Lesson 5: Memory

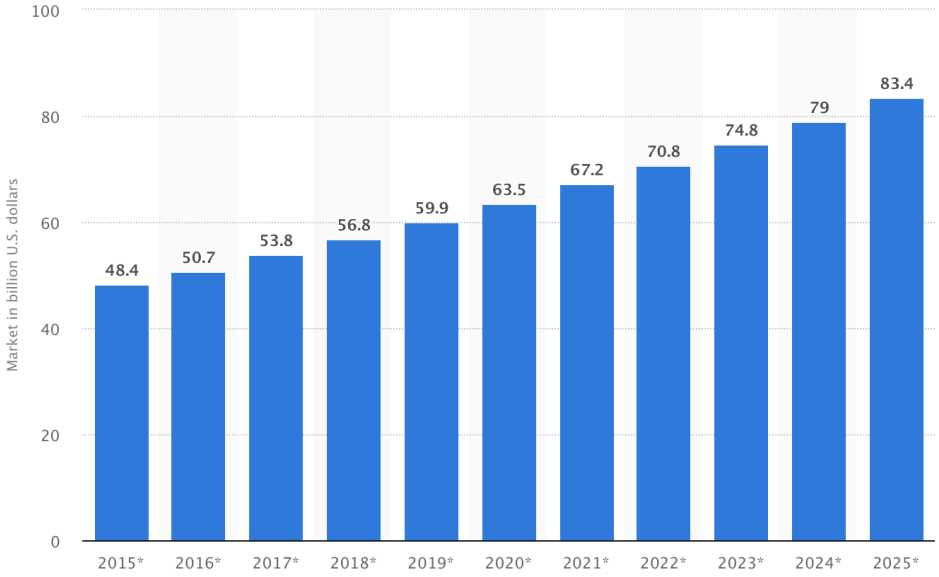
Technologies and organization

Outline

1. Introduction
2. Characteristics:
   - Access methods
   - Parameters
   - Technologies
3. Embedded SRAMs in FPGAs
   - General structure
   - Timing

Where can we find memories?

Size of the market of DRAMs, in billion U.S. dollars
(Source: Statista)



intel Quark™

SoC X1000:

- 16 KB I/D cache
- 512 KB on-die SRAM

(Source: Intel)

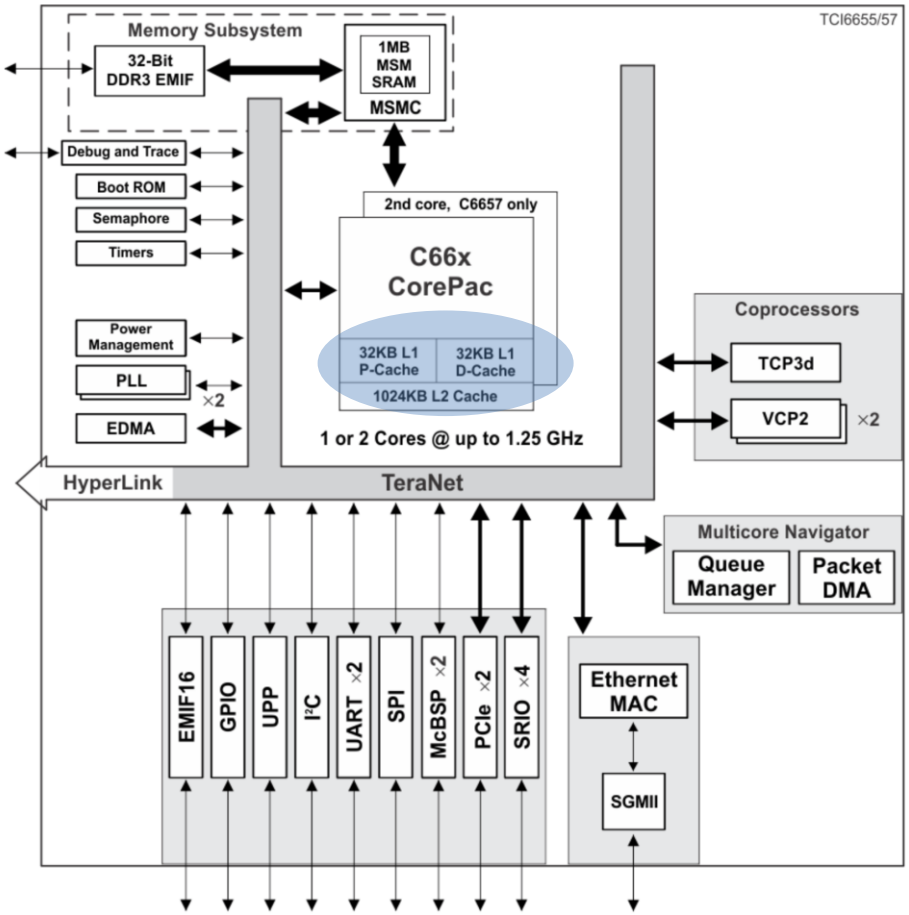
ARM®

Mali-G72 High Performance GPU

- L2 shared cache:128–2048 KiB

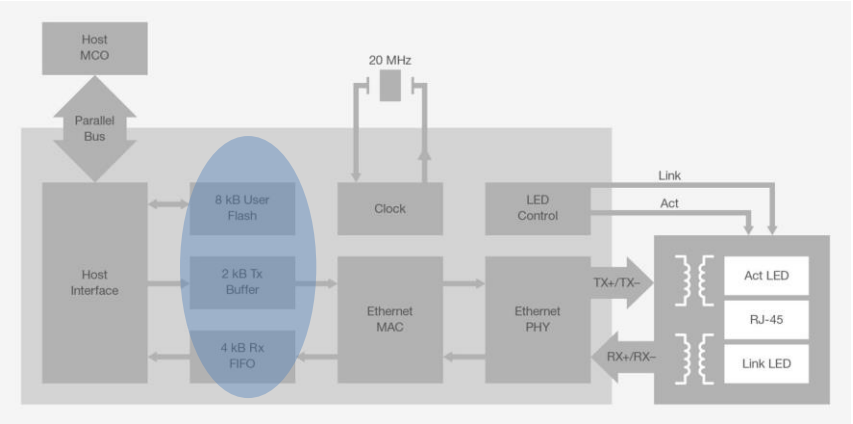
(Source: ARM)

More examples



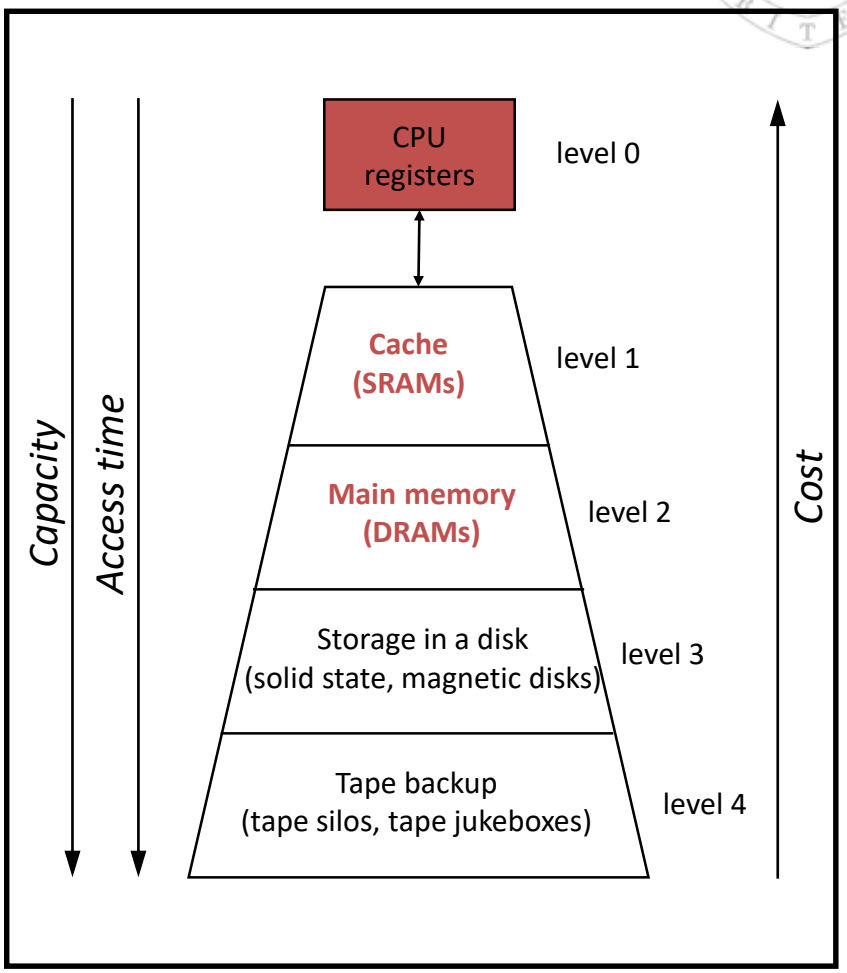
DSP C665x (Source: Texas Instruments Inc.)

CP220x Ethernet Controller (Source: Silicon Labs)



Memory hierarchy

- Typically, the memory available by a computer is organized hierarchically in several levels:
  - CPU registers
  - Cache memory
  - Main memory
  - Secondary memory (disks)
  - CDs - DVDs
- The cost of the whole memory system exceeds the cost of the CPU itself.
  - Optimizing their use is very important.



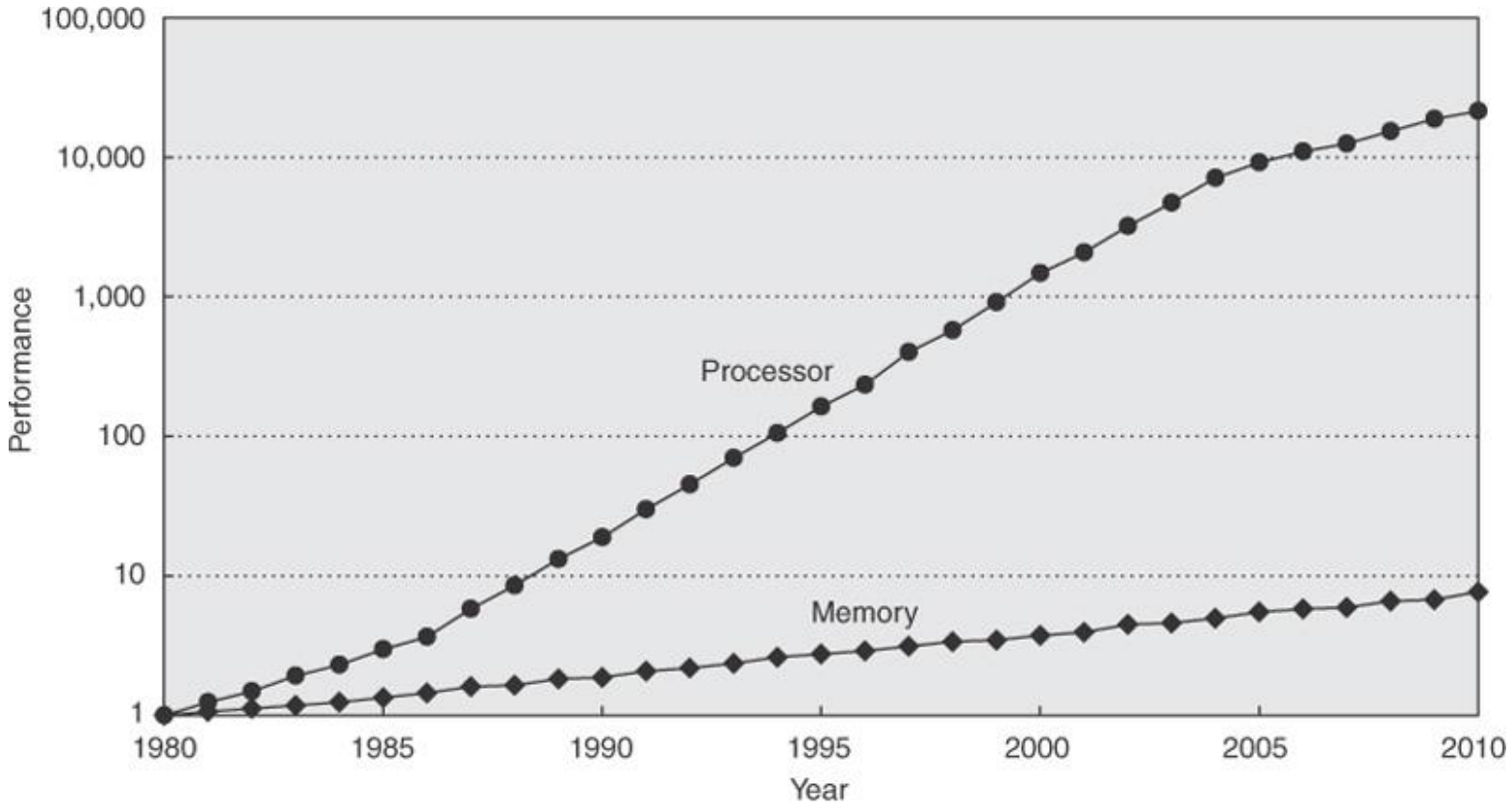
It's the Memory, Stupid! ('96)

- Since the 90's, the processor works increasingly faster than the memory:
  - 3 out of 4 cycles are invested in waiting for the memory to respond (data & instructions memory).
  - In addition, instructions may occupy several consecutive memory positions (several memory accesses/fetch)

