

# **Projeto 3 - Tópicos em Arquitetura e Hardware**

**Nome:** Cleidiana Reis dos Santos

**Matrícula:** 298254

## **Introdução**

A plataforma Litex é utilizada para a construção de SoCs (System-on-Chip) sobre FPGAs. FPGAs são dispositivos que podem ser reconfigurados para desempenhar diversas funções. O Litex simplifica a integração rápida de componentes essenciais como CPU, memória e periféricos. Este projeto teve como objetivo otimizar um programa no FPGA por meio dos seguintes passos:

- Escolher um programa para executar.
- Executar o programa medindo o desempenho.
- Acelerar o desempenho do programa e medir o desempenho novamente.

Todo o projeto é executado na placa Tang Nano 9k, utilizando como base o desenvolvimento do Projeto 2, portanto não será tratado comandos de desenvolvimento com a placa, compilação do programa, etc.

## **Passo 1 – Escolher um programa para executar.**

### **Multiplicação de matrizes:**

A multiplicação de matrizes é uma operação matemática que combina duas matrizes para resultar uma terceira matriz. Para que a multiplicação seja possível, o número de colunas da primeira matriz deve ser igual ao número de linhas da segunda matriz. O resultante na matriz produto é obtido pela soma dos produtos dos elementos da linha da primeira matriz com os elementos da coluna da segunda matriz.

Seja duas matrizes:

$$A (m \times n) = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B (n \times p) = \begin{bmatrix} 1 & 1 \\ 4 & 5 \end{bmatrix}$$

Cálculo dos elementos:

$$C_{11} = (2 * 1) + (2 * 4) = 10$$

$$C_{12} = (2 * 1) + (2 * 5) = 12$$

$$C_{21} = (3 * 1) + (4 * 4) = 19$$

$$C_{22} = (3 * 1) + (4 * 5) = 23$$

Resultado:

$$C (m \times p) = \begin{bmatrix} 10 & 12 \\ 19 & 23 \end{bmatrix}$$

O programa em C para realização da multiplicação de matrizes na FPGA ficou dessa maneira:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <irq.h>
#include <libbase/uart.h>
#include <libbase/console.h>
#include <generated/csr.h>

#define CLOCK_FREQUENCY 27000000 // 27 MHz

volatile uint32_t botao_user = 0;
#define SIZE 20

int A[SIZE][SIZE];
int B[SIZE][SIZE];
int C[SIZE][SIZE];

int main(void) {

#ifdef CONFIG_CPU_HAS_INTERRUPT
    irq_setmask(0);
    irq_setie(1);
#endif
```

```

uart_init();
uint32_t tempo_inicio, tempo_fim;
uint64_t ciclos;
uint32_t mili_segundos;
leds_out_write(0x00);
puts("\nAperte o botão para iniciar o teste\n");
botao_user = gpio_in_read();

//inicializa as matrizes
for (int i = 0; i < SIZE; i++){
    for (int j = 0; j < SIZE; j++) {
        A[i][j] = i;
        B[i][j] = j;
        C[i][j] = 0;
    }
}

while(botao_user){
    botao_user = gpio_in_read();
}

leds_out_write(0x3F);
printf("Iniciando multiplicação de matrizes...\n");

timer0_reload_write(0xFFFFFFFF);
timer0_load_write(0xFFFFFFFF);
timer0_en_write(1);

timer0_update_value_write(1);
tempo_inicio = timer0_value_read();

//multiplicação matrizes A * B
for (int i = 0; i < SIZE; i++){
    for (int j = 0; j < SIZE; j++){
        for (int k = 0; k < SIZE; k++){
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}

timer0_update_value_write(1);
tempo_fim = timer0_value_read();

ciclos = (uint64_t)(tempo_inicio - tempo_fim);
mili_segundos = (ciclos / (CLOCK_FREQUENCY/1000));

printf("\nTempo decorrido: %lld ciclos miliseg %ld\n", ciclos,
mili_segundos);

```

```
    leds_out_write(0x00);
    puts("\nFim teste\n");

    printf("Resultado Matriz C:\n");
    int PRINT_C = 0;
    if(PRINT_C){
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                printf("%3d ", C[i][j]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

## Passo 2 – Executar o programa medindo o desempenho.

Para cálculo do tempo de execução em ciclos e milissegundos do programa escolhido será utilizado o módulo de timer incorporado no Projeto 2 da disciplina. Para iniciar a execução do programa basta apertar o botão de usuário disponível na placa, também já desenvolvido no Projeto 2.

