

Projeto 5 - Simulação de Banco de Registradores em VHDL

Yan Tavares - 202014323

2 de dezembro de 2023

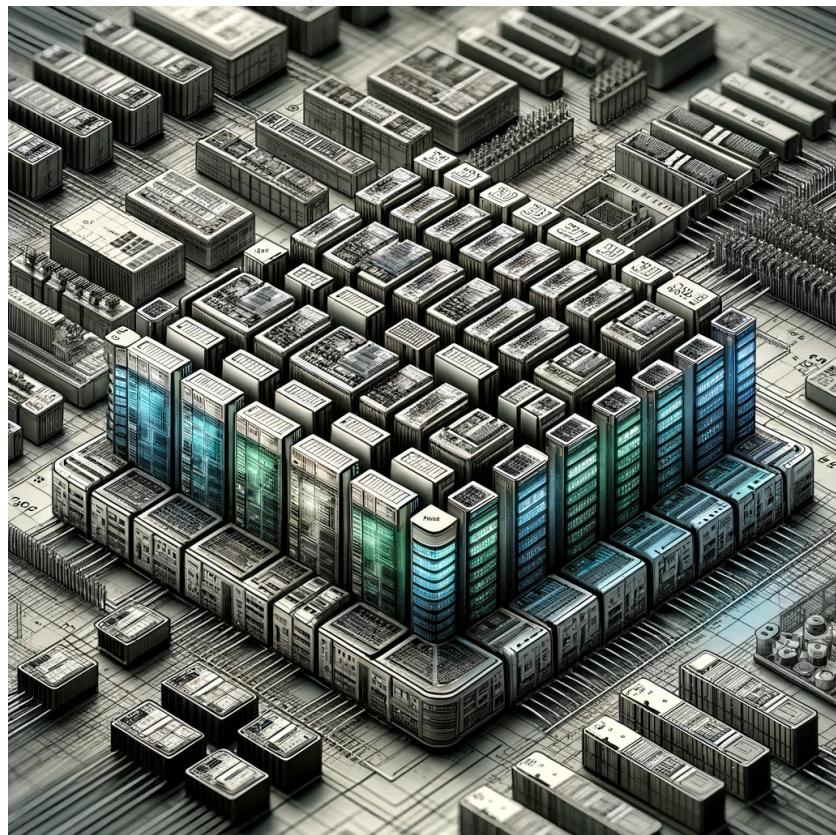


Figura 1: Representação digital de um Banco de Registradores (DALL-E 3)

Conteúdo

1	Introdução	3
2	Características do Banco	3
3	Simulação e Verificação	3
4	Testes e Verificação	4
5	Descrição Detalhada	5
5.1	Opção usada para simular a constante zero em XREGS[0]	5
5.2	Capturas de Tela e Código	5
5.2.1	Ondas geradas (GTKWave)	6
5.2.2	Arquivo XREG.vhdl	7
5.2.3	Arquivo Testbench_XREG.vhdl	8
6	Conclusão	9

1 Introdução

Este relatório apresenta o projeto e a simulação de um Banco de Registradores compatível com a arquitetura RISC-V. O objetivo é demonstrar a implementação e verificação dos registradores em VHDL.

2 Características do Banco

O banco de registradores desenvolvido possui as seguintes características:

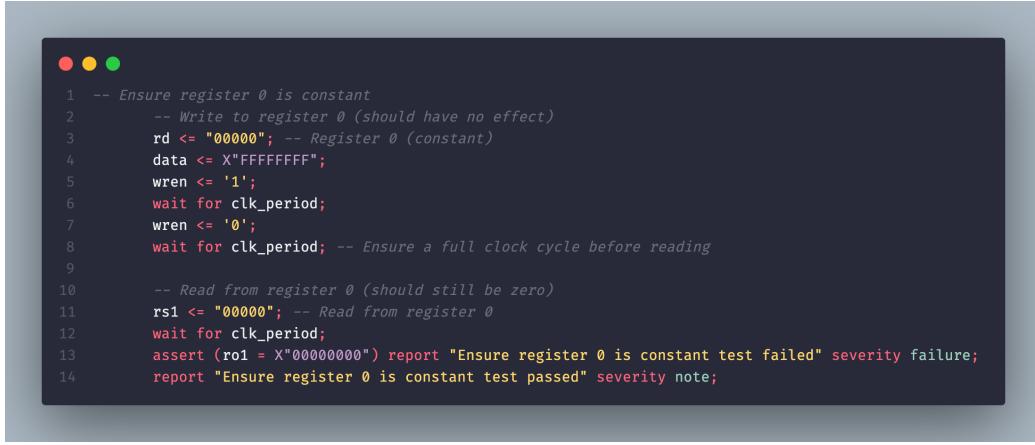
- 32 registradores de 32 bits;
- Dois barramentos de leitura;
- Um barramento para escrita de dado;
- Registrador 0 (índice zero) é constante, não pode ser alterado. Qualquer leitura deste registrador retorna o valor zero e escritas não afetam o seu valor.

