# Trabalho: Projeto do Banco de Registradores do RISC-V

Aluno: Adriano Ulrich do Prado Wiedmann
Matrícula: 202014824

## Objetivo:

Projetar, simular e sintetizar um Banco de Registradores do RISC-V de 32 bits utilizando o *ModelSim*.

## Versão ModelSim:

O trabalho foi realizado utilizando o *ModelSim* na versão 20.1.1.

## Código:

Este trabalho consiste em dois arquivos *vhd*. O arquivo *xregs.vhd* e *tb_xregs.vhd* (***testbench***).

### *1. Arquivo xregs.vhd*

É definido uma entidade chamada "*XREGS*". Nela é criada um parâmetro genérico chamado *WSIZE* de tamanho 32.

As portas da entidade definidas são:

- **clk** – Entrada de dados que representa o clock;
- **wren** – Entrada de dados que representa a habilitação de escrita;
- **rs1** – Entrada de 5 bits que representa o endereço do registrador a ser lido em ro1;
- **rs2** – Entrada de 5 bits que representa o endereço do registrador a ser lido em ro2;
- **rd** – Entrada de 5 bits que representa o endereço do registrador para armazenar o conteúdo de data;
- **data** – Entrada de 32 bits que possui o valor a ser escrito no registrador endereçado por rd;
- **ro1** – Saída de 32 bits para leitura do registrador endereçado por rs1;
- **ro2** – Saída de 32 bits para leitura do registrador endereçado por rs2;

O código disponibilizado na Figura 1 implementa o Banco de Registradores.

**Clock:** O sinal de *clock* (*clk*) é utilizado para sincronizar as operações de escrita e leitura.

**Banco de Registradores:** Foi definido um tipo *reg_array* que é um *array* de registradores, onde cada registrador é um vetor de 32 bits. Um sinal chamado *regs* é declarado usando esse tipo e é inicializado com registradores zerados.

**Para escrita no Banco de Registradores:** Se o *wren* estiver habilitado (*wren* = '1') e o registrador de destino (*rd*) não for o x0, então o dado de entrada (*data*) é gravado no registrador correspondente determinado pelo valor de *rd*.

**Para leitura do Banco de Registradores:** Os dados dos registradores correspondentes aos índices especificados pelos sinais *rs1* e *rs2* são lidos e fornecidos nas portas de saída *ro1* e *ro2*, respectivamente.

```
1   -- Adriano Ulrich do Prado Wiedmann 202014824
2   library IEEE;
3   use IEEE.std_logic_1164.all;
4   use IEEE.numeric_std.all;
5
6   -- wren -> habilitação de escrita.
7
8   entity XREGS is
9           generic (WSIZE : natural := 32);
10          port (
11                  clk, wren       : in std_logic;
12                  rs1, rs2, rd    : in std_logic_vector(4 downto 0);
13                  data            : in std_logic_vector(WSIZE-1 downto 0);
14                  ro1, ro2        : out std_logic_vector(WSIZE-1 downto 0)
15          );
16  end XREGS;
17
18  architecture behavioral of XREGS is
19          -- array de tamanho 32 e cada posição tem 32 bits
20          type reg_array is array (0 to WSIZE-1) of std_logic_vector(WSIZE-1 downto 0);
21          signal regs : reg_array := (others => "00000000000000000000000000000000");
22
23  begin
24          process (clk)
25          begin
26                  if rising_edge(clk) then
27                          -- ler os registradores cujo índice é determinado pelo conteúdo de rs1 e rs2
28                          ro1 <= regs(to_integer(unsigned(rs1)));
29                          ro2 <= regs(to_integer(unsigned(rs2)));
30                          -- Escreve no registrador especificado se wren está habilitado e se não for o registrador x0
31                          if wren = '1' and rd /= "00000" then
32                                  regs(to_integer(unsigned(rd))) <= data;
33                          end if;
34                  end if;
35          end process;
36
37  end behavioral;
38
```

Figura 1. Implementação do Banco de Registradores.

## 2. *Arquivo tb_xregs.vhd*

No *testbench* é realizado a associação (*port mapping*) dos sinais entre a unidade de teste (*tb_xregs*) e a unidade de design (*xregs*) da seguinte forma:

```
uut: xregs port map(
        clk => clk, wren => wren,
        rs1 => rs1, rs2 => rs2, rd => rd,
        data => data,
        ro1 => ro1, ro2 => ro2
);
```

No *testbench*, primeiro é verificado o registrador 0. Para isso, *wren* é habilitado, rd indica o registrador 0 e é atribuído um valor para data. Ao simular é possível notar que nenhum valor é foi armazenado em x0.

Para verificar os registradores é utilizado um loop for que vai de 1 até 31. Neste loop é armazenado valores em cada registrador e verificado ro1 e ro2 se os valores estão corretos.