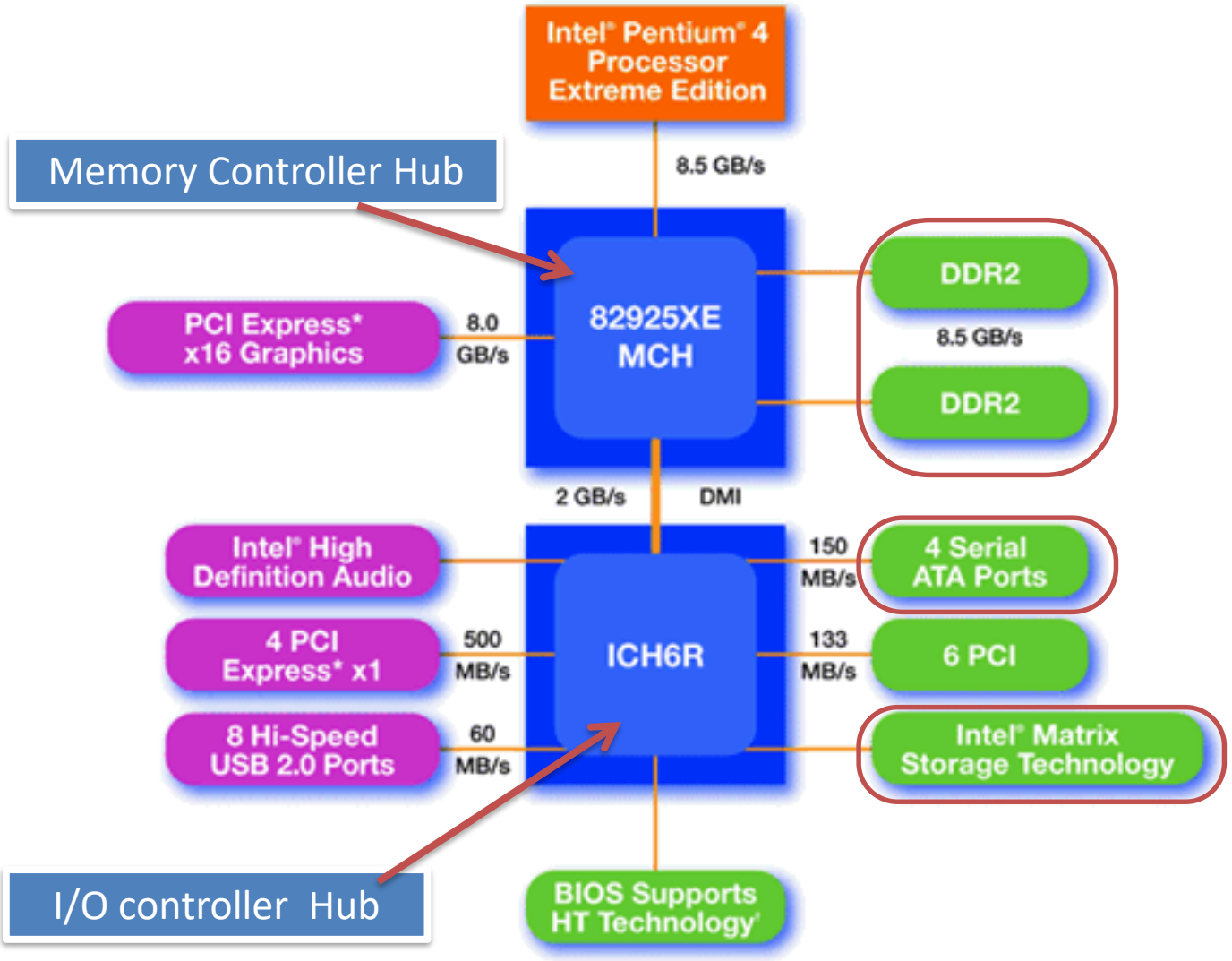
Type of memory	Bytes/access	Access time (ns)	$ per MByte	Energy
SRAM Cache (on-chip)	10	0.1	1 – 100	1 nJ
SRAM Cache (off-chip)	100	2 – 5	1 – 10	10 -100 nJ
DRAM	1.000	10 – 100	0.1	1 – 100 nJ per chip
Magnetic disk	1.000	5.000.000-20.000.000	0.001	100-1.000 mJ

Memory hierarchy

- CPU clock time: 0.3ns
- The performance gap between processors and memory has been increasing and increasing in the last decades.

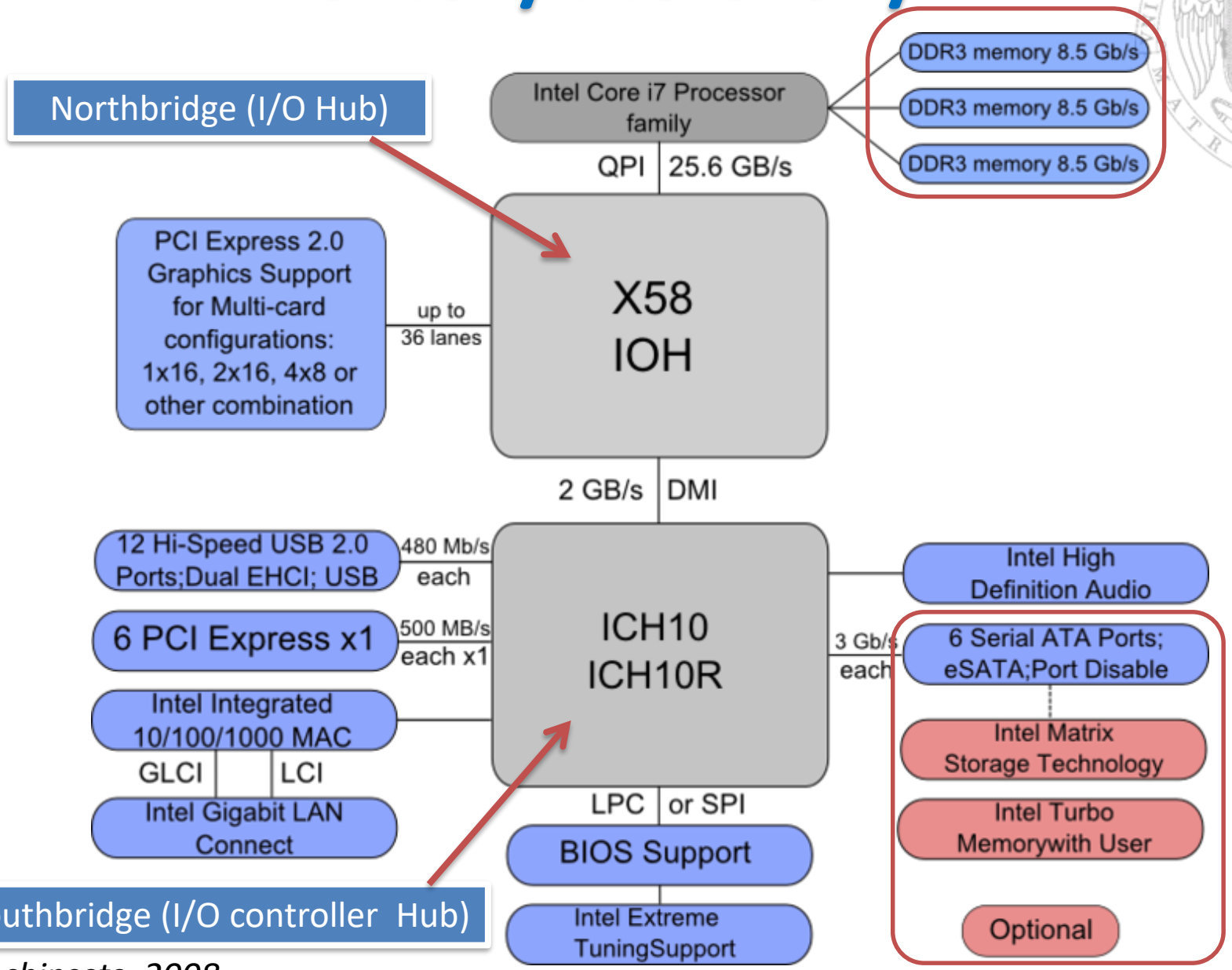


Memory hierarchy



Intel chipsets. 2004

Memory hierarchy



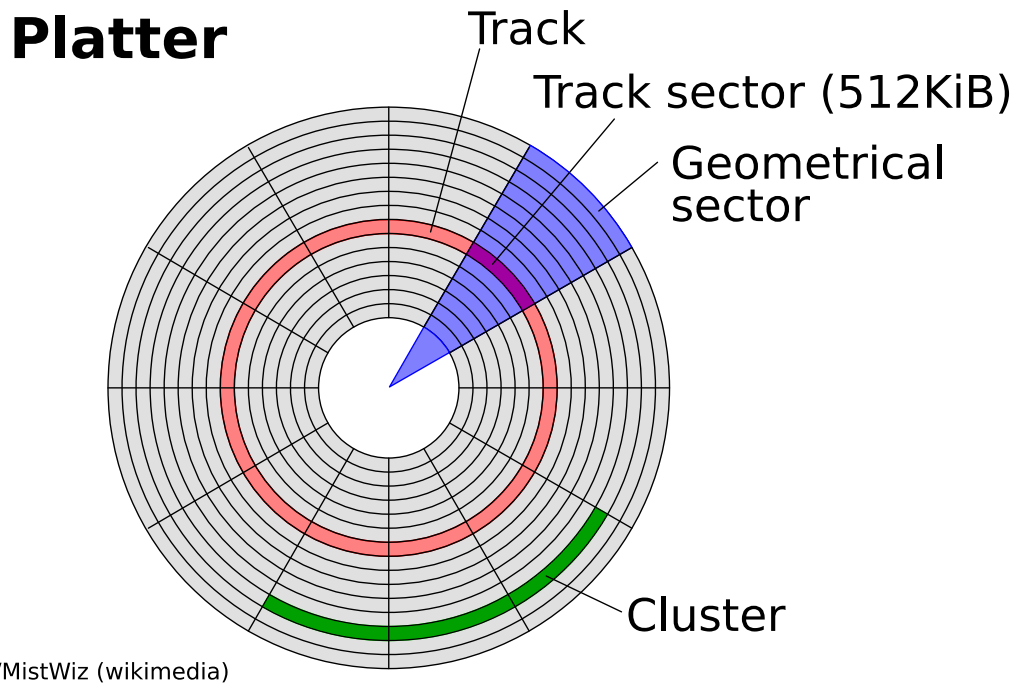
Intel chipsets. 2008

Outline

1. Introduction
2. Characteristics:
   - Access methods
   - Parameters
   - Technologies
3. Embedded SRAMs in FPGAs
   - General structure
   - Timing

Access methods

- *Direct access* (e.g. HHDs):
  - Information is organized intro piled physical disks (*platters*). Each platter is divided into *tracks* and each *track* into *sectors* of the same size (512KiB). A group of *tracks* located one on top of another constitute a *cylinder*:



By Heron2/MistWiz (wikimedia)

  - The memory access time is variable: it depends on the initial position of the *heads* (which actually read the information) with respect to the data to obtain.

Access methods

- ***Random access*** (e.g. main memory, SSDs)
  - Each position in the memory has a unique method of access physically cabled.
  - The memory access time for a given position is independent of its address or previous accesses.
- *Associative access* (e.g. Cache memory)
  - Associative memories are random access memories which words are not sorted by their address.
  - Each word has an associated *tag* (typically, the tag is somewhat associated with its address or at least, part of it).
  - A given word can be stored anywhere in the memory. In other words, the specific address of the data that we are looking for is initially unknown. Thus, in order to access to a piece of information, we need to check the tag of each word in the memory with the tag of the desired word.

Parameters

- **Access time (T_{access})**
  - In direct access memories:
    - Time elapsed to physically place the read/write head over the desired position.
  - In random access memories:
    - Time elapsed from specifying a given memory address until:
      a) the data is stored in the device, or
      b) it has been retrieved from the memory.
- **Memory cycle time**
  - Only applicable to random access memories.
    - Minimum access time elapsed between two consecutive accesses.
    - It's greater than the access time (it takes into account the period of time that the signals are in the buses).

Parameters

- **Bandwidth or transfer speed (Bi) for reading/writing from/to the memory module:**
  - In direct access memories:
    - It depends on the average access time and the device speed transfer.
  - In random access memories:
    - It can be calculated as the inverse of the memory cycle time.
- **Size of the memory (*Si*)**
  - Number of bytes that can be stored in the memory.
- **Cost per byte (*Ci*)**
  - Estimated average cost per memory byte.
    - The total cost of the memory is calculated as follows: *Si * Ci*
- **Transfer unit (*Xi*)**
  - Information unit that a given memory operates with.
    - It can range from just a byte or a word, to several blocks or even KB.

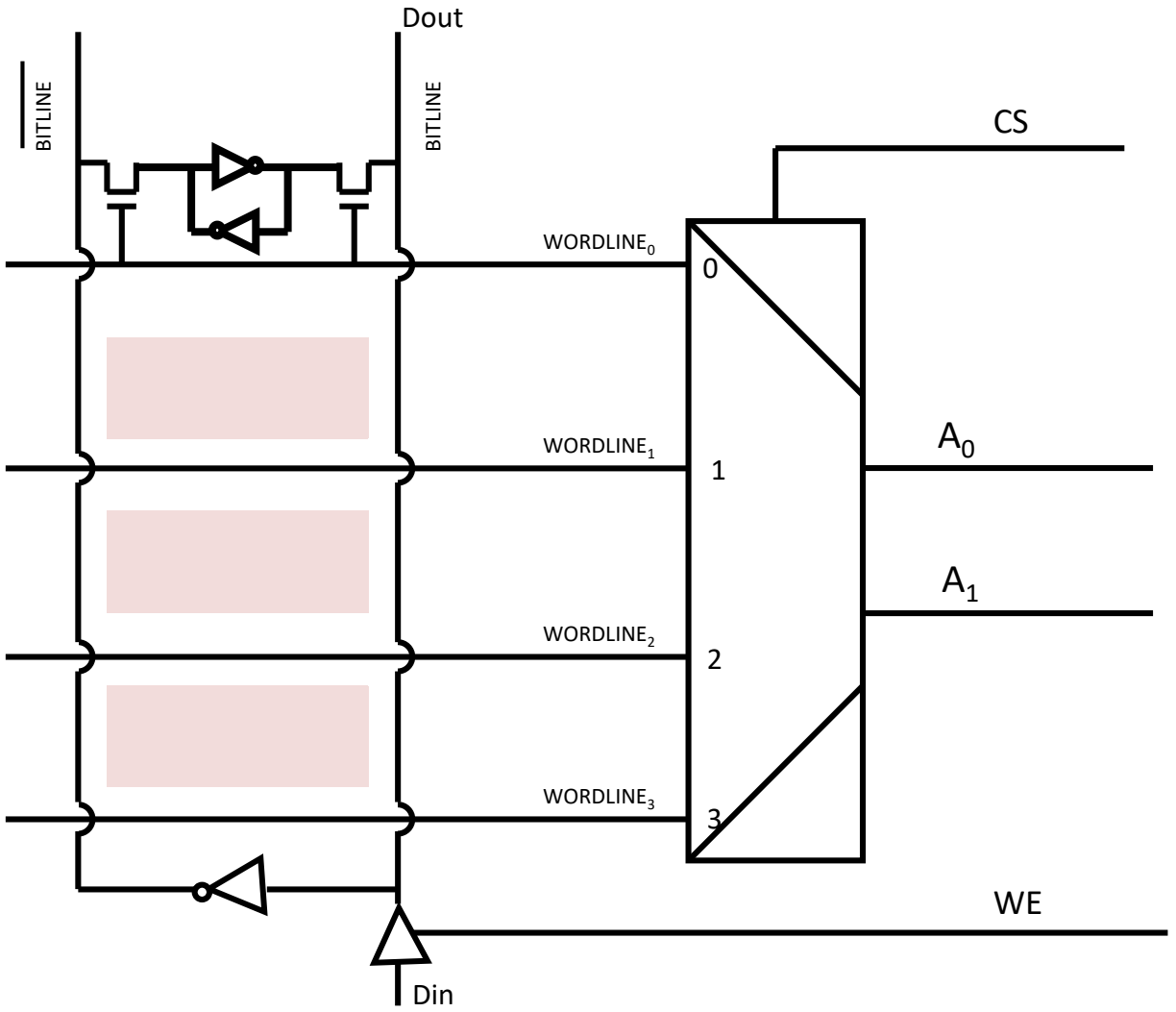
Solid state technologies

- SRAM: **S**tatic **R**AM
- DRAM: **D**ynamic **RAM**
- SDRAM: **S**ynchronous **DRAM**
- EEPROM: **E**lectrically **E**rasable **P**rogramm. ROM
  - NAND-flash and NOR-flash

