UNIVERSIDADE CATÓLICA DE PELOTAS
CENTRO POLITÉCNICO
MESTRADO EM ENGENHARIA ELETRÔNICA E COMPUTAÇÃO

PEDRO LUÍS CARNEIRO MARQUES

Arquitetura para Inversão de Matrizes usando Circuito Divisor Eficiente Baseado no Algoritmo Goldschmidt

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Engenharia Eletrônica e Computação

Orientador: Prof. Dr. Sergio José Melo de Almeida

Pelotas
2016

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Marques, Pedro Luís Carneiro

Arquitetura para Inversão de Matrizes usando Circuito Divisor Eficiente Baseado no Algoritmo Goldschmidt / Pedro Luís Carneiro Marques. – Pelotas: 2016.

61 f.: il.

Dissertação (mestrado) – Universidade Católica de Pelotas. 2016. Orientador: Sergio José Melo de Almeida.

1. Inversão de matrizes. 2. Redução de potência. 3. Circuito divisor. 4. Algoritmo Goldschmidt. 5. Implementação ASIC. I. Almeida, Sergio José Melo de. II. Título.

DEDICATÓRIA

A minha esposa, Mari, pelo incentivo e por abdicar de muitas horas de nosso convívio para que fosse possível realizar este trabalho.

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— SIR ISAAC NEWTON

AGRADECIMENTOS

Aos meus filhos João Pedro, Luís Otávio e Lílian pela compreensão e paciência pela minha ausência;

Ao meu orientador, Prof. Dr. Sérgio José Melo de Almeida, pela cobrança, pelo incentivo e pela contribuição e tempo desprendido ao longo de todo mestrado;

Ao Prof. Dr. Eduardo Antonio César da Costa, pelo tempo, paciência e apoio total na execução deste trabalho;

Ao Prof. Dr. Adenauer Corrêa Yamin, pelo "pontapé inicial", sem o que eu não teria começado, pelas sempre belas palavras de incentivo e pela pronta disposição, sempre que precisei;

Ao amigo Guilherme Pereira Paim pela dedicação e ajuda na execução deste trabalho;

Aos meus amigos da primeira turma de mestrado em Engenharia Eletrônica e computação, Julio, Cleiton, Suzuki e Ott, pelas horas de trabalho conjunto;

A todos professores e funcionários do programa de pós-graduação da UCPEL;

A todos que, de uma forma ou de outra, contribuíram para a execução deste trabalho.

RESUMO

O cálculo de inversão de matrizes está presente em várias aplicações da área de Processamento de Sinais. Entre essas aplicações, a filtragem adaptativa, baseada no algoritmo de Projeções Afins, inclui o cálculo de inversão de matrizes, que agrega uma elevada complexidade computacional. Existem vários algoritmos para o cálculo de inversão de matrizes. A complexidade do algoritmo está associada ao tamanho da matriz, que varia de acordo com a aplicação alvo. Essa dissertação propõe a implementação em *hardware* dedicado do algoritmo analítico de inversão de matrizes. Esse algoritmo é o mais apropriado para a implementação de uma matriz de tamanho 2x2, que é o tamanho adequado para uma implementação do algoritmo de Projeções Afins para diversas aplicações práticas. No bloco de inversão de matriz, o circuito divisor é o que agrega a maior complexidade computacional. Dentre os algoritmos de divisão presentes na literatura, os algoritmos baseados em iterações funcionais são considerados os mais rápidos, pois são capazes de tirar proveito de multiplicadores de alta velocidade, para convergir de forma quadrática para um resultado. Dentre os algoritmos baseados em iterações funcionais, destacam-se os algoritmos de Newton-Raphson e de Goldschmidt. Entretanto, o algoritmo de Goldschmidt tem sido mais utilizado em aplicações que demandam alta velocidade de processamento, pois ao contrário do algoritmo Newton-Raphson, onde as multiplicações são dependentes umas das outras, no algoritmo Goldschmidt as multiplicações são realizadas em paralelo. Nesse trabalho, propõe-se a implementação em *hardware* de um circuito divisor eficiente baseado no algoritmo Goldschmidt. O circuito divisor usa um multiplicador na base 4 da literatura, que torna o divisor mais eficiente em termos de dissipação de potência, quando comparado ao circuito divisor usando o multiplicador da ferramenta de síntese. O circuito divisor proposto aumenta a faixa de valores de operação através do uso do padrão Q7.8, que permite valores entre -127.99609375 e +127.99609375, ao contrário do divisor Goldschmidt original, que admite uma estreita faixa de valores ente 1 e 2. Os principais resultados mostram que o uso do divisor Goldschmidt eficiente proposto torna o circuito inversor de matriz com uma menor dissipação de potência, o que se torna um atrativo para uma futura implementação da arquitetura completa do algoritmo de Projeções Afins.

Palavras-chave: Inversão de matrizes. redução de potência. circuito divisor. algoritmo Goldschmidt. implementação ASIC.

Matrix Inversion Architecture Using Efficient Divider Circuit Based on Goldschmidt Algorithm

ABSTRACT

The matrix inversion calculation is present in several applications in the area of Signal Processing. Among these applications, the adaptive filtering, based on the algorithm of Affine Projections, includes the calculation of matrix inversion, which adds a high computational complexity. There are several algorithms for calculating matrix inversion. The complexity of the algorithm is associated with the size of the matrix, which varies according to the target application. This dissertation proposes the implementation in dedicated hardware of the analytical algorithm of matrix inversion. This algorithm is most appropriate for the implementation of a 2x2 size matrix, which is the appropriate size for an implementation of the algorithm of Affine Projections for several practical applications. In the matrix inversion block, the divisor circuit is that adds the highest computational complexity. Among the division algorithms from the literature, algorithms based on functional iterations are considered the fastest, because they are able to take advantage of high speed multipliers to converge in a quadratic form to a result. Among the algorithms based on functional iterations, Newton-Raphson and Goldschmidt algorithms are the most used algorithms. However, the Goldschmidt algorithm has been more used in applications that demand high processing speed, because unlike the Newton-Raphson algorithm, where the multiplications are dependent on each other, in the Goldschmidt algorithm the multiplications are performed in parallel. In this work, it is proposed the hardware implementation of an efficient divisor circuit based on the Goldschmidt algorithm. The divider circuit uses a radix-4 multiplier from the literature, which is more efficient in terms of power dissipation, when compared to the divider circuit using the multiplier from the synthesis tool. The proposed divider circuit increases the range of operating values by using the Q7.8 standard, which allows values between -127.99609375 and +127.99609375, rather than the original Goldschmidt divider, which supports a narrow range of values between 1 and 2. The main results show that the use of the proposed efficient Goldschmidt divider circuit makes the matrix inverter circuit with a lower power dissipation, which becomes an attractive for a future implementation of the complete affine projections algorithm in dedicated hardware.

Keywords: matrix inversion, power reduction, division circuit, Goldschmidt algorithm, efficient division, ASIC implementation .

LISTA DE ABREVIATURAS E SIGLAS

AP	Affine Projection
ASIC	Application-Specific Integrated Circuit
DSP	Digital Signal Processors
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
LMS	Least Mean Square
LU	Lower Upper
LUT	Look-Up Table
NLMS	Normalized Least Mean Squares
RCA	Ripple Carry Adders
RLS	Recursive Least-Squares
RTL	Register Transfer Level
SDF	Standard Delay Format
SRT	Sweeney, Robertson, and Tocher
SSIM	Structural Similarity
ULP	Unit in the Last Place
VCD	Value Change Dump
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a dissipação de potência vem sendo apontado como um dos principais parâmetros em projetos de circuitos integrados. Na verdade, o maior desafio diz respeito ao projeto de uma nova geração de produtos que dissipem o mínimo de potência, sem comprometer o alto desempenho desejado e sem penalidade excessiva em área de silício. Neste contexto, aspectos de baixa dissipação de potência exigem a pesquisa de novas arquiteturas confiáveis que possam ser testadas e fabricadas e que levem em consideração as necessidades de sistemas eletrônicos portáteis de funções diversas.

Uma classe de algoritmos que tem merecido especial atenção, com aspectos de baixa potência, são os de processamento digital de sinais. De fato, esse interesse tem sido intensificado desde a proliferação de equipamentos eletrônicos portáteis operados por baterias, tais como telefones celulares, laptops, equipamentos biomédicos, etc.

De fato, o progresso recente no desenvolvimento da microeletrônica, disponibilizou processadores digitais de sinais (DSP- *Digital Signal Processors*) mais velozes, o que propiciou um rápido aumento no campo de aplicações dos filtros adaptativos. Este aumento na utilização de filtros adaptativos e a implementação de algoritmos em tempo real tornou necessário o desenvolvimento de modelos analíticos que sejam capazes de prever o comportamento dos algoritmos em condições reais de operação ou muito próximos disso (ALMEIDA, 2004).

Entre os diversos algoritmos adaptativos existentes atualmente, o algoritmo de Projeções Afins (AP - *Affine Projection*) tem se destacado por sua robustez e velocidade de convergência. Entre suas características importantes em aplicações práticas, está seu excelente desempenho quando submetido a operações com sinais de entrada muito correlacionados, como por exemplo, no cancelamento de eco (ALMEIDA, 2004)(GARCIA, 2012). Entretanto, este desempenho superior aos demais algoritmos tem o custo de uma maior complexidade computacional.

Para a implementação do algoritmo AP, o *hardware* mais complexo é aquele responsável por efetuar a inversão de matriz. Dessa forma, torna-se importante a implementação de um método analítico eficiente que possa contribuir para o aumento de velocidade de processamento e redução da dissipação de potência em uma futura implementação do circuito baseado no algoritmo AP. Um exemplo típico de aplicação dos algoritmos AP são os aparelhos auditivos, que formam uma classe especial de sistemas de áudio. Como os aparelhos auditivos são alimentados por bateria, logo, o consumo de energia desse aparelho de processamento de sinal é uma questão importante (BREINING et al., 1999).

O processo de inversão de matrizes está presente em várias aplicações de sistemas de

comunicação e processamento de sinais. Nessas aplicações, a inversão de matrizes é um processo intensivo em termos de processamento, e sua implementação em *hardware* de ponto-fixo é um desafio, visto que a maior precisão dos resultados está relacionado a arquiteturas de ponto-flutuante. Entretanto, em termos de *hardware* a implementação em ponto- fixo é menos custosa, em relação a uma implementação em ponto-flutuante.

Para o cálculo de inversão de matriz, a complexidade do método analítico a ser empregado, cresce exponencialmente com o tamanho da matriz (IRTÜRK, 2009). Desta forma, alguns métodos analíticos são empregados para a decomposição de matrizes, tais como decomposição LU (THERRIEN, 1992), decomposição QR (MALAJOVICH, 2007), decomposição Cholesky (THERRIEN, 1992), entre outros.

No cálculo da inversão de matrizes, a operação de divisão é uma das mais complexas, pois envolve uma elevada complexidade computacional. Logo, a operação aritmética da divisão representa um dos principais desafios quando faz parte de operações em algoritmos onde se pretende implementá-lo em uma arquitetura dedicada de *hardware*. Essa complexidade se torna maior ainda quando envolve processos de inversões de matrizes na execução de operações matemáticas necessárias no algoritmo. Tais operações se fazem necessárias em diversos algoritmos voltados para aplicações práticas na engenharia. Citando algumas, por exemplo, cancelamento de eco acústico, controle ativo de ruído, estimação de parâmetros biomédicos, entre vários outros.

Dentro deste contexto, diversos algoritmos de divisão são propostos na literatura, tais como *restoring*, *non-restoring*, SRT e os métodos por convergência, baseados nos algoritmos Newton-Raphson (OBERMAN; FLYNN, 1995) e Goldschmidt (MARKSTEIN, 2004). Os métodos de divisão por convergência são considerados os mais rápidos, pois são baseados em aproximações sucessivas, realizadas por multiplicações sucessivas. Entretanto, o algoritmo Goldschmidt tem sido a a escolha preferida em diversas aplicações, pois agrega uma maior velocidade de convergência, visto que as multiplicações são realizadas em paralelo, ao contrário do algoritmo de Newton-Raphson, onde as multiplicações sucessivas são dependentes umas das outras.

Nessa dissertação propõe-se um circuito divisor baseado no algoritmo Goldschmidt. O divisor proposto opera para diferentes faixas de valores, ao contrário do divisor original, que realiza as operações dentro de uma faixa limitada de valores que varia de 1 a 2. Também são apresentadas as condições nas quais esse operador aritmético pode ser utilizado no sentido de reduzir a atividade de chaveamento nos barramentos de dados na arquitetura de inversão de matrizes.

1.1 Motivação

A principal motivação desta dissertação é a otimização de circuitos digitais para algoritmos de Processamento de Sinais, que levem em conta a utilização de técnicas que possam aumentar o desempenho e reduzir a dissipação de potência dos circuitos implementados. Em particular, o projeto de arquiteturas envolvendo operações de inversão de matrizes de baixa complexidade e de alta velocidade de operação é buscado neste trabalho.

Com os desafios atuais nas novas tecnologias para a redução da dissipação de potência em circuitos integrados, torna-se importante o estudo e implementação de técnicas que possam ter um impacto direto na redução da dissipação de potência nos diferentes circuitos digitais. Em particular, em aplicações, tais como multimídia, comunicações e métodos numéricos, torna-se necessário a implementação de estruturas de *hardware* eficientes com aspecto de baixa dissipação de potência.

Em particular, técnicas de filtragem adaptativa possuem amplo espectro de utilização, como por exemplo, no cancelamento de eco acústico e elétrico, cancelamento de interferências, cancelamento de harmônicas, entre outras.

As características desejáveis para um filtro adaptativo são a habilidade para operar em um ambiente desconhecido e seguir as variações no tempo dos sinais de entrada. A diferença essencial entre as diversas aplicações de filtros adaptativos é como os sinais são conectados, existindo, assim, quatro classes básicas de filtros adaptativos: identificação de sistemas, modelagem inversa, predição linear e cancelamento de interferências (WIDROW BERNARD.; HOFF, 1988).

Embora as características de operação do algoritmo adaptativo de projeções afins (AP) se mostrem bastante interessantes, sua complexidade computacional pode ser maior do que a de outros algoritmos mais simples, tais como LMS *Least Mean Square* e NLMS *Normalized Least Mean square*. Por este motivo, este algoritmo (AP) ainda não foi tão estudado quanto à implementação em *hardware*, em relação aos demais. Entretanto, em decorrência da crescente evolução da área da microeletrônica, o algoritmo AP se tornou uma solução viável para diversas aplicações, dada a velocidade dos novos processadores digitais de sinais, permitindo projetos e implementações de estruturas de *hardware* e *software* que possibilitam a utilização de algoritmos que antes não eram viáveis de serem implementados em função da sua complexidade em termos de *hardware*.

Esta dissertação de mestrado visa contribuir para uma futura implementação eficiente de uma arquitetura completa do algoritmo de projeções afins AP. Visto que nesse algoritmo, as

operações que agregam a maior complexidade computacional são as divisões e a inversão de matrizes, logo esta trabalho propõe um circuito dedicado de inversão de matrizes, que utiliza circuitos divisores eficientes em termos de velocidade computacional e dissipação de potência. O divisor proposto, baseado no algoritmo Goldschmidt, utiliza técnicas para o aumento da faixa de valores a serem processados, bem como agrega o aspecto de redução de dissipação de potência, usando técnicas mais simples de serem implementadas em *hardware*.

