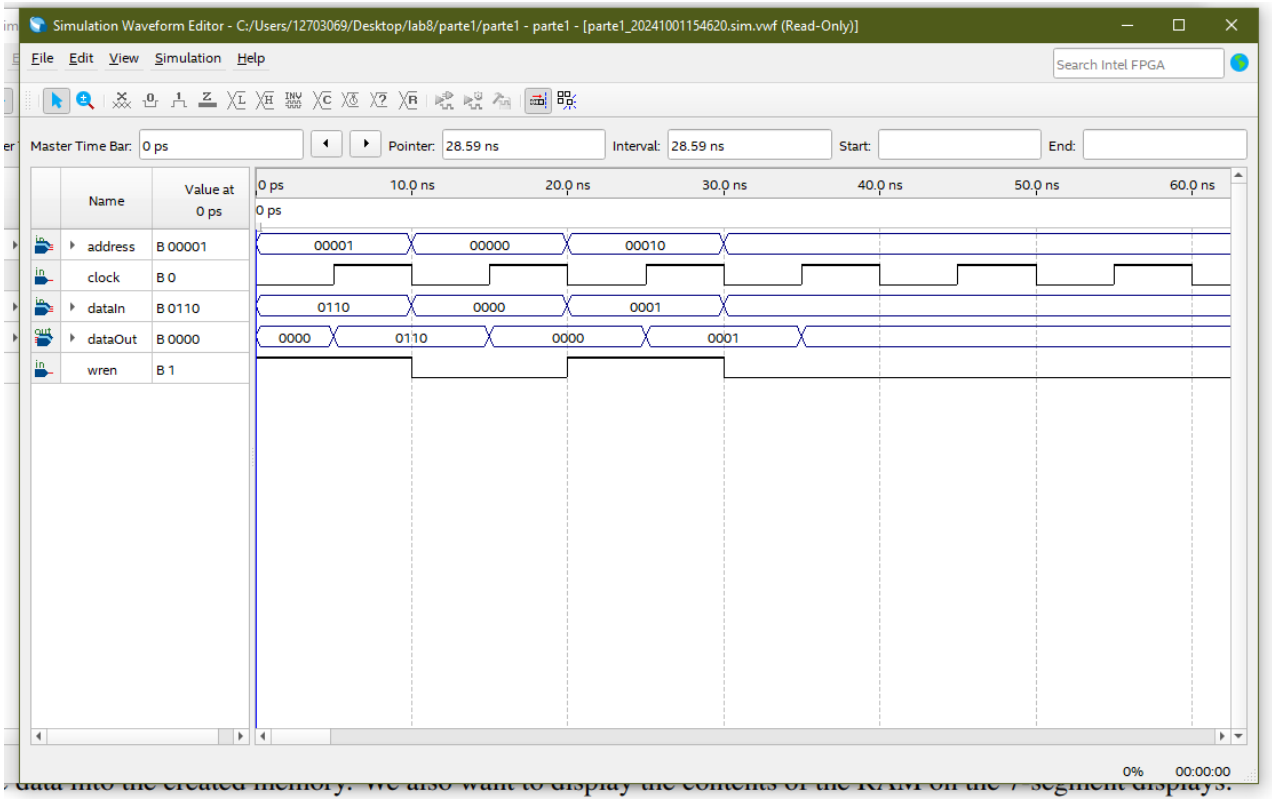
Aula 7 - Memory Blocks

Parte 1: implementamos uma memória RAM de 32 palavras de 4 bits usando módulos predefinidos do Quartus e com o seguinte código fornecido pelo PDF de exercícios:

```
ENTITY ram32x4 IS
   PORT (  address    : IN    STD_LOGIC_VECTOR (4 DOWNTO 0);
           clock      : IN    STD_LOGIC := '1';
           data       : IN    STD_LOGIC_VECTOR (3 DOWNTO 0);
           wren       : IN    STD_LOGIC ;
           q          : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0) );
END ram32x4;
```

Depois, instanciamos esse módulo em um arquivo VHDL e testamos a memória:



A partir de uma borda de subida de clock, enquanto o sinal wren estiver ligado, qualquer conteúdo armazenado em dataIn será escrito na memória no endereço guardado por address. Por padrão, o conteúdo de uma memória não inicializada é 0000. Na imagem acima, o valor 0110 foi armazenado no endereço 00001, pois quando o valor de address é 00001, dataOut, que

representa o conteúdo de um endereço de memória específico, exibe 0110. O mesmo pode ser dito para os outros endereços e seus respectivos valores.

