# Trabalho 2 de Sistemas Digitais

Projeto PC-PO/HLS
Multiplicação de matrizes 3x3

Gabriel Patrocínio - 00340812
Leonardo Santos - 00341831

# Resumo

1. Implementar um algoritmo de multiplicação de duas matrizes 3x3.
2. Solução HLS
   - Programa em C
   - Solução básica com HLS
   - Uma solução básica e duas otimizadas com HLS
3. Solução PC-PO
   - Fluxograma PC-PO
   - Esquemático da PO
   - FSM da PC
   - Programa em VHDL e simulações
4. Comparação dos resultados

# 2. Solução HLS

- Programa em C

```c
#ifndef MATRIX_MULT_H
#define MATRIX_MULT_H

#include <stdint.h>

#define N 3
typedef uint8_t mat_in_t;
typedef int32_t mat_out_t;

void matrix_mult(mat_in_t A[N][N], mat_in_t B[N][N], mat_out_t R[N][N]);

#endif
```

Código 1: *matrix_mult.h*

# 2. Solução HLS

- Programa em C

```
1   #include "matrix_mult.h"
2
3   void matrix_mult(mat_in_t A[N][N], mat_in_t B[N][N], mat_out_t R[N][N]) {
4       Row_Loop: for (int i = 0; i < N; i++) {
5           Col_Loop: for (int j = 0; j < N; j++) {
6               mat_out_t sum = 0;
7               Prod_Loop: for (int k = 0; k < N; k++) {
8                   sum += A[i][k] * B[k][j];
9               }
10              R[i][j] = sum;
11          }
12      }
13  }
```

Código 2: *matrix_mult.cpp*

# 2. Solução HLS

- Uma solução básica e duas otimizadas com HLS

Estimated Quality of Results

Timing Estimate

| TARGET | WITH UNCERTAINTY | ESTIMATED |
|---|---|---|
| 10.00 ns | 7.30 ns | 2.301 ns |

Performance & Resource Estimates

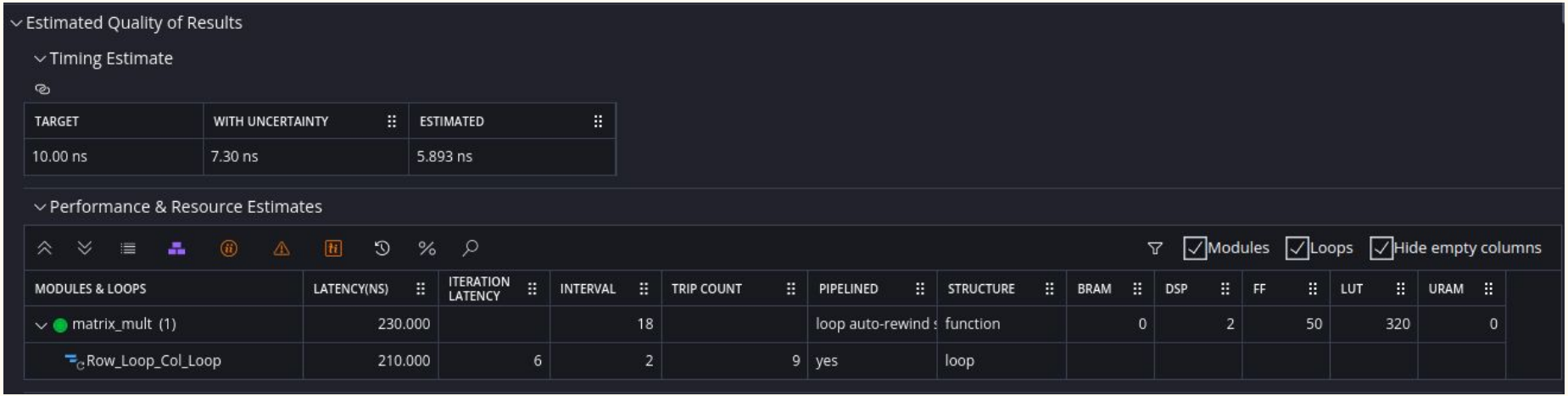
Modules Loops Hide empty columns

| MODULES & LOOPS | LATENCY(NS) | ITERATION LATENCY | INTERVAL | TRIP COUNT | PIPELINED | STRUCTURE | BRAM | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| matrix_mult (1) | 1.600E3 | | 161 | | no | function | 0 | 1 | 48 | 186 | 0 |
| Row_Loop (1) | 1.590E3 | 53 | - | 3 | no | loop | | | | | |

