Relatório sobre a Parte 1: Implementação do Módulo de Memória

Descrição da Tarefa
Na Parte 1 do exercício, foi realizado o desenvolvimento de um módulo de memória RAM 32x4 em FPGA utilizando os recursos disponíveis no Quartus e a biblioteca de IPs fornecida pela ferramenta. A tarefa envolveu o uso do módulo de memória "RAM: 1-PORT" para criar uma memória com 32 palavras de 4 bits.

Passos Realizados

1. **Criação do Projeto**
   Um novo projeto no Quartus foi iniciado para implementar o módulo de memória.

2. **Configuração do Módulo de Memória**

   - No **IP Catalog**, foi selecionado o módulo **RAM: 1-PORT** (encontrado em Basic Functions > On Chip Memory).
   - O tamanho da memória foi configurado para 32 palavras com 4 bits por palavra.
   - Foi escolhida a tecnologia de memória adequada ao FPGA disponível (M9K ou M10K).
   - Os sinais de entrada foram registrados, enquanto o sinal de saída foi configurado como não registrado.

3. **Instanciação no VHDL**
   O módulo gerado foi instanciado em um arquivo VHDL de nível superior que definiu os sinais de entrada e saída conforme a estrutura representada na Figura 1b do exercício.

4. **Compilação e Relatório**
   A compilação foi realizada com sucesso, e o relatório de compilação confirmou que o módulo utilizou 128 bits em um bloco de memória do FPGA.

5. **Simulação**
   A funcionalidade do circuito foi verificada via simulação, demonstrando operações corretas de leitura e escrita na memória. A saída simulada confirmou que os dados podiam ser armazenados e recuperados corretamente nas posições de memória especificadas.

Resultados e Conclusões
A Parte 1 demonstrou a implementação bem-sucedida de uma memória RAM 32x4 utilizando os blocos de memória do FPGA. Os principais resultados foram:

- O uso eficiente dos recursos de memória do FPGA.

- A verificação da funcionalidade básica de leitura e escrita via simulação.

