Lab Session #3 – Theory

Verilog: Memories

Digital Design Concepts – Digitale ontwerpprincipes

Outline

- **Part #1**: Digital Memories

- **Part #2**: Verilog – Describing Memories

Objectives

After completing this session, you will be able to:

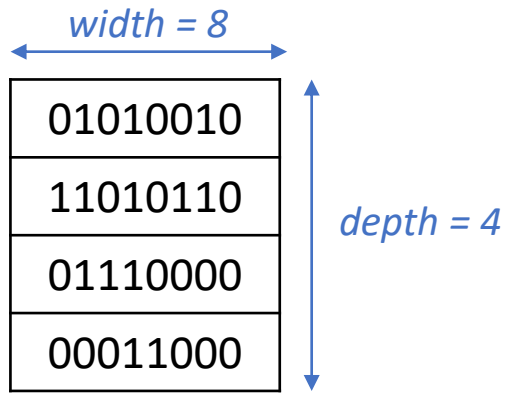
- Understand the interfaces and concept of ROM and RAM memories
- Discern between single port and dual-port memories
- Describe memories in Verilog using always blocks
- Initialize memory contents in Verilog using the *$readmemb* macro

Part #1
Digital Memories

Memories

- Digital components for **large storage**
- In digital design, we can model a memory as an **array** parametrized by:
  1. **Depth**: number of cells in the memory
  2. **Width**: length (in bits) of each cell

Example: 4 x 8-bit memory



- Memory cells are indexed using an **address**
- How many bits should an address have?

$$address\ bits = \lceil log_2(depth) \rceil$$

- All digital memories we will use operate **synchronously** with the clock signal

Examples

Memory Depth	Address Bits
7	3
8	4
9	4
255	8

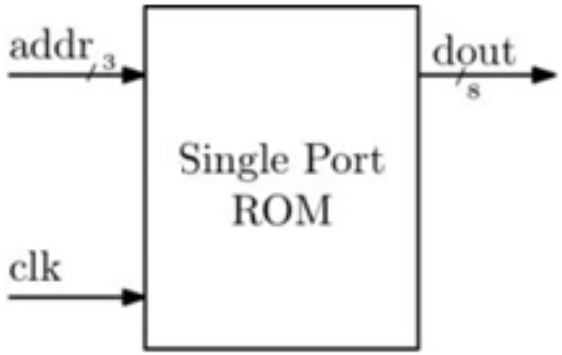
ROM Memories

- Property: contents cannot be modified
- In other words: they are **read-only**
- Interfaces:
  - **Address** (input port) : selects a memory cell
  - **Data Out** (output port) : returns the cell contents

- Concept:

  " at each rising edge of the **clock** signal,
  the port **data out** returns the contents
  of the cell pointed by **address** "

Example: single-port ROM memory



Notice that in this example:

- the *addr* port is 3 bits, hence the maximum **depth** of this memory can be $2^3 = 8$ cells
- the *dout* port is 8 bits, hence the **width** of this memory is 8 bits

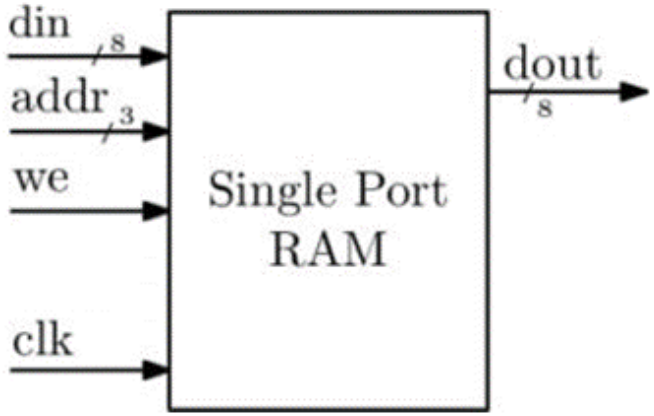
RAM Memories

- Property: contents can be modified
- In other words: they are **read + write**
- Interfaces:
  - **Address** (input port) : selects a memory cell
  - **Data Out** (output port) : returns the cell contents when reading
  - **Data In** (input port) : provides the cell contents when writing
  - **Write Enable** (input port) : indicates to the memory whether we want to perform a read or a write

