

RECONSTRUÇÃO DE IMAGENS TOMOGRÁFICAS BASEADA EM
ARQUITETURAS RECONFIGURÁVEIS

Luiz Maltar Castello Branco

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS
PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE
FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS
PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA
DE SISTEMAS E COMPUTAÇÃO.

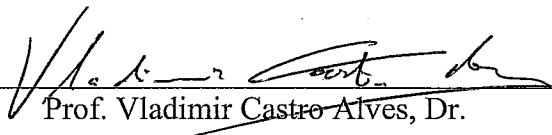
Aprovada por:



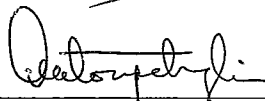
Prof. Cláudio Luís de Amorim, Ph.D.



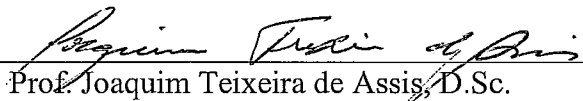
Prof. Felipe Maia Galvão França, Ph.D.



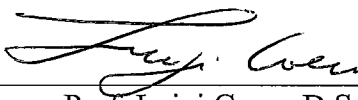
Prof. Vladimir Castro Alves, Dr.



Prof. Antonio Petraglia, Ph.D.



Prof. Joaquim Teixeira de Assis, D.Sc.



Prof. Luigi Carro, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2000

BRANCO, LUIZ MALTAR CASTELLO

Reconstrução de Imagens

Tomográficas Baseada em Arquiteturas

Reconfiguráveis

[Rio de Janeiro] 2000

X, 146p. 29,7 cm (COPPE/UFRJ,

D.Sc., Engenharia de Sistemas e

Computação, 2000)

Tese – Universidade Federal do Rio

de Janeiro, COPPE

1. Reconstruções de Imagens

2. Arquiteturas Reconfiguráveis

3. Aritmética de Resíduos

I. COPPE/UFRJ II. Título (série)

Dedico este trabalho a minha esposa Sonia
e a minha filha Bruna.

Agradecimentos

Agradeço à minha família pela paciência e apoio dado a esta longa jornada e peço desculpas pela minha ausência.

Agradeço aos professores do Programa de Engenharia Nuclear, em particular, professores Ricardo Lopes e Edgar Jesus, pelo apoio e confiança no meu trabalho. E também ao professor Nilson Roberty pela orientação inicial do meu trabalho.

Agradeço aos professores do Programa de Engenharia de Sistemas pela agradável jornada de aprendizado e qualidade dispensada durante o curso.

Agradeço aos membros da banca, Joaquim Teixeira, Luigi Carro, Antonio Petraglia, cujas observações e correções permitiram melhoria do trabalho, bem como outras discussões e esclarecimentos fornecidos durante a realização do trabalho.

Agradeço aos professores Sérgio Diniz e Manuel Lois pelo apoio dado, em especial, a imensa bibliografia disponibilizada.

Agradeço ao amigo Luiz Monnerat pela cooperação e força dada nos trabalhos durante o período de realização das cadeiras e qualificação.

Agradeço ao pessoal das bibliotecas do CT e do NCE , pela cordialidade. Agradeço aos amigos da COPPE pela preocupação e atenção dispensada neste período, em especial ao pessoal da secretaria do PESC, Registro e do GRH.

Agradeço ao pessoal do NCE, em particular ao Meslin, Sidney e Professor Salek, que forneceram as ferramentas utilizadas neste trabalho, além da exatidão nas respostas solicitadas.

Agradeço ao pessoal do LIN, funcionários e alunos, pela cobertura e compreensão ao longo do período em que me ausentei. Em particular ao pessoal da eletrônica, Zé Roberto, Rizzo, Carlos Augusto, Sandro e Osmar.

Agradeço ao Amigo Luís Fernando pelas longas horas de idéias trocadas e sua sempre disponibilidade.

Agradeço aos orientadores Claudio Amorim, Felipe França e ao quase orientador Vladimir Castro pelas orientações seguras não somente da tese, como também em outras atividades exercidas durante este período, as quais engrandeceram em muito a minha formação como Doutor.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

RECONSTRUÇÃO DE IMAGENS TOMOGRÁFICAS BASEADAS EM ARQUITETURAS RECONFIGURÁVEIS

Luiz Maltar Castello Branco

Junho/2000

Orientadores: Cláudio Luís de Amorim

Felipe Maia Galvão França

Programa: Engenharia de Sistemas e Computação

A reconstrução de imagens tomográficas é uma tarefa computacional de grande escala que atinge alto desempenho quando executada em arquiteturas paralelas e/ou dedicadas. Entretanto, a aritmética convencional adotada nessas arquiteturas limita a utilização eficiente da moderna tecnologia de arquiteturas reconfiguráveis baseadas em FPGAs. Esta tese propõe e avalia a utilização da aritmética de resíduos em arquiteturas reconfiguráveis de alto desempenho baseadas em FPGAs para a reconstrução de imagens tomográficas.

Utilizando simuladores de projeções para feixes paralelos e divergentes, bem como programas de reconstrução de imagens, este trabalho é o primeiro a demonstrar que o paralelismo das operações aritméticas em resíduos pode ser explorado eficazmente de modo a permitir implementações de alto desempenho em RAM FPGAs de funções básicas tais como somadores, multiplicadores, conversores binário-resíduo-binário, filtragem, retroprojeção e recíproco.

Além disso, os resultados experimentais mostram que a reconfigurabilidade de componentes RAM FPGAs pode ser explorada eficientemente no projeto de uma arquitetura de alto desempenho para reconstrução de imagem e também ser utilizada em outras etapas do processamento de imagens, utilizando-se estruturas dedicadas e carregando-se configurações apropriadas. Como consequência, o desempenho da arquitetura proposta supera a arquitetura binária convencional em cerca de 1-2 ordens de grandeza.

Esses resultados permitem concluir que arquiteturas reconfiguráveis baseadas em aritmética de resíduos são alternativas eficazes para reconstrução de imagens tomográficas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

RECONSTRUCTION OF TOMOGRAPHIC IMAGES BASED ON RECONFIGURABLE ARCHITECTURES

Luiz Maltar Castello Branco

June/2000

Advisors: Cláudio Luís de Amorim

Felipe Maia Galvão França

Department: Computing and Systems Engineering

Reconstruction of tomographic images is a large-scale computational task, which reaches high performance when executing in parallel and/or dedicated architectures. However, the conventional arithmetic adopted on these architectures limits the efficient use of modern technology of reconfigurable architecture based on FPGAs (Field-Programmable Gate Arrays). This thesis proposes and evaluates the use of the Residue Number Systems in high-performance reconfigurable architectures based on FPGAs for reconstruction of tomographic images.

Using simulators of parallel and fan-beam projections and programs for image reconstruction, this work is the first to demonstrate that the parallelism of residue-based arithmetic can effectively be exploited so as to allow for high-performance implementations into RAM FPGAs of basic arithmetic functions such as adders, multipliers, binary-to-residue and residue-to-binary converters, filtering, back-projection, and reciprocal.

In addition, the experimental results show that the reconfigurability of RAM FPGA components can be efficiently exploited in high-performance architectures for image reconstruction and still be applied to other image processing steps, using dedicated structures and loading appropriate configurations. As a consequence, the performance of the proposed architecture outperforms conventional binary-based architectures by approximately 1-2 orders of magnitude.

These results allow us to conclude that reconfigurable architectures based on residue number systems are effective alternatives for reconstruction of tomographic images.

Nomenclatura

$\lceil x \rceil$ - Primeiro número inteiro maior que x , para x um número fracionário

$\lfloor x \rfloor$ - Primeiro número inteiro menor que x , para x um número fracionário

\equiv - Congruência

$|x|_m$ - Primeiro resíduo positivo de x em relação a base modular m

\otimes - Convolução

$+_m$ - Soma modular em base m

\times_m - Multiplicação modular em base m

$\max |x|$ - Valor máximo de um número em módulo

\mathbb{N} - Conjunto dos números Naturais.

FOURIER - Transformada de Fourier

→

Índice

	Página
CAPÍTULO 1	1
INTRODUÇÃO	1
1.1 – RECONSTRUÇÃO DE IMAGENS	1
1.2 – MOTIVAÇÕES	7
1.3 – ALTO DESEMPENHO COMPUTACIONAL EM RECONSTRUÇÃO DE IMAGENS	8
1.4 – SISTEMAS RECONFIGURÁVEIS	10
1.5 – CONTRIBUIÇÕES DA TESE	12
1.6 – ORGANIZAÇÃO DA TESE	14
 CAPÍTULO 2	 16
AVALIANDO OPERAÇÕES ARITMÉTICAS EM FPGAS	16
2.1 – INTRODUÇÃO AOS DISPOSITIVOS LÓGICOS PROGRAMÁVEIS	16
2.2 – INTRODUÇÃO AOS RAM FPGAS	18
2.2.1 – Metodologia de projeto de circuitos usando FPGAs	21
2.3 – MAPEAMENTO DE FUNÇÕES GENÉRICAS EM SRAM FPGAS	22
2.4 – OPERAÇÕES ARITMÉTICAS EM PONTO FLUTUANTE EM FPGAS	24
2.4.1 – Representação em ponto flutuante	25
2.4.2 – Operações em ponto flutuante	27
2.4.3 – Mapeamento de Operações em ponto flutuante em FPGAS	28
 CAPÍTULO 3	 30
ARITMÉTICA DE RESÍDUOS EM FPGAS	30
3.1 – INTRODUÇÃO A SISTEMAS NUMÉRICOS BASEADOS EM RESÍDUOS	30
3.2 – PROPRIEDADES DE SISTEMAS BASEADOS EM RESÍDUOS	31
3.3 – IMPLEMENTAÇÃO DE OPERAÇÕES ARITMÉTICAS RNS EM FPGAS	32
3.3.1 – Mapeamento direto das operações	33
3.3.2 – Implementação de um somador RNS	34
3.3.3 – Implementação de um multiplicador RNS	37

3.4 – CONVERSÃO ENTRE SISTEMAS NUMÉRICOS BINÁRIO E RNS	43
3.4.1 – Conversão Binário Resíduo	43
3.4.2 – Inversão Resíduo Binário	51
3.4.3 – Discussão	55
CAPÍTULO 4	56
APLICANDO ARITMÉTICA DE RESÍDUOS AO ALGORITMO DE	
RETROPROJEÇÃO FILTRADA	56
4.1 – ALGORITMO PARA FEIXES PARALELOS	56
4.1.1 – Filtragem	57
4.1.2 – Retroprojeção	59
4.1.3 – Determinação da Faixa dinâmica	61
4.1.4 – Análise dos resultados	66
4.2 – ALGORITMO PARA FEIXES DIVERGENTES	69
4.2.1 – Filtragem	69
4.2.2 – Retroprojeção	69
4.2.3 – Determinação da Faixa dinâmica	71
4.2.4 – Análise dos resultados	71
CAPÍTULO 5	74
UMA ARQUITETURA RECONFIGURÁVEL PARA RECONSTRUÇÃO DE	
IMAGENS TOMOGRÁFICAS	74
5.1 – PROPOSTA	74
5.2 – UNIDADE DE FILTRAGEM	76
5.2.1 – Implementação	81
5.2.2 – Discussão	83
5.3 – UNIDADE DE RETROPROJEÇÃO PARA FEIXES PARALELOS COM	
INTERPOLAÇÃO LINEAR	85
5.3.1 – Discussão	86

5.4 – UNIDADE DE RETROPROJEÇÃO PARA FEIXES DIVERGENTES COM INTERPOLAÇÃO LINEAR	88
5.4.1 – Operações não-lineares na retroprojeção	90
5.4.2 – Implementação de s'	92
5.4.2.1 – Discussão	96
5.4.3 – Computação de $1/U^2$	97
5.4.3.1 – Aplicando-se o método <i>Symmetric Bipartite Tables</i> em $1/U^2$	98
5.4.3.2 – Discussão	100
5.5 – ANÁLISE DOS RESULTADOS	101
CAPÍTULO 6	104
TRABALHOS CORRELATOS	104
CAPÍTULO 7	112
CONCLUSÕES E TRABALHOS FUTUROS	112
REFERÊNCIAS BIBLIOGRÁFICAS	115
APÊNDICE A	131
DIVISÃO SRT	131
APÊNDICE B	137
INTRODUÇÃO AO MÉTODO <i>SYMMETRIC BIPARTITE TABLES</i>	137
APÊNDICE C	141
TRANSFORMADA DE RADON	141
TEOREMA DA FATIA	142

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo é introduzido o problema de reconstrução de imagens, através de uma breve descrição da geometria de aquisição de dados e dos algoritmos de reconstrução. É também discutido a escolha do algoritmo a ser utilizado levando-se em consideração o objetivo desta tese e algumas características da complexidade para sua implementação. Discute-se a tecnologia utilizada neste trabalho, exposta dentro de um contexto atual e bastante promissor que é a computação reconfigurável.

1.1 RECONSTRUÇÃO DE IMAGENS

O problema da Reconstrução de imagens a partir de projeções vem sendo estudado e aplicado em importantes áreas da ciência tais como: Tomografia e suas variações, Microscopia Eletrônica, Rádio Astronomia, Radar e Sismologia [1-5]. Os princípios matemáticos para reconstrução de imagens foram, inicialmente, estabelecidos por J. Radon por volta de 1917. Sua primeira aplicação foi em Rádio Astronomia [6], contudo a reconstrução de imagens, como uma área de interesse, tomou um enorme impulso devido às aplicações na área médica. O problema de reconstrução de imagens, para vários problemas práticos, teve seu crescimento acentuado com o advento do computador, devido ao grande número de operações matemáticas que as soluções exigem.

Este trabalho está centrado no modelo de reconstrução de imagens obtido da *Tomografia por Transmissão*. Tomografia é uma técnica não invasiva que permite a visualização de uma seção transversal de um objeto a partir de suas projeções transaxiais. As Figuras 1 e 2 apresentam os principais elementos para a coleta de dados do modelo de Tomografia por Transmissão. Nesse modelo, a fonte de irradiação, que em geral é de raios X, é externa ao corpo, e o sistema fonte de irradiação/detector gira em torno do objeto em ângulos bem definidos. O conjunto de dados coletados a cada nova posição angular é chamado de projeção. Empregam-se algumas denominações aos sistemas de coleta de dados, em função do conjunto fonte-detector, levando-se em consideração a geometria dos feixes da fonte de irradiação e o formato do detector. O modelo de feixes paralelos, Fig.1,

apresenta um conjunto de raios paralelos igualmente espaçados. Já o modelo da Fig.2 é conhecido como o de feixes divergentes com detetor plano, onde os elementos detetores se encontram igualmente espaçados num plano e a fonte de irradiação é puntiforme. Existe um outro modelo bastante conhecido, que não será tratado neste trabalho, onde a fonte é de feixes divergentes e o detetor é curvo.

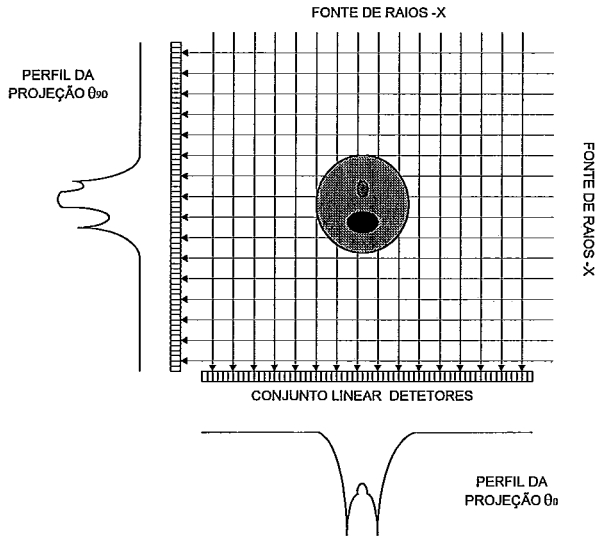


Fig.1: Projeções com feixes paralelos.

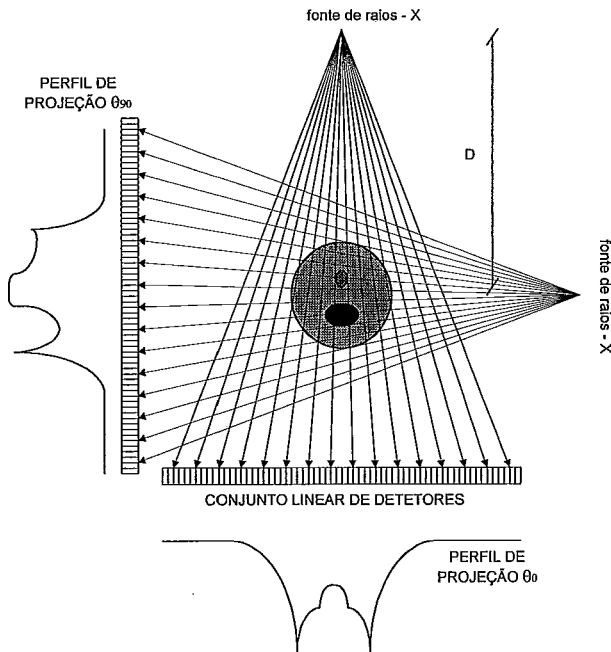


Fig.2: Projeções com feixes divergentes (leque) com detetor plano.

Uma projeção obtida a partir de uma fonte de raios paralelos, Fig. 3, é equivalente a um conjunto de integrais de linha, ou raios soma, igualmente espaçadas ao longo de um eixo S . O eixo S está inclinado de um ângulo θ em relação ao eixo coordenado X do plano $X-Y$, onde se encontra o corpo. Esta definição é equivalente a *transformada direta de Radon*, $g(s, \theta)$, para uma função $f(x,y)$ no plano $X-Y$ [1-3]. Em Tomografia por raios X ou γ , um raio soma é visto como um somatório dos coeficientes de atenuação do raio X ou γ no corpo interceptado, ao longo de sua trajetória da fonte ao detector. Por analogia, pode-se interpretar uma projeção obtida por raios divergentes como também integrais de linha, que partem de uma fonte puntiforme .

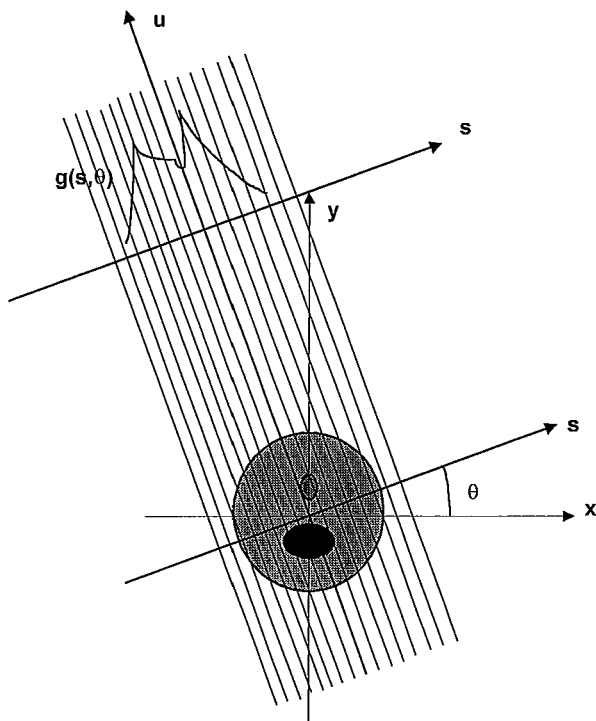


Fig. 3: Transformada de Radon , $g(s, \theta)$, para um ângulo θ .

Muitos pesquisadores elaboraram novos algoritmos ou aperfeiçoaram os já existentes, no sentido de melhorar cada vez mais a qualidade da imagem reconstruída [2,4]. Pode-se classificar os algoritmos utilizados na reconstrução de imagens em duas principais categorias: (i) Algoritmos Algébricos (iterativos): e (ii) Algoritmos Analíticos.

Os algoritmos algébricos trabalham com um modelo onde a imagem $f(x,y)$ a ser reconstruída é formada por um conjunto de células que possuem valores constantes f_j , formando uma grade regular superposta à imagem original, como ilustrado na Figura 4. Os raios soma possuem largura “ τ ”, de valor próximo a largura das células. A contribuição de cada célula para o raio soma (integral de linha) é proporcional à área da célula por onde ele passa e é chamado de peso w_{ij} , onde i é o número do raio soma e j o número da célula.

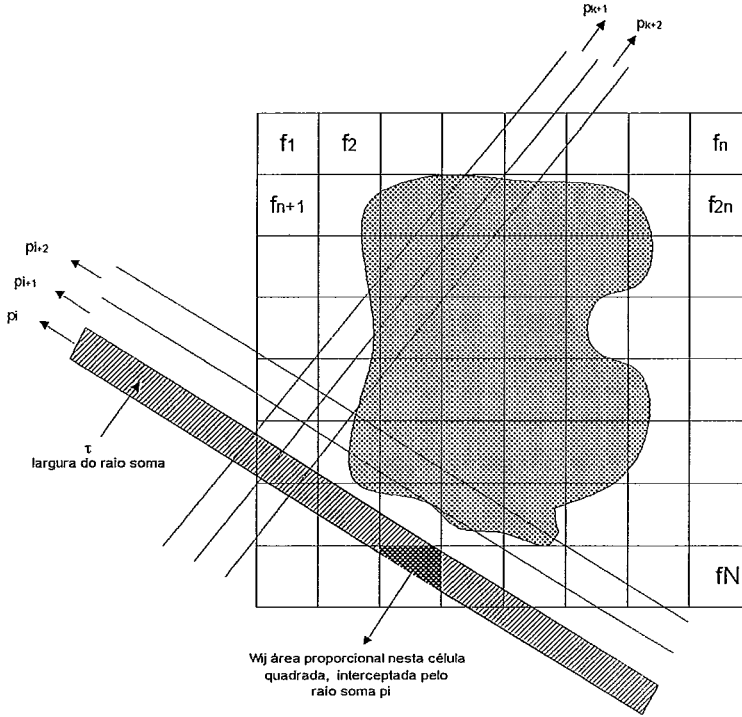


Fig. 4: Modelo de reconstrução da imagem usando-se algoritmos algébricos.

Para o conjunto total de raios soma p_i , tem-se o seguinte equacionamento:

$$\sum_{j=1}^N w_{ij} f_j = p_i \quad , \quad i=1,2, \dots, M \quad (1)$$

Onde M é o número total de raios soma para cada projeção e N é o número total de células. A reconstrução, isto é, a obtenção dos valores de f_j é feita resolvendo o seguinte sistema de equações:

$$\begin{aligned}
w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + \dots + w_{1N}f_N &= p_1 \\
w_{21}f_1 + w_{22}f_2 + w_{23}f_3 + \dots + w_{2N}f_N &= p_2 \quad (2) \\
w_{M1}f_1 + w_{M2}f_2 + w_{M3}f_3 + \dots + w_{MN}f_N &= p_M
\end{aligned}$$

A solução desse sistema através de métodos convencionais de inversões de matrizes é inapropriada, devido principalmente aos seguintes fatores [2]:

- i) Imagens de grande tamanho e um elevado número de projeções podem levar a um sistema de equações muito elevado;
- ii) Supondo $M < N$, mesmo para N pequeno, o ruído na obtenção da projeções torna impraticável a solução por inversão de matrizes.

Por causa dessas restrições as soluções para este sistema são baseadas em métodos iterativos. Tais métodos necessitam de uma atribuição inicial para imagem, ao começar a sua execução, e funcionam com sucessivas correções da imagem, baseando-se nas diferenças entre a imagem atual e a imagem anterior. As correções são feitas através da aplicação sucessiva dos novos valores de f_j s ao sistema de equações. O processo só pára ao satisfazer um critério de convergência definido.

Em geral, os algoritmos algébricos são muito mais lentos do que os analíticos. Os métodos algébricos podem reconstruir imagens de qualidade melhor do que as produzidas com métodos analíticos somente sob as seguintes condições: (i) um número pequeno de projeções; (ii) as projeções não estão uniformemente espaçadas; (iii) ou na falta de alguma projeção, devido a restrições físicas do sistema de aquisição de dados. Estes algoritmos permitem a incorporação de informações conhecidas a priori da imagem, devido ao estabelecimento de uma imagem inicial no processo de convergência. Entretanto, além de lentos para convergir, não é possível prever o número de passos para seu término.