Após a gravação do código apresentado no Passo 1, o tempo resultante para multiplicação de duas matrizes de dimensão 20x20 foi de 99206 ciclos e 3ms. A Figura 1 mostra a saída resultante.

```

Aperte o botão para iniciar o teste

Iniciando multiplicação de matrizes...

Tempo decorrido: 99206 ciclos miliseg 3

Fim teste

Result Matrix C:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 20 40 60 80 100 120 140 160 180 200 220 240 260 280 300 320 340 360 380
0 40 80 120 160 200 240 280 320 360 400 440 480 520 560 600 640 680 720 760
0 60 120 180 240 300 360 420 480 540 600 660 720 780 840 900 960 1020 1080 1140
0 80 160 240 320 400 480 560 640 720 800 880 960 1040 1120 1200 1280 1360 1440 1520
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900
0 120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800 1920 2040 2160 2280
0 140 280 420 560 700 840 980 1120 1260 1400 1540 1680 1820 1960 2100 2240 2380 2520 2660
0 160 320 480 640 800 960 1120 1280 1440 1600 1760 1920 2080 2240 2400 2560 2720 2880 3040
0 180 360 540 720 900 1080 1260 1440 1620 1800 1980 2160 2340 2520 2700 2880 3060 3240 3420
0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800 3000 3200 3400 3600 3800
0 220 440 660 880 1100 1320 1540 1760 1980 2200 2420 2640 2860 3080 3300 3520 3740 3960 4180
0 240 480 720 960 1200 1440 1680 1920 2160 2400 2640 2880 3120 3360 3600 3840 4080 4320 4560
0 260 520 780 1040 1300 1560 1820 2080 2340 2600 2860 3120 3380 3640 3900 4160 4420 4680 4940
0 280 560 840 1120 1400 1680 1960 2240 2520 2800 3080 3360 3640 3920 4200 4480 4760 5040 5320
0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100 5400 5700
0 320 640 960 1280 1600 1920 2240 2560 2880 3200 3520 3840 4160 4480 4800 5120 5440 5760 6080
0 340 680 1020 1360 1700 2040 2380 2720 3060 3400 3740 4080 4420 4760 5100 5440 5780 6120 6460
0 360 720 1080 1440 1800 2160 2520 2880 3240 3600 3960 4320 4680 5040 5400 5760 6120 6480 6840
0 380 760 1140 1520 1900 2280 2660 3040 3420 3800 4180 4560 4940 5320 5700 6080 6460 6840 7220

```

Figura 1: Tempo inicial da execução do programa.

## Passo 3 – Acelerar o desempenho do programa e medir o desempenho novamente

A fim de reduzir o número de ciclos é proposto a divisão da matriz em matrizes menores (submatrizes).

Seja duas matrizes:

A (m x n) =  $\begin{bmatrix} 2 & 2 & 1 & 1 \\ 3 & 4 & 4 & 4 \\ 2 & 2 & 1 & 1 \\ 3 & 4 & 4 & 4 \end{bmatrix}$

B (n x p) =  $\begin{bmatrix} 3 & 3 & 2 & 2 \\ 2 & 5 & 5 & 5 \\ 3 & 3 & 2 & 2 \\ 2 & 3 & 3 & 3 \end{bmatrix}$

As matrizes A e B pode ser divida em 4 matrizes de 2x2, ficando da seguinte forma:

A00 =  $\begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$  A01 =  $\begin{bmatrix} 1 & 1 \\ 4 & 4 \end{bmatrix}$  A10 =  $\begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$  A11 =  $\begin{bmatrix} 1 & 1 \\ 4 & 4 \end{bmatrix}$

$$\begin{array}{cc} B_{00} = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} & B_{01} = \begin{bmatrix} 2 & 2 \\ 5 & 5 \end{bmatrix} & B_{10} = \begin{bmatrix} 3 & 3 \\ 2 & 3 \end{bmatrix} & B_{11} = \begin{bmatrix} 2 & 2 \\ 3 & 3 \end{bmatrix} \end{array}$$

O cálculo da matriz resultante C, pode ser feito por blocos, utilizando os submatrizes menores das matrizes A e B:

- $C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10}$ :

$$A_{00} \times B_{00} = \begin{bmatrix} 2 \times 3 + 2 \times 2 & 2 \times 3 + 2 \times 5 \end{bmatrix} = \begin{bmatrix} 10 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 3 \times 3 + 4 \times 2 & 3 \times 3 + 4 \times 5 \end{bmatrix} = \begin{bmatrix} 17 & 29 \end{bmatrix}$$

$$A_{01} \times B_{10} = \begin{bmatrix} 1 \times 3 + 1 \times 2 & 1 \times 3 + 1 \times 3 \end{bmatrix} = \begin{bmatrix} 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 4 \times 3 + 4 \times 2 & 4 \times 3 + 4 \times 3 \end{bmatrix} = \begin{bmatrix} 20 & 24 \end{bmatrix}$$

$$C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10} = \begin{bmatrix} 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 37 & 53 \end{bmatrix}$$