VHDL:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY aula7part1 IS
    port(
        Adress:     in std_logic_vector(4 downto 0);
        Clock:      in std_logic := '1';
        DataIn:     in std_logic_vector(3 downto 0);
        WriteMode:  in std_logic;
        DataOut:        out std_logic_vector(3  downto 0)
    );
END aula7part1;
ARCHITECTURE Behavioral OF aula7part1 IS
    COMPONENT 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 COMPONENT;
begin
    inst1: entity work.ram32x4
        port map(
            address => Adress,
            clock => Clock,
            data => DataIn,
            wren => WriteMode,
            q => DataOut
        );
END Behavioral;
```

Relatório sobre a Parte 2: Extensão da Memória com Exibição em Displays de 7 Segmentos

Descrição da Tarefa

Na Parte 2, o módulo de memória RAM 32x4 foi estendido para incluir uma interface com displays de 7 segmentos, permitindo visualizar os dados e endereços da memória. A tarefa consistiu na integração de componentes de exibição para mostrar os valores armazenados, os dados de entrada e os endereços ativos.

Passos Realizados

1. **Criação do Projeto VHDL**
   Um novo projeto foi criado, instanciando os componentes de memória (**ram32x4**) e

de exibição (**hex_display**) para implementar a funcionalidade.

2. **Componentes Utilizados**
    - **RAM 32x4**: Módulo de memória que armazena e fornece dados com base no endereço e no modo de escrita.
    - **HEX Display**: Módulo de conversão para exibição de números binários em formato de 7 segmentos.
3. **Mapeamento de Portas**
    - **Entrada e Saída do Módulo de Memória**:
        - `Address`: Endereço da célula de memória sendo acessada.
        - `DataIn`: Dados a serem escritos na memória.
        - `WriteMode`: Sinal para habilitar a escrita.
        - `dataout`: Dados lidos da memória.
    - **Saídas para os Displays**:
        - `hex5`: Exibe o bit mais significativo (Address[4]).
        - `hex4`: Exibe os quatro bits menos significativos do endereço.
        - `hex2`: Exibe os dados de entrada (DataIn).
        - `hex0`: Exibe os dados de saída (dataout).
4. **Lógica de Exibição**
    - Os bits menos significativos do endereço (`Address(3 downto 0)`) e o dado de entrada são diretamente conectados aos displays correspondentes.
    - O bit mais significativo (`Address[4]`) é concatenado com três bits de zero para formar uma entrada de 4 bits exibida no display `hex5`.
    - O dado de saída da memória é conectado ao display `hex0`.
5. **Comportamento do Sistema**
    - Quando um endereço é fornecido, ele é exibido nos displays `hex4` e `hex5`.
    - O dado fornecido na entrada (`DataIn`) aparece no display `hex2`.
    - Após a escrita ou leitura, o dado armazenado na memória no endereço especificado é exibido no display `hex0`.

Resultados e Conclusões

A extensão da memória para incluir a interface com displays de 7 segmentos foi concluída com sucesso. Os principais resultados foram:

- Visualização clara e simultânea do endereço ativo, dos dados de entrada e dos dados armazenados na memória.
- Integração funcional entre os módulos de memória e exibição.

Essa etapa demonstrou como a funcionalidade de memória pode ser aprimorada para interação com o usuário, essencial para depuração e operação em sistemas digitais.

VHDL:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY aula7part2 IS
    port(
        Address     : in std_logic_vector(4 downto 0);
        Clock           : in std_logic := '1';
        DataIn      : in std_logic_vector(3 downto 0);
        WriteMode   : in std_logic;
        hex0, hex2, hex4, hex5  : out std_logic_vector(6 downto 0)
    );
END aula7part2;
ARCHITECTURE Behavioral OF aula7part2 IS
    COMPONENT 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 COMPONENT;
    COMPONENT hex_display IS
        port(
            num4    : in std_logic_vector(3 downto 0);
            hex : out std_logic_vector(6 downto 0)
        );
    END COMPONENT;
    signal dataout : std_logic_vector(3 downto 0);
    signal display4 : std_logic_vector(3 downto 0);
begin
    display4 <= (0 => Address(4), 1 => '0', 2 => '0', 3 => '0');
    ram1: entity work.ram32x4
        port map(
            address => Address,
            clock => Clock,
            data => DataIn,
            wren => WriteMode,
            q => dataout
        );
    hd5: entity work.hex_display
        port map(
            num4 => display4,
            hex => hex5
        );
    hd4: entity work.hex_display
        port map(
            num4 => Address(3 downto 0),
            hex => hex4
        );
    hd2: entity work.hex_display
        port map(
            num4 => DataIn,
            hex => hex2
        );
    hd0: entity work.hex_display
        port map(
            num4 => dataout,
            hex => hex0
        );
END Behavioral;
```

Relatório sobre a Parte 3: Implementação de Memória em VHDL com Array Multidimensional

Descrição da Tarefa

Na Parte 3, foi implementada uma memória RAM 32x4 diretamente em VHDL, utilizando um array multidimensional para representar a memória. Essa abordagem elimina a dependência de blocos de memória predefinidos no FPGA (como M9K/M10K) e demonstra como a lógica de memória pode ser especificada por meio de código.

Passos Realizados

1. **Criação do Tipo e Sinal de Memória**

A memória foi definida como um array multidimensional com 32 palavras de 4 bits:
TYPE mem IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL memory_array : mem := (others => (others => '0'));

   -
   - O conteúdo inicial da memória foi configurado para valores nulos (`'0'`).
2. **Lógica de Escrita e Leitura**

   - Um **processo síncrono ao clock** foi usado para controlar a escrita na memória:
     - Quando ocorre uma borda de subida do clock e o sinal `WriteMode` está ativado (`'1'`), os dados de entrada (`DataIn`) são armazenados na posição indicada por `Address`.
   - Um **sinal auxiliar** (`add`) foi usado para armazenar o endereço atual, permitindo que a memória seja lida fora do processo síncrono.
3. **Conversão de Endereços**

O endereço foi convertido de um vetor lógico para um inteiro utilizando:
to_integer(unsigned(Address))

   -
   - Isso permitiu acessar diretamente as posições do array de memória.
4. **Lógica de Exibição**

   - Assim como na Parte 2, foram utilizados componentes `hex_display` para exibir os valores relevantes nos displays de 7 segmentos:
     - `hex5`: Mostra o bit mais significativo (`Address(4)`).
     - `hex4`: Exibe os quatro bits menos significativos do endereço (`Address(3 downto 0)`).
     - `hex2`: Exibe os dados de entrada (`DataIn`).
     - `hex0`: Exibe os dados de saída lidos da memória (`dataout`).
5. **Comportamento do Sistema**

   - Quando o clock ativa uma borda de subida:
     - Se `WriteMode = '1'`, o valor de `DataIn` é armazenado na posição de memória indicada.
   - O dado lido da memória na posição especificada por `Address` é exibido no display correspondente.