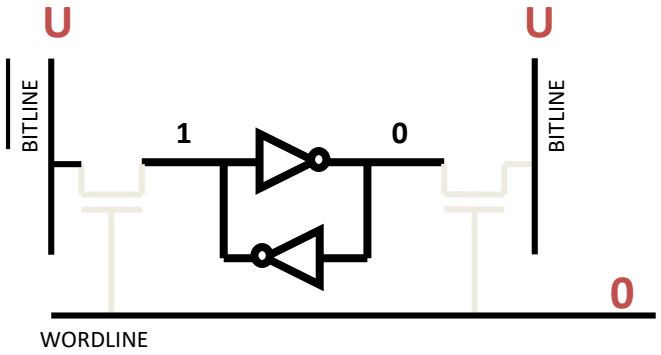
Static RAM (SRAM)

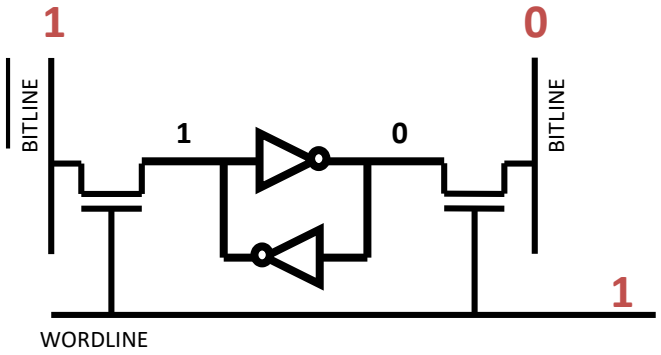
- **S**tatic **R**andom-**A**ccess **M**emory (**SRAM**)
  - A bit is stored in a flip-flop, or in a memory cell
    - The values stay stored as long as the memory is powered up.
  - Advantage → Speed.
  - Drawback → **6 transistors** are needed per cell.
    - They cannot achieve a high integration density.

Basic SRAM cell



SRAM Operation





We store or retrieve information if and only if *WORDLINE* = '1'

Dynamic RAM (DRAM)

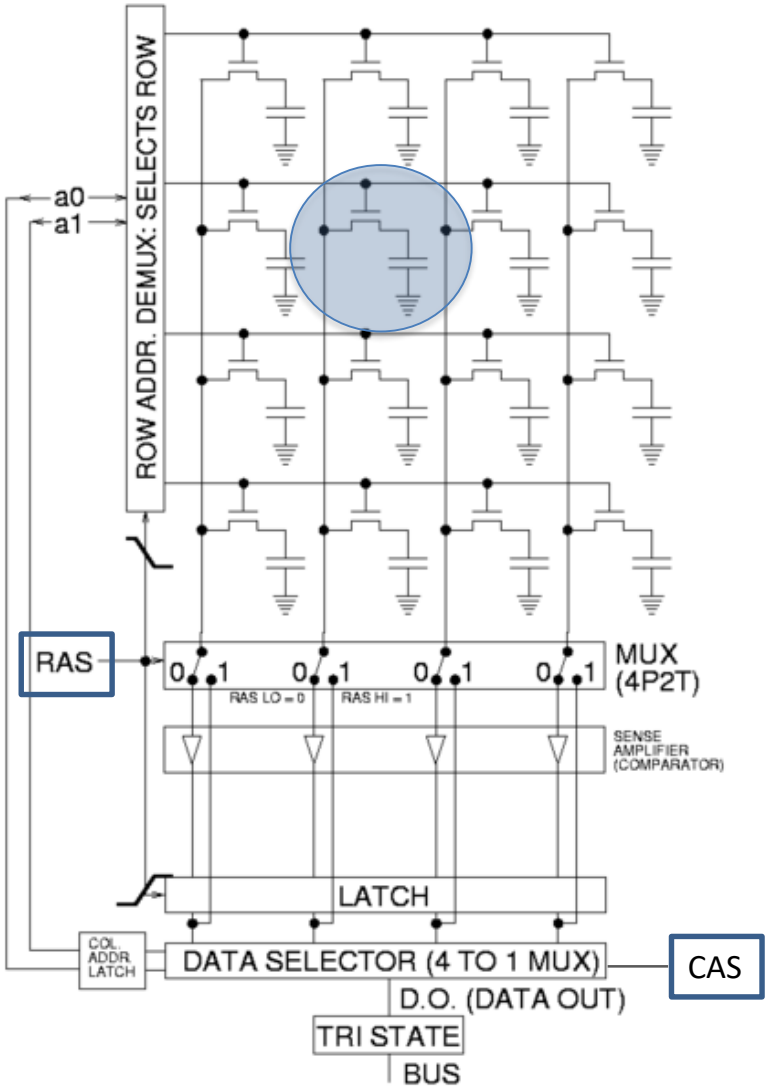
- **D**ynamic **R**andom-**A**ccess **M**emory (**DRAM**)
  - A bit is stored in a capacitor (charged – uncharged)
    - The capacitor loses charge with time (leakage), therefore it must be *refreshed* periodically. For this reason, they are known as **dynamic** memories.
  - Advantage: **Simplicity**.
    - Only 1 capacitor and **1 transistor** are needed per bit (a SRAM needs at least, 6 transistors).
    - This allows reaching high integration densities (up to several billions of bits per chip).

DRAM Basic cell (reading)

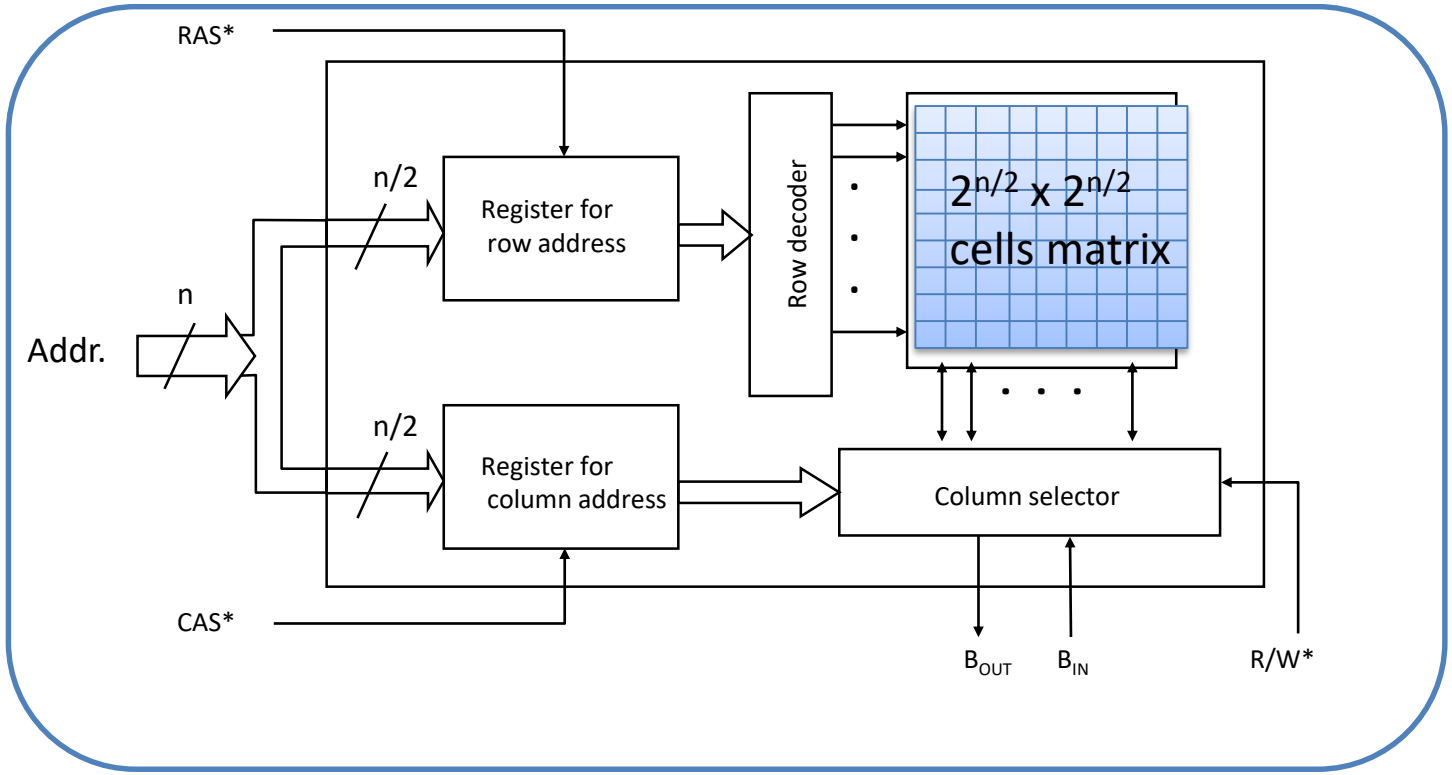
Destructive reading!!!

Reading operation RAS:

- 0 reading
- 1 writing (read data)



DRAM Organization



RAS*: Row selection line
CAS*: Column selection line

DRAM Read operations

1. The *word-lines* are pre-loaded with a voltage level between 0 and 1 (they behave as large capacitors) by using half of the address referring to the row.
   - This value is higher than what is stored in the DRAM cells (for efficiency)
2. When receiving the RAS signal, a row is selected and a connection between a line and the capacitors of the cells is established.
   - The initial charges of both capacitors are shared. The resulting charge distribution follows the following rules: $q_1 + q_2 = q'1 + q'2 \qquad \frac{q'_1}{c_1} = \frac{q'_2}{c_2}$
   - q'_1 and q'_2 will be different depending on the initial charge of the cells (q_1 and q_2).
3. The differences between q_2 and q'_2 are amplified, therefore "rounded" to 0 or to 1.
   - These amplifiers read the low-power signals stored in the memory and power them to logic levels that can be read by the logic placed outside the memory.
4. The outputs of the sense amplifiers are stored in *latches.*
5. The CAS signal and the other half of the address (referring to the column) are used to select the bit given as output.
6. Finally, the original information should be rewritten in the cells (Step 2 is destructive).

DRAM Read operations

t_{RC}: Read cycle time.
Minimum time that is elapsed between two consecutive and successful read operations.

t_{ASR} y t_{RAH} : Setup and hold times for the row sub-address.
Minimum time that the row sub-address must be stable before and after the activation of RAS*, respectively.

t_{ASC} y t_{CAH} : Setup and hold times for the column sub-address.
Minimum time that the row sub-address must be stable before and after the activation of CAS*, respectively.

t_{CAS}: Pulse width of the signal CAS* (Column Address Strobe).
Minimum time that CAS* must stay activated.

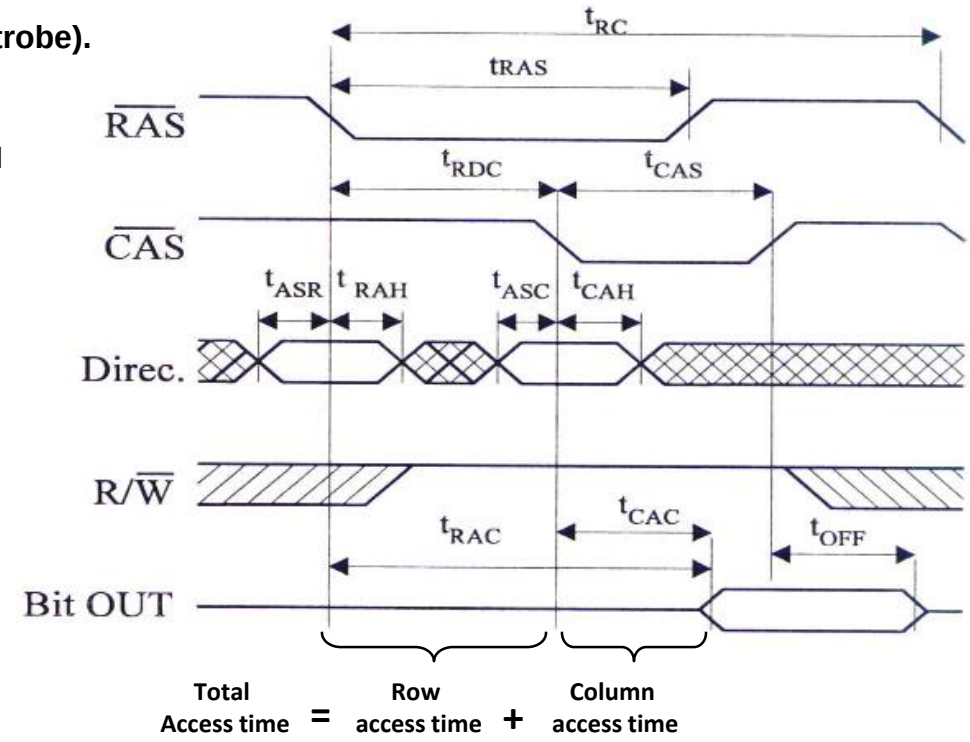
t_{RAS}: Pulse width of the signal RAS* (Row Address Strobe).
Minimum time that RAS* must stay activated.

t_{RDC}: Delay between RAS* and CAS*.
Time that must elapse between the activation of RAS* and the activation of CAS*.

t_{RAC}: Access time from the activation of RAS*.
Maximum amount of time elapsed between the activation of RAS* and the appearance of stable data at the outputs.