Os algoritmos analíticos são baseados na inversão formal da transformada de Radon, para tal faz uso dos algoritmos como: Convolução, Transformadas de Fourier e

Retroprojeção. Nesses algoritmos, o número de passos para sua execução é conhecido e determinado em função do tamanho da entrada de dados. Em geral, esses algoritmos reconstróem imagens com melhor qualidade e em menos tempo que os algoritmos algébricos. Dentre os algoritmos analíticos o mais disseminado é o denominado Retroprojeção Filtrada, devido a sua fácil implementação e boa qualidade da imagem. Nele, a reconstrução da imagem é feita filtrando-se inicialmente cada projeção unidimensional e em seguida aplicando-se a operação de retroprojeção (vide Apêndice C). Caso o filtro seja aplicado no domínio do espaço a operação de filtragem é a convolução da resposta impulsiva do filtro com a projeção, e se for utilizado o domínio da freqüência, aplica-se a Transformada Direta de Fourier na projeção, seguida do produto pelo filtro e por fim aplica-se a Transformada Inversa de Fourier ao resultado do produto.

A *Retroprojeção* é uma operação inversa ao processo da aquisição de projeções. Ela consiste na atribuição dos valores dos raios soma, de todas as projeções, para as células interceptadas por eles no plano X - Y , durante o processo de aquisição. Assim, a partir de uma função unidimensional, que é a projeção, cria-se uma função bidimensional no plano X - Y através da Retroprojeção. A soma dos valores de todas as funções bidimensionais, ou retroprojeções, no plano X Y , é chamada de imagem reconstruída.

Na operação de retroprojeção, considera-se o valor a ser retroprojetado passando pelo centro de cada pixel que compõe a imagem, como mostra a Figura 5. Contudo, devido ao posicionamento do eixos S_i inclinados em relação ao plano X - Y e a natureza discreta dos valores dos raios soma das projeções $p_i(s, \theta_i)$, somente alguns pontos retroprojetados incidem sobre o centro do pixel. Portanto, é necessário realizar uma interpolação com os raios soma adjacentes ao centro do pixel para estimar o valor a ser retroprojetado. A qualidade da imagem e a velocidade da reconstrução também dependem da técnica de interpolação utilizada na retroprojeção. Neste trabalho usou-se a interpolação linear que tem sido bastante empregada com sucesso nos algoritmos de retroprojeção filtrada, pois produz imagens com boa qualidade e é de fácil implementação [5].

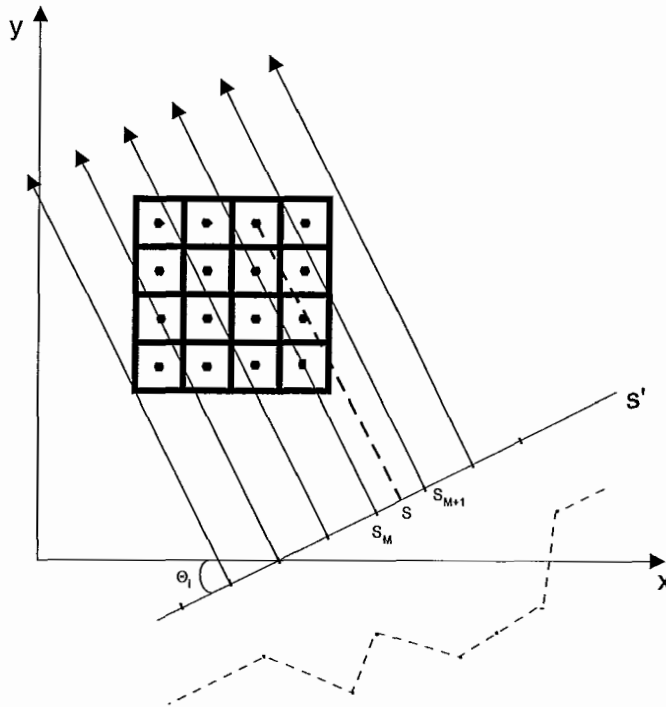


Fig. 5: Retroprojeção de uma projeção $p_i(s, \theta_j)$.

1.2 MOTIVAÇÕES

A evolução tecnológica em várias áreas de pesquisa tem colaborado para os avanços alcançados em Tomografia, podendo-se destacar:

- i) Novos métodos para geração de raios X;
- ii) O avanço da Tecnologia da Computação;
- iii) Melhoria do sistema de coleta de dados.

Em particular, os atuais sistemas de coleta de dados permitem:

- i) Aumentar o número de detetores por área, oferecendo maior resolução;
- ii) Diminuir o tempo de aquisição de dados.

Por outro lado, ao aumentar a quantidade de dados a serem processados por unidade de tempo, criou-se a necessidade de sistemas computacionais com maior desempenho.

Um perfeito exemplo, que influenciou fortemente o desenvolvimento desta tese, foi a necessidade computacional do Laboratório de Instrumentação Nuclear da COPPE, com a introdução de sistemas de detecção de raios X baseadas em CCD (*Charged Coupled Device*) para o uso em Microtomografia. Este sistema possibilita a rápida coleta de dados, por exemplo: linha de 512 pontos e imagem de 512 x 512 pontos, num tempo da ordem de décimos de segundo por projeção. A tecnologia CCD permite agregar numa pequena área uma grande quantidade de detetores. O tempo de exposição para adquirir uma projeção, em geral é muito pequeno, principalmente para se evitar a saturação do CCD e a integração do ruído térmico. Também para se evitar o ruído térmico, é necessário fazer uma coleta de dados rápida. Garantida a coleta rápida de dados surge o problema de se atingir alto desempenho no processamento desses dados.

1.3 ALTO DESEMPENHO COMPUTACIONAL EM RECONSTRUÇÃO DE IMAGENS

Em algoritmos do tipo retroprojeção filtrada, para feixes paralelos, somente a etapa de retroprojeção para uma imagem com 512 x 512 pontos, requer cerca de 10^8 operações de adição e multiplicação. No caso de algoritmos para feixes divergentes, tem-se a mais, e na mesma ordem de grandeza, operações de divisão e recíproco de um número elevado ao quadrado. Logo, a demanda computacional para algoritmos de feixe divergente é muito superior a de algoritmos de feixe paralelo.

A reconstrução de imagens tomográficas é uma tarefa de grande esforço computacional, que pode alcançar um desempenho satisfatório em dispositivos especializados. No entanto, em aplicações que requerem: (i) imagens de grandes dimensões; (ii) reconstrução em tempo real, ou ambas, o tempo de reconstrução torna-se crítico. Assim muitos esforços foram direcionados para melhorar o desempenho dos algoritmos de reconstrução, quando estes são executados dentro de limites estritos de tempo. Neste sentido, os algoritmos analíticos são vantajosos, já que são mais rápidos que os algoritmos iterativos e são naturalmente paralelizáveis. O paralelismo do algoritmo está

no fato que cada filtragem e retroprojeção de uma projeção pode ser feita independentemente das demais projeções. Nesses algoritmos o elevado número de operações está mais relacionado ao grande número de dados a serem processados do que ao número de operações efetuadas sobre eles.

Inicialmente, os algoritmos de reconstrução de imagens foram implementados em minicomputadores que possuíam placas com acelerador matemático. O aumento do tamanho das imagens e a necessidade de diminuir o tempo de reconstrução motivaram pesquisas voltadas para a exploração de paralelismo nos algoritmos de reconstrução. Encontra-se na literatura vários trabalhos que implementam algoritmos de reconstrução de imagem em computadores paralelos [8-20], onde os elementos de processamento variam de processadores de uso geral a processadores especializados como DSPs (*Digital Signal Processors*)[12,14,19]. Com o avanço da microeletrônica e áreas afins, passou-se a pesquisar e desenvolver circuitos integrados dedicados à reconstrução de imagens [13,15-18].

Entre os sistemas multiprocessadores utilizados para aumentar o desempenho dos algoritmos de reconstrução de imagens, o “*Parallel Pipeline Projection Engine*” ou P³E [8] foi um dos mais inovadores. O P³E é formado por um *pipeline* de elementos processadores, cada um contém uma ALU (Unidade lógica aritmética), DSP, memória e tabelas *Look up*. O P³E foi concebido para suportar uma grande variedade de algoritmos para processamento de imagens, além de reconstrução de imagens. O alto *throughput* atingido no P³E para reconstrução de imagens é devido ao tratamento dos dados da imagem de forma *raster scan* e à exploração do paralelismo inerente do algoritmo de retroprojeção filtrada.

Assim, inicialmente as projeções são uniformemente distribuídas entre os nós e os *DSPs* dos nós realizam a filtragem das projeções. O formato *raster scan* adotado no processamento faz com que cada pixel da imagem seja produzido por vez, como se a imagem fosse um vetor unidimensional. A medida que um pixel é retroprojetado no nó de processamento, seu valor é somado à contribuição vinda do nó anterior e passado ao nó seguinte formando um *pipeline*, até que no último estágio do *pipeline* o resultado será os pontos da imagem reconstruída.

Várias arquiteturas foram derivadas do P³E com o propósito de efetuar apenas o algoritmo de retroprojeção filtrada. Nesta abordagem, os DSPs foram retirados dos nós de processamento, ficando apenas um nó responsável pela filtragem de todas as projeções e pela distribuição das projeções filtradas para os nós do *pipeline* [12-18]. Tal mudança é possível sem alterar muito o desempenho global, pois o tempo gasto na retroprojeção é muito maior do que o tempo necessário para efetuar a filtragem em uma projeção. Essa separação entre *hardware* para filtragem e para retroprojeção será também explorada neste trabalho.

1.4 SISTEMAS RECONFIGURÁVEIS

O desenvolvimento dos dispositivos programáveis em campo [23-24], FPGAs (*Field Programmable Gate Arrays*), em especial os baseados em memória RAM, viabilizou um novo modelo de computação, híbrido dos tradicionais modelos de computação, de uso geral e computação de aplicação específica. O uso dos FPGAs é bastante atrativo pelo baixos custos de projeto e implementação, para quantidades em pequena escala. Mesmo tendo um desempenho inferior a um ASIC (*Application Specific Integrated Circuit*) equivalente e quase sempre menos eficiente em área, a flexibilidade de alteração de um projeto feito em FPGAs, seja para resolver erros de projeto ou para evolução, é de custo diminuto comparado aos de ASICs. Já a computação de propósito geral, efetivamente apoiada por compiladores, e outras ferramentas de programação, fornecem um ambiente no qual qualquer aplicação pode ser facilmente mapeada, rapidamente modificada e testada a custo ínfimo. Entretanto, o tempo de resposta para uma determinada tarefa é às vezes maior do que o desejado.

A este novo modelo de computação tem se dado diversos nomes, incluindo, *sistemas reconfiguráveis* ou *computação configurável*. A noção de “configurável” está no fato dos dispositivos poderem alterar o uso de seus recursos internos em *hardware* em função da configuração carregada [24-63].

Os primeiros protótipos de computação configurável demonstraram ser capazes de oferecer alto desempenho a custo baixo. Em certas aplicações o desempenho é superior ao de super-computadores [33, 36], apesar da baixa capacidade de recursos dos primeiros FPGAs. Recentemente, a computação em ambiente reconfigurável tem recebido muita atenção e fomentado intensamente a pesquisa e o desenvolvimento de aplicações tanto na área acadêmica como na área industrial. Atualmente, os sistemas reconfiguráveis estão presentes em aplicações onde é exigido alto desempenho e execução em tempo real. Por exemplo, o desenvolvimento de algoritmos para criptografia RSA explora o paralelismo e a particularização de multiplicadores para operandos muito grande e módulo de exponenciação [24, 25]; em processamento de sinais e imagens é possível encontrar uma quantidade imensa de sistemas configuráveis de alto desempenho onde o uso da técnica de *pipeline* e a implementação de multiplicadores dedicados são extensivamente explorados [35, 41, 42, 46, 47, 48, 52, 53, 54, 56, 62].

De fato, no uso de filtragem e convolução, onde os coeficientes são conhecidos, os multiplicadores podem ser facilmente implementada de forma dedicada. No casamento de cadeias de DNA, o alto desempenho é atingido pela replicação das unidades que fazem o casamento dos códigos [33]. Entre outros exemplos, tem-se o uso de sistemas híbridos em telecomunicações favorecendo a flexibilização de protocolos [61]; reconhecimento de padrões [51, 55]; sistemas embarcados [29] e várias transformadas como coseno, Hough, [41, 54], e emuladores de *hardware* de alto desempenho [27, 38, 45].

As principais condições para que as aplicações numéricas atinjam alto desempenho utilizando computação configurável, são:

- i) Dispositivos com grande densidade de recursos;
- ii) Utilização de técnicas de paralelização tal como múltiplas estruturas *pipeline* [35, 50, 54, 58, 62];
- ii) Personalização de funções aritméticas ou lógicas, geralmente com quantidade de bits não padrão mas adequado ao problema [26, 32, 34, 35, 42, 47, 49, 50, 52, 53, 56].

Individualmente, as implementações de unidades aritméticas com um número médio de bits, 16 a 32 bits, mapeadas em FPGAs oferecem desempenho da ordem de poucas dezenas de Megahertz, portanto muito inferior aos processadores modernos que trabalham em centenas de Megahertz. No entanto, é possível agregar um conjunto grande de operações aritméticas nas arquiteturas que suportam computação configurável para aumentar substancialmente o desempenho.

Em sistemas tradicionais, a arquitetura dedicada é obtida com a perda de flexibilidade, assim, uma arquitetura dedicada para uma área de aplicação é invariavelmente ineficiente ou inadequada para atender outras aplicações. Já nos sistemas configuráveis baseados em FPGAs, a especialização não sacrifica a flexibilidade. Os FPGAs permitem a reconfiguração de seus recursos de acordo com a aplicação a ser executada. Isto é bastante vantajoso para sistemas tomográficos, pois além da reconstrução de imagens, o processamento tomográfico ainda requer um conjunto de operações de grande intensidade numérica tais como: segmentação de imagens, visualização, filtragens, reconhecimento de padrões e outras tarefas que auxiliem os diagnósticos em imagens reconstruídas.

1.5 CONTRIBUIÇÕES DA TESE

Esta tese explora o potencial da computação configurável baseada em FPGAs para propor uma solução nova para o problema de reconstrução de imagens. Em particular, esta tese investiga e propõe novas soluções para duas tarefas específicas do algoritmo de reconstrução de imagens: filtragem das projeções e operação de retroprojeção. O uso de operações em ponto flutuante para implementação das operações aritméticas existentes no algoritmo foi descartado devido à alta relação custo/desempenho. Em particular, o alto custo em elementos lógicos necessário para implementar multiplicadores em FPGAs também desmotivou o uso de aritmética de ponto fixo.

Esta tese propõe como solução o uso da aritmética não convencional conhecida como aritmética de Resíduos (*Residue Number System*) para o problema de reconstrução de imagens. Muitos esforços foram dirigidos ao estudo de RNS e às suas aplicações nas

principalmente nas décadas de 70 e 80 [64-106], cujas principais contribuições foram na implementação de filtros digitais [65, 67, 72, 74, 75, 76, 79, 80] de alta velocidade em relação à aritmética binária convencional e aos sistemas aritméticos tolerantes à falha [69, 81]. A principal vantagem de RNS é a possibilidade de subdividir as operações de adição e multiplicação em módulos menores que podem trabalhar em paralelo. Assim, a busca de uma alternativa ao uso da aritmética de ponto flutuante, permitiu estabelecer, com pioneirismo, as bases preliminares para o uso de aritmética de resíduos na solução do problema de reconstrução de imagens utilizando-se o algoritmo de retroprojeção filtrada.

Além disso, o emprego de FPGAs nas diversas áreas da computação numérica é relativamente novo. Assim, nesta tese analisou-se vários aspectos associados tanto à implementação de operações aritméticas de resíduos em FPGAs, como na conversão de um sistema binário convencional para o baseado em resíduo e sua inversão, isto é, a passagem de resíduos para o sistema binário convencional. Tais análises permitiram mostrar a viabilidade de implementações de alto desempenho usando os recursos disponíveis nos FPGAs [106]. Com a tecnologia de FPGA existente é possível atingir desempenho de dezenas à centena de Megahertz.

Mostrou-se que o uso de arquiteturas reconfiguráveis baseadas em FPGAs, para aplicações que utilizam aritmética de resíduo podem ter o custo da inversão resíduo-binário muito pequeno. Para tal basta carregar, nos FPGAs ao término da aplicação, uma nova configuração a qual implementa o circuito que realizará inversão resíduo-binário.

Os resultados experimentais demonstram que, sem explorar o paralelismo intrínseco do algoritmo de retroprojeção filtrada, a proposta apresentada neste trabalho tem um desempenho de 1 a 2 ordens de grandeza superior às implementações que utilizam processadores de uso geral.

Mais especificamente, as contribuições dessa tese são:

- (i) Introdução da primeira biblioteca para FPGAs, com operações de adição e multiplicação em resíduos para todos números primos de 3 à 6 bits;
- (ii) Introdução de circuitos de conversão binário-resíduo para tecnologia FPGA empregada;
- (iii) Um mapeamento eficiente do algoritmo “teorema do resto Chinês” para inversão resíduo binário;
- (iv) Uma implementação da unidade que realiza convolução em RNS, com desempenho elevado, que combina características do cálculo de índices, aplicado na multiplicação, com um novo modelo de multissomador modular adequado à tecnologia FPGA;
- (v) Uma implementação da unidade que realiza retroprojeção para feixes paralelos em RNS com alto desempenho.
- (vi) O desenvolvimento da unidade de retroprojeção para feixes divergentes, em particular a proposta e implementação de soluções para operações tais como divisão e inverso de um número elevado ao quadrado, utilizando de forma otimizada os recursos disponíveis do modelo FPGA, e não comprometendo o alto desempenho atingido pelas sub-unidades que trabalham em resíduo nessa mesma unidade.

1.6 ORGANIZAÇÃO DA TESE

Esta tese está organizada da seguinte forma:

No Capítulo 2 são apresentados o modelo do dispositivo reconfigurável usado no trabalho e os problemas surgidos com sua aplicação em aritmética em ponto flutuante.

No Capítulo 3 é apresentada uma breve introdução à aritmética de Resíduos e custos e desempenhos associados à implementação de operações aritméticas e as passagens de sistema binário convencional para RNS e vice-versa em FPGA.

No Capítulo 4 são apresentadas as expressões numéricas do algoritmo de retroprojeção filtrada para feixes paralelos e feixes divergentes, alguns aspectos relacionados à implementação desse algoritmo em RNS, e uma estimativa da faixa dinâmica para reconstrução de imagens com 512 x 512 pontos.

Uma proposta de uma arquitetura para o algoritmo de retroprojeção filtrada com seus custos e desempenhos associados são apresentadas no Capítulo 5.

Uma revisão dos diversos trabalhos que exploram o paralelismo do algoritmo de retroprojeção filtrada é apresentada no Capítulo 6.

As conclusões e trabalhos futuros estão presentes no Capítulo 7.

AVALIANDO OPERAÇÕES ARITMÉTICAS EM FPGAS

Neste capítulo é introduzido resumidamente alguns dos dispositivos lógicos programáveis mais utilizados em projetos de circuitos digitais, situando a posição dos FPGAs em relação aos demais dispositivos. A seguir é apresentado o modelo de FPGA a ser empregado nesta tese, sob o qual é feita uma discussão preliminar do ponto de vista de mapeamento de funções genéricas. Além disso são mostradas as limitações de se empregar as operações aritmética em ponto flutuante em FPGAs convencionais.

2.1 INTRODUÇÃO AOS DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

Existe uma grande variedade de dispositivos lógicos programáveis disponíveis no mercado [107]. Os primeiros dispositivos surgiram com o objetivo de auxiliar a integração de componentes de maior capacidade, como memórias, microprocessadores, controladores, entre outros. Atuavam como decodificadores, codificadores, em lógica de seleção e geração de funções muito simples que necessitavam de pequenos recursos para serem implementadas. Entre os principais dispositivos desta categoria estão os PALs (*Programmable Array Logic*) e PLAs (*Programmable Logic Array*), que basicamente implementam funções em dois níveis do tipo soma de produtos. A estrutura interna dos PALs é formada por um conjunto de portas *AND* com entradas programáveis, cujas saídas alimentam portas tipo *OR* com um número fixo de entradas, já os PLAs têm uma estrutura semelhante aos PALs, com portas *ANDs* alimentando as portas *ORs*, porém em ambos as entradas são programáveis. Uma evolução simples destes circuitos foi a incorporação de *Flip Flops* programáveis a sua saída, permitindo-se mapear circuitos seqüenciais como máquinas de estados. O sucesso desses dispositivos foi em parte devido a fácil utilização, uma vez que o próprio usuário poderia (re)programá-los em seu ambiente de desenvolvimento. Tais dispositivos hoje são chamados genericamente de PLD simples.

O crescente desenvolvimento da tecnologia de integração permitiu aumentar a capacidade dos circuito lógicos programáveis, criando-se uma nova classe de dispositivos

lógicos programáveis chamada de PLDs complexos. Os PLDs complexos agrupam um conjunto de PLDs simples num mesmo dispositivo que internamente estão associados à uma estrutura de interconexão programável que permite a ligação entre eles, como também aos pinos de entrada e saída. Os primeiros PLDs complexos, inicialmente, tinham sua capacidade medida através da multiplicidade de seus recursos em relação aos PALs. Os PLDs complexos mais recentes, devido à sua grande capacidade, em geral, são medidos em forma de equivalência de portas NANDs de duas entradas, fazendo alusão à medida utilizada nos GATE ARRAYS. Uma das principais características dos PLDs complexos é a previsibilidade do tempo de resposta de um circuito mapeado, devido à baixa complexidade no roteamento interno de suas estruturas.

Existem também os dispositivos semi-personalizados de grande capacidade de integração, classificados como *Standard Cell* e *Programmable Gate Arrays (PGAs)*. Esses dispositivos permitem ao usuário projetar circuitos digitais dedicados de grande capacidade e de alto desempenho a partir de circuitos padrão já existentes. Em particular, os PGAs dispensam as etapas dispendiosas como confecção de máscaras de integração e a produção dos pastilhas (difusão no processo de integração). Neste tipo de tecnologia, os circuitos projetados são mapeados em células padrão, como portas NANDs ou NORs de duas entradas, já previamente integradas no semicondutor, faltando apenas a etapa de ligação metálica. Simplificadamente, pode-se dizer que o projetista ao, término das etapas de projeto, validação e simulação envia ao fabricante destes dispositivos semi-prontos o conjunto de ligações metálicas a serem feitas. Assim, um projeto dedicado feito nessa tecnologia é caro, devido a elaboração de máscaras para integração das ligações metálicas. Logo, o custo por unidade cai de acordo com o volume produzido de circuitos integrados. Portanto, esta tecnologia não é adequada para atender a poucas unidades de circuitos integrados ou ser usada na produção de circuitos protótipos.

Pode-se dizer que os FPGAs preenchem o espaço existente entre os PLDs complexos e os PGAs, pois atualmente, têm uma capacidade em portas lógicas equivalente a muitos modelos de PGAs e maior do que os PLDs complexos. Sendo um dispositivo programável no ambiente de desenvolvimento, os custos de implementação, correção e

modificação são muito pequenos em relação ao uso de PGAs. A principal característica que diferencia os FPGAs dos PLDs complexos é o abandono da matriz de AND/ORs na implementação de funções através da utilização de tabelas de pequeno tamanho. Num FPGA existe uma grande quantidade de tabelas de poucas entradas que se encontram distribuídas no dispositivo, que de uma certa maneira lembra as portas NANDs num PGA.

Em particular, os FPGAs baseados em memória RAM permitiram o seu uso em computação, entre outros fatores, pela não necessidade de se retirar o componente da placa para realizar a mudança de sua configuração. Assim, cada nova aplicação computacional é estabelecida através do carregamento de uma configuração apropriada no dispositivo. Além dos FPGAs que são capazes de implementar funções digitais, surgiram outros dispositivos configuráveis baseados em memória RAM, que implementam apenas interconexões programáveis [108, 109]. Estes dispositivos atuando em conjunto com FPGAs oferecem uma maior flexibilidade na ligação, ou roteamento, dos componentes de uma arquitetura reconfigurável [32, 39].

