

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 10

Ano Letivo 2022/23

Modelação em VHDL de memórias  
de um porto e multi-porto



# Conteúdo

- Tipos de memórias
  - ROMs e RAMs
    - Conceitos fundamentais
- Definição de *arrays* e subtipos em VHDL
- Modelação de memórias em VHDL
  - Definição e instanciação
    - Vários portos e tipos de acesso
    - Tamanho fixo e parametrizável



# Tipos de Memórias

- *Read Only Memory* (**ROM**)

- Memória apenas de leitura

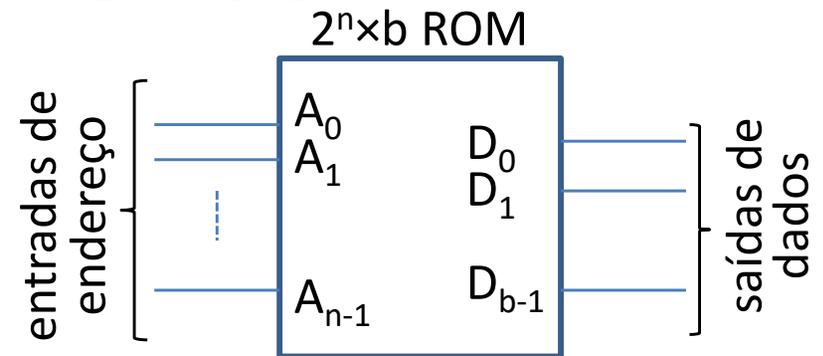
- Num dado instante pode ser lida qualquer posição de memória

- Útil para armazenamento de informação de forma fixa / não volátil

- Tabelas / lookup tables estáticas

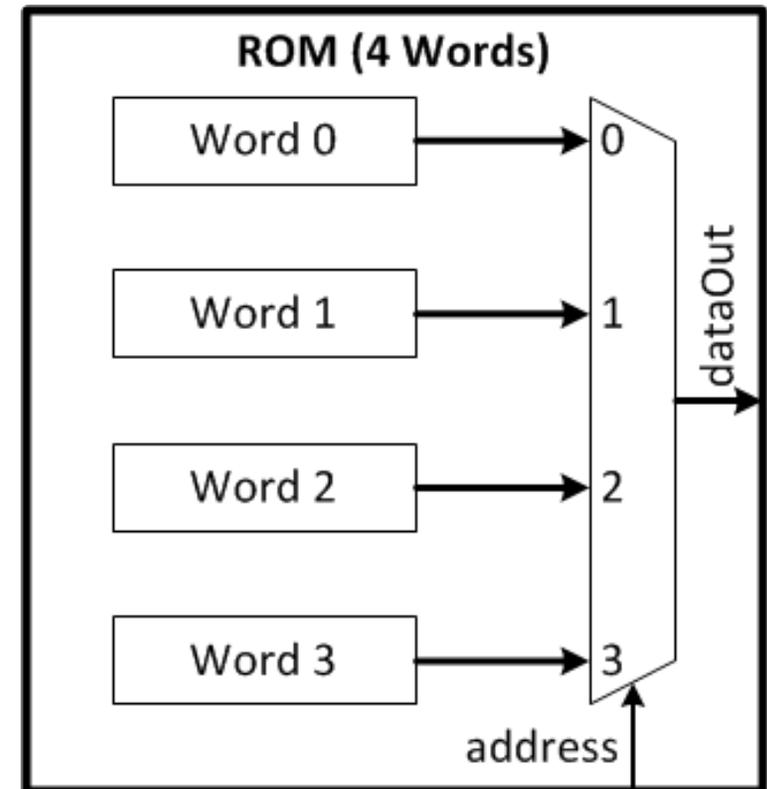
- Mensagens, strings, etc.

- O conteúdo é definido aquando da fabricação ou programação do dispositivo



# Estrutura Interna (conceptual) de uma ROM

- Exemplo com uma ROM de 4 palavras (words)
- A saída de dados (dataOut) é da mesma dimensão que cada palavra de memória (tipicamente potências de base 2)
- Neste caso, a leitura é síncrona ou assíncrona?
- Quantos bits possui a entrada address?
  - Tipicamente potência de base 2
- Para uma memória de 1M words, qual a dimensão da entrada address?

