UNIVERSIDADE FEDERAL DE SÃO CARLOS
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Arquiteturas de Alto Desempenho

Implementação de arquitetura mesh para stream de vídeo

Guilherme Gomes Arcencio - 769731 - Engenharia de Computação
Guilherme Silva Castro - 769763 - Ciência da Computação

1 Introdução

Neste experimento, foi construída uma arquitetura mesh de processadores implementando filtros de máximos e mínimos para remoção de ruído em uma imagem ruidosa. A arquitetura mesh foi implementada em hardware, utilizando uma FPGA para realizar o processamento em tempo real e exibir a imagem através de um controlador VGA.

2 Desenvolvimento teórico

2.1 Filtro de Dilatação

O efeito esperado do filtro de dilatação [1] é o de ampliar uma figura a partir do aumento de suas bordas, em pixels. Se tratando da nossa imagem em preto e branco, precisamos apenas replicar um pixel de uma só cor quando for o caso necessário. Para isso verificaremos para determinado pixel se na sua vizinhança existe algum outro pixel com valor verdadeiro (que na nossa aplicação se traduz em um pixel branco na tela). Essa transformação será feita em hardware, pixel a pixel, com uma comparação OR com os pixels vizinhos.

2.2 Filtro de Erosão

O efeito esperado desse filtro é reduzir o ruído, ou seja, pontos da imagem que não condizem com o seu redor. Aqui, se um pixel possui um valor (ou uma cor) mas os pixels adjacentes se diferem deste, ele será convertido para o valor de seus vizinhos. Essa transformação será feita de forma parecida com a anterior, mas fazendo uma comparação AND entre os adjacentes.

3 Desenvolvimento prático

Para implementar e testar o código desenvolvido, foi utilizado um kit FPGA DE10 da Altera juntamente com o software Quartus. Os circuitos foram criados a partir de uma mistura de diagrama de blocos e código em Verilog.

Primeiramente, foi utilizado um módulo controlador de VGA para exibir a imagem original ruidosa no monitor. Dado que a resolução do VGA é 640x480 pixels e a imagem tem tamanho 256x128, foi criado um módulo de janela que garante que o pixel sendo calculado faz parte da imagem. Esta, por sua vez foi carregada em uma ROM que é lida de acordo com as coordenadas geradas pelo VGA e concatenadas em outro módulo. As implementações em Verilog do módulo de janela e do módulo concatenador são respectivamente mostradas a seguir.

```
module janela(
        input [9:0] h,
        input [9:0] v,
        output s
);
        assign s = h < 10'd256 && v < 10'd128;
endmodule

module conc(
        input [9:0] a,
        input [9:0] b,
        output [14:0] s
);
        assign s = {a[6:0], b[7:0]};
endmodule
```

Em seguida, foram desenvolvidos os módulos dos filtros de erosão e dilatação. Ambos apresentam a mesma estrutura, guardando pixels em um *buffer* e produzindo como saída o pixel central de uma janela filtrada 3x3. Como a erosão é uma operação de mínimo e a dilatação uma operação de máximo, a diferença está no uso das operações AND e OR, respectivamente. O seu código em Verilog é mostrado a seguir.

```
module erosao(
        input clk,
        input img,
        output img_out
);
        reg [0:514] buffer;

        always @(posedge clk) begin
                buffer <= {buffer[1:514], img};
        end

        assign img_out = buffer[0]   & buffer[1]   & buffer[2]   &
                         buffer[256] & buffer[257] & buffer[258] &
                         buffer[512] & buffer[513] & buffer[514];
endmodule
```

```
16
17 module dilatacao(
18         input clk,
19         input img,
20         output img_out
21 );
22         reg [0:514] buffer;
23
24         always @(posedge clk) begin
25                 buffer <= {buffer[1:514], img};
26         end
27
28         assign img_out = buffer[0]   | buffer[1]   | buffer[2]   |
29                          buffer[256] | buffer[257] | buffer[258] |
30                          buffer[512] | buffer[513] | buffer[514];
31 endmodule
```