Desta forma, a principal motivação desta dissertação de mestrado é a implementação de uma arquitetura de inversão de matrizes eficiente, em termos de velocidade de processamento e redução de dissipação de potência, que possa contribuir para uma futura implementação de uma arquitetura de baixa dissipação de potência do algoritmo de projeções afins, que possa ser utilizada em uma aplicação que demande aspecto de redução de potência, tal como em aparelhos auditivos.

1.2 Objetivos

O objetivo geral desta proposta é desenvolver metodologias de projeto de um circuito de inversão de matrizes usando circuitos divisores eficientes em termos de aumento de desempenho e redução de dissipação de potência.

Em relação ao projeto dos circuitos divisores o principal objetivo é apresentar uma estrutura modificada do algoritmo Goldschmidt, que permita a operação de divisão em uma maior faixa de valores, que seja compatível com o circuito de inversão de matrizes (o circuito divisor Goldschmidt original apresenta uma faixa limitada de valores entre 1 e 2).

Os circuitos divisores implementados são aplicados a uma arquitetura dedicada de inversão de matrizes de tamanho 2x2, que tem um tamanho adequado para uma futura implementação de uma arquitetura completa do algoritmo de projeções afins de baixa dissipação de potência.

1.3 Principais Contribuições

A partir do estudo realizado para a exploração de novas arquiteturas de divisores de baixa dissipação de potência, para aplicação em uma arquitetura dedicada de inversão de matrizes, apresentam-se as seguintes contribuições do trabalho:

- Implementação de um circuito divisor Goldschmidt em ponto fixo, para operações em complemento de 2, com modificações para operação em uma maior faixa de valores;

- Implementação de um circuito dedicado para a inversão de matrizes utilizando os circuitos divisores propostos.

1.4 Organização do Trabalho

No Capítulo 2 são apresentadas as principais características dos diferentes algoritmos para inversão de matrizes, com uma comparação entre os diferentes métodos, fundamentando a escolha pelo método analítico. Os diferentes algoritmos de divisão são abordados no Capítulo 3. O Capítulo 4 aborda os algoritmos e as arquiteturas de divisão propostas para o algoritmo Goldschmidt. O circuito de inversão de matrizes proposto, baseado no método analítico, com os circuitos divisores propostos é apresentado no Capítulo 5. Os resultados das otimizações são apresentados no Capítulo 6 e no Capítulo 7 são apresentadas as conclusões e trabalhos futuros.

2 MÉTODOS PARA INVERSÃO DE MATRIZES

Neste capítulo, apresenta-se uma breve abordagem sobre os métodos clássicos para inversão de matrizes. O objetivo desse estudo é obter uma referência matemática que permita uma escolha comparativa adequada do método a ser utilizado na arquitetura em *hardware* proposta no trabalho.

2.1 Introdução

Seja A uma matriz quadrada de ordem n. Uma matriz B é chamada inversa de A se, e somente se a equação 2.1 for respeitada, onde $B = A^{-1}$ e I_n é a matriz identidade de ordem n.

$$A.B = B.A = I_n \tag{2.1}$$

Na literatura existem diversos métodos analíticos para a inversão de matrizes. A seguir serão discutidos alguns mais utilizados.

2.2 Inversão de Matriz Baseada na Decomposição LU

Diversos problemas podem ser modelados matematicamente em termos de sistemas de equações lineares, conforme a equação (2.2). Para estes sistemas existem vários métodos de resolução já bastante utilizados, principalmente no que diz respeito a matrizes quadradas.

$$AX = B \tag{2.2}$$

O processo de fatoração para sistemas de equações lineares, desse tipo, consiste em decompor a matriz A em um produto de dois ou mais fatores, e em seguida, resolver uma sequência de sistemas triangulares que será a solução geral do sistema.

A fatoração LU consiste em dividir a matriz A em duas submatrizes triangulares: L (de *lower*, inferior em inglês) e U (de *upper*, superior em inglês) (THERRIEN, 1992). Esse método funciona apenas para matrizes não-singulares. Assim, a identidade na equação 2.3 é satisfeita.

$$(LU)X = B \tag{2.3}$$

Para a inversão da matriz A, pode-se usar as matrizes L e U, isto é, A^{-1} pode ser obtido

diretamente da expressão (2.4).

$$A^{-1} = (LU)^{-1} = U^{-1}L^{-1} \tag{2.4}$$

A solução segue quatro etapas:

1. Decomposição LU da matriz dada;
2. Inversão da matriz triangular inferior L;
3. Inversão da matriz triangular superior U;
4. Multiplicação das matrizes inversas de L e de U.

Destaca-se aqui a vantagem em que uma vez a matriz A decomposta, pode-se encontrar diferentes vetores solução X, do sistema $AX = B$, para diferentes valores de B, bem como a inversão da matriz A. Vale ressaltar também que, embora bastante usada na solução de sistemas lineares, a decomposição LU apresenta uma complexidade computacional elevada, embora as outras três etapas sejam relativamente simples, em virtude da estrutura triangular das matrizes L e U.

2.3 Inversão de Matriz Baseada na Decomposição Cholesky

A decomposição de Cholesky procura decompor uma matriz A na forma $A = G.G^T$, sendo G uma matriz triangular inferior, cujos elementos da diagonal principal são estritamente positivos. Nessa decomposição, a matriz A deve ser definida positiva (THERRIEN, 1992).

A solução para inversão da matriz A (A^{-1}) pelo método de decomposição Cholesky pode ser avaliado conforme representação da equação (2.5).

$$A^{-1} = G^{-1}(G^T)^{-1} \tag{2.5}$$

A solução consiste basicamente em quatro etapas:

1. Decomposição Cholesky da matriz dada;
2. Inversão da transposta da matriz triangular inferior;
3. Inversão da matriz triangular inferior.
4. Multiplicação de matrizes, de acordo com a fórmula 2.5

Assim como o método da decomposição LU, o método da decomposição de Cholesky apresenta uma complexidade computacional elevada em função das dimensões matriciais.

2.4 Inversão de Matriz Baseada na Decomposição QR

A inversão da matriz A usando o método de decomposição QR é representada pela equação (2.6) abaixo.

$$A^{-1} = R^{-1}Q^T \tag{2.6}$$

Este processo pode ser resumido em três etapas básicas, ou seja,

1. Decomposição QR da matriz A;
2. Inversão da matriz triangular superior R;
3. Multiplicação das matrizes de acordo com a equação (2.6)

A estrutura da expressão para inversão da matriz A é similar às anteriores, com a diferença de que apenas uma das matrizes do método de decomposição será invertida. Considerando que a essa matriz inversa é triangular (triangular superior), a complexidade computacional desse método será menor que os métodos anteriores.

2.5 Inversão de Matriz Baseada na Redução de Gauss-Jordan

A redução ou transformação de Gauss-Jordan procura, através de técnicas de escalonamento e pivoteamento, transformar $[A|I]$ (Matriz A concatenada com a matriz I) em $[I|A^{-1}]$. Note que, I é uma matriz identidade

A solução para inversão da matriz A pelo método de redução de Gauss-Jordan, é obtida conforme a execução das seguintes etapas:

1. Formar a matriz $[A|I_n]_{(n\times 2n)}$ (Concatenação da matriz dada $A_{(n\times n)}$ com a matriz identidade I_n);
2. Calcular a forma escalonada reduzida por linhas da matriz aumentada obtida no passo 1 utilizando operações elementares nas linhas;
3. Assumindo que o passo 2 resultou numa matriz $[C|D]$, na forma escalonada reduzida por linhas, um dos dois resultados abaixo será obtido:
   - se $C = I_n$, então $D = A^{-1}$;
   - se $C \neq I_n$, então C tem linha nula, e neste caso, A é singular e A^{-1} não existe.

2.6 Inversão de Matriz Usando Método Analítico

Para a inversão de uma matriz, usando o método analítico, são calculadas a matriz adjunta, $Adj([A])$, e o determinante $Det([A])$, conforme a equação (2.7).

$$[A]^{-1} = \frac{Adj([A])}{Det[A]} \tag{2.7}$$

A matriz adjunta de uma matriz quadrada A é a matriz transposta da matriz que se obtém substituindo cada termo $A_{i,j}$ pelo determinante da matriz resultante A, para cada linha i e coluna j (determinante menor) multiplicado por $(-1)^{i+j}$ (ANTON; RORRES, 2016).

O processo de cálculo é feito em três etapas: Primeiro é calculada a transposta da matriz de entrada, A (trocando-se as linhas pelas colunas). Em seguida, é calculado o determinante da matriz menor, que é formada após eliminar os elementos na linha e coluna específica ao índice i e j. Finalmente, o cofator de cada elemento é obtido multiplicando-se a matriz de menores por $(-1)^{(i+j)}$, conforme mostrado na figura 2.1.

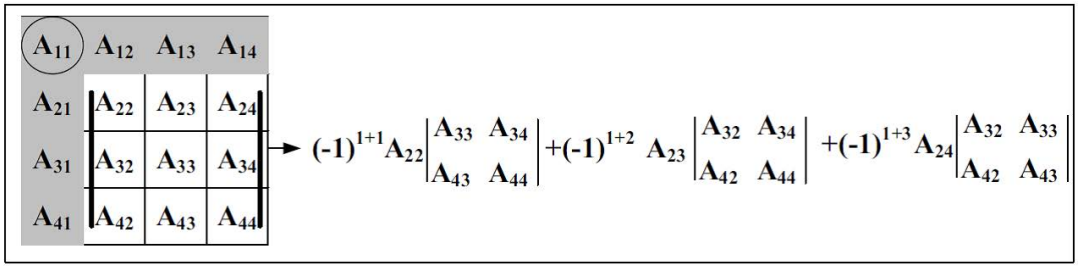

Figura 2.1 – Inversão de Matriz com abordagem analítica, mostrando como exemplo o primeiro elemento da matriz dos cofatores C_{11}

Fonte: (IRTÜRK, 2009)

2.7 Comparação entre os Métodos de Inversão de Matrizes

Nas seções anteriores deste capítulo foram abordados brevemente alguns métodos clássicos de inversão de matrizes. Nas figuras 2.2 e 2.3, são apresentados gráficos comparativos relativos ao número de operações, entre o método de decomposição de Cholesky, LU e QR e entre os métodos QR e o analítico, respectivamente (IRTÜRK, 2009).

A figura 2.2 apresenta uma relação entre o número de operações necessárias para a obtenção da inversão de uma determinada matriz A, em função de suas dimensões ($n \times n$). Conforme pode ser observado, para todas as dimensões de matrizes simuladas, o método de

decomposição QR mostrou um maior número de operações necessárias para a inversão da matriz. Entretanto, para os outros dois métodos um fato interessante ocorre quando as dimensões da matriz vão aumentando. No caso de matrizes com dimensões inferiores a 3×3 a inversão utilizando o método de fatoração LU apresentou um número de operações levemente inferior ao método de Cholesky. A partir dessa dimensão, os resultados relativos ao número de operações se invertem, ou seja, o método de Cholesky apresenta um número também levemente inferior em relação ao método LU. Dessa forma, considerando a complexidade computacional para o cálculo de inversão de uma matriz, as dimensões devem ser observadas para a adequada escolha do método a ser utilizado.

Vale ressaltar que os métodos de decomposição Cholesky e LU apresentam restrições com relação ao tipo de matriz, ou seja, a decomposição Cholesky é utilizada apenas para matrizes definidas positivas e para a decomposição LU a matriz deve ser estritamente diagonal dominante, ao passo que o método de decomposição QR e o método analítico não apresentam restrições, isto é, ambos os métodos podem ser aplicados a qualquer tipo matriz, desde que a mesma seja inversível.

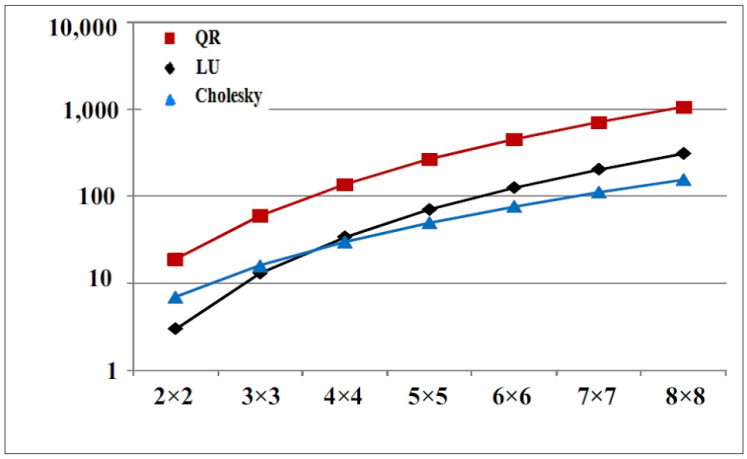

Figura 2.2 – Número Total de Operações em função do tamanho da matriz, para diferentes métodos de decomposição (Cholesky, LU e QR), em escala logarítmica.
Fonte: (IRTÜRK, 2009)

Na figura 2.3 são comparados os métodos de decomposição QR e analítico para inversão de matrizes. No gráfico fica evidente o aumento do número de operações no método analítico à medida que a dimensão da matriz aumenta. O mesmo ocorre para o método QR, mas de

forma bem mais suave. Dessa forma, o método analítico, em comparação com os demais métodos abordados, é o que apresenta uma maior complexidade computacional quando tem seu algoritmo implementado, para matrizes maiores do que 4×4.

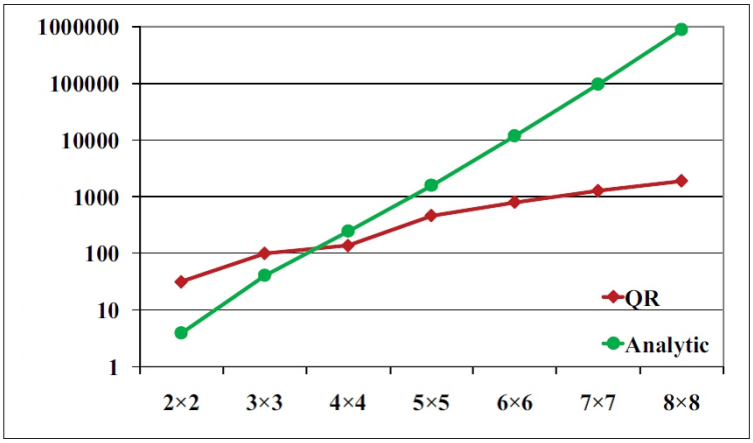

Figura 2.3 – Número Total de Operações para os métodos de inversão de matrizes baseado em decomposição QR e a método analítico, em escala logarítmica.
Fonte: (IRTÜRK, 2009)

De acordo com o trabalho de (GARCÍA, 2011), e como observado nesse capítulo, os métodos de decomposição são métodos preferidos para a inversão de matrizes em virtude de apresentarem um aumento suave no número de operações necessárias para matrizes com ordem elevada ($n = 5, 6, 7, \ldots$). De forma oposta, a complexidade algébrica dos métodos analíticos aumenta linearmente à medida que a ordem da matriz aumenta.

2.8 Resumo do Capítulo

Este capítulo apresentou os principais algoritmos para a inversão de matrizes, onde foi mostrado que a complexidade do método a ser escolhido, depende da aplicação, que define o tamanho necessário da matriz a ser implementada. Também foi mostrada uma comparação entre os diferentes métodos para matrizes de diferentes tamanhos. De acordo com o gráfico apresentado nas comparações, foi possível verificar que o método analítico é o mais adequado para a implementação de uma inversão de matriz 2x2, que é o tamanho alvo dessa dissertação de mestrado, pois, como será visto, este é o tamanho suficiente para uma futura implementação do algoritmo AP. O próximo capítulo aborda os principais algoritmos de divisão.

3 ALGORITMOS DE DIVISÃO

Neste capítulo se apresenta uma explanação sobre alguns dos algoritmos divisores consolidados na literatura, abordando suas principais características.

3.1 Algoritmos de Dígitos Recorrentes

A classe de algoritmos de divisão mais simples e mais amplamente implementada é a de dígitos recorrentes (OBERMAN; FLYNN, 1995). Isto se deve à sua baixa complexidade, considerando o número de operações necessárias para o cálculo.

Um ponto fundamental no projeto desse divisor é a escolha da base e da raiz (*radix*), os quais definem os dígitos do quociente permitidos e a representação do resto intermediário, respectivamente.

O algoritmo de dígitos recorrentes calcula o quociente (um dígito de cada vez) de maneira semelhante ao que é feito no cálculo aritmético convencional, com lápis e papel.

Como já destacado, a principal vantagem dos algoritmos de dígitos recorrentes é a sua simplicidade de implementação. Isto permite que sua implementação em uma arquitetura dedicada de *hardware* seja uma opção viável.

3.1.1 Algoritmo de Divisão *Restoring*

Uma operação de divisão pode ser representada matematicamente por um operador denominado dividendo (*N*) e um operador denominado divisor (*D*), resultando em uma operação *Q*, conforme mostra a equação (3.1).

$$Q = \frac{N}{D} \tag{3.1}$$

Como os resultados de uma divisão nem sempre são exatos, é possível reescrever a equação na forma da equação (3.1).

$$N = (Q \times D) + R \qquad onde\ R < D \tag{3.2}$$

De acordo com (OBERMAN; FLYNN, 1997), na divisão *Restoring* convencional, o resto da divisão obedece à seguinte condição: $0 \leq R_{(i+1)} < D$. Para isso, o dígito do quociente

q_{i+1} é obtido executando uma sequência de subtrações e deslocamentos ($Q = ...q_2q_1q_0$). Assim, em cada iteração, D é subtraído do resto parcial $r^{(i)} \times R_{(i)}$ (r é o *radix*) até que a diferença se torne negativa. A partir disso, $r^{(i)} \times R_{(i)}$ é adicionado novamente a esta diferença negativa, justificando a nomenclatura de algoritmo de *Restoring*. Finalmente, a última subtração é cancelada pela adição. Isso resultará no quociente de dígitos, que é determinado pelo número de subtrações ($q_{(i)} = N^{\text{o}}$ de subtrações -1). No pior caso, será necessária $q_{i+1} + 1$ subtração e mais uma adição para se obter $q_{(i)}$. Assim, tem-se a equação 3.3.

$$R_{(i)} = R_{(i+1)} - q_{(i)} \times D \times r^{(i)} \tag{3.3}$$

onde $i = n-1, n-2, ..., 1, 0$.

Para um sistema numérico binário ($r = 2$), e sendo os possíveis dígitos do quociente $\{0, 1\}$, logo se determina a equação 3.4.

$$R_{(i+1)} - q_{(i)} \times d \times 2^{(i)} = R_{(i)} \tag{3.4}$$

Assumindo como pressuposto inicial $q_{(i)} = 1$, a seguinte subtração pode ser representada pela equação 3.5.

$$R_{(i+1)} - D \times 2^{(i)} = R_{(i)} \tag{3.5}$$

Considerando para simplificação que, tanto o dividendo como o divisor são números positivos, tem-se o seguinte:

$$q_{(i)} = \begin{cases} 1, & \text{se} \quad R_{(i)} \geq 0 \quad \text{, então o pressuposto está correto.} \\ 0, & \text{se} \quad R_{(i)} < 0 \quad \text{, então o pressuposto está incorreto} \\ & \qquad\qquad\qquad \text{e uma operação } \textit{restoring} \text{ é necessária.} \end{cases} \tag{3.6}$$

Assim, a divisão *Restoring* binária requer no máximo uma subtração e uma adição para determinar um dígito do quociente. Essa adição é necessária para restaurar o resto parcial correto, conforme mostra a equação 3.7.

$$R_{(i+1)} + D \times 2^{(i)} = R_{(i)} \tag{3.7}$$

Desta forma, pode-se concluir que, em um caso limite, a divisão *restoring* pode necessitar de até duas vezes o número de ciclos de relógio para concluir a divisão.

Para ilustrar o algoritmo de divisão *restoring*, a tabela 3.1 mostra o processo para a

divisão binária de $7/2$.

Tabela 3.1 – Exemplo de divisão com algoritmo *restoring*

$7 - 2 \times 2^4 = -25$		$q_{(4)} = 1$
$-25 + 2 \times 2^4 = 7$	*restore*	$q_{(4)} = 0$
$7 - 2 \times 2^3 = -9$		$q_{(3)} = 1$
$-9 + 2 \times 2^3 = 7$	*restore*	$q_{(3)} = 0$
$7 - 2 \times 2^2 = -1$		$q_{(2)} = 1$
$-1 + 2 \times 2^2 = 7$	*restore*	$q_{(2)} = 0$
$7 - 2 \times 2^1 = 3$		$q_{(1)} = 1$
$3 - 2 \times 2^0 = 1$		$q_{(0)} = 1$

Fonte: O Autor

3.1.2 Algoritmo de divisão *Non-Restoring*

Um método de divisão melhorado em relação ao apresentado na subseção 3.1.1 é o método de divisão chamado *Non-Restoring*. Neste método não é necessária a "adição *restoring*".

A operação de divisão *Q* pode ser expressa também de acordo com a equação 3.8.

$$N = (Q \times D) + R \qquad com\ |R| < D \tag{3.8}$$

onde:

- Q é o quociente;
- D é o divisor;
- N é o dividendo;
- R é o resto.

Na divisão *non-restoring* o pressuposto de que $D > 0$ e que $|R_{(i+1)}| < D$ permanece. Porém, o resto parcial $R_{(i+1)}$ pode assumir valores positivos ou negativos (HA; LIM, 2007). A operação a ser executada pode ser uma adição ou uma subtração, dependendo do valor do resto parcial, de acordo com a equação 3.9.

$$\begin{cases} R_{(i+1)} - D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} \geq 0; \\ R_{(i+1)} + D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} < 0. \end{cases} \tag{3.9}$$

Por consequência, o dígito do quociente será $+1$ ou -1, nunca igual a zero ($q_{(i+1)} \in \{-1, 1\}$). A seleção do dígito do quociente pode ser representada da seguinte forma:

$$q_{(i)} = \begin{cases} 1, & \text{se} \quad R_{(i)} > 0; \\ -1, & \text{se} \quad R_{(i)} < 0 \end{cases} \tag{3.10}$$

Quando o valor de $R_{(i)} = 0$ o processo de divisão pode ser concluído (LI; CHU, 1996). Como resultado, o quociente Q é representado por um código de dígitos sinalizados, o qual não contém zeros.

Na divisão *restoring*, quando $(2R_i - D) < 0$, o resto pode ser assumido $2R_i$. Após, o deslocamento e uma nova subtração de D, $(4R_i - D)$ são obtidos. Na divisão *non-restoring*, quando $(2R_i - D) < 0$, o processo continua com uma diferença negativa, que será corrigido pela adição de D na próxima iteração. Desta forma, $(2(2R_i - D)) + D = 4R_{(i)} - D$. Este resultado é idêntico ao encontrado na divisão *restoring*, mas sem a necessidade de ter o processo de restauração. Pode-se concluir assim que, a divisão *non-restoring* necessita de n ciclos de relógio para efetuar a divisão.

Para ilustrar o algoritmo de divisão *non-restoring*, a tabela 3.2 mostra o processo para a divisão binária de $7/2$.

Tabela 3.2 – Exemplo de divisão com algoritmo *non-restoring*

$7 - 2 \times 2^4 = -25$	$q_{(4)} = -1$
$-25 + 2 \times 2^3 = -9$	$q_{(3)} = -1$
$-9 + 2 \times 2^2 = -1$	$q_{(2)} = -1$
$-1 + 2 \times 2^1 = 3$	$q_{(1)} = 1$
$3 - 2 \times 2^0 = 1$	$q_{(0)} = 1$

Fonte: O Autor

Na Figura 3.1, mostra-se uma representação gráfica dos processos de divisão *restoring* e *non-restoring*. Da equação (3.8), tem-se a relação $N = QD + R$, representada pela equação eq4nrest.

$$Q = q_{(4)} \times 2^4 + q_{(3)} \times 2^3 + q_{(2)} \times 2^2 + q_{(1)} \times 2^1 + q_{(0)} \times 2^0 \tag{3.11}$$

A partir dos exemplos da Figura 3.1, obtidos das tabelas 3.1 e 3.2, têm-se os seguintes cálculos:

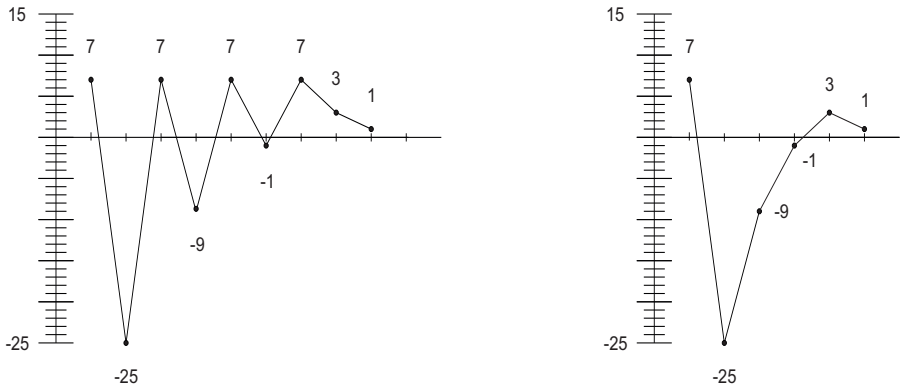

Figura 3.1 – Representação gráfica do cálculo do resto parcial nas divisões *Restoring* e *Non-Restoring*.
Fonte: O autor

$N = 7$ e $D = 2$

$Q = 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3$

$7 = (3 \times 2) + 1$

3.1.3 Algoritmo de divisão SRT

O algoritmo de divisão SRT é uma variação amplamente utilizada da divisão de dígitos recorrentes. Este algoritmo leva o nome de seus inventores Sweeney, Robertson e Tocher. Trabalhando de forma independente e praticamente ao mesmo tempo, D. W. Sweeney da IBM, J. E. Robertson da Universidade de Illinois e K. D. Tocher do Colégio Imperial de Londres propuseram um novo método de divisão binária. A motivação que levou ao algoritmo SRT foi a tentativa de acelerar a divisão *non-restoring* (que consiste em n operações de adição / subtração), permitindo que o digito zero seja um dígito do quociente para o qual nenhuma operação de adição / subtração seja necessária. Com isso, reduz-se a região de convergência do algoritmo *non-restoring*, obtendo-se redundância na seleção do quociente (HA; LIM, 2007).

No algoritmo SRT, assume-se que o divisor é uma fração normalizada na forma 0.1 d_2 d_3...d_n. Assim, tem-se a relação da equação 3.12.

$$\frac{1}{2} < |D| < 1 \tag{3.12}$$

É assumido também a relação da equação 3.13.

$$\frac{1}{2} < |2 \times R_{(i)}| < 1 \tag{3.13}$$

Isto significa que todos os dividendos parciais também são frações normalizadas. É importante lembrar que no algoritmo *non-restoring* o $q_{(i+1)} \in \{-1, 1\}$. Enquanto no algoritmo SRT, o conjunto de valores permitidos para o quociente será agora dado de acordo com a relação em 3.14.

$$q_{(i+1)} \in \{-1, 0, 1\} \tag{3.14}$$

Dessa forma, o divisor será deslocado ou somado ou subtraído do dividendo parcial, a partir da seleção $\{-1, 0, 1\}$, isto é:

$$\begin{cases} R_{(i+1)} + D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} < -D; \\ R_{(i+1)} = R_{(i)}, & \text{se} \quad -D \leq R_{(i+1)} \leq D; \\ R_{(i+1)} - D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} > D. \end{cases} \tag{3.15}$$

onde a regra e seleção do dígito do quociente serão definidos por:

$$q_{(i)} = \begin{cases} -1, & \text{se} \quad R_{(i)} < -D; \\ 0, & \text{se} \quad -D \leq R_{(i)} \leq D; \\ 1, & \text{se} \quad R_{(i)} > D. \end{cases} \tag{3.16}$$

Observar que tanto o divisor D, quanto o resto parcial $R_{(i+1)}$, são frações normalizadas ($\frac{1}{2} < |D| < 1$) e ($\frac{1}{2} < |2 \times R_{(i)}| < 1$). Se for utilizada a representação considerando que ($\frac{1}{2} < |D|$), a equação (3.15) tomará a forma da equação 3.17.

$$\begin{cases} R_{(i+1)} + D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} < \frac{1}{2}; \\ R_{(i+1)} = R_{(i)}, & \text{se} \quad -\frac{1}{2} \leq R_{(i+1)} \leq \frac{1}{2}; \\ R_{(i+1)} - D \times 2^{(i)} = R_{(i)}, & \text{se} \quad R_{(i+1)} > \frac{1}{2}. \end{cases} \tag{3.17}$$

onde a regra e seleção do dígito do quociente, representada na equação (3.16) fica reduzida a:

$$q_{(i)} = \begin{cases} -1, & \text{se} \quad |R_{(i)}| < -\frac{1}{2}, \quad \text{e o sinal de } q_{(i)} \text{ é negativo;} \\ 0, & \text{se} \quad |R_{(i)}| \leq \frac{1}{2}; \\ 1, & \text{se} \quad |R_{(i)}| > \frac{1}{2}, \quad \text{e o sinal de } q_{(i)} \text{ é positivo .} \end{cases} \tag{3.18}$$

A vantagem da utilização deste novo conjunto de regras para determinar o resto parcial e a seleção do dígito do quociente se resume ao fato de ser necessário apenas uma simples

comparação com a constante $\frac{1}{2}$ ou $-\frac{1}{2}$.

3.2 Algoritmos de Iterações Funcionais

Até agora foram abordadas algumas características e formas de funcionamento de alguns algoritmos de dígitos recorrentes mais amplamente utilizados. Nessa seção serão apresentados alguns algoritmos, denominados algoritmos de iterações funcionais, que apresentam características distintas dos estudados até o momento.