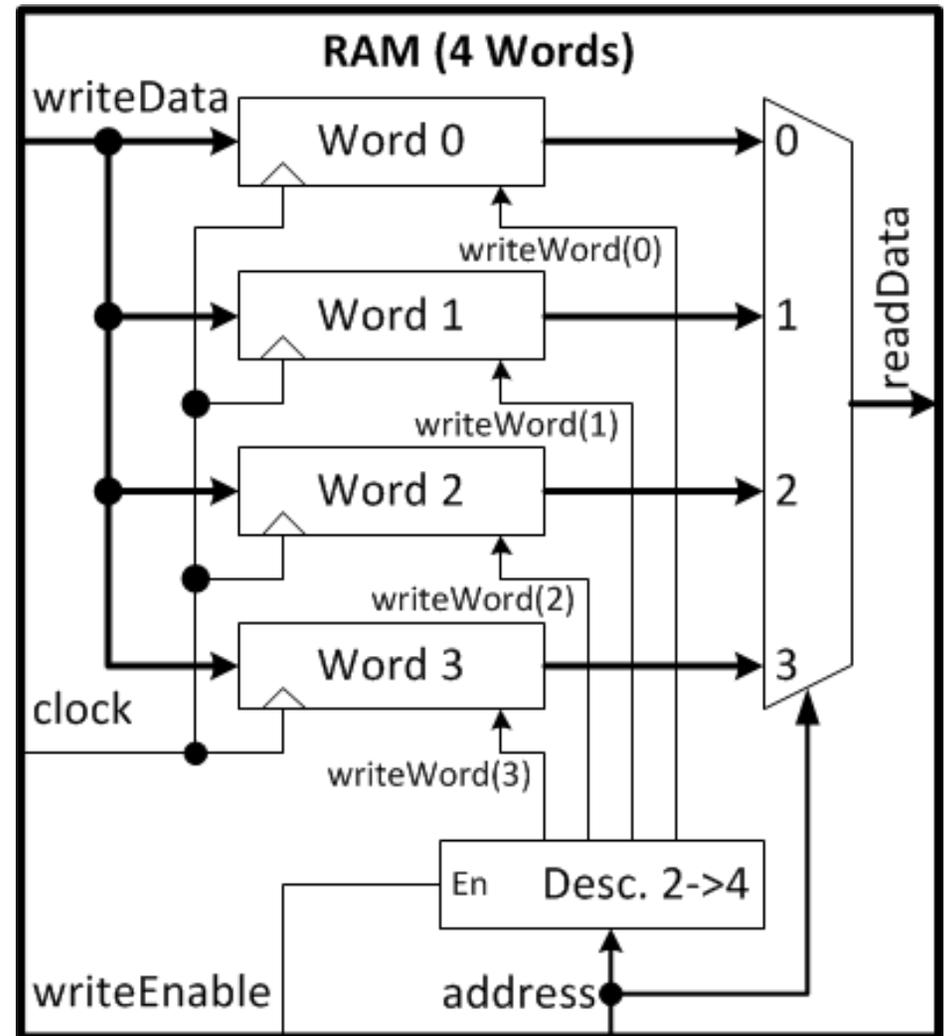


# Tipos de Memórias

- *Random Access Memory* (**RAM**)
  - Memória de leitura / escrita
  - Num dado instante pode ser lida ou escrita qualquer posição de memória
  - Útil para armazenamento de informação arbitrária que pode ser alterada durante o funcionamento do sistema
    - O conteúdo é escrito durante a operação do sistema
  - Pode ser de vários tipos
    - Estática, dinâmica (a abordar noutras UCs – AC2, ...)
  - Pode disponibilizar
    - 1 Porto (apenas uma operação de leitura ou escrita num dado instante) – caso mais frequente
    - Multi-porto (várias operações de leitura e/ou escrita num dado instante)

# Estrutura Interna (conceptual) de uma RAM

- Exemplo com uma RAM de 4 palavras (words)
- A entrada de dados (**writeData**) e a saída de dados (**readData**) são da mesma dimensão de cada palavra de memória
- Neste caso
  - A leitura é síncrona ou assíncrona?
  - A escrita é síncrona ou assíncrona?



# Modelação em VHDL de Memórias

- As estruturas de memória anteriores podem ser modeladas em VHDL
  - Recorrendo a registos, multiplexadores e descodificadores, mas
    - Seria complexo, penoso e sujeito a erros
      - Problemas que se agravam com o aumento da dimensão da memória
    - Não permitiria, no caso das FPGAs, tirar partido dos blocos de memória disponibilizados nas arquiteturas das FPGAs atuais
  - **Recomendação: modelação comportamental de memórias em VHDL**
    - Apresentada nos próximos slides



# Definição de Arrays em VHDL

- Sintaxe para definição do tipo do *array*

```
type type_name is array (range) of element_type;
```

- Declaração de um sinal do tipo definido

```
signal signal_name : type_name;
```

- Exemplo para um array de 4 posições, cada uma do tipo `std_logic_vector` de 8 bits

```
type TMemory is array (0 to 3) of std_logic_vector(7 downto 0);
```

```
signal s_memory : TMemory;
```



# Modelação de uma ROM - Entidade

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;
```

**Subtype** define um conjunto restringido de valores do seu tipo base

```
entity ROM_8x4 is  
    port(address : in  std_logic_vector(2 downto 0);  
          dataOut : out std_logic_vector(3 downto 0));  
end ROM_8x4;
```

**Constant** permite definir valores constantes

