UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DISCIPLINA: SISTEMAS DIGITAIS
Prof. Fernanda Lima Kastensmidt

**2022-1**

**Trabalho 1 – Sistemas Digitais - Individual**

# Projeto do Processador Neander em VHDL

**Aluno: Matheus Almeida Silva - 00316326**

O computador NEANDER foi criado com intenções didáticas pelo prof. Raul Weber da UFRGS. Neste site há referencias e link para o simulador: http://www.dcc.ufrj.br/~gabriel/neander.php

**O objetivo deste trabalho de SD é implementar o NEANDER usando a linguagem de descrição de hardware VHDL, simular esse circuito em um simulador lógico sem atraso, depois realizar a síntese lógica, mapeamento tecnológico, posicionamento e roteamento para um FPGA, realizar a simulação com atraso e prototipar o processador em uma placa de prototipação.**

1) Deve-se inserir a instrução de Subtração (SUB) conforme os modo de operandos da instrução ADD e a instrução de XOR conforme o modelo de instrução da AND.

2) Programas a serem implementado no NEANDER na memoria embarcada BRAM (descreva .coe para inicializar a BRAM)

1) Soma de duas matrizes A e B 3x3 com dados de 8 bits, onde os dados das matrizes estão armazenados em memoria
2) Multiplicação de dois valores A e B por soma sucessiva
3) Programa a ser definido pelo aluno que use as instruções de subtração com no mínimo 10 instruções no total.
4) Programa que use a instrução de XOR com no mínimo 10 instruções no total.

** **IMP: o endereço 0 da BRAM deve ter a instrução NOP. Logo a primeira instrução do programa estará no endereço 01 de BRAM.**

**TEMPLATE DE ENTREGA E APRESENTAÇÃO:**

1) Descrição do trabalho

O trabalho consiste em uma implementação do NEANDER utilizando a ferramenta ISE da Xilinx para descrever em VHDL o computador. Conforme a especificação do trabalho, foram adicionadas duas operações extras ao NEANDER original, sendo elas a operação XOR, com funcionalidade semelhante a AND existente e uma operação SUB de subtração, similar a ADD.
O datapath do NEANDER foi implementado previamente conforme a Parte 1 do trabalho e foram utilizados 4 programas de testes para a simulação posterior.
A descrição em VHDL recebeu como entrada os valores de clock e reset e tem como saída os sinais N, Z e a saída de dados que vai para a memória. A memória foi implementada como uma BRAM single port, de widht 8 e depth 256 conforme o computador original.
Para a Unidade de Controle se fez uma máquina de estados, com 10 estados, foram utilizados dois process a fim de otimizar o código, sendo um deles responsável por preservar o estado atual e prosseguir para o próximo e outro process para a implementação dos sinais de cada estados.

