Relatório do Quinto Trabalho de OAC

BREG MIPS

Gabriel Araujo - 12/0050943
09/06/2018

Descrição do Problema

Registradores são um dos vários tipos de memória disponível no computador e estão no topo da hierarquia de memória. Registrador é tipo de circuito digital constituído por um conjunto sequencial, arranjo, de n *flip-flops* usado para armazenamento de dados temporário. A Unidade Central de Processamento, *CPU*, de um computador tem disponível um conjunto de registradores, o Banco de Registradores, para operações aritméticas e manipulação de dados feito pelas instruções.

Descrição da Implementação

Interface do Componente

Para a declaração do componente foi utilizado a mesma interface especificada no trabalho, utilizando o parâmetro *WSIZE*, como tamanho da palavra, configurado como 32 *bits*. O Banco de Registradores descrito neste trabalho é Banco do *MIPS* 32, onde o Banco contém 32 registradores de 32 *bits*. O Banco tem como sinais de entrada:

- *clk*, *rst* e *wren* para o relógio, *reset* e habilitação de escrita para os registradores;
- *radd1* e *radd2*: endereços dos registradores a serem lidos na saída;
- *wadd*: endereço do registrador que será escrito com o valor de *wdata*;
- *wdata*: valor a ser escrito no registrador endereçado por *wadd*.

E o Banco tem como sinais de saída:

- *r1* e *r2*: porta de saída para leitura do registrador endereçado por *radd1* e *radd2*, respectivamente.

Comportamento do Componente

O comportamento do componente foi descrito como uma seleção entre os vários registradores do banco para as saídas *r1* e *r2*. Quando há uma subida de relógio e o sinal de escrita está habilitado o registrador endereçado pela entrada *wadd* é escrito com o valor da entrada *wdata*.

Código VHDL do BREG

O código VHDL do Banco de Registradores está no Anexo 1 e o código VHDL do *Test Bench* está no Anexo 2. O projeto completo do Banco de Registradores foi enviado junto com este relatório.

Testes e Resultados

Para esse trabalho, o Banco de Registradores (*BREG*) foi testada usando um *Test Bench* que configura o *BREG* para escrever uma sequência de valores 1, 2, 3, ..., 31 nos registradores. Após a escrita, ler em *r1* os registradores de 1 ao 10 e em *r2* os registradores de 11 ao 20 e então ler simultaneamente em *r1* e *r2* os registradores de 21 ao 31.

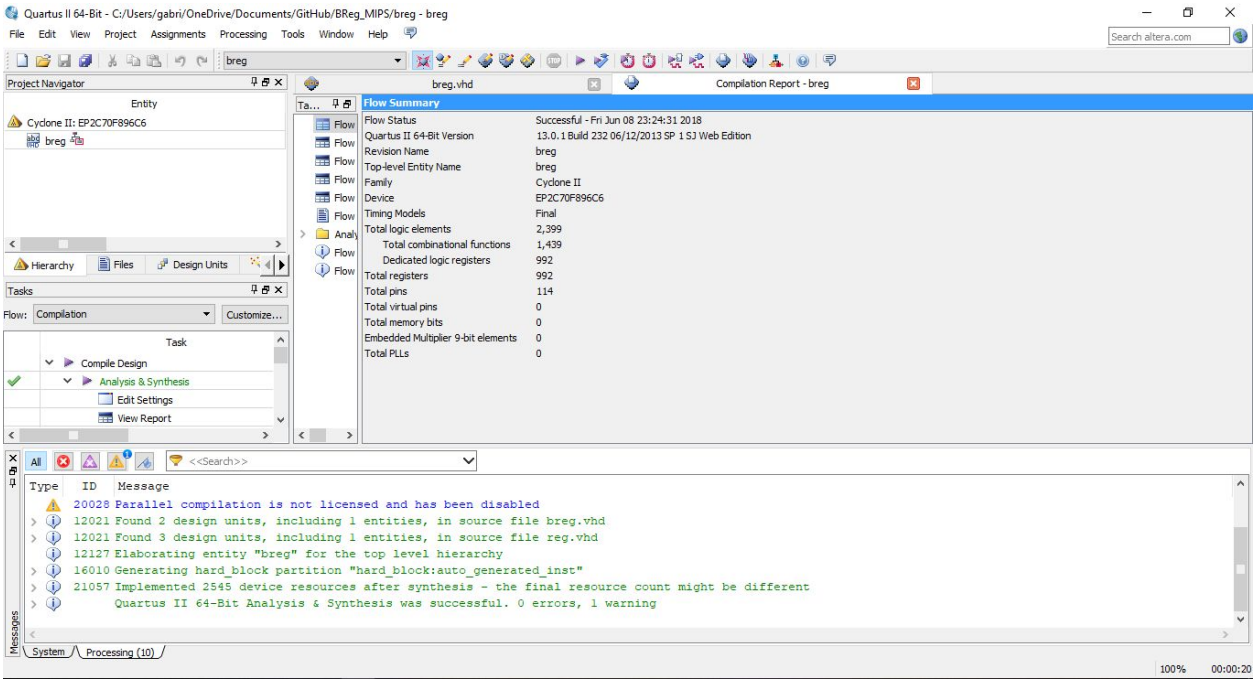

Figura 1

A Figura 1 mostra o resultado da síntese do componente no Quartus II. E as demais figuras mostram o resultado da simulação. Todos os resultados da simulação condizem com a implementação e o esperado do circuito real.