Síntese 1: solução não otimizada

# 2. Solução HLS

- Uma solução básica e duas otimizadas com HLS



Estimated Quality of Results

Timing Estimate

| TARGET | WITH UNCERTAINTY | ESTIMATED |
|---|---|---|
| 10.00 ns | 7.30 ns | 5.893 ns |

Performance & Resource Estimates

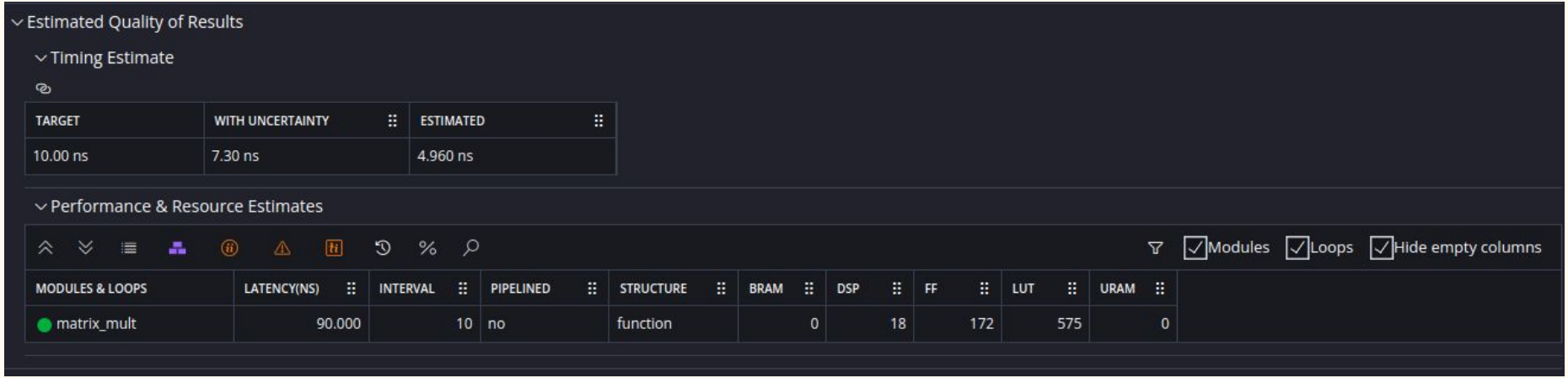
Modules Loops Hide empty columns

| MODULES & LOOPS | LATENCY(NS) | ITERATION LATENCY | INTERVAL | TRIP COUNT | PIPELINED | STRUCTURE | BRAM | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| matrix_mult (1) | 230.000 | | 18 | | loop auto-rewind s | function | 0 | 2 | 50 | 320 | 0 |
| Row_Loop_Col_Loop | 210.000 | 6 | 2 | 9 | yes | loop | | | | | |

Síntese 2: solução otimizada com **pipeline**

# 2. Solução HLS

- Uma solução básica e duas otimizadas com HLS



Estimated Quality of Results

Timing Estimate

| TARGET | WITH UNCERTAINTY | ESTIMATED |
|---|---|---|
| 10.00 ns | 7.30 ns | 4.960 ns |

Performance & Resource Estimates

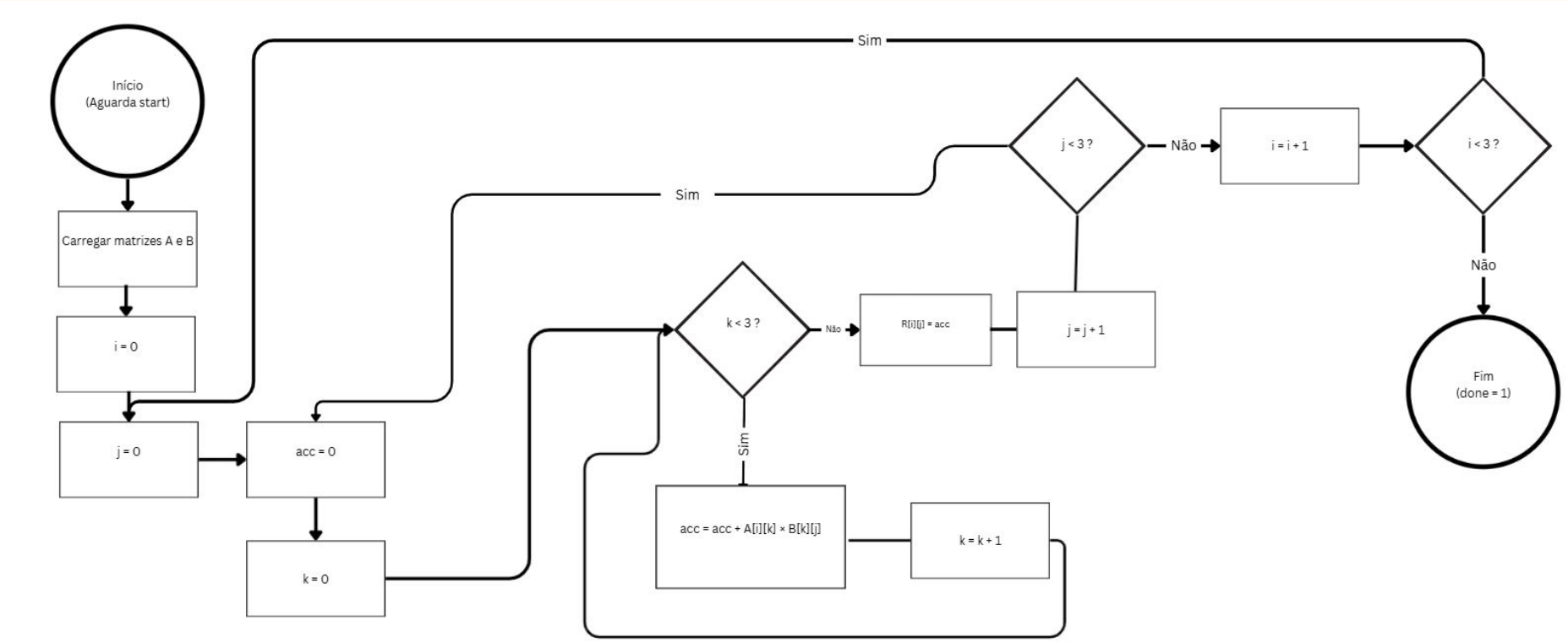
Modules Loops Hide empty columns

| MODULES & LOOPS | LATENCY(NS) | INTERVAL | PIPELINED | STRUCTURE | BRAM | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|---|---|---|---|
| matrix_mult | 90.000 | 10 | no | function | 0 | 18 | 172 | 575 | 0 |