Parte 2:

Criamos um programa em VHDL que instancia a mesma memória RAM, recebe dados por meio das chaves da FPGA e exibe os conteúdos da RAM em displays de 7 segmentos.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity parte2 is
    port (
        clock   : in STD_LOGIC;
        wren    : in STD_LOGIC;
        address : in STD_LOGIC_VECTOR(4 downto 0);
        data    : in STD_LOGIC_VECTOR(3 downto 0);
        q       : out STD_LOGIC_VECTOR(3 downto 0);
        HEX4    : out STD_LOGIC_VECTOR(6 downto 0); -- Endereço 1 digito (4 BITS)
        HEX5    : out STD_LOGIC_VECTOR(6 downto 0); -- Endereço 2 digito (0, 0, 0, MSB)
        HEX2    : out STD_LOGIC_VECTOR(6 downto 0); -- Input dados
        HEX0    : out STD_LOGIC_VECTOR(6 downto 0)  -- Output dados
    );
end parte2;

architecture Behavioral of parte2 is
    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       : buffer STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    function int_to_7seg(d : integer) return std_logic_vector is
    begin
        case d is
            when 0   => return "1000000"; -- 0
            when 1   => return "1111001"; -- 1
            when 2   => return "0100100"; -- 2
            when 3   => return "0110000"; -- 3
            when 4   => return "0011001"; -- 4
            when 5   => return "0010010"; -- 5
            when 6   => return "0000010"; -- 6
            when 7   => return "1111000"; -- 7
            when 8   => return "0000000"; -- 8
            when 9   => return "0010000"; -- 9
            when 10  => return "0001000"; -- A
            when 11  => return "0000011"; -- B
            when 12  => return "1000110"; -- C
            when 13  => return "0100001"; -- D
            when 14  => return "0000110"; -- E
            when 15  => return "0001110"; -- F
            when others => return "0000000"; -- Desconhecido
        end case;
    end function;

    signal outputRAM : STD_LOGIC_VECTOR(3 downto 0);
    signal inputRAM : STD_LOGIC_VECTOR(3 downto 0);
    signal endereco : STD_LOGIC_VECTOR(4 downto 0);
begin

    -- outputRAM <= q; -- Saida da RAM
    -- inputRAM <= data; -- Dados de entrada
    -- endereco <= address;


    memoria : ram32x4
        port map (
            address => endereco,
            clock   => clock,
            data    => inputRAM,
            wren    => wren,
            q       => outputRAM
        );

    process(clock)
    begin
        if rising_edge(clock) then
            endereco <= address(4 downto 0);
            inputRAM <= data(3 downto 0);
        end if;
    end process;

    HEX0 <= int_to_7seg(to_integer(unsigned(outputRAM)));
    HEX2 <= int_to_7seg(to_integer(unsigned(inputRAM)));
    HEX4 <= int_to_7seg(to_integer(unsigned(endereco(3 downto 0)))); -- Endereço 4 bits
    HEX5 <= int_to_7seg(to_integer(unsigned(endereco(4 downto 4)))); -- Endereço MSB