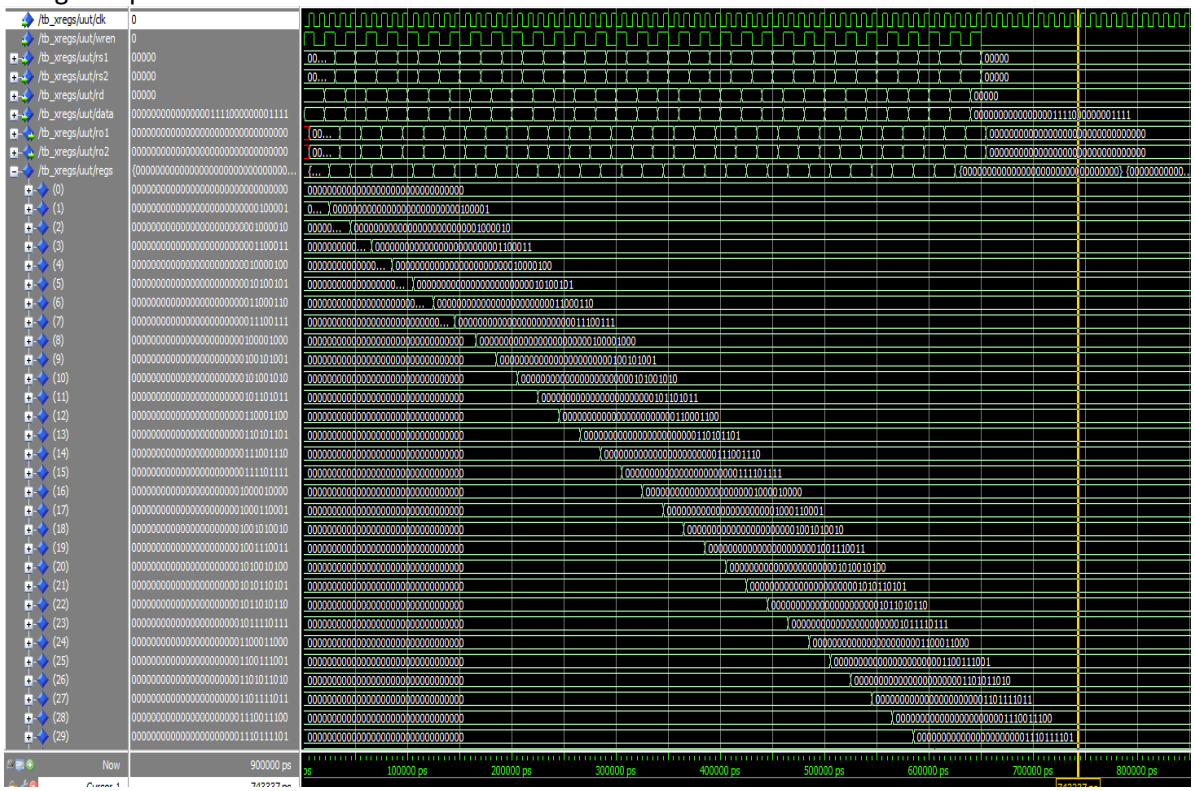
Por último, é feito novamente a verificação do registrador x0. Da mesma forma que foi realizado na primeira vez.

O código para verificação pode ser visualizado na Figura 2.

```vhdl
-- Adriano Ulrich do Prado Wiedmann 202014824

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

entity tb_xregs is
end tb_xregs;

architecture tb of tb_xregs is
component xregs is
    generic (WSIZE : natural := 32);
    port(
                clk, wren       : in std_logic;
                rs1, rs2, rd    : in std_logic_vector(4 downto 0);
                data            : in std_logic_vector(WSIZE-1 downto 0);
                ro1, ro2        : out std_logic_vector(WSIZE-1 downto 0)
                );
end component;

signal clk, wren : std_logic := '0';
signal rs1 : std_logic_vector(4 downto 0) := (others => '0');
signal rs2 : std_logic_vector(4 downto 0) := (others => '0');
signal rd : std_logic_vector(4 downto 0) := (others => '0');
signal data : std_logic_vector(31 downto 0) := (others => '0');
signal ro1, ro2 : std_logic_vector(31 downto 0);

begin
        uut: xregs port map(
                clk => clk, wren => wren,
                rs1 => rs1, rs2 => rs2, rd => rd,
                data => data,
                ro1 => ro1, ro2 => ro2
        );

        clk <= not clk after 5 ns;

    stimulus: process
    begin

            -- Verifica o registrador 0
            wren <= '1';
            rd <= "00000";
            data <= X"0000000F";
            wait for 10 ns;

            wren <= '0';
            wait for 10 ns;

            assert wren = '0' and ro1 = X"00000000" report "Register 0 modified" severity error;
            assert wren = '0' and ro2 = X"00000000" report "Register 0 modified" severity error;

            -- Verifica outros registradores
            for i in 1 to 31 loop
                wren <= '1';
                rd <= std_logic_vector(to_unsigned(i, 5));
                data <= std_logic_vector(to_unsigned((i+(i*32)),32));
                wait for 10 ns;

                wren <= '0';
                rs1 <= std_logic_vector(to_unsigned(i, 5));
                rs2 <= std_logic_vector(to_unsigned(i, 5));
                wait for 10 ns;

                assert ro1 = std_logic_vector(to_unsigned((i+(i*32)), 32)) report "Register Error" severity error;
                assert ro2 = std_logic_vector(to_unsigned((i+(i*32)), 32)) report "Register Error" severity error;
            end loop;

            -- Segunda verificação para o registrador 0
            wren <= '1';
            rd <= "00000";
            data <= X"0000F00F";
            wait for 10 ns;

            wren <= '0';
            rs1 <= "00000";
            rs2 <= "00000";
            wait for 10 ns;

            assert ro1 = X"00000000" report "Register 0 modified" severity error;
            assert ro2 = X"00000000" report "Register 0 modified" severity error;

            wait;

    end process;
```

Figura 2. Código para verificação.

A seguir é possível verificar a *Wave:*




O Cursor está posicionado aonde foi na última verificação. O caso em que é verificado o registrador 0.

Nenhuma mensagem de erro é reportada dos casos de testes realizados utilizando o ASSERT: