SISTEMAS DIGITAIS

Enzo Lisbôa Peixoto e Pedro Scholz Soares

Dezembro de 2025

DESCRIÇÃO

- Implementar um algoritmo de multiplicação de duas matrizes 3x3.
- HLS e duas opções de otimização
- Fazer o projeto PC-PO
- Comparar as 3 implementações em dados de área e desempenho

CÓDIGO EM C


```c
#include <stdint.h>

#define N 3

void matrix_mult_3x3(uint8_t A[N][N],
                     uint8_t B[N][N],
                     uint16_t R[N][N]) {

    uint16_t soma;

    linha_loop: for (int i = 0; i < N; i++) {

        coluna_loop: for (int j = 0; j < N; j++) {
            soma = 0;

            produto_loop: for (int k = 0; k < N; k++) {
                soma += A[i][k] * B[k][j];
            }

            R[i][j] = soma;
        }
    }
}
```

MELHORAR VAZÃO

- Adição de pipeline
- O Initiation Interval foi deixado em automático
- Objetivo: Melhorar a vazão, respeitando os limites da memória

- ARRAYPARTITION nas variáveis e UNROLL nos loops
- Converte a memória RAM em registradores distribuídos (FF), permitindo acesso simultâneo.
- Objetivo: Obter a latência mínima possível, aceitando o aumento consideravel na área do chip

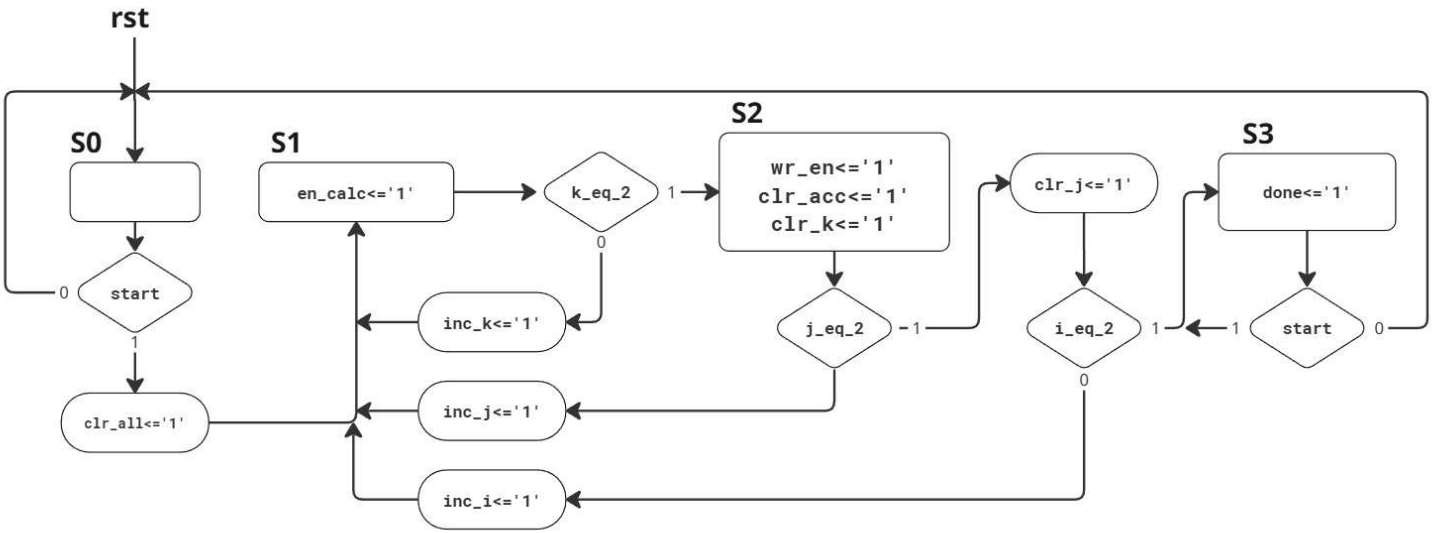
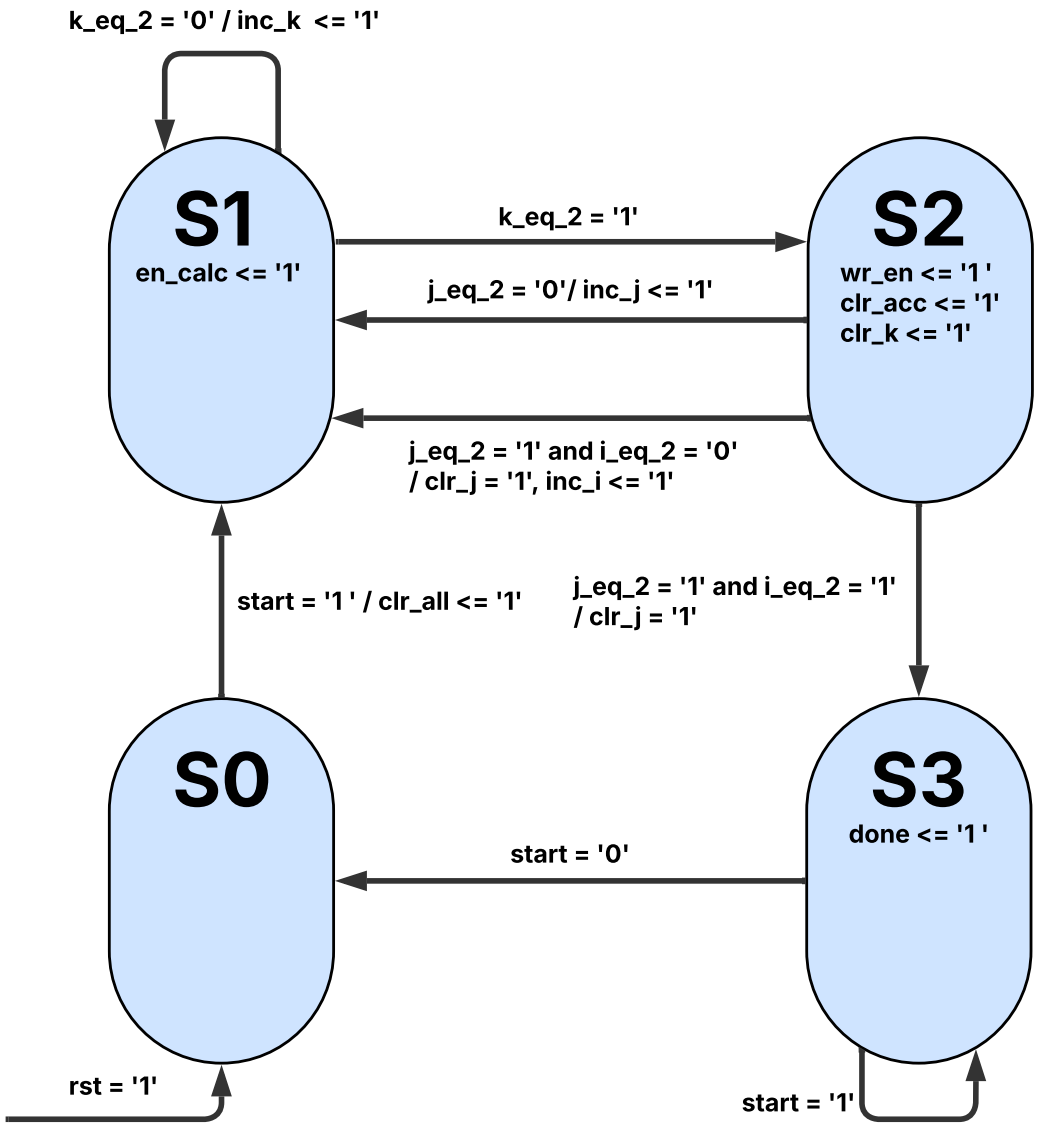