2) VHDL completo do Neander

```
-- Sistemas Digitais Para Computadores A
-- Matheus Almeida Silva - 00316326
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;
use IEEE.std_logic_arith.ALL;


entity neander is
   Port (  clk : in  STD_LOGIC;
         rst : in  STD_LOGIC;
                                 Z : out  STD_LOGIC;
                                 N : out  STD_LOGIC;
                                 outputData : out STD_LOGIC_VECTOR (7 downto 0));
end neander;

architecture Behavioral of neander is
COMPONENT memory
  PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;
-- Mux
signal selMux: std_logic :='0';
signal outputMux: STD_LOGIC_VECTOR (7 downto 0);
--Decod
signal instruction: STD_LOGIC_VECTOR(15 downto 0);
signal decod: STD_LOGIC_VECTOR(3 downto 0);
-- REM
signal cargaREM : std_logic := '0';
signal regREM: STD_LOGIC_VECTOR(7 downto 0);
signal outputREM: STD_LOGIC_VECTOR(7 downto 0);
-- ULA
signal selULA: std_logic_vector (2 downto 0) := "001";
signal input1ULA : STD_LOGIC_VECTOR(7 downto 0);
signal input2ULA : STD_LOGIC_VECTOR(7 downto 0);
signal outputULA: STD_LOGIC_VECTOR(7 downto 0);
signal regULA: STD_LOGIC_VECTOR(7 downto 0);
-- PC
signal cargaPC, incPC: std_logic := '0';
signal regPC, outputPC: STD_LOGIC_VECTOR (7 downto 0);
--AC
signal cargaAC : std_logic := '0';
signal inputAC: STD_LOGIC_VECTOR(7 downto 0);
signal outputAC: STD_LOGIC_VECTOR(7 downto 0);
--NZ
signal cargaNZ : std_logic := '0';
signal regN: STD_LOGIC;
signal regZ: STD_LOGIC;
signal outputN: STD_LOGIC;
signal outputZ: STD_LOGIC;
--RI
signal cargaRI : std_logic := '0';
signal inputRI: STD_LOGIC_VECTOR(7 downto 4);
signal outputRI: STD_LOGIC_VECTOR(7 downto 4);
-- Memoria
signal writeMem: STD_LOGIC_VECTOR(0 to 0) :="0";
signal outputMem: STD_LOGIC_VECTOR(7 downto 0);
-- Estados
type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9);
signal state, nextState : state_type;


begin
--Memoria
mem : memory
  PORT MAP (
```

```
    clka => clk,
    wea => writeMem,
    addra => outputREM,
    dina => outputAC,
    douta => outputMem
  );

--Mux
process (selMux, outputPC, outputMem)
begin
    if (selMux = '0') then
        outputMux <= outputPC;
    else
        outputMux <= outputMem;
    end if;
end process;

--PC
process (clk, rst)
begin
    if rst ='1' then
        regPC <= "00000000";
    elsif (clk'event and clk='1') then
        if (cargaPC='1') then
            regPC<= outputMem;
        elsif(incPC='1') then
            regPC <= regPC + 1;
        else
            regPC <= regPC;
        end if;
     end if;
end process;
outputPC <= regPC;


--REM
process(clk, rst)
begin
    if rst='1' then
        regREM <= "00000000";
    elsif (clk'event and clk='1') then
        if (cargaREM ='1') then
                                    regREM <= outputMux;
        else
                                    regREM<= regREM;
        end if;
    end if;
end process;
outputREM <= regREM;
```

```
--ULA
input1ULA <= outputAC;
input2ULA <= outputMem;

process(selULA, input1ULA, input2ULA)
begin

   case selULA is

      when "000" => regULA <= (input1ULA + input2ULA);
      when "001" => regULA <= (input1ULA AND input2ULA);
      when "010" => regULA <= (input1ULA OR input2ULA);
      when "011" => regULA <= (NOT input1ULA);
      when "100" => regULA <= input2ULA;
                  when "101" => regULA <= (input1ULA - input2ULA);
                  when "110" => regULA <= (input1ULA XOR input2ULA);

      when others => regULA <= "00000000";
                  end case;
end process;
outputULA <= regULA;

--AC
process (clk, rst)
begin
   if rst='1' then
      inputAC <= "00000000";
   elsif (clk'event and clk='1') then
      if (cargaAC='1') then
      inputAC <= outputULA;
      else
      inputAC <= inputAC;
      end if;
   end if;
end process;
outputAC <= inputAC;

--RI
process (clk, rst)
begin
   if rst='1' then
      inputRI<= "0000";
   elsif (clk'event and clk='1') then
      if (cargaRI='1') then
       inputRI <= outputMem(7 DOWNTO 4);
      else
       inputRI <= inputRI;
```

```
        end if;
    end if;
end process;
outputRI <= inputRI;


--NZ
process (clk, rst)
begin
    if rst='1' then
        regN <= '0';
        regZ <= '0';
    elsif(clk'event and clk='1') then
                    if (cargaNZ='1') then
                                    if outputAC = "00000000" then
                                            regZ <= '1';
                                    else
                                            regZ <= '0';
                                    end if;
                                    regN <= outputAC(7);
                    end if;
    end if;
end process;
outputZ <= regZ;
outputN <= regN;

--DECOD
decod <= outputRI (7 downto 4);

process(decod)
begin
    instruction <= "0000000000000000";
        case decod is
          when "0000" => instruction(0) <= '1'; --NOP
          when "0001" => instruction(1) <= '1'; --STA
          when "0010" => instruction(2) <= '1'; --LDA
          when "0011" => instruction(3) <= '1'; --ADD
          when "0100" => instruction(4) <= '1'; --OR
          when "0101" => instruction(5) <= '1'; --AND
          when "0110" => instruction(6) <= '1'; --NOT