2.2 INTRODUÇÃO AOS FPGAS BASEADOS EM MEMÓRIA RAM

Os principais componentes das mais recentes arquiteturas reconfiguráveis são os FPGAs baseados em memórias RAM estáticas. Nos RAM FPGAs as células de memória RAM se encontram distribuídas dentro do dispositivo que, devidamente organizadas, formam blocos que implementam funções lógicas, chaves do sistema de interconexão e outras funções programáveis como seleção do tipo de funcionamento do pinos do dispositivo, seleção dos sinais a serem conectados ao sistema de interconexão e etc. Do ponto de vista funcional, um RAM FPGA convencional pode ser visto como composto por um grande número de tabelas *Look Ups*, sistema de interconexão e elementos de entrada/saída, conforme apresentado na Figura 2.1, que são definidos a seguir:

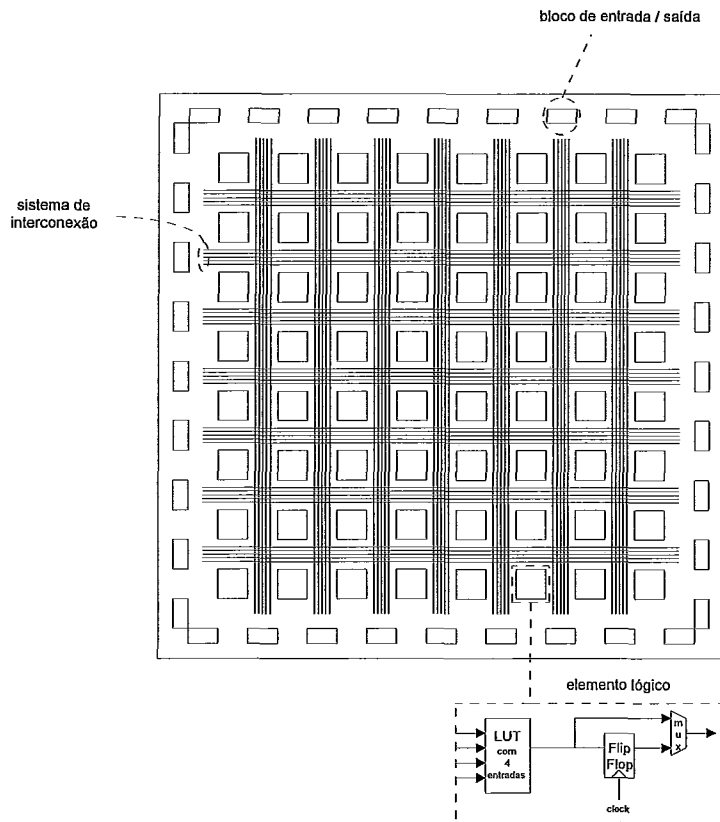


Fig. 2.1: Modelo convencional de FPGA

i) *Elementos lógicos reconfiguráveis (LEs)* possuem internamente: uma ou mais tabelas *look-up* (LUTs) de poucas entradas (4 a 5); *flip-flops*; multiplexadores que selecionam os sinais de entrada e de saída para serem ligados aos canais de interconexão. Nos LEs são realizadas funções lógicas combinacionais ou seqüenciais. Alguns modelos de FPGAs permitem a conexão serial de elementos lógicos de forma a oferecer outras funções tais como contadores e somadores do tipo *ripple carry*. Nessas configurações especiais são utilizadas estruturas internas dedicadas, que permitem uma rápida propagação do *carry* gerado num elemento lógico para a sua inclusão no elemento seguinte.

ii) *Elementos de entrada /saída*: são usados para ligar o interior do FPGA à parte externa. São formados por pinos do dispositivo e circuitos *driver/tri-state* que podem ser programados individualmente de três maneiras: somente com saída, ou somente com entrada e como saída e entrada, isto é, bidirecional. Alguns modelos possuem ainda *flip-*

flops que associados aos sinais de relógio realizam saídas ou entradas síncronas e mecanismos que permitem variar a velocidade de resposta do circuito de *driver*.

iii) *Sistema de interconexão*: formam os caminhos por onde trafegam os sinais trocados entre os elementos lógicos, memória e os blocos de entrada /saída, e o relógio. É composto por um conjunto de canais que formam caminhos horizontais e verticais entre os elementos lógicos e blocos de memória. Em alguns modelos de FPGA existem ainda chaves programáveis que permitem ligações dos canais que se encontram segmentados dentro do sistema de interconexão.

Outros elementos que vêm se tornando comum em vários modelos de FPGAs são os *blocos de memórias*, sejam distribuídos ao longo do dispositivo ou em blocos de pequena capacidade com posições fixas no dispositivo [110]. O modelo de FPGA adotado nesse trabalho segue o modelo de memória composta por pequenos blocos de memória envolvidos pelo sistema de interconexão. Em geral, a memória pode ser utilizada na implementação de funções lógicas ou ter função de armazenamento, como memórias RAM, ROM e FIFOS. Neste trabalho um bloco de memória é formado por um conjunto de 2048 bits que podem ser configurados segundo um dos seguintes modelos de memória:

- i) 256 posições com 8 bits de largura;
- ii) 512 posições com 4 bits de largura;
- iii) 1024 posições com 2 bits de largura;
- iv) 2048 posições com 1 bit de largura.

Pode-se dizer que os FPGAs, em geral, são ricos em registradores e pobres em entradas para elementos lógicos. Logo, o mapeamento de funções com um número médio de variáveis, ou entradas, poderá envolver uma quantidade considerável de elementos lógicos. O desempenho dessa função está relacionado ao caminho crítico formado pela passagem do sinal pelo elementos lógicos envolvidos, mais os atrasos associados aos canais de interconexão utilizados para ligá-los.

Uma técnica muito comum e bastante empregada para aumentar o desempenho de funções mapeadas em FPGAs é a inserção de *flip-flops* às funções, aproveitando-se os *flip-flops* existentes nos elementos lógicos [111]. Assim, a implementação dessas funções sob uma forma síncrona no estilo *pipeline* é facilitada. Esta técnica aumenta a latência para produzir um primeiro resultado, contudo permite aos circuitos trabalharem em frequências mais elevadas.

2.2.1 Metodologia de projeto de circuitos usando FPGAs

Simplificadamente, a realização de um projeto em FPGAs pode ser dividida em 3 partes principais: projeto digital, implementação e verificação. As ferramentas utilizadas no projeto são editores de esquemáticos, diagramas de tempo e linguagens de descrição de *hardware*. O resultado da etapa de projeto geralmente é um arquivo, *netlist*, que descreve o projeto no nível de portas lógicas e pode conter ainda informações de restrições de tempo. A etapa de implementação utiliza o *netlist*, dividindo-o e mapeando as subdivisões entre os elementos lógicos, os elementos de entrada e saída, ou nos blocos de memória que compõem o FPGA. O circuito particionado deve ser mapeado nos elementos lógicos de forma a atender às restrições de tempo e facilitar o roteamento. Após o roteamento do circuito são gerados *netlists* com a situação real da implementação no FPGA. Estes *netlists* são as entradas para a etapa de simulação, que será utilizada na verificação e validação do projeto, além de proporcionar realimentações para fases futuras de otimização do projeto.

Todos os recursos disponíveis num FPGA são configurados através de uma fila de *bits* que é carregada a partir de um dispositivo externo. Os RAM FPGAs podem ser configurados inúmeras vezes, e podem ser classificados em classes, segundo o tipo de configuração: Estática ou dinâmica. Nos FPGAs de configuração estática, a configuração deve ser carregada de uma só vez e só é permitido mudança após o término da execução. Os FPGAs com configuração dinâmica permitem que sua configuração seja totalmente, ou parcialmente, modificada enquanto o resto do circuito se mantém funcionando.

Os principais objetivos quando se utiliza uma configuração estática são: aumentar o desempenho de uma dada função e otimizar a utilização dos recursos [63]. A configuração dinâmica pode ser vista como uma técnica que reduz o tempo de configuração e de uma certa forma pode reduzir os requisitos totais da quantidade de *hardware* necessário para implementar certas aplicações. Isto permite alterar parte da configuração de forma temporal, isto é, à medida em que a aplicação não utiliza mais uma parte dos recursos alocados, pode-se carregar nessas regiões uma nova configuração parcial que será necessária à aplicação num momento adiante. Cria-se o conceito de “*hardware virtual*” com capacidade “ilimitada” de *hardware* [43, 44, 59, 60, 112, 113].

Um grande número de funções existentes neste trabalho não se encontram disponíveis em bibliotecas para FPGAs, como por exemplo, multiplicação e adição modular, e portanto, deverão ser implementadas e ter seus desempenhos avaliados. Simplificadamente, pode-se dizer que as ferramentas CAD estabelecem duas alternativas básicas para se mapear uma função: (i) Minimizar o número de elementos lógicos ou; (ii) Tentar explorar o máximo de velocidade, o que em geral implica em uma maior utilização de elementos lógicos. Entretanto, as ferramentas disponíveis geralmente não são capazes de oferecer, com sucesso, o mapeamento de circuito seqüenciais. O desejado de uma ferramenta na implementação de algumas funções seqüenciais existentes neste trabalho, seria: i) Fornecer juntamente com a função, esta descrita de maneira combinacional, o número de níveis de profundidade a ser atingindo no mapeamento; ii) Permitir a transformação desta função, descrita em forma combinacional, em função seqüencial através da inclusão de *flip flops* que se encontram disponíveis nos elementos lógicos. Por causa das atuais limitações, adotou-se uma forma simples para mapear diretamente as funções sob a forma *pipeline*, que será mostrada a seguir.

2.3 MAPEAMENTO DE FUNÇÕES GENÉRICAS EM SRAM FPGAS

O mapeamento de funções com um número de variáveis maior do que o número de entradas da tabela *look up*, em geral, pode envolver um número variado de LUTs, mesmo quando o número de variáveis for igual. Contudo, existe um número máximo de LUTs para o pior caso, chamado neste trabalho de *função genérica*. Por exemplo: algumas funções do

tipo $f(a,b,c,d,e) \rightarrow \{0,1\}$, que possuem 5 entradas e 1 saída, podem ser mapeadas em combinações de 2 LUTs de quatro entradas, como pode ser visto nas Figuras 2.2a, 2.2b e 2.2c. Entretanto, essas estruturas individualmente não são capazes de mapear qualquer função de 5 entradas. A estrutura mostrada na Figura 2.2d, que possui 3 LUTs de quatro entradas, é capaz de mapear qualquer função de cinco entradas. Esse modelo de mapeamento, que só leva em consideração o número de bits de entrada da função a ser mapeada é chamado de *mapeamento de função genérica*.

Neste trabalho, ao se mapear funções genéricas utilizou-se também os *flip-flops* existentes nos elementos lógicos com o propósito de obrigar a função síncrona a trabalhar sob a forma *pipeline*, para se obter maior desempenho. As Figuras 2.3 e 2.4 representam os mapeamentos de funções genéricas de 5 e 6 variáveis de entrada com 1 bit de saída síncronas com o relógio. Pode-se ver que são introduzidos outros elementos lógicos necessários à realização do *pipeline* que é representado por tracejados nas figuras.

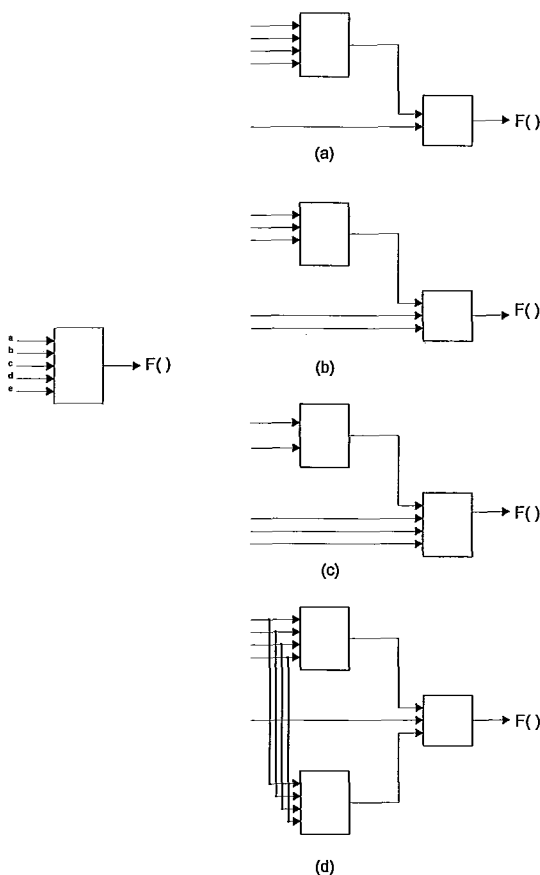


Fig. 2.2: Modelos de mapeamentos para função com 5 entradas mapeadas em LUTs com 4 entradas.

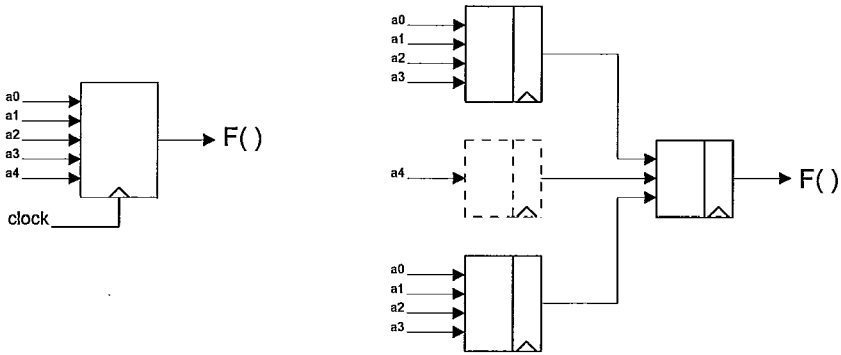


Fig. 2.3: Mapeamento *pipeline* de uma função genérica com 5 entradas.

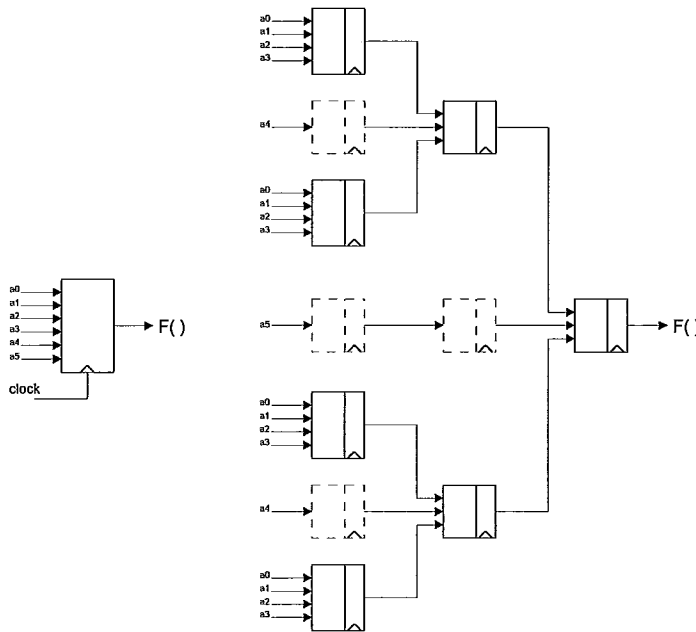


Fig. 2.4: Mapeamento *pipeline* de uma função genérica com 6 entradas.

2.4 OPERAÇÕES ARITMÉTICAS EM PONTO FLUTUANTE EM FPGAS

A computação científica, em geral, pode ser caracterizada pela manipulação de números reais de valores muito elevados e valores muito pequenos. A representação direta para esses números é muito custosa, exigindo um número muito grande de bits. Uma

alternativa conhecida como *aritmética de ponto fixo*, possui uma faixa limitada para representação de valores e requer o uso de escalonamento de valores a fim de que os resultados não ultrapassem os valores permitidos da faixa, podendo tal escalonamento ser feito em nível da aplicação ou em nível das operações aritméticas [114].

O escalonamento feito em nível das aplicações requer uma análise do problema para determinar os fatores de multiplicação que devem ser aplicados aos valores iniciais e aos valores parciais a fim de que as computações reais envolvidas no problema se mantenham sempre na faixa permitida. O resultado do problema deve ser restaurado para seu valor real através de multiplicações. No nível das operações aritmética os operandos podem ser escalonados antes ou após as operações de forma a se manterem dentro de um formato apropriado. Todo esse trabalho de escalonamento, além de tornar a computação de problemas simples numa tarefa mais trabalhosa, torna sua implementação mais suscetível a erros de codificação. A representação em ponto flutuante surge como uma alternativa capaz de representar números reais dentro de um número finito de bits, através de uma representação um pouco mais complexa. Contudo, retira do programador o trabalho de escalonamento. No entanto, esta facilidade tem um preço: a realização das operações em ponto flutuante requer um número grande de passos para sua implementação [114-116].

2.4.1 Representação em Ponto flutuante

O padrão de ponto flutuante IEEE 754 é um dos utilizados em dispositivos aritméticos e é composto por 3 campos principais:

Sinal	E xpoente polarizado	Mantissa
--------------	-----------------------------	-----------------

O campo de sinal, S, está na posição mais significativa da representação, onde o valor 1 indica que o número é negativo e o valor 0 que o número é positivo. O campo dos expoentes, E, está na forma polarizada, isto é, seu valor encontra-se somado a um valor constante. Esse tipo de notação permite que a comparação de magnitude entre números de ponto flutuante possa ser feito de forma simples, como se fossem números inteiros. A

mantissa, M, tem seus bits representando um número fracionário não sinalizado da forma $1.f_0f_1f_2\dots$, onde o valor inteiro 1 não necessita ser representado, estando implícito na representação.

Nesse padrão são definidos 4 formatos para representar números de ponto flutuante, como nota a tabela 2.1, a seguir:

	Simple	Simple estendido	Duplo	Duplo estendido
Tamanho da representação	32 bits	44 bits	64 bits	80 bits
Parte fracionária + bit escondido	23 + 1 bits	32 + 1 bits	52 + 1 bits	64 + 1 bits
Expoente	8 bits	11 bits	11 bits	15 bits
Polarização do expoente	127	1023	1023	16383
Faixa dinâmica Aproximada	2^{128}	2^{1024}	2^{1024}	2^{16384}
Menor número normalizado	2^{-126}	2^{-1022}	2^{-1022}	2^{-16382}
Precisão aproximada	2^{-23}	2^{-32}	2^{-52}	2^{-64}

O valor de um número no formato precisão simples é dado por:

$$\text{Valor} = (-1)^S \times 2^{E-127} \times 1.f_0f_1f_2\dots f_{23} \quad (2.1)$$

Em geral, as operações em ponto flutuante necessitam de outros bits além dos apresentados acima: alguns utilizados no resultado de operações, como por exemplo indicando *overflow* e outros bits, 3 aproximadamente, de segurança utilizados para realização do arredondamento dos resultados das operações aritméticas.

2.4.2 Operações em ponto flutuante

A seguir são listadas resumidamente as etapas necessárias para a realização de soma/subtração e multiplicação em ponto flutuante [114].

(i) Etapas da soma/subtração.

- Calcular a diferença entre os expoentes;
- Deslocar a mantissa do menor número para à direita um número de vezes igual a diferença entre os expoentes;
- Adicionar as mantissas e atribuir ao expoente e ao sinal os respectivos valores do maior operando;
- Normalizar a mantissa e ajustar o expoente;
- Arredondar o resultado normalizado;
- Se necessário normalizar a mantissa e ajustar o expoente;

(ii) Etapas da multiplicação:

- Multiplicar as mantissas, multiplicar os sinais, adicionar os expoentes. Como os expoentes se encontram polarizados deve se subtrair o valor de uma polarização, para que o expoente fique com o resultado correto;
- Normalizar a mantissa e ajustar do expoente;
- Arredondar o resultado normalizado;
- Se necessário normalizar a mantissa e ajustar o expoente;

2.4.3 Mapeamento de operações em ponto flutuante em FPGAs

As implementações de operações em ponto flutuante em FPGAs têm se mostrado bastante custosas devido à própria complexidade das operações e às limitações dos FPGAs para se implementar operações aritméticas simples com um número razoável de bits. O resultado são implementações com baixo desempenho. Por exemplo, a operação de multiplicação em ponto flutuante, embora apresente um número de passos menor do que a adição, está condicionada ao uso ineficiente dos recursos do FPGA para se implementar multiplicação binária em FPGA. A literatura aponta que a implementação de multiplicadores em FPGA tem um custo em área cerca de 100 vezes maior e apresenta um desempenho cerca de 10 vezes menor do que uma implementação do multiplicador diretamente em *hardware* [117]. Os passos de arredondamento, detecção e o tratamento de exceções têm sido evitados, simplificados ou parcialmente implementados nos trabalhos descritos na literatura. As causas são o grande conjunto de regras e o custo de suas implementações. Entre os primeiros trabalhos utilizando-se FPGAs para realizar aritmética de ponto flutuante tem-se o trabalho de Bakkes *et al.* em [49], que utilizam dispositivos externos matemáticos para contornar os baixos recursos disponíveis nos primeiros dispositivos FPGAs e o baixo desempenho para a implementação de multiplicações. Nessa implementação, os FPGAs realizam o controle das operações e a manipulação dos operandos em ponto flutuante.

Um dos primeiros resultados conhecidos de operações em ponto flutuante totalmente mapeados em FPGAs [47], utilizou um formato próprio, que reduziu a precisão, isto é, o número de bits da mantissa é menor do que o sugerido pelo padrão IEEE para simples precisão. Além disso possui um número menor de bits para representar o expoente. Logo, as operações neste formato não padrão necessitam de uma menor área empregada em suas implementações. Os resultados alcançados com um formato de 18 bits para mantissa e 6 bits para expoente indicavam 8.6 MFLOPS de pico para soma/ subtração e 4.9 MFLOPS para multiplicação.

Em [50] Louca *et al.* realizaram implementações das operações de soma e multiplicação em ponto flutuante com precisão simples atingindo respectivamente 7 MFLOPS e 2.3 MFLOPS. Para contornar o custo do multiplicador, foi utilizado a técnica *dígito serial* em sua implementação. A aritmética tipo dígito serial [118] se apresenta como uma solução intermediária entre o processamento paralelo de todos os bits numa operação e o processamento serial de apenas um bit por vez. Nessa técnica é processado um número fixo de bits a cada passo oferecendo a melhor solução de compromisso considerando-se velocidade, área utilizada, *throughput* e limitações de pinos [48, 50, 58].

Em consequência dos baixos desempenhos alcançados nas implementações de ponto flutuante em FPGAs, investigamos a implementação de operações aritméticas em uma forma não convencional denominada *aritmética de resíduos* em SRAM FPGAs, uma vez que tradicionalmente esta oferece soluções mais rápidas do que as soluções em aritmética binária convencional para um mesmo tipo de tecnologia de integração empregada. O próximo capítulo introduz essa forma alternativa de aritmética discutindo suas vantagens e limitações.

CAPÍTULO 3

ARITMÉTICA DE RESÍDUOS EM FPGAS

Neste capítulo são introduzidos os sistemas numéricos baseados em resíduos e algumas de suas principais propriedades. São propostos e avaliados alguns modelos para a implementação em FPGA das operações de adição e multiplicação em resíduos. Também são apresentados: (i) o mapeamento em FPGA do algoritmo “Teorema do Resto Chinês” para a realizar a inversão resíduo-binário; (ii) as regras e as implementações utilizadas para mapear os conversores binário-resíduo em FPGA.