Síntese 3: solução otimizada com **unroll**

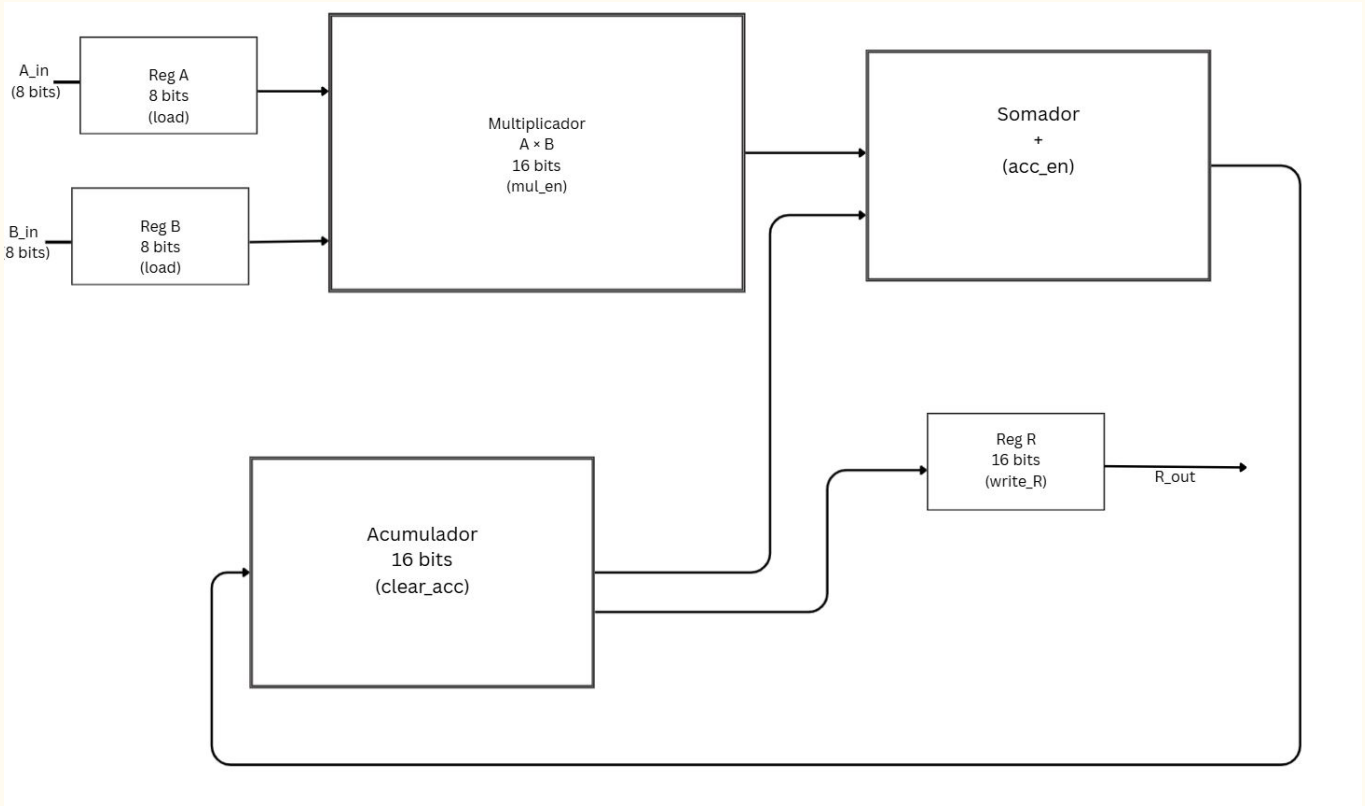
# 3. Solução PC-PO

- Fluxograma da PC-PO



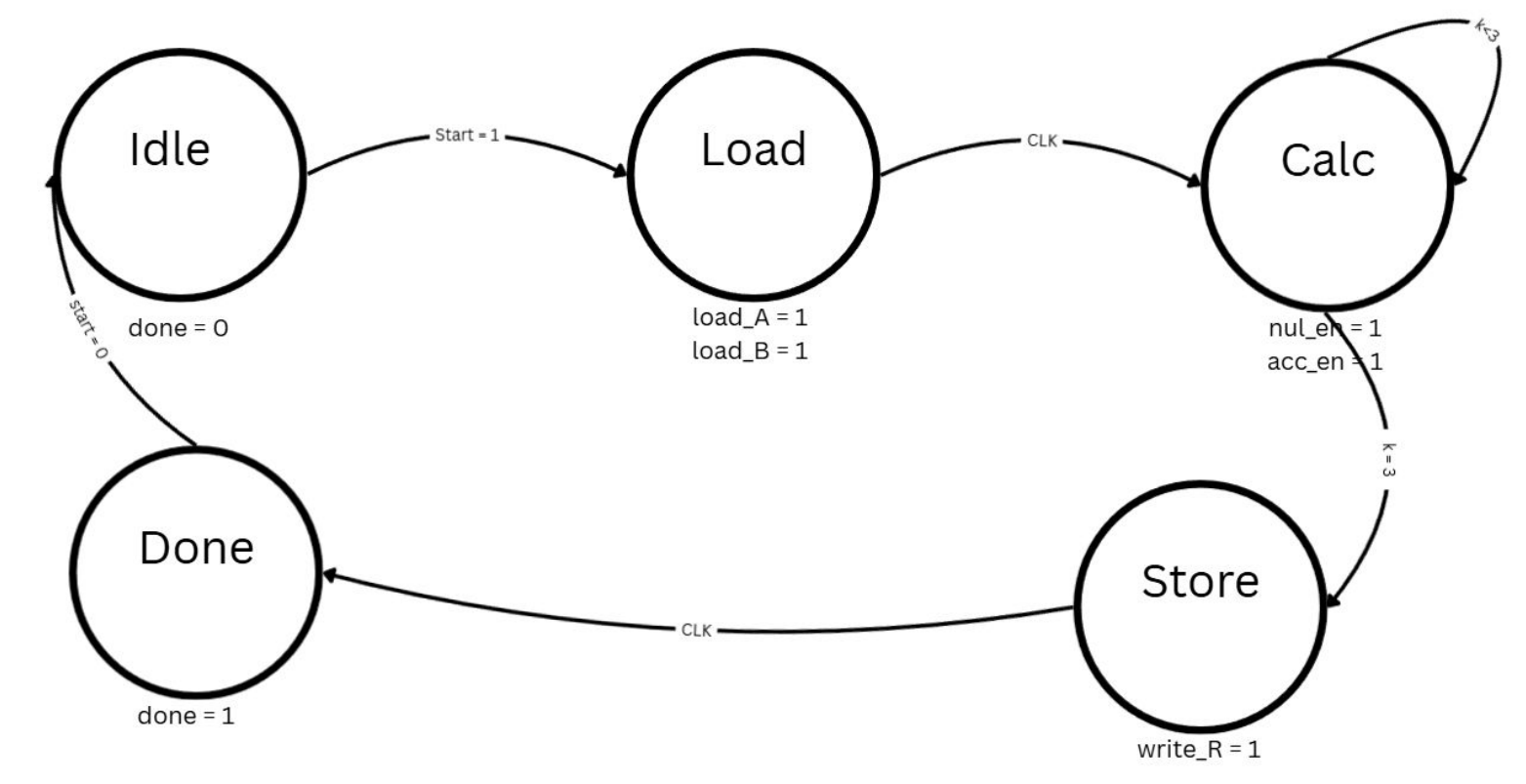
# 3. Solução PC-PO

- Esquemático da PO



# 3. Solução PC-PO

- FSM da PC



# 3. Solução PC-PO

- Programa em VHDL

Código 1: *matrix_mult_top.vhd*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.pkg_matrix.ALL;

entity matrix_mult_top is
    Port (
        clk     : in  std_logic;
        rst     : in  std_logic;
        start   : in  std_logic;
        A_in    : in  t_matrix_in;
        B_in    : in  t_matrix_in;
        done    : out std_logic;
        R_out   : out t_matrix_out
    );
end matrix_mult_top;