          when "0111" => instruction(7) <= '1'; --NOP
          when "1000" => instruction(8) <= '1'; --JMP
          when "1001" => instruction(9) <= '1'; --JZ
          when "1010" => instruction(10) <= '1'; --JN
          when "1011" => instruction(11) <= '1'; --NOP
          when "1100" => instruction(12) <= '1'; --SUB
          when "1101" => instruction(13) <= '1'; --XOR
          when "1110" => instruction(14) <= '1'; --NOP
          when "1111" => instruction(15) <= '1'; --HLT
```

```
          when others => instruction <= "0000000000000000";
        end case;
end process;

--FSM
process(rst, clk)
begin
   if rst = '1' then
      state <= S0;
   elsif(clk'event and clk='1') then
      state <= nextState;
   end if;
end process;

-- UC
process(instruction, state, nextState, outputZ, outputN, outputMem)
begin
                cargaAC <='0';
                cargaNZ <='0';
                cargaPC <='0';
                selULA <="000";
                cargaRI <='0';
                incPC <= '0';
                writeMem <= "0";
                selMUX <= '0';
                cargaREM <='0';

        case state is
                when S0=>
                        cargaREM <='1';
                        nextState <=S1;

                when S1=>
                        cargaREM <='0';
                        incPC<='1';
                        nextState <= S2;

                when S2=>
                        incPC<='0';
                        cargaRI <='1';
                        nextState<=S3;

                when S3=>
                        incPC <='0';
                        cargaRI<='0';
                        if instruction(6) = '1' then    --NOT
                                selULA <= "011";
                                cargaAC <= '1';
                                cargaNZ <='1';
```

```
                        nextState <= S0;
                    elsif(instruction(9) = '1' and outputZ = '0') then        --JZ para Z=0
                        incPC <= '1';
                        nextState <=S0;
                    elsif(instruction(10) = '1' and outputN = '0') then        --JN para =0
                        incPC <='1';
                        nextState <=S0;
                    elsif(instruction(0) = '1') then --NOP
                        nextState <= S0;
                    elsif(instruction(15) = '1') then  --HALT
                        incPC <= '0';
                        nextState <= S9;
                    else
                        selMux <= '0';
                        cargaREM <= '1';
                        nextState <= S4;
                    end if;

                when S4 =>
                    selMux <= '0';
                    incPC <= '0';
                    cargaAC <= '0';
                    cargaNZ <='0';
                    cargaREM <= '0';
                    if(instruction(1) = '1' OR instruction(2) = '1' OR instruction(3) = '1' OR
instruction(4) = '1' OR instruction(5) = '1' OR instruction(12) = '1' OR instruction(13) = '1') then
                        incPC <= '1';
                    end if;
                    nextState <= S5;

                when S5 =>
                incPC <= '0';
                    if(instruction(1) = '1' OR instruction(2) = '1' OR instruction(3) = '1' OR
instruction(4) = '1' OR instruction(5) = '1' OR instruction(12) = '1' OR instruction(13) = '1') then
                        selMux <= '1';
                        cargaREM <= '1';
                        nextState <= S6;
                    else
                        cargaPC <= '1';
                        nextState <= S0;
                    end if;

                when S6 =>
                    incPC <= '0';
                    selMux <='0';
                    cargaREM <= '0';
                    cargaPC <='0';
                    nextState <= S7;
```

```
                when S7 =>
                        if instruction(1) ='1' then
                                writeMEM <= "1";
                        elsif instruction(2) ='1' then
                                selULA <= "100";
                        elsif instruction(3) ='1' then
                                selULA <= "000";
                        elsif instruction(4) ='1' then
                                selULA <= "010";
                        elsif instruction(5) ='1' then
                                selULA <= "001";
                        elsif instruction(12) ='1' then
                                selULA <= "101";
                        elsif instruction(13) ='1' then
                                selULA <= "110";
                        end if;
                        if instruction(1) = '1' then
                                nextState <= S0;
                        else
                                cargaAC <= '1';
                                cargaNZ <= '1';
                                nextState <= S0;
                        end if;

                when S9 =>
                        nextState <=S9;

                when others =>
                        nextState <= S0;
        end case;
end process;

Z <= outputZ;
N <= outputN;
outputData <= outputMem;

end Behavioral;
```

