Relatório da Lista 2 - Projeto de Sistemas em Chip

Jessé Barreto de Barros
Matrícula: 17/0067033
Sistemas Mecatrônicos
Universidade de Brasília

I. Exercício 1 - Filtro de Sobel

a) Usando como exemplo a arquitetura RTL desenvolvida em sala de aula para o filtro Sobel $3x3$, implemente em VHDL um filtro de Sobel $5x5$. Implemente o filtro com os K_x para detectar bordas verticais e K_y para detectar bordas na direção horizontal. Use um multiplexador e um pino de entrada para selecionar o *Kernel* desejado. Parametrize o código de forma que a arquitetura possa ser testada para diferentes tamanhos de imagens e diferentes tamanhos de pixel.

b) Modifique o arquivo de *testbench* desenvolvido em sala de aula de forma que possa realizar simulações comportamentais do circuito que implementa o filtro de Sobel $5x5$. As simulações devem ser realizadas com base no último número da matrícula (17/0067033). No caso, as imagens são board.tif com resolução de $300x300$ com tamanho de pixel de 8 bits e coins.png com resolução de $250x250$ e com tamanho de pixel de 10 pixel.

c) Com base nas simulações estime a latência e o *troughput* do circuíto. Considere que cada ciclo de relógio é de 100 MHz (período de 10 ns).

d) Sintetize o circuito e coloque em uma tabela o reporte de síntese com o consumo de recursos de hardware (LUTs, Flip-flops, DSPs e BRAMs).

A. Respostas

a) O código desenvolvido utilizou as seguintes matrizes como *Kernel* para encontrar bordas verticais K_x e bordas horizontais K_y, conforme a equação 1 e a equação 2.

$$Z_x = \begin{bmatrix} 1 & 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 \\ 2 & 2 & 0 & -2 & -2 \\ 1 & `1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 \end{bmatrix} \tag{1}$$

$$Z_y = \begin{bmatrix} 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -2 & -1 & -1 \\ -1 & -1 & -2 & -1 & -1 \end{bmatrix} \tag{2}$$

O componente *top-module* recebe como entrada os pixeis da imagem e como saída os *pixel* da imagem resultante após a aplicação do filtro de Sobel, além de um sinal de controle para indicar o fim da aplicação do filtro e um sinal de ativação para indicar que a operação possui o número de pixels o bastante para iniciar além do sinal de *clock* para a sincronização e o *reset* para reiniciar o sistema.

O funcionamento da arquitetura funciona através da contagem do número de pixels o bastante para aplicação do *kernel* do filtro no canto superior esquerdo da imagem e com a máscara aplica a convolução pixel a pixel da imagem e da máscara.

Com o intuito de generalizar a criação e a aplicação de um *kernel* de diferentes tamanhos o processo de convolução foi criado utilizado laços generate do VHDL.

b) Com as devidas modificações do arquivo de *testbench* os seguintes arquivos foram obtidos para as simulações.

Para a simulação do arquivo board.tif de 8 bits e resolução de 300x300, vide Figura 1, temos os resultados presentes na Figura 2 para o sobel vertical e na Figura 3 os resultados da aplicação do sobel horizontal.

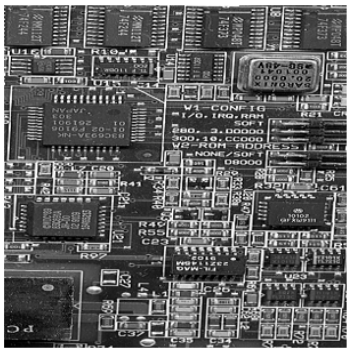

Figura 1. Figura utilizada na operação do filtro de Sobel para uma máscara de dimensão 5x5.

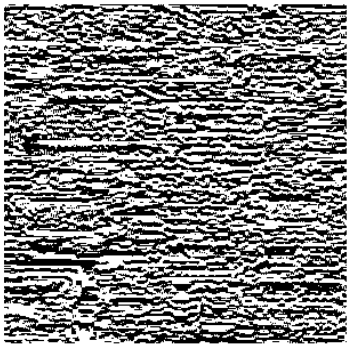

Figura 2. Resultado da aplicação do Sobel vertical.