Figura: Fluxograma ASM PCPO


k_eq_2 = '0' / inc_k <= '1'
S1
en_calc <= '1'
k_eq_2 = '1'
S2
wr_en <= '1 '
clr_acc <= '1'
clr_k <= '1'
j_eq_2 = '0'/ inc_j <= '1'
j_eq_2 = '1' and i_eq_2 = '0'
/ clr_j = '1', inc_i <= '1'
start = '1 ' / clr_all <= '1'
j_eq_2 = '1' and i_eq_2 = '1'
/ clr_j = '1'
S0
S3
done <= '1 '
start = '0'
rst = '1'
start = '1'

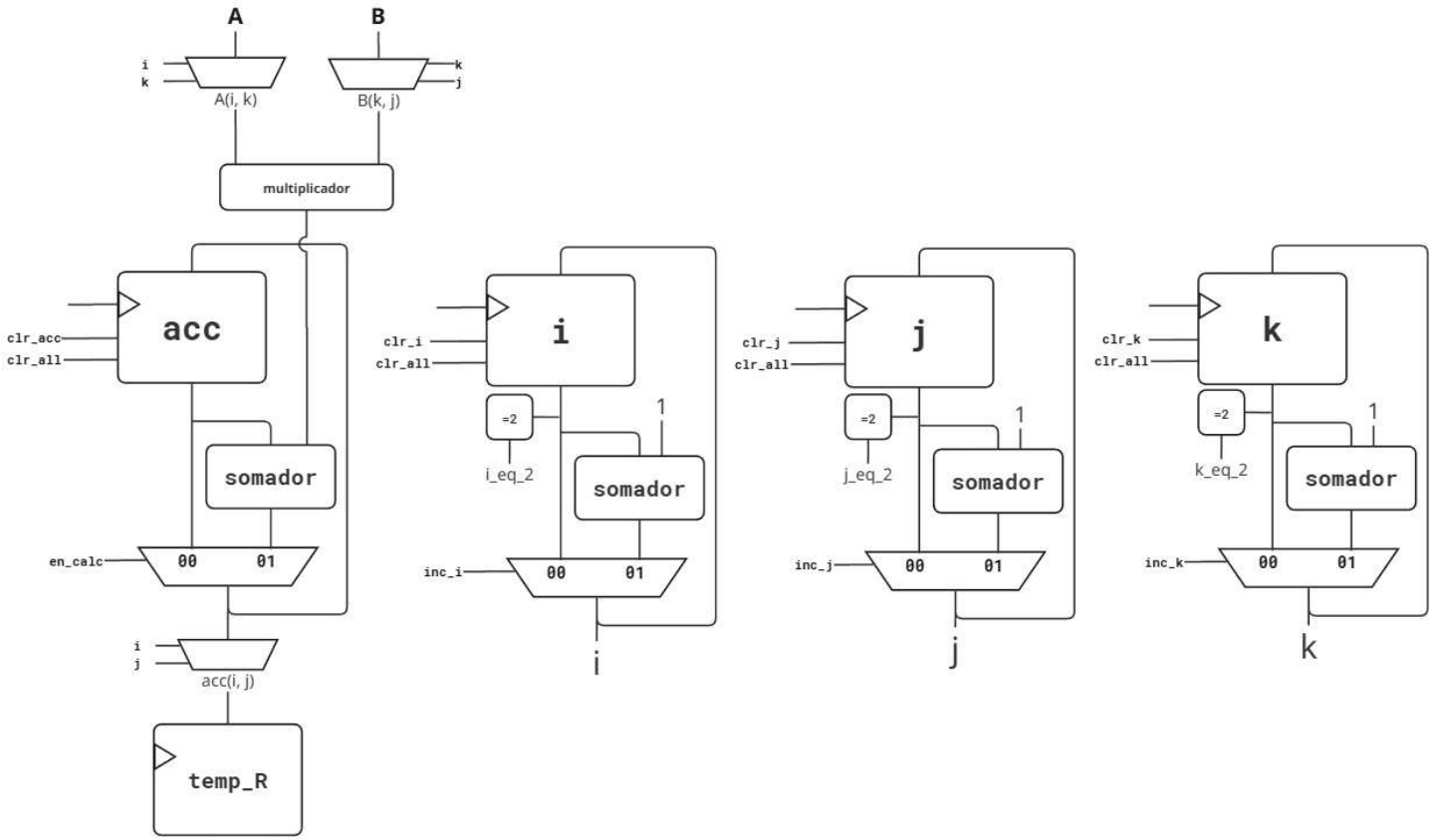


Figura: Parte Operativa

```
process(clk)
    begin
        if rising_edge(clk) then
            if clr_all = '1' then
                i <= 0; j <= 0; k <= 0; acc <= (others => '0');
            else
                if inc_k = '1' then k <= k + 1; end if;
                if clr_k = '1' then k <= 0;     end if;

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

                if inc_i = '1' then i <= i + 1; end if;
            end if;

            if clr_acc = '1' then
                acc <= (others => '0');
            elsif en_calc = '1' then
                acc <= acc + (matrizA(i, k) * matrizB(k, j));
            end if;

            if wr_en = '1' then
                matriz_R_interna(i, j) <= acc;
            end if;
        end if;
    end process;
    k_eq_2 <= '1' when k = 2 else '0';
    j_eq_2 <= '1' when j = 2 else '0';
    i_eq_2 <= '1' when i = 2 else '0';
    resul <= matriz_R_interna;
```

```
process(clk, rst)
begin
    if rst = '1' then
        estado_atual <= s0;
    elsif rising_edge(clk) then
        case estado_atual is
            when s0 =>
                if start = '1' then
                    estado_atual <= s1;
                end if;

            when s1 =>
                if k_eq_2 = '1' then
                    estado_atual <= s2;
                else
                    estado_atual <= s1;
                end if;

            when s2 =>
                if j_eq_2 = '0' then
                    estado_atual <= s1;
                else
                    if i_eq_2 = '0' then
                        estado_atual <= s1;
                    else
                        estado_atual <= s3;
                    end if;
                end if;
```

```
            when s3 =>
                if start = '0' then
                    estado_atual <= s0;
                end if;
        end case;
    end if;
end process;

--COMBINACIONAL PC
process(estado_atual, k_eq_2, j_eq_2, i_eq_2, start)
begin
    clr_all <= '0'; en_calc <= '0'; inc_k <= '0'; clr_k <= '0';
    inc_j <= '0'; clr_j <= '0'; inc_i <= '0';
    wr_en <= '0'; clr_acc <= '0'; done <= '0';

    case estado_atual is
        when s0 =>
            if start = '1' then clr_all <= '1'; end if;

        when s1 =>
            en_calc <= '1';
            if k_eq_2 = '0' then inc_k <= '1'; end if;

        when s2 =>
            wr_en   <= '1';
            clr_acc <= '1';
            clr_k   <= '1';
```

```
            if j_eq_2 = '0' then
                inc_j <= '1';
            else
                clr_j <= '1';
                if i_eq_2 = '0' then
                    inc_i <= '1';
                end if;
            end if;

        when s3 =>
            done <= '1';
    end case;
end process;
```