```
architecture Behavioral of ROM_8x4 is  
    subtype TDataWord is std_logic_vector(3 downto 0);  
    type TROM is array (0 to 7) of TDataWord;  
    constant c_memory: TROM := ("0000", "0001", "0010", "0100",  
                                "1000", "1111", "1010", "0101");  
begin  
    dataOut <= c_memory(to_integer(unsigned(address)));  
end Behavioral;
```

Função **to\_integer** converte valores (un)signed em valores do tipo (natural) integer.

# Exemplo de Instanciação de uma ROM

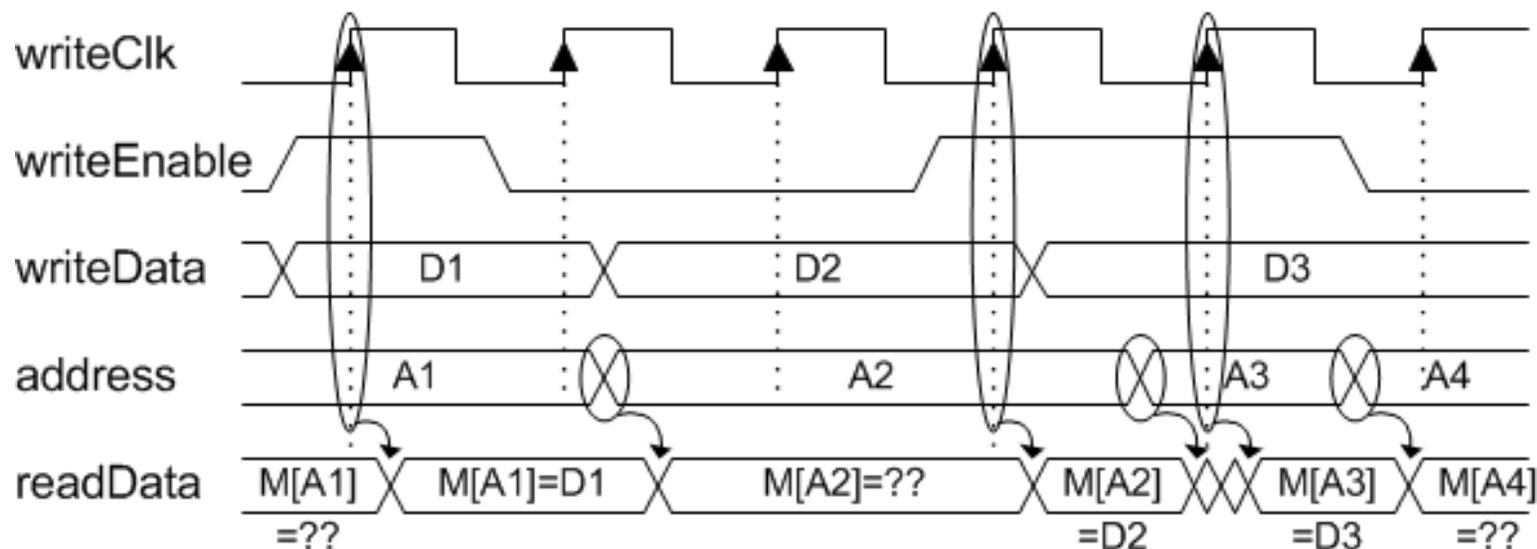
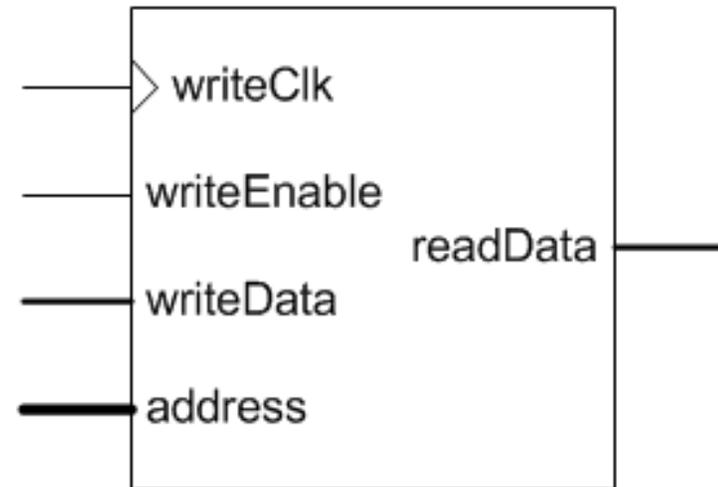
- Exemplo com um contador ligado à entrada de endereço da ROM para efetuar uma leitura sequencial e cíclica de todas as posições da memória

```
...  
addr_counter : entity work.Counter(Behavioral)  
    port map(clk      => s_clk,  
            reset    => s_reset,  
            cntOut   => s_address);  
  