3) Testbench VHDL completo

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY neander_testbench IS
END neander_testbench;
```

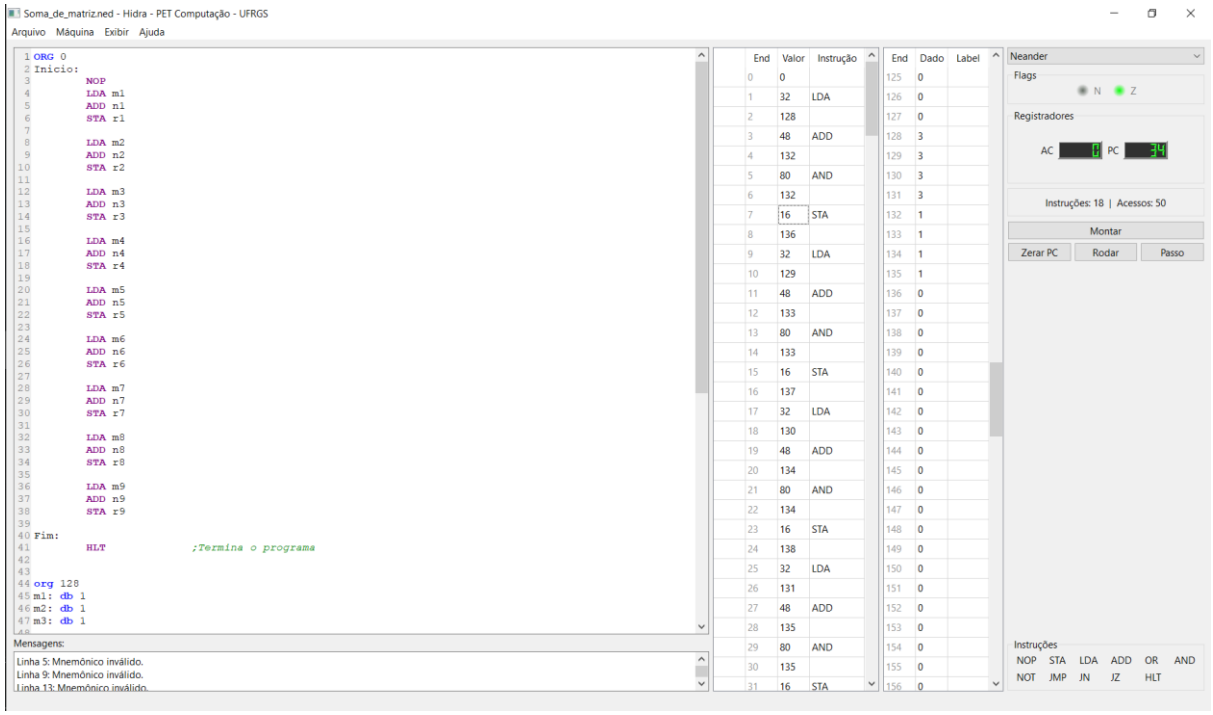
```
ARCHITECTURE behavior OF neander_testbench IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT neander
   PORT(
        clk : IN  std_logic;
        rst : IN  std_logic;
        Z : OUT  std_logic;
        N : OUT  std_logic;
        outputData : OUT  std_logic_vector(7 downto 0)
       );
   END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal rst : std_logic := '0';

