  - True dual-port RAM: two read/write ports

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width
  - number and type of access ports
  - control signals (e.g., read/write/byte enable)

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width
  - number and type of access ports
  - control signals (e.g., read/write/byte enable)
  - parity bits (error detection)

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width
  - number and type of access ports
  - control signals (e.g., read/write/byte enable)
  - parity bits (error detection)
- Memory in modern FPGAs is (almost) always synchronous

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width
  - number and type of access ports
  - control signals (e.g., read/write/byte enable)
  - parity bits (error detection)
- Memory in modern FPGAs is (almost) always synchronous
  - Read and write operations only happen at a clock edge

Memory Blocks in FPGAs

- Memory blocks are an important FPGA resource
  - Logic-based memory (often) impractical
  - Highest speed/lowest latency memory in a design
- Highly configurable in terms of
  - data width
  - address width
  - number and type of access ports
  - control signals (e.g., read/write/byte enable)
  - parity bits (error detection)
- Memory in modern FPGAs is (almost) always synchronous
  - Read and write operations only happen at a clock edge
  - Some older FPGAs support asynchronous memory

Intel FPGAs

Table 3. 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

Source: Embedded Memory User Guide

Intel FPGAs

Table 3. 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

Source: Embedded Memory User Guide

Intel FPGAs

Table 3. 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

Source: Embedded Memory User Guide

Simple Dual-Port RAM


```
1  entity simple_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read port
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
12   -- write port
13   wr_en : in std_ulogic;
14   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
15   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
16  );
17 end entity;
```

Simple Dual-Port RAM


```
1  entity simple_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read port
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
12   -- write port
13   wr_en : in std_ulogic;
14   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
15   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
16  );
17 end entity;
```

Generics to set the data and address width of the memory.

Simple Dual-Port RAM


```
1  entity simple_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read port
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
12   -- write port
13   wr_en : in std_ulogic;
14   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
15   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
16  );
17 end entity;
```

The clock signal that controls all operations on both the read and the write port.

Simple Dual-Port RAM


```
1  entity simple_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read port
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
12   -- write port
13   wr_en : in std_ulogic;
14   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
15   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
16  );
17 end entity;
```

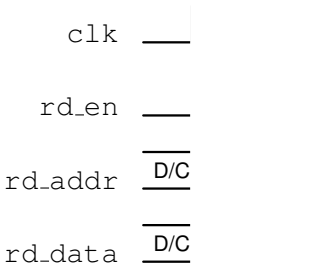
Memory read port.

Simple Dual-Port RAM

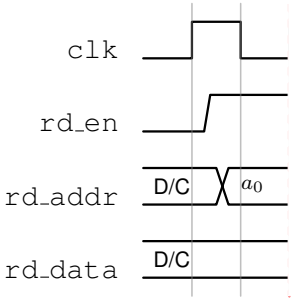

```
1  entity simple_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read port
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
12   -- write port
13   wr_en : in std_ulogic;
14   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
15   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
16  );
17 end entity;
```

Memory write port.