3 Simulação e Verificação

A simulação presente neste relatório foi realizada usando o software GHDL e GTKWave para a análise de ondas. Instruções de instalação de ambos os softwares podem ser encontradas no README do projeto. É importante ressaltar que o código também foi testado no EDA Playground e funciona como esperado.

4 Testes e Verificação

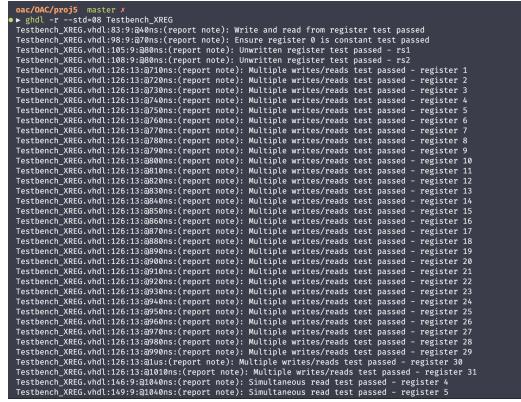
Foram realizados testes para cada registrador, garantindo o correto funcionamento para qualquer registrador. Para isso, foram adicionados asserções no arquivo de testbench para garantir que o resultado é o esperado, além da análise das ondas.



```
1 -- Ensure register 0 is constant
2     -- Write to register 0 (should have no effect)
3     rd <= "00000"; -- Register 0 (constant)
4     data <= X"FFFFFF";
5     wren <= '1';
6     wait for clk_period;
7     wren <= '0';
8     wait for clk_period; -- Ensure a full clock cycle before reading
9
10    -- Read from register 0 (should still be zero)
11    rs1 <= "00000"; -- Read from register 0
12    wait for clk_period;
13    assert (rs1 = "00000000") report "Ensure register 0 is constant test failed" severity failure;
14    report "Ensure register 0 is constant test passed" severity note;
```

Figura 2: Exemplo de testes realizados

Aqui está o report completo do arquivo testbench



```
pac/DAC/proj5 master %
$ ./tb_XREG
Testbench_XREG.vhd:1:83:9:80ns:(report note); Write and read from register test passed
Testbench_XREG.vhd:1:83:9:80ns:(report note); Ensure register 0 is constant test passed
Testbench_XREG.vhd:1:83:9:80ns:(report note); Unwritten register test passed - register 1
Testbench_XREG.vhd:1:83:9:80ns:(report note); Multiple writes/reads test passed - register 1
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 2
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 3
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 4
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 5
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 6
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 7
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 8
Testbench_XREG.vhd:1:26:13:9740ns:(report note); Multiple writes/reads test passed - register 9
Testbench_XREG.vhd:1:26:13:9801ns:(report note); Multiple writes/reads test passed - register 10
Testbench_XREG.vhd:1:26:13:9801ns:(report note); Multiple writes/reads test passed - register 11
Testbench_XREG.vhd:1:26:13:9801ns:(report note); Multiple writes/reads test passed - register 12
Testbench_XREG.vhd:1:26:13:9801ns:(report note); Multiple writes/reads test passed - register 13
Testbench_XREG.vhd:1:26:13:9801ns:(report note); Multiple writes/reads test passed - register 14
Testbench_XREG.vhd:1:26:13:9850ns:(report note); Multiple writes/reads test passed - register 15
Testbench_XREG.vhd:1:26:13:9850ns:(report note); Multiple writes/reads test passed - register 16
Testbench_XREG.vhd:1:26:13:9850ns:(report note); Multiple writes/reads test passed - register 17
Testbench_XREG.vhd:1:26:13:9850ns:(report note); Multiple writes/reads test passed - register 18
Testbench_XREG.vhd:1:26:13:9890ns:(report note); Multiple writes/reads test passed - register 19
Testbench_XREG.vhd:1:26:13:9890ns:(report note); Multiple writes/reads test passed - register 20
Testbench_XREG.vhd:1:26:13:9901ns:(report note); Multiple writes/reads test passed - register 21
Testbench_XREG.vhd:1:26:13:9920ns:(report note); Multiple writes/reads test passed - register 22
Testbench_XREG.vhd:1:26:13:9940ns:(report note); Multiple writes/reads test passed - register 23
Testbench_XREG.vhd:1:26:13:9940ns:(report note); Multiple writes/reads test passed - register 24
Testbench_XREG.vhd:1:26:13:9974ns:(report note); Multiple writes/reads test passed - register 25
Testbench_XREG.vhd:1:26:13:9960ns:(report note); Multiple writes/reads test passed - register 26
Testbench_XREG.vhd:1:26:13:9970ns:(report note); Multiple writes/reads test passed - register 27
Testbench_XREG.vhd:1:26:13:9980ns:(report note); Multiple writes/reads test passed - register 28
Testbench_XREG.vhd:1:26:13:9990ns:(report note); Multiple writes/reads test passed - register 29
Testbench_XREG.vhd:1:26:13:9995ns:(report note); Multiple writes/reads test passed - register 30
Testbench_XREG.vhd:1:146:9:8104ns:(report note); Simultaneous read test passed - register 31
Testbench_XREG.vhd:1:49:9:8104ns:(report note); Simultaneous read test passed - register 5
```

Figura 3: Resultado dos testes no console (Todos os 32 registradores testados)

5 Descrição Detalhada

5.1 Opção usada para simular a constante zero em XREGS[0]

Para simular a constante zero no registrador XREGS[0], foi adicionada uma lógica condicional para que, não importa a operação, o valor contido no registrador 0 será sempre 0.

```
● ● ●  
1 if wren = '1' and to_integer(unsigned(rd)) /= 0 then  
2     registers(to_integer(unsigned(rd))) <= data;  
3 end if;
```

Figura 4: Lógica para XREGS[0]

No código abaixo, primeiro verifica-se se o valor de ‘rd’ é diferente de zero antes de armazenar os dados dentro do registrador, assim garantindo que o registrador na posição 0 jamais será alterado.