- Concept:

" at each rising edge of the **clock** signal, the **write enable** port is checked:
if it equals 1, the contents of the cell pointed by **address** are overwritten with the value in **data in**;
otherwise, the port **data out** returns the contents of the cell pointed by **address** "

- Example: single-port RAM memory

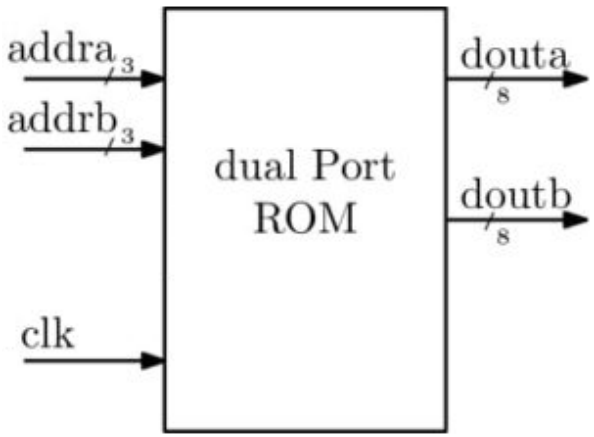


Dual-Port Memories

- In digital applications, it is often the case that two (or more) modules need to access a memory **independently**
- In this situation, a dual- (or multi-) port memory is required
- Such memories can be built by replicating the single-port interfaces

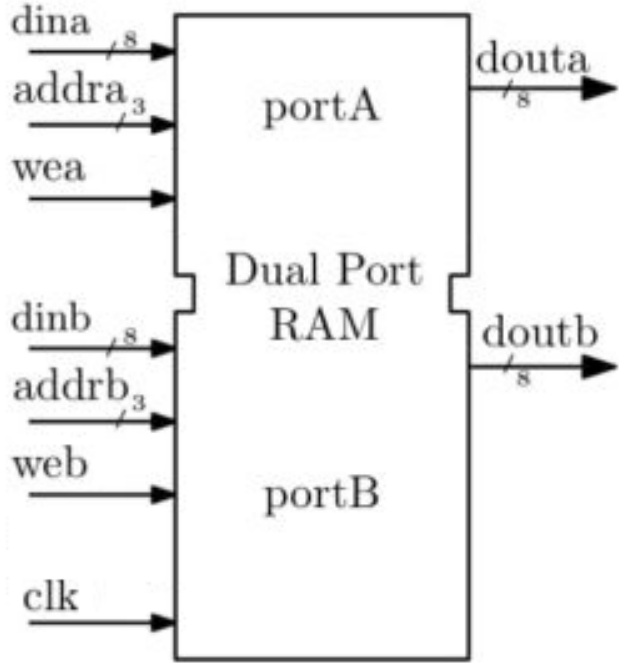
- Example: dual-port ROM
  - Port A for reading (*addrA* and *doutA*)
  - Port B for reading (*addrB* and *doutB*)

- At the same clock cycle, two independent reads can be **simultaneously** performed



Dual-Port Memories

- Example: dual-port RAM
  - Port A for reading + writing
    (*addrA*, *doutA*) + (*addrA, dinA, weA*)
  - Port B for reading + writing
    (*addrB*, *doutB*) + (*addrB, dinB, weB*)