end Behavioral;
```

Utilizamos novamente a função “int_to_7seg”, que traduz sinais binários em símbolos numéricos na base 16. Criamos 4 vetores binários que armazenarão os sinais binários dos displays: HEX4, representa os 4 bits menos significativos do endereço (1 dígito na base 16), HEX5, representa os 4 bits mais significativos do endereço, HEX2, representa o conteúdo que pode ser escrito na memória, HEX0 representa o conteúdo lido na memória.

A cada ciclo de clock, os sinais “endereco” e “inputRAM” são atualizados para os atuais valores dos sinais “address” e “data” respectivamente. Ambos pertencem a instância da memória. Já “outputRAM” é mapeada para o sinal “q” da memória, que simboliza o conteúdo gravado nela em determinado endereço.

Parte 3: ao contrário da parte 1, implementamos a memória RAM sem os módulos predefinidos do Quartus, mas usando um array de 32 elementos, onde cada elemento é um vetor de 4 bits.

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

entity parte3 is
    port (
        clock   : in STD_LOGIC;
        wren    : in STD_LOGIC;
        address : in STD_LOGIC_VECTOR(4 downto 0);
        data    : in STD_LOGIC_VECTOR(3 downto 0);
        q       : out STD_LOGIC_VECTOR(3 downto 0);
        HEX4    : out STD_LOGIC_VECTOR(6 downto 0); -- Endereço 1 dígito (4 BITS)
        HEX5    : out STD_LOGIC_VECTOR(6 downto 0); -- Endereço 2 dígito (0, 0, 0, MSB)
        HEX2    : out STD_LOGIC_VECTOR(6 downto 0); -- Input dados
        HEX0    : out STD_LOGIC_VECTOR(6 downto 0)  -- Output
    );
end parte3;

architecture Behavioral of parte3 is
    TYPE mem IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL memory_array : mem;
    signal outputRAM : STD_LOGIC_VECTOR(3 downto 0);
    signal inputRAM : STD_LOGIC_VECTOR(3 downto 0);
    signal endereco : INTEGER range 0 to 31;

    function int_to_7seg(d : integer) return std_logic_vector is
    begin
        case d is
            when 0   => return "1000000"; -- 0
            when 1   => return "1111001"; -- 1
            when 2   => return "0100100"; -- 2
            when 3   => return "0110000"; -- 3
            when 4   => return "0011001"; -- 4
            when 5   => return "0010010"; -- 5
            when 6   => return "0000010"; -- 6
            when 7   => return "1111000"; -- 7
            when 8   => return "0000000"; -- 8
            when 9   => return "0010000"; -- 9
            when 10  => return "0001000"; -- A
            when 11  => return "0000011"; -- B
            when 12  => return "1000110"; -- C
            when 13  => return "0100001"; -- D
            when 14  => return "0000110"; -- E
            when 15  => return "0001110"; -- F
            when others => return "0000000"; -- Desconhecido
        end case;
    end function;

begin
    process(clock)
    begin
        if falling_edge(clock) then
            endereco <= to_integer(unsigned(address));

            if wren = '1' then
                memory_array(endereco) <= data;
            end if;

            outputRAM <= memory_array(endereco);
        end if;
    end process;

    HEX0 <= int_to_7seg(to_integer(unsigned(outputRAM)));
    HEX2 <= int_to_7seg(to_integer(unsigned(data)));
    HEX4 <= int_to_7seg(endereco mod 16);
    HEX5 <= int_to_7seg(endereco / 16);

end Behavioral;
```

O comportamento da memória em relação ao clock e ao sinal de controle wren foi preservado. Quando há uma borda de descida no clock, um dos 32 endereços disponíveis é escolhido pelo usuário e registrado na memória e se o sinal “wren” estiver habilitado, o dado inserido pelo usuário é armazenado no endereço selecionado anteriormente. Após o fim desse processo, os mesmos vetores usados para os displays para as mesmas funcionalidades na parte 2 são atualizados, fazendo com que os displays representem os dados correspondentes.

Parte 4: desenvolvemos uma memória RAM mais especializada que recebe dois endereços: 1 para operações de escrita na memória, e outro para operações de leitura. Para isso, usamos o modelo predefinido do Quartus denominado RAM: 2-PORT, e criamos um arquivo MIF para inicializar essa memória com determinados valores. Usamos um contador de 1 segundo como o endereço de leitura, fazendo com que os dados da memória sejam lidos automaticamente. O usuário fornecerá o endereço de escrita de memória e seu respectivo dado que será escrito. Um botão será usado para resetar o contador. Todas essas informações serão exibidas na FPGA em displays de 7 segmentos.

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

entity part4 is
    port (
        clock   : in STD_LOGIC;                         -- Clock de 50 MHz
        wren    : in STD_LOGIC;                         -- Sinal de escrita na memória
        address : in STD_LOGIC_VECTOR(4 downto 0);      -- Endereço da memória
        data    : in STD_LOGIC_VECTOR(3 downto 0);      -- Dados para escrita na memória
        q       : out STD_LOGIC_VECTOR(3 downto 0);     -- Dados lidos da memória
        HEX1    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe os dados lidos da memória
        HEX2    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe parte do endereço
        HEX3    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe parte do endereço
        HEX4    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe parte do endereço de escrita
        HEX5    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe parte do endereço de escrita (MSB)
        HEX0    : out STD_LOGIC_VECTOR(6 downto 0);     -- Exibe os dados lidos da memória
        reset   : in  STD_LOGIC;                        -- Reset
        SW      : in STD_LOGIC_VECTOR(8 downto 0)       -- Switches de controle
    );
end part4;

architecture Behavioral of part4 is

    constant COUNT_MAX : integer := 50000000; -- Contador para 1 segundo (50 MHz * 1 s)
    constant ADDRESS_MAX : integer := 32;

    signal counter_1s : integer := 0;
    signal clk_1s : STD_LOGIC := '0';  -- Sinal de clock para 1 segundo

    signal address_counter : integer := 0;
    signal outputRAM : STD_LOGIC_VECTOR(3 downto 0);
    signal inputRAM  : STD_LOGIC_VECTOR(3 downto 0);
    signal endereco  : STD_LOGIC_VECTOR(4 downto 0);
    signal writeadr : STD_LOGIC_VECTOR(4 downto 0);