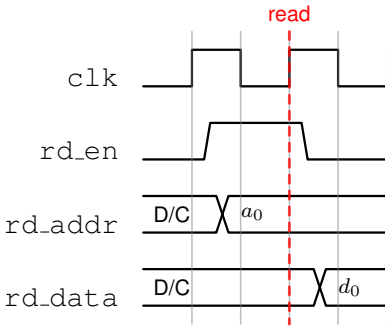
Read Access



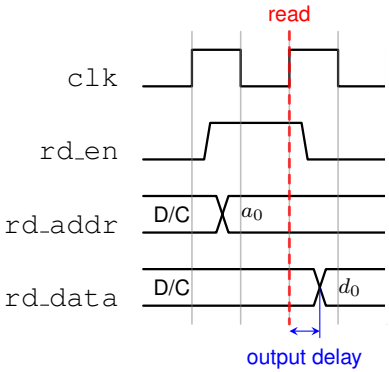
Read Access



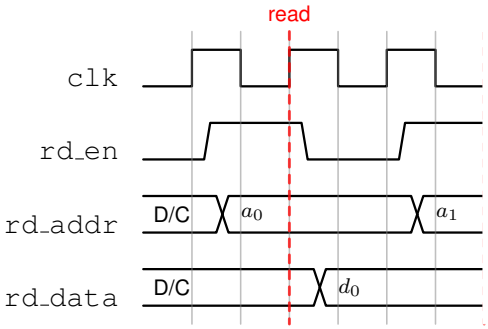
Read Access



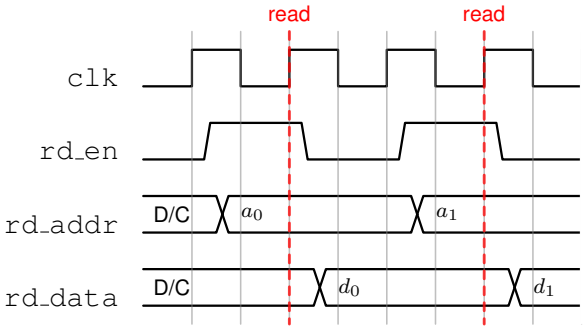
Read Access



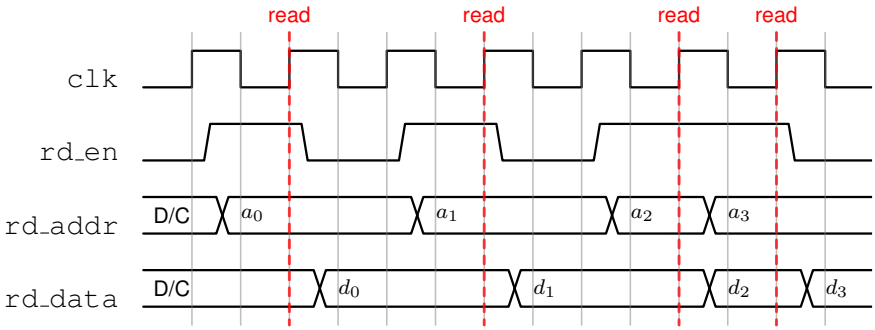
Read Access



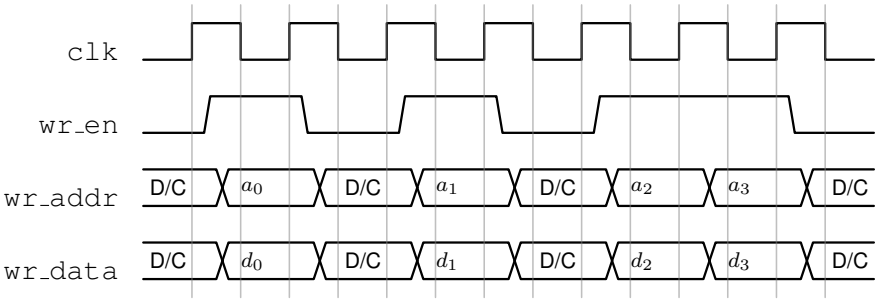
Read Access



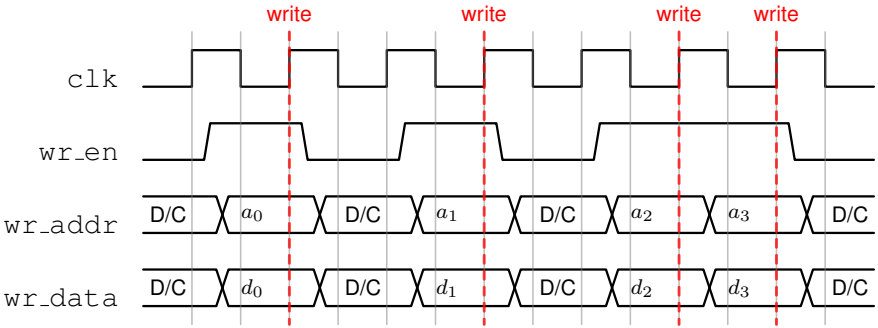
Read Access


read
read
read
read
clk
rd_en
rd_addr
D/C
a_0
a_1
a_2
a_3
rd_data
D/C
d_0
d_1
d_2
d_3