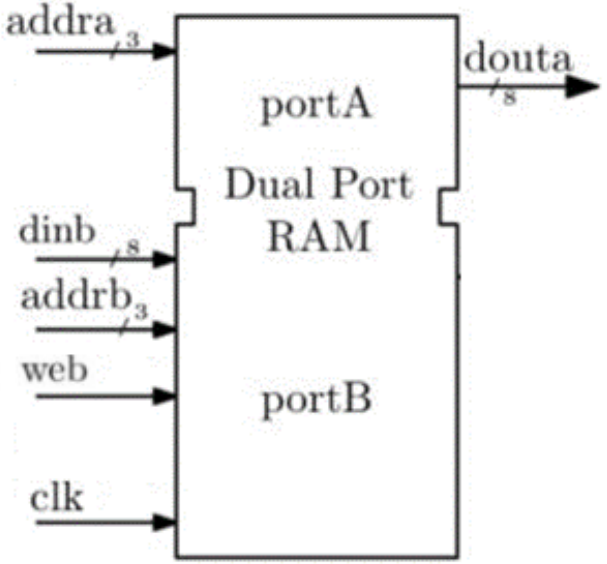


- At the same clock cycle, different operations (read or write) can be **simultaneously** performed

- Important! In this case there needs to be an **arbitration mechanism** to prevent conflicts
- Most common conflict: both ports want to write to the same address
- Out of scope for these labs …

Dual-Port Memories

- Example: **simplified** dual-port RAM
  - Port A for reading
    (*addrA*, *doutA*) + (*addrA, dinA, weA*)
  - Port B for reading + writing
    (*addrB*, *doutB*) + (*addrB, dinB, weB*)

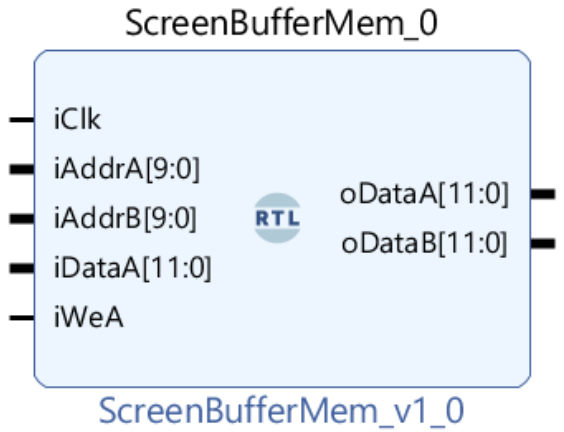


- We will use this simplified memory in our final lab project
- Advantage: no need for arbitration of conflicts, since only one write operation can take place at a certain clock cycle

Part #2
Verilog – Describing Memories

Memories in Verilog

```
module ScreenBufferMem #(
  parameter   WIDTH =  12,
  parameter   DEPTH =  600
  )
  (
  input   wire                          iClk,
  input   wire [$clog2(DEPTH)-1:0]      iAddrA, iAddrB,
  input   wire [WIDTH-1:0]              iDataB,
  input   wire                          iWeB,
  output  wire [WIDTH-1:0]              oDataA,oDataB
  );
```



- Let's see how to create a memory module in Verilog
- We will use as an example a **simplified Dual-Port RAM** with:
  - One reading port (Port A)
  - One reading + writing port (Port B)
- The module declaration contains two parameters:
  - **WIDTH** : number of cells in the memory
  - **DEPTH** : length (in bits) of each cell
- The bit lengths of the data/address ports are automatically adjusted from the parameters:
  - Data ports need have *WIDTH* bits
  - Address ports need to have **$clog2(*DEPTH*)** bits

Memories in Verilog

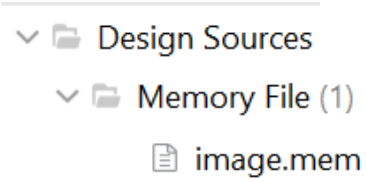
```
// define the memory
reg [WIDTH-1:0] rMem  [DEPTH-1:0];
```

```
// Initial contents of the memory
initial
begin
  $readmemb("image.mem", rMem);
end
```

image.mem

```
1   111111101110
2   111111101110
3   111111101110
4   111111101110
5   111111111110
6   111111111110
7   111111111110
8   111111111110
    ...
597 001000000001
598 001000000000
599 001000000000
600 001000000000
```