3.1 INTRODUÇÃO A SISTEMAS NUMÉRICOS BASEADOS EM RESÍDUOS

Os sistemas numéricos baseados em resíduos, RNS, se apresentam como uma alternativa de implementação aos sistemas binários tradicionais, devido à sua propriedade de decomposição de suas operações aritméticas de adição, subtração e de multiplicação, em um conjunto de sub-operações que podem ser executadas em paralelo [64-106]. Outras operações aritméticas como a divisão, detecção de sinal, *overflow*, escalonamento e comparação de magnitude, que além de não serem modulares, são bastante complexas para serem implementadas em *hardware* de alta velocidade. Consequentemente, sistemas baseados em resíduos são inadequados para computação de uso geral. No entanto, RNS tem sido aplicado em implementações de alto desempenho visando aplicações dedicadas, como em processamento digital de sinais [65, 67, 72, 74, 75, 76, 79, 80] e sistemas aritméticos tolerantes a falhas [69,81]. RNS é um sistema de números inteiros que trabalha com aritmética finita. Operações como adição, subtração e multiplicação não possuem problemas de arredondamento. Contudo, para prevenir a perda dos bits mais significativos devido ao *overflow*, sistemas RNS podem requerer uma grande faixa dinâmica, cujo valor deve ser estimado antes da execução do algoritmo utilizado

Um sistema baseado em Resíduos é construído a partir de um conjunto de números primos relativos entre si, chamados de *módulos*. Assim, seja $R = \{m_0, m_1, \dots, m_n\}$ uma

base RNS logo, tem-se que o maior divisor comum, $MDC(m_i, m_j) = 1$, para todos os valores de i e j , tal que $i \neq j$. A faixa dinâmica, M , isto é, o maior número inteiro num RNS, é determinada pelo produto dos módulos que compõem a base RNS, equação 3.1. Definindo conseqüentemente o conjunto de valores inteiros possíveis que se pode utilizar sem que ocorra *overflow*,

$$M = \prod_{i=0}^n m_i \quad (3.1)$$

Em geral, divide-se a faixa dinâmica em duas regiões: uma com números positivos e a outra com números negativos, de acordo com o valor de M . No caso de M ser um número par atribui-se a faixa dinâmica abrangendo o intervalo $[-M/2, M/2 - 1]$, em caso de M ímpar o intervalo é $[-(M-1)/2, (M-1)/2]$.

Um número inteiro x pode ser representado em um RNS por uma $n+1$ tupla de dígitos que representam o resíduo de x para cada *módulo*: $x = (r_0, r_1, \dots, r_n)$, onde r_i significa o primeiro resíduo positivo de x em relação ao *módulo* m_i . Se x for positivo então cada resíduo, $r_i = |x| m_i$, é o resto da divisão de x pelo respectivo *módulo* m_i . Os números negativos tem seus resíduos calculados pela expressão:

$$r_i = |m_i - |x|m_i| m_i. \quad (3.2)$$

3.2 PROPRIEDADES DE SISTEMAS BASEADOS EM RESÍDUOS

As operações aritméticas são composta de operações modulares independentes em cada *módulo* que forma a base do sistema RNS. Duas importantes propriedades existentes em operações modulares [64, 114, 115, 119], muito úteis neste trabalho são:

$$r_i = |x + y| m_i = | |x| m_i + |y| m_i | m_i \quad (3.3)$$

$$r_i = |x \cdot y| m_i = | |x| m_i \cdot |y| m_i | m_i \quad (3.4)$$

Algumas relações aritméticas em RNS recebem denominações especiais:

i) Inverso aditivo: O Inverso aditivo de um número x , sob *módulo* m_i , é um número que somado a x produz um valor cujo resíduo é zero. Assim tem-se:

$$| x + Inv_ad | m_i = 0 \quad (3.5)$$

ii) Inverso multiplicativo: O inverso multiplicativo de um número x , sob *módulo* m_i , é um número que multiplicado a x , produz um valor cujo resíduo igual a 1. Assim tem-se:

$$| x.C | m_i = 1 \quad (3.6)$$

onde C , recebe uma notação especial $| 1/x | m_i$. A existência do inverso multiplicativo é garantido se o *módulo* m_i for número primo e x for diferente de zero [119].

A execução de uma operação de adição, subtração ou multiplicação em RNS é feita independentemente em cada *módulo* e pode portanto ser executada em paralelo. Não ocorrendo propagação de *carry* entre os *módulos* nas operações de adição ou subtração, a multiplicação poderá ser realizada sem a necessidade de geração de produtos parciais.

3.3 IMPLEMENTAÇÃO DE OPERAÇÕES ARITMÉTICAS RNS EM FPGAS

As primeiras implementações das operações em resíduos foram feitas, principalmente, através do mapeamento das operações em tabelas utilizando-se memórias [65-75]. Portanto, o cálculo de uma simples operação em RNS era limitado pelo tempo de consulta à memória. A evolução dos dispositivos semicondutores associados às novas técnicas de integração, permitiu a construção de memórias a baixo custo, mais velozes e de maior capacidade, oferecendo aos pesquisadores subsídios para pesquisa e desenvolvimento de sistemas baseados em resíduos.

A próxima seção discute uma implementação inovadora de operações em RNS utilizando-se dispositivos do tipo *RAM* FPGAs. São analisadas várias opções de implementação de operações em RNS, utilizando-se o modelo de FPGA discutido no Capítulo 2.

3.3.1 Mapeamento direto das operações

O primeiro experimento consistiu em medir os custos de *hardware* associados a um simples mapeamento das operações, levando-se em conta o número de bits de entrada e o número de bits na saída. Adotou-se o modelo de mapeamento genérico descrito no Capítulo 2.

A Tabela 3.1 apresenta os custos associados à implementação do mapeamento direto utilizando-se duas abordagens: usando-se somente elementos lógicos, LEs, e usando-se apenas blocos de memória. O mapeamento direto em LEs é claramente ineficiente, uma vez que o número de LEs aumenta exponencialmente com o número de bits dos operandos da operação. O número de blocos de memória também aumenta exponencialmente, porém para operações com operandos que possuem poucos bits, o uso do mapeamento em memória pode ser útil no sentido de economizar uma quantidade significativa de LEs.

Tabela 3.1: Mapeamento de operações RNS genéricas.

Tamanho do resíduo em bits	Entrada, no. bits	Saída, no. bits	No. LEs	No. blocos de memória RAM
3	6	3	21	1
4	8	4	124	1
5	10	5	635	3
6	12	6	2466	6

3.3.2 Implementação de um somador RNS

Uma forma alternativa ao uso de tabelamentos para se implementar operações de adição em RNS foi apresentada por Dugdale [78], que, ao invés de se utilizar tabelas, empregava dois somadores e um multiplexador. O segundo experimento adotou o modelo de Dugdale, considerando que a maioria dos RAM FPGAs pode ser configurada para realizar somadores do tipo *ripple carry*, devido à existência de estruturas internas que permitem a rápida propagação de *carry* entre LEs adjacentes. Estes somadores atingem alto desempenho quando o número de bits a ser somado é pequeno [120].

Um somador do tipo Dugdale de módulo m , $(x + y) \bmod m$, requer $x, y < m$, isto é, os valores dos operandos devem se encontrar em módulo m e se baseia nas conhecidas regras de soma modular:

$$\text{resultado} = x + y, \quad \text{se } x + y < m \text{ ou} \quad (3.7)$$

$$= x + y - m, \quad \text{se } x + y \geq m \quad (3.8)$$

Com a utilização de dois somadores de módulo 2^n , $n = \lceil \log_2 m \rceil$, onde n é o número de bits dos operandos, realiza-se as seguintes adições:

$$\text{suma} = x + y \quad (3.9), \text{ e } \quad \text{sumb} = \text{suma} + (2^n - m) \quad (3.10)$$

O valor correto da soma, $(x + y) \bmod m$, é dependente dos *carries* originários das somas *suma* e *sumb*. Dugdale mostrou que se ocorrer algum *carry*, o resultado correto é o valor dado por *sumb*, caso contrário *suma*.

Foi proposta e avaliada [106] uma versão *pipeline* em 3 níveis para um somador RNS, Fig.3.1. A Tabela 3.2 mostra o custo máximo em LEs e a velocidade atingindo pelo somador RNS em função do número de bits do módulo baseado no componente FPGA ALTERA 10K10-3. Se o m tiver seu valor igual $2^n - 1$, o somador pode ser simplificado e ficando com apenas 2 níveis de *pipeline*. Nessa implementação os custo de implementação

em LEs tem um crescimento linear de acordo com aumento do número de bits do módulo, daí todas as implementações apresentam um custo menor do que a apresentada pelo mapeamento direto.

Tabela 3.2: Desempenho e custos aproximados de alguns somadores RNS.

no. bits	Les	no. estágios	Velocidades (MHz)
3	15	3	>100
4	19	3	>100
5	23	3	>100
6	27	3	>100
7	31	3	>100
8	35	3	>100

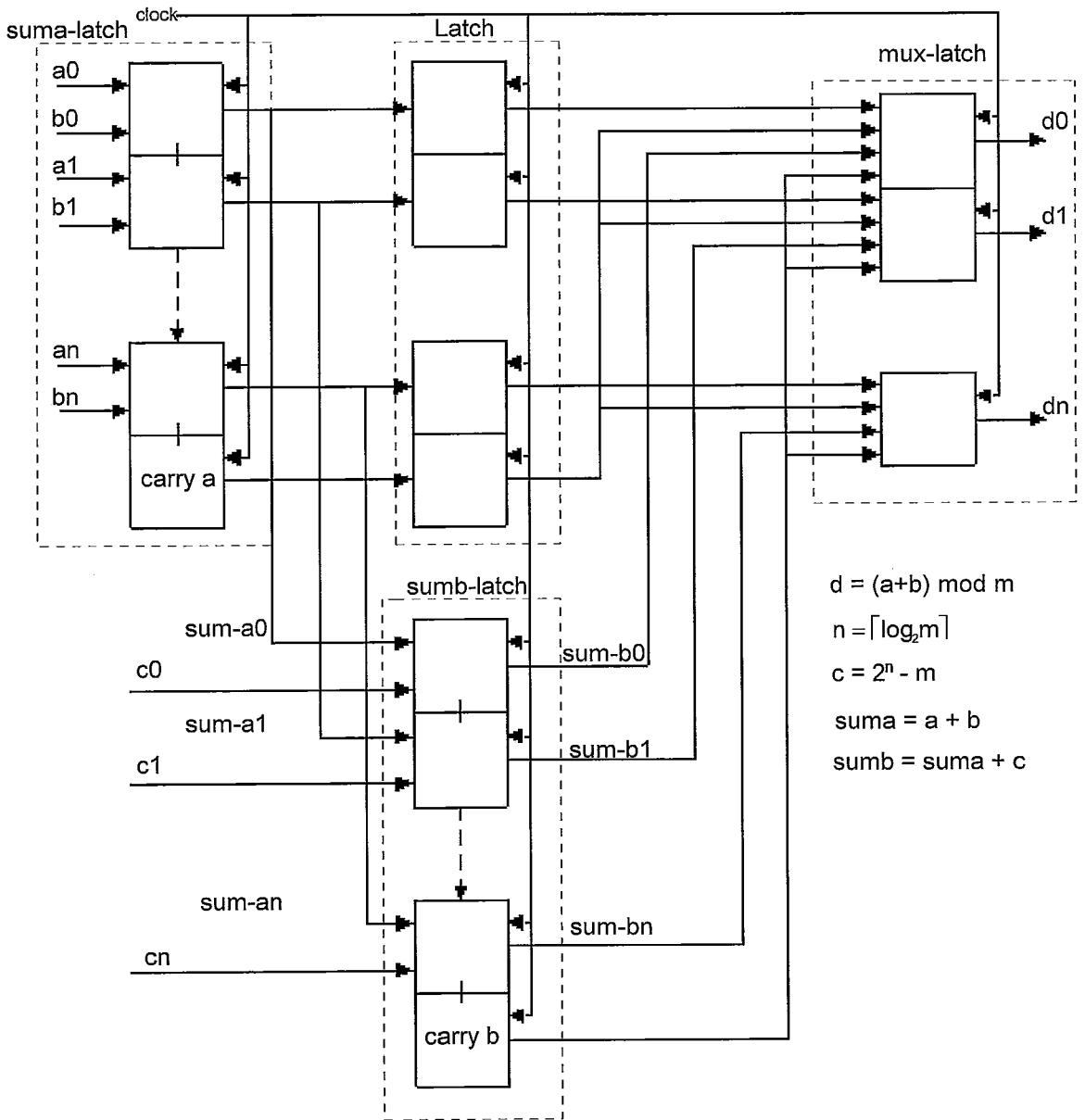


Figura 3.1: Diagrama do somador *pipelined* módulo m .

3.3.3 Implementação de um multiplicador RNS em FPGAs

A utilização do mapeamento direto para implementação de multiplicadores em RNS também é bastante custosa, conforme apresentado na Tabela 3.1. A seguir são apresentadas duas alternativas possíveis:

i) A primeira alternativa é realizada a partir de um multiplicador convencional seguido por um conversor binário-resíduo. Para efeito de comparação de custos em LEs, utilizou-se como multiplicador simples uma versão *pipeline* do multiplicador tipo *ripple carry*. Esses multiplicadores utilizam somadores do tipo *ripple carry* para efetuar as somas dos produtos parciais existentes na multiplicação convencional. Entre os multiplicadores convencionais esse modelo é o que apresenta o menor custo quando mapeado em FPGAs.

Um multiplicador convencional, Figura 3.2, possui duas etapas distintas: geração dos produtos parciais e soma dos produtos parciais.

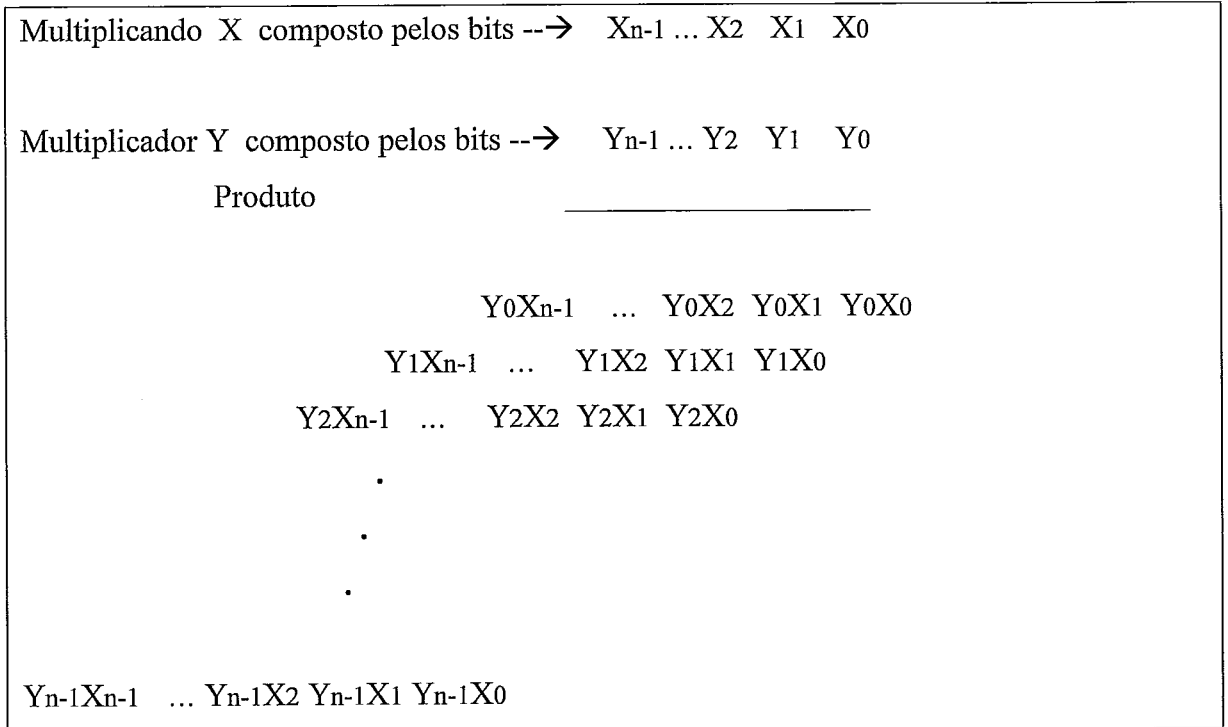


Figura 3.2: Geração dos produtos parciais da multiplicação convencional.

A geração de produtos parciais é feita numa mesma etapa, e a soma dos produtos parciais pode ser feita de forma *pipeline* numa árvore de somadores binários. A Figura 3.3 mostra um diagrama de blocos da implementação de um multiplicador modular de 4 bits utilizando-se apenas LEs com duas partes distintas. A primeira, de fundo cinza, mostra os produtos parciais e suas somas numa árvore de somadores e a segunda parte representa o conversor binário-resíduo.

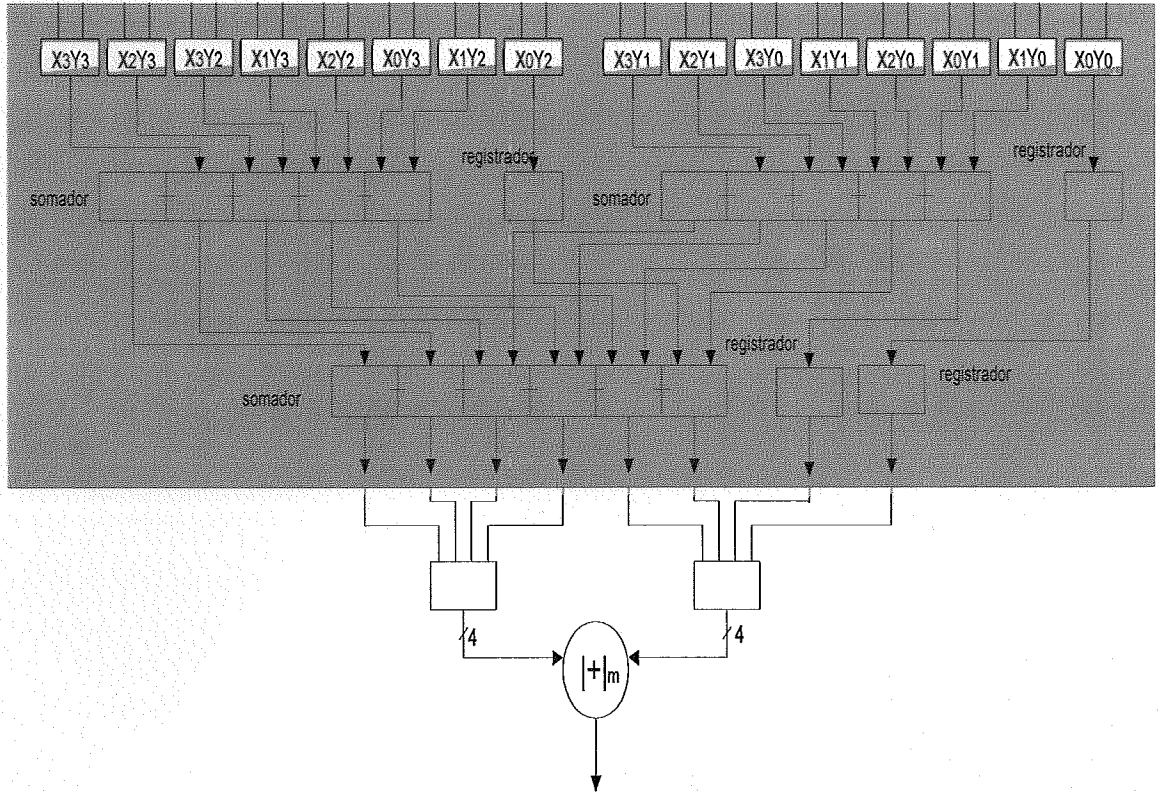


Figura 3.3: Modelo de um multiplicador modular de 4 bits.

Os custos em LEs associados à implementação deste modelo é apresentado na Tabela 3.3. Os valores são aproximados, uma vez que o valor exato deveria levar em consideração o valor do módulo. Conforme será visto na próxima seção, a implementação da conversão binário-resíduo para módulos de mesmo número de bits pode apresentar um número diferente de LEs.

Tabela 3.3: Custo estimado para implementação do modelo de multiplicador RNS

No. de bits	No. les , produto parcial	No. les somador	No. les conversão	Custo total em les
3	9	4	3	16
4	16	20	27	63
5	25	44	40	109
6	36	55	91	162
7	49	72	86	207
8	64	82	100	246

Uma desvantagem desta abordagem é o aumento no número de somadores à medida que o número de bits dos operadores aumenta, pois o tamanho da estrutura e sua forte conexão exigem um grande custo de recursos, tanto de LEs como do sistema de interconexão.

ii) A segunda alternativa implementa o multiplicador RNS baseado no Cálculo de Índices. Esse modelo de multiplicador está fundamentado nas propriedades da teoria de Corpos de Galois [64, 82, 83]. Dado um corpo de Galois $GF(p)$, onde p é um número primo, é possível gerar todos os elementos diferentes de zero a partir de uma raiz primitiva de p .

Seja ρ uma raiz primitiva de p . Os elementos diferentes de zero $\{1, 2, \dots, p-1\}$ são gerados por $\{\rho^i\}_p$, cujos índices são $\{0, 1, \dots, p-2\}$. Esses elementos formam um grupo multiplicativo em **módulo** p , e os índices formam um grupo aditivo em **módulo** $(p-1)$. O multiplicador é obtido pelo uso do isomorfismo entre estes dois grupos. Dados x e y , elementos de $GF(p)$ diferentes de zero, o resultado da multiplicação x por y , **módulo** p , é o

elemento do grupo multiplicativo cujo índice é obtido pela soma modular, em $p-1$, dos índices dos operandos:

$$|x \cdot y|_p = \rho^{|ix + iy|_{p-1}} \quad (3.11)$$

Portanto, nessa abordagem são necessárias 3 etapas para a realização da multiplicação: (i) descobrir os índices dos operandos; (ii) adicioná-los em **módulo** $p-1$; e (iii) descobrir o resultado da operação através do índice do resultado. A implementação dos multiplicadores baseados no Cálculo de Índices foi feita de acordo com o diagrama da Figura 3.4.

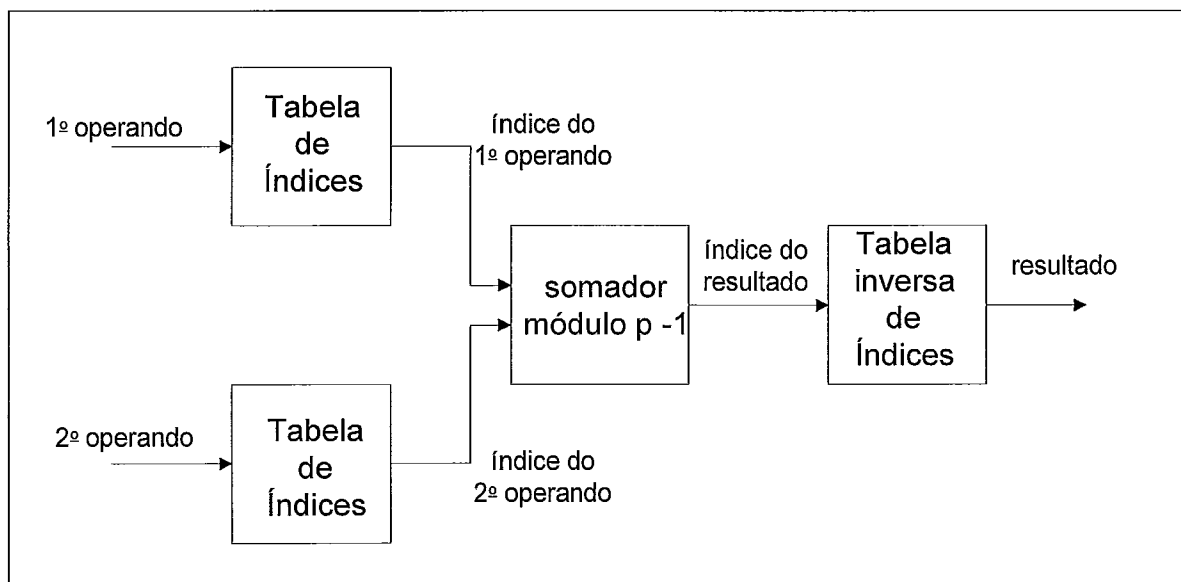


Figura 3.4: Diagrama de blocos do multiplicador baseado no cálculo de índices.