TEST.VHDL I


```
        library ieee;
entity tb_PCPO is
end tb_PCPO;

architecture tb of tb_PCPO is

    component PCPO
        port (clk     : in std_logic;
              rst     : in std_logic;
              start   : in std_logic;
              matrizA : in mat3x3_8bit;
              matrizB : in mat3x3_8bit;
              resul   : out mat3x3_16bit;
              done    : out std_logic);
    end component;

    signal clk     : std_logic := '0';
    signal rst     : std_logic := '0';
    signal start   : std_logic := '0';

    signal matrizA : mat3x3_8bit := (others => (others => (others => '0')));
    signal matrizB : mat3x3_8bit := (others => (others => (others => '0')));
    signal resul   : mat3x3_16bit;
    signal done    : std_logic;

    constant TbPeriod : time := 10 ns;
```

TEST.VHDL II


```
begin

    dut : PCPO
    port map (clk     => clk,
              rst     => rst,
              start   => start,
              matrizA => matrizA,
              matrizB => matrizB,
              resul   => resul,
              done    => done);

    process
    begin
        clk <= '0';
        wait for TbPeriod / 2;
        clk <= '1';
        wait for TbPeriod / 2;
    end process;

    stimuli : process
    begin
        rst <= '1';
        start <= '0';
        wait for 100 ns;
        rst <= '0';
        wait for 20 ns;

        -- CASO 1: A (1 a 9) x B (Identidade)
```

TEST.VHDL III


```
wait until falling_edge(clk);

matrizA(0,0) <= to_unsigned(1, 8); matrizA(0,1) <= to_unsigned(2, 8);
     matrizA(0,2) <= to_unsigned(3, 8);
matrizA(1,0) <= to_unsigned(4, 8); matrizA(1,1) <= to_unsigned(5, 8);
     matrizA(1,2) <= to_unsigned(6, 8);
matrizA(2,0) <= to_unsigned(7, 8); matrizA(2,1) <= to_unsigned(8, 8);
     matrizA(2,2) <= to_unsigned(9, 8);

matrizB(0,0) <= to_unsigned(1, 8); matrizB(0,1) <= to_unsigned(0, 8);
     matrizB(0,2) <= to_unsigned(0, 8);
matrizB(1,0) <= to_unsigned(0, 8); matrizB(1,1) <= to_unsigned(1, 8);
     matrizB(1,2) <= to_unsigned(0, 8);
matrizB(2,0) <= to_unsigned(0, 8); matrizB(2,1) <= to_unsigned(0, 8);
     matrizB(2,2) <= to_unsigned(1, 8);

wait for 20 ns;

-- Start Sincronizado
wait until falling_edge(clk);
start <= '1';
wait for TbPeriod; -- Pulso de 1 clock
start <= '0';

wait until done = '1';
wait for 50 ns;
```

TEST.VHDL IV


```
--A x B (Zero)

-- Reset entre casos
rst <= '1';
wait for 20 ns;
rst <= '0';

wait until falling_edge(clk);
matrizB <= (others => (others => (others => '0'))); -- Zera B

wait for 20 ns;

wait until falling_edge(clk);
start <= '1';
wait for TbPeriod;
start <= '0';

wait until done = '1';
wait for 50 ns;


--A (Tudo 2) x B (Identidade)

rst <= '1';
wait for 20 ns;
rst <= '0';

wait until falling_edge(clk);
```

TEST.VHDL V


```
        matrizA <= (others => (others => to_unsigned(2, 8)));

        matrizB(0,0) <= to_unsigned(1, 8); matrizB(0,1) <= to_unsigned(0, 8);
             matrizB(0,2) <= to_unsigned(0, 8);
        matrizB(1,0) <= to_unsigned(0, 8); matrizB(1,1) <= to_unsigned(1, 8);
             matrizB(1,2) <= to_unsigned(0, 8);
        matrizB(2,0) <= to_unsigned(0, 8); matrizB(2,1) <= to_unsigned(0, 8);
             matrizB(2,2) <= to_unsigned(1, 8);

        wait for 20 ns;

        wait until falling_edge(clk);
        start <= '1';
        wait for TbPeriod;
        start <= '0';

        wait until done = '1';
        wait for 50 ns;

        wait;
    end process;

end tb;
```