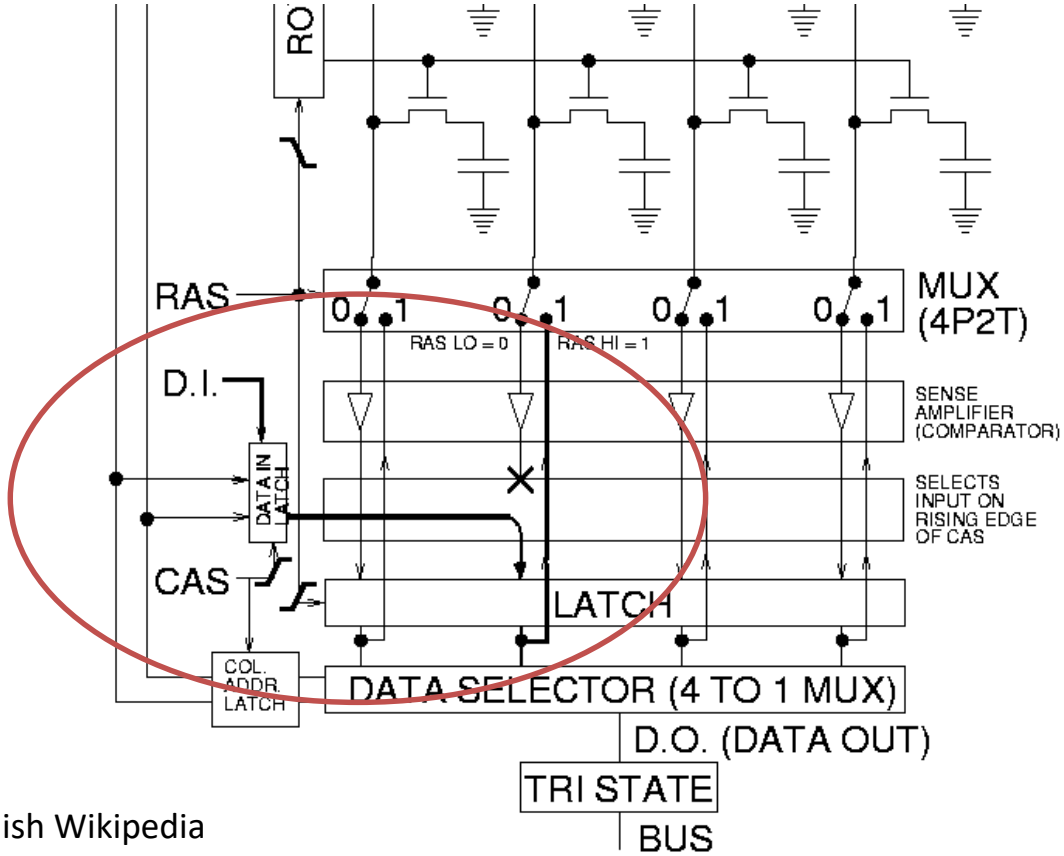
t_{CAC}: Access time from the activation of CAS*.
Maximum amount of time elapsed between the activation of CAS* and the appearance of stable data at the outputs.

t_{OFF}: Offset time.
Time the output data are displayed out after deactivating RAS* and CAS*.



DRAM Write operations

- The sense amplifiers are forced to 0 or to 1 in order to charge/uncharge the capacitors of the cell that has been selected.



By Glogger at English Wikipedia

DRAM Write operations

t_{RC}: Write cycle time.
Minimum time that is elapsed between two consecutive and successful write operations.

t_{ASR} y t_{RAH} : Setup and hold times for the row sub-address.
Minimum time that the row sub-address must be stable before and after the activation of RAS*, respectively.

t_{ASC} y t_{CAH} : Setup and hold times for the column sub-address.
Minimum time that the row sub-address must be stable before and after the activation of CAS*, respectively.

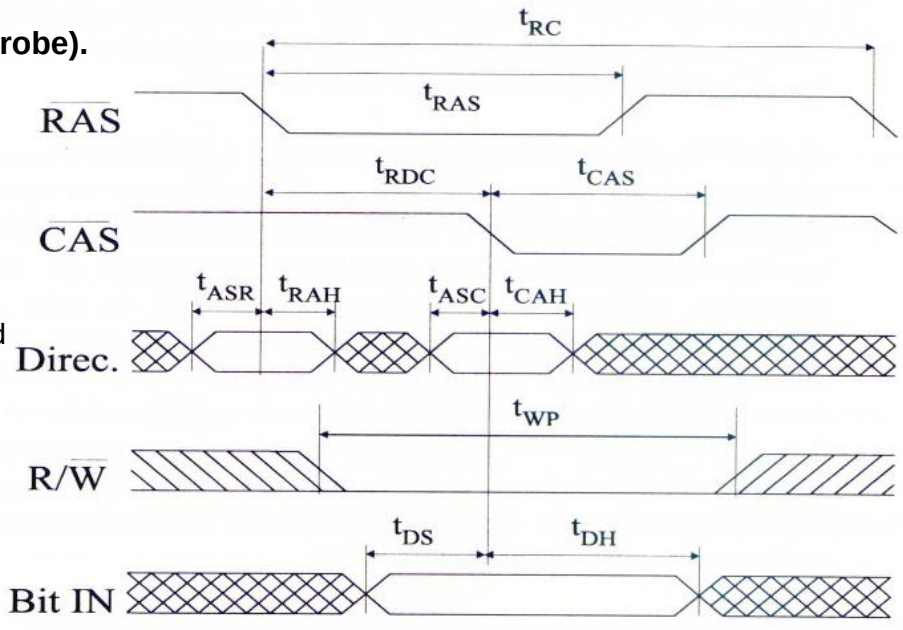
t_{CAS}: Pulse width of the signal CAS* (Column Address Strobe).
Minimum time that CAS* must stay activated

t_{RAS}: Pulse width of the signal RAS* (Row Address Strobe).
Minimum time that RAS* must stay activated

t_{RDC}: Delay between RAS* and CAS*.
Time that must elapse between the activation of RAS* and the activation of CAS*.

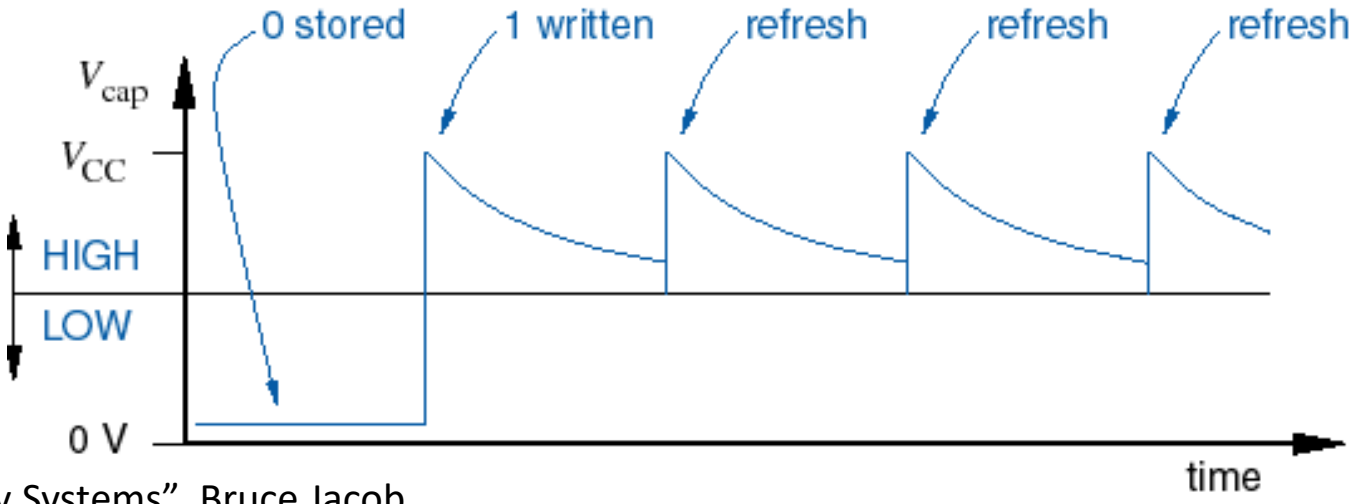
t_{WP}: Pulse width of the Read/Write signal
Time the R/W* signal must be set to '0'.

t_{DS} y t_{DH} : Setup and hold times for the input data
Minimum time that the input data must be stable before and after the activation of CAS*



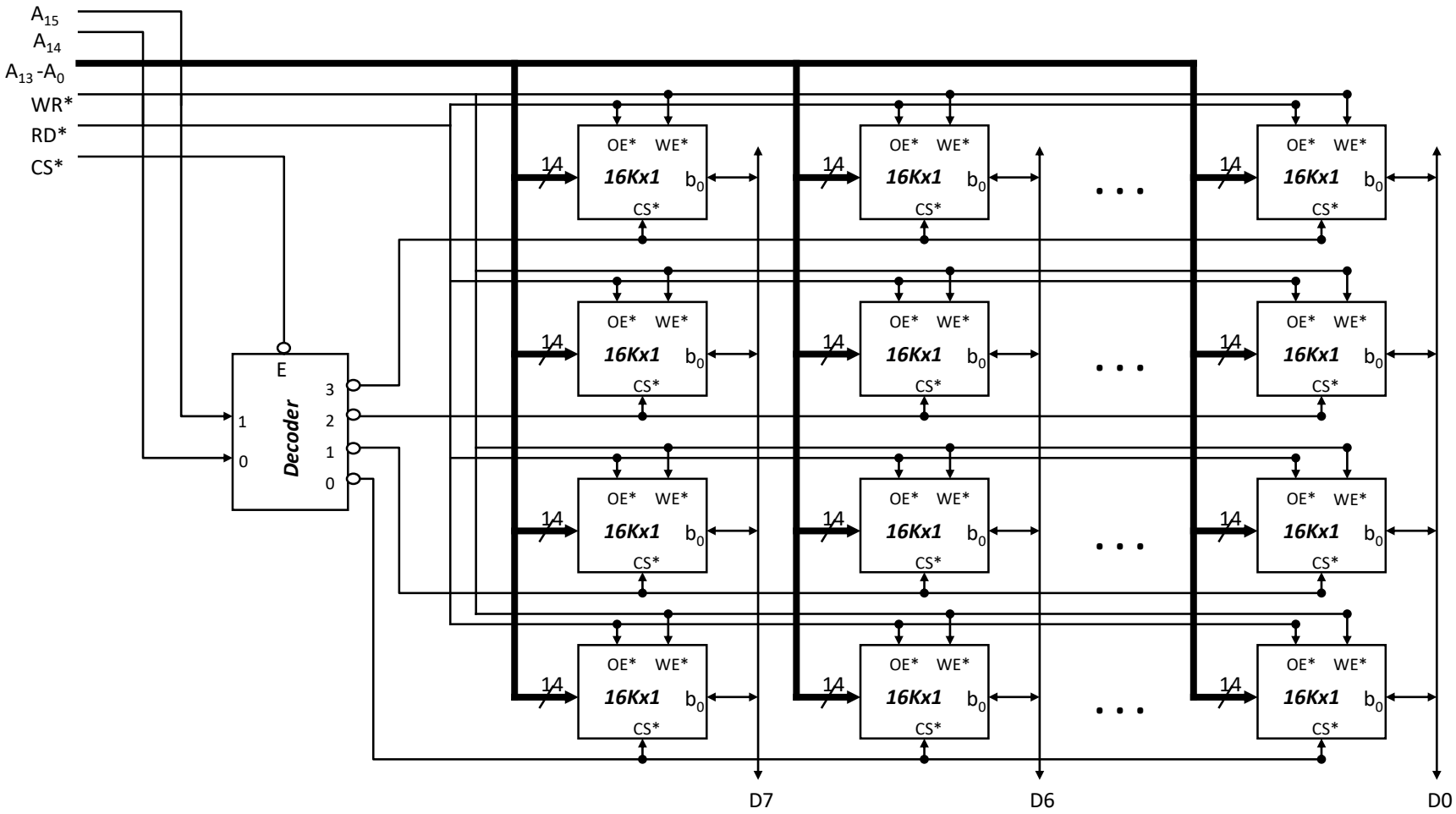
DRAM Refresh

- Refresh modes
  - *Burst refresh*: a number of refresh cycles are performed one after another until all the rows have been refreshed. After that, the memory can be used normally.
  - *Distributed refresh*: refresh cycles are performed at regular intervals, interspersed with memory accesses.
  - Both modes need a counter that indicates which is the next row to be refreshed (it is implemented inside the chip).