Diferente da divisão de dígitos recorrentes, que utiliza somas e subtrações como operações fundamentais, a divisão por iterações funcionais utiliza a multiplicação como operação fundamental. A principal dificuldade da divisão subtrativa é a convergência linear para o quociente (OBERMAN; FLYNN, 1997). Algoritmos de divisão baseados em multiplicações, porém, são capazes de tirar proveito de multiplicadores de alta velocidade, para convergir de forma quadrática para um resultado. Assim, ao invés de retirar um número fixo de bits de quocientes em cada ciclo, os algoritmos baseados em multiplicação são capazes de duplicar o número de bits de quociente corretos em cada iteração (OBERMAN; FLYNN, 1995).

3.2.1 Algoritmo de divisão Newton-Raphson

O algoritmo de Newton-Raphson efetua a divisão baseado em um processo multiplicativo.

Uma operação de divisão, onde se deseja obter o quociente Q, pode ser descrita como o produto do dividendo N pelo inverso do divisor D, como mostrado na expressão (3.19).

$$Q = \frac{N}{D} = N \times \frac{1}{D}, \tag{3.19}$$

neste ponto, o desafio está em como calcular de forma eficiente o inverso do divisor.

No algoritmo de Newton-Raphson, deve-se escolher uma função de iniciação que tem uma raiz no inverso do divisor (FLYNN, 1970). De forma geral, existem diversas raízes que poderiam ser usadas, tais como $\frac{1}{D}$, $\frac{1}{D^2}$, $\frac{N}{D}$ e $(1 - \frac{1}{D})$.

A escolha de qual raiz utilizar pode ser considerada como sendo de forma arbitrária, isto é, esta escolha é feita com base na conveniência de qual forma iterativa usar, sua taxa de convergência e o acréscimo necessário no tamanho da palavra binária durante o processamento. Isto se deve ao fato de usar uma raiz alvo diferente do verdadeiro quociente.

De acordo com (OBERMAN; FLYNN, 1995), a raiz mais comumente utilizada é o recíproco do divisor $\frac{1}{D}$ o qual é a raiz da função primitiva, ou seja

$$f_{(X)} = \frac{1}{X} - D = 0, \tag{3.20}$$

e a equação representando a convergência quadrática de Newton-Raphson é definida por

$$x_{(i+1)} = x_{(i)} - \frac{f(x_{(i)})}{f'(x_{(i)})} \tag{3.21}$$

Aplicando a equação de convergência quadrática de Newton-Raphson, equação (3.21), na equação (3.20), obtém-se a função e sua primeira derivada avaliadas no ponto $X_{(0)}$ (equações (3.22) e (3.23))

$$f_{(X_{(0)})} = \frac{1}{X_{(0)}} - D \tag{3.22}$$

$$f'_{(X_{(0)})} = -\frac{1}{X^2_{(0)}}. \tag{3.23}$$

Esses resultados serão usados para se obter uma aproximação para o inverso do divisor D. Assim, a partir da equação (3.21), têm-se:

$$X_{(1)} = X_{(0)} - \frac{f(X_{(0)})}{f'(X_{(0)})} \tag{3.24}$$

$$X_{(1)} = X_{(0)} + \frac{\frac{1}{X_{(0)}} - D}{\frac{1}{X^2_{(0)}}} \tag{3.25}$$

$$X_{(1)} = X_{(0)} \times (2 - D \times X_{(0)}) \tag{3.26}$$

onde da equação (3.26) é possível se obter uma equação geral de atualização do inverso do divisor do algoritmo de Newton-Raphson. Desta forma:

$$X_{(i+1)} = X_{(i)} \times (2 - D \times X_{(i)}) \tag{3.27}$$

Concluindo, a partir do inverso do divisor, com uma simples multiplicação pelo dividendo N, é possível obter o quociente da divisão Q, usando a divisão multiplicativa pelo método de Newton-Raphson.

3.2.2 Algoritmo de divisão Goldschmidt

O algoritmo de Goldschmidt (GOLDSCHMIDT, 1964) pertence à classe de algoritmos de iterações funcionais que se beneficia das convergências de séries matemáticas. Em particular, o algoritmo de Goldschmidt tira proveitos da convergência da série de Maclaurin (OBERMAN; FLYNN, 1997).

Como pôde ser verificado pela equação (3.27), para cada iteração do algoritmo de Newton-Raphson são necessárias duas multiplicações e uma subtração. A subtração é feita por uma operação simples de complemento, mas, as multiplicações, são dependentes uma da outra, isto é, somente é possível realizar uma das multiplicações a partir do resultado da multiplicação anterior. Este problema de dependência das operações de multiplicação, é resolvido pelo algoritmo de Goldschmidt, que o torna mais rápido do que o algoritmo Newton-Raphson (MARKSTEIN, 2004). Por este motivo, o algoritmo de divisão de Goldschmidt será alvo de abordagem dessa dissertação, e sua implementação numa arquitetura dedicada é proposta no próximo capítulo.

3.3 Resumo do Capítulo

Este capítulo abordou os principais algoritmos de divisão encontrados na literatura. Como pôde ser discutido, os métodos de iterações funcionais são mais rápidos em relação aos métodos de dígitos recorrentes, pois usam um método de aproximação, baseado em multiplicações sucessivas, que chega mais rapidamente à convergência de valores corretos. Entre os algoritmos de iterações funcionais, o método de Goldschmidt é mais rápido, pois as multiplicações são realizadas em paralelo, ao contrário do método de Newton-Raphson, onde as multiplicações são dependentes umas das outras. Esse aspecto, maior velocidade de convergência, norteou essa dissertação para a proposta de um novo algoritmo e uma nova arquitetura para o divisor Goldschmidt, cujos principais detalhes serão explicados no próximo capítulo.

4 ALGORITMO E ARQUITETURA DE GOLDSCHMIDT PROPOSTOS

A operação de divisão é geralmente considerada como uma operação de baixa frequência e alta latência em operadores aritméticos típicos (OBERMAN; FLYNN, 1997). Vários pesquisadores têm tentado propor algoritmos de divisão eficientes. Entretanto, essa não tem sido uma tarefa fácil, devido à dificuldade inerente da sua implementação em *hardware*.

Embora o uso da operação de divisão não seja tão comum, quando comparada à adição e multiplicação, em algumas aplicações, como algoritmos adaptativos (*Normalized Least Mean Square* - NLMS, *Affine Projection*-AP, entre outros) (ANGHEL et al., 2010) e SSIM *quality metric for image* (WANG et al., 2004), a divisão é uma das principais operações aritméticas envolvidas. Entre as operações aritméticas mencionadas, a operação de divisão é certamente a mais difícil de implementar em *hardware*, devido aos ajustes necessários para garantir a precisão das operações.

Alguns algoritmos de divisão conhecidos, tais como *restoring*, *non-restoring*, e SRT (BEHROOZ, 2000), são classificados como divisores lentos. Entretanto, quando se considera a operação rápida e a alta precisão, a divisão por convergência tem sido a escolha mais adequada, onde a divisão é realizada por multiplicações repetidas. Nesse aspecto, o método iterativo baseado em Newton-Raphson apareceu como o mais eficiente (KUCUKKABAK; AKKAS, 2004). No entanto, o algoritmo de Goldschmidt propicia uma grande melhoria ao método de refinamento recíproco, uma vez que reduz em grande parte o número de multiplicações necessárias para acelerar a divisão de convergência (KONG; SWARTZLANDER, 2008).

4.1 Aspectos dos Métodos Propostos

Neste trabalho de dissertação, propõe-se um novo e eficiente método para melhorar o algoritmo iterativo de Goldschmidt. O método proposto reduz o número de iterações para produzir o resultado final, com um circuito muito simples para realizar as operações. O método se baseia na escolha correta do valor ótimo do denominador, que faz o ajuste da primeira iteração do algoritmo, o que contribui para a convergência rápida. Além disso, o algoritmo proposto não tem restrições em relação à faixa de valores (o algoritmo convencional tem uma restrição de valores entre 1 e 2) e nem em relação ao sinal das entradas, aceitando quaisquer entradas na faixa dos valores suportados pelo padrão Q7.8, ou seja, valores entre -127.99609375 e +127.99609375. Observe que, embora esteja sendo usada esta faixa de valores, a solução proposta permite outros intervalos de valores, tendo-se apenas que usar o valor inicial apropriado

de aproximação no algoritmo proposto.

Neste ponto, cabe um pequeno comentário sobre o formato Q de representação de ponto-fixo.

A representação padronizada Q é um formato de número de ponto fixo onde o número de bits fracionários e o número de bits inteiros é especificado. As notações são escritas como Qm.n, onde:

- Q designa que o número está na notação de formato Q como uma alusão ao símbolo padrão para o conjunto de números racionais;
- m é o número de bits reservados para designar a porção inteira do número, em complemento de 2, que pode incluir ou não o bit de sinal (portanto, se m não é especificado, é tomado como zero ou um);
- n é o número de bits utilizado para designar a parte fracionária do número, isto é, o número de bits à direita do ponto binário. (Se n = 0, os números Q são inteiros - o que representa uma degeneração do padrão);

Existem duas convenções para Q, uma inclui o bit de sinal no valor de m, e a outra convenção não. A escolha da convenção pode ser determinada somando $m + n$. Se o valor for igual ao tamanho do registrador, o bit de sinal está incluso no valor de m. Se for um bit a menos do que o tamanho do registrador, o bit de sinal não está incluso no valor de m. Para um determinado formato $Qm.n$, tendo $m + n + 1$ bits, com o bit de sinal, com n bits fracionários:

- a faixa possível de ser representada é dada por: $[-(2^m), 2^m - 2^{-n}]$;
- e a resolução é dada por 2^{-n}.

Como exemplo, um número Q15 teria 15 bits fracionários; Um número Q1.14 teria 1 bit inteiro e 14 bits fracionários. O formato Q é frequentemente usado em hardware que não tem uma unidade de ponto flutuante e em aplicativos que utilizem resolução constante. No presente trabalho, é adotado uma representação em formato Q7.8, ou seja, 7 bits inteiros e 8 bits fracionários, e por simplificação, adotado a faixa de valores entre -127.99609375 e +127.99609375.

A seguir são apresentados os principais pontos de fundamentação teórica relacionados ao algoritmo e à arquitetura de Goldschmidt original. Em seguida, são apresentados os principais aspectos relacionados ao método proposto para o aumento da eficiência do algoritmo e da arquitetura de Goldschmidt.

4.1.1 Algoritmo de Goldschmidt Original

O algoritmo de Goldschmidt (GOLDSCHMIDT, 1964) pertence à classe de algoritmos de iterações funcionais que se beneficia das convergências de séries matemáticas. Em particular, o algoritmo de Goldschmidt tira benefícios da convergência da série de Maclaurin (OBERMAN; FLYNN, 1997).

O algoritmo de Goldschmidt para calcular a divisão $Q = \frac{N_{-1}}{D_{-1}}$ pode ser descrito da seguinte forma (MARKSTEIN, 2004). Seja *Dopt*= F_{-1} uma estimativa adequada para $\frac{1}{D_{-1}}$, onde uma aproximação adequada é aquela que atenda a $\frac{3}{4} \leq F_{-1}D_{-1} < \frac{3}{2}$. Note que para um valor inteiro n, a aproximação adequada é $\frac{3}{4} \leq |2^n D_{-1}| < \frac{3}{2}$. Portanto, fazendo-se $F_{-1} = sgn(D_{-1})2^n(2 - sgn(D_{-1})2^n D_{-1}$, onde $sgn(.)$ é a função sinal, esse critério de aproximação adequada é satisfeito. No entanto, normalmente o pressuposto inicial recíproco é muitas vezes determinado por tabelas (*look-up tables*) (CORPORTATION, 2001). Logo, multiplicando-se os termos dividendo e divisor por F_{-1}, modifica o cálculo de Q, de acordo com (4.1), onde o subíndice -1 representa a primeira amostra a ser processada.

$$Q = \frac{N_{-1}}{D_{-1}} = \frac{N_{-1}F_{-1}}{D_{-1}F_{-1}} = \frac{N_0}{D_0} \tag{4.1}$$

O objetivo do algoritmo de Goldschmidt é encontrar os valores F_i para que a multiplicação do dividendo e divisor de (4.1) possa levar o divisor para o valor unitário. Como consequência, o dividendo se torna o quociente. O erro relativo do pressuposto inicial é dado por (4.2), onde $D_0 = D_{-1}F_{-1} = 1 - e$.

$$e = \frac{\frac{1}{D_{-1}}F_{-1}}{\frac{1}{D_{-1}}} = 1 - D_{-1}F_{-1} \tag{4.2}$$

Pela escolha de F_{-1}, $|e| < 1$, e tomando $F_0 = 1 + e$ como o próximo multiplicador do dividendo e divisor, levaria o novo divisor a $1 - e^2$. Da equação (4.2), pode-se também escrever $F_0 = 2 - D_0$. Por outro lado, o termo Q pode ser escrito de acordo com (4.3), onde $N_i = N_{i-1}F_{i-1}$, $D_i = D_{i-1}F_{i-1}$ e $F_i = 2 - D_i$

$$Q = \frac{N_i}{D_i} = \frac{N_{i-1}F_{i-1}}{D_{i-1}F_{i-1}} = \frac{N_{i+1}}{D_{i+1}} \quad i = 0, 1, 2, ... \tag{4.3}$$

O atrativo da técnica proposta, para a implementação em *hardware*, é que $F_i = 2 - D_i$ pode ser derivado diretamente de D_i por inversão binária, cuja implementação é simples e requer pouco recurso de *hardware*. As grandezas D_i e F_i estão essencialmente disponíveis simultaneamente. Assim que N_i e D_i fiquem disponíveis, N_i, D_i e F_i são reaplicadas às entradas dos

multiplicadores. Este processo continua até que D_i se torne suficientemente próximo de 1, ou seja, na prática, para um número fixo de iterações determinado pela qualidade de $F_{-1} = Dopt$. Correspondendo ao divisor final D_i, o termo N_i é considerado o quociente. Até esse ponto, a aritmética foi assumida como exata. Entretanto, na prática, cada multiplicação é arredondada com alguma precisão, e a subtração $2 - D_i$ também pode ser submetida a arredondamentos. Com um multiplicador paralelo, os cálculos de N_i são intercalados com os de D_i, e toda a iteração pode ocorrer em um único ciclo do multiplicador.

4.1.2 A Arquitetura Goldschmidt Original

O diagrama para o divisor Goldschmidt original é apresentado na Figura 4.1 (SÁNCHEZ et al., 2009). O valor da aproximação inicial do termo divisor é lido a partir de uma *Look-up table* (valor $L1$). Após a aproximação inicial, os valores D e N são inicialmente submetidos a multiplicações por $L1$, e depois disso, entregues aos multiplicadores ($q1$ e $e1$), que para a primeira iteração ($i = 0$), habilita os resultados parciais para novas multiplicações. Para as novas iterações ($i > 0$), os resultados parciais produzidos anteriormente são multiplicados novamente pelo termo de erro ($e1$), que representa a operação ($2 - D$). Este processo permanece até que haja um número suficiente de iterações capaz de reduzir o erro para um valor predeterminado. No circuito divisor Goldschmidt original da Figura 4.1 são utilizados circuitos multiplicadores da ferramenta de síntese.