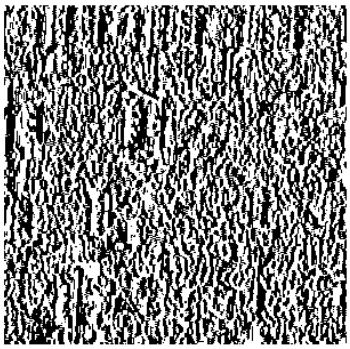

Figura 3. Resultado da aplicação do Sobel horizontal.

Devido a baixa resolução a visualização das bordas além do tamanho da máscara tornam a visualização das bordas sujeita a ruídos e também porque a imagem possui diversas linhas horizontais e verticais cujas bordas ”poluem”os resultados. No entanto, é possível visualizar as bordas correspondentes nas seguintes direções.

Para a simulação do arquivo coins.png de 10 bits e resolução de 250x250, vide Figura 4, temos os resultados presentes na Figura 2 para o Sobel vertical e na Figura 3 para a aplicação do Sobel horizontal.

Figura 4. Figura utilizada na operação do filtro de Sobel para uma máscara de dimensão 5x5.

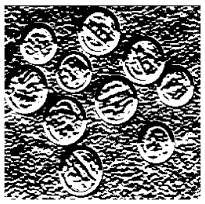

Figura 5. Resultado da aplicação do Sobel vertical.

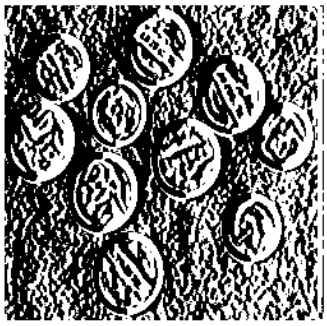

Figura 6. Resultado da aplicação do Sobel horizontal.

Nessa imagem devido ao seu baixo número de bordas é possível claramente ver as bordas das moedas presentes na imagem mesmo com a presença de ruídos.

c) A latência do sistema depende da resolução da imagem, mais precisamente, do tamanho do comprimento da imagem. Dessa formas imagens maiores demoram um maior tempo

de latência, por exemplo, durante as simulações a imagem de 250x250 pixels de resolução necessitou de 756 ciclos de relógio e a imagem de 300x300 necessitou de 906 ciclos de relógio.

Para o *thoughtput* o mesmo padrão se aplica porque cada convolução necessita de um ciclo de clock, percebe-se que a imagem de 250x250 pixels para ser finalizada necessita de 62500 ciclos de clock e a imagem de 300x300 necessita 90000 ciclos de clock.

d) Como maior a imagem e maior é o tamanho de seus pixels maior é a quantidade recursos necessários para a aplicação do filtro como é possível visualizar através dos relatórios de recursos obtidos é possível verificar na Tabela III o uso de recursos para a imagem com resolução 300x300 e na Tabela IV o uso de recursos para a imagem com resolução 250x250.

Tipo	Usado	Disponível	Utilização (%)
Slice LUTs	589	20800	2.83
LUT as Logic	259	20800	1.25
LUT as Memory	330	9600	3.44
Slice Registers	739	41600	1.78
Register as Flip Flop	739	41600	1.78
Register as Latch	0	41600	0
F7 Muxes	0	16300	0
F8 Muxes	0	8150	0

Tabela I. TABELA COM O *report* DE UTILIZAÇÃO DE RECURSOS DO FPGA PARA A IMAGEM COM RESOLUÇÃO DE 300X300.

Tipo	Usado	Disponível	Utilização (%)
Slice LUTs	612	20800	2.94
LUT as Logic	292	20800	1.40
LUT as Memory	320	9600	3.33
Slice Registers	787	41600	1.89
Register as Flip Flop	787	41600	1.89
Register as Latch	0	41600	0
F7 Muxes	0	16300	0
F8 Muxes	0	8150	0

Tabela II. TABELA COM O *report* DE UTILIZAÇÃO DE RECURSOS DO FPGA PARA A IMAGEM COM RESOLUÇÃO DE 250X250.