    component teste
        port (
            rdaddress    : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            wraddress    : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            clock        : IN STD_LOGIC;
            data         : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            wren         : IN STD_LOGIC;
            q            : buffer STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
    end component;

    function int_to_7seg(d : integer) return std_logic_vector is
    begin
        case d is
            when 0  => return "1000000"; -- 0
            when 1  => return "1111001"; -- 1
            when 2  => return "0100100"; -- 2
            when 3  => return "0110000"; -- 3
            when 4  => return "0011001"; -- 4
            when 5  => return "0010010"; -- 5
            when 6  => return "0000010"; -- 6
            when 7  => return "1111000"; -- 7
            when 8  => return "0000000"; -- 8
            when 9  => return "0010000"; -- 9
            when 10 => return "0001000"; -- A
            when 11 => return "0000011"; -- B
            when 12 => return "1000110"; -- C
            when 13 => return "0100001"; -- D
            when 14 => return "0000110"; -- E
            when 15 => return "0001110"; -- F
            when others => return "0000000"; -- Desconhecido
        end case;
    end function;
```

```
begin
    -- Instanciação da memória
    memoria : teste
        port map (
            rdaddress => endereco,
            wraddress => writeadr,
            clock     => clock,
            data      => inputRAM,
            wren      => wren,
            q         => outputRAM
        );

    -- Processo de controle de exibição e escrita na memória
    process(clock, reset)
    begin
        if reset = '1' then
            -- Resetar o contador
            counter_1s <= 0;
            address_counter <= 0;
            HEX0 <= int_to_7seg(0);  -- Exibe 0 no display
            HEX4 <= int_to_7seg(0);  -- Exibe 0 no display
            HEX5 <= int_to_7seg(0);  -- Exibe 0 no display
        elsif rising_edge(clock) then
            -- Incrementar o contador de 1 segundo
            if counter_1s = COUNT_MAX - 1 then
                counter_1s <= 0;

                -- Atualizar displays de endereço e conteúdo
                if address_counter = ADDRESS_MAX - 1 then
                    address_counter <= 0;
                else
                    address_counter <= address_counter + 1;
                end if;

                endereco <= std_logic_vector(to_unsigned(address_counter, endereco'length));

                HEX0 <= int_to_7seg(to_integer(unsigned(outputRAM(3 downto 0))));  -- Dado do endereço
                HEX3 <= int_to_7seg(to_integer(unsigned(endereco(4 downto 4))));  -- Exibe MSB do endereço
                HEX2 <= int_to_7seg(to_integer(unsigned(endereco(3 downto 0))));  -- Exibe LSB do endereço

                -- Atualiza o write address e write data
                writeadr <= SW(8) & SW(7) & SW(6) & SW(5) & SW(4);  -- Pega o endereço dos switches
                inputRAM <= SW(3 downto 0);  -- Pega o dado dos switches

                HEX4 <= int_to_7seg(to_integer(unsigned(writeadr(3 downto 0))));  -- Exibe parte do endereço no HEX4
                HEX5 <= int_to_7seg(to_integer(unsigned(writeadr(4 downto 4))));  -- Exibe o MSB do endereço no HEX5
                HEX1 <= int_to_7seg(to_integer(unsigned(inputRAM(3 downto 0))));  -- Exibe o dado no HEX1

            else
                counter_1s <= counter_1s + 1;
            end if;
        end if;
    end process;

end Behavioral;
```

```
DEPTH = 32;
WIDTH = 4;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
00 : 0000;
01 : 0001;
02 : 0010;
03 : 0011;
04 : 0100;
05 : 0101;
06 : 0110;
07 : 0111;
08 : 1000;
09 : 1001;
0A : 1010;
0B : 1011;
0C : 1100;
0D : 1101;
0E : 1110;
0F : 1111;
10 : 0000;
11 : 0001;
12 : 0010;
13 : 0011;
14 : 0100;
15 : 0101;
16 : 0110;
17 : 0111;
18 : 1000;
19 : 1001;
1A : 1010;
1B : 1011;
1C : 1100;
1D : 1101;
1E : 1110;
1F : 1111;
END;
```

Antes do primeiro ciclo de clock, os sinais endereco, writeadr, clock, inputRAM, wren e outputRAM são mapeados em seus sinais correspondentes na memória instanciada. A cada ciclo de clock em que o sinal de reset estiver desligado, counter_1s é incrementado. Quando esse sinal chega ao valor de COUNT_MAX – 1, 1 segundo se passou, e algumas instruções são executadas. A cada segundo, o contador de endereco e atualizado e exibido em um display de 7 segmentos. Depois, todas as entradas fornecidas pelo usuário através das chaves atualizam os sinais adequados e são exibidos em displays.