          --Outputs
   signal Z : std_logic;
   signal N : std_logic;
   signal outputData : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant clk_period : time := 20 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
   uut: neander PORT MAP (
         clk => clk,
         rst => rst,
         Z => Z,
         N => N,
         outputData => outputData
        );

   -- Clock process definitions
   clk_process :process
   begin
                    clk <= '0';
                    wait for clk_period/2;
                    clk <= '1';
                    wait for clk_period/2;
   end process;
```

```
  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
                rst <= '1';
    wait for clk_period*5;
                rst <= '0';
    wait for clk_period*5;
    wait;
  end process;

END;
```

4) Explicação e descrição das aplicações em Assembly

Programa 1 – Soma de Matrizes 3x3

O primeiro programa consiste em uma soma de matrizes 3x3, sendo basicamente uma sequência de somas para os valores da matriz na memória, que estão dispostas como uma lista. Assim o primeiro valor(linha 1, coluna 1) da matriz m é somado com o primeiro valor da matriz n(linha 1, coluna 1) e armazenado na matriz r na mesma posição. Abaixo segue o código no Hidra e parte do coe resultante.

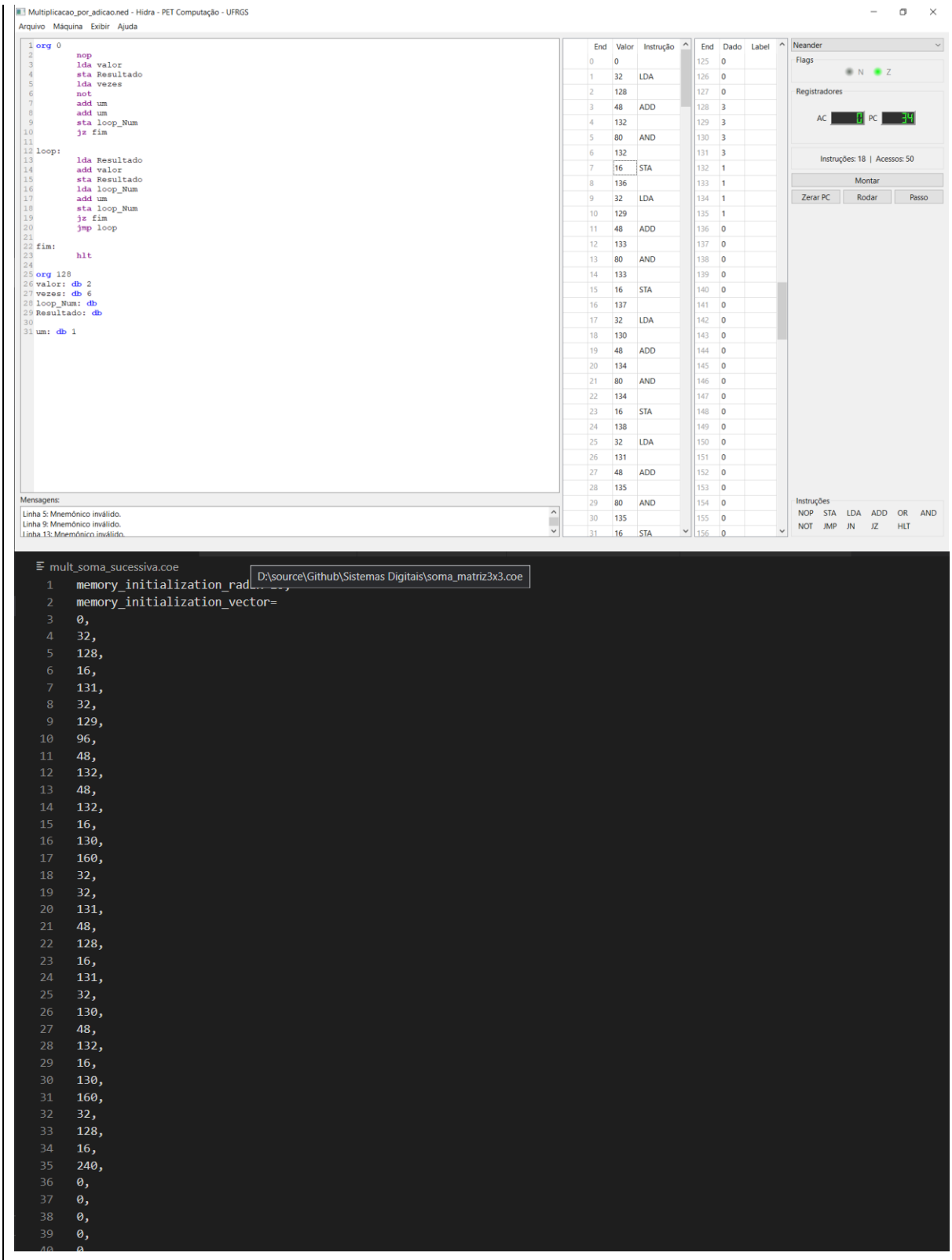


soma_matriz3x3.coe

