Circuitos Sequenciais e Projeto RTL

Arthur Simões Gonzaga
Engenharia Eletrônica
Universidade de Brasília
Campus Gama
Email: arthursgonzaga@gmail.com
Matrícula: 14/0016775

Leonardo Borges Brandão de Freitas
Engenharia Eletrônica
Universidade de Brasília
Campus Gama
Email: leonardobbfga@gmail.com
Matrícula: 14/0025197

***Resumo*—Este presente relatório visa apresentar a aplicação de circuitos sequenciais e projetos RTLs *(Register-transfer Level)* em FPGAs *(Field-programmable Gate Arrays)*. As aplicações em processamentos de sinais, filtros Sobel e Kalman, foram desenvolvidos em VHDL *(VHSIC Hardware Description Language)* voltados para a plataforma da Xilinx, Basys 3. Desenvolveu-se os aspectos teóricos da disciplina de Projetos de Circuitos Reconfiguráveis, tais como: desenvolvimento dos diagramas de blocos RTL, criação de *testbenchs*, utilização de *IPCores* e afins. Por fim, analisou-se os aspectos de tempo e de energia, e os resultados a fim de obter uma comparação com ferramentas já consolidadas, tais como o *MatLab*.**

I. Introdução

Circuitos eletrônicos têm diversas aplicações no cotidiano. Vê-se diversas aplicações nos campos de processamentos de sinais, telecomunicações ou no controle de plataformas, tais como satélites. Com o avanço tecnológico das tecnologias, hoje é possível atingir *sistemas on chip* com características reprogramáveis. São os casos das FPGAs, dispositivos PLD *(programmable logic device)*, que, segundo [2], permitem configuração e reconfiguração de qualquer circuito digital.

Para as aplicações mostradas neste experimento, abordou-se processamentos de sinais, a partir do filtro Sobel, e cálculo matricial, com o cálculo do ganho do filtro de Kalman. Organizou-se então a recursividade das FPGAs a favor do processamento paralelo e a utilização de IPCores, explorados na seção II. Após isto, aborda-se um breve estudo sobre a utilização do hardware disponível e seu consumo energético. Conclui-se o presente artigo, na comparação de valores obtidos, já na seção III, com o processamento em hardware e software, visando a análise temporal.

A. Fundamentação Teórica

1) Filtro Sobel: A detecção de bordas nas imagens pode ser um recurso muito poderoso nas aplicações de deep learning, reconhecimento de formas e imagens. O filtro Sobel, é uma aplicação de processamento de sinais, um algoritmo que usufrui de diferenças finitas para cálculo do gradiente da imagem.

Matematicamente, o filtro Sobel é uma convolução de um kernel de 3 pixels por 3 pixels com uma imagem desejada. O kernel utilizado influi na extração do gradiente e altera a sensibilidade quanto à borda. Além do mais, há dois tipos de kernel, um para extração em ordem vertical e outro para ordem horizontal.

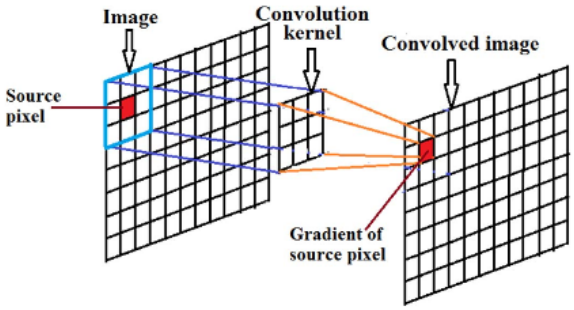

Figura 1. Exemplificação da convolução entre kernel e imagem. Fonte: A FPGA based implementation of Sobel edge detection.

Sendo A uma matriz 5 por 5 de pixels originais retirados da imagem, conforme figura 2, matriz G, também 5 por 5, o kernel do operador Sobel, o algoritmo escolhido para esta aplicação segue os seguintes passos:

(i-2,j-2)	(i-2,j-1)	(i-2,j)	(i-2,j+1)	(i-2,j+2)
(i-1,j-2)	(i-1,j-1)	(i-1,j)	(i-1,j+1)	(i-1,j+2)
(i,j-2)	(i,j-1)	(i,j)	(i,j+1)	(i,j+2)
(i+1,j-2)	(i+1,j-1)	(i+1,j)	(i+1,j+1)	(i+1,j+2)
(i+2,j-2)	(i+2,j-1)	(i+2,j)	(i+2,j+1)	(i+2,j+2)

Figura 2. Exemplo de matriz 5 por 5 utilizada nas operações.