Nota

- O último teste é o teste de escrita e leitura simultâneo do mesmo registrador, nesse teste a leitura e a escrita são iniciadas na mesma subida do relógio. O valor lido na saída da memória é o valor anterior do registrador, após a

segunda subida do relógio o novo valor, que foi escrito, aparece na saída da memória. Resultado coerente com a teoria.

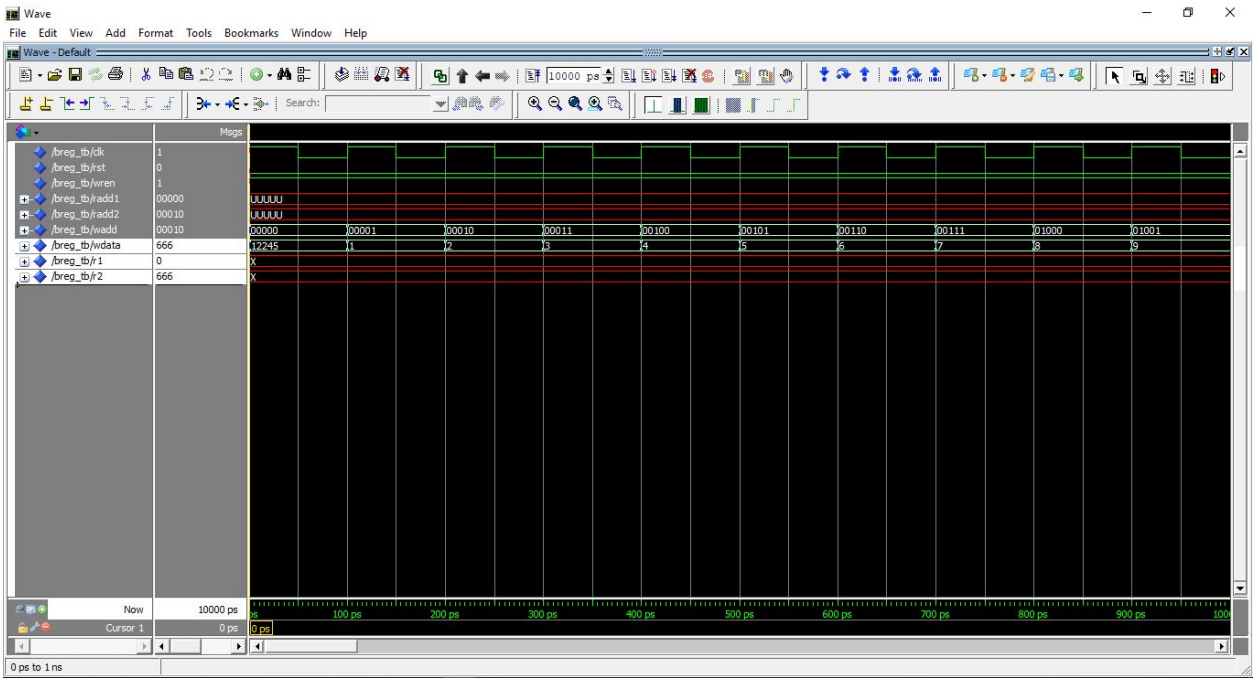

Figura 2

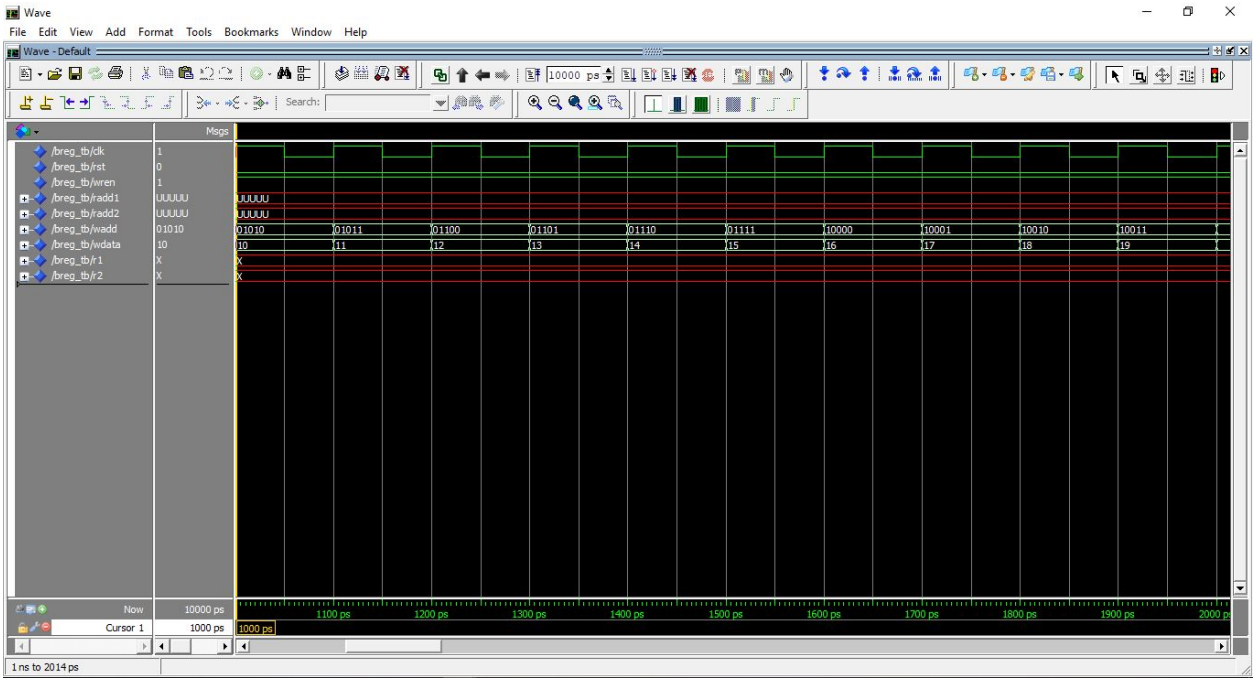

Figura 3