Resultados e Conclusões

A implementação da memória diretamente no VHDL demonstrou os seguintes pontos:

- **Flexibilidade**: A memória foi definida inteiramente em código, permitindo personalizações adicionais.
- **Simplicidade Conceitual**: A lógica de escrita e leitura foi facilmente compreensível e bem integrada aos demais módulos.
- **Visualização**: Assim como nas etapas anteriores, a interface com displays de 7 segmentos garantiu uma exibição clara e simultânea dos endereços, dados de entrada e valores armazenados.

VHDL:

```
    port(
        Address     : in std_logic_vector(4 downto 0);
        Clock           : in std_logic := '1';
        DataIn      : in std_logic_vector(3 downto 0);
        WriteMode   : in std_logic;
        hex0, hex2, hex4, hex5  : out std_logic_vector(6 downto 0)
    );
END aula7part3;
ARCHITECTURE Behavioral OF aula7part3 IS
    COMPONENT hex_display IS
        port(
            num4    : in std_logic_vector(3 downto 0);
            hex : out std_logic_vector(6 downto 0)
        );
    END COMPONENT;

    TYPE mem IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL memory_array : mem := (others => (others => '0'));
    signal add : std_logic_vector(4 downto 0);
    signal dataout : std_logic_vector(3 downto 0);
    signal display4 : std_logic_vector(3 downto 0);
begin
    process (Clock) is
    begin
        if(rising_edge(Clock)) then
            add <= Address;
            if(WriteMode = '1') then
                memory_array(to_integer(unsigned(Address))) <= DataIn;
            end if;
        end if;
    end process;
    dataout <= memory_array(to_integer(unsigned(add)));
    display4 <= (0 => Address(4), 1 => '0', 2 => '0', 3 => '0');
    hd5: entity work.hex_display
        port map(
            num4 => display4,
            hex => hex5
        );
    hd4: entity work.hex_display
        port map(
            num4 => Address(3 downto 0),
            hex => hex4
        );
    hd2: entity work.hex_display
        port map(
            num4 => DataIn,
            hex => hex2
        );
    hd0: entity work.hex_display
        port map(
            num4 => dataout,
            hex => hex0
        );
END Behavioral;
```

Relatório sobre a Parte 4: Memória RAM de Duas Portas com Contador e Exibição em Displays de 7 Segmentos

Descrição da Tarefa
Na Parte 4, foi implementada uma memória RAM de duas portas, permitindo operações de leitura e escrita simultâneas em endereços independentes. Além disso, um contador foi integrado para percorrer automaticamente os endereços de leitura, e a interface de displays de 7 segmentos foi configurada para exibir os valores e endereços relevantes.

Passos Realizados

1. **Configuração da Memória RAM de Duas Portas**
    - Foi utilizado o componente `ram32x4` para implementar uma memória com:
        - **Entrada de Escrita**:
            - Endereço: `waddress`
            - Dados: `DataIn`
            - Controle de escrita: `w`
        - **Saída de Leitura**:
            - Endereço: `raddress` (gerado automaticamente pelo contador).
            - Dados lidos: `display0`.
2. **Contador para Endereços de Leitura**
    - O componente `contador` foi configurado para gerar endereços sequenciais de leitura (`display2_3`) em um intervalo controlado pelo clock (50 MHz):
        - **Reset**: Ativado pela tecla `Key0` (inverso do estado lógico).
        - **Geração de Endereços**: Gera valores de 0 a 31, compatíveis com a largura da RAM.
3. **Conversão para Displays de 7 Segmentos**
    - Foi utilizado o componente `display7seg` para converter números binários em formato decimal e exibi-los nos displays:
        - `hex0`: Dados lidos da memória (`display0`).
        - `hex1`: Dados de entrada para escrita (`DataIn`).
        - `hex2` e `hex3`: Endereço de leitura atual gerado pelo contador.
        - `hex4` e `hex5`: Endereço de escrita fornecido pelo usuário.
4. **Integração dos Componentes**
    - Todos os sinais relevantes foram conectados, garantindo que os endereços de leitura e escrita, bem como os dados associados, fossem exibidos corretamente nos displays.