```
memory_initialization_radix=10;
memory_initialization_vector=
0,
32,
128,
48,
137,
16,
146,
32,
129,
48,
138,
16,
147,
32,
130,
48,
139,
16,
148,
32,
131,
48,
140,
16,
149,
32,
132,
48,
141,
16,
150,
32,
133,
48,
142,
16,
151,
32,
```

Programa 2 – Multiplicação Por Somas Sucessivas

O segundo programa consiste em uma multiplicação por somas sucessivas utilizando um loop e valores auxiliares de controle. Abaixo segue o código no Hidra e parte do coe resultante.




Programa 3 – Subtração de Matrizes 2x2

De maneira análoga ao programa 1, o programa 3 realiza a operação de subtração entre duas matrizes. Abaixo segue o código no Hidra e parte do coe resultante

Sub.ned - Hidra - PET Computação - UFRGS

Arquivo Máquina Exibir Ajuda

```
ORG 0
Inicio:
        NOP
        LDA m1
        SUB n1
        STA r1

        LDA m2
        SUB n2
        STA r2

        LDA m3
        SUB n3
        STA r3

        LDA m4
        SUB n4
        STA r4

Fim:
        HLT             ;Termina o programa


org 128
m1: db 3
m2: db 3
m3: db 3
m4: db 3


n1: db 1
n2: db 1
n3: db 1
n4: db 1


r1: db
r2: db
r3: db
r4: db

```

sub_matriz2x2.coe

```
memory_initialization_radix=10;
memory_initialization_vector=
0,
32,
128,
192,
137,
16,
136,
32,
129,
192,
133,
16,
137,
32,
130,
192,
134,
16,
138,
32,
131,
192,
135,
16,
139,
240,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
```

Programa 4 – Subtração e Xor de Matrizes 2x2

Praticamente igual ao Programa 3 com o detalhe de que após a operação de subtração é realizado um xor entre o valor resultante e o da matriz n. Abaixo segue o código no Hidra e parte do coe resultante.

subexor.ned - Hidra - PET Computação - UFRGS

Arquivo Máquina Exibir Ajuda

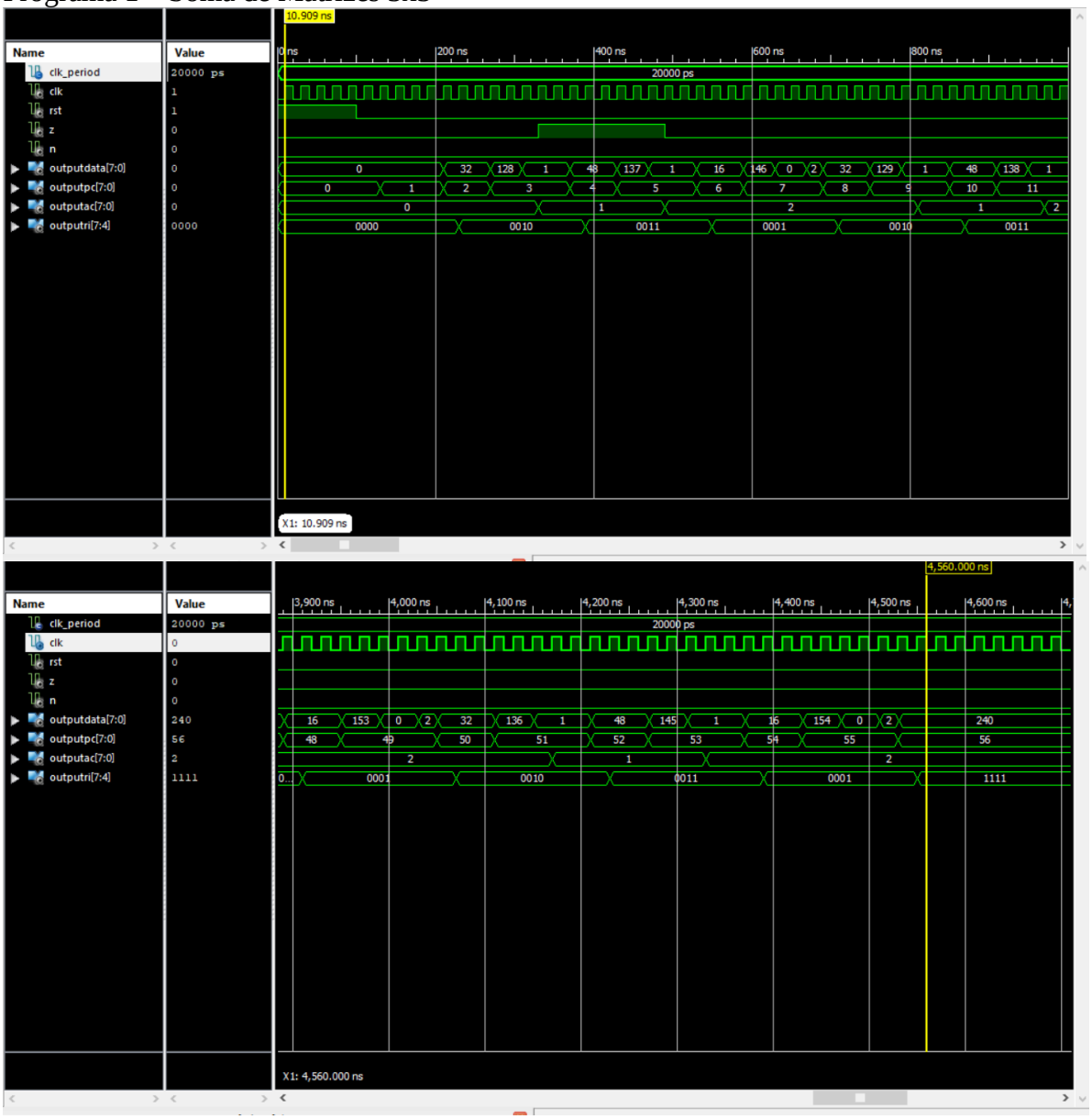
```
ORG 0
Inicio:
        NOP
        LDA m1
        SUB n1
        XOR n1
        STA r1