architecture Structural of matrix_mult_top is
    signal ld_regs, clr_acc, ld_acc, wr_res : std_logic;
    signal inc_k, inc_j, inc_i, clr_k, clr_j, clr_i : std_logic;
    signal k_done, j_done, i_done : std_logic;
begin
    inst_PC: entity work.matrix_mult_pc
    port map (
        clk => clk, rst => rst, start => start, done => done,
        k_done => k_done, j_done => j_done, i_done => i_done,
        ld_regs => ld_regs, clr_acc => clr_acc, ld_acc => ld_acc, wr_res => wr_res,
        inc_k => inc_k, inc_j => inc_j, inc_i => inc_i,
        clr_k => clr_k, clr_j => clr_j, clr_i => clr_i
    );

    inst_PO: entity work.matrix_mult_po
    port map (
        clk => clk, rst => rst,
        ld_regs => ld_regs, clr_acc => clr_acc, ld_acc => ld_acc, wr_res => wr_res,
        inc_k => inc_k, inc_j => inc_j, inc_i => inc_i,
        clr_k => clr_k, clr_j => clr_j, clr_i => clr_i,
        k_done => k_done, j_done => j_done, i_done => i_done,
        A_in => A_in, B_in => B_in, R_out => R_out
    );
end Structural;
```

# 3. Solução PC-PO

- Programa em VHDL

## Código 2: *matrix_mult_pc.vhd*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity matrix_mult_pc is
    Port (
        clk, rst, start : in  std_logic;
        done            : out std_logic;

        -- Inputs da PO
        k_done, j_done, i_done : in  std_logic;

        -- Outputs para PO
        ld_regs, clr_acc, ld_acc, wr_res : out std_logic;
        inc_k, inc_j, inc_i : out std_logic;
        clr_k, clr_j, clr_i : out std_logic
    );
end matrix_mult_pc;
```

```vhdl
architecture Behavioral of matrix_mult_pc is
    type state_type is (IDLE, SETUP, INIT_I, INIT_J, INIT_K, CALC, CHECK_K, WRITE_RES, CHECK_J, CHECK_I, FINISHED);
    signal current_state, next_state : state_type;
begin

    process(clk, rst)
    begin
        if rst = '1' then current_state <= IDLE;
        elsif rising_edge(clk) then current_state <= next_state;
        end if;
    end process;

    process(current_state, start, k_done, j_done, i_done)
    begin
        -- Defaults
        ld_regs<='0'; clr_acc<='0'; ld_acc<='0'; wr_res<='0';
        inc_k<='0'; inc_j<='0'; inc_i<='0';
        clr_k<='0'; clr_j<='0'; clr_i<='0';
        done<='0'; next_state <= current_state;

        case current_state is
            when IDLE =>
                if start = '1' then next_state <= SETUP; end if;
            when SETUP =>
                ld_regs <= '1'; next_state <= INIT_I;
            when INIT_I =>
                clr_i <= '1'; next_state <= INIT_J;
            when INIT_J =>
                clr_j <= '1'; next_state <= INIT_K;
            when INIT_K =>
                clr_k <= '1'; clr_acc <= '1'; next_state <= CALC;
            when CALC =>
                ld_acc <= '1'; next_state <= CHECK_K;
            when CHECK_K =>
                if k_done = '1' then next_state <= WRITE_RES;
                else inc_k <= '1'; next_state <= CALC; end if;
            when WRITE_RES =>
                wr_res <= '1'; next_state <= CHECK_J;
            when CHECK_J =>
                if j_done = '1' then next_state <= CHECK_I;
                else inc_j <= '1'; next_state <= INIT_K; end if;
            when CHECK_I =>
                if i_done = '1' then next_state <= FINISHED;
                else inc_i <= '1'; next_state <= INIT_J; end if;
            when FINISHED =>
                done <= '1';
                if start = '0' then next_state <= IDLE; end if;
            when others => next_state <= IDLE;
        end case;
    end process;
end Behavioral;
```

# 3. Solução PC-PO

- Programa em VHDL

## Código 3: *matrix_mult_po.vhd*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.pkg_matrix.ALL;

entity matrix_mult_po is
    Port (
        clk         : in  std_logic;
        rst         : in  std_logic;

        -- Controle
        ld_regs     : in  std_logic;
        clr_acc     : in  std_logic;
        ld_acc      : in  std_logic;
        wr_res      : in  std_logic;
        inc_k, inc_j, inc_i : in  std_logic;
        clr_k, clr_j, clr_i : in  std_logic;

        -- Status (Done quando contadores chegam a 2)
        k_done      : out std_logic;
        j_done      : out std_logic;
        i_done      : out std_logic;

        -- Dados
        A_in        : in  t_matrix_in;
        B_in        : in  t_matrix_in;
        R_out       : out t_matrix_out
    );
end matrix_mult_po;
```

```vhdl
architecture Behavioral of matrix_mult_po is
    signal reg_A : t_matrix_in;
    signal reg_B : t_matrix_in;
    signal reg_R : t_matrix_out := (others => (others => (others => '0')));

    -- Contadores agora vão de 0 a 2 (3 iterações)
    signal i, j, k : integer range 0 to 2 := 0;

    signal mult_res : unsigned(15 downto 0);
    signal acc      : unsigned(19 downto 0);
begin

