Relatório

Tópico: Aula 7 - Memory Blocks
Grupo: Maicon Chaves Marques 14593530
Pedro Calciolari Jardim 11233668
João Vitor Pereira Candido 13751131

Parte 1:

A RAM possui:

- **32 posições de memória**.
- **4 bits de largura por posição**. Essa arquitetura permite operações de leitura e escrita controladas por um sinal de clock e um sinal de habilitação de escrita.

---

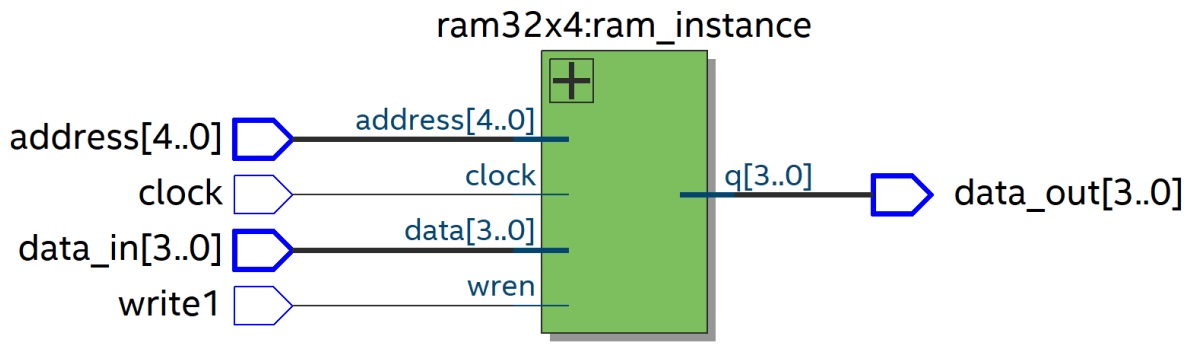
1. **Entradas**:
    - `address` (5 bits): especifica o endereço da posição de memória a ser acessada (0 a 31).
    - `clock`: controla a sincronização das operações de leitura e escrita na RAM.
    - `data_in` (4 bits): representa os dados a serem escritos na RAM.
    - `write1` (1 bit): habilita ou desabilita a operação de escrita na RAM.
2. **Saída**:
    - `data_out` (4 bits): fornece os dados lidos da posição de memória especificada.
-

Código VHDL:

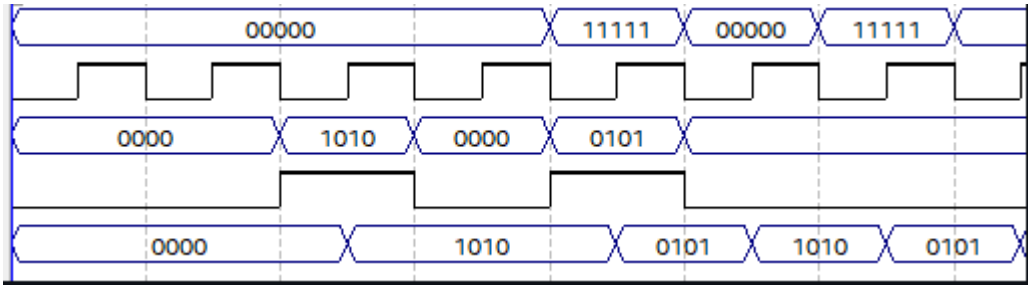
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- Entidade principal que instanciará a RAM
entity aula07 is
   port(
      address   : in  std_logic_vector(4 downto 0); -- 5 bits para o endereço
      clock     : in  std_logic;                    -- sinal de clock
      data_in   : in  std_logic_vector(3 downto 0); -- 4 bits de entrada de dados
      write1    : in  std_logic;                    -- sinal de escrita
      data_out  : out std_logic_vector(3 downto 0)  -- 4 bits de saída de dados
   );
```

```
end aula07;

architecture behavior of aula07 is

    -- Instanciando o componente ram32x4
    component ram32x4
        port(
            address  : in  std_logic_vector(4 downto 0);
            clock    : in  std_logic;
            data     : in  std_logic_vector(3 downto 0);
            wren     : in  std_logic;
            q        : out std_logic_vector(3 downto 0)
        );
    end component;

begin

    -- Instância da RAM com os sinais conectados
    ram_instance: ram32x4
        port map (
            address  => address,
            clock    => clock,
            data     => data_in,
            wren     => write1,
            q        => data_out
        );