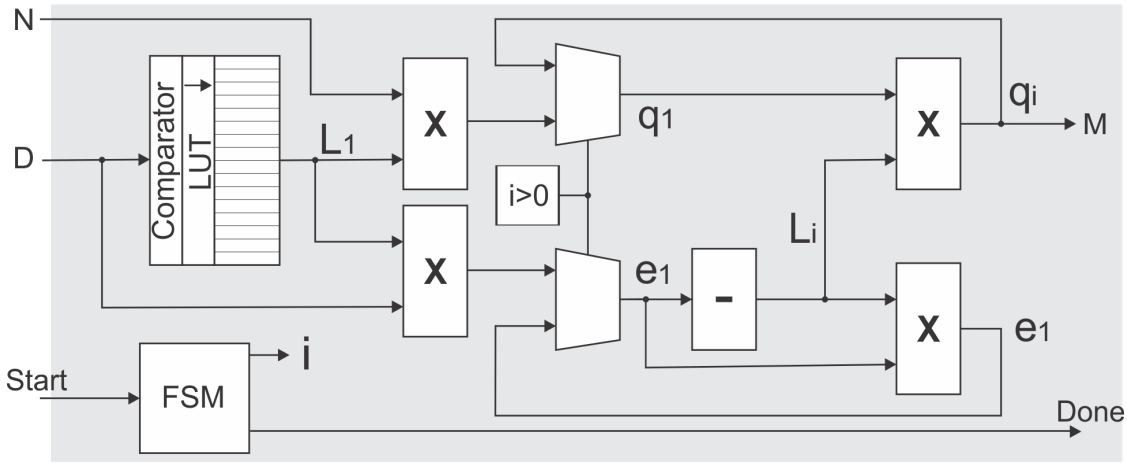

Figura 4.1 – Diagrama de blocos do divisor Goldschmidt original
Fonte: (Modificado de (SÁNCHEZ et al., 2009))

4.1.3 Trabalhos relacionados

Ao longo dos anos, alguns trabalhos têm proposto diferentes métodos tentando acelerar a convergência do algoritmo de Goldschmidt. Em (ERCEGOVAC et al., 2000), por exemplo, é proposto um método para acelerar cálculos de divisão, raiz quadrada e raiz quadrada recíproca quando o método de Goldschmidt é usado, em uma abordagem multiplicadora em paralelo. O método propõe substituir a última iteração adicionando um termo de correção que pode ser pesquisado durante as iterações iniciais usando uma *look-up table* (LUT). Os autores relatam que os resultados corretos só são obtidos a partir da quarta iteração do algoritmo. O uso de LUTs também é explorado em (ITO; TAKAGI; YAJIMA, 1997) e (MATULA, 2001), onde enquanto em (ITO; TAKAGI; YAJIMA, 1997) é proposto um método eficiente para a abordagem inicial em divisões-multiplicativas, em (MATULA, 2001), propõe-se uma melhoria da LUT para um método de divisão pós-escalonada.

O método de arredondamento proposto em (KONG; SWARTZLANDER, 2008) aplica um método de truncamento no passo de iteração final. Embora os autores garantam resultados com apenas 3 iterações do método de Goldschmidt para a divisão, não é tratada qualquer mudança em relação à escolha do valor inicial de F_{-1} para a primeira iteração, que é o foco do trabalho proposto no âmbito desta dissertação. Além disso, diferente dos outros algoritmos, o método que é proposto nessa dissertação, assegura valores de entrada fora do intervalo $1 \leq D < 2$.

Em termos de simplificações em *hardware*, o trabalho proposto em (ROY,) usa uma unidade de realimentação, onde a área total consumida pode ser reduzida da implementação original, uma vez que os blocos multiplicadores extras são removidos. Por outro lado, em (HOSSEINY; JABERIPUR, 2016) é proposta uma divisão radix-10 para o algoritmo de Goldschmidt, com um tamanho reduzido da LUT, que é obtido através de análise matemática. No entanto, em ambos os trabalhos mencionados não há comentários sobre a frequência de operação, dissipação de energia e eficiência na convergência. Todos os trabalhos mencionados da literatura exploram algoritmos de divisão em arquitetura de ponto flutuante, o que demanda muitos recursos em termos de *hardware*. Por outro lado, a solução proposta nesta dissertação apresenta um divisor Goldschmidt rápido, com apenas 3 iterações, eficiente em termos energéticos e em uma arquitetura de ponto-fixo, que é menos custosa em termos de *hardware* do que uma arquitetura em ponto flutuante.

4.2 Algoritmo e Arquitetura Goldschmidt Propostos

Como pôde ser visto na seção anterior, a escolha correta do valor ótimo do denominador ($F_{-1} = Dopt$), que faz o ajuste da primeira iteração do algoritmo, é essencial para a convergência rápida. Essa escolha, na prática, determina o número de iterações necessárias para obter o resultado da divisão $Q = \frac{N_i}{D_i}$.

4.2.1 Algoritmo Proposto

O *hardware* que foi implementado, usa uma arquitetura com sinal, que opera em complemento de 2, com aritmética de ponto fixo no padrão Q7.8. O algoritmo para escolher o multiplicador ótimo para a primeira iteração é baseado apenas nos dois primeiros dígitos significativos do denominador, e é realizado de acordo com as seguintes etapas:

- Encontre a posição do primeiro bit 1 mais significativo de D. Considere n como o valor do expoente da representação binária desse bit. Isso determina uma transformação radix 2^b , onde $b = -(n + 1)$;
- Coloque em 1 o bit na posição $b = -(n + 1)$ de $Dopt = F_{-1}$;
- Ajuste o bit na posição $(b - 1)$ de $Dopt$ como a inversão do bit na posição $n - 1$ de D;
- Mantenha todos os outros bits de $Dopt$ em zero.

Para mostrar a formação do valor de $Dopt$, de acordo com o algoritmo, tomaremos como exemplo um denominador com valor igual a 20,69921875. Este número, com os respectivos pesos de suas casas em binário, representado em formato Q7.8 é mostrado na Tabela 4.1.

Tabela 4.1 – Algoritmo de formação de $Dopt$ - Obtenção de n.

Ex: 20,69921875	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Denominador	0	0	0	1	0	1	0	0,	1	0	1	1	0	0	1	1
				↑												
Valor de n:				4												

A partir da Tabela 4.1 pode ser visto que o primeiro valor 1 mais significativo do Denominador está na posição 2^4, o que significa que o valor de n deve ser tomado como igual a 4. Com $n = 4$, calcula-se $b = -(n + 1) = -5$

Tabela 4.2 – Algoritmo de formação de $Dopt$ - Obtenção do bit na posição b.

Ex: 20,69921875	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Denominador	0	0	0	1	0	1	0	0,	1	0	1	1	0	0	1	1
Dopt:													1			

Então deve ser colocado em 1 o bit da posição de $b = -5$ de $Dopt$, ou seja, setado o bit na posição 2^{-5} como pode ser visto na Tabela 4.2.

Tabela 4.3 – Algoritmo de formação de $Dopt$ - Obtenção do bit na posição $(b-1)$.

Ex: 20,69921875	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Denominador	0	0	0	1	0	1	0	0,	1	0	1	1	0	0	1	1
					↑									↓		
Dopt:													1	1		

Na etapa seguinte, deve ser ajustado o bit na posição $(b-1)$ de $Dopt$ como o inverso do bit na posição $(n-1)$ do Denominador, conforme pode ser visto na Tabela 4.3.

Tabela 4.4 – Algoritmo de formação de $Dopt$ - Valor final de $Dopt$.

Ex: 20,69921875	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Denominador	0	0	0	1	0	1	0	0,	1	0	1	1	0	0	1	1
					↑									↓		
Dopt:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Após, devem ser mantidos todos os bits restantes em zero, conforme pode ser visto na Tabela 4.4. Por este exemplo, sendo o Denominador igual a 00010100,10110011, resultou em um valor de aproximação inicial $Dopt = 00000000,00001100$

Seguindo as etapas do algoritmo, é possível gerar todos os valores possíveis de $Dopt$ com base nas faixas de valores do denominador. Estes valores são mostrados na Tabela 4.5. Na Tabela 4.5, os módulos das variáveis Min e Max representam os limites do intervalo de valores que são possíveis em função do valor do denominador. Como um outro exemplo, na primeira linha da Tabela 4.5, tem-se o último intervalo de valores permitido pelo formato Q7.8 (96.0 a 127.99609375). O primeiro dígito mais significativo em 1, determina $n = 6$, e consequentemente, $b = -7$, ou seja, sendo o denominador = 011XXXXX.XXXXXXXX, tem-se o primeiro bit 1 na posição 6 de D, o que determina um bit em 1 na posição -7 de $Dopt$ ($Dopt = 0000000000000010$). O bit na posição $b-1$ (posição -8) de $Dopt$ é o inverso do bit da posição $n-1$ de D (posição 5) ($Dopt = 0000000000000010$). Todos os outros bits são iguais a 0. Os bits representados por X significam que são bits irrelevantes para a operação.

Tabela 4.5 – Faixa de cobertura e *Dopt* em função do Denominador.

Denominador	|Min|	|Max|	*Dopt*
011XXXXXXXXXXXX	96.0	127.99609375	0000000000000010
010XXXXXXXXXXXX	64.0	95.99609375	0000000000000011
0011XXXXXXXXXX	48.0	63. 99609375	0000000000000100
0010XXXXXXXXXX	32.0	47. 99609375	0000000000000110
00011XXXXXXXXXX	24.0	31. 99609375	0000000000001000
00010XXXXXXXXXX	16.0	23. 99609375	0000000000001100
000011XXXXXXXXX	12.0	15. 99609375	0000000000010000
000010XXXXXXXXX	8.0	11. 99609375	0000000000011000
0000011XXXXXXXX	6.0	7. 99609375	0000000000100000
0000010XXXXXXXX	4.0	5. 99609375	0000000000110000
00000011XXXXXXX	3.0	3. 99609375	0000000001000000
00000010XXXXXXX	2.0	2. 99609375	0000000001100000
000000011XXXXXX	1.5	1. 99609375	0000000010000000
000000010XXXXXX	1.0	1. 49609375	0000000011000000
0000000011XXXXX	0.75	0. 99609375	0000000100000000
0000000010XXXXX	0.5	0. 74609375	0000000110000000
00000000011XXXX	0.375	0. 49609375	0000001000000000
00000000010XXXX	0.25	0.37109375	0000001100000000
000000000011XXX	0.1875	0.24609375	0000010000000000
000000000010XXX	0.125	0.18359375	0000011000000000
0000000000011XX	0.09375	0.12109375	0000100000000000
0000000000010XX	0.0625	0.08984375	0000110000000000
00000000000011X	0.046875	0.05859375	0001000000000000
00000000000010X	0.03125	0.04296875	0001100000000000
000000000000011	0.0234375	0.02734375	0010000000000000
000000000000010	0.015625	0.01953125	0011000000000000
000000000000001	0.0078125	0.01171875	0110000000000000

4.2.2 Arquitetura Proposta

O circuito divisor implementado agrega internamente um circuito multiplicador eficiente da literatura (COSTA, 2002), que opera na base-4.

4.2.2.1 Circuito Multiplicador na Base 4

Entre os operadores aritméticos, os módulos multiplicadores são os mais comuns em operações DSP. Uma nova arquitetura para operações de multiplicação em complemento de 2 está apresentada em (COSTA, 2002). A arquitetura proposta mantém a forma original de um multiplicador *array* convencional. Todos os bits nos produtos parciais são tratados como

bits sem sinal, exatamente como um multiplicador *array* normal, exceto para o último bit de todas as linhas de produto parcial e todos os bits da última linha. Esta arquitetura é estendida para codificação na base 2^m, que permite a redução do número de linhas de produtos parciais, proporcionando significativos ganhos em desempenho e redução da dissipação de potência. nessa dissertação de mestrado, utiliza-se o valor m=2, ou seja, multiplicador na base 4. Na

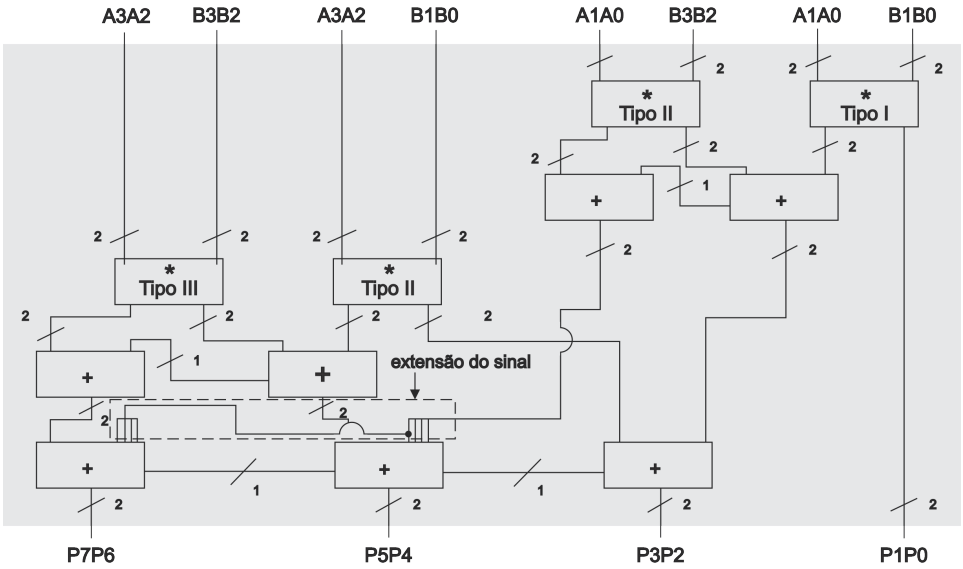

Figura 4.2 – Estrutura do multiplicador *array* de 4 bits na base 4.
Fonte: Modificado de (COSTA, 2002)

operação de multiplicação na base 4, os operandos são divididos em grupos de 2 bits. Cada um desses grupos pode ser visto como uma representação de um dígito em uma base 4. A estrutura da arquitetura do multiplicador *array* na base 4 é a mesma do que um multiplicador *array* convencional. Entretanto, cada linha do produto parcial opera em grupos de 2 bits, ao invés de uma operação bit a bit. Isso reduz o número de linhas de produtos para W/2, onde W é o número de bits. A Figura 4.2 mostra a arquitetura de um multiplicador *array* de 4 bits na base 4.

Como pode ser observado na Figura 4.2, em um multiplicador de W bits são necessárias W/2 linhas de produtos, tendo cada uma W/2 módulos básicos multiplicadores 2 por 2, e o mesmo número de 2 por 2 somadores. Uma linha adicional composta por (W/2) +1 desses elementos somadores básicos é responsável pela soma dos termos dos produtos parciais.

Para a construção de uma arquitetura *array* na base 4, os módulos do produto parcial no lado esquerdo e na parte inferior precisam ser diferentes para produzir o sinal dos operandos. Desta forma, são construídos três tipos de módulos. Tipo I são módulos de multiplicação sem sinal. Por outro lado, os módulos Tipo-II produzem os produtos parciais de uma operação de um valor sem sinal com um valor em complemento de 2. Finalmente, os módulos Tipo III realizam

a operação de dois valores com sinal. Note na Figura 4.2 a indicação da extensão de sinal que é necessária para a realização das operações de somas dos produtos parciais em complemento de 2.