    -- Multiplicação Padrão: Linha de A x Coluna de B
    -- A(i,k) * B(k,j)
    mult_res <= reg_A(i, k) * reg_B(k, j);

    process(clk, rst)
    begin
        if rst = '1' then
            i <= 0; j <= 0; k <= 0;
            acc <= (others => '0');
            reg_R <= (others => (others => (others => '0')));
            reg_A <= (others => (others => (others => '0')));
            reg_B <= (others => (others => (others => '0')));
        elsif rising_edge(clk) then
            -- Carregar Entradas
            if ld_regs = '1' then
                reg_A <= A_in;
                reg_B <= B_in;
            end if;

            -- Acumulador
            if clr_acc = '1' then
                acc <= (others => '0');
            elsif ld_acc = '1' then
                acc <= acc + resize(mult_res, 20);
            end if;
                -- Escrever Resultado
                if wr_res = '1' then
                    reg_R(i, j) <= acc;
                end if;

                -- Contadores
                if clr_k = '1' then k <= 0;
                elsif inc_k = '1' then k <= k + 1; end if;

                if clr_j = '1' then j <= 0;
                elsif inc_j = '1' then j <= j + 1; end if;

                if clr_i = '1' then i <= 0;
                elsif inc_i = '1' then i <= i + 1; end if;
            end if;
        end process;

        -- Sinais de Status: Done quando o contador é 2 (Tamanho - 1)
        k_done <= '1' when k = 2 else '0';
        j_done <= '1' when j = 2 else '0';
        i_done <= '1' when i = 2 else '0';

        R_out <= reg_R;

end Behavioral;
```

# 3. Solução PC-PO

- Testbench

## Código 1: *tb_matrix_mult.vhd*

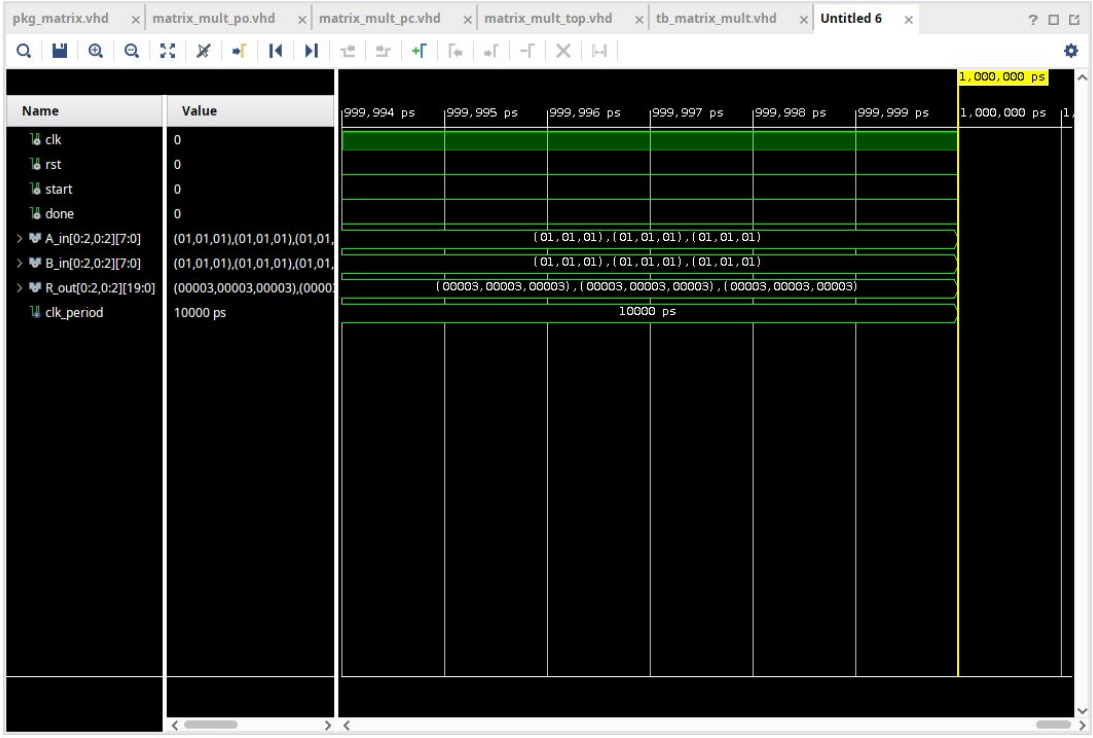
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.pkg_matrix.ALL;

entity tb_matrix_mult_3 is
-- Testbench 3: Entradas = 3
end tb_matrix_mult_3;

architecture Behavioral of tb_matrix_mult_3 is
    component matrix_mult_top
        Port (
            clk, rst, start : in std_logic;
            A_in, B_in : in t_matrix_in;
            done : out std_logic;
            R_out : out t_matrix_out
        );
    end component;

    signal clk, rst, start, done : std_logic := '0';
    signal A_in, B_in : t_matrix_in := (others => (others => (others => '0')));
    signal R_out : t_matrix_out;
    constant clk_period : time := 10 ns;

begin
    uut: matrix_mult_top port map (
        clk => clk, rst => rst, start => start,
        A_in => A_in, B_in => B_in, done => done, R_out => R_out
    );

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

    stim_proc: process
    begin
        rst <= '1'; wait for 20 ns; rst <= '0'; wait for 20 ns;
        --  ----- ALTERAR AQUI PARA CADA TESTE----- --
        -- TESTE 3: Entradas com valor 3
        -- Esperado: 3*3 + 3*3 + 3*3 = 27 (Hex: 1B)
        report "Iniciando Teste 3 (Valores = 3)";
        A_in <= (others => (others => to_unsigned(3, 8)));
        B_in <= (others => (others => to_unsigned(3, 8)));

        start <= '1'; wait for clk_period; start <= '0';

        wait until done = '1';
        wait for clk_period;

        report "Teste 3 Concluido.";
        wait;
    end process;
end Behavioral;
```