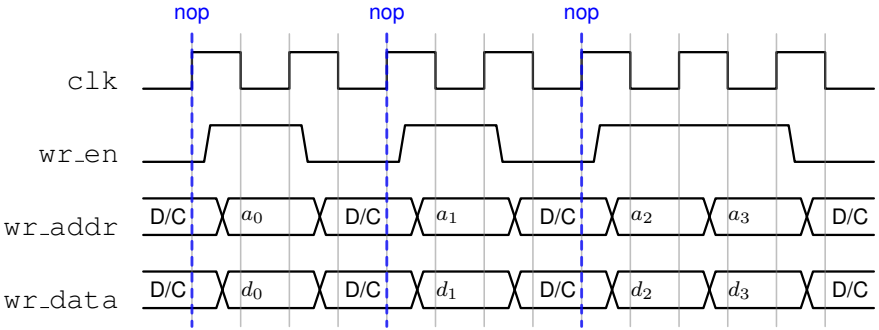
Write Access



Write Access



Write Access



True Dual-Port RAM


```
1  entity true_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read/write port 0
9    rw0_rd_en : in std_ulogic;
10   rw0_wr_en : in std_ulogic;
11   rw0_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
12   rw0_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
13   rw0_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
14   -- read/write port 1
15   rw1_rd_en : in std_ulogic;
16   rw1_wr_en : in std_ulogic;
17   rw1_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
18   rw1_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
19   rw1_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0)
20  );
21 end entity;
```

True Dual-Port RAM

```
1  entity true_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read/write port 0
9    rw0_rd_en : in std_ulogic;
10   rw0_wr_en : in std_ulogic;
11   rw0_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
12   rw0_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
13   rw0_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
14   -- read/write port 1
15   rw1_rd_en : in std_ulogic;
16   rw1_wr_en : in std_ulogic;
17   rw1_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
18   rw1_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
19   rw1_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0)
20  );
21 end entity;
```

First read/write memory port.

True Dual-Port RAM


```
1  entity true_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    clk : in std_ulogic;
8    -- read/write port 0
9    rw0_rd_en : in std_ulogic;
10   rw0_wr_en : in std_ulogic;
11   rw0_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
12   rw0_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
13   rw0_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
14   -- read/write port 1
15   rw1_rd_en : in std_ulogic;
16   rw1_wr_en : in std_ulogic;
17   rw1_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
18   rw1_wr_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0);
19   rw1_rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0)
20  );
21 end entity;
```

Second read/write memory port.

Dual-Clock RAM

- Separate/independent clock for each port of a dual-port memory $\Rightarrow$ dual-clock memory
- Use case: clock-domain-crossing interfaces

Dual-Clock RAM

- Separate/independent clock for each port of a dual-port memory $\Rightarrow$ dual-clock memory
- Use case: clock-domain-crossing interfaces
- Usually supported by FPGA block RAM

Dual-Clock RAM

- Separate/independent clock for each port of a dual-port memory $\Rightarrow$ dual-clock memory
- Use case: clock-domain-crossing interfaces
- Usually supported by FPGA block RAM
- Simultaneous read and write to the same location must be handled carefully

Dual-Clock RAM

- Separate/independent clock for each port of a dual-port memory $\Rightarrow$ dual-clock memory
- Use case: clock-domain-crossing interfaces
- Usually supported by FPGA block RAM
- Simultaneous read and write to the same location must be handled carefully
- $\Rightarrow$ dual-clocked/bisynchronous FIFOs

Dual-Clocked RAM - Example


```
1  entity dualclock_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    -- read port
8    rd_clk : in std_ulogic;
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0)
12   -- write port
13   wr_clk : in std_ulogic;
14   wr_en : in std_ulogic;
15   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
16   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
17  );
18 end entity;
```

Dual-Clocked RAM - Example


```
1  entity dualclock_dp_ram is
2   generic (
3    ADDR_WIDTH : positive;
4    DATA_WIDTH : positive
5   );
6   port (
7    -- read port
8    rd_clk : in std_ulogic;
9    rd_en : in std_ulogic;
10   rd_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
11   rd_data : out std_ulogic_vector(DATA_WIDTH - 1 downto 0)
12   -- write port
13   wr_clk : in std_ulogic;
14   wr_en : in std_ulogic;
15   wr_addr : in std_ulogic_vector(ADDR_WIDTH - 1 downto 0);
16   wr_data : in std_ulogic_vector(DATA_WIDTH - 1 downto 0)
17  );
18 end entity;
```

RAM Implementation

	Vendor-Library Instantiation	Synthesis Inference

RAM Implementation

	Vendor-Library Instantiation	**Synthesis Inference**
Portability/Flexibility	Limited to vendor/device, requires vendor-specific knowledge	Generic and portable, simpler to work with and maintain

RAM Implementation

	Vendor-Library Instantiation	**Synthesis Inference**
Portability/Flexibility	Limited to vendor/device, requires vendor-specific knowledge	Generic and portable, simpler to work with and maintain
Performance	Optimized for target hardware	May be suboptimal