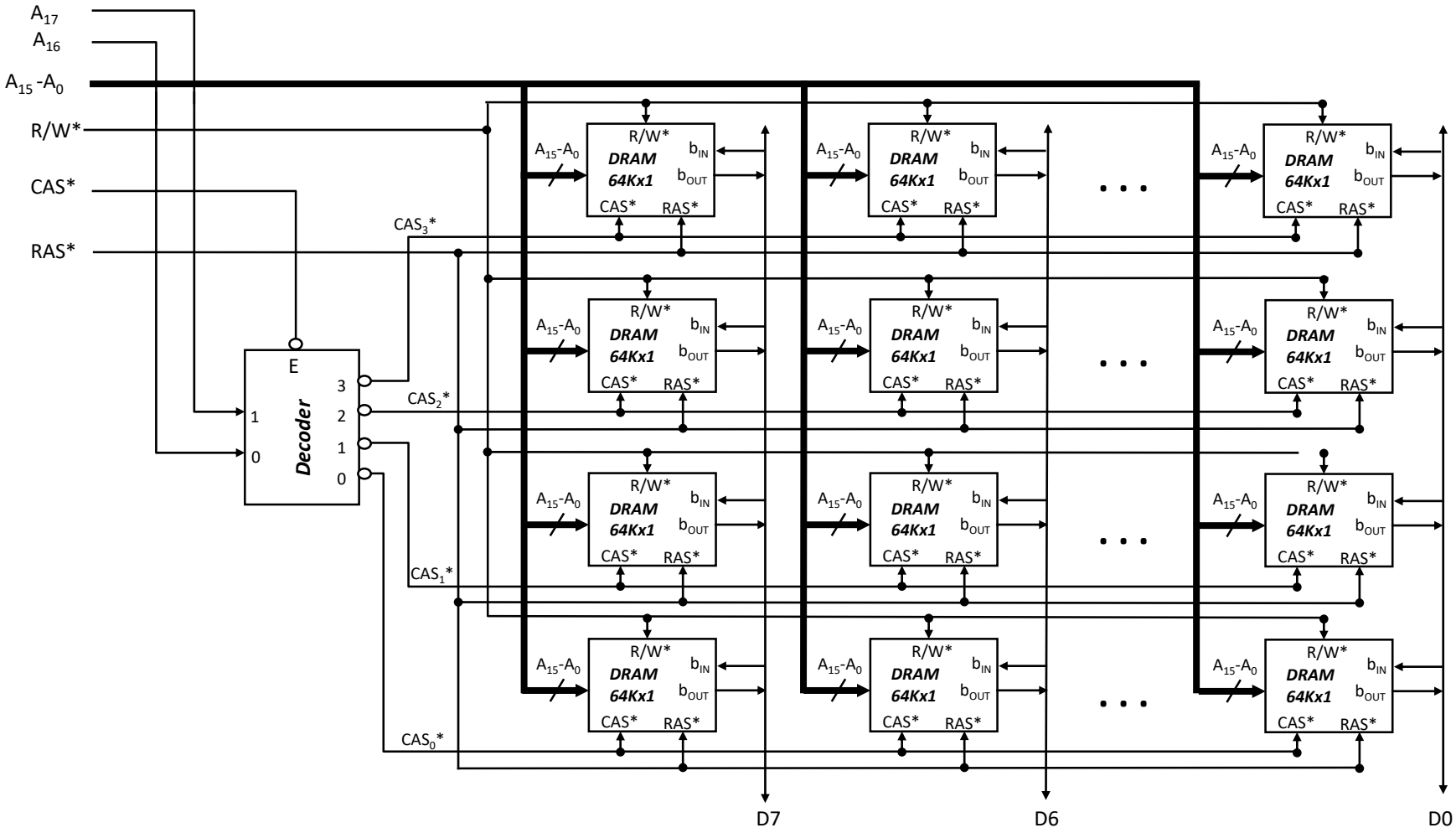
DRAM Example of design I

- Design a 64Kx8 bits memory from 16Kx1 bits chips



DRAM Example of design II

- Design a 256Kx8 bits memory from 64Kx1 bits chips



Synchronous DRAM (SDRAM)

- **S**ynchronous **D**ynamic **R**andom **A**ccess Memory (SDRAM)
  - In a nutshell, it's a synchronous DRAM.
    - Typically, DRAMs are asynchronous to minimize the access time.
  - They are usually synchronized with the system bus.
  - The memory is synchronized with the rising edge of the clock signal. Prior to returning a value, this rising edge of the clock is used to trigger the operation of the FSM of the memory. Thus, it can respond to complex access patterns.
  - SDRAM memories are usually divided into 2 or 4 independent banks.

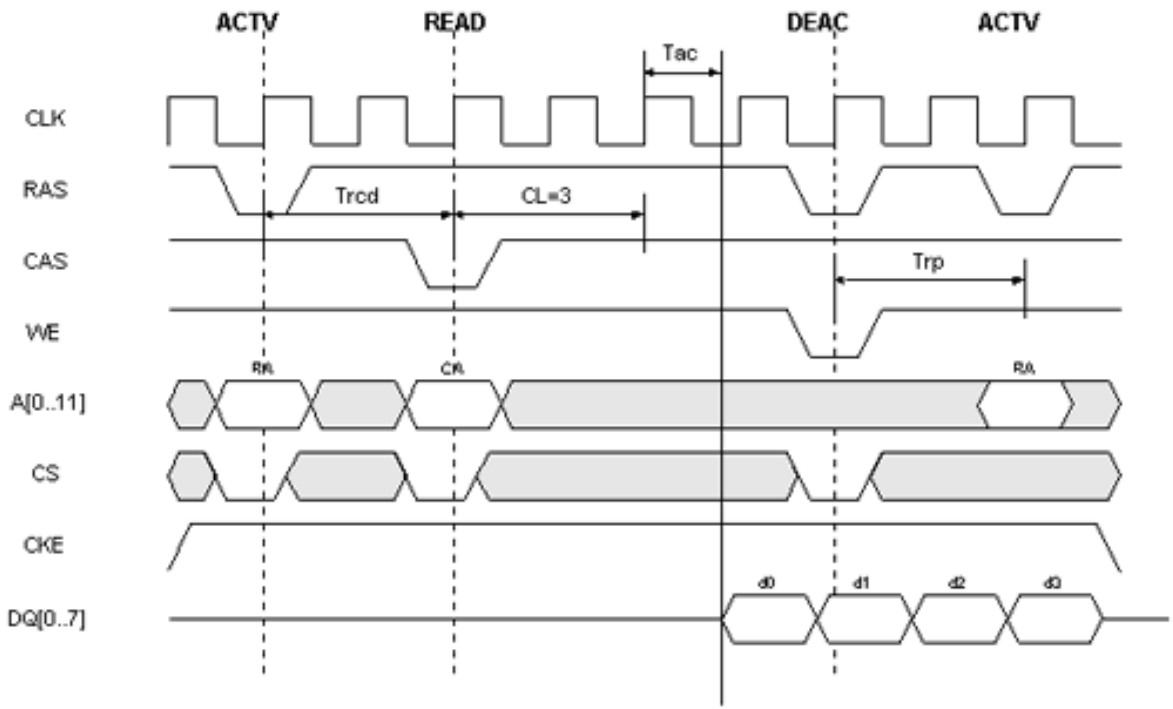
SDRAM Control signals

- Besides the clock, there are 6 control signals:
  - **CKE** (Clock Enable): When CKE = '1', this signal enables the clock. Otherwise, it disables the clock.
  - **CS*** (Chip Select): When CS* = '1', the remainder of the inputs are ignored (except CKE); otherwise, the inputs below are taken into account.
  - **DQM** (Data Mask): When DQM = '1', the data transfers with the outside world are disabled. In other words, nothing is written or read in/from the memory unless DQM is set to '1' for, at least, two clock cycles before the memory operation request.
    - A DQM line exists for each 8 bits in each x16 or DIMM chip.
  - **RAS*** Row Address Strobe.
  - **CAS*** Column Address Strobe.
  - **WE*** Write enable.

SDRAM Reading

i) Bank activation
ii) Row address
iii) Column address
iv) Data ready

All this in ***n*** clock cycles



SDRAM burst read timing

Outline

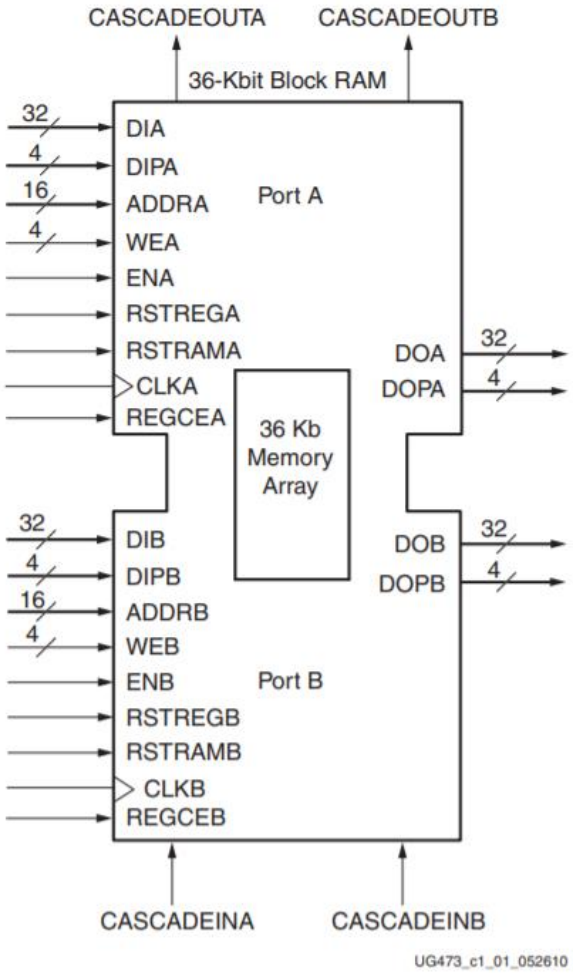
1. Introduction
2. Characteristics:
   - Access methods
   - Parameters
   - Technologies
3. Embedded BRAMs in FPGAs
   - General structure
   - Timing

BRAMs

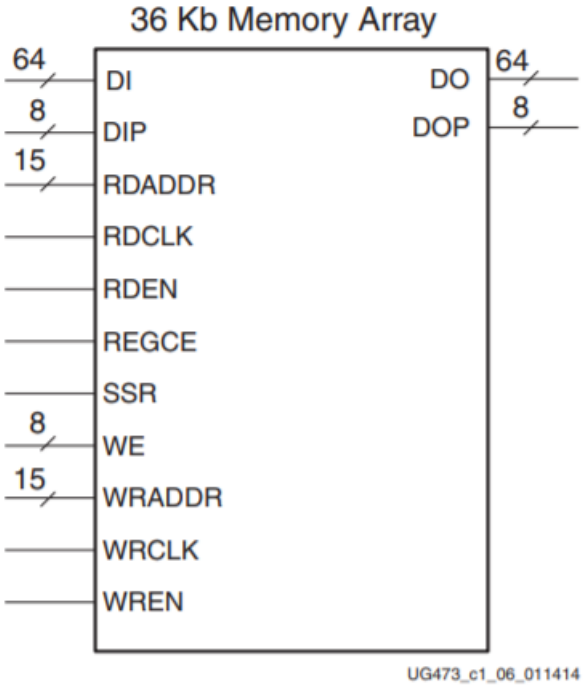
- Many FPGA families also have memory blocks. They are named Block RAMs (BRAMs).
  - Each block RAM can store up to 36Kb of data.
  - They can work as two independent 18Kb blocks, or a single 36Kb block RAM.
  - They feature different organizations:
    - From 64Kx1 bit to 512x72 bits.
    - The 7-series 7A35T FPGA (existing in the Basys 3 board) contains 50 blocks with 36Kb each (32Kb of data + 4Kb of "parity").
  - Types:
    - Simple dual port.
    - True dual port.
    - If two W operations are made on the same memory position simultaneously ("collision"), the written data is not deterministic.
  - Read and write operations are carried out synchronously.
  - They also feature parity bits: for error control purposes, but they can be used separately.

https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

BRAMs



True dual port (ports A and B for R/W)



Simple dual port (one port for reading, other one for writing)

BRAMs

Table 1-3: **True Dual-Port Functions and Descriptions**

Port Function	Description
DI[A | B]	Data input bus.
DIP[A | B](1)	Data input parity bus. Can be used for additional data inputs.
ADDR[A | B]	Address bus.
WE[A | B]	Byte-wide write enable.
EN[A | B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
RSTREG[A | B]	Synchronous Set/Reset the output registers (DO_REG = 1). The RSTREG_PRIORITY attribute determines the priority over REGCE.
RSTRAM[A | B]	Synchronous Set/Reset the output data latches.
CLK[A | B]	Clock input.
DO[A | B]	Data output bus.
DOP[A | B](1)	Data output parity bus. Can be used for additional data outputs.
REGCE[A | B]	Output Register clock enable.
CASCADEIN[A | B]	Cascade input for 64K x 1 mode.
CASCADEOUT[A | B]	Cascade output for 64K x 1 mode.

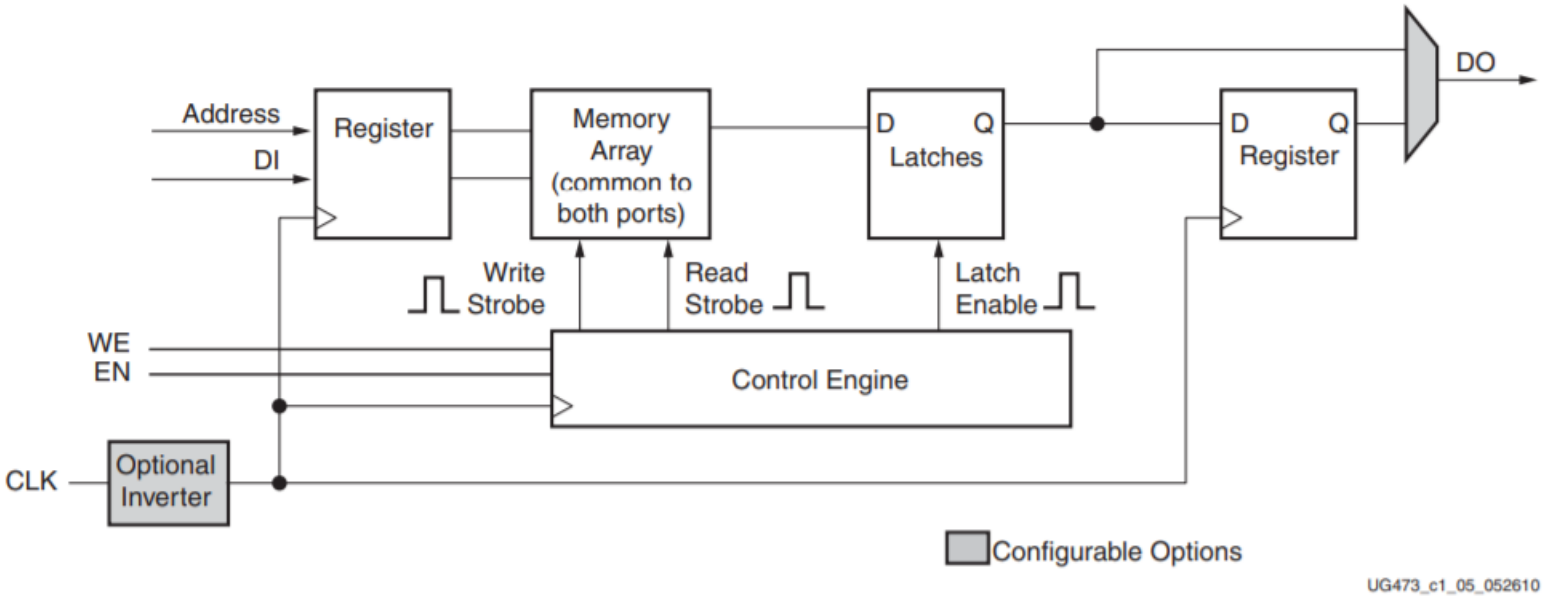
Notes:

1. The Data-In Buses - DIADI, DIPADIP, DIBDI, and DIPBDIP section has more information on data parity pins.
2. Block RAM primitive port names can be different from the port function names.

https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

BRAMs

- BRAMs in 7-series Xilinx FPGAs have optional (embedded) output registers, which store the data without any need to store it anywhere else (i.e., FFs in CLBs).
  - This makes possible to pipeline these BRAMs into 2-stage devices.



