Projeto 6 - Simulação de Memórias RAM e ROM em VHDL

Yan Tavares - 202014323

5 de dezembro de 2023

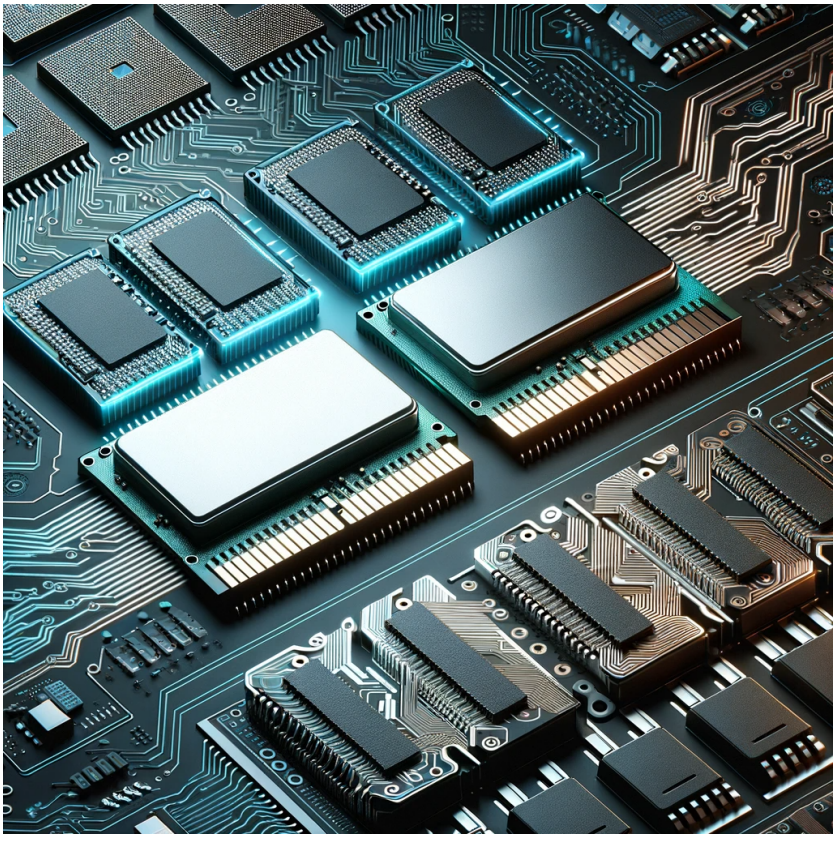


Figura 1: Representação digital de memórias RAM e ROM (DALL-E 3)

Conteúdo

1 Introdução

Este relatório apresenta o projeto, simulação e síntese de memórias RAM e ROM para a arquitetura RISC-V uniciclo. O foco é na implementação de memórias internas para armazenamento de dados e instruções, essenciais para o funcionamento do processador.

2 Características do Banco de Memória

2.1 Memória RAM

A memória RAM possui as seguintes características:

- Barramento de endereço de 8 bits;
- Barramento de dados de entrada e saída de 32 bits;
- Sinal de habilitação de escrita.

2.2 Memória ROM

A memória ROM, utilizada para armazenar instruções, apresenta:

- Barramento de endereço de 8 bits;
- Barramento de dados de saída de 32 bits.

3 Simulação e Verificação

As simulações foram realizadas utilizando o software GHDL e GTKWave para análise das formas de onda, garantindo o funcionamento conforme especificado para cada tipo de memória. É importante ressaltar que os códigos também foram testados no EDA Playground e funcionam como o esperado.

4 Testes e Verificação

Os testes para a RAM envolveram a escrita e leitura de uma sequência de valores. Para a ROM, um conjunto de instruções foi carregado a partir

de um arquivo texto, e a correta leitura dessas instruções foi verificada. O arquivo de texto lido pela ROM possuia uma sequência numérica (0x00000000 - 0x000000FF) para facilitar as asserções dos testes realizados. Foi realizado um teste para cada posição da memória para ambos RAM e ROM. Parte do relatório pode ser encontrado abaixo.