memory : entity work.ROM_8x4(Behavioral)  
    port map(address => s_address,  
            dataOut => s_romData);  
...
```

- As palavras “0000”, “0001”, “0010”, “0100”, “1000”, “1111”, “1010” e “0101” (conteúdo predefinido da ROM) surgirão na saída à medida que o contador vai sendo incrementado

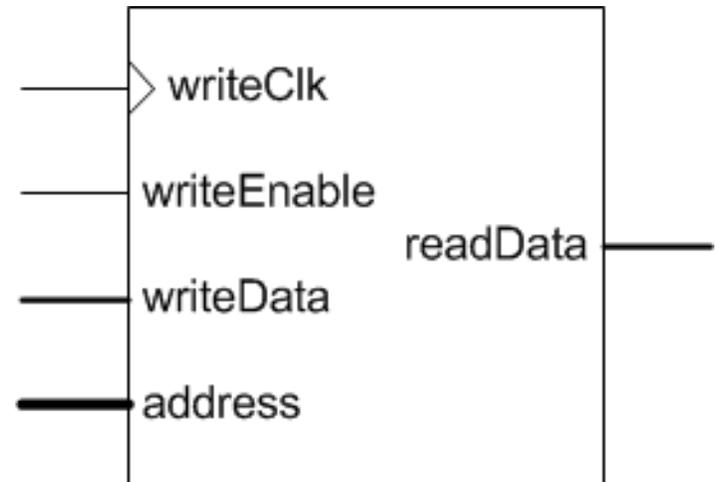
# Modelação de uma RAM 32x8 bit (1 porto; leitura assíncrona)

- Uma entrada de endereço
- Escrita síncrona
- Leitura assíncrona



# Modelação de uma RAM 32x8 bit (1 porto; leitura assíncrona) - Entidade

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
entity RAM_1_SWR_ARD is  
    port (writeClk      : in  std_logic;  
          writeEnable   : in  std_logic;  
          writeData     : in  std_logic_vector(7 downto 0);  
          address       : in  std_logic_vector(4 downto 0);  
          readData      : out std_logic_vector(7 downto 0));  
end RAM_1_SWR_ARD;
```



# Modelação de uma RAM 32x8 bit (1 porto; leitura assínc.) - Arquitetura

```
architecture Behavioral of RAM_1_SWR_ARD is
```

```
    constant NUM_WORDS : integer := 32;  
    subtype TDataWord is std_logic_vector(7 downto 0);  
    type TMemory is array (0 to NUM_WORDS-1) of TDataWord;  
    signal s_memory : TMemory;
```

```
begin
```

```
    process(writeClk)
```

```
    begin
```

```
        if (rising_edge(writeClk)) then
```

```
            if (writeEnable = '1') then
```

```
                s_memory(to_integer(unsigned(address))) <= writeData;
```

```
            end if;
```

```
        end if;
```

```
    end process;
```

```
    readData <= s_memory(to_integer(unsigned(address)));
```

```
end Behavioral;
```



# Exemplo de Instanciação de uma RAM numa Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

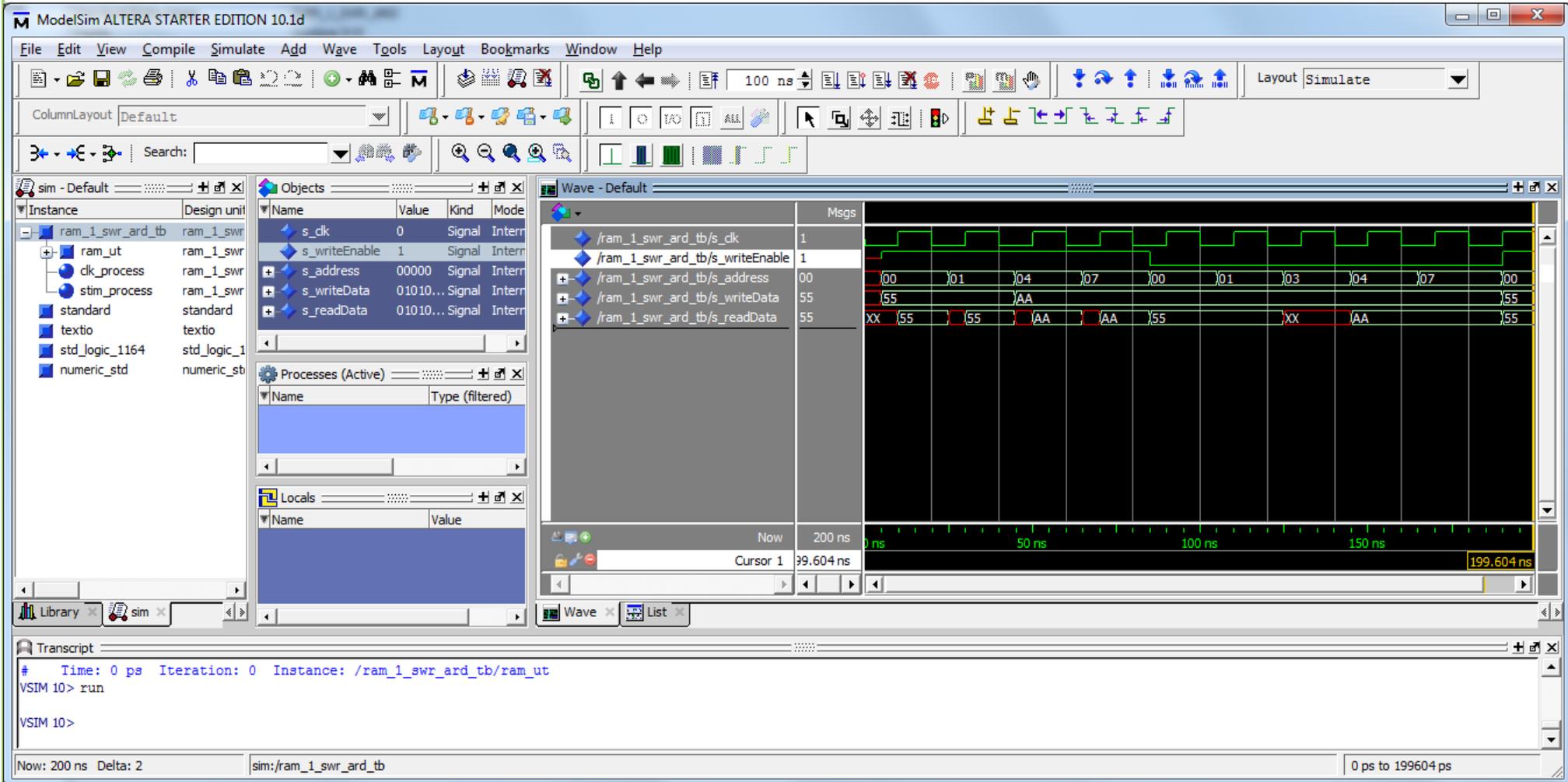
entity RAM_1_SWR_ARD_Tb is
end RAM_1_SWR_ARD_Tb;