# 3. Solução PC-PO

- Simulações

Simulação comportamental 1:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

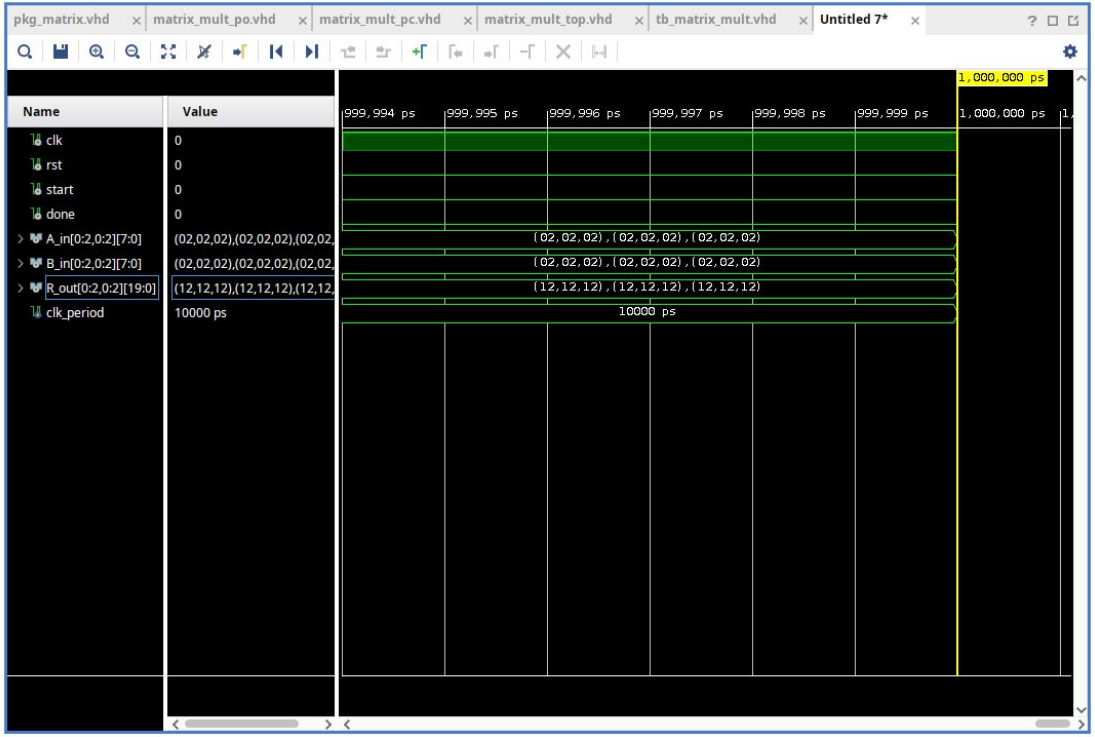


# 3. Solução PC-PO

- Simulações

Simulação comportamental 2:

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \times \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 12 & 12 & 12 \\ 12 & 12 & 12 \\ 12 & 12 & 12 \end{bmatrix}$$