Nas arquiteturas dos multiplicadores *array* na base 4, pode-se observar o relacionamento entre a complexidade dos blocos básicos e a quantidade de linhas de produtos parciais. Quanto maior o valor da base, menor será o número de linhas de produtos parciais, e maior a complexidade dos módulos somador e multiplicador, conforme mostrado em (COSTA, 2002). Entretanto, é possível gerar módulos otimizados para valores na base 4, que é utilizada nessa dissertação.

4.2.2.2 Arquitetura Completa do Divisor

O circuito que implementa o algoritmo de Goldschmidt proposto pode ser visto na Figura 4.3. No circuito proposto, o denominador é entregue ao bloco que calcula a primeira aproximação $Dopt$. Esse valor é então entregue ao primeiro conjunto de multiplicadores e um subtrator, quando é feito o cálculo de erro (representado por F_0). Após, os termos F_0, D_0, e N_0 são aplicados ao segundo bloco, e depois ao terceiro bloco de multiplicadores e subtrator, quando a operação é concluída, uma vez que o algoritmo obtém os resultados em apenas 3 iterações. Os blocos de controle são responsáveis por ajustar o sinal do resultado, em complemento de 2. A principal contribuição para o aumento da velocidade de convergência é o algoritmo que calcula $Dopt$, levando a multiplicação inicial para um valor próximo ao valor ótimo, o que reduz o número de iterações necessárias para reduzir o erro a um valor predeterminado. Reduzir o número de iterações faz com que ocorra redução na quantidade de multiplicadores, o que leva à redução da dissipação de energia e ao aumento da velocidade. O *hardware* que foi implementado, utiliza entradas de 16 bits, opera os multiplicadores internos em 24 bits e a saída é ajustada para 16 bits. A partir do circuito otimizado, são obtidos resultados ótimos com apenas 3 iterações, como pode ser visto nos exemplos da Tabela 4.6. Observe nos exemplos que, a partir das escolhas adequadas dos valores de $Dopt$, os cálculos já produzem boas aproximações dos valores desejados já na primeira iteração. As duas próximas iterações servem para refinar ainda mais os resultados desejados.

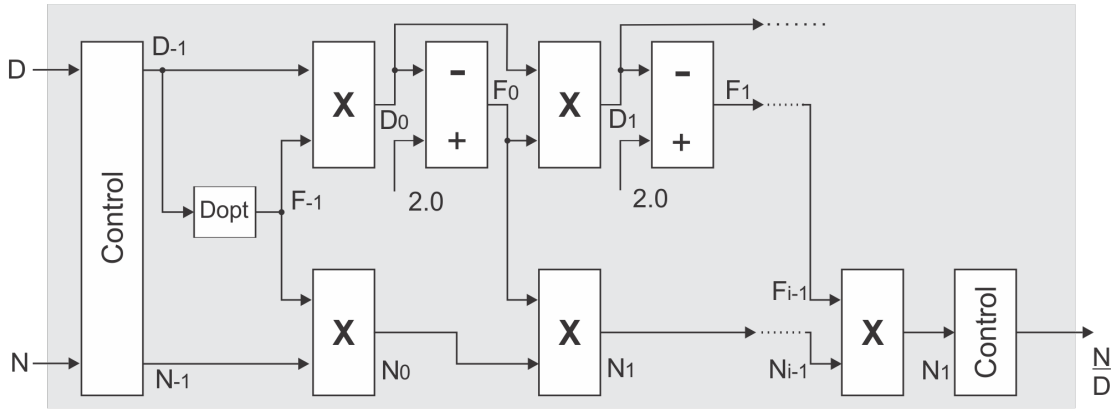

Figura 4.3 – Diagrama de blocos do divisor Goldschmidt melhorado proposto.
Fonte: O Autor

Tabela 4.6 – Exemplos de divisões obtidas pelo algoritmo proposto, com apenas 3 iterações.

	Exemplo 1	Exemplo 2	Exemplo 3	Exemplo 4
N	1,515625	32,125	0,140625	0,1640625
D	1,15234375	12,5	0,359375	0,44921875
Valor desejado	**1,3125**	**2,56640625**	**0,390625**	**0,36328125**
Iteração 1	1,291015625	2,447021484	0,388916016	0,361450195
Iteração 2	1,314804077	2,564102173	0,391281128	0,36517334
Iteração 3	**1,3125**	**2,56640625**	**0,390625**	**0,36328125**

Fonte: O Autor

4.3 Resumo do Capítulo

Este capítulo mostrou um algoritmo Goldschmidt melhorado, orientado a *hardware*, com melhor capacidade de convergência, para permitir realizar a divisão com mais paralelismo em uma única interação. Também foi mostrada a arquitetura proposta do divisor Goldschmidt, que utiliza um multiplicador da literatura, que opera na base 4. Como pôde ser observado, o algoritmo proposto permite que o algoritmo Goldschmidt opere em uma faixa mais ampla de valores a partir do uso do formato Q7.8. O algoritmo proposto permite que a arquitetura do divisor alcance a convergência em apenas 3 iterações. O próximo capítulo aborda o uso da solução Goldschmidt proposta em arquitetura de inversão de matrizes.

5 ARQUITETURAS DOS CIRCUITOS DE INVERSÃO DE MATRIZES

Neste capítulo será abordado a arquitetura dos circuitos de inversão de matrizes propostos neste trabalho. O objetivo desta arquitetura de inversão de matrizes é sua utilização futura em uma implementação do algoritmo adaptativo de projeções afins. Na seção seguinte será feita uma breve abordagem sobre o algoritmo AP e suas equações de atualização para um entendimento da necessidade da arquitetura de inversão de matrizes.

5.1 O algoritmo adaptativo de Projeções Afins (*Affine Projection* - AP)

O algoritmo de projeções afins (AP) proposto por Ozeki e Umeda em 1984 (OZEKI; UMEDA, 1984) aplica pesos de atualizações em direções que são ortogonais aos últimos P vetores de entrada. Isto descorrelaciona o sinal de entrada e acelera a convergência (RUPP, 1998), fazendo o algoritmo atrativo para aplicações com sinais de entrada altamente correlacionados (colorido) (SANKARAN; BEEX, 2000). O preço a ser pago para o melhor desempenho é o aumento da complexidade computacional, quando comparado com outros algoritmos, tais como o *normalized least mean squares* (NLMS) (ALMEIDA et al., 2005), que, por sua vez, tem uma complexidade computacional muito maior quando comparado ao LMS tradicional. Esse aumento da complexidade computacional vem deixando de ser um fator inviabilizador devido aos avanços recentes na indústria de semicondutores.

O algoritmo AP executa as seguintes operações para atualizar os coeficientes do filtro adaptativo:

$$d(n) = \mathbf{w}_o^T \mathbf{u}(n) + r(n) \tag{5.1}$$

$$e(n) = d(n) - \mathbf{w}^T(n)\mathbf{u}(n) \tag{5.2}$$

$$\hat{\mathbf{a}}(n) = [U^T(n)U(n)]^{-1}U^T(n)\mathbf{u}(n) \tag{5.3}$$

$$\Phi(n) = \mathbf{u}(n) - U(n)\hat{\mathbf{a}}(n) \tag{5.4}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \alpha \frac{\Phi(n)}{\Phi^T(n)\Phi(n)} e(n) \tag{5.5}$$

onde:

- $\mathbf{u}(n)$ - sinal de excitação (vetor de entrada);
- $r(n)$ - ruído aditivo;
- $d(n)$ - resposta desejada;
- $e(n)$ - erro de estimação;
- $\hat{\mathbf{a}}(n)$ - vetor de P erros mínimos quadráticos;
- $\mathbf{w}(n)$ - vetor dos coeficientes do filtro;
- α - passo de adaptação (no caso do AP, $\alpha = 1$);
- $U(n)$ - matriz formada por P vetores passados de entrada;
- $\Phi(n)$ - vetor ortogonal a P vetores passados do sinal de entrada.

Pode ser visto na Equação 5.3 a inversão de matriz necessária para a atualização de $\hat{\mathbf{a}}(n)$, o vetor de P erros mínimos quadráticos. Esta inversão de matrizes deve fazer a inversão da matriz resultado da operação $[U^T(n)U(n)]$, que têm dimensão $P \times P$. Conforme estudos realizados, o algoritmo AP quando utilizado em aplicações como cancelamento de eco acústico, se mostra eficiente com apenas 2 vetores passados do sinal de entrada, o que representa a necessidade de inversão de uma matriz de dimensão 2×2.

5.2 Arquitetura Geral de Inversão de Matrizes

No capítulo 2 foi feita uma análise da complexidade computacional das diversas formas de inversão de matrizes. A proposta neste trabalho de dissertação é implementar um circuito eficiente de inversão de matrizes, e a opção foi a de implementar a inversão de matrizes 2×2. Essa escolha está baseada no fato de que para uma futura implementação do algoritmo de Projeções Afins (AP), esse tamanho de matriz é suficiente.

Para esse tamanho de matriz, conforme mostrou a Figura 2.3, o método analítico de inversão de matrizes apresenta menor complexidade computacional do que os outros métodos. Desta forma, o circuito a ser implementado deve efetuar a operação $A^{-1} = \frac{1}{detA} \times Adj(A)$

Todas essas escolhas foram direcionadas para que se tenha a implementação do circuito de divisão de matriz com eficiência em termos de velocidade de operação e aspecto de redução na dissipação de potência. A partir das premissas acima, foi descrita a arquitetura básica que compõe os diversos circuitos implementados. Esta arquitetura básica é apresentada na Figura 5.1. Nos diversos circuitos utilizados, a principal alteração está na forma de como realizar a

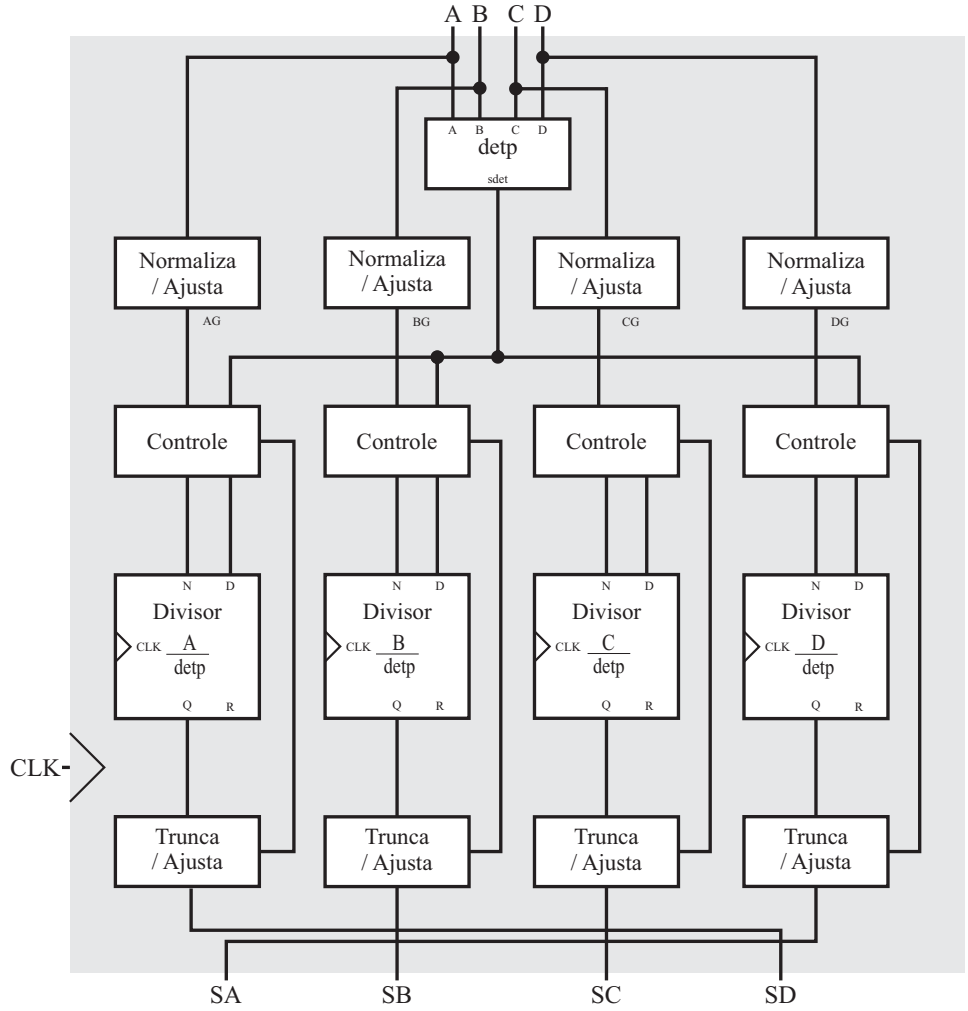

Figura 5.1 – Diagrama de Blocos do circuito inversor de matrizes básico.
Fonte: O Autor

divisão $\frac{Adj(A)}{detA}$. Nos circuitos de inversão de matrizes apresentados, o circuito que agrega a maior complexidade em termos de *hardware* é o divisor. Desta forma, os diversos circuitos divisores apresentados no capítulo anterior são explorados na arquitetura para a inversão de matrizes.

Na Figura 5.1 é apresentado o diagrama de blocos do circuito inversor de matrizes básico utilizado para implementação das várias estruturas sob estudos neste trabalho. A estrutura geral, da Figura 5.1, é detalhada nas subseções seguintes:

5.2.1 Estrutura de cálculo do Determinante da matriz

O bloco denominado "detp" na Figura 5.1 é responsável por calcular o determinante da matriz A. Seu diagrama de blocos é representado na Figura 5.2, onde pode ser observada a existência de dois blocos chamados "controle de sinal" (na entrada e na saída), dois blocos

"multiplicadores", um bloco "subtrator" e um bloco "Normalização e ajuste". A seguir é feito um breve detalhamento de suas funções:

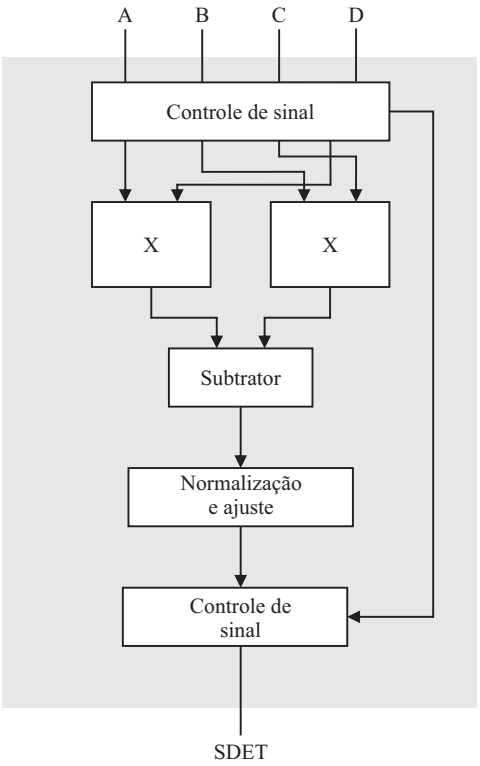

Figura 5.2 – Diagrama de Blocos do circuito de cálculo do determinante.
Fonte: O Autor

- Controle de sinal de entrada: Este bloco é responsável por realizar a verificação dos sinais das quatro entradas. Isto é necessário pois nem todos os multiplicadores utilizados nas diversas implementações efetuadas, operam com números negativos. Após esta verificação, e após armazenar os sinais das entradas, é efetuado um complemento de 2 dos valores negativos para entregar somente valores positivos aos multiplicadores;
- Multiplicadores: Responsáveis por efetuar as multiplicações $A \times D$ e $B \times C$. Estes circuitos foram implementados para utilizar entradas em 16 bits, em arquitetura de ponto-fixo e formato Q7.8. As saídas são em 32 bits.
- Subtrator: Este bloco efetua o cálculo de $A \times D$ - $B \times C$. Isto é feito através de uma soma de $A \times D$ com o complemento de 2 de $B \times C$;
- Normalização e ajuste: Este bloco efetua o arredondamento / truncagem do valor resultado da subtração, para entregar o formato correto para o bloco de controle do sinal de saída;

- Controle de sinal de saída: Este bloco faz a correção de sinal, efetuando uma operação de complemento de 2 ou não, em função das informações provenientes do bloco de controle de sinal de entrada e resultado do subtrator, produzindo uma saída com sinal correto do cálculo do determinante da matriz A.