architecture Stimulus of RAM_1_SWR_ARD_Tb is
    signal s_clk, s_writeEnable : std_logic;
    signal s_address      : std_logic_vector(4 downto 0);
    signal s_writeData   : std_logic_vector(7 downto 0);
    signal s_readData    : std_logic_vector(7 downto 0);
begin
    ram_ut : entity work.RAM_1_SWR_ARD(Behavioral)
        port map(writeClk      => s_clk,
                writeEnable => s_writeEnable,
                writeData   => s_writeData,
                address     => s_address,
                readData    => s_readData);

    clk_process : process
    begin
        s_clk <= '0'; wait for 10 ns;
        s_clk <= '1'; wait for 10 ns;
    end process;
```

```
stim_process : process
begin
    wait for 5 ns;
    s_writeEnable <= '1';
    s_writeData   <= X"55";
    s_address <= "00000";
    wait for 20 ns;
    s_address <= "00001";
    wait for 20 ns;
    s_writeData <= X"AA";
    s_address <= "00100";
    wait for 20 ns;
    s_address <= "00111";
    wait for 20 ns;
    s_writeEnable <= '0';
    s_address <= "00000";
    wait for 20 ns;
    s_address <= "00001";
    wait for 20 ns;
    s_address <= "00011";
    wait for 20 ns;
    s_address <= "00100";
    wait for 20 ns;
    s_address <= "00111";
    wait for 20 ns;
end process;
end Stimulus;
```

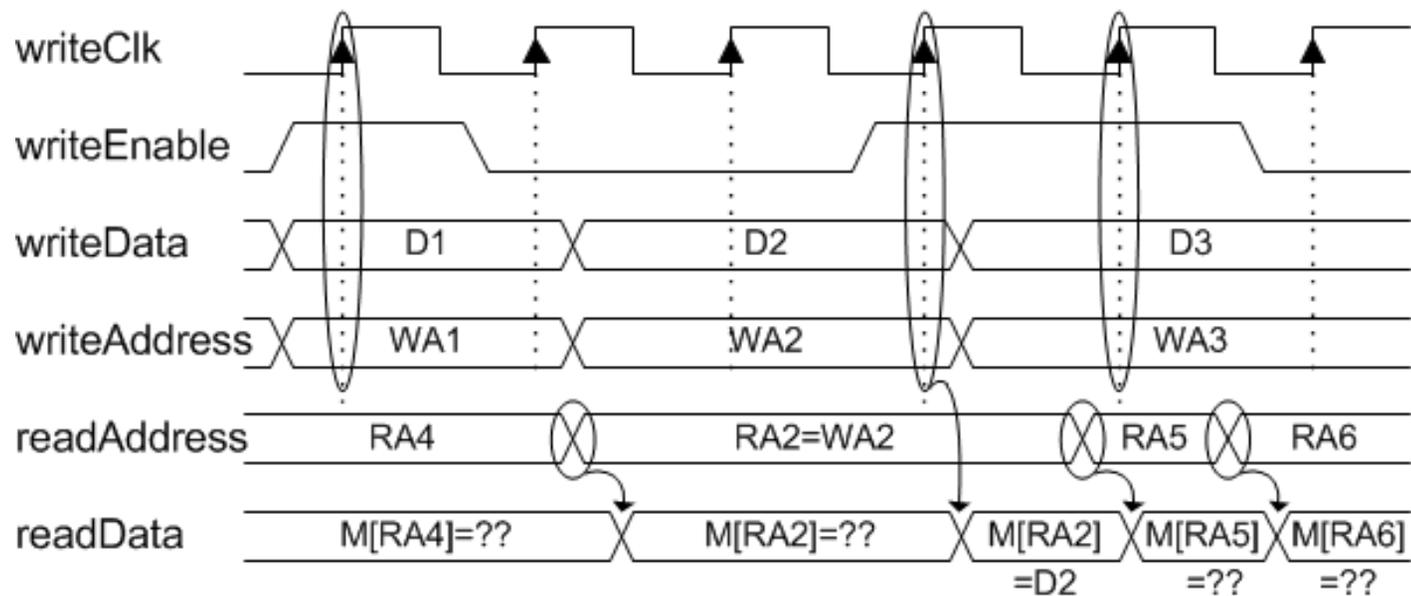
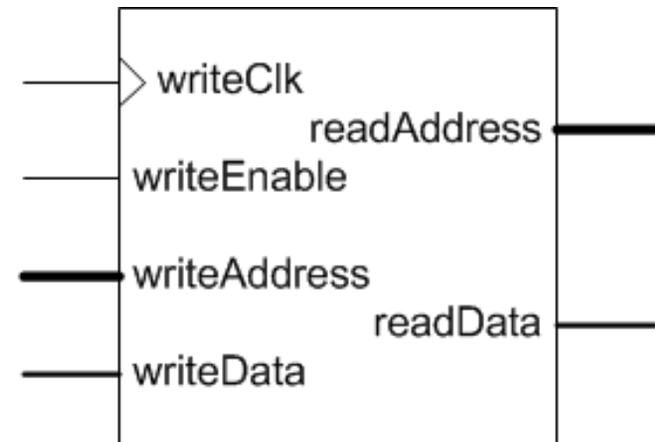
# Resultado da Simulação



- Porquê **XX** em alguns instantes da simulação?
- Porque não dotar a memória de um sinal de *reset*?

# Modelação de uma RAM de 2 portos (1 de escrita; 1 de leitura assíncrona)

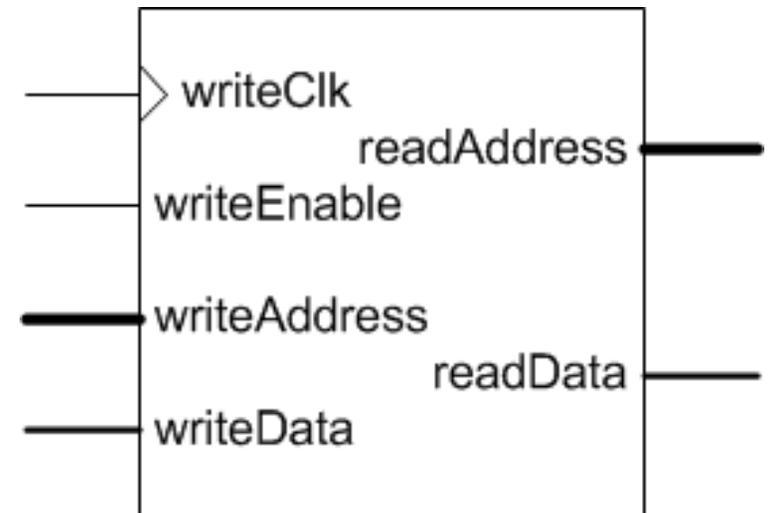
- Duplo porto de endereço
- Escrita síncrona
- Leitura assíncrona



# Modelação de uma RAM de 2 portos (escrita; leitura assíncrona) - Entidade

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity RAM_1SWR_1ARD is
    port(writeClk      : in  std_logic;
          writeEnable  : in  std_logic;
          writeAddress  : in  std_logic_vector(4 downto 0);
          writeData     : in  std_logic_vector(7 downto 0);
          readAddress   : in  std_logic_vector(4 downto 0);
          readData      : out std_logic_vector(7 downto 0));
end RAM_1SWR_1ARD;
```