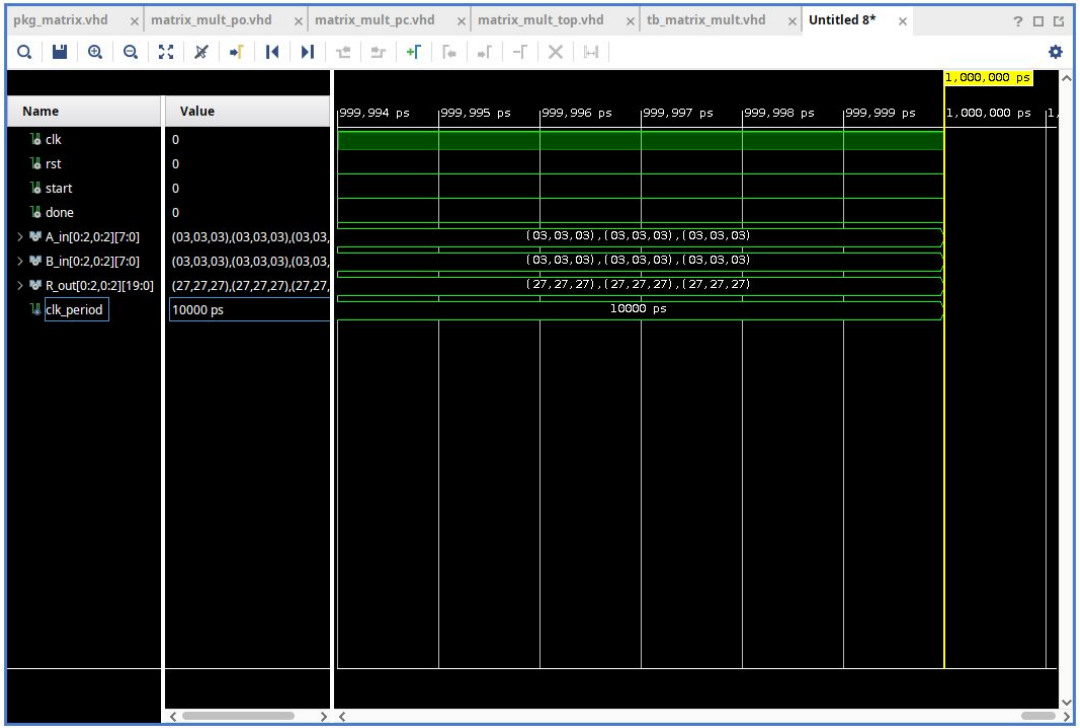


# 3. Solução PC-PO

- Simulações

Simulação comportamental 3:

$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 27 & 27 & 27 \\ 27 & 27 & 27 \\ 27 & 27 & 27 \end{bmatrix}$$



# 3. Solução PC-PO

- Simulações

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 175 | 133800 | 0.13 |
| FF | 361 | 267600 | 0.13 |
| IO | 328 | 400 | 82.00 |
| BUFG | 1 | 32 | 3.13 |

Síntese pós-implementação

# 4. Comparação dos resultados

1. PC-PO:
   - FF: **361**
   - LUT: **175**
   - DSP: **0**
   - Ciclos: **~34**
   - Memória (BRAM): **0**
   - Pinos I/O: **328**

2. HLS básico:
   - FF: **48**
   - LUT: **186**
   - DSP: **1**
   - Ciclos: **160**
   - Memória (BRAM): **0**
   - Pinos I/O: **328***

3. HLS com pipeline:
   - FF: **50**
   - LUT: **320**
   - DSP: **2**
   - Ciclos: **23**
   - Memória (BRAM): **0**
   - Pinos I/O: **328***

4. HLS com unroll:
   - FF: **172**
   - LUT: **575**
   - DSP: **18**
   - Ciclos: **9**
     Memória (BRAM): **0**
   - Pinos I/O: **328***

# 4. Comparação dos resultados

1. PC-PO
   - Teve o maior uso de Flip-Flops (361) devido aos registradores explícitos de dados.
   - O uso de blocos DSP foi nulo. A síntese converteu as multiplicações em lógica distribuída (LUTs).
   - O desempenho foi mediano, exigindo cerca de 34 ciclos para completar a tarefa.
   - A velocidade é limitada pela máquina de estados que processa os dados sequencialmente.

# 4. Comparação dos resultados

## 2. HLS básico

- Foi a implementação mais lenta de todas, necessitando de 160 ciclos de clock.
- O hardware reutiliza o mesmo recurso de cálculo para todas as operações, uma por vez.
- Não há paralelismo nem otimização de fluxo de dados.

# 4. Comparação dos resultados

## 3. HLS com pipeline

- A otimização insere registradores entre as etapas do cálculo, permitindo que uma nova operação inicie a cada ciclo de clock antes da anterior terminar.
- A técnica de pipeline permitiu iniciar novos cálculos antes do término dos anteriores.
- O tempo de execução caiu drasticamente para apenas 23 ciclos.
- Houve um leve aumento na área, subindo para 2 DSPs.
- Esta opção representa o melhor equilíbrio entre custo de hardware e velocidade.

# 4. Comparação dos resultados

## 4. HLS com unroll

- A diretiva "desenrola" o laço de repetição, criando cópias físicas do hardware para executar todas as iterações do loop ao mesmo tempo.
- O hardware foi replicado para realizar todos os cálculos simultaneamente (paralelismo total).
- Foi a versão mais rápida, concluindo a operação em apenas 9 ciclos.
- O custo de área foi altíssimo, consumindo 18 DSPs e triplicando o uso de LUTs.
- É a solução ideal para desempenho máximo, mas ocupa muito espaço no chip.