



**Universidade Federal de Santa Catarina**  
**Centro Tecnológico – CTC**  
**Departamento de Engenharia Elétrica**



# **“EEL5105 - Circuitos e Técnicas Digitais”**

**Prof. Eduardo Augusto Bezerra**

**Eduardo.Bezerra@ufsc.br**

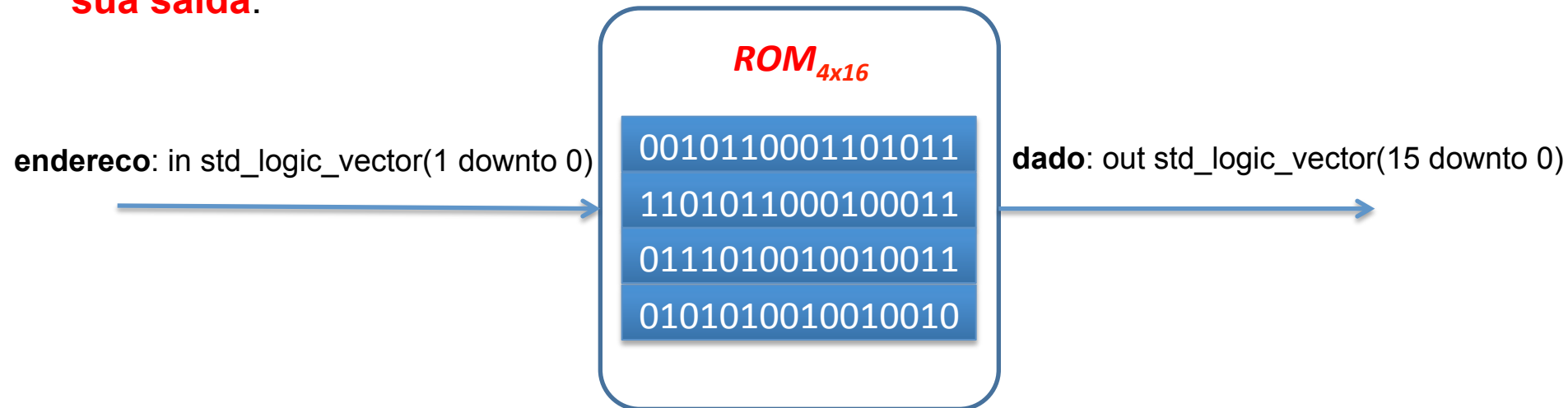
**Florianópolis, agosto de 2015.**

## ***Memória ROM***

# Organização de uma memória ROM

Assumir:

- Memória com capacidade para armazenar 4 palavras, sendo que cada palavra possui um tamanho de 2 bytes (16 bits).
- Não existe nenhum tipo de controle adicional na leitura dos dados, ou seja, basta fornecer o **endereço** do dado desejado, e a memória deverá fornecer esse **dado na sua saída**.



- Na posição (endereço) 0 dessa ROM está armazenado o valor x"2C6B"
- Na posição (endereço) 1 dessa ROM está armazenado o valor x"D623"
- Na posição (endereço) 2 dessa ROM está armazenado o valor x" 7493"
- Na posição (endereço) 3 dessa ROM está armazenado o valor x" A492"

Exemplo de ROM em VHDL obtido no site:  
<http://www.edaboard.com/thread38052.html>

O VHDL original do site foi adaptado para se adequar ao exemplo desses slides. Esse exemplo foi compilado e simulado utilizando a ferramenta on-line:  
<http://www.edaplayground.com/>

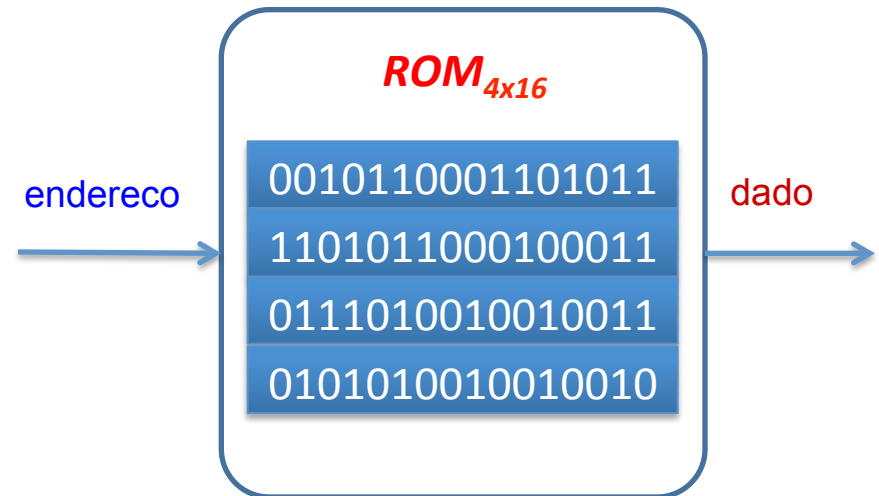
Não foi testado com o Quartus/ModelSim, logo não existe garantia de funcionamento, principalmente por ter sido encontrado na Internet.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM IS
PORT(
    endereco: in std_logic_vector(1 downto 0);
    dado: out std_logic_vector(15 downto 0)
);
END ENTITY;

ARCHITECTURE BEV OF ROM IS
type memoria is array ( 0 to 2**2 - 1) of std_logic_vector(15 downto 0);
constant minhaROM: memoria := (
    0 => "0010110001101011",
    1 => "1101011000100011",
    2 => "0111010010010011",
    3 => "0101010010010010"
);