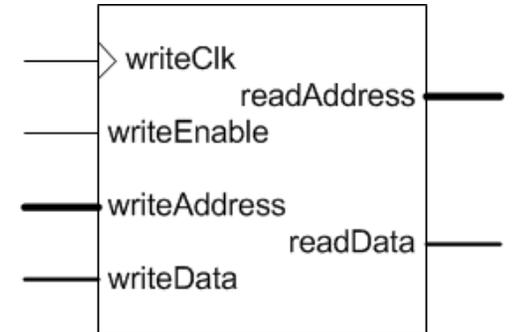


# Modelação de uma RAM de 2 portos (escrita; leitura assíncrona) – Arquit.

```
architecture Behavioral of RAM_1SWR_1ARD is
    constant NUM_WORDS : integer := 32;
    subtype TDataWord is std_logic_vector(7 downto 0);
    type TMemory is array (0 to NUM_WORDS-1) of TDataWord;
    signal s_memory : TMemory;

begin
    process (writeClk)
    begin
        if (rising_edge(writeClk)) then
            if (writeEnable = '1') then
                s_memory(to_integer(unsigned(writeAddress))) <= writeData;
            end if;
        end if;
    end process;

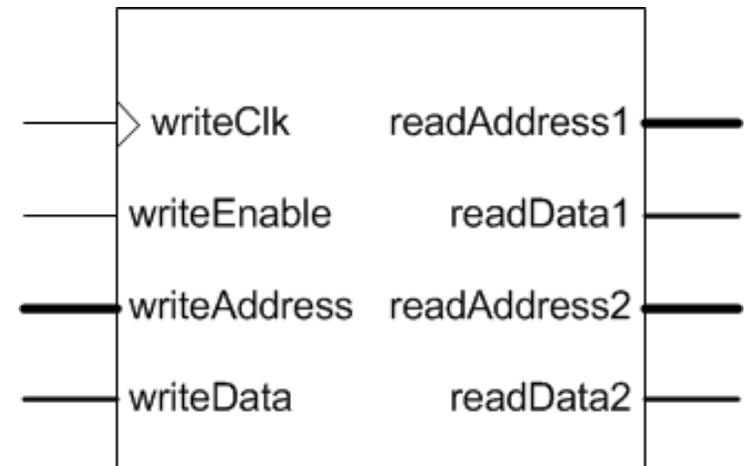
    readData <= s_memory(to_integer(unsigned(readAddress)));
end Behavioral;
```