II. EXERCÍCIO 2 - MULTIPLICAÇÃO MATRICIAL

a) Usando como exemplo o circuito de multiplicação matricial 2x2 desenvolvido em sala de aula, implemente em VHDL um circuito de multiplicação matricial 3x3. Faça uso dos operadores de cálculo em ponto flutuante e implemente a arquitetura com maior capacidade de paralelismo.

b) Crie um arquivo de *tesbench* de simulação automática usando arquivos texto com 100 valores aleatórios para cada entrada. O valor máximo e mínimo dos valores de entrada depende do último número da sua matrícula (17/0067033). No caso, as faixas de valores são $[-0.2, 0.2]$ para $[-10000, 10000]$ e o tamanho dos números em ponto flutuante são 27 e 32 bits, respectivamente.

c) Com base nas simulações estime a latência e o troughput do circuito. Considere que cada ciclo de relógio é de 100 MHz (período de 10 ns).

d) Sintetize o circuito e coloque em uma tabela o reporte de síntese com o consumo de recursos de hardware (LUTs, Flip-flops, DSPs e BRAMs)

A. Respostas

a) O componente *top-module* recebe como entrada as matrizes de entrada, como *arrays* de *std_logic_vector* e recebe como saída a matriz resultante, além de um sinal de controle indicando que a operação foi finalizada.

O funcionamento da arquitetura funciona através da multiplicação em paralelo dos elementos das matrizes e logo em seguida as somas desses elementos. Para a matriz 3x3 cada elemento da matriz equivale a 3 multiplicações em ponto flutuante e 2 somas.

Com o intuito de generalizar a multiplicação dos elementos e as suas somas utilizou-se laços-generate para gerar os elementos de multiplicação e soma.

b) Com as devidas modificações do arquivo de *testbench* os seguintes arquivos foram obtidos para as simulações.

Para a simulação com matrizes com elementos variando entre $[-0.2, 0.2]$ e tamanho de 27 bits temos a saída da simulação presente na Figura **??** e para as matrizes variando entre $[-10000, 10000]$ e tamanho 32 de ponto flutuante temos a saída da simulação presente na Figura **??**.

c) Independente do tamanho da palavra em ponto flutuante a latência e o throuput são iguais e equivalem a 6 ciclos de clock.

d) Como maior é o tamanho da palavra em ponto flutuante maior é a quantidade de recursos necessários para a execução da multiplicação matricial podemos verificar através dos relatórios de recursos obtidos é possível verificar na Tabela **??** o uso de recursos para uma matriz formada por pontos flutuantes de 27 bits e na Tabela **??** o uso de recursos para matrizes com ponto flutuante de 32 bits.

Tipo	Usado	Disponível	Utilização (%)
Slice LUTs	8316	20800	39.98
LUT as Logic	8316	20800	39.98
LUT as Memory	0	9600	0
Slice Registers	1710	41600	4.11
Register as Flip Flop	1710	41600	4.11
Register as Latch	0	41600	0
F7 Muxes	0	16300	0
F8 Muxes	0	8150	0

Tabela III. TABELA COM O *report* DE UTILIZAÇÃO DE RECURSOS DO FPGA PARA A MATRIZ DE 27 BITS.

Tipo	Usado	Disponível	Utilização (%)
Slice LUTs	8514	20800	40.93
LUT as Logic	8514	20800	40.93
LUT as Memory	0	9600	0
Slice Registers	2025	41600	4.87
Register as Flip Flop	2025	41600	4.87
Register as Latch	0	41600	0
F7 Muxes	0	16300	0
F8 Muxes	0	8150	0

Tabela IV. TABELA COM O *report* DE UTILIZAÇÃO DE RECURSOS DO FPGA PARA MATRIZES COM FLOAT DE 32 BITS.

REFERÊNCIAS

[1] V. A. Pedroni, *Circuit design with VHDL*. MIT press, 2004.