Um circuito adicional deve estar associado ao multiplicador da Figura 3.4, para detectar operandos de valor zero e para, neste caso, corrigir o valor do produto. Existem várias formas para se implementar o detetor de zero em *hardware*, cujo custo é muito pequeno comparado ao do multiplicador.

A seguir são apresentados os resultados para o multiplicador *pipeline* baseado no Cálculo de Índices. O somador módulo $p-1$, foi implementado da forma descrita na Seção 3.2.2. As pequenas tabelas usadas para realizar a busca dos índices, *Tabela de índices*, e a

operação inversa, *Tabela inversa de índices*, conforme apresentadas no diagrama da Figura 3.4, podem ser implementadas de diversas maneiras empregando diferentes combinações dos recursos do FPGA modelo. Por exemplo, foram investigadas três alternativas para implementação: i) usando somente LEs, ii) usando somente blocos de memória, iii) combinação de LEs e blocos de memória. Para comparação de desempenho foram simuladas algumas dessas possíveis implementações e os resultados são apresentados nas Tabelas 3.4-3.7.

Tabela 3.4: Tabelas de índices e Tabela inversa de índices, mapeados em LEs.

Módulo bits	Les	Velocidade (MHz)	no. estágios
3	24	> 100	5
4	31	> 100	5
5	70	> 100	7
6	162	> 100	9
7	363	> 100	11
8	811	----	-

Tabela 3.5: Tabelas de índices e Tabela inversa de índices mapeadas em 3 blocos de memória.

Módulo Bits	Memória bits:%	Les	Velocidade (MHz)	no. estágios
5	450: 7%	37	87	9
6	1080: 17%	45	87	9
7	2646: 43%	51	82.6	9
8	6000: 97%	58	78.1	9

Tabela 3.6 Tabelas de índices mapeadas em LEs e Tabela inversa de índices mapeada em 1 bloco de memória.

Módulo Bits	LEs	Memória bits, Tabela inversa de índices %	Velocidade (MHz)	no. estágios
5	59	150: 7%	87.7	8
6	121	360: 17%	87.7	9
7	259	882: 43%	82.6	10
8	569	2000: 97%	82	11

Tabela 3.7: Tabelas de índices mapeadas em 2 blocos de memória e Tabela inversa de índices mapeada em LEs.

Módulo Bits	Memória bits, Tabelas de índices	LEs	Velocidade (MHz)	no. estágios
5	300:7%	48	87.7	8
6	720: 17%	82	87.7	9
7	1764: 43%	155	86.9	10
8	4000: 97%	313	80.6	11

Os resultados mostram que o custo para se implementar um multiplicador RNS para módulos com 3 bits apresenta um menor número de LEs para o modelo multiplicador seguido de conversor binário-resíduo. As demais implementações sugerem um dos modelos baseados no cálculo de índices. Nestes casos, o desempenho medido mostrado nas Tabelas 3.4 à 3.7 revelam que não existe uma implementação superior às demais. Os resultados indicam que multiplicadores RNS para módulos de 4 ou 5 bits devem ser implementados utilizando-se somente LEs. Enquanto que para módulos de 8 bits o uso de blocos de memória é o mais indicado. Soluções híbridas utilizando LEs e blocos de memória são eficientes para módulos de 6 e 7 bits.

O sinal de maior nas Tabelas 3.2 e 3.4, significa que todos os resultados atingem uma frequência um pouco maior do que 100 MHz, onde o maiores desempenhos são atingidos nas implementações cujos resíduos possuem um menor número de bits. Foi empregado o dispositivo AlteraFLEX20K-3 nestas simulações, e o valor de 100MHz é considerado um ótimo resultado para os nossos propósitos.

3.4 CONVERSÃO ENTRE SISTEMAS NUMÉRICOS BINÁRIO E RNS

Embora o RNS ofereça soluções de alto desempenho para aplicações específicas, o trabalho adicional de conversão e inversão de/para o sistema binário convencional é uma potencial desvantagem que deve ser levada em consideração, tanto em termos de custos quanto em termos de tempo de conversão e inversão de acordo com uma dada aplicação. Nesta seção, apresentaremos as regras e custos associados à implementação desses conversores em FPGA.

3.4.1 Conversão Binário Resíduo

A conversão de um número representado no sistema binário convencional pode ser facilmente implementada usando-se a identidade 3.3 da Seção 3.2. Assim, seja X um número inteiro de n bits, com sua representação binária dada por:

$$X = \sum X_j 2^j \quad (3.12a)$$

ou

$$X = X_{n-1} 2^{n-1} + X_{n-2} 2^{n-2} + \dots + X_1 2^1 + X_0 2^0 \quad (3.12b)$$

onde $X_i \in \{0, 1\}$, para todo $i = 0 \dots n-1$.

A obtenção do resíduo de X em relação ao módulos m , isto é, o resultado da operação ($x \bmod m$) pode ser obtida por:

$$|X|_m = |\sum X_j |2^j|_m|_m, \quad (3.13a)$$

ou

$$|X|_m = X_{n-1} \cdot |2^{n-1}|_m + X_{n-2} \cdot |2^{n-2}|_m + \dots + X_t 2^t \dots + X_1 2^1 + X_0 2^0 \quad (3.13b)$$

onde $2^t < m$ e $+_m$ é operação de adição módulo m .

ou

usando tabelas com k-entradas:

$$|X|_m = |X_{n-1} \cdot 2^{n-1} + \dots + X_{n-2} \cdot 2^{n-1-k}|_m + \dots + |X_t 2^{k-1} \dots + X_1 2^1 + X_0 2^0|_m \quad (3.13c)$$

onde $+_m$ é operação de adição módulo m .

A aplicação de (3.13a) no modelo FPGA adotado é equivalente ao mapeamento direto de funções mostrado na Seção 3.3.1, que só será eficiente para os casos onde os módulos possuem um número de bits menor que 4 e o número inteiro a ser convertido possua poucos bits de entrada (4 ou 5 bits). Em outras condições o número de LEs a ser utilizado nesse tipo de solução poderá ser relativamente grande.

Por outro lado, o uso de várias operações de soma *módulo m* (3.13b e 3.13c) também pode encarecer a solução, em número de LEs. Uma outra solução possível para o problema de custo da conversão binário-resíduo é a utilização de um *pipeline* com a redução do número de bits de entrada de cada estágio, isso pode ser realizado utilizando-se sucessivamente operações como adição e pequenos mapeamentos, até que o último estágio seja uma adição modular onde um dos operandos terá $n-1$ bits, e portanto é um resíduo de módulo m , e o outro operando poderá ter n bits tal que seu valor seja um resíduo do módulo m .

Pequenos mapeamentos, na verdade, podem reduzir ou aumentar o número de bits, mas sempre produzirá um resíduo módulo m .

Regras de redução:

i) A soma binária de dois números de n bits resultará em um número que precisará de no máximo $n+1$ bits para ser representado. Neste caso, temos uma redução de $2n$ bits na entrada para $n+1$ bits na saída.

ii) O alinhamento dos bits da adição modular pode ser modificado em função do valor dado por 2^j e as congruências de $2^j / m$ [84, 85, 86]. A Tabela 3.8 mostra os valores de $2^j / m$, onde m são números primos e as entradas são potência de 2 completa até 16 bits. Observando a Tabela 3.8 é possível ver a formação periódica dos valores de $2^j / m$. O menor número de posições entre dois valores distintos j tal que $2^j / m = 1$ é chamado de *período*.

Na Tabela 3.8, pode-se observar também o surgimento da chamada de *meia-periodicidade*, existente somente em alguns módulos, que é atingida quando o valor de $2^j / m$ é igual $m-1$. A meia periodicidade, quando existe, possui um valor igual a metade do período definido para periodicidade. Seja $p(m)$ o valor do período atingido $2^j / m$ e $h(m)$ o valor da meia periodicidade, logo:

$$|2^{k.p(m)+j} / m| = |2^j / m|, \text{ para todo } k \text{ e } j \in \mathbb{N} \text{ (números naturais)}. \quad (3.14)$$

$$|2^{h(m)+j} / m| = |-2^j / m| = |m-2^j / m|, \text{ para } 0 \leq j \leq h(m) - 1. \quad (3.15)$$

A propriedade mostrada a seguir, demonstrada em [85], permite explorar a meia-periodicidade na redução de bits através da redução do peso binário do número.

$$||2^{h(m)+j} / m|_m \times |_m = |2^{h(m)+j} / m| + |2^j / m|_m \times |_m, \quad (3.16)$$

para $0 \leq j \leq h(m) - 1$ e \underline{x} é a inversão binária x , cujos valores possíveis são 0 e 1.

Assim, um número binário antes de ser convertido para resíduo é visto como um somatório de potência binária. No entanto, as parcelas que têm um valor de potência binária que é maior que a meia periodicidade, $2^{h(m)+j}$, podem ser representadas por uma parcela de potência 2^j , e, portanto, de menor peso binário, associada a um peso obtido da inversão binária do peso original mais um valor constante.

Outros alinhamentos são possíveis e em geral têm uma redução de bits menor do que aquelas obtidas a partir da periodicidade/meia-periodicidade.

A seguir são apresentadas algumas implementações de conversão binário-resíduo que ilustram a utilização das regras de redução de bits adotado neste trabalho.

Tabela 3.8: Valor modular para entrada binária completa até 16 bits, com os valores em negrito para (meia) periodicidade.

Número	₅	₇	₁₁	₁₃	₁₇	₁₉	₂₃	₂₉	₃₁	₃₇	₄₁	₄₃	₄₇	₅₃	₅₉	₆₁
$2^0 = 1$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$2^1 = 2$	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
$2^2 = 4$	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
$2^3 = 8$	3	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8
$2^4 = 16$	1	2	5	3	16	16	16	16	16	16	16	16	16	16	16	16
$2^5 = 32$	2	4	10	6	15	13	9	3	1	32	32	32	32	32	32	32
$2^6 = 64$	4	1	9	12	13	7	18	6	2	27	23	21	17	11	5	3
$2^7 = 128$	3	2	7	11	9	14	13	12	4	17	5	42	34	22	10	6
$2^8 = 256$	1	4	3	9	1	9	3	24	8	34	10	41	21	44	20	12
$2^9 = 512$	2	1	6	5	2	18	6	19	16	31	20	39	42	35	40	24
$2^{10} = 1024$	4	2	1	10	4	17	12	9	1	25	40	35	37	17	21	48
$2^{11} = 2048$	3	4	2	7	8	15	1	18	2	13	39	27	27	34	42	35
$2^{12} = 4096$	1	1	4	1	16	11	2	7	4	26	37	11	7	15	25	9
$2^{13} = 8192$	2	2	8	2	15	3	4	14	8	15	33	22	14	30	50	18
$2^{14} = 16384$	4	4	5	4	13	6	8	28	16	30	25	1	28	7	41	36
$2^{15} = 32768$	3	1	10	8	9	12	16	27	1	23	9	2	9	14	23	11
$2^{18} = 262144$										36						
$2^{23} = 8388608$													1			
$2^{26} = 67108864$														52		
$2^{29} = 536870912$															58	
$2^{30} = 1073741824$																60

Exemplo 1:

A Figura 3.5 representa a implementação *pipeline* adotada para realizar a conversão de um número inteiro a de 12 bits, $a = a_{11}2^{11} + a_{10}2^{10} + \dots + a_12^1 + a_02^0$, para seu respectivo resíduo em relação ao módulo de valor 7. Utilizando-se a Tabela 3.8, que para o módulo 7, um número de 12 bits pode ser interpretado com a soma de 4 números de 3 bits. Assim, inicialmente são efetuadas duas somas de números com 3 bits que resulta em dois números de 4 bits. A seguir são aplicados dois pequenos mapeamentos, bloco f na figura, de 4 entradas nos respectivos resultados da soma produzindo números de 3 bits que são resíduo em *módulo 7*. Novamente, faz-se uma simples soma binária obtendo-se novamente um número de 4 bits e por fim um último mapeamento que produz o resíduo. Nesta implementação não houve a necessidade de se utilizar a soma modular, uma vez que são necessários apenas 3 bits para representar o resíduo, e assim todo mapeamento de 4 entradas gera uma saída com apenas 3 bits.

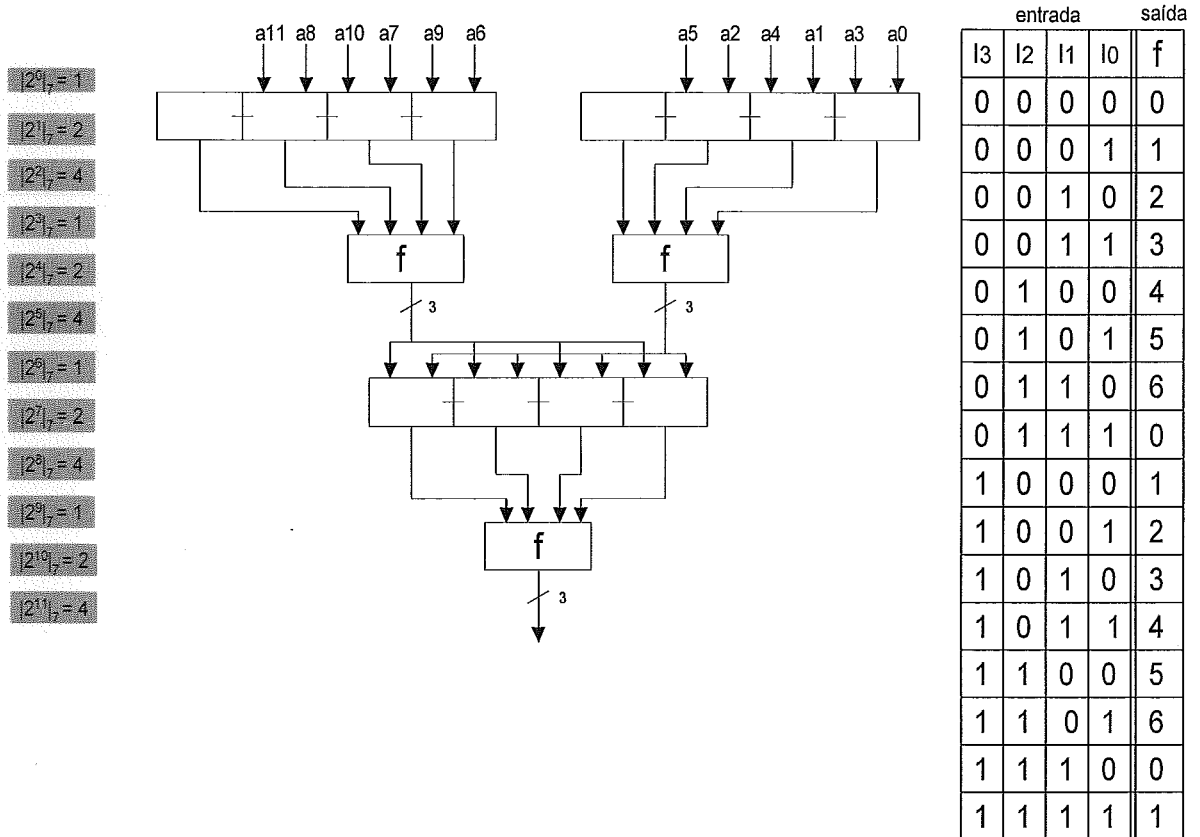


Figura 3.5: Modelo para Conversão binário Resíduo, módulo 7, para números de 12 bits.

Exemplo 2:

Consultando a Tabela 3.8, verifica-se que os módulos de 6 bits, 37,47,53,59,61 não podem se beneficiar das propriedades de periodicidade ou meia periodicidade para entradas com 14 bits. A Figura 3.6 representa um modelo genérico de um conversor binário-resíduo que pode ser aplicado para a esses módulos, para o caso de uma entrada com 14 bits. Diferenciando-os pelos apropriados mapeamentos fa , fb e fc , e a soma modular correspondente a cada módulo em questão. Neste exemplo, o número a ser convertido possui 14 bits, $a = a_{13}2^{13} + a_{12}2^{12} + \dots + a_12^1 + a_02^0$. Inicialmente, particiona-se o número em 3 partes de forma a facilitar o mapeamento. Os mapeamentos fa e fb produzem números de 6 bits. Em seguida, utiliza-se uma árvore de somadores para somar 3 números de 6 bits. O resultado é um número de 8 bits que será particionado em dois números, cujos 4 bits mais significativos são entradas para um pequeno mapeamento, fc . Ao final é feita uma soma modular que resultará no resíduo procurado.

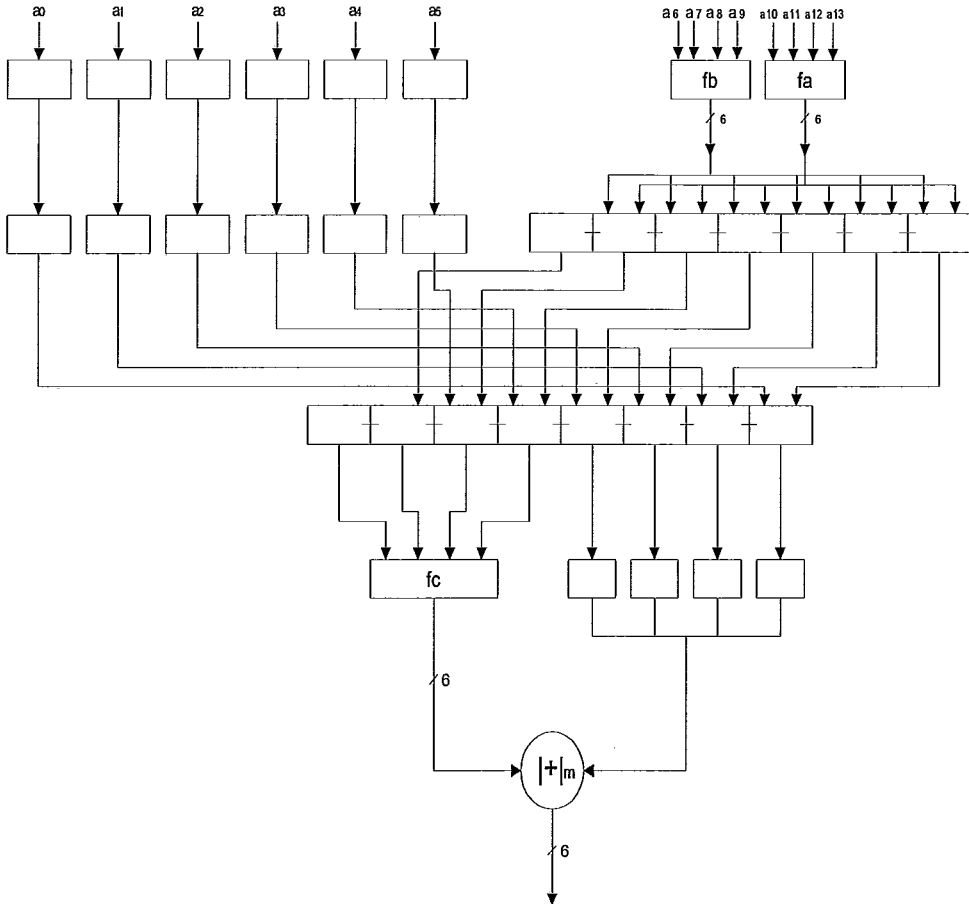


Figura 3.6: Modelo para Conversão binário Resíduo para módulo de 6 bits, 37,47,53,59,61 para uma entrada de 14 bits.

Exemplo 3:

O Conversor mostrado na Figura 3.7, beneficia-se tanto da periodicidade como da meia periodicidade para redução eficiente de bits para realização da conversão binário-resíduo. A primeira soma envolve os 16 bits do número a ser convertido em resíduos módulo 17. Usando propriedades de periodicidade e de congruência, a entrada pode ser vista como dois números de 8 bits. Nos 8 primeiros bits resultantes da primeira soma aplica-se a propriedade de meia-periodicidade para efetuar uma soma de 4 bits. As potências de $|2^{h(m)+j}|_{17}$, que surgem devido ao uso da propriedade meia-periodicidade são mapeadas na função f , juntamente com 2 bits mais significativos dos resultados parciais. E por último, faz-se a soma modular que irá fornecer o resíduo em módulo 17.

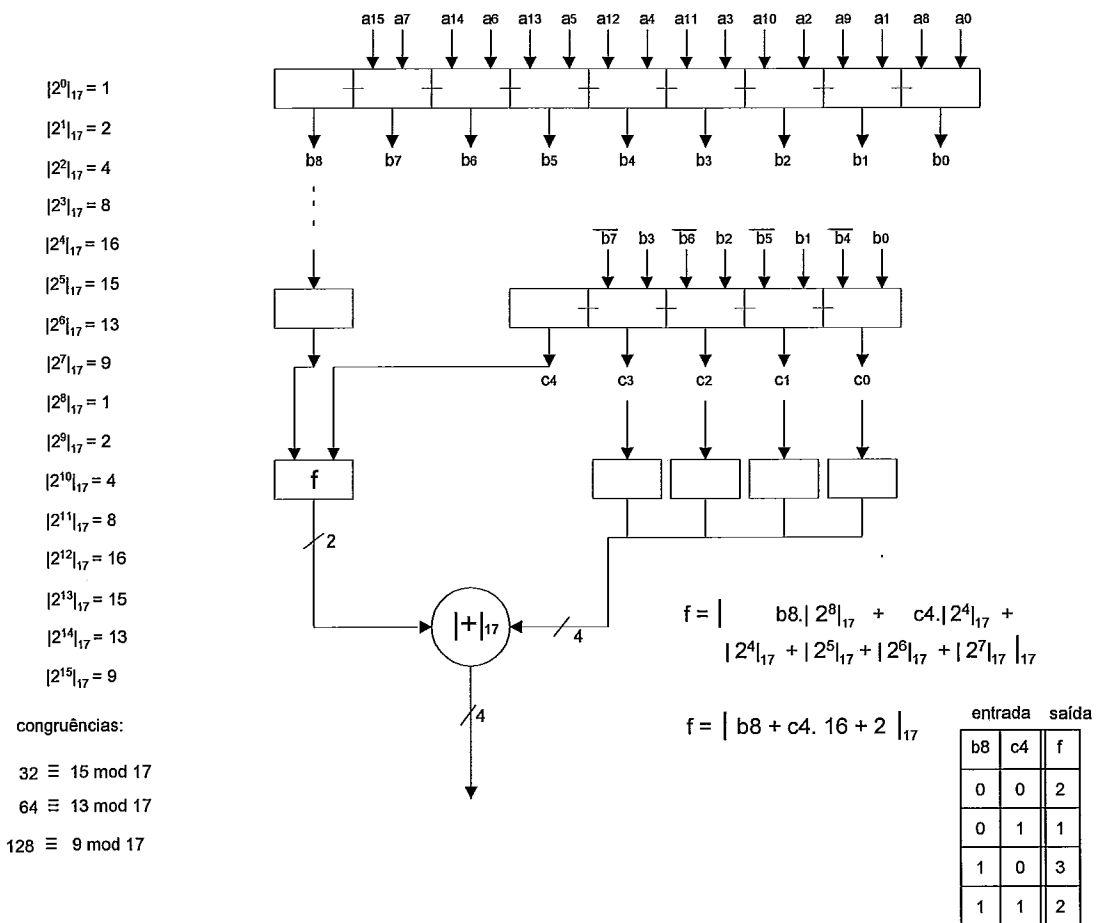


Figura 3.7: Modelo para Conversão binário Resíduo, módulo 17, para 16 bits de entrada.

Os custos associados a conversão binário Resíduo para números inteiros de 12, 14 e 16 bits de largura encontram-se na Tabela 3.9.

Tabela 3.9: Custo em LEs para conversão binário resíduo.

Módulo	Largura com 12 bits	Largura com 14 bits	Largura com 16 bits
5	21	31	40
7	21	23	40
11	35	43	48
13	33	36	46
17	44	45	46
19	49	49	52
23	52	52	65
29	62	64	65
31	29	44	46
37	72	76	82
41	58	58	58
43	38	55	58
47	72	76	82
53	72	76	82
59	72	76	82
61	72	76	82

3.4.2 Inversão Resíduo-Binário

Os principais algoritmos utilizados para realizar a inversão resíduo para binário, são baseados nos métodos *Mixed Radix Conversion* (MRC) e o *Teorema do Resto Chinês* (CRT). Na literatura encontram-se várias implementações dos dois métodos [64, 87-101]. Nesse trabalho adotou-se o método CRT que é de fácil paralelização e possui baixa complexidade.