        LDA m2
        SUB n2
        XOR n2
        STA r2

        LDA m3
        SUB n3
        XOR n3
        STA r3

        LDA m4
        SUB n4
        XOR n4
        STA r4

Fim:
        HLT             ;Termina o programa


org 128
m1: db 3
m2: db 3
m3: db 3
m4: db 3


n1: db 1
n2: db 1
n3: db 1
n4: db 1


r1: db
r2: db
r3: db
r4: db

```

subexor_matriz2x2.coe

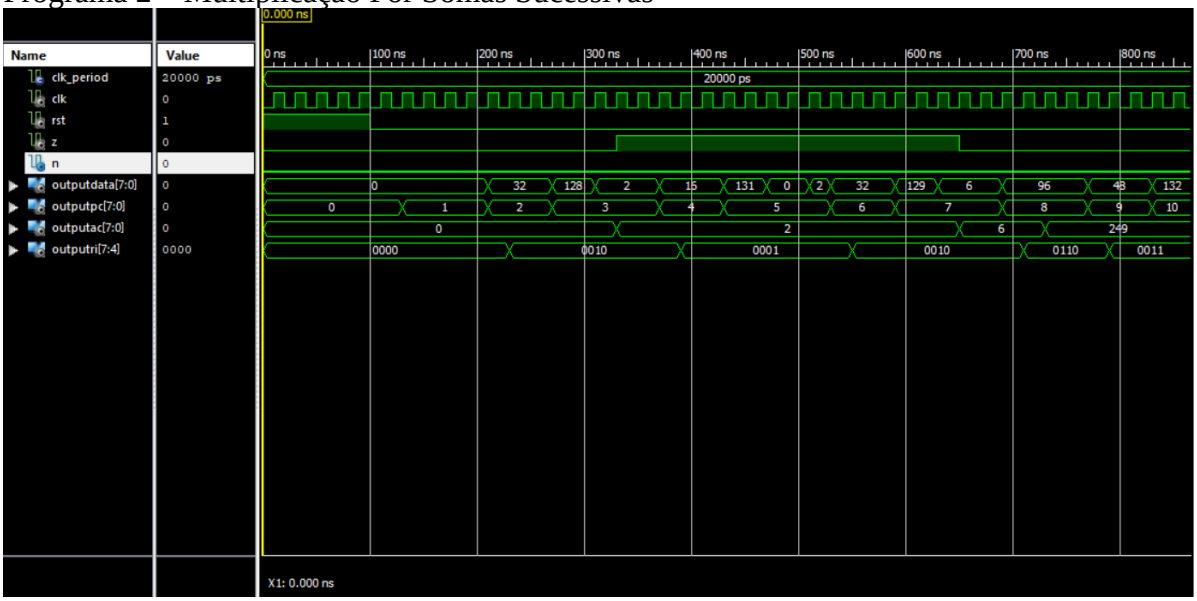
```
memory_initialization_radix=10;
memory_initialization_vector=
0,
32,
128,
192,
132,
208,
132,
16,
136,
32,
129,
192,
133,
208,
133,
16,
137,
32,
130,
192,
134,
208,
134,
16,
138,
32,
131,
192,
135,
208,
135,
16,
139,
240,
0,
0,
0,
```

5) Simulações sem e com atraso com detalhes e flechas mostrando inicio meio e final do programa e resultados

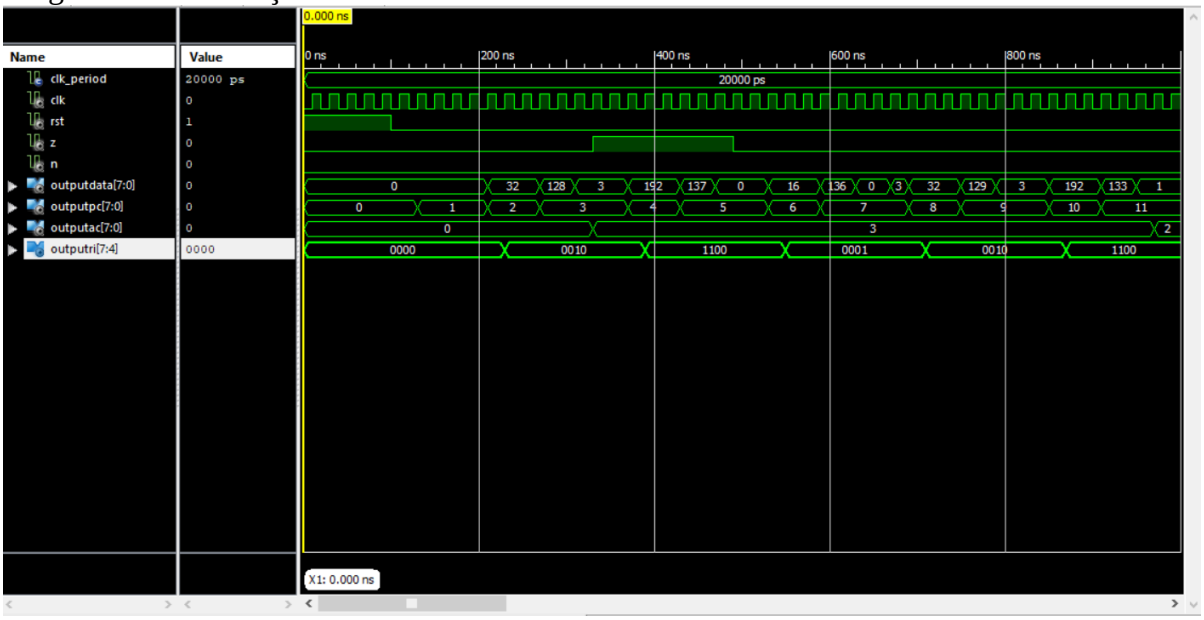
Programa 1 – Soma de Matrizes 3x3

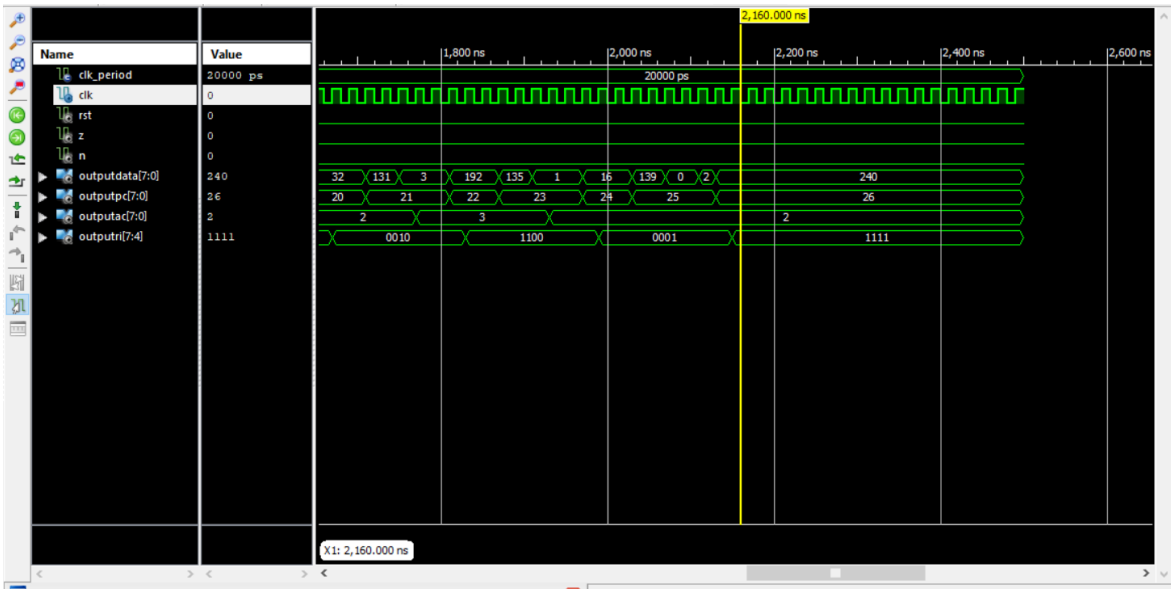


Programa 2 – Multiplicação Por Somas Sucessivas

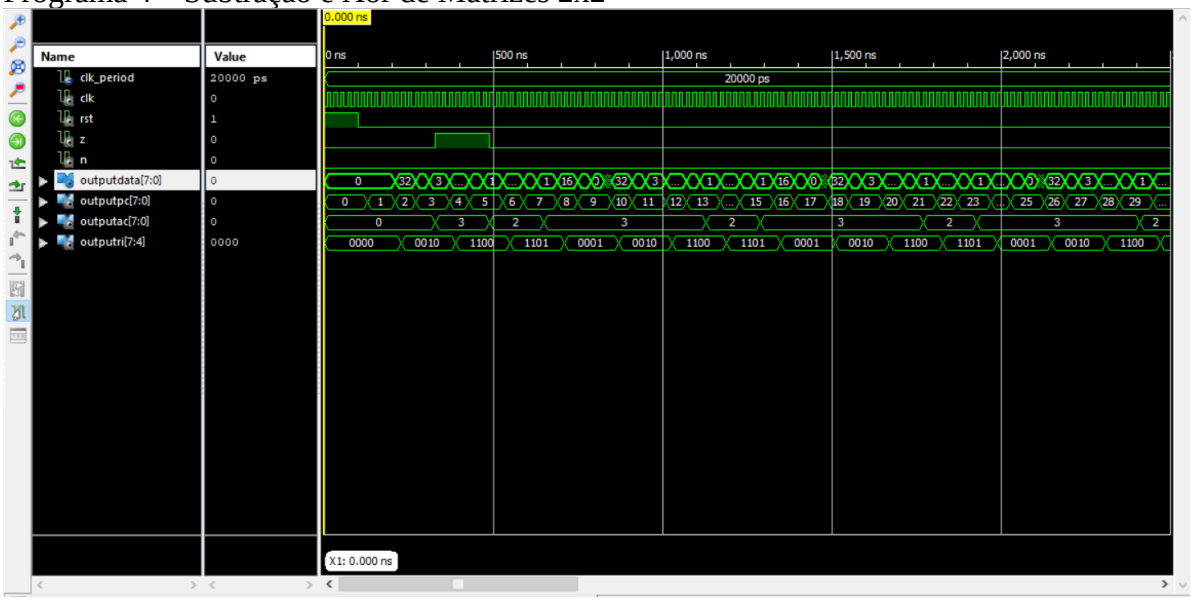


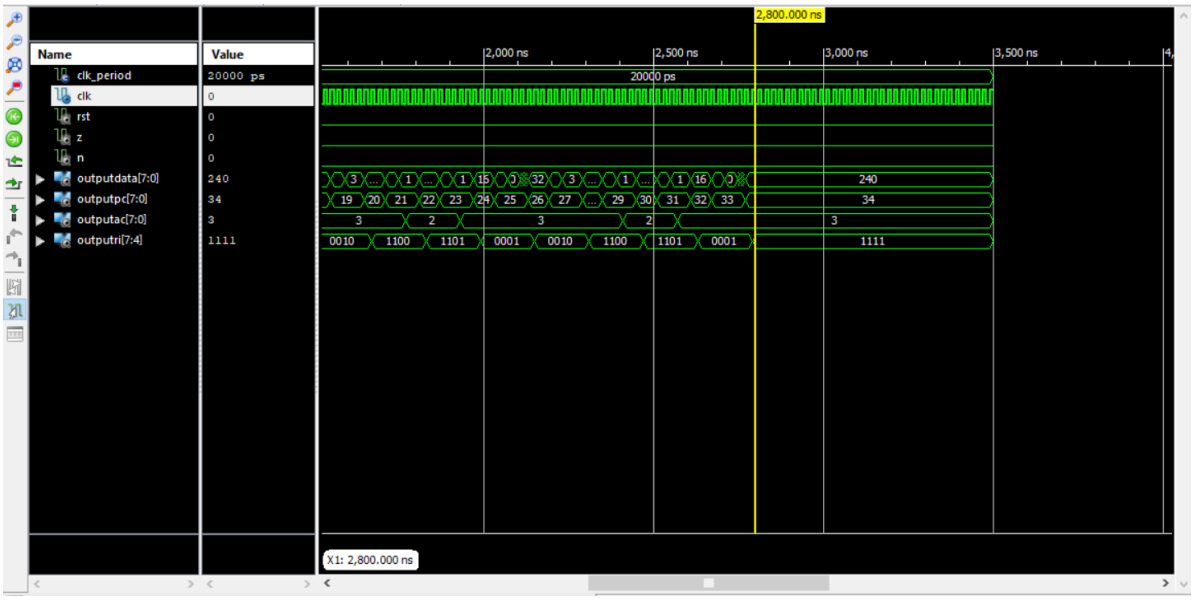

Programa 3 – Subtração de Matrizes 2x2






Programa 4 – Subtração e Xor de Matrizes 2x2






6) Dados de área, tempo de execução em ciclos de relógio e tempo em segundos deve ser apresentado dado um determinado clock usado.

| Programa | Numero de Instruções Executadas | Tempo de execução em # de ciclos de relógio (c.c.) | Tempo de execução em Segundos (Neander operando a 50 MHz) |