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  GITLENS  AZURE  DEBUG CONSOLE

Testbench_RAM_ROM.vhdl:88:13:@4920ns:(report note): RAM test passed at address 244
Testbench_RAM_ROM.vhdl:88:13:@4940ns:(report note): RAM test passed at address 245
Testbench_RAM_ROM.vhdl:88:13:@4960ns:(report note): RAM test passed at address 246
Testbench_RAM_ROM.vhdl:88:13:@4980ns:(report note): RAM test passed at address 247
Testbench_RAM_ROM.vhdl:88:13:@5us:(report note): RAM test passed at address 248
Testbench_RAM_ROM.vhdl:88:13:@5020ns:(report note): RAM test passed at address 249
Testbench_RAM_ROM.vhdl:88:13:@5040ns:(report note): RAM test passed at address 250
Testbench_RAM_ROM.vhdl:88:13:@5060ns:(report note): RAM test passed at address 251
Testbench_RAM_ROM.vhdl:88:13:@5080ns:(report note): RAM test passed at address 252
Testbench_RAM_ROM.vhdl:88:13:@5100ns:(report note): RAM test passed at address 253
Testbench_RAM_ROM.vhdl:88:13:@5120ns:(report note): RAM test passed at address 254
Testbench_RAM_ROM.vhdl:88:13:@5140ns:(report note): RAM test passed at address 255
Testbench_RAM_ROM.vhdl:104:13:@5150ns:(report note): ROM test passed at address 0
Testbench_RAM_ROM.vhdl:104:13:@5160ns:(report note): ROM test passed at address 1