Dado um número x cuja representação em RNS é dado pelo resíduos $\{r_1, \dots, r_N\}$, sua representação binária usando CRT é obtida através da implementação da seguinte fórmula:

$$X = \left[\sum \hat{m}_j \left| \frac{r_j}{m_j} \right|_{m_j} \right]_M \quad (3.17)$$

onde:

$$M = \prod_{j=1}^N m_j \quad , \text{ é a faixa dinâmica;}$$

$$\hat{m}_j = \frac{M}{m_j} \quad ;$$

$$\left| \frac{1}{\hat{m}_j} \right|_{m_j} \quad , \text{ multiplicativo inverso de } \hat{m}_j \text{ no} \\ \text{modulo } j.$$

Observando-se a fórmula (3.17) é imediato estabelecer o mapeamento em tabelas para os operandos que formam o somatório, cujo endereçamento é pequeno, pois seus valores estão entre 0 e $m_j - 1$, fórmula (3.18). Os mapeamentos correspondem a valores múltiplos de seus respectivos \hat{m}_j , logo podem se aproximar do valor da faixa dinâmica M .

$$\left\lfloor \frac{r_j}{\hat{m}_j} \right\rfloor_{m_j} = \left\lfloor r_j \left\lfloor \frac{1}{\hat{m}_j} \right\rfloor_{m_j} \right\rfloor_{m_j} \quad (3.18)$$

Assim, o valor do somatório pode ultrapassar o valor M , e portanto é necessário fazer subtrações com o subtraendo de valor igual a M , até que o resultado seja menor do que M , o que é equivalente a se fazer a operação Módulo M da expressão de CRT.

De acordo com o trabalho de Shenoy e Kumaresan [92, 97], utilizando-se um módulo redundante é possível obter uma expressão nova para o CRT onde a operação mais externa, módulo M , é substituída por uma única subtração, mostrada na expressão (3.19). Entretanto, o módulo redundante deve possuir valor maior do que o número de módulos que formam a base do RNS.

$$X = \sum_{j=1}^n \hat{m}_j \left\lfloor \frac{r_j}{\hat{m}_j} \right\rfloor_{m_j} - r_x M \quad (3.19)$$

onde r_x é obtido por:

$$r_x = \left\lfloor \frac{1}{M} \right\rfloor_{m_{\text{redundante}}} \cdot \left(\left(\sum_{j=1}^n \hat{m}_j \left\lfloor \frac{r_j}{\hat{m}_j} \right\rfloor_{m_j} \right)_{m_{\text{redundante}}} - r_{m_{\text{redundante}}} \right)_{m_{\text{redundante}}} \quad (3.20)$$

A implementação da equação (3.19) segue os mesmos passos descritos para a expressão (3.17). Da equação (3.20), são obtidas as equações (3.21) e (3.22), cujos mapeamentos têm custos em LEs reduzidos, pois essas expressões são resíduos em módulo redundante.

$$\left[\left[\frac{1}{M} \right]_{m_redundante} \bullet \hat{m}_j \left[\frac{r_j}{m_j} \right]_{m_redundante} \right] \quad (3.21)$$

$$\left[\left[\frac{1}{M} \right]_{m_redundante} \bullet r_{m_redundante} \right]_{m_redundante} \quad (3.22)$$

Para a realização das adições da equação (3.20) são necessários somadores modulares para o módulo redundante. Se para o módulo redundante escolhido for um número par que atenda a restrição do método, a implementação desses somadores modulares terão a forma dos somadores binários módulo 2^n , onde n é número de bits para representar o resíduo do módulo redundante.

Uma solução natural para implementar simultaneamente as adições da equação (3.19) está em se utilizar uma combinação de CSA (*carry save adders*) seguidos de um CPA (*carry-propagate adder*). O mapeamento dos operandos da equação (3.19) é muito custoso devido ao número de bits de saída e ao endereçamento genérico para os módulos, cujo resíduos necessitam de um número de bits maior que o número de entradas do FPGA modelo. Entretanto, para esses módulos, aplicando-se o mapeamento em duas etapas é possível reduzir o número de LEs a serem gastos no mapeamento. Note que a equação (3.18) tem entrada e saída com o mesmo número de bits, entretanto, a saída é um número binário, cujos bits têm peso em potência de 2. Logo, podem ser facilmente divididos para em seguida serem combinados em grupos de quatro entradas, obtendo-se então um custo menor do que com o mapeamento direto.

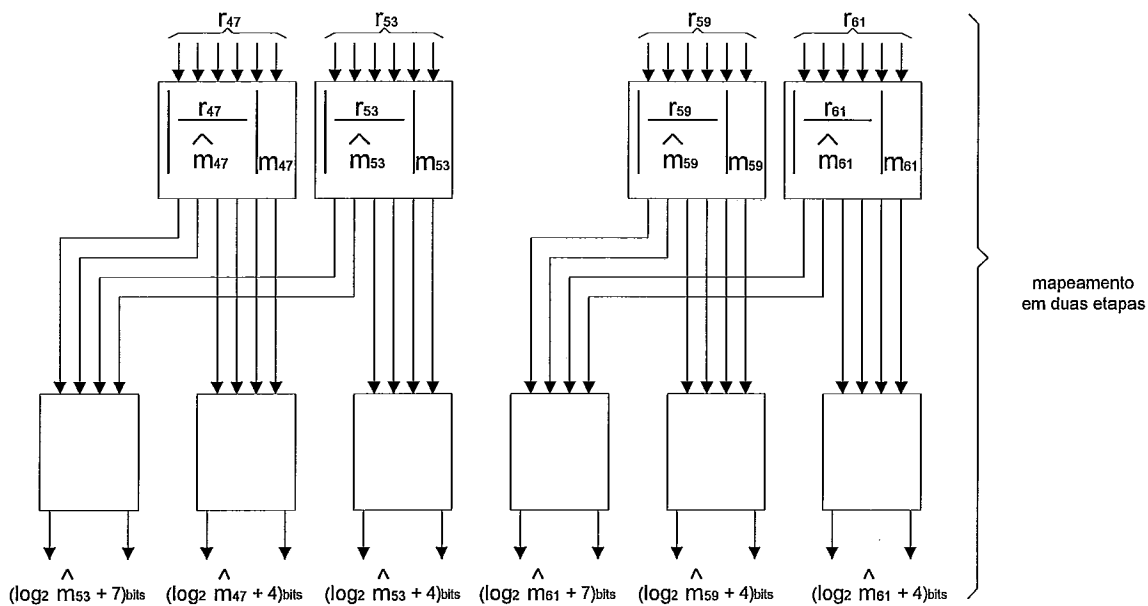
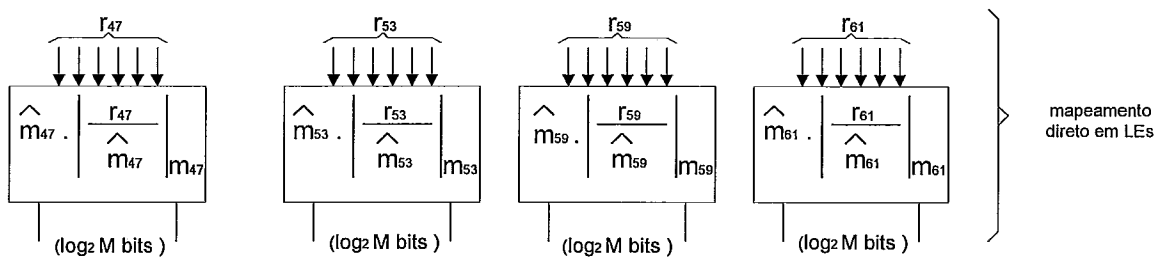


Fig. 3.8: Exemplo de um mapeamento em 1 nível e em 2 níveis para os operandos da expressão CRT.

Para ilustrar a redução de LEs utilizando-se mapeamento em duas etapas é apresentado a seguir um exemplo.

Exemplo 4:

Seja $\{5,7,11,13,17,19,23,29,31,47,53,59,61\}$ a base de um RNS e adicionando-se o módulo 16 para tornar o sistema redundante. A representação de M , faixa dinâmica, requer 59 bits. Para esse RNS tem-se na Tabela 3.9 os custos em LEs necessários para o mapeamento direto dos operandos da fórmula (3.19).

Tabela 3.9: Custo para o mapeamento dos operandos da expressão CRT do exemplo 4.

modulo bits	3	4	5	6	4
modulo	5,7	11, 13	17,19,23,29,31	47,53 59,61	16
no. Les por modulo	59	59	177	413	63

O total de LEs usado somente para mapear os operandos do exemplo 4 é aproximadamente 2836 LEs. Por simplicidade utilizou-se 13 somadores do tipo *ripple carry* com 63 bits para implementar a expressão CRT, ao invés da solução com CSA-CPA. O que resulta num total de aproximadamente 3655 LEs para implementar a fórmula (3.19) de CRT para o exemplo 4. Utilizando-se o mapeamento dos operandos da fórmula (3.19) em dois níveis como sugerido na Figura 3.8, tem-se um total de 2254 LEs, para implementar a expressão CRT, onde 1246 LEs foram utilizados para mapear os operandos e 1008 LEs para mapear os 16 somadores do tipo *ripple carry*.

3.4.4 Discussão:

Embora a conversão resíduo-binária seja muito simples de ser implementada o seu desempenho, utilizando-se somadores do tipo *ripple carry*, é menor do que o atingido pelas operações RNS. A implementação do Exemplo 4, [121], trabalha no máximo por volta de 30 MHz, utilizando-se o componente ALTERA FLEX 10K20-3. Isso se deve a utilização de somadores com longa propagação do *carry*. Portanto, é aconselhável fazer a conversão resíduo-binária em uma ou mais unidades dedicadas que trabalhem sob a forma *pipeline* separadas da unidade que realizará a aplicação em RNS.

CAPÍTULO 4

APLICANDO ARITMÉTICA DE RESÍDUOS AO ALGORITMO DE RETROPROJEÇÃO FILTRADA

Neste capítulo são apresentadas as equações referentes à filtragem das projeções e as operações de retroprojeção para feixes paralelos e para feixes divergentes. A aplicação de aritmética de resíduos nesses algoritmos, a princípio, parece ser imediata, uma vez que a maior parte de suas operações são adições e multiplicações. Entretanto, algumas transformações numéricas devem ser investigadas inicialmente, pois em RNS todas as operações são realizadas como se estivessem utilizando números inteiros. Assim, os coeficientes do filtro e outros números que possuam parte fracionária devem ser primeiro transformados em números inteiros e em seguida transformados em resíduos para serem utilizados nos algoritmos. Estas transformações são realizadas multiplicando-se os valores fracionários por um valor constante, seguido de arredondamento ou truncamento, de forma a transformá-los em números inteiros. No entanto, deve-se considerar algum critério que ajude a estabelecer a quantidade de bits a ser utilizada nas constantes multiplicadoras, de forma que a imagem reconstruída possua qualidade aceitável. Consequentemente, são estabelecidos: (i) a faixa dinâmica de todo o algoritmo com o propósito de se evitar overflow nos resultados, e (ii) o conjunto de módulos que formarão a base RNS para o problema. Sem perda de generalidade, assume-se que o objetivo é reconstruir imagens bidimensionais com 512 x 512 pontos a partir de 100 projeções paralelas e 200 projeções divergentes, utilizando-se o algoritmo de retroprojeção filtrada.

4.1 ALGORITMO PARA FEIXES PARALELOS

O algoritmo de retroprojeção filtrada para feixes paralelos é composto de duas etapas distintas: Filtragem da projeções e a retroprojeção das projeções filtradas, [1-5]. O APÊNDICE C apresenta, resumidamente, a demonstração para se chegar a estas duas etapas a partir das projeções. A seguir, é apresentada uma análise que auxilia a escolha do método para realizar a filtragem em RNS e em seguida é investigada a retroprojeção.

4.1.1 Filtragem

Tradicionalmente, a filtragem das projeções pode ser feita por convolução linear ou utilizando transformadas de Fourier. A implementação da filtragem por convolução linear da resposta impulsiva discreta do filtro e pela seqüências de raios soma de uma projeção tem complexidade da ordem $O(n^2)$, onde n é o número de raios soma de cada projeção, e supondo-se que as seqüências e o filtro possuam o mesmo tamanho n . Na operação de retroprojeção são utilizados somente os n valores centrais da seqüência que representa a projeção filtrada, cujo tamanho é $N = n + n - 1$. Logo, são necessárias cerca de 196.600 operações de somas e produtos, um pouco menor que 262.144 (512×512), para se fazer a filtragem dada uma projeção com 512 raios soma. Utilizou-se o filtro de Ram-Lak (Vide Apêndice C) na operação de filtragem cuja resposta impulsiva discreta possui metade de seus coeficientes com valor igual zero (ver Figura 4.1). Devido a este fato é possível diminuir o número total de operações da filtragem.

$$h(n\tau) = 1/4\tau^2, n = 0$$

o filtro é dado por:

$$h(n, \tau) = 0, n \text{ par}$$

$$h(n, \tau) = -1/(n^2 \pi^2 \tau^2), n \text{ impar}$$

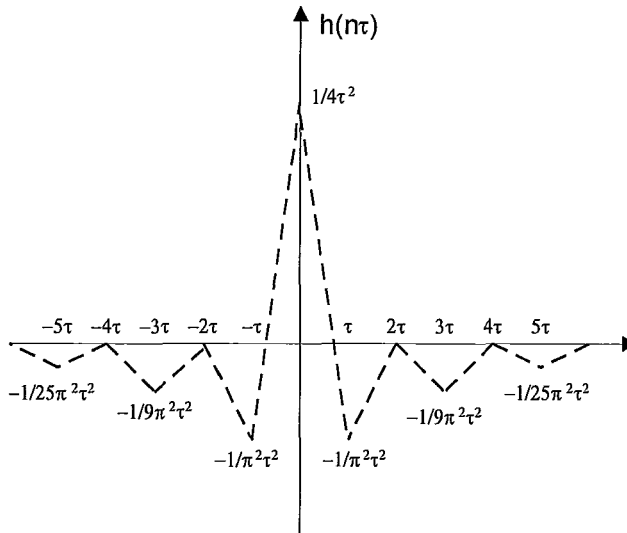


Fig. 4.1: Resposta impulsiva do filtro de Ram-Lak.

Por outro lado, pode-se fazer a filtragem usando a *Transformada rápida de Fourier*, FFT. Neste caso, a filtragem por projeção tem um número de operações da ordem $O(2N\log(N))$ para somas e produtos de números complexos. Observe-se que a soma de números complexos requer 2 operações de números reais e a multiplicação complexa requer 4 multiplicações reais e duas somas reais. Para a realização da filtragem via FFT é necessário fazer a transformada direta de Fourier das projeções, em seguida multiplicar pela resposta em frequência do filtro desejado e por fim realizar a transformada inversa para se obter as projeções filtradas no domínio do espaço. No caso de projeções com 512 pontos, o uso de FFT para filtragem requer um número de operações muito menor do que a filtragem por convolução. Esta solução é bastante atraente quando se utiliza um computador seqüencial ou paralelo, ou dispositivos especializados como DSPs para implementação da filtragem usando aritmética convencional como ponto flutuante ou ponto fixo. Entretanto, o cálculo de FFT em sistemas de resíduos convencional (RNS), ou em RNS quadrático (QRNS) apresenta um crescimento geométrico da faixa dinâmica devido ao aninhamento de multiplicações que ocorre no cálculo subsequentes dos *butterflies* existentes durante a evolução da FFT. Este aumento pode ser controlado pela inserção de operações de escalonamento em trechos do cálculo da FFT [102, 104, 105]. Contudo, as operações de escalonamento, por serem operações não modulares, isto é, necessitam de todos os resíduos de todos os módulos que compõem a base RNS, podem ser bastante custosas quando implementadas em circuitos visando alta velocidade.

Baseando-se em implementações destes algoritmos em máquinas seqüenciais, tem-se que a maior parte do tempo do algoritmo de retroprojeção filtrada é gasto fazendo-se a retroprojeção. Logo, o impacto do uso de FFT é pequeno em relação ao tempo total para a execução do algoritmo. Além disso, a implementação da filtragem por convolução é bastante simples e não requer o uso de números complexos. Assim, neste trabalho a filtragem é feita por convolução. Não será analisada nesta tese a implementação de FFT direta e inversa para realizar a filtragem em RNS ou QRNS, embora QRNS tenha sido adotado com sucesso para trabalhar em sistemas que efetuem operações complexas em resíduos, com a vantagem de reduzir de 4 para 2 o número de operações de multiplicação quando se realiza o produto complexo [104, 105]. A desvantagem do uso de QRNS é a

restrição imposta para formação de sua base, onde se utilizam somente números primos da forma $4k + 1$, pois atendem a congruência quadrática $x^2 \equiv 1 \pmod{p}$.

A utilização de uma base RNS com cerca de 58 bits, próxima à adotada nesse trabalho, envolveria número primos com um maior número de bits, como números primos de 7 bits. O que levaria, baseado nos resultados do Capítulo 3, a um aumento no custo da implementação de multiplicadores, uma vez que os tabelamentos dos índices não é linear com o aumento do número de bits do resíduo. Assim, em termos de operações de multiplicação para uma mesma faixa dinâmica, o uso de QRNS em FPGAs requer um número maior de LEs. Isto indica a necessidade de se investigar até que ponto é vantajoso usar QRNS ao invés de RNS, pois o seu uso diminui o número de operações na implementação da multiplicação mas, no entanto, aumenta o custo para se implementar um multiplicador RNS.

4.1.2 Retroprojeção

A complexidade ao se fazer a operação de retroprojeção, dada uma projeção filtrada, para obter uma imagem representada por uma matriz quadrada $n \times n$ é da ordem $O(n^2)$, pois a imagem é composta por n^2 pontos que receberam alguma contribuição de cada projeção filtrada durante a retroprojeção. Supondo que a imagem a ser reconstruída esteja centrada num sistema coordenado X-Y, a Equação 4.2 abaixo [1,2,4,5], representa a imagem reconstruída por retroprojeções das projeções filtradas do tipo feixes paralelos.

$$f(x, y) \approx \frac{\pi}{k} \sum_{i=1}^k Q \theta_i (x \cos \theta_i + y \sin \theta_i) \quad (4.2)$$

$$s = x \cos \theta_i + y \sin \theta_i \quad (4.3)$$

$$Q \theta_i (s) = Q \theta_i (s_M) (s_{M+1} - s) + Q \theta_i (s_{M+1}) (s - s_M) \quad (4.4)$$

onde x e y variam de $n/2$ a $-n/2$, k é o número de projeções e $Q \theta_i$ é a projeção filtrada interpolada para a posição s .

O número de operações aritméticas para se obter cada pixel da imagem, considerando-se todas as retroprojeções, é dado pelo produto do número de projeções vezes a soma das operações para realizar a determinação de s (Equação 4.3), interpolação (Equação 4.4) e a adição com a imagem parcialmente reconstruída, ou seja:

i) A determinação de s (4.3) para um ponto da imagem numa dada projeção requer dois produtos e uma soma, supondo que as duas funções transcendentais, $sen(\theta_i)$ e $cos(\theta_i)$, podem ser tabeladas pois os valores de θ_i são conhecidos a priori.

ii) O número de operações para se fazer a interpolação linear (4.4), deveria corresponder a dois produtos e 3 somas/subtrações, entretanto os valores de $S_M = \lceil s \rceil$ e $S_{M+1} = \lfloor s \rfloor$ estão separados por uma unidade, o que simplifica sua implementação, como pode ser visto nas Equações 4.5 a 4.8, sendo necessários apenas um produto e três operações de soma/subtração.

$$s_{M+1} = s_M + 1 \quad (4.5)$$

$$Q \theta_i(s) = Q \theta_i(s_M)(s_M + 1 - s) + Q \theta_i(s_{M+1})(s - s_M) \quad (4.6)$$

$$Q \theta_i(s) = Q \theta_i(s_M) + Q \theta_i(s_M)(s_M - s) + Q \theta_i(s_{M+1})(s - s_M) \quad (4.7)$$

$$Q \theta_i(s) = Q \theta_i(s_M) + (Q \theta_i(s_{M+1}) - Q \theta_i(s_M))(s - s_M) \quad (4.8)$$

A implementação do algoritmo pode ser feita de forma a reduzir o número de multiplicações no cálculo de s através do uso de recorrência [12-17], pois o valor de s para dois pontos consecutivos da imagem numa mesma linha varia de $\cos(\theta_i)$, e de dois pontos consecutivos da imagem numa mesma coluna varia de $\sin(\theta_i)$, dada uma mesma projeção. O pseudo-código abaixo, Fig. 4.3, ilustra a aplicação de recorrência na obtenção do valor de s utilizado na retroprojeção de feixes paralelos. Repare-se que é necessária apenas uma operação de multiplicação por projeção.

```

for ( ang = 0 ; ang < 180 ; ang = ang + delta_ang)
{
    Cos = cosine(ang) ;
    Sin = sine(ang) ;
    N0 = -256.( Cos + Sin );
    for ( y = 0 ; y < 512; y++)
    {
        N0 = N0 + Sin ;
        Num = N0 ;
        for ( x = 0 ; x < 512 ; x++)
        {
            S = Num + Cos;
        }
    }
}

```

Fig. 4.3: Trecho em pseudo código para o algoritmo de retroprojeção para feixes paralelos.

4.1.3 Determinação da faixa dinâmica

Um sistema RNS é um sistema finito de números inteiros, onde a determinação da faixa dinâmica é necessária para que a solução seja única, isto é, não ocorra *overflow* durante a realização das operações aritméticas.

A faixa dinâmica necessária para se implementar o algoritmo de retroprojeção filtrada é função do tamanho da imagem, do número de projeções e da qualidade e precisão que se deseja na imagem reconstruída. Por questão de simplicidade, iniciou-se o trabalho de estimativa da faixa dinâmica para o caso de projeções com feixes paralelos. Neste caso, é suposto que a imagem em questão se encontra numa matriz quadrada de 512 x 512 pontos e que são realizadas cerca de 100 projeções. Numa abordagem muito próxima de casos práticos, é suposto que os raios soma foram digitalizados por conversores analógico-digital de 14 ou 16 bits.

A Equação (4.5) será utilizada para a determinação do limite superior da faixa dinâmica quando se realiza a filtragem utilizando-se a convolução linear, ou filtro FIR [103].

$$M - 1 \geq \max |x| * 2 \sum_{k=0}^{N-1} |h_k| \quad (4.5)$$

Onde valor de $\max |x|$ é o valor máximo que um raio soma de uma dada projeção pode possui. Logo, para o cálculo da faixa dinâmica, $\max |x|$ contribui com 14 bits ou 16 bits, devido ao sistema de aquisição de dados. Os valores $|h_k|$ para o filtro Ram-Lak são valores fracionários e menores que 1, e portanto impróprios para serem diretamente utilizados em RNS. Neste momento surge a primeira questão: qual é o valor constante pela qual se deve multiplicar os coeficientes de $|h_k|$, e em seguida arredondá-los para número inteiros, para que a filtragem possa ser realizada em RNS, e atenda ainda a algum critério de qualidade da imagem. Outra incógnita surge durante a aplicação da interpolação linear, pois o valor de s (Equação 4.3) possui uma parte fracionária, fazendo com que o valor de $(S-S_M)$ seja um valor fracionário e menor que 1. Portanto, neste caso também existe a necessidade de transformar frações em números inteiros, surgindo a mesma questão sobre quantos bits devem ser utilizados nesta conversão de fração para inteiro. Estas duas incógnitas estão diretamente relacionadas com a precisão da imagem a ser reconstruída, e consequentemente afetam sua qualidade.

Os quatro parâmetros listados abaixo estão relacionados a qualidade de uma imagem tomográfica quanto aos aspectos físicos do problema [2, 7]:

(i) A resolução espacial num conjunto unidimensional de detetores, como CCDs, está associada ao teorema da amostragem e portanto limitada pela frequência de Nyquist [1,2]. A amostragem é feita segundo o espaçamento entre detetores, assim, a maior frequência espacial permitida é limitada por $1/(2\Delta d)$, onde Δd é a distância entre dois detetores consecutivos. Entretanto, na prática os detetores têm largura de tamanho diferente de zero, o que equivale a agir como um filtro passa baixa, limitando ainda mais a resolução espacial.