1) Tratamento da imagem para binário;
2) Implementação de um registrador para extrair a matriz A;
3) Multiplicação em paralelo de cada elemento entre a matriz A e o kernel Gx ou Gy;
4) Utilização de um acumulador, somando todos os elementos da multiplicação;
5) Comparação do valor acumulado para detecção de borda.

Têm-se então como formulação geral:

$$pixel_{out} = \sum_{i=0}^{4}\sum_{j=0}^{4} G(i,j) * A(i,j) \quad (1)$$

Para a tomada de decisão, têm-se então as seguintes preposições:

$$pixel_{out} = \begin{cases} pixel_{out} > 255, pixel_{out} = 255 \\ 0 < pixel_{out} < 255, pixel_{out} = pixel_{out} \\ pixel_{out} < 0, pixel_{out} = 0 \end{cases} \quad (2)$$

Por fim, a imagem se torna composta por valores entre 0 e 255 (utilizando o caso de 8 bits), sendo 0 um pixel preto e 255 um pixel branco.

2) Cálculo do Filtro de Kalman: Rudolf Kalman, matemático húngaro, desenvolveu o filtro de Kalman, um algoritmo matemático que tem por objetivo predizer resultados próximos a grandezas medidas utilizando de medidas realizadas ao longo do tempo na presença de ruídos. O filtro de Kalman possui aplicação na telecomunicação, principalmente em PLLs *(phased-locked loop)*.

Nesta aplicação, busca-se realizar o cálculo do seu ganho a partir do paralelismo de fatores matriciais. Paralelismo é um recurso intrísico a ser explorado nas FPGAs, tendo em vista que a descrição de hardware possibilita a utilização de LUTs *(look-up tables)* e outros componentes de forma paralela.

O cálculo do filtro de Kalman é dado por:

$$K_{3x2} = A_{3x3}B^T_{3x2}[B_{2x3}A_{3x3}B^T_{3x2} + C_{2x2}]^{-1} \quad (3)$$

Na matriz inversa, usufrui-se de uma operação analítica baseada na matriz de cofatores. Para tal, utiliza-se então:

$$X^{-1} = \frac{1}{|X|}C^T_{(i,j)} \quad (4)$$

Sendo X a matriz a qual deseja-se calcular a inversa, $|X|$ o determinante da matriz X e C^T a matriz dos cofatores transposta.

II. Procedimentos Experimentais

A. Filtro Sobel

No filtro Sobel utilizou-se do diagrama de blocos da figura **??**, na qual segue os passos do algoritmo explicado na descrição I-A1.

Após desenho da estrutura lógica do filtro sobel, buscou-se implementar a solução dedicada à placa Basys 3, da Xilinx, da família Artix-7, a partir da ferramenta Vivado Design Suite.

A metodologia utilizada foi de implementar partes do filtro e testar, por meio dos testbench dedicado, se determinada parte era funcional ou estava com algum problema.

Iniciou-se a implementação do filtro sobel para uma matriz de 250 por 250 pixels, sendo cada pixel de 8 bits. Em seguida a implementação deste filtro, foi realizada a codificação em binário da imagem por meio do *MatLab* para que a simulação pudesse ser executada. Criou-se o testbench para tal ação e após cada simulação, obtinha-se um log de quais eram os erros, podendo assim, corrigí-los.

Depois de obter valores satisfatórios, pôde-se buscar uma implementação genérica do filtro, de forma atingir os requisitos do projeto: utilização de uma chave seletora para detecção de bordas verticais ou horizontais; alteração do código para parametrização, com o intuito de poder aplicar entradas (imagens) de diferentes tamanhos e com maior resolução, isso nos traz a capacidade de processar imagens de 5x5 até valores muito maiores.

Os pontos mais relevantes observados nas simulações foram: ajuste do tempo de simulação, para que toda a imagem pudesse ser lida e processada; regulagem da matriz kernel do filtro, tanto da matriz vertical quanto da horizontal. Neste ponto, não foi encontrado referências consolidadas com exemplos de matrizes 5 por 5 dedicados ao filtro sobel, então, estimou-se valores para realizar as simulações;

Enfim, foram feitos 4 testbenchs com 2 parâmetros a serem modificados: tamanho da imagem, que poderia ser de 100 pixels por 100 pixels, e tamanho do pixel, de 8 ou 10 bits.