Esses processadores de máximo e mínimo foram conectados em uma arquitetura mesh, cujo código é mostrado a seguir.

```
1  module mesh(
2          input clk,
3          input img,
4          output img_out
5  );
6          wire [3:0] proc_in [3:0];
7          wire [3:0] proc_out [3:0];
8
9          assign proc_in[0][0] = img;
10         assign img_out = proc_out[3][3];
11
12         genvar i, j;
13
14         generate
15                 for (i = 0; i < 4; i = i + 1) begin : lines
16                         for (j = 0; j < 4; j = j + 1) begin :
                           ↪ columns
17                                 if ((i + j) % 2 == 0) begin
18                                         erosao proc_e (
```

```
19                                              .clk (clk),
20                                              .img (proc_in[i][j]),
21                                              .img_out
                                                ↪  (proc_out[i][j])
22                                      );
23                                  end else begin
24                                      dilatacao proc_d (
25                                              .clk(clk),
26                                              .img(proc_in[i][j]),
27                                              .img_out(proc_out[i][j])
28                                      );
29                                  end
30                          end
31                  end
32          endgenerate
33
34          // Tabela de roteamento
35          assign proc_in[1][0] = proc_out[0][0];
36          assign proc_in[1][1] = proc_out[1][0];
37          assign proc_in[1][2] = proc_out[1][1];
38          assign proc_in[2][2] = proc_out[1][2];
39          assign proc_in[2][3] = proc_out[2][2];
40          assign proc_in[3][3] = proc_out[2][3];
41
42  endmodule
```

Os 16 núcleos ficaram dispostos como na Figura 1, com uma tabela de roteamento responsável por definir o caminho dos pixels.

Por fim, todos os módulos foram conectados em um diagrama de blocos e os *pins* de entrada e saída da placa foram atribuídos aos sinais adequados. Um *switch* foi designado como chave de seleção entre a imagem original e a imagem filtrada. O diagrama de blocos completo é mostrado na Figura 2.

4 Discussão dos resultados

A exibição da imagem ruidosa é mostrada na Figura 3. A imagem sem ruído, resultante da aplicação dos filtros, é mostrada na Figura 4. A exibição da imagem filtrada revela que a construção da arquitetura mesh foi exitosa e que o circuito implementado funciona como o planejado. No entando, a imagem ficou levemente

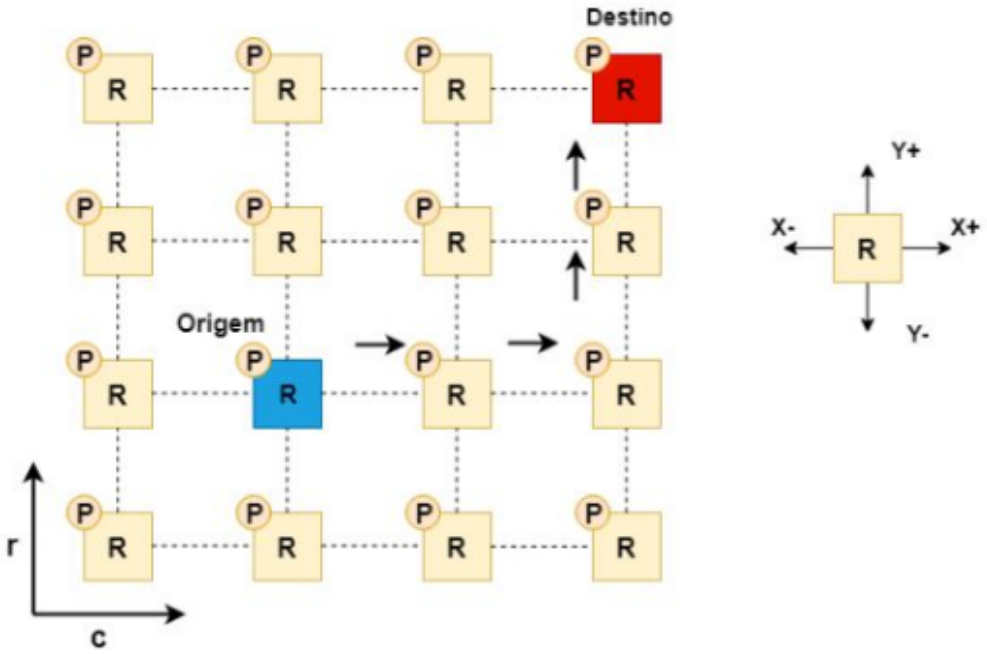

Figura 1: Diagrama representativo da arquitetura mesh.

deslocada, já que cada processador leva alguns ciclos para armazenar dados suficientes em *buffer*.