(ii) A resolução em contraste descreve a capacidade do sistema em diferenciar duas regiões de coeficiente de atenuação levemente diferentes, e está diretamente limitada pelo ruído eletrônico e de natureza randômica.

(iii) O ruído de natureza randômica está associado a várias etapas da irradiação e aquisição dos raios X. Essas etapas incluem o processo de geração de fótons, espalhamento ao longo de sua trajetória, sua interação com a matéria e a quantidade de fótons que é absorvida no detector. Assim, duas medidas feitas de forma idêntica para uma mesma amostra poderão apresentar valores ligeiramente diferentes.

(iv) Diferentemente do ruído, os artefatos são de origem sistemática, seja na aquisição de dados por instabilidade na eletrônica que gera o fluxo de raios X, ou no detector devido a problemas de normalização ou calibração quando mais de um detector é utilizado na aquisição da projeção. Como resultado a imagem reconstruída pode apresentar traços geométricos tais como riscos, estrelas e anéis não existentes na imagem original. Outros tipos de artefatos estão associados a problema de *aliasing* e ao endurecimento do feixe de raios X [2].

Investigou-se o número de bits necessário para cada incógnita surgida na determinação da faixa dinâmica, de modo que a imagem reconstruída em um sistema RNS

apresentasse uma precisão próxima a da imagem reconstruída sob um sistema numérico do tipo ponto-flutuante com precisão simples para o algoritmo de reconstrução de imagem adotado neste trabalho.

Utilizou-se simulações para investigar a resolução dessas incógnitas. Para tal, foi implementado um simulador de projeções para feixes paralelos simples, sem levar em consideração problemas de ordem randômica como a flutuação dos fótons ou problemas de natureza sistemática, e ainda foi suposto que os detetores possuem abertura zero. Isto foi feito com o propósito de se obter a imagem reconstruída mais próxima, numericamente, da imagem original, e portanto, traduz o melhor caso obtido pelo algoritmo de reconstrução em relação precisão, conseqüentemente maior número de bits, para as imagens reconstruídas. Foram também implementadas duas versões do algoritmo de retroprojeção filtrada, uma em aritmética de ponto flutuante e outra em RNS. Utilizou-se o erro médio quadrático (MSE) entre as imagens originais digitalizadas e as imagens reconstruídas para estabelecer-se o número de bits para as incógnitas. Para isto, utilizou-se duas imagens consideradas padrão, para avaliar a qualidade das imagens reconstruídas.

Entre os modelos utilizados em simulações com o propósito de avaliar a resolução em contraste, o “Head Phantom” de Shepp e Logan é um dos mais empregados, tendo como característica a resolução em contraste de 0.5% [1,2]. A imagem para esse modelo é composta por 10 figuras geométricas no plano, elipses e círculos, para as quais são atribuídas densidades. A interseção das figuras implica na soma de suas densidades, fazendo com que a imagem possua 13 regiões de densidades bem definidas conforme pode ser visto na Figura 4.4 (a). O outro modelo usado foi o clássico objeto cilíndrico de densidade homogênea contendo furos de tamanhos variados, tendendo o menor furo a ficar próximo da resolução espacial. A figura 4.5 representa a imagem reconstruída para o modelo do cilindro com furos .

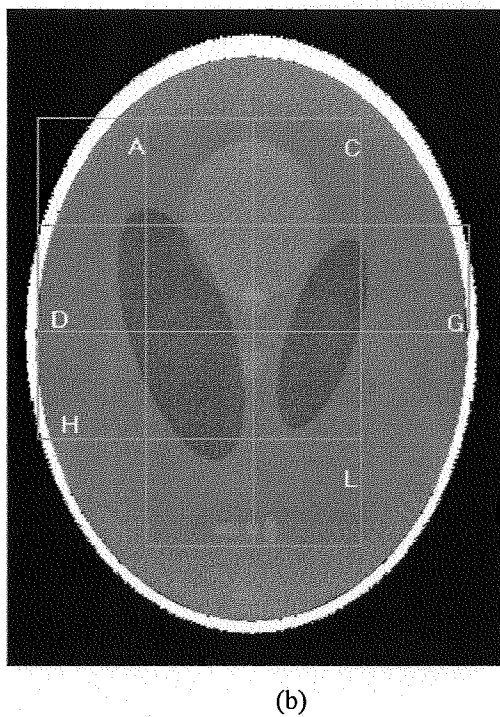
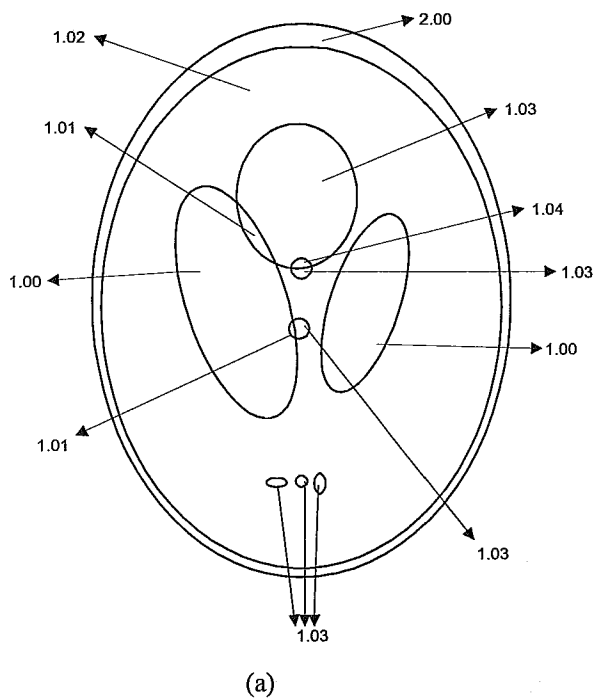


Figura 4.4: Modelo “Head Phantom” de Shepp e Logan , em a) definição das densidades e b) imagem reconstruída com destaque para regiões onde foram feitas análise de MSE.

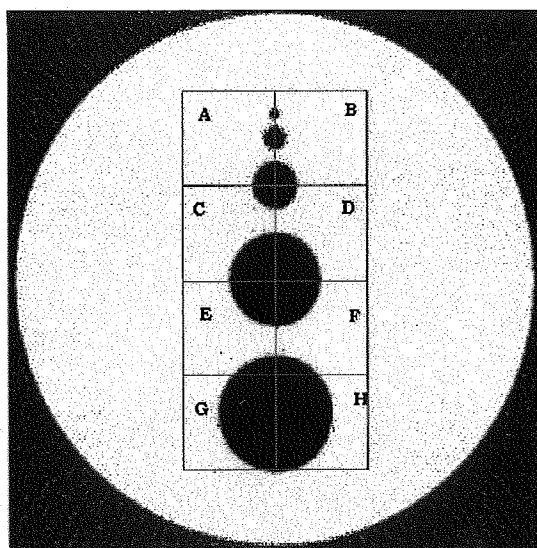


Fig. 4.5: Imagem reconstruída para o modelo “cilindro com furos” com destaque para regiões onde foram feitas análise de MSE.

4.1.4 Análise dos resultados:

Os resultados das simulações, Tabelas 4.1 e 4.2, indicam que a filtragem tem um peso significativo na qualidade da imagem requerendo cerca de 22 bits no processo de transformar os coeficientes do filtro em números inteiros. Um outro resultado importante é que o aumento do número de bits melhora a imagem até um certo limite, não se observando mais melhorias com o aumento de seu valor. O uso do MSE para análise de toda a imagem pode não revelar as diferenças de qualidade entre as diversas simulações. Por exemplo, no modelo de Shepp Logan, o uso do MSE para regiões internas do modelo permitiu estabelecer uma melhor estimativa dos parâmetros de multiplicação nas imagens, indicando cerca de 8 a 10 bits para interpolação e 22 bits para a filtragem.

Repare-se que estas Tabelas apresentam o cálculo do MSE para algumas regiões da figuras modelos, Figuras 4.4 (b) e 4.5. As diversas simulações possuem a identificação dos níveis de quantificação utilizados na transformação dos números fracionários em resíduos: Os dois primeiros dígitos indicam quantos bits foram utilizados na digitalização dos raios soma que compõem a projeção; os bits intermediários estão relacionados ao número de bits da constante multiplicativa aplicada ao filtro; os últimos dígitos estão relacionados ao número de bits da constante multiplicativa utilizada na interpolação. Por exemplo, a quantificação indicada por 14-16-10, significa que foram utilizados 14 bits nos raios soma da projeções, 16 bits na filtragem e 10 bits na interpolação. As simulações indicadas por *digit% número* representam simulações que auxiliam o entendimento da tabela, através da atribuição de erros percentuais aos valores da imagem original digitalizadas, ou seja não são imagens reconstruídas. São apenas variações da imagem original onde os valores digitalizado para cada pixel está com um erro fixo. Nestes casos são apresentados os valores de MSEs relativos a erros de 0.25 %, 0.5% e 1%.

Aplicando-se a Fórmula (4.5) aos resultados da simulação, chega-se a um valor que representa o somatório de $|h_k|$ com cerca de 21 bits, e admitindo-se $\max |x|$ representado por 14 bits implica-se no uso de 36 bits para representar a faixa dinâmica de uma projeção filtrada. Com o uso de interpolação utilizando 10 bits tem-se uma retroprojeção

necessitando de 46 bits. Como é utilizado um total de 100 projeções para reconstruir uma imagem, a faixa dinâmica total estimada para uma situação pessimista é cerca de 53 bits.

Essas simulações revelam que o procedimento adotado permite estabelecer uma quantidade de bits, não ideal, mas adequada para se utilizar resíduos no algoritmo de retroprojeção filtrada. Mostrando que é possível ter imagens de qualidade reconstruídas em RNS próximas as das imagens reconstruídas pela utilização de aritmética de ponto flutuante, utilizando-se um número finito e não muito elevado de bits na faixa dinâmica. Na prática, estima-se que os valores empregados para as constantes multiplicadoras, que estão relacionado ao tamanho da faixa dinâmica, poderiam ser menores. Com isso, a diminuição da precisão, e conseqüentemente da faixa dinâmica, pode ser feita sem muito prejuízo, uma vez que ela estaria mascarada devido as limitações introduzidas pelo sistema de aquisição de imagens como ruídos e flutuações estatísticas inerentes a geração e detecção de raios X.

Tabela 4.1: Avaliação em MSE do modelo *HEAD Phantom* de Shepp e Logan para imagens reconstruídas em RNS a partir da atribuições de valores as constantes multiplicativas, e imagem reconstruída em aritmética de ponto flutuante na reconstrução para feixes paralelos.

Níveis de quantificação	MSE total	MSE Região A	MSE Região B	MSE Região C	MSE Região D	MSE Região E	MSE Região F	MSE Região G	MSE Região H	MSE Região I	MSE Região J	MSE Região K	MSE Região L
Ponto- flut	0.133022	0.000010	0.000010	0.000722	0.000001	0.000010	0.001306	0.000001	0.000004	0.011687	0.008732	0.000009	0.000009
14-16-10	0.182668	0.001739	0.001760	0.005587	0.000342	0.000369	0.006471	0.000378	0.00385	0.014594	0.025842	0.001461	0.001471
14-18-8	0.118449	0.000038	0.000039	0.002586	0.000020	0.000026	0.003967	0.000017	0.000020	0.010888	0.018936	0.000037	0.000040
14-20-8	0.120887	0.000054	0.000055	0.002506	0.000086	0.000091	0.003868	0.000087	0.000095	0.010851	0.018998	0.000058	0.000061
14-20-10	0.121003	0.000022	0.000022	0.002442	0.000041	0.000047	0.003788	0.000042	0.000047	0.010848	0.018879	0.000024	0.000026
14-22-10	0.122115	0.000008	0.000008	0.002402	0.000003	0.000010	0.003733	0.000004	0.000006	0.010843	0.018944	0.000007	0.000007
14-25-10	0.122180	0.000008	0.000007	0.002402	0.000004	0.000011	0.003733	0.000004	0.000007	0.010843	0.018945	0.000007	0.000007
Digi%0.25	-	0.000006	0.000007	0.000007	0.000006	0.000006	0.000007	0.000006	0.000006	0.000007	0.000007	0.000007	0.000007
Digi%0.50	-	0.000026	0.000026	0.000028	0.000025	0.000026	0.000029	0.000026	0.000026	0.000029	0.000030	0.000026	0.000026
Digi%0.10	-	0.000104	0.000104	0.000111	0.000101	0.000102	0.000114	0.000102	0.000103	0.000115	0.000119	0.000105	0.000105

Tabela 4.2: Avaliação em MSE do modelo “cilíndrico com furos”, para imagens reconstruídas em RNS a partir da atribuições de valores as constantes multiplicativas, e imagem reconstruída em aritmética de ponto flutuante na reconstrução para feixes paralelos.

Níveis de quantificação	MSE total	MSE Região A	MSE Região B	MSE Região C	MSE Região D	MSE Região E	MSE Região F	MSE Região G	MSE Região H
Ponto-flut.	0.0 64504	0.003467	0.003247	0.002528	0.004558	0.006111	0.007459	0.008129	0.006504
16-15-10	0.287718	0.074219	0.074110	0.082396	0.082450	0.096118	0.098981	0.094075	0.088883
16-20-10	0.071831	0.005574	0.005145	0.005435	0.006208	0.008298	0.009873	0.007782	0.006824
16-30-12	0.072202	0.005582	0.005163	0.005405	0.006171	0.008270	0.009873	0.007922	0.006890

4.2 ALGORITMO PARA FEIXES DIVERGENTES

4.2.1 Filtragem

A filtragem das projeções obtidas por feixes divergentes deve ser precedida da operação de normalização da projeção (Equação 4.9) [2,4]. A normalização da projeção é função da distância da fonte ao centro de rotação do sistema (D) e da posição do detetor no conjunto unidimensional de detetores. Portanto, fora este produto, tudo que foi utilizado para investigar a filtragem para feixes paralelos serve para feixes divergentes, e assim o método utilizado para filtragem para feixes divergentes também será a convolução linear. Numericamente, o valor de normalização é da ordem de décimos da unidade de distância utilizada no valor de D. Admitindo-se que a multiplicação pode ser efetuada de forma que o valor resultante se mantenha entre 14 ou 16 bits, igual a resolução do sistema de conversão analógico-digital, pode-se seguir o mesmo raciocínio utilizado para determinar a faixa dinâmica da filtragem em projeções paralelas.

$$Q(s_m, \theta_i) = [1/2h(s_m)] \otimes [r(s_m, \theta_i) \cdot \frac{D}{\sqrt{D^2 + s_m^2}}] \quad (4.9)$$

4.2.2 Retroprojeção

A operação de retroprojeção para projeções obtidas a partir de feixes divergentes tem a complexidade de ordem $O(n^2)$, igual a ordem da retroprojeção para feixes paralelos. No entanto, o número de operações por passo aumenta, assim como a complexidade para implementá-las, pois introduziu-se uma operação de divisão e uma operação de recíproco de um número elevado ao quadrado, como demonstram as Equações 4.10, 4.11, 4.12 e 4.13,

$$f(x, y) \approx \frac{2\pi}{N-1} \sum_{i=0}^{N-1} \frac{1}{U^2(x, y, \theta_i)} \bar{Q}(s', \theta_i) \quad (4.10)$$

$$\overline{Q}(s', \theta_i) = Q(s_m, \theta_i)(s_{m+1} - s'(x, y, \theta_i)) + Q(s_{m+1}, \theta_i)(s'(x, y, \theta_i) - s_m) \quad (4.11)$$

$$s'(x, y, \theta_i) = \frac{(x \cos \theta_i + y \sin \theta_i)}{U(x, y, \theta_i)} \quad (4.12)$$

onde $f(x, y)$ é a imagem a ser reconstruída, $\overline{Q}(s', \theta_i)$ representa a interpolação linear das projeções filtradas $Q(S_m, \theta_i)$ and $Q(S_{m+1}, \theta_i)$, $S_m = \lfloor S' \rfloor$ and $S_{m+1} = \lceil S' \rceil$ e U é definido como

$$U(x, y, \theta_i) = 1 + \frac{(x \sin \theta_i + y \cos \theta_i)}{D} \quad (4.13)$$

Nesta formulação também é assumido que os valores de θ_i são conhecidos a priori e a recursividade pode ser aplicada com o mesmo propósito de se diminuir o número de multiplicações, conforme descrito no pseudo-código abaixo:

```

for ( ang = 0 ; ang < 360 ; ang = ang + delta_ang)
{
    Cos = cossine(ang) ;
    Sin = sine(ang) ;
    N0 = -256.(Cos + Sin);
    D0 = 1 - 256. Cos + 256. Sin ;
                D          D
for ( x = 0 ; x < 512 ; x++)
{
    N0 = N0 + Cos ;
    D0 = D0 + Sin / D ;
    Num = N0 ;
    U = D0 ;
for ( y = 0 ; y < 512 ; y++)
{
    Num = Num + Sin;
    U = U - Cos/D;
    S' = Num/U ;
} } }

```

Fig. 4.6: Trecho em pseudo código para o algoritmo de retroprojeção para feixes divergentes.

A precisão do valor de s' e o tamanho da imagem determinam o número de bits para representar o dividendo e o divisor, respectivamente NUM e U , na formulação acima. Novamente a abordagem por recorrência simplifica o número de multiplicações por projeção, os valores de $\cos(\theta_j)$ e $\sin(\theta_j)$ divididos por D podem ser tabelados e assim somente é necessário computar uma divisão para se obter s' . Repare-se que a mesma formulação fornece o valor de U que deverá ser invertido e elevado ao quadrado para ser empregado no cálculo da retroprojeção para cada ponto na imagem dada uma projeção.

4.2.3 Determinação da faixa dinâmica.

Observando a Equação 4.10, verifica-se que a faixa dinâmica para a retroprojeção de feixes divergentes deve ser um pouco maior do que para feixes paralelos, devido ao fator $1/U^2$ existente a mais em relação a expressão de feixes paralelos. Os valores $1/U^2$ possuem parte fracionária, devendo ser tratados apropriadamente para serem utilizados em resíduos. Assumindo-se que os fatores multiplicativos para a filtragem e interpolação podem ser inicialmente examinados a partir dos valores já obtidos para feixes paralelos, isto é, 22 bits para filtragem e 10 bits para interpolação, buscou-se por simulação o número de bits para o fator multiplicativo aplicado a $1/U^2$. Utilizou-se também o erro médio quadrático (MSE) para comparar as imagens reconstruídas utilizando aritmética de ponto flutuante e aritmética de resíduos com a imagem original digitalizada.

4.2.4 Análise dos resultados:

De fato, como pode ser visto na Tabela 4.3, o número de bits utilizado para o fator multiplicativo aplicado na filtragem para feixes paralelos parece atender também o caso de feixes divergentes. Por extensão não foram testados outros valores para o fator multiplicativo da interpolação, isto é, utilizou-se 10 bits em todas as simulações. Verificando-se os resultados, o uso de 8 bits para o fator multiplicativo aplicado a $1/U^2$, em geral, apresenta bons resultados. Entretanto, o uso de 10 bits em algumas regiões tem uma aproximação igual aos resultados obtidos para ponto-flutuante. Repare-se que a Tabela 4.3

tem a denominação utilizada para expressar os níveis de quantificação é semelhante à utilizada para feixes paralelos, entretanto, utiliza um campo a mais com o número de bits utilizado na constante multiplicativa aplicada a $1/U^2$. A figura 4.6 representa a imagem reconstruída do modelo de Shepp Logan.

Numa aplicação para imagens com 512×512 pontos, admitindo-se valores práticos para distância entre a fonte de irradiação e o centro de rotação, i. e. $D > 512$, tem-se os valores de U no seguinte intervalo (0.2910, 1.707). Logo, os valores de $1/U^2$ se encontram no intervalo (11.809, 0.3431). No uso do fator multiplicativo de 8 bits o número inteiro resultante necessitará de 12 bits para sua representação. Assim, pode-se assumir que a faixa dinâmica para o algoritmo de retroprojeção filtrada para feixes divergentes requer cerca de 12 a 13 bits a mais que a faixa dinâmica estimada para os feixes paralelos, o que dá aproximadamente cerca de 65 bits para uma estimativa muito pessimista.

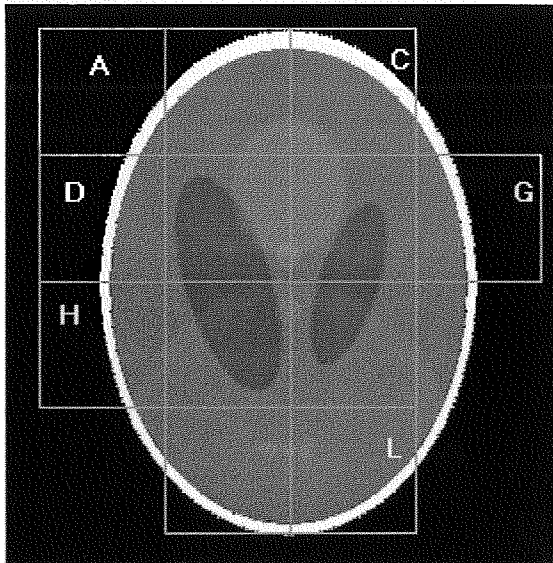


Figura 4.6: Modelo "Head Phantom" de Shepp e Logan reconstruída com destaque para regiões onde foram feitas análise de MSE.

Tabela 4.3: Avaliação em MSE do modelo *HEAD Phantom* de Shepp e Logan para imagens reconstruídas em RNS a partir da atribuições de valores as constantes multiplicativas, e imagem reconstruída em aritmética de ponto flutuante na reconstrução para feixes divergentes.

Níveis de quantificação	MSE total	MSE Região A	MSE Região B	MSE Região C	MSE Região D	MSE Região E	MSE Região F	MSE Região G	MSE Região H	MSE Região I	MSE Região J	MSE Região K	MSE Região L
Ponto-flu	0.092141	0.015560	0.004275	0.009307	0.000003	0.000002	0.003880	0.000004	0.000002	0.037756	0.003379	0.016678	0.044598
14-20-10-10	0.095844	0.017744	0.012761	0.017321	0.000019	0.000019	0.015038	0.000018	0.000018	0.031404	0.010318	0.019464	0.038332
14-22-10-6	0.096011	0.017950	0.013014	0.017417	0.000060	0.000060	0.015165	0.000056	0.000059	0.031456	0.010469	0.019697	0.038455
14-22-10-8	0.096154	0.017779	0.012779	0.017447	0.000006	0.000006	0.015155	0.000006	0.000006	0.031450	0.010468	0.019452	0.038275
14-22-10-10	0.096203	0.017748	0.012730	0.017461	0.000004	0.000003	0.015160	0.000004	0.000003	0.031454	0.010469	0.019401	0.038239
14-25-10-10	0.096224	0.017747	0.012730	0.017456	0.000004	0.000003	0.015156	0.000004	0.000003	0.031452	0.010466	0.019401	0.038241

CAPÍTULO 5

UMA ARQUITETURA RECONFIGURÁVEL PARA RECONSTRUÇÃO DE IMAGENS TOMOGRÁFICAS

Neste Capítulo é apresentada uma proposta de uma arquitetura reconfigurável para tratar a reconstrução de imagens através da implementação do algoritmo de retroprojeção filtrada. São discutidos o funcionamento, custos e desempenhos das unidades que realizam a filtragem e a retroprojeção. Nestas unidades os dados são tratados em aritmética de resíduos. A passagem para o sistema binário convencional, já discutida no Capítulo 3, não é apresentada. São apresentados dois tipos de unidades de retroprojeção, uma para feixes paralelos e outra para feixes divergentes. A título de ilustração, o desempenho esperado para a proposta é comparado ao desempenho do algoritmo implementado em máquinas comerciais baseados em processadores de propósito geral.