|---|---|---|---|
| Soma de matrizes | 29 | 228 | 4560e-9s |
| Multiplicação por somas sucessivas | 13 | 70 | 1440e-9s |
| Programa com SUB | 14 | 108 | 2160e-9s |
| Programa com XOR | 18 | 140 | 2800e-9s |

**Dados de Area do Neander**
FPGA device: xc3s100e-5cp132
Numero de 4-LUTs: 102
Numero de ffps: 38
Numero de BRAM: 1
Numero de DSP: 1

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Flip Flops | 38 | 1,920 | 1% | |
| Number of 4 input LUTs | 102 | 1,920 | 5% | |
| Number of occupied Slices | 59 | 960 | 6% | |
| Number of Slices containing only related logic | 59 | 59 | 100% | |
| Number of Slices containing unrelated logic | 0 | 59 | 0% | |
| Total Number of 4 input LUTs | 109 | 1,920 | 5% | |
| Number used as logic | 102 | | | |
| Number used as a route-thru | 7 | | | |
| Number of bonded IOBs | 12 | 83 | 14% | |
| Number of RAMB16s | 1 | 4 | 25% | |
| Number of BUFGMUXs | 1 | 24 | 4% | |
| Average Fanout of Non-Clock Nets | 3.77 | | | |

7) **(1 ponto extra)** Se o Neander for prototipado na placa de prototipação, mostrar vídeos do funcionamento mostrando dados da memoria do Neander (debugger com memoria BRAM dual port, chaves para controlar os endereços de memória e display 7seg para mostrar os resultados).

**Não implementado.**