# Modelação de uma RAM de 3 portos (1 de escrita; 2 de leitura assíncrona)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity RAM_1SWR_2ARD is
    port(writeClk      : in  std_logic;
          writeEnable  : in  std_logic;
          writeAddress : in  std_logic_vector(4 downto 0);
          writeData    : in  std_logic_vector(7 downto 0);
          readAddress1 : in  std_logic_vector(4 downto 0);
          readData1    : out std_logic_vector(7 downto 0);
          readAddress2 : in  std_logic_vector(4 downto 0);
          readData2    : out std_logic_vector(7 downto 0));
end RAM_1SWR_2ARD;
```



# Modelação de uma RAM de 3 portos (1 de escrita; 2 de leitura assíncrona)

```
architecture Behavioral of RAM_1SWR_2ARD is
    constant NUM_WORDS : integer := 32;
    subtype TDataWord is std_logic_vector(7 downto 0);
    type TMemory is array (0 to NUM_WORDS-1) of TDataWord;
    signal s_memory : TMemory;

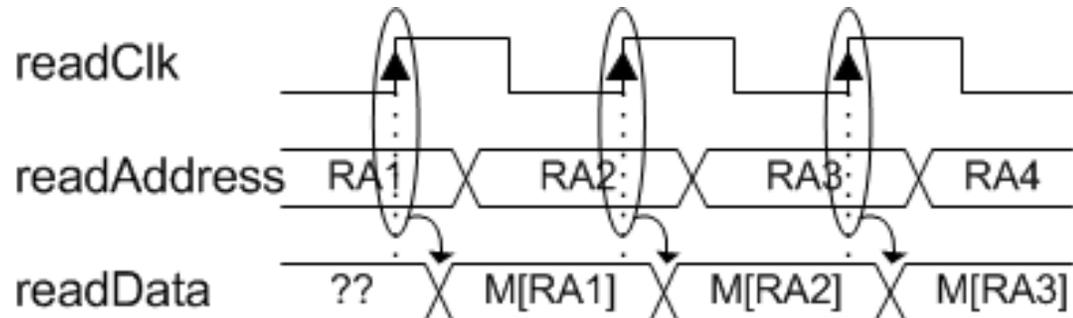
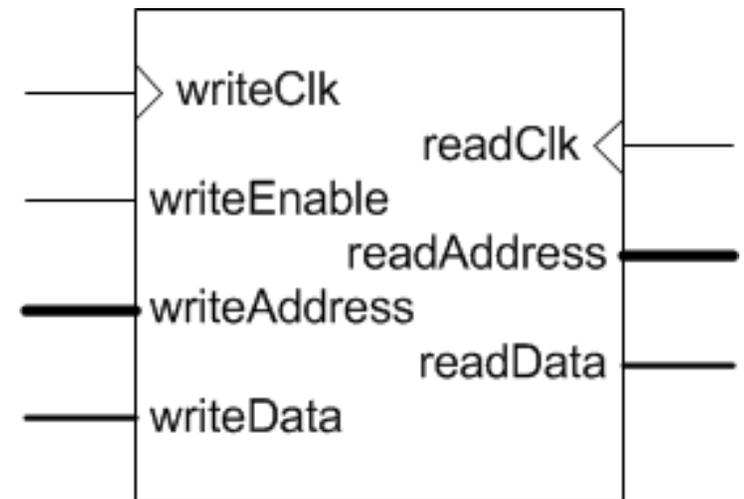
begin
    process (writeClk)
    begin
        if (rising_edge(writeClk)) then
            if (writeEnable = '1') then
                s_memory(to_integer(unsigned(writeAddress))) <= writeData;
            end if;
        end if;
    end process;

    readData1 <= s_memory(to_integer(unsigned(readAddress1)));
    readData2 <= s_memory(to_integer(unsigned(readAddress2)));
end Behavioral;
```



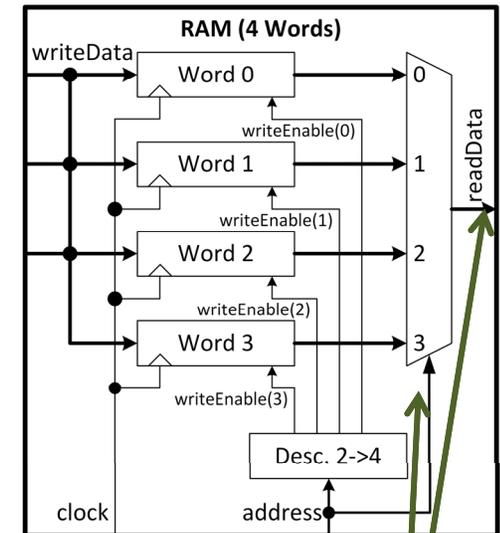
# Modelação de uma RAM de 2 portos (1 de escrita; 1 de leitura síncrona)

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
entity RAM_1SWR_1SRD is  
    port (writeClk      : in  std_logic;  
          writeEnable   : in  std_logic;  
          writeAddress  : in  std_logic_vector(4 downto 0);  
          writeData     : in  std_logic_vector(7 downto 0);  
          readClk       : in  std_logic;  
          readAddress   : in  std_logic_vector(4 downto 0);  
          readData      : out std_logic_vector(7 downto 0));  
end RAM_1SWR_1SRD;
```