Comportamento do Sistema

1. **Leitura Automática**:
    - O contador gera automaticamente os endereços de leitura, percorrendo os 32 endereços da memória sequencialmente.
    - Os dados lidos da memória no endereço atual são exibidos no display `hex0`.
2. **Escrita Manual**:
    - O endereço de escrita é fornecido por `waddress`.
    - Quando `w` está ativado (`'1'`), o dado em `DataIn` é armazenado na posição de memória indicada.
3. **Exibição em Displays**:
    - Os valores de leitura, escrita e os endereços associados são exibidos simultaneamente, proporcionando uma visualização clara do estado do sistema.

Resultados e Conclusões

A implementação da RAM de duas portas com contador e interface de exibição foi bem-sucedida, apresentando os seguintes benefícios:

- **Operação Simultânea**: Leitura e escrita podem ocorrer em endereços independentes.
- **Automação**: O contador permite uma leitura cíclica e automática de toda a memória.
- **Visualização**: A interface com displays de 7 segmentos facilita a verificação do funcionamento do sistema, exibindo endereços e dados em tempo real.

VHDL:

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

entity aula7part4 is
 port(
  clk: in STD_LOGIC;
  w: in STD_LOGIC;
  Key0: in STD_LOGIC;
  waddress: in STD_LOGIC_VECTOR (4 downto 0);
  raddress: in STD_LOGIC_VECTOR (4 downto 0);
  DataIn: in STD_LOGIC_VECTOR (3 downto 0);
  hex0: out STD_LOGIC_VECTOR (6 downto 0);
  hex1: out STD_LOGIC_VECTOR (6 downto 0);
  hex2: out STD_LOGIC_VECTOR (6 downto 0);
  hex3: out STD_LOGIC_VECTOR (6 downto 0);
  hex4: out STD_LOGIC_VECTOR (6 downto 0);
  hex5: out STD_LOGIC_VECTOR (6 downto 0)
 );
end entity;

architecture Behaviour of aula7part4 is

 component ram32x4 is
  port(
   clock: in STD_LOGIC  := '1';
   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  := '0';
   q: out STD_LOGIC_VECTOR (3 downto 0)
  );
 end component;

 component contador is
  generic(n: natural:=5; k: STD_LOGIC_VECTOR:="11111"; c: natural:=50000
  port (
   clk: in STD_LOGIC;
   reset: in STD_LOGIC;
   enable: in STD_LOGIC;
   output: out STD_LOGIC_VECTOR (n-1 downto 0);
   rollover: out STD_LOGIC :='0'
  );
 end component;

 component display7seg is
  port(
   in_disp: in STD_LOGIC_VECTOR (3 downto 0);
   out_disp: out STD_LOGIC_VECTOR (6 downto 0)
 );
 end component;

 signal display0: STD_LOGIC_VECTOR (3 downto 0);
 signal display2_3: STD_LOGIC_VECTOR (4 downto 0);

begin

 c: contador
 generic map(
  n => 5,
  k => "11111",
  c => 50000000
 )
```

```
);
port map(
 clk => clk,
 reset => not Key0,
 enable => '1',
 output => display2_3,
 rollover => open
);

ram: ram32x4
port map(
 clock => clk,
 data => DataIn,
 rdaddress => display2_3,
 wraddress => waddress,
 wren => w,
 q => display0
);

d0: display7seg
port map(
 in_disp => display0,
 out_disp => hex0
);

d1: display7seg
port map(
 in_disp => DataIn,
 out_disp => hex1
);

d2: display7seg
port map(
 in_disp => display2_3 (3 downto 0),
 out_disp => hex2
);

d3: display7seg
port map(
 in_disp => "000" & display2_3(4),
 out_disp => hex3
);

d4: display7seg
port map(
 in_disp => waddress (3 downto 0),
 out_disp => hex4
);

d5: display7seg
port map(
 in_disp => "000" & waddress(4),
 out_disp => hex5
);

end architecture Behaviour;
```