B. Cálculo do filtro Kalman

Para explorar o paralelismo intrínseco das operações no cálculo do ganho do filtro Kalman, representado pela equação 3, foi implementado em VHDL um circuito sequencial de acordo com o diagrama de blocos a seguir.

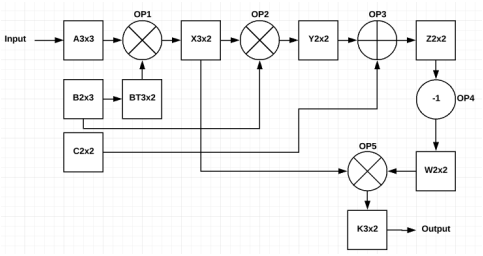

Figura 3. Diagrama de Blocos 01.

Como pode ser observado no diagrama acima utilizou-se registradores matriciais para armazenar o resultado das operações intermediárias do cálculo do ganho do filtro Kalman, como mostrado na tabela em anexo, II.

1) Operações Intermediárias: **OP1:** Para essa operação utilizou-se a matriz constante B2x3 em configuração transposta (BT3x2) multiplicando-a pela matriz de entradas do usuário A3x3 e armazenando os resultados na matriz auxiliar X3x2. de acordo como abaixo:

$$X(2,3) = B(2,3) * A(3,3) \quad (5)$$

Realizando essas operações com o maior grau de paralelismo possível se faz necessário a implementação em VHDL de 3 componentes de multiplicação e 2 componentes de soma para cada um dos 6 elementos da matriz X. Totalizando 18 multiplicações paralelas, seguidas de 6 somas paralelas, e por último 6 somas paralelas finais.

OP2: Para essa operação utilizou-se a matriz constante B2x3 multiplicando-a pela matriz X3x2 resultante da operação anterior (OP1), armazenando os resultados na matriz auxiliar Y2x2. de acordo com o equacionamento abaixo:

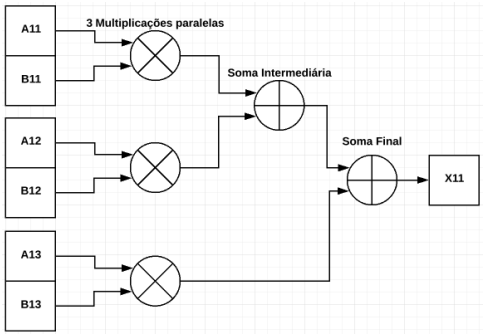

Figura 4. Diagrama de Blocos 02.

$$Y(2,2) = X(2,3) * A(3,2) \tag{6}$$

Realizando essas operações com o maior grau de paralelismo possível se faz necessário a implementação em VHDL de 3 componentes de multiplicação e 2 componentes soma para cada um dos 4 elementos da matriz Y. Totalizando 12 multiplicações paralelas, seguidas de 4 somas paralelas e por último 4 somas paralelas finais.

OP3: Para essa operação utilizou-se a matriz constante C2x2 somando-a com a matriz Y2x2 resultante da operação anterior (OP2), armazenando os resultados na matriz auxiliar Z2x2. de acordo com o equacionamento abaixo:

$$Z(2,2) = C(2,2) * Y(2,2) \tag{7}$$

Essa operação é bem simples de se realizar sendo necessário apenas 4 componentes de soma que trabalham em paralelo entre si.

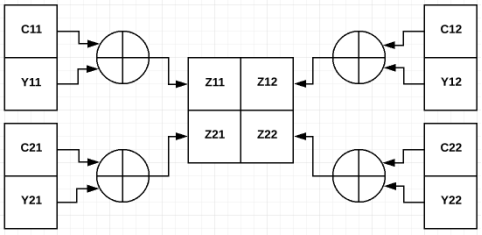

Figura 5. Diagrama de Blocos 03.

OP4: Nessa operação realizou-se a inversão da matriz Z2x2 resultante da operação anterior (OP3) armazenado o resultado na matriz auxiliar W2x2. Para inverter uma matriz 2x2 é necessário trocar de posição os elementos de sua diagonal principal, negativar os elementos da diagonal secundaria e finalmente dividir cada elemento pelo determinante da matriz.

$$W_{2x2} = Z_{2x2}^{-1} \tag{8}$$

$$W_{2x2} = \frac{1}{Z_{11} * Z_{22} - Z_{12} * Z_{21}} * \begin{bmatrix} Z_{22} & -Z_{12} \\ -Z_{21} & Z_{11} \end{bmatrix} \tag{9}$$

Dessa forma se faz necessário 2 multiplicações em paralelo, seguidas de uma subtração resultando no determinante de Z, e finalmente 4 divisões paralelas entre cada elemento da matriz pelo determinante previamente calculado, formatando, desta forma, a matriz auxiliar W.