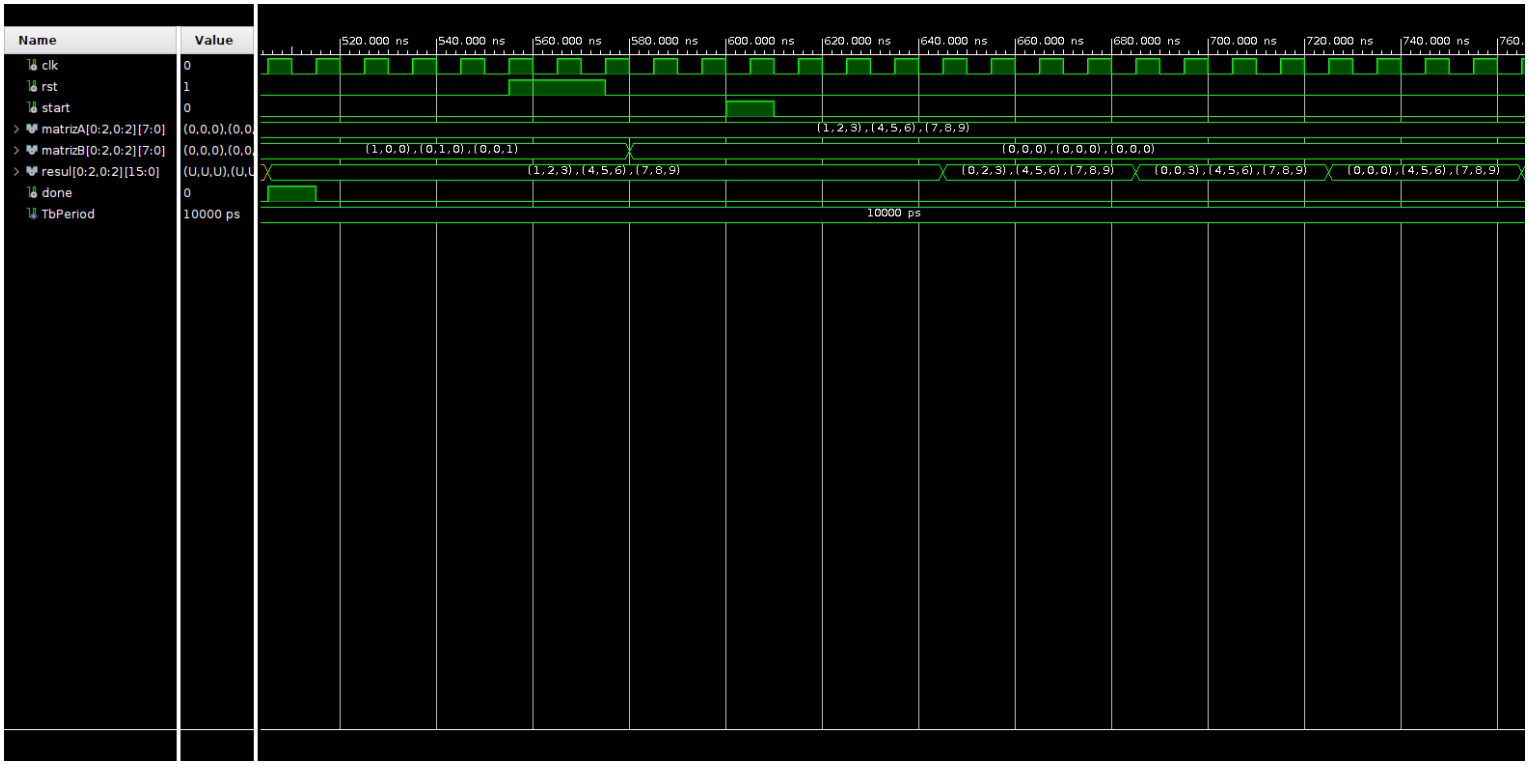


Figura: Multiplicação pela identidade