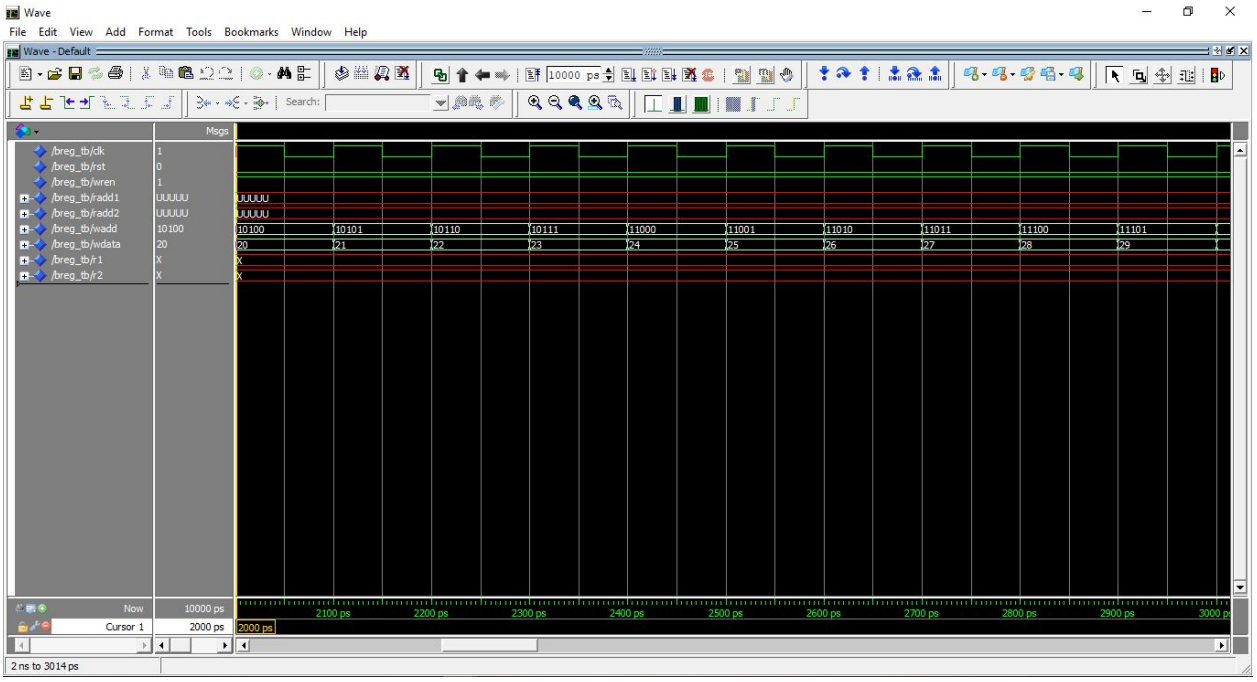

Figura 4

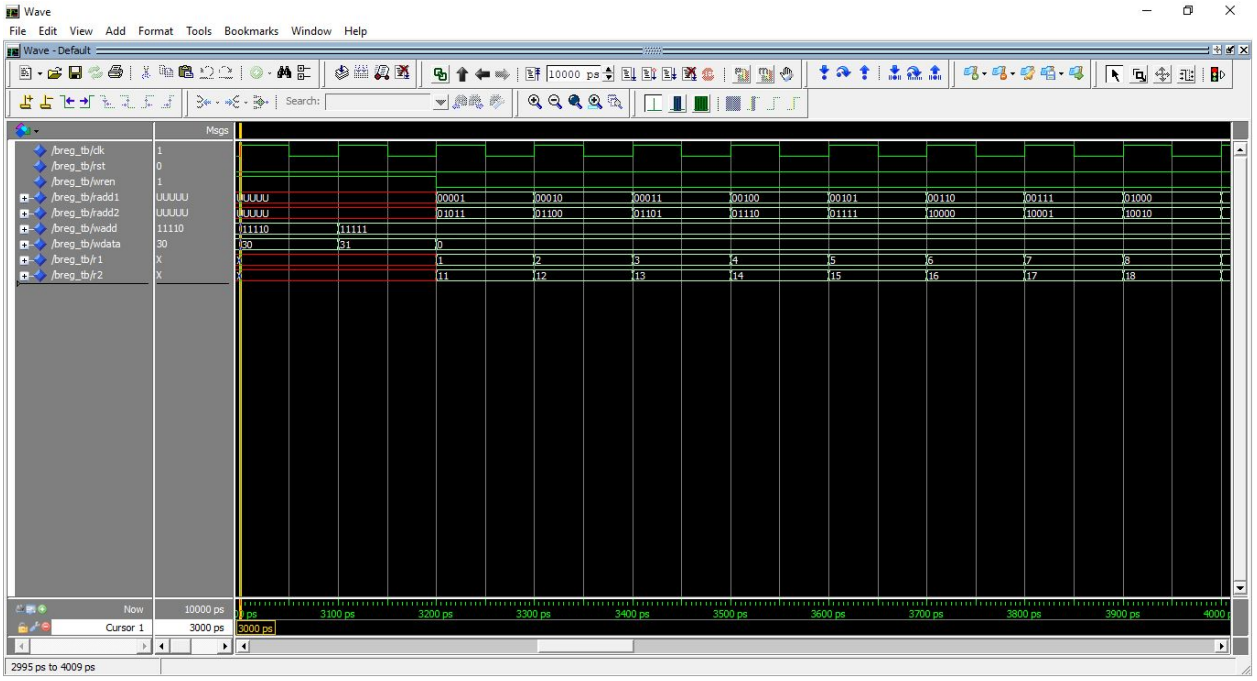

Figura 5

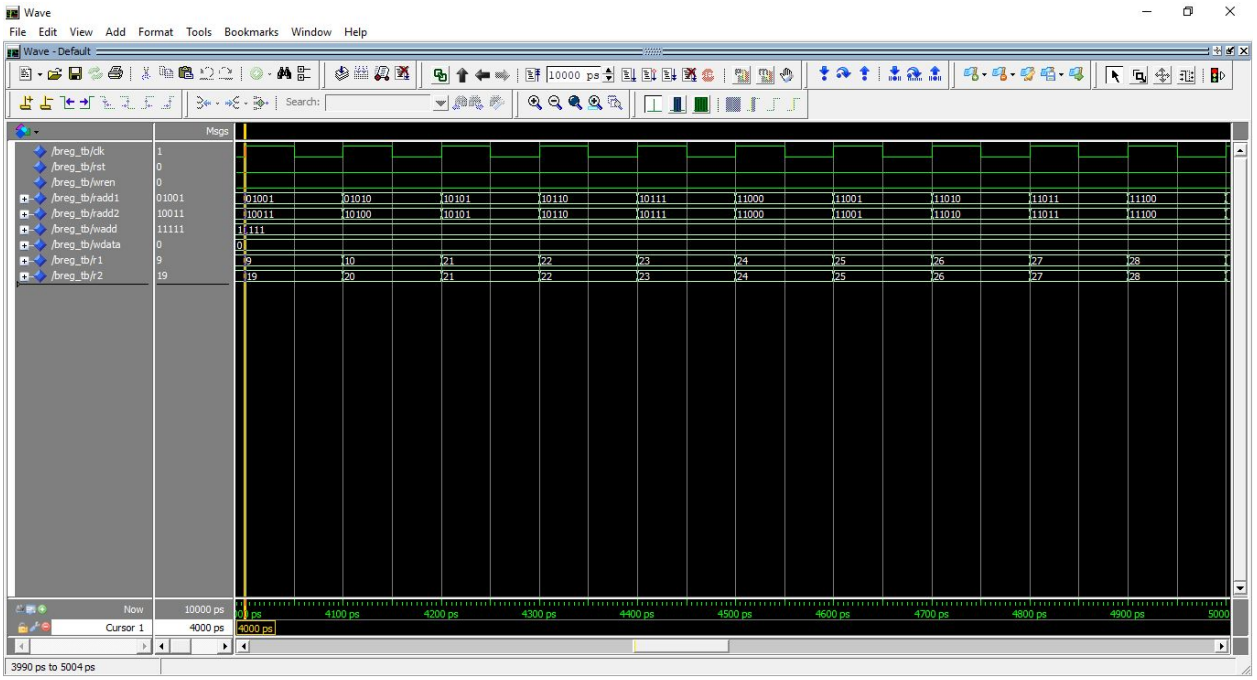

Figura 6

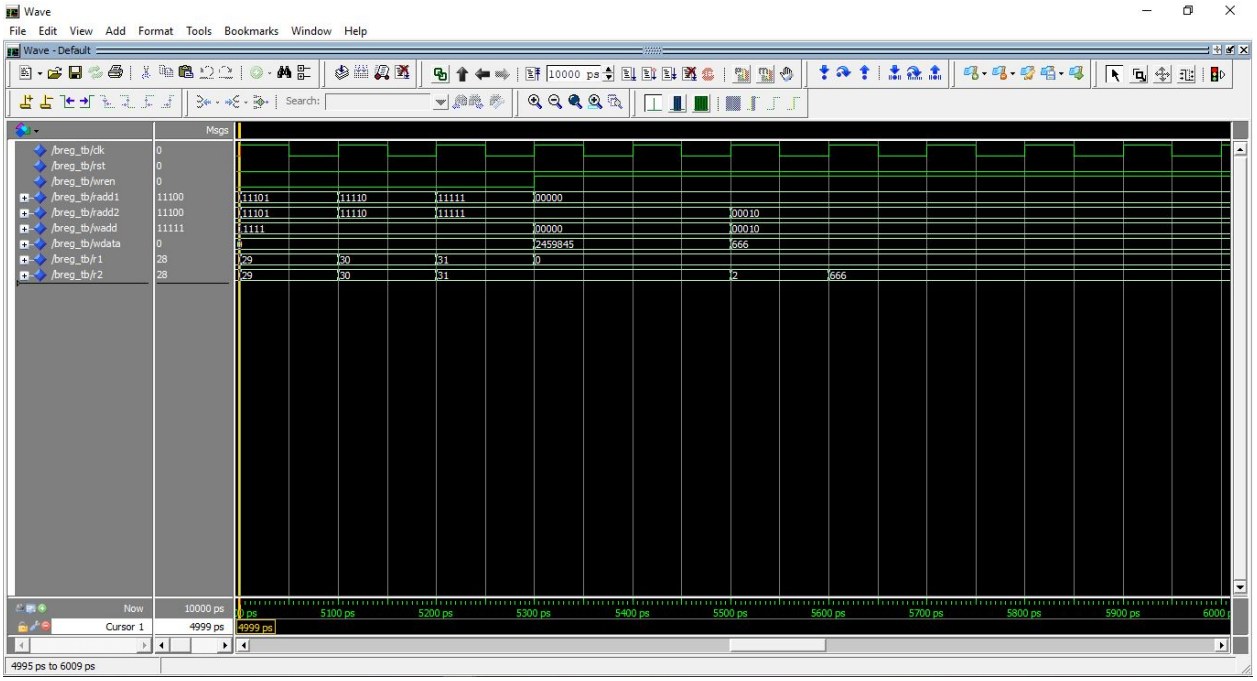

Figura 7

Anexo 1

June 9, 2018

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use work.reg_p.all;

entity breg is
    --generic (WSIZE : natural := 32);
  port (
        clk, wren, rst: in std_logic;
        radd1, radd2: in std_logic_vector(4 downto 0);
        wadd: in std_logic_vector(4 downto 0);
        wdata: in std_logic_vector(31 downto 0);
        r1, r2: out std_logic_vector(31 downto 0)
  );
end entity ; -- breg

architecture arch of breg is
    type registers is array (1 to 31) of
    std_logic_vector(31 downto 0);
    signal zero, out1: std_logic_vector(31 downto 0);
    signal out2: std_logic_vector(31 downto 0);
    signal regs: registers;
begin
    r1 <= out1;
    r2 <= out2;
    breg_proc :
    process( clk, wren, rst, radd1, radd2, out1, out2 )
    begin
        zero <= X"00000000";
        -- write registers
        if rising_edge(clk) then
            if wren = '1' then
                case( wadd ) is
                    when "00000" => -- 1
                        zero <= X"00000000";
                    when "00001" => -- 1
                        regs(1) <= (wdata);
                    when "00010" => -- 2
                        regs(2) <= (wdata);
                    when "00011" => -- 3
```