BRAMs – Organization

- By default, BRAMs have 32-bit words. However, they can be programmed to feature one of the following organizations:
  - 32Kx1
  - 16Kx2
  - 8Kx4
  - 4Kx9
  - 2Kx18
  - 1Kx36
  - 512x72

4 Kb of parity

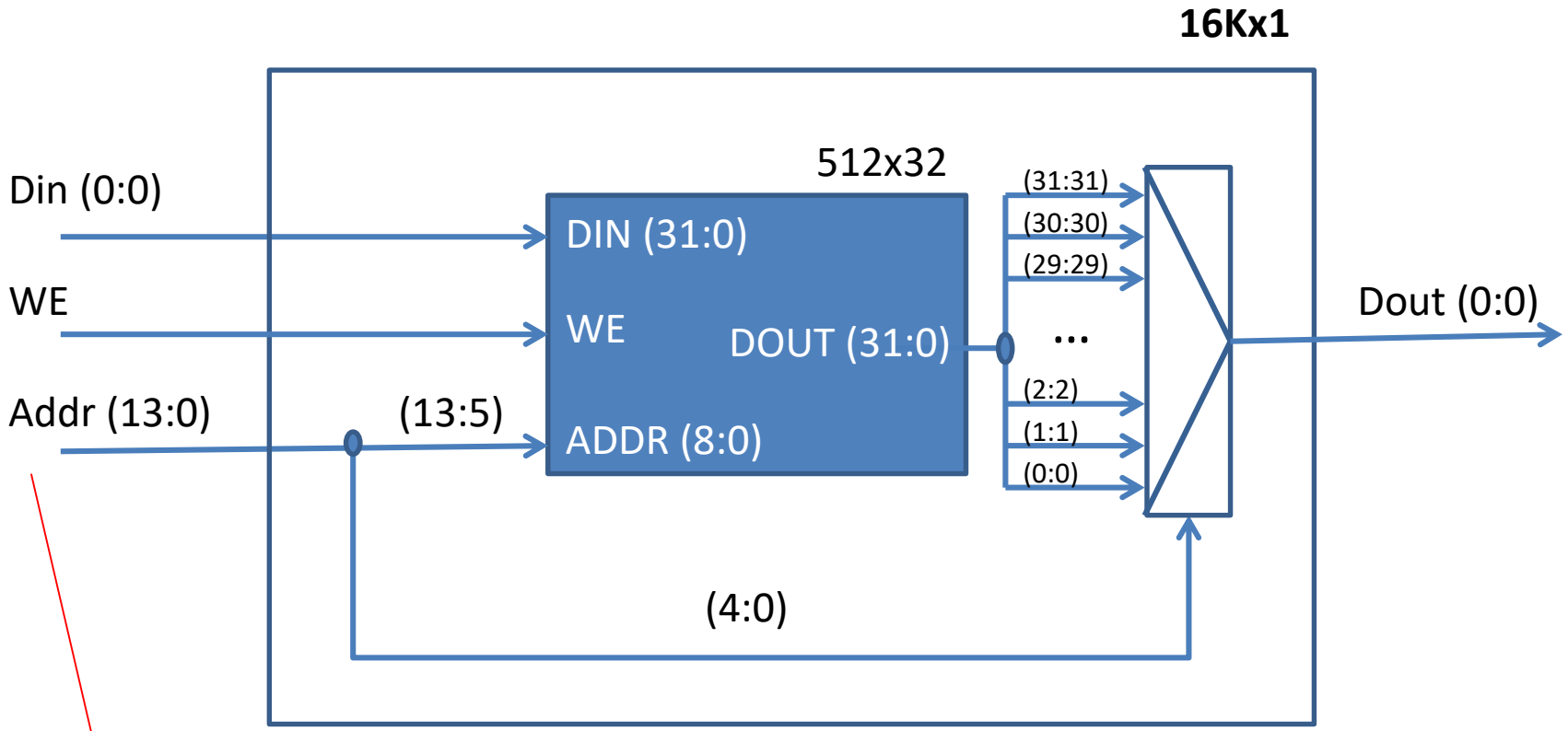
This one is possible if the BRAM is configured as dual port

BRAMs – Organization

- Simplified original organization (a submodule of a BRAM – 18Kb, reduced to 16Kb without parity):

BRAMs – Organization

- 16Kx1 organization:



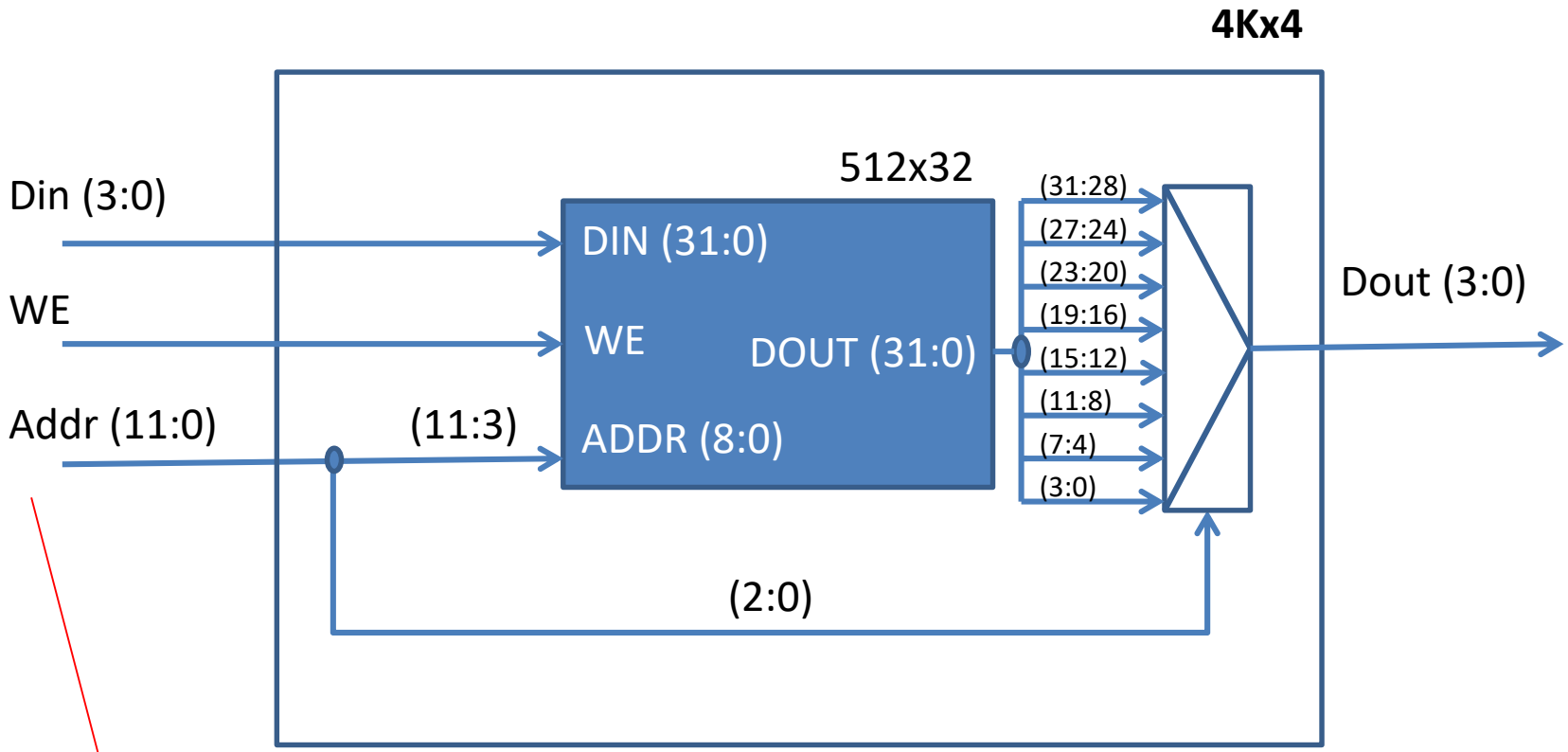
Address port length: 16K positions = 2^5 x 2^9 positions = 2^{14} positions

BRAMs – Organization

- Exercises:
  1. Which is the HW needed to carry out read operations in a 4Kx4 organization?
  2. Which is the HW needed to carry out read operations in 1Kx16?
  3. Which is the HW needed to carry out read operations in 256x64?

BRAMs – Organization (Ex. 1)

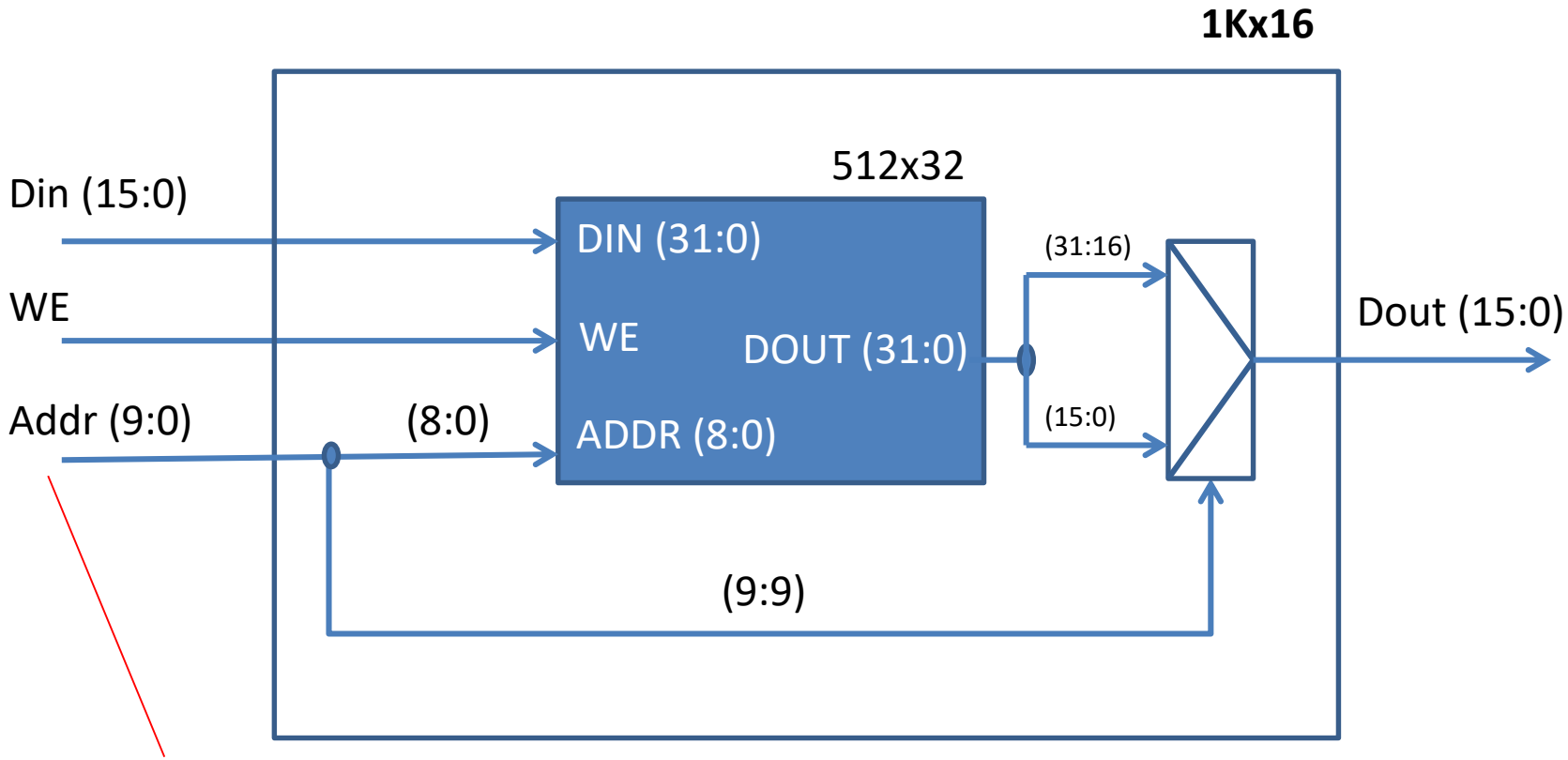
- Read for 4Kx4 configuration:



Length for address port : 4K positions = 2^3 x 2^9 positions = 2^{12} positions

BRAMs – Organization (Ex. 2)