5.2.2 Blocos de Normalização / Ajuste e Blocos de Controle

Estes blocos têm função similar as funções de mesmo nome do circuito de cálculo do determinante. Sua função é fazer o ajuste de sinal e tamanho da palavra binária, bem como seu formato, para entrega de entradas adequadas, para cada um dos diversos tipos de divisores a serem testados nesta estrutura, a saber:

- Divisores baseados no algoritmo *non-restoring*, para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2;
- Divisores para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2, baseados no algoritmo Goldschmidt modificado, utilizando multiplicadores da ferramenta;
- Divisores para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2,baseados no algoritmo Goldschmidt modificado, utilizando multiplicadores binários RCA *radix* 4 da literatura;

5.2.3 Divisores

Nesta etapa da estrutura é onde se utilizam os diversos tipos de divisores já citados na subseção anterior para efetuar comparativo de desempenho entre as estruturas de inversão de matriz, baseados em cada um dos algoritmos implementados.

Estes quatro blocos, efetuam as operações responsáveis por produzir $\frac{1}{detA} \times Adj(A)$. Esta etapa é implementada com estrutura onde as operações são efetuadas em paralelo, para acelerar o desempenho. Os divisores efetuam $\frac{Entrada(A)}{detA}$, $\frac{Entrada(B)}{detA}$, $\frac{Entrada(C)}{detA}$ e $\frac{Entrada(D)}{detA}$, respectivamente, onde, os temos (A), (B), (C) e (D) nos numeradores das frações são as entradas do inversor de matrizes, já devidamente ajustadas pelos blocos anteriores. No denominador, o termo $detA$ se refere ao determinante da matriz, calculado no bloco de mesmo nome.

As saídas dos divisores são entregues aos blocos de truncagem e ajuste do sinal das saídas.

5.2.4 Truncagem e Ajuste de Saída

Estes blocos fazem a interconexão das saídas dos divisores com a saída da estrutura de inversão de matrizes, e efetuam, quando necessário, a adequação das saídas para 16 bits em formato Q7.8. Na próxima subseção é apresentando um exemplo de simulação do cálculo de inversão de matrizes, que esclarece a truncagem e os ajustes necessários para a obtenção dos resultados corretos.

5.3 Simulação e Testes da Arquitetura de Inversão de Matrizes Proposta

Foram implementadas diversas versões da arquitetura de inversão de matrizes, conforme relacionado a seguir:

- Inversor de matrizes baseado nos divisores com algoritmo *non-restoring*, para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2;
- Inversor de matrizes baseado nos divisores para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2, implementados com base no algoritmo Goldschmidt modificado, utilizando multiplicadores da ferramenta;
- Inversor de matrizes baseado nos divisores para arquitetura de ponto fixo e aritmética com sinal, em complemento de 2, implementados com base no algoritmo Goldschmidt modificado, utilizando multiplicadores binários RCA *radix* 4 da literatura.

Todas as arquiteturas implementadas apresentaram resultados corretos e idênticos para a inversão de matrizes, testados com inúmeras matrizes. Os resultados de seu funcionamento com uma matriz exemplo, estão apresentados na Figura 5.3. Estes testes foram efetuados utilizando a ferramenta Quartus II da Altera (ALTERA, 2009). Nesse ponto se faz necessário um comentário sobre os resultados obtidos, que estão em arquitetura de ponto-fixo e apresentados em formato Q7.8.

Na Figura 5.3 podem ser verificados os valores de entrada 896; 640; 384 e 1152. Como esta simulação está apresentando valores em formato decimal, logo deve ser feita a conversão de valores, como mostrado abaixo:

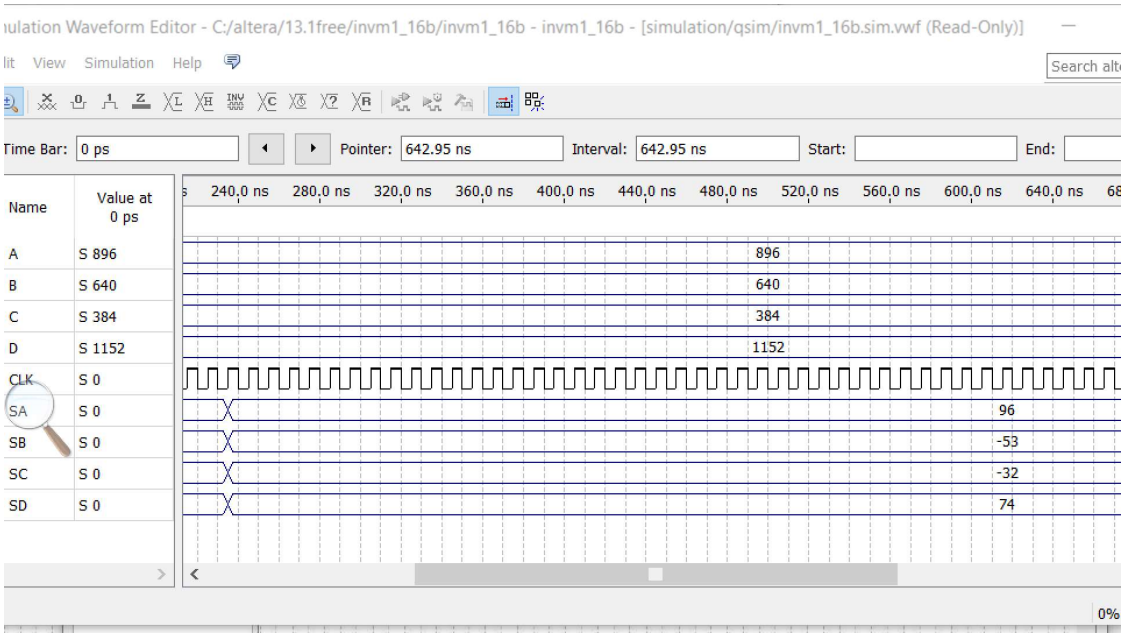

Figura 5.3 – Resultado da arquitetura de inversão de matrizes proposta.
Fonte: O Autor

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 896_{10} & 640_{10} \\ 384_{10} & 1152_{10} \end{bmatrix} = \begin{bmatrix} 0000001110000000_2 & 0000001010000000_2 \\ 0000000110000000_2 & 0000010010000000_2 \end{bmatrix}$$

Os valores devem respeitar a representação do formato Q7.8. Logo, os valores podem ser reescritos nesse formato, colocando-se a vírgula em sua posição, ou seja:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 00000011,10000000_2 & 00000010,10000000_2 \\ 00000001,10000000_2 & 00000100,10000000_2 \end{bmatrix} = \begin{bmatrix} 3,5_{10} & 2,5_{10} \\ 1,5_{10} & 4,5_{10} \end{bmatrix}$$

Fazendo o mesmo processo para os resultados:

$$\begin{bmatrix} SA & SB \\ SC & SD \end{bmatrix} = \begin{bmatrix} 98_{10} & -53_{10} \\ -32_{10} & 74_{10} \end{bmatrix} = \begin{bmatrix} 0000000001100000_2 & 1111111111001011_2 \\ 1111111111100000_2 & 0000000001001010_2 \end{bmatrix}$$

então, a representação do formato Q7.8 pode ser reescrita, colocando-se a vírgula em sua posição, logo:

$$\begin{bmatrix} SA & SB \\ SC & SD \end{bmatrix} = \begin{bmatrix} 00000000,01100000_2 & 11111111,11001011_2 \\ 11111111,11100000_2 & 00000000,01001010_2 \end{bmatrix} = \begin{bmatrix} 0,375_{10} & -0,207_{10} \\ -0,125_{10} & 0,289_{10} \end{bmatrix}$$

A inversão da matriz exemplo foi efetuada utilizando o ferramenta Matlab (MATHWORKS, 2014), e seu resultado está representado abaixo.

$$\begin{bmatrix} 3,5_{10} & 2,5_{10} \\ 1,5_{10} & 4,5_{10} \end{bmatrix}^{-1} = \begin{bmatrix} 0,375_{10} & -0,207_{10} \\ -0,125_{10} & 0,289_{10} \end{bmatrix}$$

Logo, pode ser observado que os resultados são os mesmos, o que comprova o funcionamento da arquitetura implementada.

5.4 Resumo do Capítulo

Este capítulo apresentou a arquitetura geral do circuito para a inversão de matrizes, baseado no método analítico. Os detalhes explicativos dos blocos componentes da arquitetura geral foram apresentados. Um exemplo de cálculo para a inversão de matriz, baseado na arquitetura proposta, foi comparado com os valores produzidos pela ferramenta Matlab, onde foi possível observar a igualdade dos resultados produzidos por ambas as estruturas. O próximo capítulo apresenta os principais resultados de síntese obtidos dos circuitos divisores e de inversão de matriz implementados nesta dissertação.

6 RESULTADOS OBTIDOS

Este capítulo aborda os principais resultados de síntese obtidos com os divisores propostos e com o circuito de inversão de matriz. A metodologia para os resultados obtidos também é apresentada.

6.1 Metodologia de Síntese

O divisor Goldschmidt proposto, em suas diversas versões, e o original (DESCHAMPS; SUTTER; CANTÓ, 2012) foram descritos em VHDL e sintetizados usando uma biblioteca de células Nangate padrão de 45nm (1.1V)(NANGATE, 2016) com a ferramenta *Cadence Encounter RTL Compiler* (ENCOUNTER, 2016). O divisor Goldschmidt original foi implementado tanto com operadores aritméticos da ferramenta como utilizando multiplicadores binários RCA *radix* 4 da literatura. O divisor Goldschmidt proposto também foi implementado com base nos mesmos operandos. A Figura 4.3 apresenta a metodologia utilizada para a estimativa de potência. Em primeiro lugar, uma síntese no Compilador RTL foi realizada para gerar o Verilog *netlist* e o arquivo de atraso .SDF (*Standard Delay Format*). A simulação, que gera os estímulos .VCD, apresenta um *testbench* composto por MATLAB (MATHWORKS, 2014) e Cadence IES, considerando a geração de vetores aleatórios de entrada.

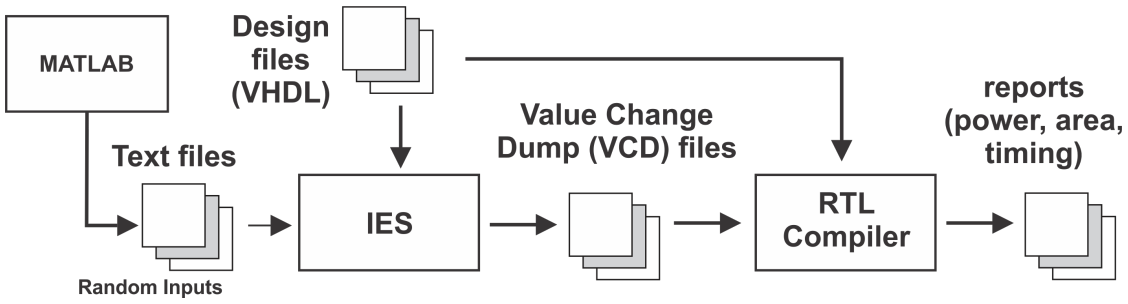

Figura 6.1 – Metodologia de síntese e estimativa de potência.
Fonte: (MARQUES et al.,)

6.2 Resultados de Síntese para os Circuitos Divisores

Os resultados da síntese, em um comparativo entre o algoritmo Goldschmidt original e o algoritmo Goldschmidt proposto, ambos implementados com multiplicadores da ferramenta, são apresentados na Tabela 6.1, onde a contagem de portas é calculada com base em *NANDs* de 2 entradas. O algoritmo Goldschmidt proposto permite o desenvolvimento da arquitetura com maior nível de paralelismo. Portanto, a arquitetura Goldschmidt proposta, ocupa mais área, 16,4 mil portas contra 5,87 mil do circuito Goldschmidt original. Entretanto, considerando as tecnologias de fabricação de circuitos integrados atuais, o principal fator limitante não é a área (embora área tenha influência direta no custo de fabricação), mas sim a dissipação de potência, consumo de energia e o processamento em tempo real.

Tabela 6.1 – Resultados da Síntese: Área, Frequência e *Throughput*.

Versão da Arquitetura	Resultados de Área		Resultados de Tempo	
	Área de Celulas (nm)	Contagem De Portas (kPortas)	Max. Freq. (MHz)	Max. *Throughput* @ Freq.Max. (Mops/s)
Goldschmidt Original	4695	5.86	124.51	31.13
Goldschmidt Melhorado Proposto	13129	16.4	58.36	58.36 **(+87.44%)**

Fonte: O Autor

A Tabela 6.1 também mostra que, na frequência alvo máxima, a arquitetura Goldschmidt proposta é capaz de atingir um rendimento de 87,4% maior, atingindo 58,36 Mop/s (operação por segundo), quando comparada com as 31,13 Mop/s do divisor Goldschmidt original.

A análise de dissipação de potência foi realizada com 100000 vetores de entrada aleatórios gerados e convertidos para ponto fixo usando a ferramenta MATLAB. A frequência alvo para estimativa de potência foi calculada para executar um *throughput* fixo de 10M operações por segundo. A frequência para executar esse *throughput* é 40Mhz para a arquitetura Goldschmidt original, que executa uma operação de divisão em quatro ciclos de relógio. Por outro lado, o divisor Goldschmidt proposto realiza uma operação por ciclo de relógio, e, consequentemente, alcança 10Mops/s a 10MHz.

Observando os resultados de dissipação de potência na Tabela 6.2, nota-se que a arquitetura do divisor Goldschmidt original, quando implementada com multiplicadores Binários RCA *radix* 4 da literatura, obtém economia na dissipação de potência de cerca de 19,9% e

uma redução de energia/operação de 25,9% ao operar em *isoperformance*, a 10Mops/s, quando comparada com o divisor Goldschmidt utilizando o operador de multiplicação da ferramenta de síntese. Este resultado reforça o impacto positivo do uso do multiplicador na base 4 da literatura, que também permite que o divisor Goldschmidt proposto encontre maiores reduções de valores de dissipação de potência e de energia/operação, quando comparado com o divisor Goldschmidt proposto com o uso do multiplicador da ferramenta de síntese. Desta forma, a combinação das técnicas propostas, com o uso de um multiplicador eficiente da literatura, proporcionam os melhores resultados de dissipação de potência e energia/operação ao divisor Goldschmidt proposto, quando comparado às demais arquiteturas de divisores da Tabela 6.2, onde foi possível obter uma redução de dissipação de potência de cerca de 37,9% e uma redução de energia/operação de 66,0% ao operar em *isoperformance*, comparado com o divisor Goldschmidt da literatura. Pode ser observado na Tabela 6.2, que o termo *isoperformance* está se referindo ao fato de que todos os divisores estão operando para produzir dez milhões de divisões por segundo.