end behavior;
```

2 - Representação das portas lógicas geradas pelo Quartus RTL Viewer a partir do código VHDL - Parte 1



4- Simulação ModelSim - Parte 1



Os valores seriam nesta ordem:
adress
clock
data_in
write1
data_out

Parte 2:

Na segunda parte do projeto, o circuito foi implementado e testado diretamente em uma FPGA. O funcionamento pode ser descrito conforme segue:

1. **Clock**: O sinal de relógio (`KEY0`) é usado como entrada para sincronizar as operações do circuito.
2. **Dados de entrada**: Os dados a serem armazenados na memória são representados pelos switches (`SW3-0`). Estes determinam os 4 bits do dado a ser gravado ou exibido.
3. **Endereço de memória**: Os switches (`SW8-4`) representam o endereço de memória onde os dados serão armazenados ou do qual serão lidos.
4. **Sinal de gravação**: O switch (`SW9`) habilita a gravação dos dados na memória. Quando ativado, o circuito grava o valor fornecido pelos switches `SW3-0` no endereço especificado pelos switches `SW8-4`.
5. **Displays de 7 segmentos**:
    - **HEX0**: Exibe o conteúdo lido da memória a partir do endereço especificado.
    - **HEX5-4**: Mostram o endereço de memória atual.
    - **HEX2-0**: Exibem os dados atualmente fornecidos pelos switches (`SW3-0`).

Parte 3:

Nesta parte, a memória foi implementada como um array bidimensional em VHDL, representando uma RAM de 32 posições com 4 bits cada. Essa memória foi configurada para operar com blocos internos do FPGA, permitindo tanto leitura quanto escrita.

Os switches e displays da Parte 2 foram reaproveitados. Os switches definem o endereço da memória e os dados a serem escritos, enquanto os displays mostram o conteúdo da posição selecionada. A gravação acontece quando o sinal `write1` está ativo, salvando os dados na posição indicada. Ao mesmo tempo, o valor armazenado no endereço escolhido é exibido continuamente nos displays.

Segue o código VHDL da parte 3:

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

entity parte03 is
    port(
        address   : in  std_logic_vector(4 downto 0); -- 5 bits para o endereço
        data_in   : in  std_logic_vector(3 downto 0); -- 4 bits para o dado de entrada
        write1    : in  std_logic;                    -- sinal de escrita (write enable)
        clock     : in  std_logic;                    -- sinal de clock
        data_out  : out std_logic_vector(3 downto 0)  -- 4 bits de saída
    );
end parte03;

architecture behavior of parte03 is
    -- Definindo uma memória de 32 palavras (2^5 = 32), cada uma com 4 bits
    type memory_array is array (31 downto 0) of std_logic_vector(3 downto 0);
    signal memory : memory_array := (others => (others => '0')); -- inicializa a memória
begin

    process(clock)
    begin
        if rising_edge(clock) then
            if write1 = '1' then
                -- Escreve o dado de entrada na posição de memória selecionada pelo endereço
                memory(to_integer(unsigned(address))) <= data_in;
            end if;
            -- Sempre lê o dado armazenado na posição de memória selecionada pelo endereço
            data_out <= memory(to_integer(unsigned(address)));
        end if;
    end process;
```

```
end behavior;
```

Parte 4:

O código VHDL implementa uma memória RAM de duas portas, permitindo operações simultâneas de leitura e escrita em diferentes endereços. A RAM foi instanciada diretamente da FPGA, otimizando o uso de recursos internos.

A leitura da RAM é feita automaticamente por meio de um contador que percorre os endereços, exibindo cada palavra armazenada no display `HEX0` por aproximadamente um segundo. O endereço correspondente à leitura é mostrado nos displays `HEX3` e `HEX2`.

Para a escrita, o usuário seleciona o endereço de memória usando os switches `SW8-4` e o valor a ser gravado com os switches `SW3-0`. O endereço e o valor escritos são exibidos nos displays `HEX5` e `HEX4`, enquanto a operação de escrita é controlada pelo sinal `write1`.

O sistema garante a exibição contínua dos dados lidos e fornece flexibilidade para escrita manual, facilitando a interação com a memória.

Segue o código VHDL da parte 4:

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

entity parte04 is
    port(
        clock      : in  std_logic;                    -- sinal de clock
        reset      : in  std_logic;
        data_in    : in  std_logic_vector(3 downto 0); -- 4 bits de entrada de dados (novo valor a
ser escrito)
        rdaddress  : in  std_logic_vector(4 downto 0); -- endereço de leitura de 5 bits
        wraddress  : in  std_logic_vector(4 downto 0); -- endereço de escrita de 5 bits
        write1     : in  std_logic;                    -- sinal de escrita (write enable)
        data_out   : out std_logic_vector(3 downto 0);  -- 4 bits de saída de dados
                     rdata_out  : out std_logic_vector(4 downto 0)  -- 4 bits de saída de dados
    );
end parte04;
```

```
architecture behavior of parte04 is

    signal counter          : integer := 0;
    signal display_address  : std_logic_vector(4 downto 0) := (others => '0');
    constant MAX_COUNT      : integer := 50000000;

    -- Instanciando o componente ram32x42
    component ram32x42
        port(
            clock     : in  std_logic;
            data      : in  std_logic_vector(3 downto 0);
            rdaddress : in  std_logic_vector(4 downto 0);
            wraddress : in  std_logic_vector(4 downto 0);
            wren      : in  std_logic;
            q         : out std_logic_vector(3 downto 0)
        );
    end component;

    -- Sinal de saída da RAM
    signal ram_output : std_logic_vector(3 downto 0);

begin

    -- Instância da RAM com os sinais conectados
    ram_instance: ram32x42
        port map (
            clock     => clock,
            data      => data_in,           -- Dados capturados (agora vindo de data_in)
            rdaddress => display_address,         -- Endereço de exibição do contador
            wraddress => wraddress,        -- Endereço capturado no botão
            wren      => write1,          -- Controle de escrita
            q         => ram_output              -- Saída da RAM
        );



    -- Processo que percorre os endereços de exibição e exibe por 1 segundo cada
    process(clock, reset)

            variable state : STATES;

    begin
        if reset = '0' then
            -- Resetar o contador e o endereço a ser exibido
            counter <= 0;
            display_address <= (others => '0');
        elsif rising_edge(clock) then
            if counter = MAX_COUNT then
```

```
            -- Quando o contador atingir 50 milhões (1 segundo)
            counter <= 0;
            if display_address = "11111" then
                -- Se já passou pelos 32 endereços, reinicia
                display_address <= (others => '0');
            else
                -- Incrementa o endereço
                display_address <= display_address + 1;
            end if;
        else
            -- Incrementa o contador a cada ciclo de clock
            counter <= counter + 1;
        end if;
    end if;
end process;

-- Saída dos dados: sempre o conteúdo lido da RAM baseado no display_address
data_out <= ram_output;
      rdata_out <= display_address;

end behavior;
```