TESTE 2

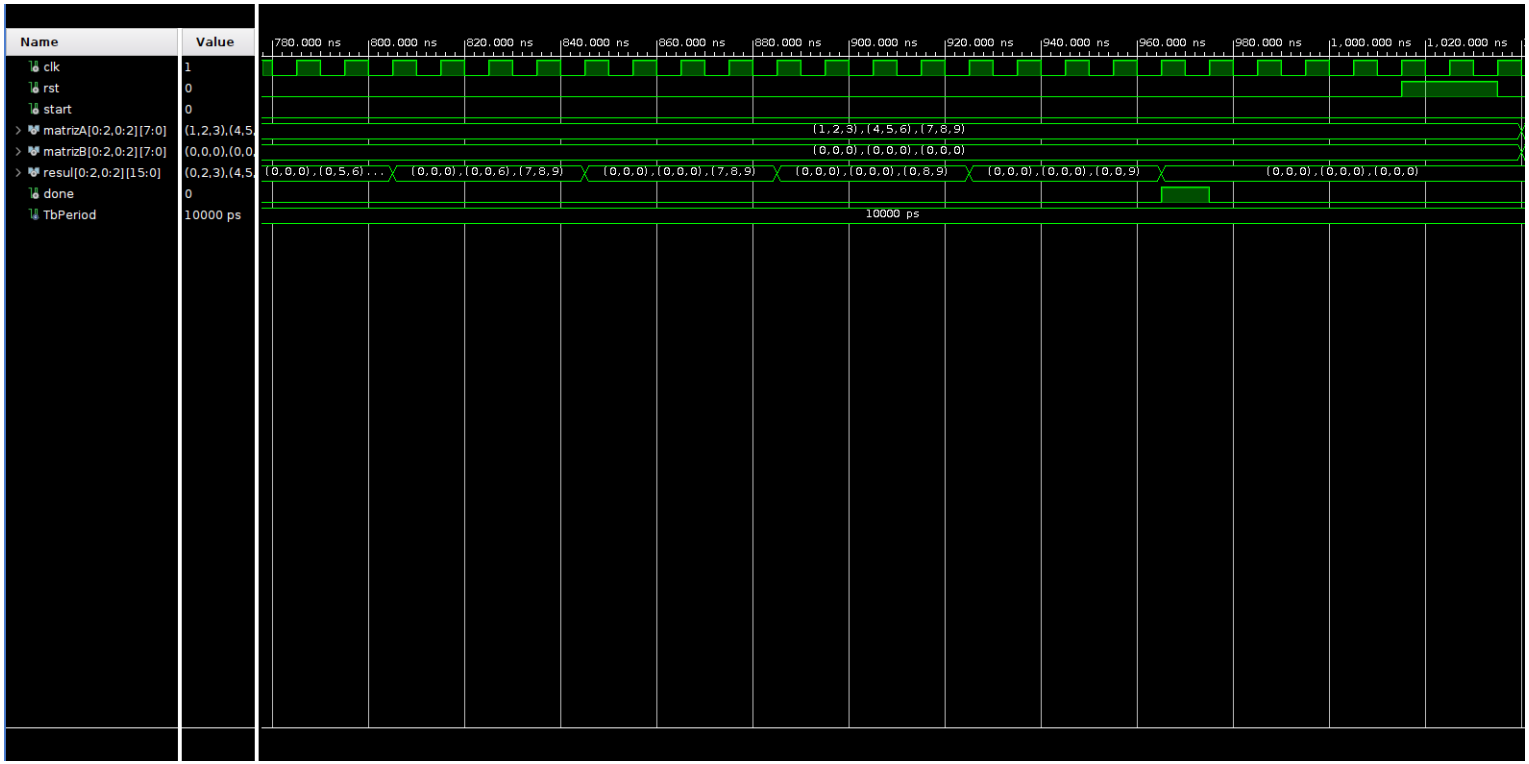


Figura: Multiplicação