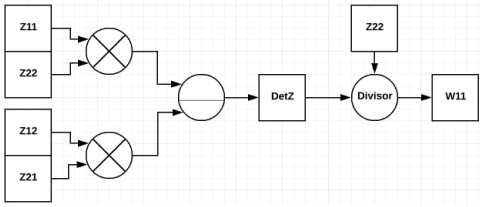

Figura 6. Diagrama de Blocos 04.

OP5: Essa é a operação final que resulta na matriz K3x2. Para tanto, utilizou-se a matriz W2x2, resultante da operação anterior (OP4), multiplicando-a pela matriz X3x2 que está armazenada em memória desde a primeira operação (OP1).

Realizando essas operações com o maior grau de paralelismo possível se faz necessário a implementação em VHDL de 2 componentes de multiplicação e 1 componentes de soma para cada um dos 6 elementos da matriz K. Totalizando 12 multiplicações paralelas, seguidas de 6 somas paralelas.

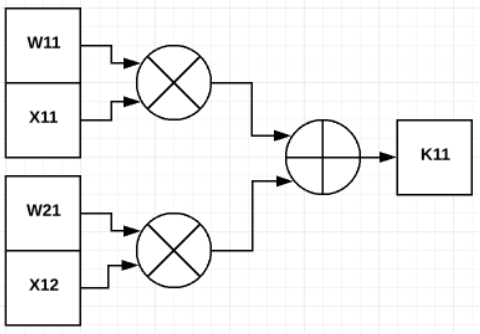

Figura 7. Diagrama de Blocos 05.

2) Funcionamento Sequencial: Primeiramente, antes das operações aritméticas ocorrerem, é preciso discutir a quantidade de memória necessária para armazenar os dados de entrada que usuários insere no sistema. Dados esses formatados em uma matriz A3x3 que possui 9 elementos representados por 16 bits cada um. Para tanto um vetor de entrada (reg_IN) de 144 bits de memória é totalmente carregado após 9 ciclos de clock com todos os dados de entrada do usuário, e logo depois organiza-os na matriz A3x3.

$$(9 ciclos de clock) * (\frac{16 bits}{ciclo de clock}) = 144 bits$$

Após a entrada de dados no sistema e o processamento desses dados a partir das operações intermediárias discutidas

na sessão anterior, o usuário deve selecionar o elemento da matriz resultante K3x2 que gostaria de observar. Para tanto, um multiplexador foi implementado ao final do sistema com 96 bits de entrada, 3 bits de seleção e 16 bits de saída para processar a saída do sistema entre os 6 possíveis elementos de 16 bits da matriz K3x2. Desta forma, o funcionamento do sistema para o cálculo do ganho do filtro kalman foi formatado no Diagrama de Blocos (RTL) e na máquina de estados Finitos (FSM) a seguir.

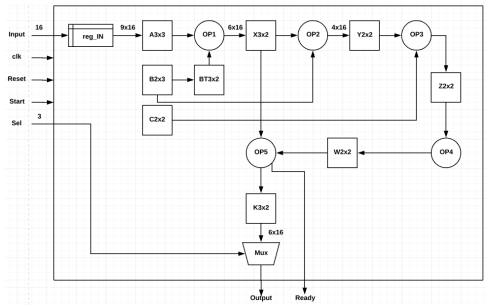

Figura 8. Diagrama de Blocos 06.

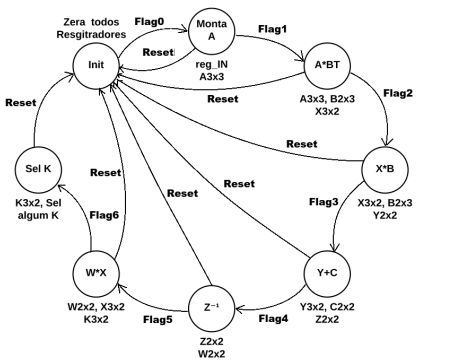

Figura 9. Máquina de Estados Implementada

Os estados da FSM do sistema estão melhor explicados na tabela III.