5.2 Capturas de Tela e Código

É importante ressaltar que todos os arquivos estão presentes no arquivo .zip incluso no envio do projeto, as capturas de tela são apenas para referência.

5.2.1 Ondas geradas (GTKWave)

Podemos notar os resultados esperados ao visualizar as ondas. Foi utilizado um período de clock de 10 ns para a simulação

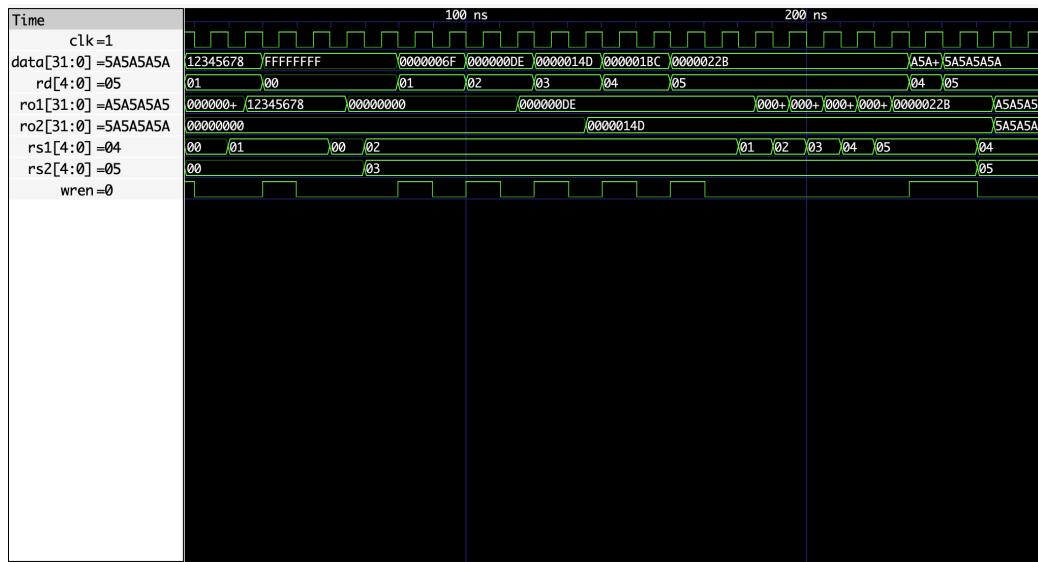


Figura 5: Ondas geradas pelo arquivo de testbench

5.2.2 Arquivo XREG.vhdl

Foram utilizadas extensões '.vhdl' ao invés de '.vhd' para diferenciar arquivos VHDL de Visual Hard Drive (VHD).



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity XREG is
6     Port ( clk      : in  STD_LOGIC;
7             wren    : in  STD_LOGIC;
8             rs1     : in  STD_LOGIC_VECTOR (4 downto 0);
9             rs2     : in  STD_LOGIC_VECTOR (4 downto 0);
10            rd      : in  STD_LOGIC_VECTOR (4 downto 0);
11            data    : in  STD_LOGIC_VECTOR (31 downto 0);
12            ro1    : out STD_LOGIC_VECTOR (31 downto 0);
13            ro2    : out STD_LOGIC_VECTOR (31 downto 0)
14        );
15 end XREG;
16
17 architecture Behavioral of XREG is
18     type reg_array is array (0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
19     signal registers : reg_array := (others => (others => '0'));
20 begin
21     process(clk)
22     begin
23         if rising_edge(clk) then
24             if wren = '1' and to_integer(unsigned(rd)) /= 0 then
25                 registers(to_integer(unsigned(rd))) <= data;
26             end if;
27
28             ro1 <= registers(to_integer(unsigned(rs1)));
29             ro2 <= registers(to_integer(unsigned(rs2)));
30         end if;
31     end process;
32 end Behavioral;
33
```

Figura 6: Captura de tela do código principal

5.2.3 Arquivo Testbench_XREG.vhd

Foram utilizados laços de repetição para o teste de cada um dos 32 registradores (incluindo o Zero).

Figura 7: Captura de tela do código de testbench do projeto

6 Conclusão

Em conclusão, conseguimos implementar um banco de registradores com sucesso em VHDL. Com a ULA, o Gerador de Imediatos e o Banco de Registradores prontos, estamos cada vez mais perto de simular o funcionamento de um processador que execute instruções RiscV.