- $C_{01} = A_{00} \times B_{01} + A_{01} \times B_{11} = \begin{bmatrix} 19 & 19 \end{bmatrix}$

$$\begin{bmatrix} 46 & 46 \end{bmatrix}$$

- $C_{10} = A_{10} \times B_{00} + A_{11} \times B_{10} = \begin{bmatrix} 15 & 22 \end{bmatrix}$

$$\begin{bmatrix} 37 & 53 \end{bmatrix}$$

- $C_{11} = A_{10} \times B_{01} + A_{11} \times B_{11} = \begin{bmatrix} 19 & 19 \end{bmatrix}$

$$\begin{bmatrix} 46 & 46 \end{bmatrix}$$

Resultado:

$$C = \begin{bmatrix} 15 & 22 & 19 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 37 & 53 & 46 & 46 \end{bmatrix}$$

$$\begin{bmatrix} 15 & 22 & 19 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 37 & 53 & 46 & 46 \end{bmatrix}$$

O programa em C com essa otimização para realização da multiplicação de matrizes na FPGA ficou dessa maneira:

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <irq.h>
#include <libbase/uart.h>
#include <libbase/console.h>
#include <generated/csr.h>

#define CLOCK_FREQUENCY 27000000 // 27 MHz

volatile uint32_t botao_user = 0;
#define SIZE 20
#define SUB_SIZE 2

int A[SIZE][SIZE];
int B[SIZE][SIZE];
int C[SIZE][SIZE];

int main(void) {

#ifdef CONFIG_CPU_HAS_INTERRUPT
    irq_setmask(0);
    irq_setie(1);
#endif
    uart_init();
    uint32_t tempo_inicio, tempo_fim;
    uint64_t ciclos;
    uint32_t mili_segundos;
    leds_out_write(0x00);
    puts("\nAperte o botão para iniciar o teste\n");
    botao_user = gpio_in_read();

    //inicializa as matrizes
    for (int i = 0; i < SIZE; i++){
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = i;
            B[i][j] = j;
            C[i][j] = 0;
        }
    }

    while(botao_user){
        botao_user = gpio_in_read();
    }

    leds_out_write(0x3F);

```

```

printf("Iniciando multiplicação de matrizes...\n");

timer0_reload_write(0xFFFFFFFF);
timer0_load_write(0xFFFFFFFF);
timer0_en_write(1);

timer0_update_value_write(1);
tempo_inicio = timer0_value_read();

//percorre todos os blocos SUB_SIZExSUB_SIZE da matriz C
for (int bi = 0; bi < SIZE; bi += SUB_SIZE) {
    for (int bj = 0; bj < SIZE; bj += SUB_SIZE) {
        int Cblk[SUB_SIZE][SUB_SIZE] = {0};

        //soma dos produtos Ablk * Bblk
        for (int bk = 0; bk < SIZE; bk += SUB_SIZE) {
            int Ablk[SUB_SIZE][SUB_SIZE];
            int Bblk[SUB_SIZE][SUB_SIZE];

            //montagem da submatriz A
            for (int i = 0; i < SUB_SIZE; i++) {
                for (int j = 0; j < SUB_SIZE; j++) {
                    Ablk[i][j] = A[bi + i][bk + j];
                }
            }

            //montagem da submatriz B
            for (int i = 0; i < SUB_SIZE; i++) {
                for (int j = 0; j < SUB_SIZE; j++) {
                    Bblk[i][j] = B[bk + i][bj + j];
                }
            }

            //multiplicação submatrizes Ablk * Bblk
            for (int i = 0; i < SUB_SIZE; i++) {
                for (int j = 0; j < SUB_SIZE; j++) {
                    for (int k = 0; k < SUB_SIZE; k++) {
                        Cblk[i][j] += Ablk[i][k] * Bblk[k][j];
                    }
                }
            }
        }

        //copia o resultado Cblk para C
        for (int i = 0; i < SUB_SIZE; i++) {
            for (int j = 0; j < SUB_SIZE; j++) {
                C[bi + i][bj + j] = Cblk[i][j];
            }
        }
    }
}

```



```

    }
}

timer0_update_value_write(1);
tempo_fim = timer0_value_read();

ciclos = (uint64_t)(tempo_inicio - tempo_fim);
mili_segundos = (ciclos / (CLOCK_FREQUENCY/1000));

printf("\nTempo decorrido: %lld ciclos miliseg %ld\n", ciclos,
mili_segundos);

leds_out_write(0x00);
puts("\nFim teste\n");

printf("Resultado Matriz C:\n");
int PRINT_C = 1;
if(PRINT_C){
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%3d ", C[i][j]);
        }
        printf("\n");
    }
}
return 0;
}

```

Segue Figura 2 resultado da separação em blocos 2x2.