3) Componentes Aritméticos: Para possibilitar as operações aritméticas entre as matrizes ao longo do sistema, três componentes em VHDL foram instanciados a partir de IPcores do próprio Vivado, sendo eles um Somador/Subtrator, um Multiplicador e um Divisor. Todos trabalhando com uma formatação de 16 bits para os dados de entrada. No entanto, apenas o Somador/Subtrator mantém os 16 bits de entrada na saída, pois ao multiplicar ou dividir números binários de 16 bits o resultado será formatado em 32 bits o que dentro do sistema desenvolvido causa overflow. Problema esse que será melhor discutido na seção III.

III. Resultados

A. Filtro Sobel

1) Temporização: Considerando que o clock utilizado era de 100MHz, estimou-se para o primeiro filtro um total de 8043 ciclos de relógio de latência, para uma imagem de 250 por 250 pixels de 8 bits cada, pois, para preencher o registrador FIFO completamente (há 4 linhas mais 5 pixels, totalizando 8040 bits de leitura) e realizar as operações de multiplicação, soma e comparação (cada uma com 1 ciclo, totalizando os 3 ciclos restantes). Essa operação inicial, totalizaria aproximadamente 80μs.

Utilizando a imagem de 100 por 100 pixels, de 10 bits cada, espera-se um tempo menor de latência, com 4053 ciclos de relógio, levando o mesmo raciocínio que a imagem maior, totalizaria aproximadamente 40μs.

O troughput do circuito 100 por 100 com 10 bits fica em torno de 104μs, para processar uma imagem de 1Kb, a velocidade fica em torno de 1Gbps. No circuito de 250 por 250, com 8 bits, consegue-se acelerar o processamento, chegando em torno de 4,5Gbps.

Tabela I
Utilização de Hardware

Recurso	Utilização	Disponível	Porcentagem
LUT	687	20800	3.30
LUTRAM	256	9600	2.67
FF	752	41600	1.81
IO	38	106	35.85
BUFG	1	32	3.13

2) Consumo de Hardware e Energético: O consumo de hardware pode ser observado na tabela abaixo:

3) Comparação Hardware e Software: Abaixo, pode-se comparar as imagens processadas por hardware e por software.

Figura 10. Figura Original - Toys Flash

Para as figuras 100 por 100, temos os seguintes resultados:

B. Cálculo do Ganho do Filtro Kalman

Não se obteve resultados satisfatórios na codificação do filtro Kalman isso devido a existência de 3 problemas principais na implementação em VHDL da solução proposta

1) Diferença do tamanho dos Dados: Os dados de entrada do usuário no sistemas são de 16 bits e espera-se dados de saída também de 16 bits. No entanto, ao longo do sistemas os números binários encontram com multiplicações e divisões o

Figura 11. Bordas Verticais - Hardware

Figura 12. Bordas Verticais - Software

que dobra a necessidade de memoria para armazenar os dados formatando-os em 32 bits, o que cascateou uma série de erros na hora da síntese do código.

2) Utilização de Ipcores: O que a princípio facilitaria o trabalho com operações aritméticas durante o desenvolvimento da solução, na verdade dificultou ainda mais o trabalho deixando o código gigante e extremamente confuso. A maior parte do problema foi a inexperiência em trabalhar com esse tipo de estrutura paralela via Ipcores.

3) Má Gestão do Tempo: Dedicou-se pouco tempo para resolver o problema já que esse parecia muito simples de se resolver, mas se demonstrou muito complicado na hora da execução. Todo o levantamento teórico foi bem estruturado, como pode ser observado na sessão II. Procedimentos Experimentais e uma tentativa frustrada de codificar a solução está disponível no código FiltroKalman.vhd que mesmo inacabado possui mais de 800 linhas.