Tabela 6.2 – Resultados de dissipação de potência e consumo de energia dos divisores implementados

Versão da Arquitetura @10Mop/s	Dissipação de Potência			Consumo de Energia	Redução de Potência Total e de Energia por Operação
	Leak. (mW)	Dinâmica (mW)	Total (mW)	Energia por Operação (pJ)	
Divisor Goldschmidt Original Ferramenta	0,24	2,30	2,54	229,9	-
Divisor Goldschmidt Original Bin. Radix 4	0,33	1,70	2,03	170,4	19,9% 25,9%
Divisor Goldschmidt Melhorado Proposto Ferramenta	0,65	1,06	1,70	105,5	32,9% 54,1%
Divisor Goldschmidt Melhorado Proposto Bin. Radix 4	0,79	0,78	1,57	78,1	37,9% 66,0%

Fonte: O Autor

6.3 Resultados de Síntese para os Inversores de Matrizes

Para aumentar o comparativo para as arquiteturas de inversão de matrizes propostas neste trabalho de dissertação, foi implementado um inversor de matrizes baseado em um algoritmo de divisão *non-restoring* da literatura (DESCHAMPS; SUTTER; CANTÓ, 2012). Para obter uma comparação adequada, foi necessário alterar o divisor da literatura para aceitar valores expandidos de entrada e valores em complemento de 2. Os resultados de síntese para os inversores de matrizes implementados neste trabalho de dissertação são apresentados na Tabela 6.3. Conforme pode ser visto, o inversor de matrizes utilizando divisores Goldschmidt propostos, com multiplicadores da ferramenta, obtiveram uma redução de dissipação de potência de cerca de 61,8% e uma redução de energia/operação de 53,6% ao operar em *isoperformance*, comparado com o inversor de matrizes baseado em divisores *non-restoring* da literatura. Por sua vez, o inversor de matrizes utilizando divisores Goldschmidt propostos, com multiplicadores Binários RCA *radix* 4, obtiveram uma redução de dissipação de potência de cerca de 72,1% e uma redução de energia/operação de 66,7% ao operar em *isoperformance*, comparado com o inversor de matrizes baseado em divisores *non-restoring* da literatura.

Tabela 6.3 – Resultados de dissipação de potência e consumo de energia das arquiteturas de inversão de matrizes.

Versão da Arquitetura @10Mop/s	Dissipação de Potência			Consumo de Energia	Redução de Potência Total e de Energia por Operação
	Leak. (mW)	Dinâmica (mW)	Total (mW)	Energia por Operação (pJ)	
Inversor de Matrizes Non-restoring	2,91	248,83	309,98	24,88	-
Inversor de Matrizes Goldschmidt proposto com multiplicadores da Ferramenta	2,99	115,36	118,35	11,54	61,8% 53,6%
Inversor de Matrizes Goldschmidt proposto com multiplicadores Radix 4	3,57	82,90	86,48	8,29	72,1% 66,7%

Fonte: O Autor

6.4 Resumo do capítulo

Neste capítulo foram apresentados os principais resultados obtidos neste trabalho em termos de área, frequência máxima, *throughput*, energia/operação e dissipação de potência, assim como foram apresentadas as devidas comparações entre os diferentes circuitos divisores e circuito de inversão de matriz implementados.

7 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação de mestrado apresentou a implementação de um circuito para inversão de uma matriz 2x2, baseado no método analítico. Este tamanho de matriz é adequado para uma futura implementação de um circuito completo de filtragem adaptativa, baseado no algoritmo de Projeções Afins (AP). Foi verificado, por uma análise comparativa de diferentes algoritmos de inversão de matrizes, que para este tamanho de matriz, 2x2, o método analítico é o mais adequado, pois agrega uma reduzida complexidade computacional, em relação aos outros algoritmos.

No circuito de inversão de matriz, o operador aritmético de divisão é o mais complexo de ser implementado em *hardware*. Verificou-se que, entre os diversos algoritmos de divisão propostos na literatura, o algoritmo de Goldschmidt é o mais utilizado, pois realiza de forma rápida e eficiente o cálculo de divisão. Desta forma, uma das principais contribuições dessa dissertação de mestrado esteve voltada à proposta de um novo algoritmo para o divisor Goldschmidt, que permite a convergência para os resultados desejados em apenas 3 iterações, contra 5 iterações no algoritmo Goldschmidt original. O algoritmo proposto, além de tornar o divisor Goldschmidt mais rápido, também proporciona a operação de divisão em uma faixa mais ampla de valores, baseado no formato Q7.8, onde é possível a realização de operações de divisão na faixa entre -127.99609375 e +127.99609375, contra a faixa reduzida de valores entre 1 e 2 proporcionada pelo circuito Goldschmidt original.

Como o circuito multiplicador é o elemento base no divisor Goldschmidt, logo, menos iterações implicam em menos multiplicações, que implicam em menor dissipação de potência, como mostraram os resultados obtidos. Os resultados também mostraram a eficiência do uso do circuito multiplicador na base 4 da literatura, onde ambos os divisores Goldschmidt original e proposto obtiveram menores resultados de dissipação de potência e energia/operação, quando comparados com o uso do multiplicador da ferramenta de síntese em ambos os circuitos divisores. Em relação aos circuitos divisores a principal conclusão é que a combinação do algoritmo proposto para o aumento do desempenho, com o uso do multiplicador na base 4 da literatura proporcionaram os melhores resultados de dissipação de potência e energia por operação ao circuito divisor Goldschmidt.

Os circuitos divisores implementados nesta dissertação foram inseridos no circuito de inversão de matriz, onde os resultados mostraram a eficiência do uso dos divisores propostos, com a redução da dissipação de potência na inversão da matriz 2x2 implementada. Esses resultados motivam, como próximo passo, à implementação de um circuito completo de filtragem

adaptativa, baseado no algoritmo de Projeções Afins, usando o circuito de inversão de matriz e os circuitos divisores eficientes propostos nessa dissertação de mestrado.

7.1 Trabalhos Futuros

Os estudos realizados com as implementações de arquiteturas dedicadas dos circuitos divisores, oportunizaram a submissão de um artigo científico para o evento internacional LASCAS2017 (IEEE *Latin American Symposium on Circuits and Systems*), com o título: *Improved Goldschmidt Algorithm for Fast and Energy-Efficient Fixed-Point Divider*.

Diante dos resultados obtidos ao longo desta dissertação, com resultados expressivos obtidos nas explorações dos circuitos divisores e circuito para inversão de matriz, abre-se uma perspectiva de novos trabalhos nesta linha. Desta forma, os novos desafios devem envolver:

- Avaliação, análise de desempenho e de redução de potência do divisor Goldschmidt proposto, com a inclusão de uma aproximação nos subtratores, onde deve-se simplificar a subtração por complemento de 2. Será testada a robustez do algoritmo utilizando apenas complemento, o que reduz bastante a implementação em *hardware*. Isto reduz a propagação de *carry* ao longo do circuito, bem como reduz atividades de transição, introduzindo apenas um erro de uma ULP. Em formato Q7.8 isto significa a introdução de um erro de 0,00390625. Este estudo deve comprovar que o algoritmo é robusto a ponto de compensar o erro introduzido;

- Implementação do circuito divisor Goldschmidt de baixa dissipação de potência, baseado na codificação de operandos. Para tal, deverá ser utilizada uma codificação diferente da codificação binária, chamada de codificação Híbrida. A ideia da codificação Híbrida é dividir os operandos em grupos de m bits, codificar cada grupo utilizando a codificação Gray e utilizar o comportamento do código binário para propagar o *carry* entre os grupos. Desta forma, o número de transições em cada grupo pode ser minimizado e uma estrutura regular pode ser obtida, onde os grupos menos significativos do resultado dependem somente dos grupos menos significativos dos operadores. Logo, o código Híbrido representa um compromisso entre a mínima dependência das entradas de dados apresentada pelo código Binário e a característica de baixa atividade de chaveamento apresentada pelo código Gray (COSTA, 2002). Desta forma, para sistemas onde a capacitância chaveada no barramento de dados é significativa, como nos filtros adaptativos, e onde os dados apresentam um alto grau de correlação, a utilização do código Híbrido pode reduzir o consumo de potência em até cerca de um terço em relação ao código Binário (COSTA, 2002),

- Implementação do circuito completo de filtragem adaptativa, baseado no algoritmo de

Projeções Afins. Para tal, deverão ser utilizados os circuitos divisores e circuito de inversão de matriz, propostos nessa dissertação de mestrado.

REFERÊNCIAS

ALMEIDA, S. J. M. et al. A statistical analysis of the affine projection algorithm for unity step size and autoregressive inputs. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 52, Jul 2005.

ALMEIDA, S. J. M. d. **Análise Estatística do Comportamento de uma Classe de Algoritmos de Projeções Afins**. Dissertação (Tese de Doutorado em Engenharia Elétrica) — Universidade Federal de Santa Catarina, Florianópolis-SC, 2004.

ALTERA. **Quartus II Version 9.0 Handbook**. 101 Innovation Drive, San Jose, CA 95134, USA: Altera Corporation, 2009.

ANGHEL, C. et al. Fpga implementation of a variable step-size affine projection algorithm for acoustic echo cancellation. In: IEEE. **Signal Processing Conference, 2010 18th European**. [S.l.], 2010. p. 532–536.

ANTON, H.; RORRES, C. **Álgebra Linear com Aplicações**. [S.l.]: 10ªed, Bookman, 2016.

BEHROOZ, P. Computer arithmetic: Algorithms and hardware designs. **Oxford University Press**, v. 19, p. 512583–512585, 2000.

CORPORTATION, I. Ia-32 intel architecture software developer's manual. **Intel Corportation**, 2001.

COSTA, E. A. C. d. **Operadores aritméticos de baixo consumo para arquiteturas de circuitos DSP**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2002.

DESCHAMPS, J.-P.; SUTTER, G. D.; CANTÓ, E. **Guide to FPGA implementation of arithmetic functions**. [S.l.]: Springer Science & Business Media, 2012. v. 149.

ENCOUNTER, R. C. Compiler v. 8.10. **Available at: www.cadense.com Accessed November**, v. 11, 2016.

ERCEGOVAC, M. D. et al. Improving goldschmidt division, square root, and square root reciprocal. **IEEE Transactions on Computers**, IEEE, v. 49, n. 7, p. 759–763, 2000.

FLYNN, M. J. On division by functional iteration. **IEEE Transactions on Computers**, IEEE, v. 100, n. 8, p. 702–706, 1970.

GARCÍA, J. A. Implementação em fpga de uma biblioteca parametrizável para inversão de matrizes baseada no algoritmo gauss-jordan, usando representação em ponto flutuante. 2011.

GOLDSCHMIDT, R. E. **Applications of division by convergence**. Tese (Doutorado) — Massachusetts Institute of Technology, 1964.

HA, J.; LIM, S. The implementation of the srt division algorithm. Citeseer, 2007.

HOSSEINY, A.; JABERIPUR, G. Decimal goldschmidt: A hardware algorithm for radix-10 division. **Computers & Electrical Engineering**, Elsevier, v. 53, p. 40–55, 2016.

IRTÜRK, A. U. **GUSTO: General architecture design utility and synthesis tool for optimization**. Tese (Doutorado) — University of California, San Diego, 2009.

ITO, M.; TAKAGI, N.; YAJIMA, S. Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification. **IEEE Transactions on Computers**, IEEE, v. 46, n. 4, p. 495–498, 1997.

KONG, I.; SWARTZLANDER, E. E. A rounding method with improved error tolerance for division by convergence. In: IEEE. **2008 42nd Asilomar Conference on Signals, Systems and Computers**. [S.l.], 2008. p. 1814–1818.

KUCUKKABAK, U.; AKKAS, A. Design and implementation of reciprocal unit using table look-up and newton-raphson iteration. In: IEEE. **Digital System Design, 2004. DSD 2004. Euromicro Symposium on**. [S.l.], 2004. p. 249–253.

LI, Y.; CHU, W. A new non-restoring square root algorithm and its vlsi implementations. In: IEEE. **Computer Design: VLSI in Computers and Processors, 1996. ICCD'96. Proceedings., 1996 IEEE International Conference on**. [S.l.], 1996. p. 538–544.

MALAJOVICH, G. Álgebra linear. **Disponível em http://s3.amazonaws.com/academia.edu.documents/38134084/algebra_linear.pdf**, v. 16, 2007.

MARKSTEIN, P. Software division and square root using goldschmidt's algorithms. In: **Proceedings of the 6th Conference on Real Numbers and Computers (RNC'6)**. [S.l.: s.n.], 2004. v. 123, p. 146–157.

MARQUES, P. et al. Improved goldschmidt algorithm for fast and energy-efficient fixed-point divider.

MATHWORKS. **Matlab R2014a Handbook**. 1 Apple Hill Drive, Natick, MA 01760-2098, USA: MathWorks Inc., 2014.

MATULA, D. W. Improved table lookup algorithms for postscaled division. In: IEEE. **Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on**. [S.l.], 2001. p. 101–108.

NANGATE, S. California (2008). 45nm open cell library. **Available at: www. nangate. com Accessed November**, 2016.

OBERMAN, S. F.; FLYNN, M. J. An analysis of division algorithms and implementations. **Rapport Technique CSL-TR-95-675, Computer Systems Laboratory, Dept. on Electrical Engineering and Computer Science Stanford University**, 1995.

OBERMAN, S. F.; FLYNN, M. J. Design issues in division and other floating-point operations. **IEEE Transactions on Computers**, IEEE, v. 46, n. 2, p. 154–161, 1997.

OZEKI, K.; UMEDA, T. An adaptive filtering algorithm using orthogonal projection to an affine subspace and its properties. **Eletronics and Communications in Japan**, v. 67, Jan 1984.

ROY, T. D. Implementation of goldschmidts algorithm with hardware reduction. **Illinois institute of technology, Chicago USA**.

RUPP, M. A family of adaptive filter algorithms with decorrelating properties. **IEEE Transactions on Signal Processing**, v. 46(3), Mar 1998.

SÁNCHEZ, D. F. et al. Parameterizable floating-point library for arithmetic operations in fpgas. In: ACM. **Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes**. [S.l.], 2009. p. 40.

SANKARAN, S. G.; BEEX, A. L. Convergence behavior of affine projection algorithms. **IEEE Transactions on Signal Processing**, v. 48, Apr 2000.

THERRIEN, C. W. **Discrete Random Signal and Statistical Signal Processing**. [S.l.]: Prentie-Hall - New Jersey, 1992.

WANG, Z. et al. Image quality assessment: from error visibility to structural similarity. **IEEE transactions on image processing**, IEEE, v. 13, n. 4, p. 600–612, 2004.

WIDROW BERNARD.; HOFF, M. E. **Adaptive switching circuits**. Cambridge, MA, USA: MIT Press, 1988.