pela matriz vazia

TESTE 3

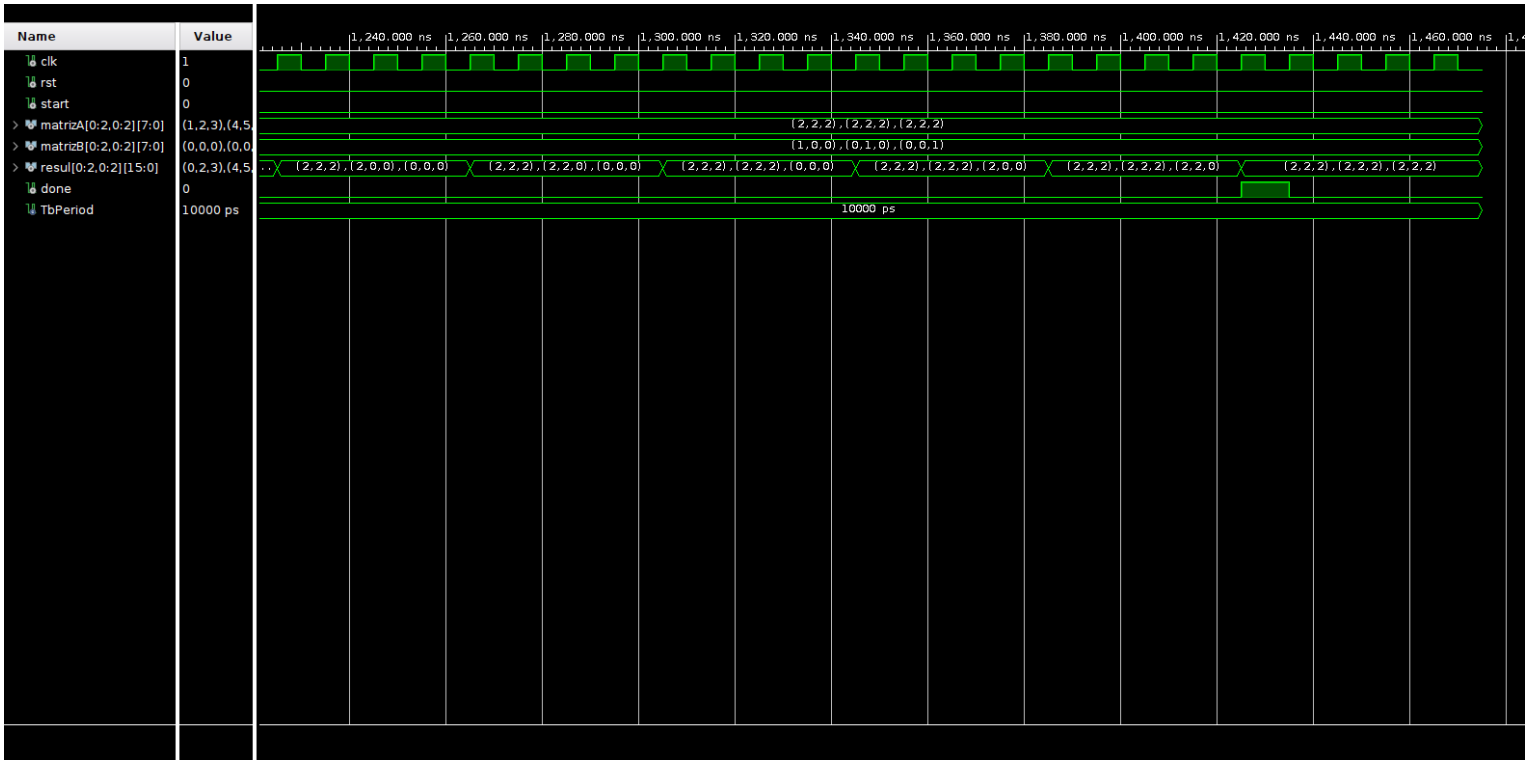