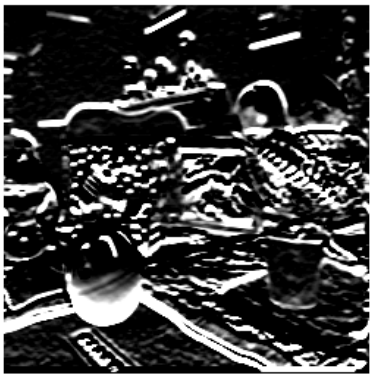

Figura 13. Bordas Horizontais - Hardware

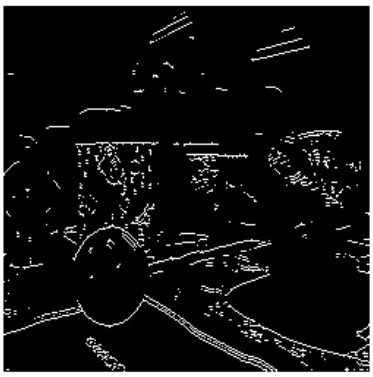

Figura 14. Bordas Horizontais - Software

Figura 15. Figura Original - Tire

IV. Conclusões

Conclui-se que a implementação do filtro sobel atingiu padrões aceitáveis quanto ao nível de hardware, contudo, possui um gasto energético além do imaginado. Os resultados

Figura 16. Bordas Verticais - Hardware

Figura 17. Bordas Verticais - Software

Figura 18. Bordas Horizontais - Hardware

Figura 19. Bordas Horizontais - Software

não são tão próximos aos apresentados pelo MatLab, tendo em vista que os coeficientes do Kernel deveriam ser ajustados.

Com o filtro Kalman, não obteve-se resultados, pois a implementação não foi realizada com sucesso, assim como descrito em III.

Foi satisfatório o conhecimento aprendido com as ferramentas, espera-se aprofundar os conceitos da matéria nos próximos trabalhos. Observa-se que as aplicações utilizadas neste experimento são utilizáveis no cotidiano da engenharia.

REFERÊNCIAS

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Valiante Filho, Filippo. Ferramentas e metodologias de desenvolvimento para sistemas parcialmente reconfiguráveis. Diss. Universidade de São Paulo, 2008.

[3] Baptista, Daniel. "Desenvolvimento de circuitos reconfiguráveis para interface série usando o protocolo RS-232."Electrónica e Telecomunicações 4.8 (2007): 900-907.

[4] Nausheen, Nazma, et al. "A FPGA based implementation of Sobel edge detection."Microprocessors and Microsystems 56 (2018): 84-91.

V. ANEXO 1

Tabela II
OPERAÇÕES E MATRIZES AUXILIARES (FILTRO KALMAN)

Operação	Matrizes Auxiliares	Resultado
OP1	X3x2 = A3x3 . (B3x2)^T	A3x3 . (B3x2)^T
OP2	Y2x2 = B2x3 . X3x2	B2x3 . A3x3 . (B3x2)^T
OP3	Z2x2 = C2x2 . Y2x2	B2x3 . A3x3 . (B3x2)^T + C2x2
OP4	W2x2 = (Z^2x2)^(-1)	[B2x3 . A3x3 . (B3x2)^T + C2x2]^(-1)
OP5	K3x2 = W2x2 . X3x2	A3x3 . B3x2 . [B2x3 . A3x3 . (B3x2)^T + C2x2]^(-1)

Tabela III
FSM IMPLEMENTADA

Estado	**Função**	**Entradas**	**Saídas**
Init	Zera todos os registradores e sinais	Todos os registradores e sinais	Flag 0 (Sinal)
Monta A	Montar matriz A3x3 a partir do registrador de entradas (reg IN)	reg IN (Registrador) Flag 0 (Sinal)	A3x3 (Matriz) Flag 1 (Sinal)
A*BT	OP1: Multiplicar as matrizes A3x3 e BT3x2	A3x3, BT3x2 (Matriz) Flag 1 (Sinal)	X3x2 (Matriz) Flag 2 (Sinal)
X*B	OP2: Multiplicar as matrizes X3x2 e B2x3	X3x2, B2x3 (Matriz) Flag 2 (Sinal)	Y2x2(Matriz) Flag 3 (Sinal)
Y+C	OP3: Somar as Matrizes Y2x2 e C2x2	Y2x2, C2x2 (Matriz) Flag 3 (Sinal)	Z2x2 (Martiz) Flag 4 (Sinal)
Z^(-1)	OP4: Inverter a matriz Z2x2	Z2x2 (Matriz) Flag 4 (Sinal)	W2x2 (Matriz) Flag 5 (Sinal)
W*X	OP5: Multiplicar as matrizes W2x2 e X3x2	W2x2, X3x2 (Matriz) Flag 5 (Sinal)	K3x2 (Matriz) Flag 6 (Sinal) Ready (Sinal)
Sel K	Selecionar o elemento de saída do sistema	K3x2 (Matriz) Sel (3 bits de seleção) Flag 6 (sinal)	Algum elemento da matriz K3x2