5 Conclusões

Foi implementada com sucesso uma arquitetura de processadores mesh em uma FPGA para a filtragem de ruído em tempo real de um stream de vídeo. A utilização de filtros de erosão e dilatação, baseados em operações lógicas AND e OR, respectivamente, provou ser eficaz na remoção de artefatos da imagem original, validando o funcionamento do circuito conforme planejado. Embora a exibição da imagem filtrada tenha confirmado o êxito da arquitetura, observou-se um leve deslocamento na imagem resultante, atribuído à latência introduzida pelo armazenamento de dados em buffer em cada um dos processadores da arquitetura.

Referências

1 LUGOD, Cyril Benedict. **Filtering and Morphological Operations**. 2023. Disponível em: <`https://cyrillugod.medium.com/filtering-and-morphological-operations-990662c5bd59`>. Acesso em: 22 mai. 2025.

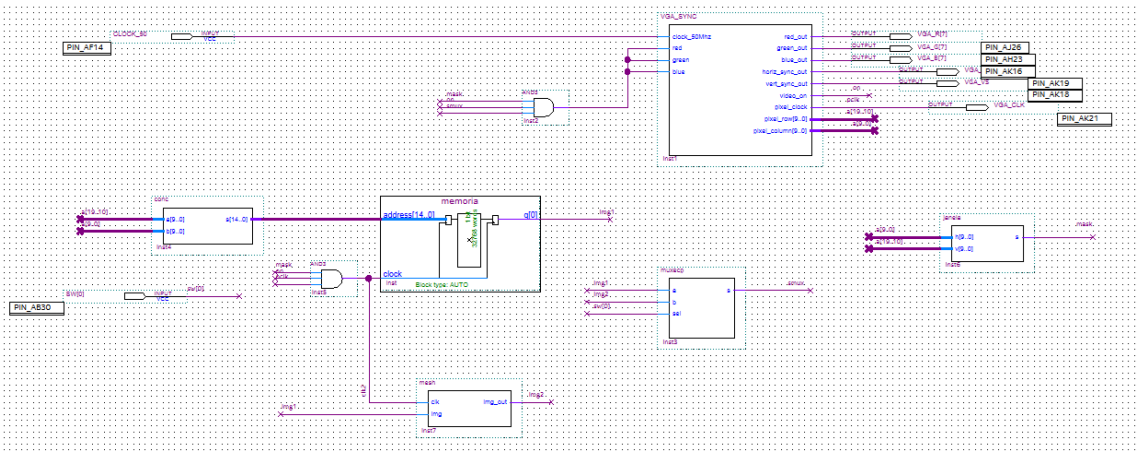

Figura 2: Diagrama de blocos do circuito de processamento e exibição da imagem binária com arquitetura mesh.