```
        regs(3) <= (wdata);
    when "00100" => -- 4
        regs(4) <= (wdata);
    when "00101" => -- 5
        regs(5) <= (wdata);
    when "00110" => -- 6
        regs(6) <= (wdata);
    when "00111" => -- 7
        regs(7) <= (wdata);
    when "01000" => -- 8
        regs(8) <= (wdata);
    when "01001" => -- 9
        regs(9) <= (wdata);
    when "01010" => -- 10
        regs(10) <= (wdata);
    when "01011" => -- 11
        regs(11) <= (wdata);
    when "01100" => -- 12
        regs(12) <= (wdata);
    when "01101" => -- 13
        regs(13) <= (wdata);
    when "01110" => -- 14
        regs(14) <= (wdata);
    when "01111" => -- 15
        regs(15) <= (wdata);
    when "10000" => -- 16
        regs(16) <= (wdata);
    when "10001" => -- 17
        regs(17) <= (wdata);
    when "10010" => -- 18
        regs(18) <= (wdata);
    when "10011" => -- 19
        regs(19) <= (wdata);
    when "10100" => -- 20
        regs(20) <= (wdata);
    when "10101" => -- 21
        regs(21) <= (wdata);
    when "10110" => -- 22
        regs(22) <= (wdata);
    when "10111" => -- 23
        regs(23) <= (wdata);
    when "11000" => -- 24
        regs(24) <= (wdata);
    when "11001" => -- 25
        regs(25) <= (wdata);
    when "11010" => -- 26
        regs(26) <= (wdata);
    when "11011" => -- 27
        regs(27) <= (wdata);
    when "11100" => -- 28
```

```
                regs(28) <= (wdata);
            when "11101" => -- 29
                regs(29) <= (wdata);
            when "11110" => -- 30
                regs(30) <= (wdata);
            when "11111" => -- 31
                regs(31) <= (wdata);
            when others =>
                zero <= zero;
        end case ;
    end if;
    -- read registers
    if rst = '1' then
        regs(1)  <= X"00000000";
        regs(2)  <= X"00000000";
        regs(3)  <= X"00000000";
        regs(4)  <= X"00000000";
        regs(5)  <= X"00000000";
        regs(6)  <= X"00000000";
        regs(7)  <= X"00000000";
        regs(8)  <= X"00000000";
        regs(9)  <= X"00000000";
        regs(10) <= X"00000000";
        regs(11) <= X"00000000";
        regs(12) <= X"00000000";
        regs(13) <= X"00000000";
        regs(14) <= X"00000000";
        regs(15) <= X"00000000";
        regs(16) <= X"00000000";
        regs(17) <= X"00000000";
        regs(18) <= X"00000000";
        regs(19) <= X"00000000";
        regs(20) <= X"00000000";
        regs(21) <= X"00000000";
        regs(22) <= X"00000000";
        regs(23) <= X"00000000";
        regs(24) <= X"00000000";
        regs(25) <= X"00000000";
        regs(26) <= X"00000000";
        regs(27) <= X"00000000";
        regs(28) <= X"00000000";
        regs(29) <= X"00000000";
        regs(30) <= X"00000000";
        regs(31) <= X"00000000";
    end if ;
end if;
case( radd1 ) is
    when "00000" => -- 0
        out1 <= zero;
    when "00001" => -- 1
```

```
        out1 <= std_logic_vector(regs(1));
    when "00010" => -- 2
        out1 <= std_logic_vector(regs(2));
    when "00011" => -- 3
        out1 <= std_logic_vector(regs(3));
    when "00100" => -- 4
        out1 <= std_logic_vector(regs(4));
    when "00101" => -- 5
        out1 <= std_logic_vector(regs(5));
    when "00110" => -- 6
        out1 <= std_logic_vector(regs(6));
    when "00111" => -- 7
        out1 <= std_logic_vector(regs(7));
    when "01000" => -- 8
        out1 <= std_logic_vector(regs(8));
    when "01001" => -- 9
        out1 <= std_logic_vector(regs(9));
    when "01010" => -- 10
        out1 <= std_logic_vector(regs(10));
    when "01011" => -- 11
        out1 <= std_logic_vector(regs(11));
    when "01100" => -- 12
        out1 <= std_logic_vector(regs(12));
    when "01101" => -- 13
        out1 <= std_logic_vector(regs(13));
    when "01110" => -- 14
        out1 <= std_logic_vector(regs(14));
    when "01111" => -- 15
        out1 <= std_logic_vector(regs(15));
    when "10000" => -- 16
        out1 <= std_logic_vector(regs(16));
    when "10001" => -- 17
        out1 <= std_logic_vector(regs(17));
    when "10010" => -- 18
        out1 <= std_logic_vector(regs(18));
    when "10011" => -- 19
        out1 <= std_logic_vector(regs(19));
    when "10100" => -- 20
        out1 <= std_logic_vector(regs(20));
    when "10101" => -- 21
        out1 <= std_logic_vector(regs(21));
    when "10110" => -- 22
        out1 <= std_logic_vector(regs(22));
    when "10111" => -- 23
        out1 <= std_logic_vector(regs(23));
    when "11000" => -- 24
        out1 <= std_logic_vector(regs(24));
    when "11001" => -- 25
        out1 <= std_logic_vector(regs(25));
    when "11010" => -- 26
```