- Inside the architecture, we start by creating the memory array
  - Array of type **reg** with **DEPTH** elements of size **WIDTH**
- Verilog provides a macro to fill in initial memory contents: **$readmemb**
- This copies the contents of a memory file (with .mem extension) to the memory array
- The memory file is a text file, with **DEPTH** lines, where each line contains a **WIDTH** bit sequence
  - In this example: 600 lines of 12 bits each
- Important! The memory file needs to be part of the Vivado project
  - You can include it under Design Sources > Memory File



Memories in Verilog

```
// Logic for Port A
//  Supports only synchronous reading

reg [WIDTH-1:0] rDataA;

always @(posedge iClk)
begin
  rDataA <= rMem[iAddrA];
end

assign oDataA = rDataA;

// Logic for Port B
//  Supports synchronous reading and writing

reg [WIDTH-1:0] rDataB;

always @(posedge iClk)
begin
  if(iWeB)
    rMem[iAddrB] <= iDataB;

  rDataB <= rMem[iAddrB];
end

assign oDataB = rDataB;
```

- The logic for Port A is very simple

1. We declare an internal **reg** variable rDataA

This is necessary for the always block (remember it does not allow destination variables of type **wire**)

2. We create a **sequential always block**

Why? This is a memory element that acts synchronously with the clock

3. We connect the internal reg variable (*rDataA*) to the output port (*oDataA*)

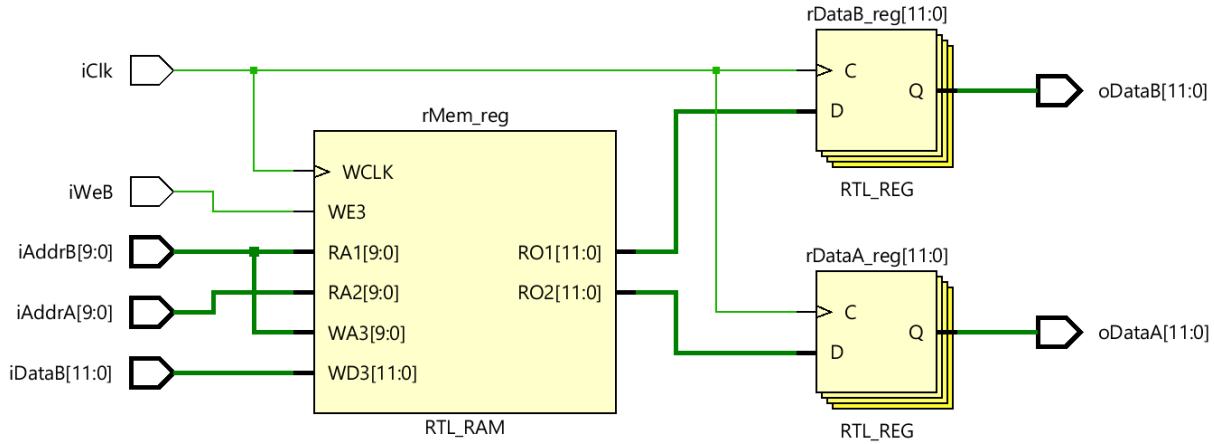
- The logic for Port B is very similar

4. But in the **sequential** always block, we add an **if** statement to check the value of *iWeB*

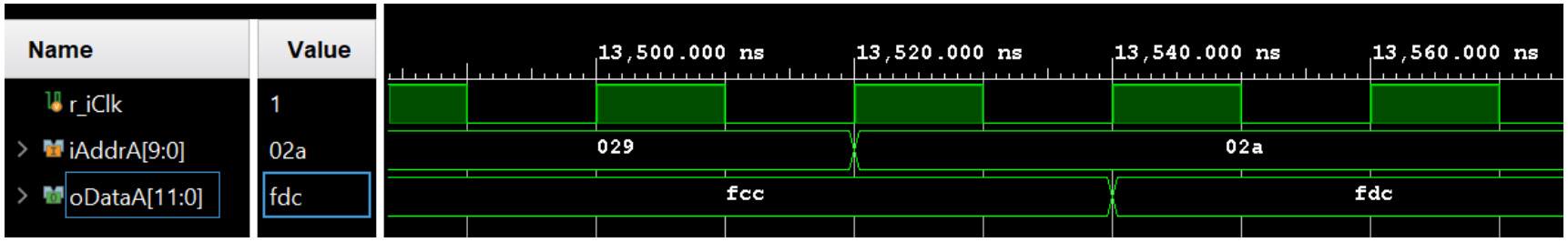
- If *iWeB* is asserted, it will perform a write operation