Testbench_RAM_ROM.vhdl:104:13:@5170ns:(report note): ROM test passed at address 2
Testbench_RAM_ROM.vhdl:104:13:@5180ns:(report note): ROM test passed at address 3
Testbench_RAM_ROM.vhdl:104:13:@5190ns:(report note): ROM test passed at address 4
Testbench_RAM_ROM.vhdl:104:13:@5200ns:(report note): ROM test passed at address 5
Testbench_RAM_ROM.vhdl:104:13:@5210ns:(report note): ROM test passed at address 6
Testbench_RAM_ROM.vhdl:104:13:@5220ns:(report note): ROM test passed at address 7
Testbench_RAM_ROM.vhdl:104:13:@5230ns:(report note): ROM test passed at address 8
```

Figura 2: Trecho das asserções realizadas no arquivo de ‘testbench‘

As formas de onda geradas foram geradas e analisadas para confirmar o comportamento esperado das memórias.

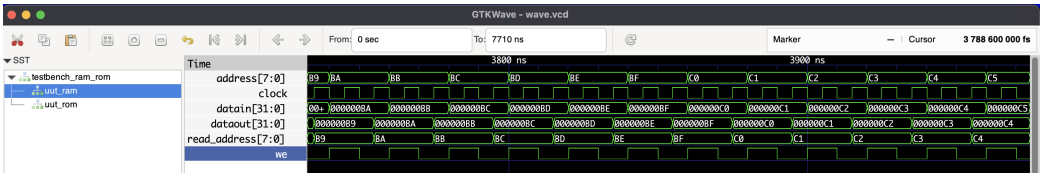
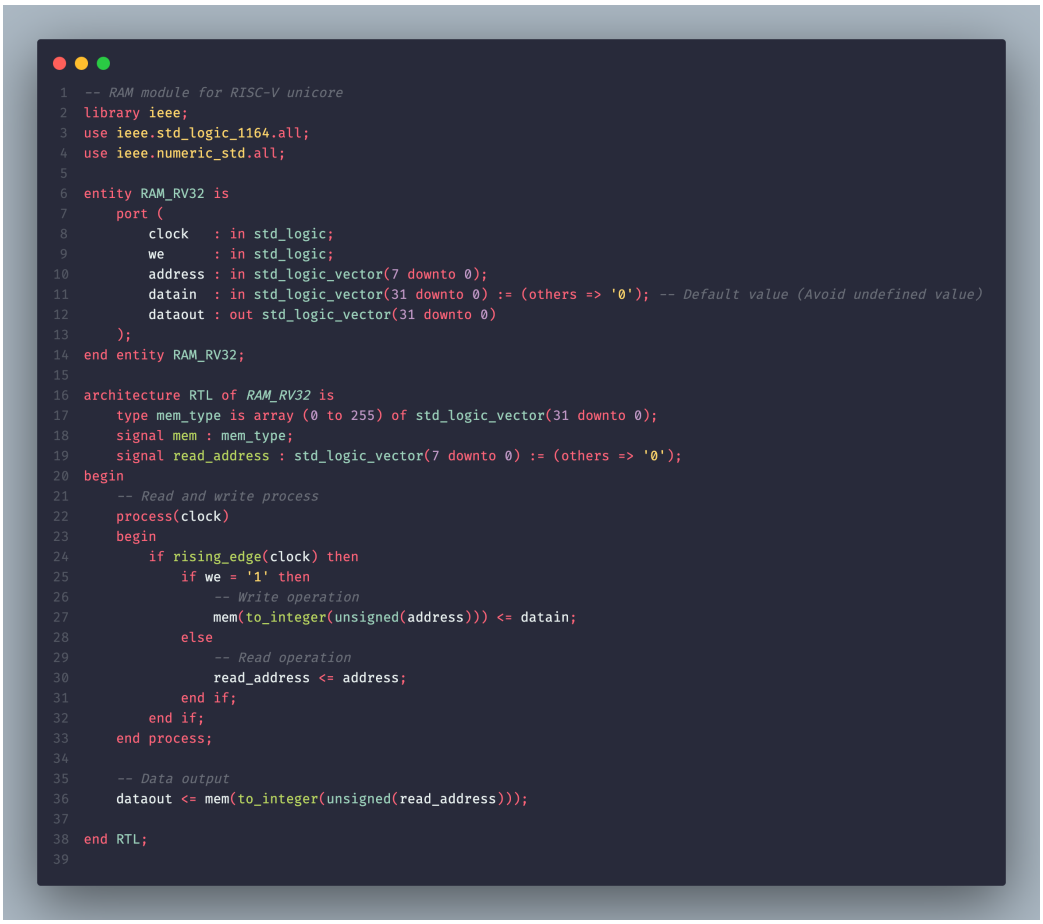


Figura 3: Trecho dos formatos de onda gerados pelo componente uut_ram no arquivo de ‘testbench‘

5 Descrição Detalhada

5.1 Implementação da RAM

A RAM foi implementada em VHDL, permitindo operações de leitura e escrita. Uma abordagem reutilizável foi adotada para a interface da RAM.

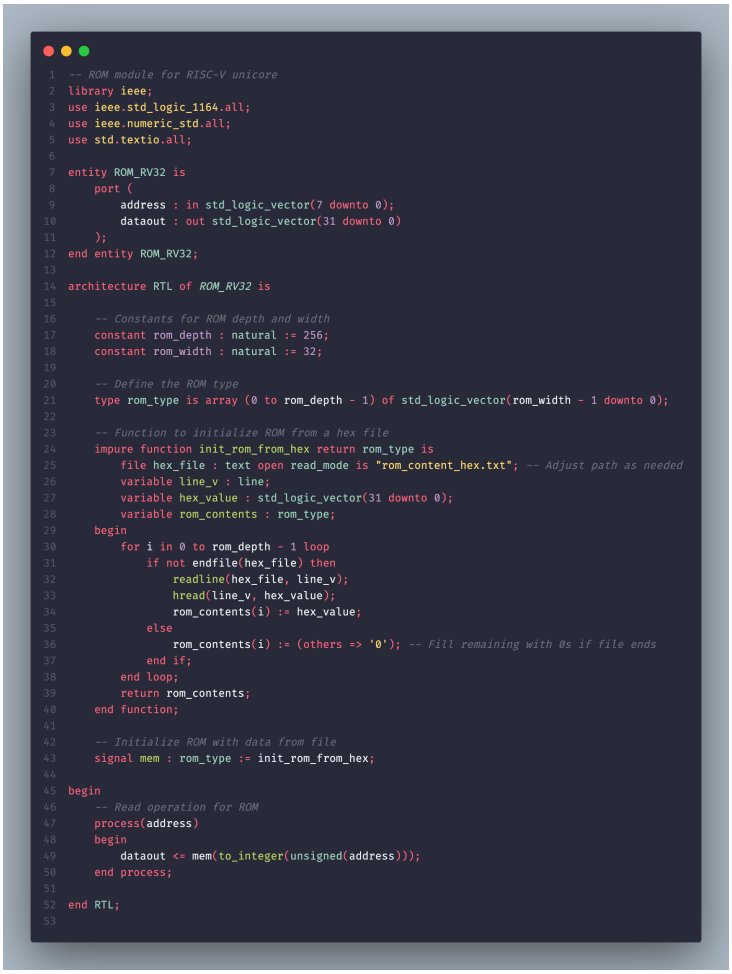