```
            out1 <= std_logic_vector(regs(26));
        when "11011" => -- 27
            out1 <= std_logic_vector(regs(27));
        when "11100" => -- 28
            out1 <= std_logic_vector(regs(28));
        when "11101" => -- 29
            out1 <= std_logic_vector(regs(29));
        when "11110" => -- 30
            out1 <= std_logic_vector(regs(30));
        when "11111" => -- 31
            out1 <= std_logic_vector(regs(31));
        when others =>
            out1 <= out1;
    end case ;
    case( radd2 ) is
        when "00000" => -- 0
            out2 <= zero;
        when "00001" => -- 1
            out2 <= std_logic_vector(regs(1));
        when "00010" => -- 2
            out2 <= std_logic_vector(regs(2));
        when "00011" => -- 3
            out2 <= std_logic_vector(regs(3));
        when "00100" => -- 4
            out2 <= std_logic_vector(regs(4));
        when "00101" => -- 5
            out2 <= std_logic_vector(regs(5));
        when "00110" => -- 6
            out2 <= std_logic_vector(regs(6));
        when "00111" => -- 7
            out2 <= std_logic_vector(regs(7));
        when "01000" => -- 8
            out2 <= std_logic_vector(regs(8));
        when "01001" => -- 9
            out2 <= std_logic_vector(regs(9));
        when "01010" => -- 10
            out2 <= std_logic_vector(regs(10));
        when "01011" => -- 11
            out2 <= std_logic_vector(regs(11));
        when "01100" => -- 12
            out2 <= std_logic_vector(regs(12));
        when "01101" => -- 13
            out2 <= std_logic_vector(regs(13));
        when "01110" => -- 14
            out2 <= std_logic_vector(regs(14));
        when "01111" => -- 15
            out2 <= std_logic_vector(regs(15));
        when "10000" => -- 16
            out2 <= std_logic_vector(regs(16));
        when "10001" => -- 17
```