```

Aperte o botão para iniciar o teste

Iniciando multiplicação de matrizes...

Tempo decorrido: 39653 ciclos miliseg 1

Fim teste

Result Matrix C:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 20 40 60 80 100 120 140 160 180 200 220 240 260 280 300 320 340 360 380
0 40 80 120 160 200 240 280 320 360 400 440 480 520 560 600 640 680 720 760
0 60 120 180 240 300 360 420 480 540 600 660 720 780 840 900 960 1020 1080 1140
0 80 160 240 320 400 480 560 640 720 800 880 960 1040 1120 1200 1280 1360 1440 1520
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900
0 120 240 360 480 600 720 840 960 1080 1200 1320 1440 1560 1680 1800 1920 2040 2160 2280
0 140 280 420 560 700 840 980 1120 1260 1400 1540 1680 1820 1960 2100 2240 2380 2520 2660
0 160 320 480 640 800 960 1120 1280 1440 1600 1760 1920 2080 2240 2400 2560 2720 2880 3040
0 180 360 540 720 900 1080 1260 1440 1620 1800 1980 2160 2340 2520 2700 2880 3060 3240 3420
0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800 3000 3200 3400 3600 3800
0 220 440 660 880 1100 1320 1540 1760 1980 2200 2420 2640 2860 3080 3300 3520 3740 3960 4180
0 240 480 720 960 1200 1440 1680 1920 2160 2400 2640 2880 3120 3360 3600 3840 4080 4320 4560
0 260 520 780 1040 1300 1560 1820 2080 2340 2600 2860 3120 3380 3640 3900 4160 4420 4680 4940
0 280 560 840 1120 1400 1680 1960 2240 2520 2800 3080 3360 3640 3920 4200 4480 4760 5040 5320
0 300 600 900 1200 1500 1800 2100 2400 2700 3000 3300 3600 3900 4200 4500 4800 5100 5400 5700
0 320 640 960 1280 1600 1920 2240 2560 2880 3200 3520 3840 4160 4480 4800 5120 5440 5760 6080
0 340 680 1020 1360 1700 2040 2380 2720 3060 3400 3740 4080 4420 4760 5100 5440 5780 6120 6460
0 360 720 1080 1440 1800 2160 2520 2880 3240 3600 3960 4320 4680 5040 5400 5760 6120 6480 6840
0 380 760 1140 1520 1900 2280 2660 3040 3420 3800 4180 4560 4940 5320 5700 6080 6460 6840 7220

```

**Figura 2: Tempo de execução após aceleração**

O tempo resultante para multiplicação de duas matrizes de dimensão 20x20 foi de 39653 ciclos e 1ms, para divisão de blocos (submatrizes) de tamanho 2x2. Foram avaliadas diferentes configurações de tamanho de bloco (SUB\_SIZE) trocando apenas o `define` no código apresentado. A Tabela 1 mostra o resultado de acordo com a variação de divisão de blocos. Apesar da expectativa inicial de que blocos maiores resultam em maior desempenho devido a redução no número total de blocos processados, foi observado que a configuração com SUB\_SIZE = 2 apresentou o melhor desempenho prático.

Dimensão (SUB_SIZE)	Quant. de ciclos	Tempo de execução em milissegundos
2	39653	1
4	182844	6
5	157220	5

**Tabela 1: Comparação de Tempo de Execução (Simulador x Tang Nano 9K)**

## Conclusão final:

O Projeto apresentou a comparação entre os testes de multiplicação de matrizes utilizando a multiplicação tradicional e divisão por blocos. Após a otimização, a divisão em blocos 2x2, o tempo de execução foi reduzido para aproximadamente 1/3 do tempo com o algoritmo inicial, foi de 99206 ciclos para 39653 ciclos de clock. Blocos com outros tamanhos não obtiveram resultado satisfatório.

Essa melhoria de desempenho se deve, principalmente, à redução dos tamanho das matrizes na multiplicação.

Além disso, pode destacar a flexibilidade para modificar o código e realizar experimentos com diferentes abordagens, bem como a facilidade de medição precisa do tempo de execução, viabilizada pelo módulo Timer desenvolvido anteriormente no Projeto 2.

O código desse projeto está disponível em um repositório no GitHub: <https://github.com/ic-unicamp/mo801-2025s1-p3-cleidiana>.