RAM Implementation

	Vendor-Library Instantiation	**Synthesis Inference**
Portability/Flexibility	Limited to vendor/device, requires vendor-specific knowledge	Generic and portable, simpler to work with and maintain
Performance	Optimized for target hardware	May be suboptimal
Special Features	Fully supported (e.g., ECC)	May be unavailable

RAM Implementation

	Vendor-Library Instantiation	**Synthesis Inference**
Portability/Flexibility	Limited to vendor/device, requires vendor-specific knowledge	Generic and portable, simpler to work with and maintain
Performance	Optimized for target hardware	May be suboptimal
Special Features	Fully supported (e.g., ECC)	May be unavailable
Predictability	Deterministic	May cause mismatch in synthesis/simulation tool

Inferred RAM


```
1  architecture beh of simple_dp_ram is
2   subtype ram_entry_t is std_ulogic_vector(DATA_WIDTH - 1 downto 0);
3   type ram_t is array(0 to (2 ** ADDR_WIDTH) - 1) of ram_entry_t;
4   signal ram : ram_t := (others => (others => '0'));
5  begin
6   process(clk)
7   begin
8    if rising_edge(clk) then
9     if wr_en = '1' then
10     ram(to_integer(unsigned(wr_addr))) <= wr_data;
11    end if;
12    if rd_en = '1' then
13     rd_data <= ram(to_integer(unsigned(rd_addr)));
14    end if;
15   end if;
16  end process;
17 end architecture;
```

Inferred RAM


```
1  architecture beh of simple_dp_ram is
2   subtype ram_entry_t is std_ulogic_vector(DATA_WIDTH - 1 downto 0);
3   type ram_t is array(0 to (2 ** ADDR_WIDTH) - 1) of ram_entry_t;
4   signal ram : ram_t := (others => (others => '0'));
5  begin
6   process(clk)
7   begin
8    if rising_edge(clk) then
9     if wr_en = '1' then
10     ram(to_integer(unsigned(wr_addr))) <= wr_data;
11    end if;
12    if rd_en = '1' then
13     rd_data <= ram(to_integer(unsigned(rd_addr)));
14    end if;
15   end if;
16  end process;
17 end architecture;
```

Type definitions and signal declaration for the signal that actually represents the memory array.

Inferred RAM


```
1  architecture beh of simple_dp_ram is
2   subtype ram_entry_t is std_ulogic_vector(DATA_WIDTH - 1 downto 0);
3   type ram_t is array(0 to (2 ** ADDR_WIDTH) - 1) of ram_entry_t;
4   signal ram : ram_t := (others => (others => '0'));
5  begin
6   process(clk)
7   begin
8    if rising_edge(clk) then
9     if wr_en = '1' then
10     ram(to_integer(unsigned(wr_addr))) <= wr_data;
11    end if;
12    if rd_en = '1' then
13     rd_data <= ram(to_integer(unsigned(rd_addr)));
14    end if;
15   end if;
16  end process;
17 end architecture;
```

Synchronous process similar to what is used in the description of flip-flops.

Inferred RAM


```
1  architecture beh of simple_dp_ram is
2   subtype ram_entry_t is std_ulogic_vector(DATA_WIDTH - 1 downto 0);
3   type ram_t is array(0 to (2 ** ADDR_WIDTH) - 1) of ram_entry_t;
4   signal ram : ram_t := (others => (others => '0'));
5  begin
6   process(clk)
7   begin
8    if rising_edge(clk) then
9     if wr_en = '1' then
10     ram(to_integer(unsigned(wr_addr))) <= wr_data;
11    end if;
12    if rd_en = '1' then
13     rd_data <= ram(to_integer(unsigned(rd_addr)));
14    end if;
15   end if;
16  end process;
17 end architecture;
```

Implementation of the write port.

Inferred RAM


```
1  architecture beh of simple_dp_ram is
2   subtype ram_entry_t is std_ulogic_vector(DATA_WIDTH - 1 downto 0);
3   type ram_t is array(0 to (2 ** ADDR_WIDTH) - 1) of ram_entry_t;
4   signal ram : ram_t := (others => (others => '0'));
5  begin
6   process(clk)
7   begin
8    if rising_edge(clk) then
9     if wr_en = '1' then
10     ram(to_integer(unsigned(wr_addr))) <= wr_data;
11    end if;
12    if rd_en = '1' then
13     rd_data <= ram(to_integer(unsigned(rd_addr)));
14    end if;
15   end if;
16  end process;
17 end architecture;
```

Implementation of the read port.

Lecture Complete!