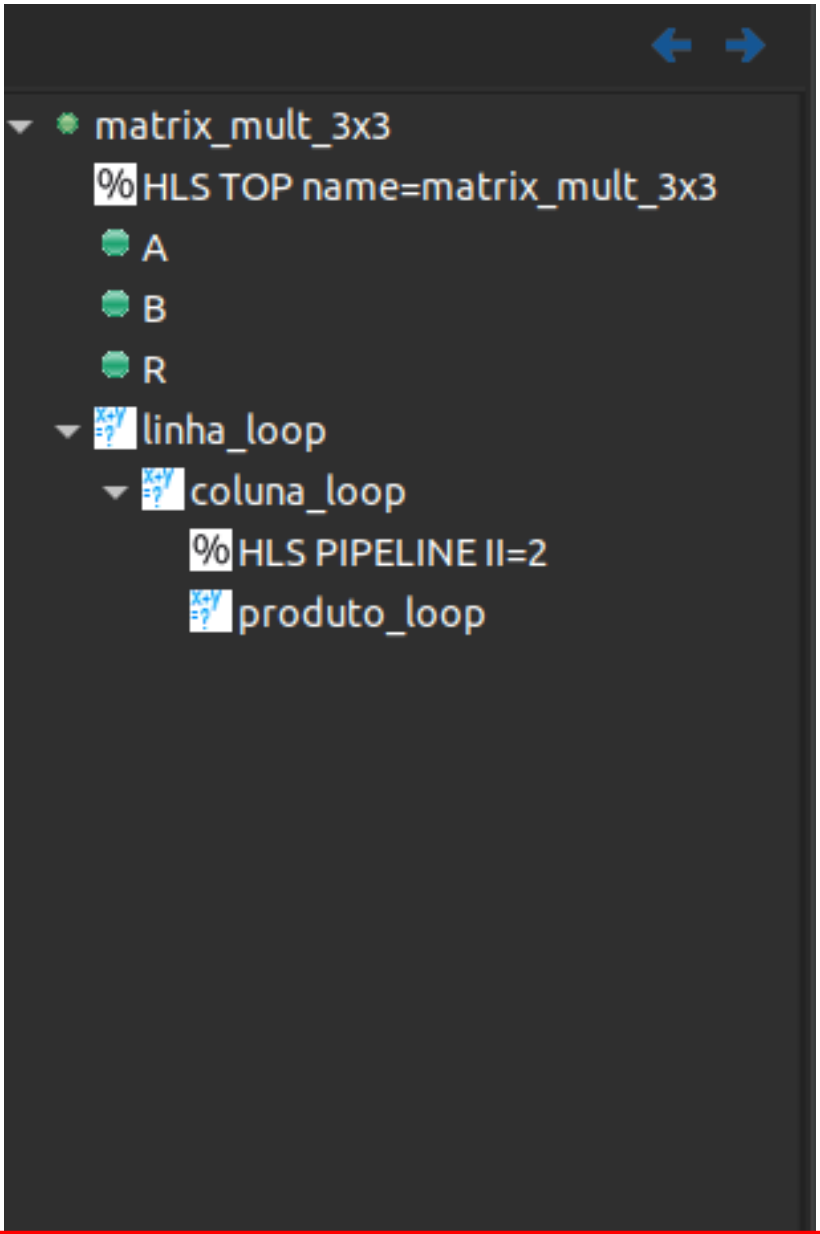
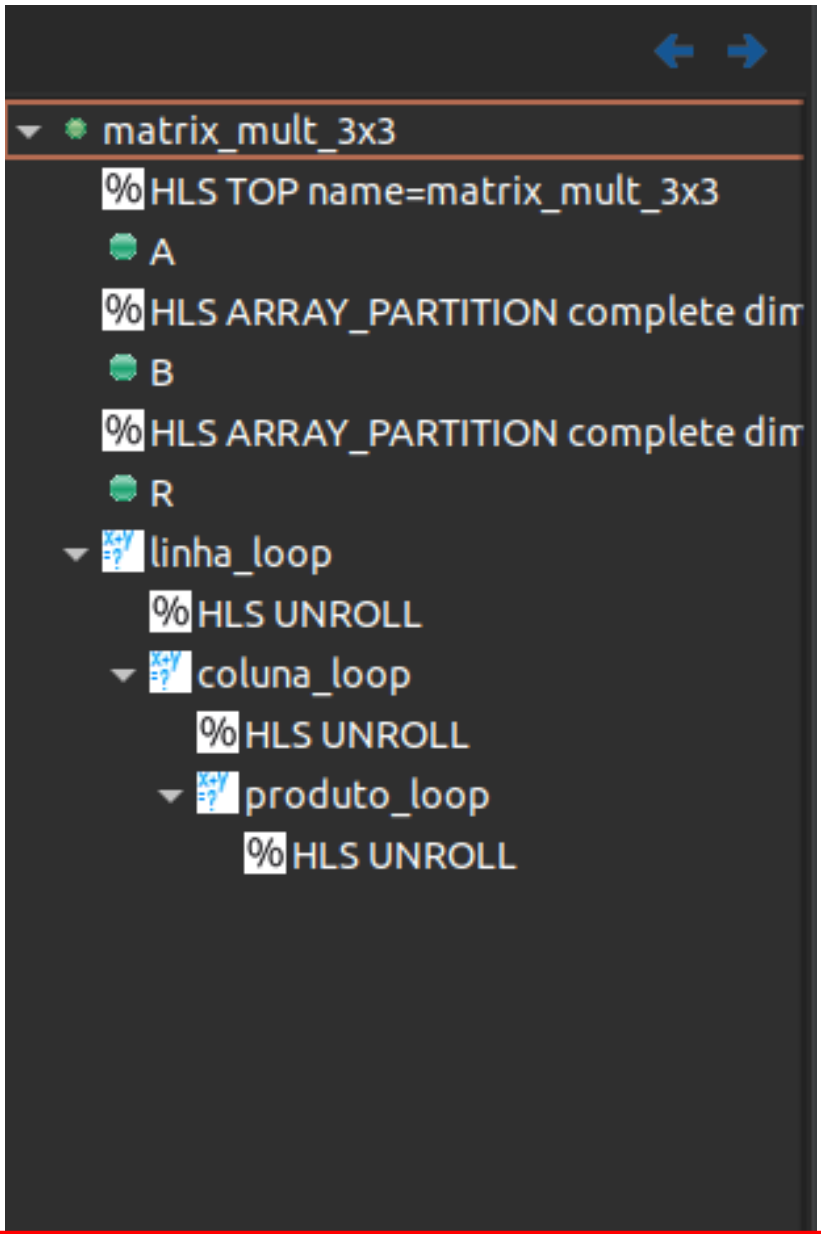