BEGIN
process (endereco)
begin
    case endereco is
        when "00" => dado <= minhaROM (0);
        when "01" => dado <= minhaROM (1);
        when "10" => dado <= minhaROM (2);
        when "11" => dado <= minhaROM (3);
        when others => dado <= (others => '0');
    end case;
end process;
END BEV;
```



## ***Template de ROM do Quartus II***

# Quartus II – Templates de descrições VHDL com processos

Quartus II 64-Bit - /home/parallels/Documents/teste/teste - teste

File Edit View Project Assignments Processing Tools Window Help

Insert Template

Language templates:

- TCL
- Verilog HDL
- VHDL
  - Full Designs
    - RAMs and ROMs
      - Single-Port RAM
      - Single-Port RAM w/ Initial Content
      - Simple Dual-Port RAM (single clock)
      - Simple Dual-Port RAM (dual clock)
      - True Dual-Port RAM (single clock)
      - True Dual-Port RAM (dual clock)
      - Mixed-Width RAM
      - Mixed-Width True Dual Port RAM
      - Byte-enabled Simple Dual Port RAM
      - Byte-enabled True Dual Port RAM
      - Single-Port ROM**
      - Dual-Port ROM
    - Shift Registers
    - State Machines
    - Arithmetic
    - Configurations
  - Constructs
  - VHDL 2008 Constructs
  - Logic
  - Synthesis Attributes
  - Altera Primitives

Preview:

```
-- Quartus II VHDL Template
-- Single-Port ROM

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity single_port_rom is
    generic
    (
        DATA_WIDTH : integer := 32;
        ADDR_WIDTH  : integer := 8;
    );
    port
    (
        clk      : in std_logic;
        addr     : in std_logic_vector(ADDR_WIDTH-1 downto 0);
        q        : out std_logic_vector(DATA_WIDTH-1 downto 0);
    );
end entity;

architecture rtl of single_port_rom is

    -- Build a 2-D array
    subtype word_t is std_logic_vector(DATA_WIDTH-1 downto 0);
    type memory_t is array (0 to 2**ADDR_WIDTH-1) of word_t;

    function init_rom
    return memory_t;
    variable tmp : memory_t;
begin
```

```
-- Quartus II VHDL Template  
-- Single-Port ROM
```

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;
```

```
entity single_port_rom is
```

```
    generic
```

```
    (
```

```
        DATA_WIDTH : natural := 8;
```

```
        ADDR_WIDTH  : natural := 8
```

```
    );
```

```
    port
```

```
    (
```

```
        clk      : in std_logic;
```

```
        addr     : in natural range 0 to 2**ADDR_WIDTH - 1;
```

```
        q        : out std_logic_vector((DATA_WIDTH - 1) downto 0)
```

```
    );
```

```
end entity;
```

architecture rtl of single\_port\_rom is

-- Build a 2-D array type for the ROM

subtype word\_t is std\_logic\_vector((DATA\_WIDTH-1) downto 0);

type memory\_t is array(2\*\*ADDR\_WIDTH-1 downto 0) of word\_t;

function init\_rom

return memory\_t is

variable tmp : memory\_t := (others => (others => '0'));

begin

for addr\_pos in 0 to 2\*\*ADDR\_WIDTH - 1 loop

-- Initialize each address with the address itself

tmp(addr\_pos) := std\_logic\_vector(to\_unsigned(addr\_pos, DATA\_WIDTH));

end loop;

return tmp;

end init\_rom;

-- Declare the ROM signal and specify a default value.

-- Quartus II will create a memory initialization file (.mif) based on the default value.

signal rom : memory\_t := init\_rom;

begin

process(clk)

begin

if(rising\_edge(clk)) then

q <= rom(addr);

end if;

end process;

end rtl;



## ***Template de RAM do Quartus II***

# Single-port RAM with initial contents (1/4)

-- Single-port RAM with single read/write address and initial contents

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all ;

entity single\_port\_ram\_with\_init is

generic (DATA\_WIDTH : natural := 8;

ADDR\_WIDTH : natural := 6);

port (

clk : in std\_logic;

addr: in natural range 0 to 2\*\*ADDR\_WIDTH - 1;

data: in std\_logic\_vector((DATA\_WIDTH-1) downto 0);

we : in std\_logic := '1';

q : out std\_logic\_vector((DATA\_WIDTH -1) downto 0)  
);

end single\_port\_ram\_with\_init;

# Single-port RAM with initial contents (2/4)

---

**architecture** rtl **of** single\_port\_ram\_with\_init **is**

-- Build a 2-D array type for the RAM

subtype word\_t is std\_logic\_vector((DATA\_WIDTH-1) downto 0);  
type memory\_t is array(2\*\*ADDR\_WIDTH-1 downto 0) of word\_t;

**function** init\_ram **return** memory\_t **is**

**variable** tmp : memory\_t := (others => (others => '0'));

**begin**

**for** addr\_pos **in** 0 **to** 2\*\*ADDR\_WIDTH - 1 **loop**

        -- Initialize each address with the address itself

        tmp(addr\_pos) := std\_logic\_vector(to\_unsigned(addr\_pos,  
  DATA\_WIDTH));

**end loop**;

**return** tmp;

**end** init\_ram;

# Single-port RAM with initial contents (3/4)

---

- Declare the RAM signal and specify a default value.
- Quartus II will create a memory initialization file (.mif) based on
- the default value.

```
signal ram : memory_t := init_ram;
```

- Register to hold the address

```
signal addr_reg : natural range 0 to 2**ADDR_WIDTH-1;
```

# Single-port RAM with initial contents (4/4)

---

```
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            if (we = '1') then
                ram(addr) <= data;
            end if;
            -- Register the address for reading
            addr_reg <= addr;
        end if;
    end process;
    q <= ram(addr_reg);
end rtl;
```