```
                out2 <= std_logic_vector(regs(17));
            when "10010" => -- 18
                out2 <= std_logic_vector(regs(18));
            when "10011" => -- 19
                out2 <= std_logic_vector(regs(19));
            when "10100" => -- 20
                out2 <= std_logic_vector(regs(20));
            when "10101" => -- 21
                out2 <= std_logic_vector(regs(21));
            when "10110" => -- 22
                out2 <= std_logic_vector(regs(22));
            when "10111" => -- 23
                out2 <= std_logic_vector(regs(23));
            when "11000" => -- 24
                out2 <= std_logic_vector(regs(24));
            when "11001" => -- 25
                out2 <= std_logic_vector(regs(25));
            when "11010" => -- 26
                out2 <= std_logic_vector(regs(26));
            when "11011" => -- 27
                out2 <= std_logic_vector(regs(27));
            when "11100" => -- 28
                out2 <= std_logic_vector(regs(28));
            when "11101" => -- 29
                out2 <= std_logic_vector(regs(29));
            when "11110" => -- 30
                out2 <= std_logic_vector(regs(30));
            when "11111" => -- 31
                out2 <= std_logic_vector(regs(31));
            when others =>
                out2 <= out2;
        end case ;
    end process ; -- breg_proc
end architecture ; -- arch
```

Anexo 2

June 9, 2018

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

entity breg_tb is
end breg_tb;
architecture breg_arch of breg_tb is
-- constants
-- signals
signal clk : std_logic;
signal rst : std_logic;
signal wren : std_logic;

signal radd1 : std_logic_vector(4 downto 0);
signal radd2 : std_logic_vector(4 downto 0);
signal wadd : std_logic_vector(4 downto 0);

signal wdata : std_logic_vector(31 downto 0);
signal r1 : std_logic_vector(31 downto 0);
signal r2 : std_logic_vector(31 downto 0);

component breg
    port (
    clk : in std_logic;
    rst : in std_logic;
    wren : in std_logic;
    radd1 : in std_logic_vector(4 downto 0);
    radd2 : in std_logic_vector(4 downto 0);
    wadd : in std_logic_vector(4 downto 0);
    wdata : in std_logic_vector(31 downto 0);
    r1 : out std_logic_vector(31 downto 0);
    r2 : out std_logic_vector(31 downto 0)
    );
end component;
begin
    i1 : breg
    port map (
-- list connections between master ports and signals
    clk => clk,
```

```
    r1 => r1,
    r2 => r2,
    radd1 => radd1,
    radd2 => radd2,
    rst => rst,
    wadd => wadd,
    wdata => wdata,
    wren => wren
    );
init : process
-- variable declarations
    constant half_period : time := 100 ns;
begin
    -- writing in register zero
    rst <= '0';
     -- 0
    wadd <= "00000";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(12245, 32)));
    wait for 100 ps;
     -- 1
    wadd <= "00001";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(1, 32)));
    wait for 100 ps;
     -- 2
    wadd <= "00010";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(2, 32)));
    wait for 100 ps;
     -- 3
    wadd <= "00011";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(3, 32)));
    wait for 100 ps;
     -- 4
    wadd <= "00100";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(4, 32)));
    wait for 100 ps;
     -- 5
    wadd <= "00101";
    wren <= '1';
    wdata <=  (std_logic_vector(to_unsigned(5, 32)));
    wait for 100 ps;
     -- 6
    wadd <= "00110";
    wren <= '1';
    wdata <=  (std_logic_vector(to_unsigned(6, 32)));
    wait for 100 ps;
```

```
 -- 7
wadd <= "00111";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(7, 32)));
wait for 100 ps;
 -- 8
wadd <= "01000";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(8, 32)));
wait for 100 ps;
 -- 9
wadd <= "01001";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(9, 32)));
wait for 100 ps;
 -- 10
wadd <= "01010";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(10, 32)));
wait for 100 ps;
 -- 11
wadd <= "01011";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(11, 32)));
wait for 100 ps;
 -- 12
wadd <= "01100";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(12, 32)));
wait for 100 ps;
 -- 13
wadd <= "01101";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(13, 32)));
wait for 100 ps;
 -- 14
wadd <= "01110";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(14, 32)));
wait for 100 ps;
 -- 15
wadd <= "01111";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(15, 32)));
wait for 100 ps;
 -- 16
wadd <= "10000";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(16, 32)));
wait for 100 ps;
```

```
 -- 17
wadd <= "10001";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(17, 32)));
wait for 100 ps;
 -- 18
wadd <= "10010";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(18, 32)));
wait for 100 ps;
 -- 19
wadd <= "10011";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(19, 32)));
wait for 100 ps;
 -- 20
wadd <= "10100";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(20, 32)));
wait for 100 ps;
 -- 21
wadd <= "10101";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(21, 32)));
wait for 100 ps;
 -- 22
wadd <= "10110";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(22, 32)));
wait for 100 ps;
 -- 23
wadd <= "10111";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(23, 32)));
wait for 100 ps;
 -- 24
wadd <= "11000";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(24, 32)));
wait for 100 ps;
 -- 25
wadd <= "11001";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(25, 32)));
wait for 100 ps;
 -- 26
wadd <= "11010";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(26, 32)));
wait for 100 ps;
```

```
 -- 27
wadd <= "11011";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(27, 32)));
wait for 100 ps;
 -- 28
wadd <= "11100";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(28, 32)));
wait for 100 ps;
 -- 29
wadd <= "11101";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(29, 32)));
wait for 100 ps;
 -- 30
wadd <= "11110";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(30, 32)));
wait for 100 ps;
 -- 31
wadd <= "11111";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(31, 32)));
wait for 100 ps;
-- read registers
wren <= '0';
wdata <= (std_logic_vector(to_unsigned(0, 32)));
-- 1 e 11
radd1 <= "00001"; radd2 <= "01011";
wait for 100 ps;
-- 2 e 12
radd1 <= "00010"; radd2 <= "01100";
wait for 100 ps;
-- 3 e 13
radd1 <= "00011"; radd2 <= "01101";
wait for 100 ps;
-- 4 e 14
radd1 <= "00100"; radd2 <= "01110";
wait for 100 ps;
-- 5 e 15
radd1 <= "00101"; radd2 <= "01111";
wait for 100 ps;
-- 6 e 16
radd1 <= "00110"; radd2 <= "10000";
wait for 100 ps;
-- 7 e 17
radd1 <= "00111"; radd2 <= "10001";
wait for 100 ps;
-- 8 e 18
```

```
radd1 <= "01000"; radd2 <= "10010";
wait for 100 ps;
-- 9 e 19
radd1 <= "01001"; radd2 <= "10011";
wait for 100 ps;
-- 10 e 20
radd1 <= "01010"; radd2 <= "10100";
wait for 100 ps;

-- 21
radd1 <= "10101"; radd2 <= "10101";
wait for 100 ps;
-- 22
radd1 <= "10110"; radd2 <= "10110";
wait for 100 ps;
-- 23
radd1 <= "10111"; radd2 <= "10111";
wait for 100 ps;
-- 24
radd1 <= "11000"; radd2 <= "11000";
wait for 100 ps;
-- 25
radd1 <= "11001"; radd2 <= "11001";
wait for 100 ps;
-- 26
radd1 <= "11010"; radd2 <= "11010";
wait for 100 ps;
-- 27
radd1 <= "11011"; radd2 <= "11011";
wait for 100 ps;
-- 28
radd1 <= "11100"; radd2 <= "11100";
wait for 100 ps;
-- 29
radd1 <= "11101"; radd2 <= "11101";
wait for 100 ps;
-- 30
radd1 <= "11110"; radd2 <= "11110";
wait for 100 ps;
-- 31
radd1 <= "11111"; radd2 <= "11111";
wait for 100 ps;
-- trying to write in register 'zero'
wadd <= "00000";
radd1 <= "00000";
wren <= '1';
wdata <= (std_logic_vector(to_unsigned(2459845, 32)));
wait for 200 ps;
-- trying to read and write in the same register
radd2 <= "00010";
```

```
    wadd <= "00010";
    wren <= '1';
    wdata <= (std_logic_vector(to_unsigned(666, 32)));
    wait for 200 ps;
wait;
end process init;
end breg_arch;
```