Figura: Matriz só com 2 X Identidade


matrix_mult_3x3
HLS TOP name=matrix_mult_3x3
A
B
R
linha_loop
coluna_loop
HLS PIPELINE II=2
produto_loop


matrix_mult_3x3
HLS TOP name=matrix_mult_3x3
A
HLS ARRAY_PARTITION complete dim
B
HLS ARRAY_PARTITION complete dim
R
linha_loop
HLS UNROLL
coluna_loop
HLS UNROLL
produto_loop
HLS UNROLL

Implementações	**Área (FFs, LUTs, DSP)**	**c.c.**	**Memória**	**pinos I/O (bits)**
Referência	46, 186, 1	160	0	54
Pipeline	106, 356, 2	23	0	79
Array	121, 528, 18	8	0	194
PC-PO	125,177,0	37	0	292

Tabela: Comparação de Resultados

- Memória = 0 porque matrizes 3x3 são pequenas demais para ocupar um bloco de RAM dedicado
- A otimização por Paralelismo Total atingiu o menor tempo de execução (8 ciclos), mas teve o maior custo de hardware
- A versão Pipeline apresentou o melhor equilíbrio. Obteve um Speedup em relação à referência com um aumento moderado de recursos, mantendo a interface de memória eficiente

Repositório:
`https://github.com/PedroSSoares/Tabalho-final-SD`

Enzo Lisbôa Peixoto e Pedro Scholz Soares
Instituto de Informática — UFRGS
elpeixoto@inf.ufrgs.brr
pedro.soares@inf.ufrgs.br