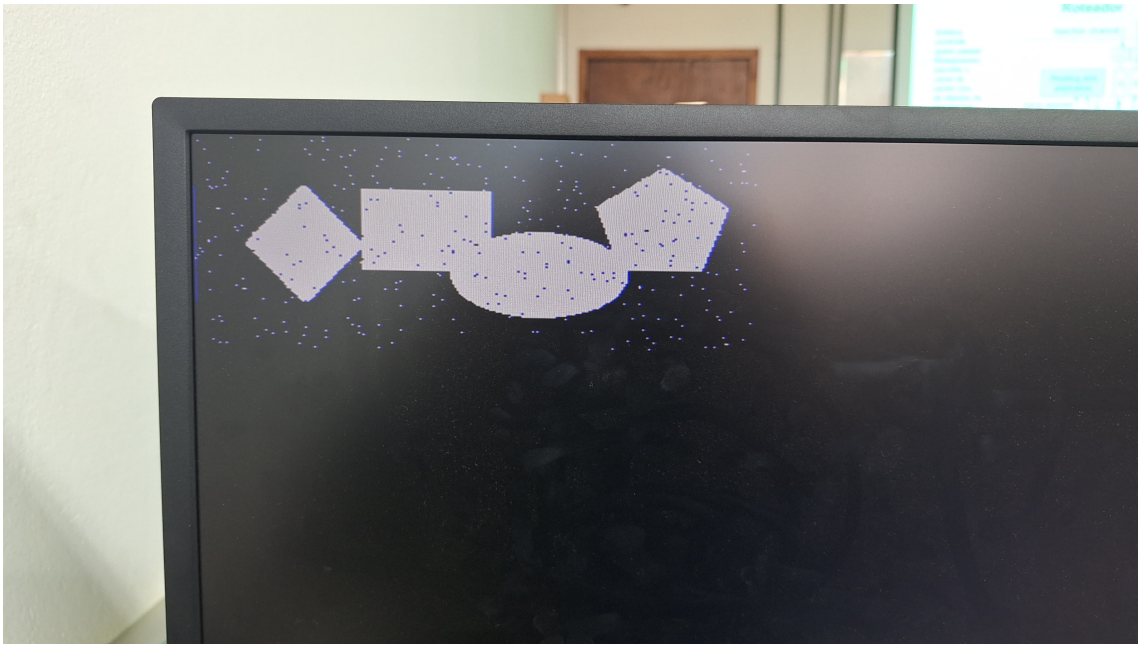

Figura 3: Imagem original, com ruído, sendo mostrada no monitor através do VGA.

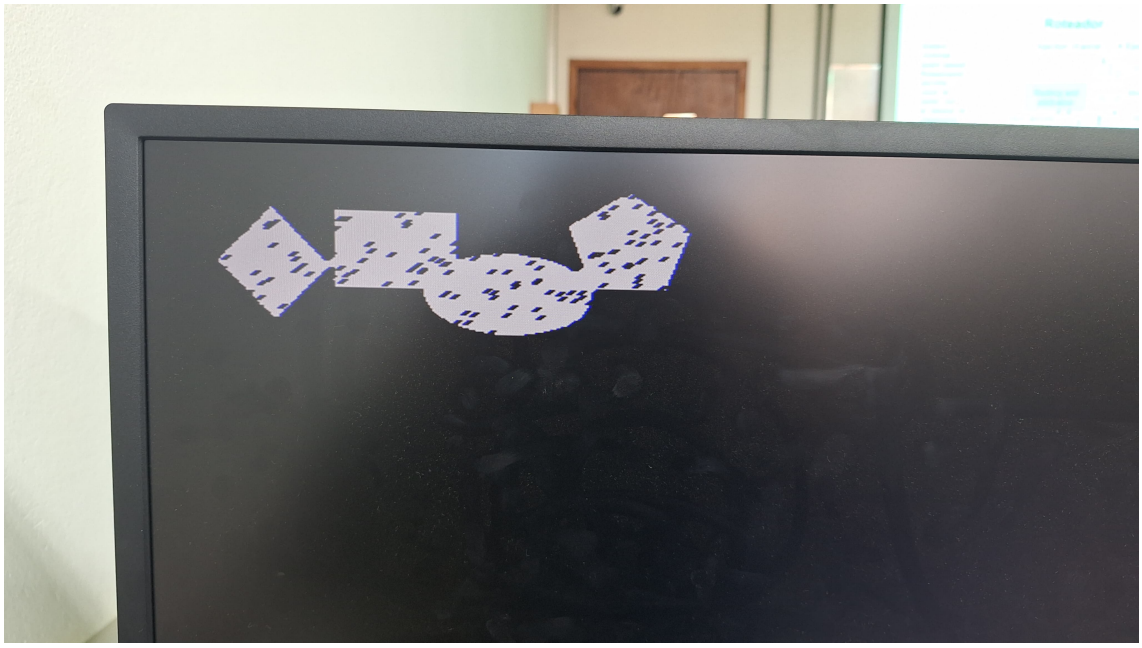

Figura 4: Imagem filtrada sendo mostrada no monitor através do VGA.