```vhdl
-- RAM module for RISC-V unicore
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RAM_RV32 is
    port (
        clock   : in std_logic;
        we      : in std_logic;
        address : in std_logic_vector(7 downto 0);
        datain  : in std_logic_vector(31 downto 0) := (others => '0'); -- Default value (Avoid undefined value)
        dataout : out std_logic_vector(31 downto 0)
    );
end entity RAM_RV32;

architecture RTL of RAM_RV32 is
    type mem_type is array (0 to 255) of std_logic_vector(31 downto 0);
    signal mem : mem_type;
    signal read_address : std_logic_vector(7 downto 0) := (others => '0');
begin
    -- Read and write process
    process(clock)
    begin
        if rising_edge(clock) then
            if we = '1' then
                -- Write operation
                mem(to_integer(unsigned(address))) <= datain;
            else
                -- Read operation
                read_address <= address;
            end if;
        end if;
    end process;

    -- Data output
    dataout <= mem(to_integer(unsigned(read_address)));

end RTL;

```

Figura 4: Código VHDL responsável pela descrição da memória RAM

5.2 Implementação da ROM

A ROM, sendo apenas de leitura, foi inicializada com instruções a partir de um arquivo de texto. A conformidade com o padrão VHDL-2008 foi mantida para a implementação.



```vhdl
-- ROM module for RISC-V unicore
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

entity ROM_RV32 is
    port (
        address : in std_logic_vector(7 downto 0);
        dataout : out std_logic_vector(31 downto 0)
    );
end entity ROM_RV32;

architecture RTL of ROM_RV32 is

    -- Constants for ROM depth and width
    constant rom_depth : natural := 256;
    constant rom_width : natural := 32;

    -- Define the ROM type
    type rom_type is array (0 to rom_depth - 1) of std_logic_vector(rom_width - 1 downto 0);

    -- Function to initialize ROM from a hex file
    impure function init_rom_from_hex return rom_type is
        file hex_file : text open read_mode is "rom_content_hex.txt"; -- Adjust path as needed
        variable line_v : line;
        variable hex_value : std_logic_vector(31 downto 0);
        variable rom_contents : rom_type;
    begin
        for i in 0 to rom_depth - 1 loop
            if not endfile(hex_file) then
                readline(hex_file, line_v);
                hread(line_v, hex_value);
                rom_contents(i) := hex_value;
            else
                rom_contents(i) := (others => '0'); -- Fill remaining with 0s if file ends
            end if;
        end loop;
        return rom_contents;
    end function;

    -- Initialize ROM with data from file
    signal mem : rom_type := init_rom_from_hex;

begin
    -- Read operation for ROM
    process(address)
    begin
        dataout <= mem(to_integer(unsigned(address)));
    end process;

end RTL;
```

Figura 5: Código VHDL responsável pela descrição da memória ROM

O código é mais extenso quando comparado ao código da memória RAM devido às definições de leitura de arquivo (fornecidas pela biblioteca TEXTIO).

6 Conclusão

O projeto concluiu com sucesso a implementação e simulação das memórias RAM e ROM em VHDL. Com estas memórias, a arquitetura RISC-V uniciclo está mais próxima de simular um processador completo com capacidade de executar instruções.