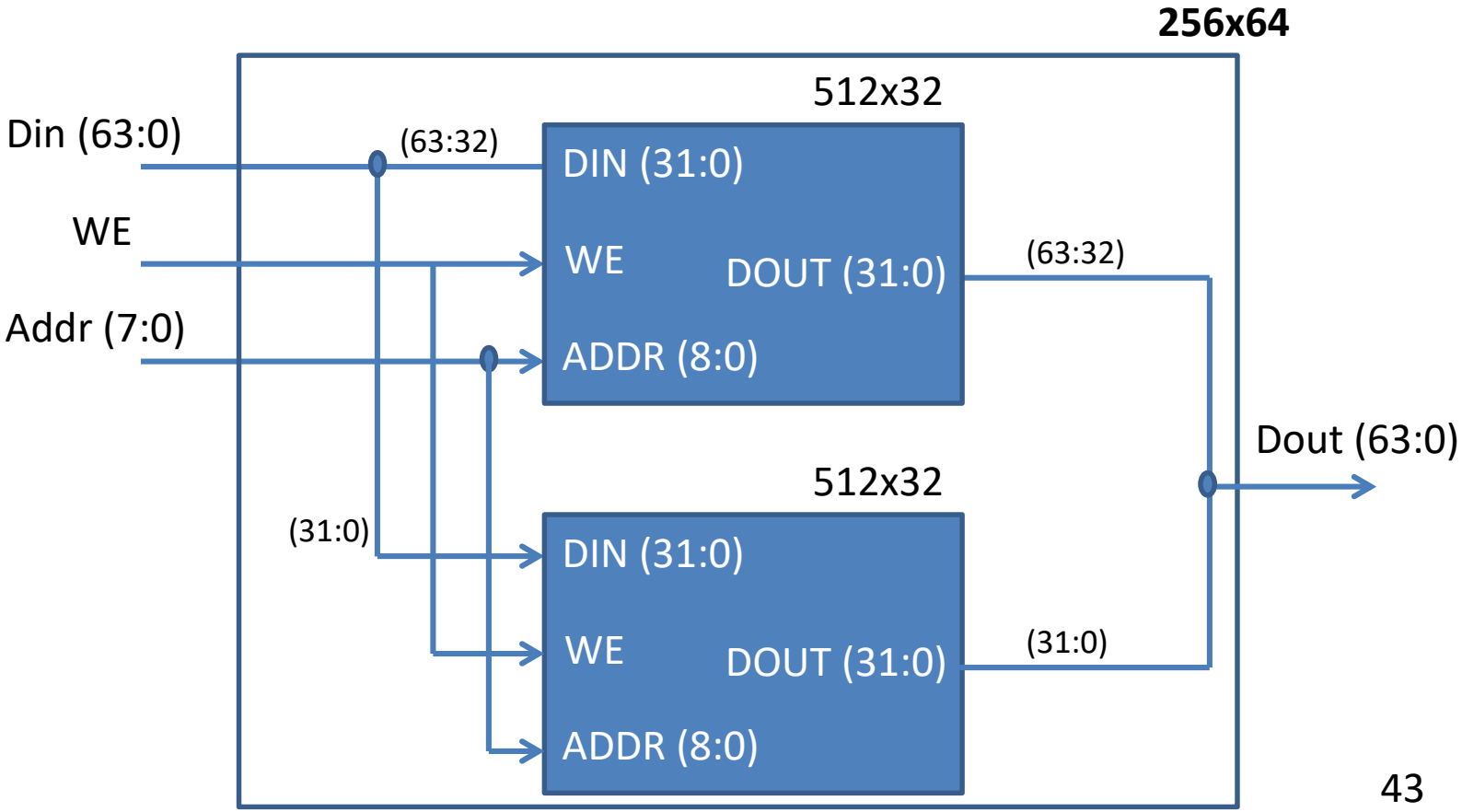
- Read for 1Kx16 configuration:



Length for address port: 1K positions = 2^1 x 2^9 positions = 2^{10} positions

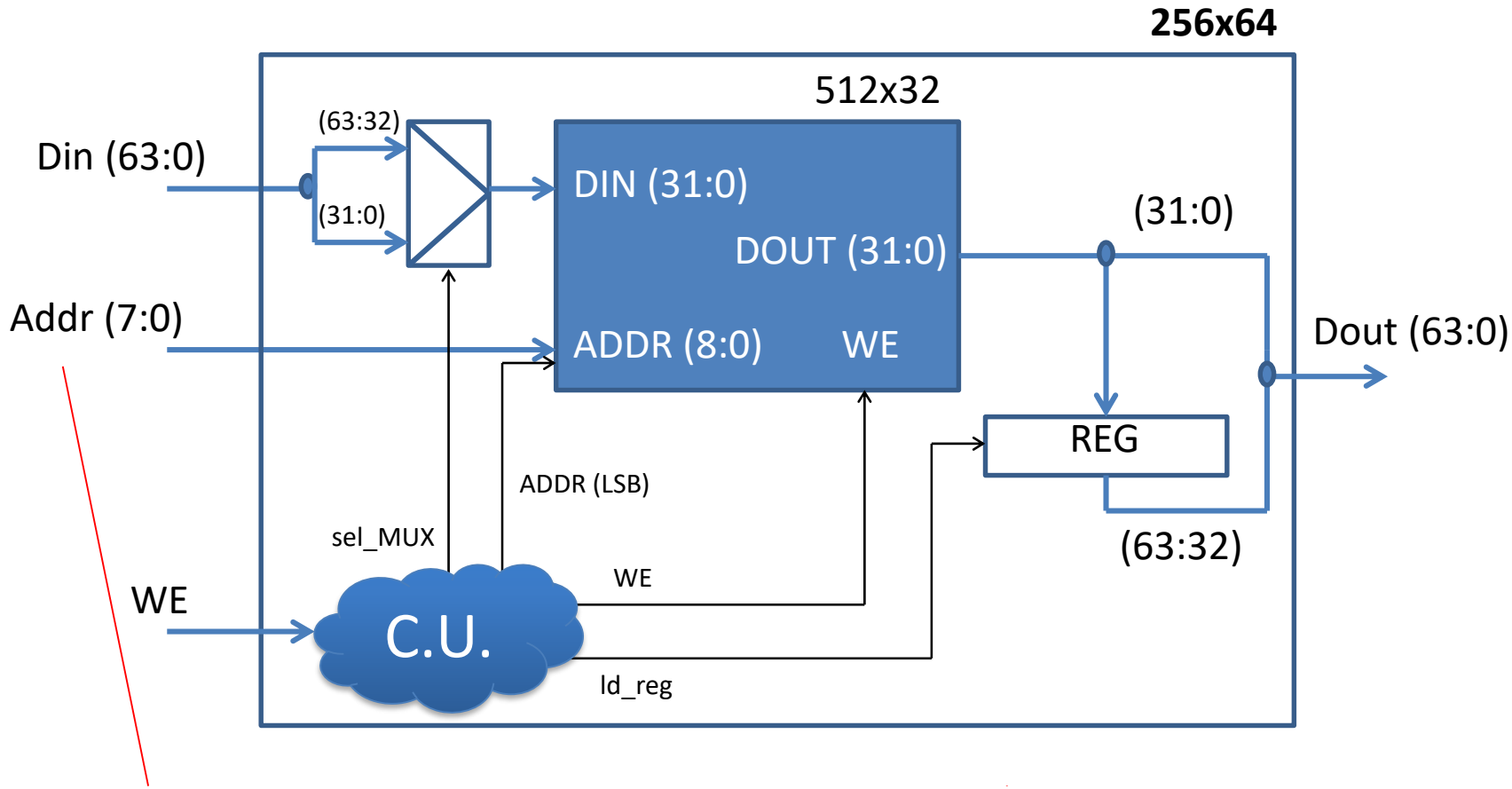
BRAMs – Organization (Ex. 3)

- Read for 256x64 configuration:
  - 2 BRAMs are needed in order to carry out read operations in 1 clock cycle



BRAMs – Organization (Ex. 3)

- Read for 256x64 configuration using one BRAM

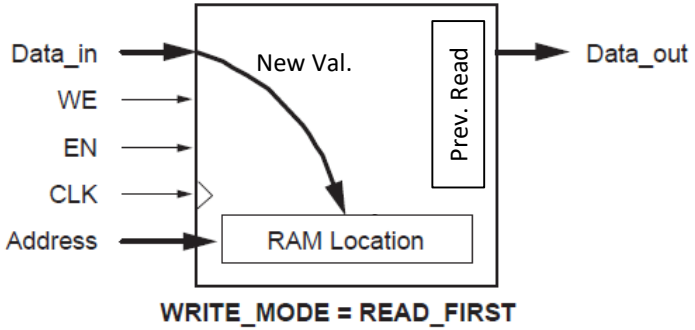


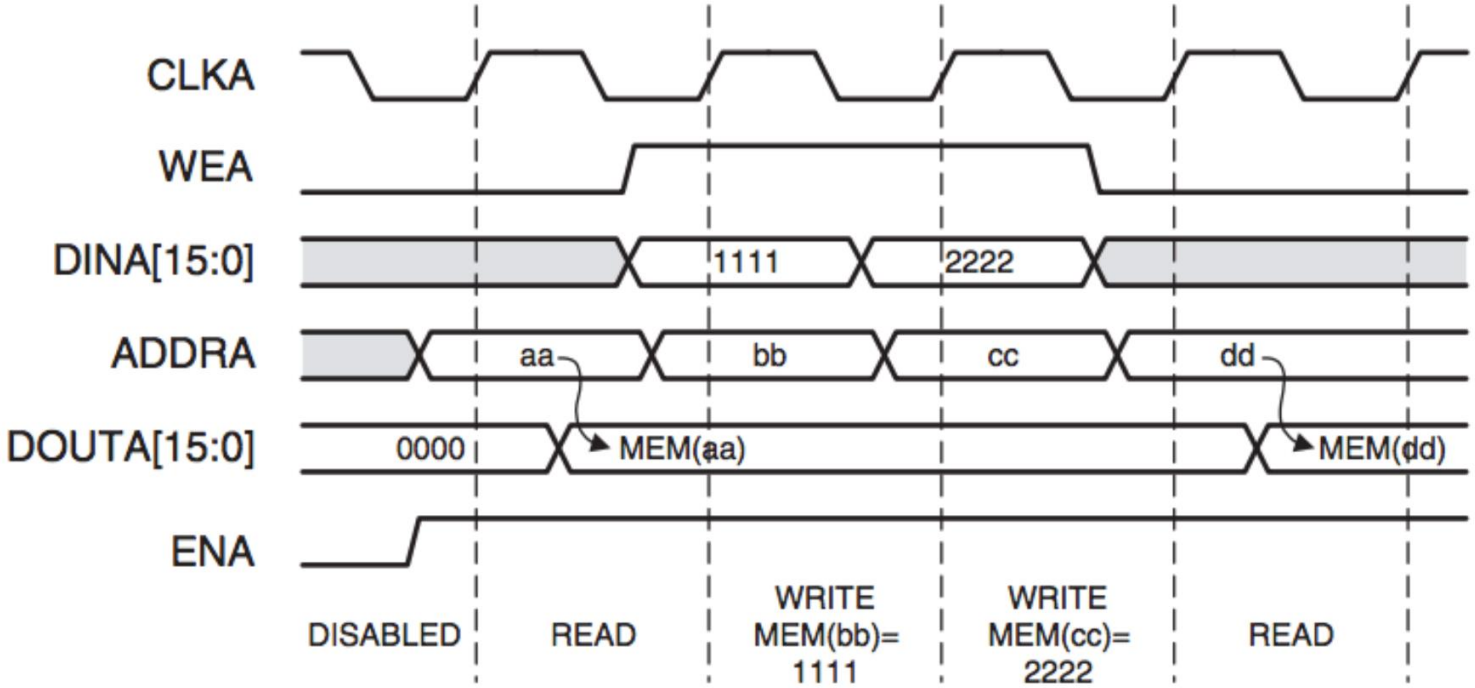
Address port width = 8 -> 256 positions

We need 2 clock cycles (CU)

BRAMs – Timing (*NO_CHANGE*)

- The output port is **not updated** when a write operation occurs





BRAMs – Timing (*NO_CHANGE*)

```
entity blockram is port (
    clk, we, en : in  STD_LOGIC;
    addr        : in  STD_LOGIC_VECTOR (8 downto 0);
    di          : in  STD_LOGIC_VECTOR (31 downto 0);
    do          : out STD_LOGIC_VECTOR (31 downto 0) );
end blockram;

architecture Behavioral of blockram is
    type ram_type is array (511 downto 0) of std_logic_vector (31 downto 0);
    signal ram : ram_type;
begin
    process ( clk )
    begin
        if rising_edge(clk) then
            if en = '1' then
                if we = '1' then
                    ram(to_integer(unsigned(addr))) <= di;
                else
                    do <= ram(to_integer(unsigned(addr)));
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

BRAMs – Timing (*NO_CHANGE*)

- Inferring the NO_CHANGE mode:
  - Messages visible in the Synthesis log:

```
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
Hierarchical RTL Component report
Module blockram
Detailed RTL Component Info :
+---Registers :
                               32 Bit     Registers := 1
+---RAMs :
                              16K Bit         RAMs := 1
---------------------------------------------------------------------------------

.
.