Memories in Verilog

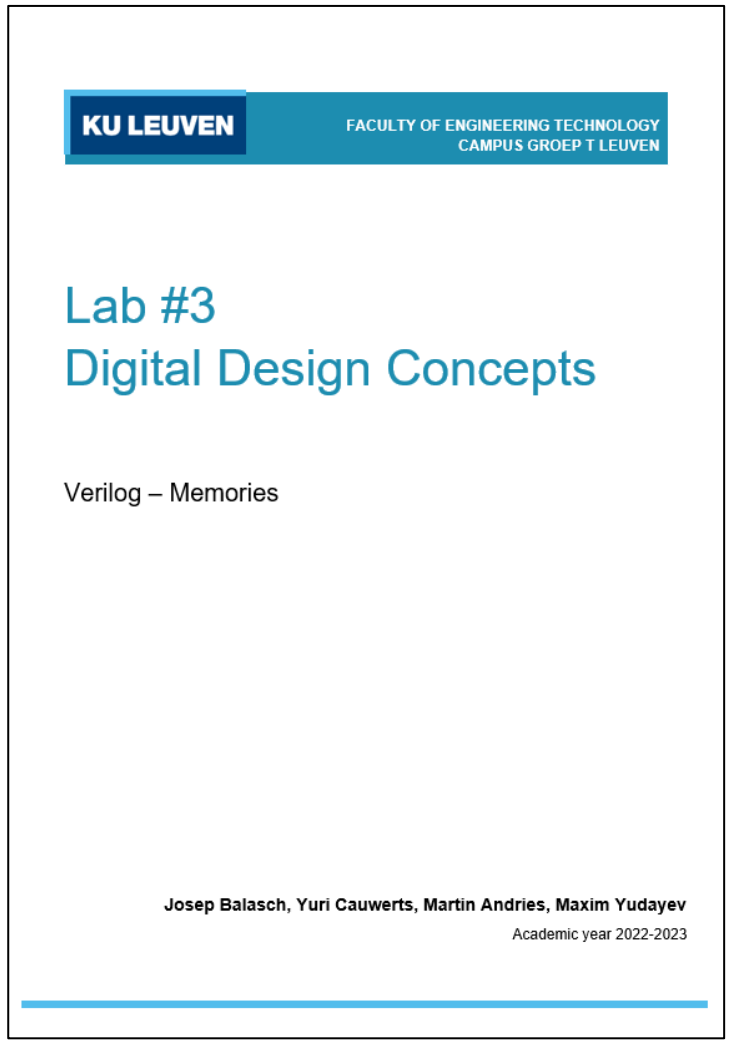
- Validation by looking at Elaborated Design
- A memory component (RTL_RAM) is inferred, with one writing port and two reading ports
- Two registers (RTL_REG) are inferred to sample the memory values before the output



- Important to notice! Both reading and writing operations have **1 cycle latency**
- What does this mean? The operation is performed 1 clock cycle after providing the address
- Example: *iAddrA* = 0x2A @ rising edge 13.52ns, but *iDataA* = 0xFDC comes only cycle later



[Hands-on Time]


KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY
CAMPUS GROEP T LEUVEN

Lab #3
Digital Design Concepts

Verilog – Memories

Josep Balasch, Yuri Cauwerts, Martin Andries, Maxim Yudayev

Academic year 2022-2023

- Now that we have learned some new Verilog concepts, let’s put them to practice !
- Open the handout of Lab #3
- Complete the assignments to get some text displayed in the monitor.