# Modelação de uma RAM de 2 portos (1 de escrita; 1 de leitura síncrona)

```
architecture Behavioral of RAM_1SWR_2ARD is
    constant NUM_WORDS : integer := 32;
    subtype TDataWord is std_logic_vector(7 downto 0);
    type TMemory is array (0 to NUM_WORDS-1) of TDataWord;
    signal s_memory : Tmemory;
begin
    write_proc : process(writeClk)
    begin
        if (rising_edge(writeClk)) then
            if (writeEnable = '1') then
                s_memory(to_integer(unsigned(writeAddress))) <= writeData;
            end if;
        end if;
    end process;
    read_proc : process(readClk)
    begin
        if (rising_edge(readClk)) then
            readData <= s_memory(to_integer(unsigned(readAddress)));
        end if;
    end process;
end Behavioral;
```



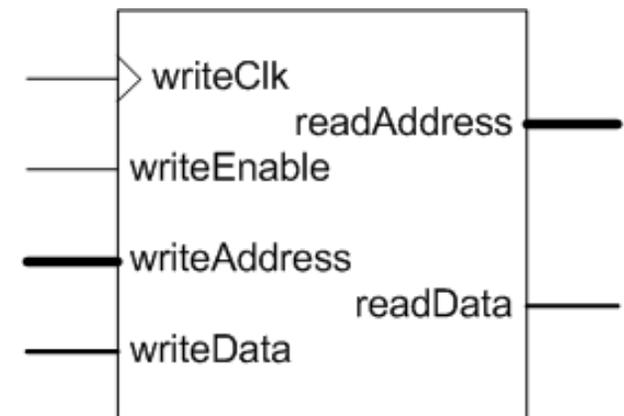
**Leitura síncrona**  
– equivale a  
colocar um  
registo num  
destes pontos

# Parametrização de uma Memória (profundidade e largura)

- RAM de duplo porto parametrizável (profundidade, i.e. número de palavras; largura, i.e. número de bits por palavra) com porto de escrita síncrono e leitura assíncrono

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;
```

```
entity RAM_SW_AR is  
  generic (addrBusSize : integer := 4;  
          dataBusSize : integer := 8);  
  port (writeClk      : in  std_logic;  
        writeEnable  : in  std_logic;  
        writeAddress  : in  std_logic_vector((addrBusSize - 1) downto 0);  
        writeData     : in  std_logic_vector((dataBusSize - 1) downto 0);  
        readAddress   : in  std_logic_vector((addrBusSize - 1) downto 0);  
        readData      : out std_logic_vector((dataBusSize - 1) downto 0));  
end RAM_SW_AR;
```



# Parametrização de uma Memória

```
architecture Behavioral of RAM_SW_AR is
    constant NUM_WORDS : integer := (2 ** addrBusSize);
    subtype TDataWord is std_logic_vector((dataBusSize - 1) downto 0);
    type TMemory is array (0 to (NUM_WORDS - 1)) of TDataWord;
    signal s_memory : Tmemory;
begin
    process(writeClk)
    begin
        if (rising_edge(writeClk)) then
            if (writeEnable = '1') then
                s_memory(to_integer(unsigned(writeAddress))) <= writeData;
            end if;
        end if;
    end process;

    readData <= s_memory(to_integer(unsigned(readAddress)));
end Behavioral;
```



# Exemplo de Instanciação de uma RAM Parametrizável (64k×32 bit)

- Uma dada RAM parametrizável pode ser instanciada no mesmo ou em diferentes projetos com diferentes valores dos parâmetros (através do *generic map*)
- Instanciação da memória num módulo ou testbench
- ...

```
memory : entity WORK.RAM_SW_AR(Behavioral)
  generic map(addrBusSize => 16,
              dataBusSize => 32)
  port map(writeClk      => clk,
           writeEnable   => writeMem,
           writeAddress  => writeAddr,
           writeData     => dataToMem,
           readAddress   => readAddr,
           readData      => dataFromMem) ;
```

- ...
- `clk, writeMem` - 1 bit `std_logic`
- `writeAddr, readAddr` - 16 bit `std_logic_vector`
- `dataToMem, dataFromMem` - 32 bit `std_logic_vector`

# Comentários Finais

- No final desta aula e do trabalho prático 10 de LSD, deverá ser capaz de:
  - Modelar em VHDL e utilizar qualquer tipo de memória
    - RAM/ROM
    - 1 ou mais portos
    - Acesso síncrono ou assíncrono
  - Construir e utilizar memórias de tamanho parametrizável
- ... bom trabalho prático 10, disponível no site da UC
  - [elearning.ua.pt](http://elearning.ua.pt)