Block RAM: Preliminary Mapping  Report (see note below)
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|Module Name | RTL Object | PORT A (Depth x Width)  | W | R | PORT B (Depth x Width)  | W | R | Ports driving FF | RAMB18 | RAMB36 |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|blockram    | ram_reg    | 512 x 32(NO_CHANGE)     | W | R |                         |   |   | Port A           | 1      | 0      |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
```

  - Messages visible in the Implementation "report_utilization" log:

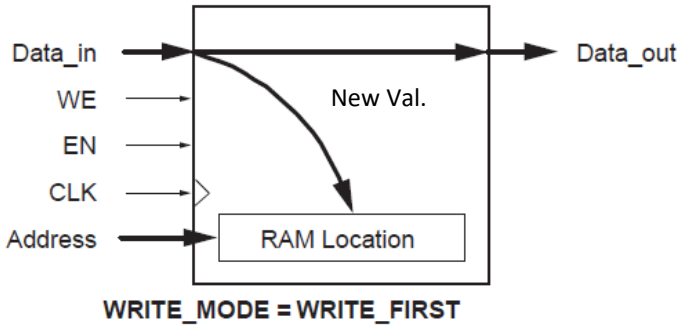
```
3. Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |        50 |  1.00 |
|   RAMB36/FIFO*    |    0 |     0 |        50 |  0.00 |
|   RAMB18          |    1 |     0 |       100 |  1.00 |
|     RAMB18E1 only |    1 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

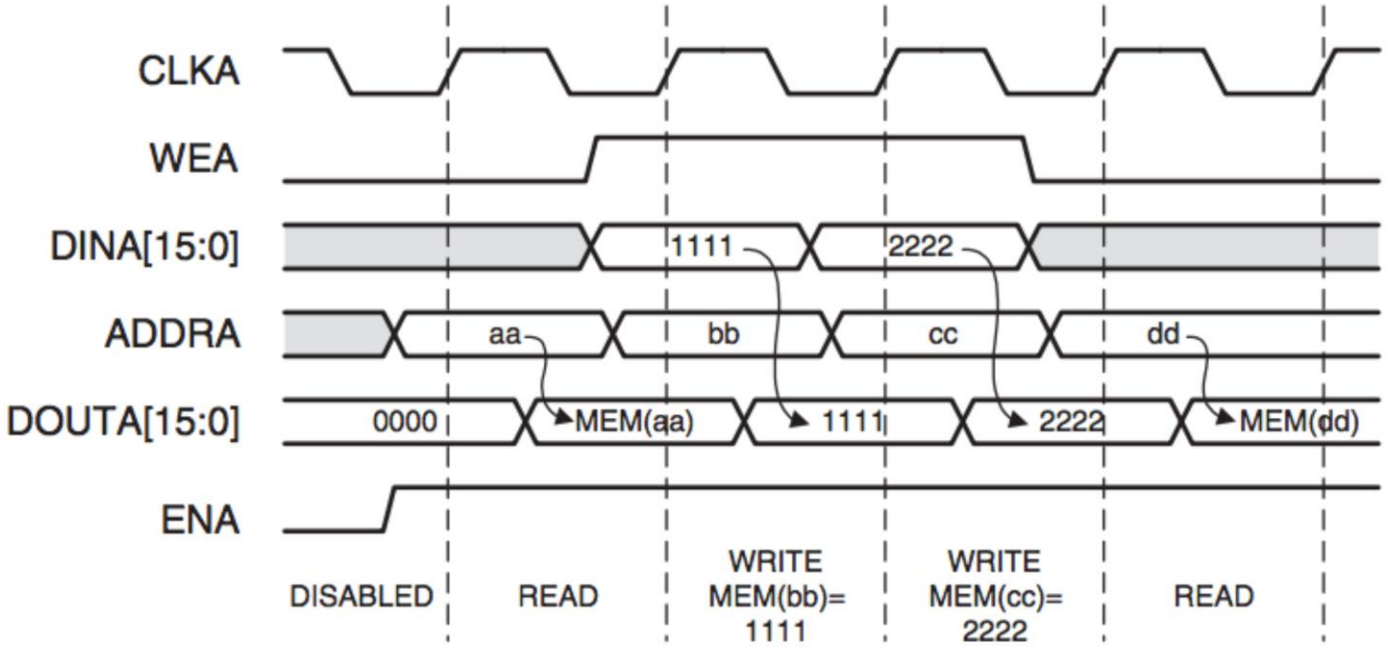
This (RAMB18E1) is a Xilinx *primitive*. These are Xilinx components that are native to the architecture you are targeting. If you instantiate a primitive in your design, after the translation process (*ngdbuild*) you will end up with the exact same component in the back end. We will not use them in this course.

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug953-vivado-7series-libraries.pdf

BRAMs – Timing (*WRITE_FIRST*)

- By default, Xilinx BRAM memories work in *WRITE_FIRST* mode
  - The input data are simultaneously written into the memory and driven on the data output. This mode offers the flexibility of using the data output bus during a write operation on the same port





BRAMs – Timing (*WRITE_FIRST*)

```
entity blockram is port (
    clk, we, en : in  STD_LOGIC;
    addr        : in  STD_LOGIC_VECTOR (8 downto 0);
    di          : in  STD_LOGIC_VECTOR (31 downto 0);
    do          : out STD_LOGIC_VECTOR (31 downto 0));
end blockram2;

architecture Behavioral of blockram is
    type ram_type is array (511 downto 0) of std_logic_vector (31 downto 0);
    signal ram : ram_type;
begin
    process( clk )
    begin
        if rising_edge(clk) then
            if en = '1' then
                if we = '1' then
                    ram(to_integer(unsigned(addr))) <= di;
                    do <= di;
                else
                    do <= ram(to_integer(unsigned(addr)));
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

BRAMs – Timing (*WRITE_FIRST*)

- Inferring the WRITE_FIRST mode:
  - Messages visible in the Synthesis log:

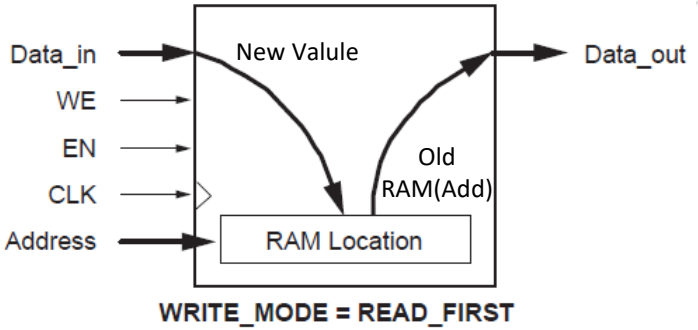
```
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
Hierarchical RTL Component report
Module blockram
Detailed RTL Component Info :
+---Registers :
                               32 Bit    Registers := 1
+---RAMs :
                              16K Bit         RAMs := 1
---------------------------------------------------------------------------------

.
.

Block RAM: Preliminary Mapping  Report (see note below)
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|Module Name | RTL Object | PORT A (Depth x Width)  | W | R | PORT B (Depth x Width)  | W | R | Ports driving FF | RAMB18 | RAMB36 |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|blockram    | ram_reg    | 512 x 32(WRITE_FIRST)   | W | R |                         |   |   | Port A           | 1      | 0      |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
```

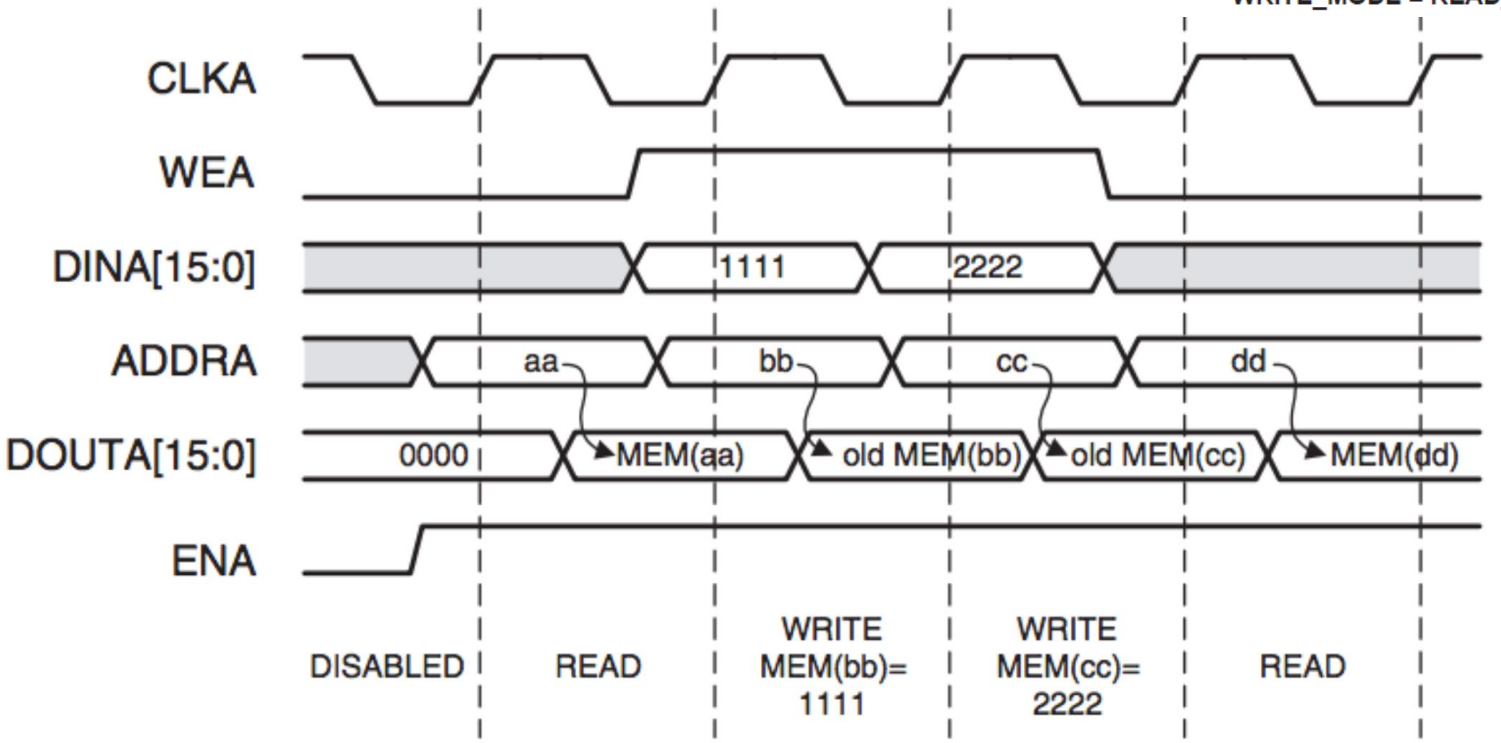
  - Messages visible in the Implementation “report_utilization” log:

```
3. Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |        50 |  1.00 |
|   RAMB36/FIFO*    |    0 |     0 |        50 |  0.00 |
|   RAMB18          |    1 |     0 |       100 |  1.00 |
|     RAMB18E1 only |    1 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

BRAMs – Timing (*READ_FIRST*)

- Another operation mode is *READ_FIRST*
  - Data previously stored at the write address appear on the data output, while other input data are being stored in the same memory position





BRAMs – Timing (*READ_FIRST*)

```
entity blockram is port (
    clk  : in  STD_LOGIC;
    we   : in  STD_LOGIC;
    en   : in  STD_LOGIC;
    addr : in  STD_LOGIC_VECTOR (8 downto 0);
    di   : in  STD_LOGIC_VECTOR (31 downto 0);
    do   : out STD_LOGIC_VECTOR (31 downto 0));
end blockram;

architecture Behavioral of blockram is
    type ram_type is array (511 downto 0) of std_logic_vector (31 downto 0);
    signal ram : ram_type;
begin
    process( clk )
    begin
        if rising_edge(clk) then
            if en = '1' then
                do <= ram(to_integer(unsigned(addr)));
                if we = '1' then
                    ram(to_integer(unsigned(addr))) <= di;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

BRAMs – Timing (*READ_FIRST*)

- Inferring the READ_FIRST mode:
  - Messages visible in the Synthesis log:

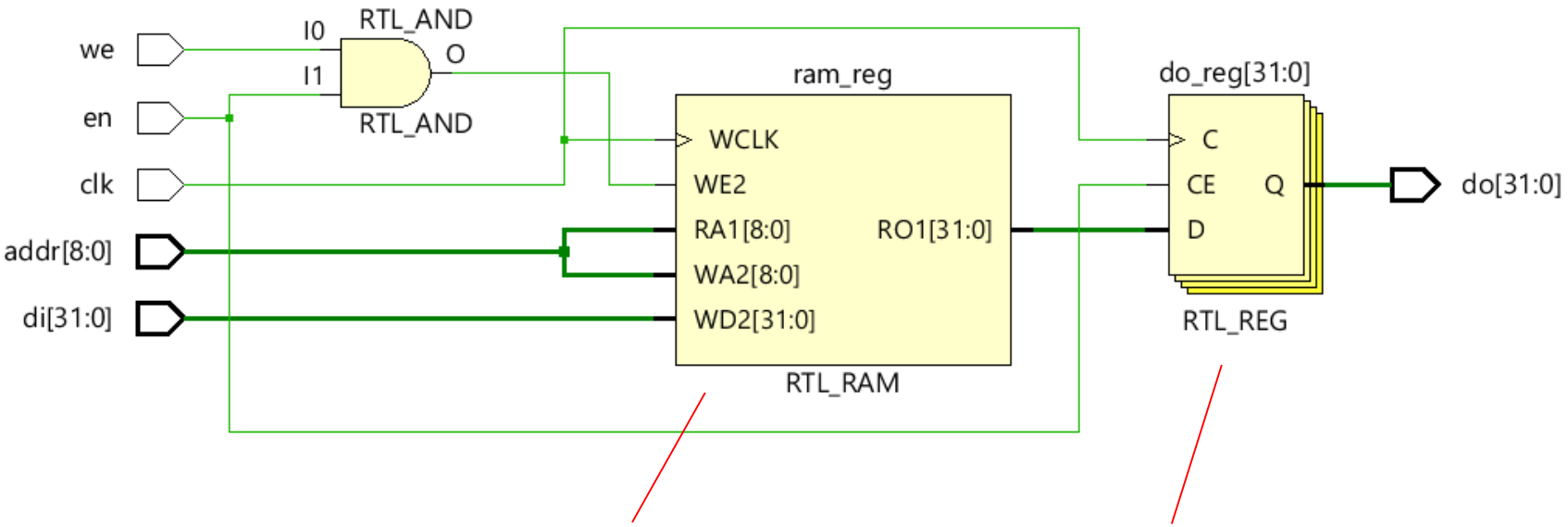
```
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
Hierarchical RTL Component report
Module blockram
Detailed RTL Component Info :
+---Registers :
                               32 Bit    Registers := 1
+---RAMs :
                              16K Bit         RAMs := 1
---------------------------------------------------------------------------------

.
.

Block RAM: Preliminary Mapping  Report (see note below)
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|Module Name | RTL Object | PORT A (Depth x Width)  | W | R | PORT B (Depth x Width)  | W | R | Ports driving FF | RAMB18 | RAMB36 |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
|blockram    | ram_reg    | 512 x 32(READ_FIRST)    | W | R |                         |   |   | Port A           | 1      | 0      |
+------------+------------+-------------------------+---+---+-------------------------+---+---+------------------+--------+--------+
```

  - Messages visible in the Implementation “report_utilization” log:

```
3. Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |        50 |  1.00 |
|   RAMB36/FIFO*    |    0 |     0 |        50 |  0.00 |
|   RAMB18          |    1 |     0 |       100 |  1.00 |
|     RAMB18E1 only |    1 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

BRAMs

- Once implemented, the RTL schematic looks like this:



BRAM array. It works as **simple dual port** (port 1 for reading, port 2 for writing)

Output register (embedded in the RAMB18E1 module)