5.1 PROPOSTA

O sistema proposto para realizar reconstrução de imagens pode ser visto na Figura 5.1. Nele, o computador hospedeiro pode se comunicar com o sistema de aquisição de projeções ou mesmo controlá-lo. Assim, cada projeção pode ser transferida para a unidade de filtragem imediatamente ao ser adquirida, ou transferida junto com as outras após o término da aquisição. Por simplicidade, é suposto que todas as projeções estarão no computador hospedeiro antes de começarem a funcionar os blocos de filtragem e de retroprojeção. Evitou-se discutir a influência dos parâmetros da aquisição de dados tais como: tempo de integração ou exposição aos raios-X e tempo de leitura do detetor no início do processamento. Estes parâmetros podem ser diferentes para cada experimento, dependendo do material que compõe o objeto a ser examinado e o tipo de energia da fonte de raios X. E assim, determinam diferentes modos e tempos de aquisição que estão relacionados a maneira mais adequada da passagem dos raios soma das projeções para o

computador hospedeiro, e por consequência o encaminhamento das projeções a unidade de filtragem. O computador hospedeiro também tem outras funções:

- i) Estabelecer o início das funções de filtragem e de retroprojeção;
- ii) Fazer a distribuição das projeções;
- ii) Controlar as filas existentes na arquitetura;
- iv) Passar os arquivos de configuração dos FPGAs.

Os valores de $\text{seno}\theta$, $\text{coseno}\theta$, $D0$ e $N0$, associados à cada projeção são conhecidos a priori e fornecidos pelo hospedeiro à medida em que a projeção for filtrada. Nessa proposta, também é atribuído ao hospedeiro a multiplicação da projeção pelo fator $D/\sqrt{(D^2 + Sm^2)}$ antes da filtragem, para o caso das projeções de feixes divergentes.

Dentro desta proposta, este trabalho foi direcionado ao desenvolvimento das unidades de filtragem e retroprojeção, considerando-se suas implementações em FPGA e as operações em aritmética de resíduos. Adotou-se a filosofia de arquitetura reconfigurável com configuração estática, e as unidades trabalham em sob a forma *pipeline*, com isso pretendeu-se extrair o máximo de desempenho dos FPGAs que compõem a arquitetura. O projeto digital foi feito através do uso de uma *linguagem de descrição de hardware*, AHDL [121], fornecido pelo fabricante dos componentes FPGAs usados neste trabalho, o que permitiu explorar ao máximo o desempenho destes componentes, porém restringindo a portabilidade da solução. No uso de configuração estática, a configuração é enviada para os FPGAs antes do início de seu funcionamento e somente após o término da execução é permitido estabelecer outra configuração. Nesse trabalho é necessário carregar em 2 etapas as configurações diferentes para realizar a retroprojeção filtrada. A primeira etapa de configuração fará com que os FPGAs implementem os blocos de filtragem e as unidades de retroprojeção, conforme mostrado no diagrama da Figura 5.1. A segunda etapa é iniciada após a imagem já ter sido reconstruída. Nesta etapa carrega-se a configuração adequada para que os FPGAs funcionem como inversor resíduo-binário. Assim, a arquitetura trabalhará nessa etapa na transformação da imagem reconstruída em resíduos para a imagem reconstruída no sistema binário convencional.

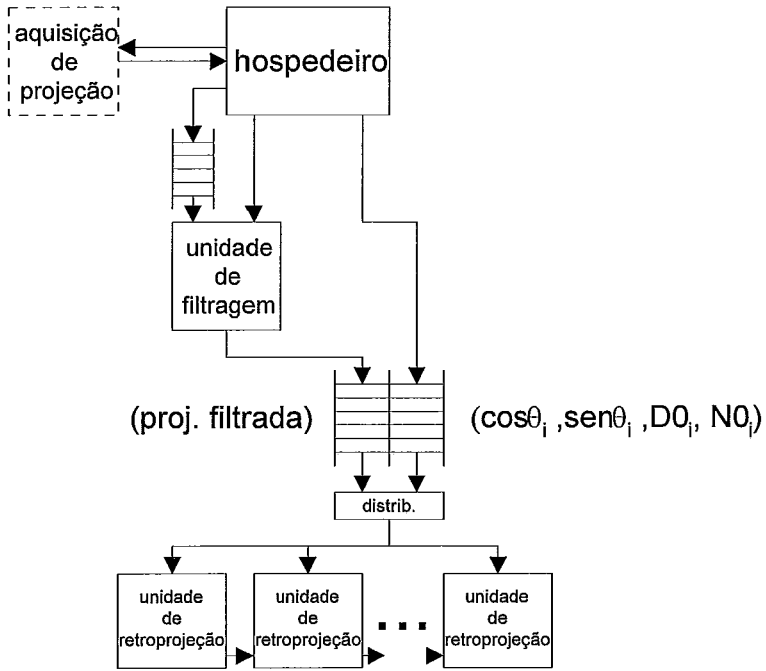


Fig. 5.1: Arquitetura para reconstrução de imagens.

As unidades de retroprojeção podem trabalhar em *pipeline* entre si, como nas arquiteturas baseadas no PE³ [8], onde as operações de retroprojeção são iniciadas simultaneamente em cada unidade de retroprojeção. Desse modo, todas as unidades calculam a retroprojeção para o mesmo ponto da imagem, e ao valor produzido é somado o valor vindo da unidade de retroprojeção anterior do *pipeline*. A soma resultante é passada para o próxima unidade de retroprojeção do *pipeline*. O último elemento do *pipeline* deve atualizar a memória de imagem, somando o valor do ponto contido na memória de imagem com o valor por ele calculado.

É importante, para o funcionamento eficiente da arquitetura com múltiplas unidades de retroprojeção, que o tempo para realizar a filtragem numa projeção seja bem menor do que o tempo para retroprojetá-la, assim é possível manter todas unidades de retroprojeção trabalhando em paralelo.

A seguir são apresentadas as implementações adotadas para unidades de filtragem e de retroprojeção para os casos de feixes paralelos e de feixes divergentes.

5.2 UNIDADE DE FILTRAGEM

Uma das estruturas mais comuns para a realização de convolução linear é a implementação obtida diretamente de sua equação, como ilustrada pela Figura 5.2. Essa implementação é formada por registradores, um multi-somador e multiplicadores em um número igual ao número de coeficientes do filtro. No caso, o filtro Ram-Lak utilizado possui simetria dos coeficientes em relação a um valor central, e quase metade dos coeficientes possui valor zero, reduzindo o custo de implementação direta. Porém, mesmo com a redução do número de multiplicadores, uma implementação direta é bastante custosa, pois o número de coeficientes não nulos é cerca de 255.

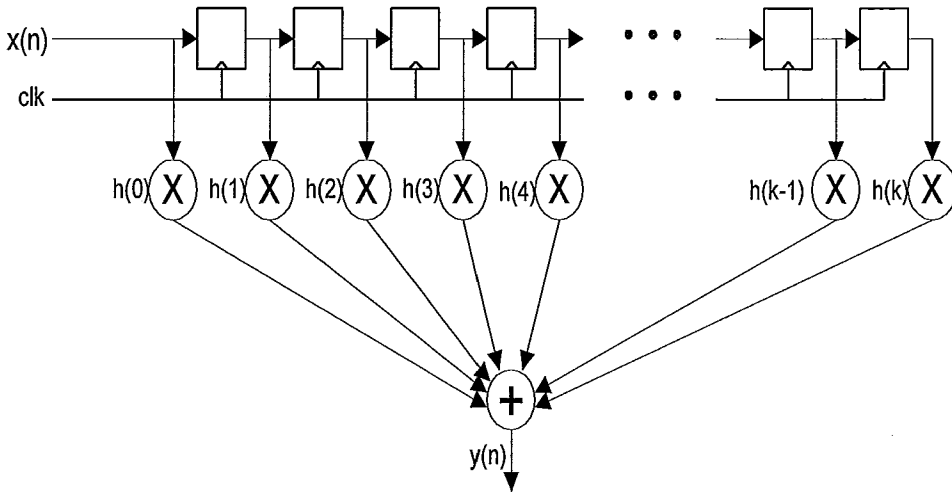


Fig. 5.2: Implementação direta da convolução linear.

Uma alternativa para simplificar a implementação direta é dividir a operação de filtragem em sub-operações, utilizando-se uma estrutura menor, isto é, com um número menor de multiplicações simultâneas. A partir da representação matricial da operação de convolução é imediato estabelecer sua divisão, como visto no exemplo da Figura 5.3. Onde, $X[]$ é o vetor que contém a entrada de dados, isto é, uma projeção a ser filtrada, $H[]$ é a matriz que contém os coeficientes do filtro e $Y[]$ é o vetor onde serão guardados os resultados da operação de filtragem, isto é, a projeção filtrada. Inicialmente, divide-se o número total de multiplicadores de forma a estabelecer o tamanho da nova estrutura. Os coeficientes são divididos e agrupados em grupos relacionados a cada multiplicador da nova estrutura.

A Figura 5.4 representa um diagrama simplificado de um exemplo com uma estrutura reduzida com cinco multiplicadores. Repare-se que cada multiplicador está associado a um conjunto de coeficientes, que receberá um desses valores a cada nova passada dos valores de entrada durante a operação de filtragem. Assim, na primeira passada são selecionados os coeficientes h_0, h_1, h_2, h_3, h_4 , na segunda passada os coeficientes h_5, h_6, h_7, h_8, h_9 e assim por diante. Os dois blocos de memória funcionam alternando na função de armazenador e fornecedor de resultados parciais atuando em conjunto com o somador s_2 .

A Figura 5.5 apresenta um esboço do funcionamento relacionado à ocupação das memórias durante a evolução das etapas da sub-operações. Repare-se que no 1º passo os dois blocos estão vazios, e portanto apenas um deles é utilizado para guardar o resultado parcial. Nos outros passos parte do resultado parcial deve ser copiado para a memória receptora da filtragem e a outra parte irá se somar ao resultado vindo da estrutura do filtro.

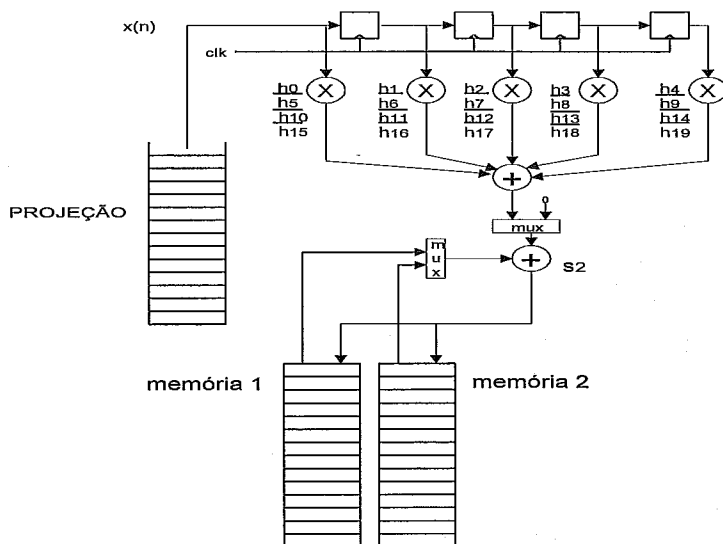


Fig. 5.4: Exemplo de estrutura reduzida para implementar a convolução linear.

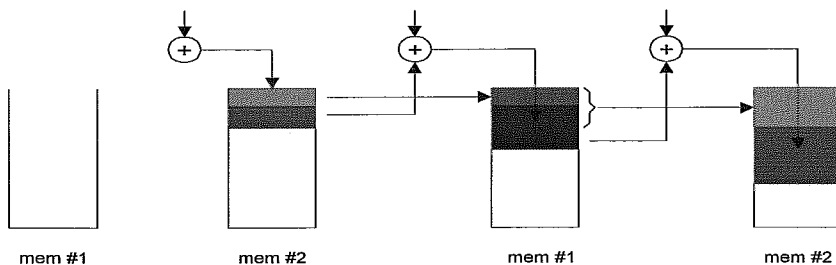


Fig. 5.5: Ocupação das memórias da estrutura reduzida.

Os coeficientes agrupados quase sempre possuem valores diferentes, impedindo o uso de multiplicadores dedicados na estrutura do filtro. Portanto, a realização da multiplicação modular também deverá ser feita utilizando-se o método de Cálculo de Índices. Conforme apresentado no Capítulo 3, a multiplicação nesse método possui 3 fases distintas: busca do índices dos operandos, soma dos índices em módulo $m-1$ e a busca do inverso do índice resultante. Adaptando-se o Cálculo de Índices à estrutura do filtro, chega-se a uma estrutura similar a da Figura 5.6, onde ao invés de se propagar o valor da entrada na estrutura, se propagam o seu índice e a informação indicando se o valor de entrada é zero. Isto é necessário uma vez que o cálculo de índices só tem sentido para valores diferentes de zero. Da mesma forma são guardados os índices dos coeficientes ao invés de seus valores. A Figura 5.6 também representa os coeficientes nulos existentes no filtro de Ram-Lak.

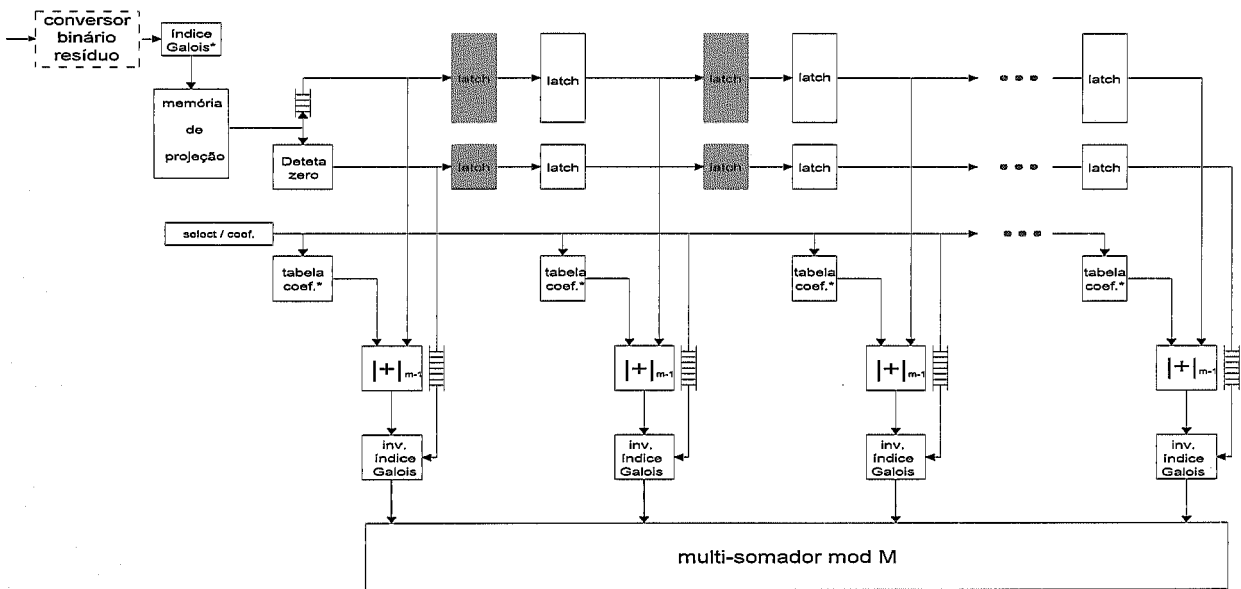


Fig. 5.6: Realização do filtro em aritmética de resíduos, utilizando-se multiplicadores baseados no cálculo de índices.

A etapa seguinte a multiplicação simultânea dos valores de entrada pelos respectivos coeficientes é a soma de seus resultados. Chamou-se de multi-somador módulo m a estrutura que realizará simultaneamente a soma de N resíduos módulo m . Uma possível implementação deste multi-somador é uma estrutura em árvore composta por somadores modular de dois operandos do tipo Dudgeale, conforme mostrado no Capítulo 3. Porém,

esses somadores, em geral, tem um custo em LEs cerca de 4 vezes ao de um somador binário comum. Como alternativa menos custosa, realizou-se a implementação do multi-somador módulo composto por dois circuitos distintos e consecutivos: uma árvore de somadores binários tipo *ripple carry* e um conversor binário-resíduo. Uma outra possibilidade é a utilização de uma estrutura do tipo CSA-CPA ao invés da árvore de somadores, com um custo adicional em termos LEs, porém com melhorias de roteamento desse circuito.

5.2.1 Implementação

Escolheu-se uma estrutura de filtragem com 8 multiplicadores. Analisando-se os coeficientes já normalizados em resíduos, foram feitas algumas modificações em relação a estrutura modelo da Figura 5.6.

A primeira modificação baseia-se na possível economia de multiplicadores, uma vez que existem valores dos coeficientes não zeros “consecutivos” e repetidos, conforme apresentado nas Tabelas 5.1 e 5.2. Essas tabelas contêm a metade dos coeficientes não nulos do filtro para módulos 61 e 5. Este comportamento dos valores dos coeficientes do filtro se mantêm para todos os módulos que compõem a base RNS. Não se agrupou os coeficientes iguais de uma só vez, pois ao se utilizar uma estrutura com cerca de 8 elementos multiplicadores, ocorreria um primeiro passo muito eficiente, mas extremamente ineficiente nos demais passos, uma vez que não é constante a multiplicidade dos coeficientes nas outras etapas. O grupamento de dois valores não nulos consecutivos foi adotado. O que representou cerca de 20% de economia no tempo total para filtragem para um acréscimo de 30 a 38% no *hardware* do multiplicador de acordo com o módulos da base RNS. A Figura 5.7 representa o esquema multiplicador que pode atuar com valores de coeficientes duplo ou simples.

A segunda modificação leva em consideração os valores simétricos dos coeficientes do filtro, portanto diminuindo pela metade o custo de armazenamento desses valores. Entretanto, é necessário acrescentar multiplexadores à estrutura, pois os dados fluem num sentido até atingir o coeficiente central do filtro e depois fluem no sentido contrário. Assim, o primeiro multiplicador da estrutura realizará a primeira multiplicação da convolução por

onde passa os dados de entrada e devido a simetria dos valores dos coeficientes também realizará a última multiplicação.

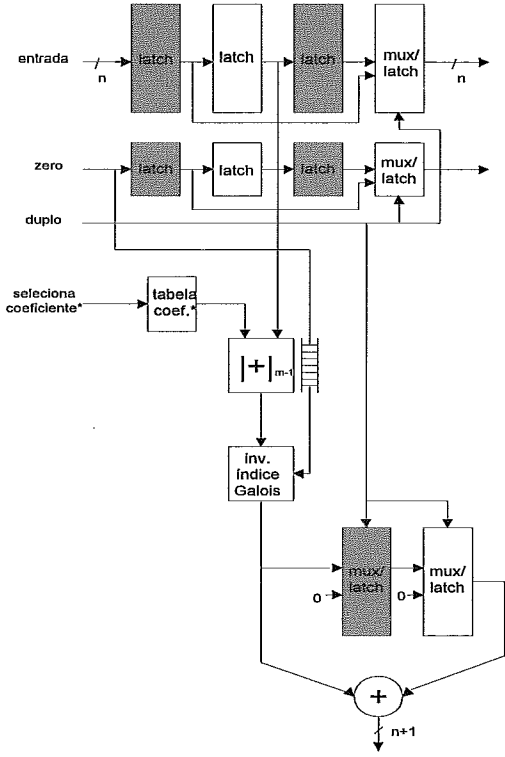


Fig.5.7 Multiplicador adotado que pode atuar com valor duplo ou simples de coeficientes consecutivos.

TABELA 5.1: Valores dos coeficientes do filtro de Ram-Lak após sua passagem para resíduos módulo 61

40	40	40	40	40	40	40	40	40	39	39	39	39	39	39	39
38	38	38	38	38	38	37	37	37	37	37	36	36	36	36	36
35	35	35	35	34	34	34	33	33	33	32	32	32	31	31	31
30	30	29	29	28	28	27	27	26	26	25	24	24	23	22	21
21	20	19	18	17	16	15	14	13	11	10	8	7	5	4	2
0	45	43	40	38	35	32	29	26	22	18	14	10	5	46	40
34	27	19	10	1	37	25	11	43	25	5	29	3	19	29	33
28	12	28	25	43	23	45	34	38	23	13	17	22	15	16	2
5															

TABELA 5.2: Valores dos coeficientes do filtro de Ram-Lak após sua passagem para resíduos módulo 5

3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2
1	1	1	1	1	1	0	0	0	0	0	4	4	4	4	4
3	3	3	3	2	2	2	1	1	1	0	0	0	4	4	4
3	3	2	2	1	1	0	0	4	4	3	2	2	1	0	4
4	3	2	1	0	4	3	2	1	4	3	1	0	3	2	0
3	1	4	1	4	1	3	0	2	3	4	0	1	1	0	4
3	1	3	4	0	4	2	3	3	0	0	2	1	0	3	0
3	0	2	0	2	1	3	0	1	0	3	3	2	1	1	3
0															

5.2.2 Discussão

É possível utilizar os blocos de memória disponíveis no FPGA para mapear tanto a memória que guarda os valores da projeção não filtrada, quanto as memórias que são empregadas no armazenamento temporário durante a execução do algoritmo de convolução linear. A memória necessária para guardar uma projeção requer 512 posições por M' , onde M' é o somatório do número de bits dos resíduos dos módulos que compõem a base RNS. Aproximando M' para 64 bits e lembrando que um bloco de memória pode ser mapeado como 512 x 4 bits, tem-se que para guardar uma projeção no formato RNS escolhido são necessários 16 blocos de memória. Para a execução da convolução linear, segundo o modelo adotado na Figura 5.4, são necessárias 2 memórias com 1024 posições por M' de largura para armazenamento temporário. Entretanto, da seqüência de 1024 valores do resultado da convolução, a retroprojeção só utilizará os 512 valores centrais, reduzindo-se à metade o tamanho das memórias empregadas. Assim, são necessários 32 blocos de memória para o armazenamento intermediário. O custo total para mapear internamente toda a quantidade de memória para realizar a convolução linear é de 48 blocos de memória. Essa quantidade de blocos é bastante elevada para ser atendida por somente um FPGA, restringindo-se o número de dispositivos capaz de atender esse requisito. No entanto, para um arranjo de dispositivos de capacidade média, esse requisito é facilmente atendido. Devido ao uso de aritmética de resíduos para a implementação da unidade de filtragem, o emprego de um FPGA de grande densidade ou vários de capacidade menor é indiferente no mapeamento da filtragem, uma vez que as operações aritméticas são executadas independentemente e em paralelo em cada módulo que compõe a base RNS. Uma outra vantagem ao se utilizar a memória interna é a economia do uso de pinos do dispositivo.

Para simulações da filtragem trabalhou-se a 80 MHz, e assim, a filtragem de uma projeção foi feita em cerca de 260 micro-segundos. Logo, a filtragem de 100 projeções gasta aproximadamente 26 mili-segundos, desprezando-se o tempo de configuração. Este tempo pode ser diminuído evitando-se a cópia de valores já definitivamente calculados que vão de uma memória para outra durante a execução do algoritmo. A latência para produzir o primeiro valor filtrado é pequena e desprezível para o total de operações a ser realizada. Entre os módulos RNS que trabalham em paralelo, a latência para os módulos de resíduos

com 6 bits, 61,59,53, 47, é maior do que para os demais módulos, pois o acesso a tabela inversora de índices tem 3 níveis de *pipeline*, contra 2 para os resíduos de 5 bits e 1 para resíduos de 4 ou menos bits.

Um aspecto interessante é a redução do custo do multiplicadores na estrutura reduzida, pois as tabelas de índices são suprimidas da estrutura de filtragem, uma vez que os dados de entrada introduzidos no *pipeline* são os índices dos valores a serem filtrados e, ao invés de se tabelar os coeficientes do filtro, tabelam-se seus índices.

A forma híbrida utilizada no multisomador modular, isto é, árvore de somadores associada ao conversor binário resíduo, é uma realização original e parece bastante promissora quanto a sua eficiência para utilização de tecnologia FPGA. Porém, esta forma híbrida necessita de mais estudos que avaliem seu comportamento para um número variado de entradas. Neste caso o uso de uma árvore de somadores do tipo *ripple carry* não compromete o desempenho por duas razões: o número baixo de operandos e o número pequeno de bits por operando.

Tabela 5.3: Representação do custo, para sub-estrutura de filtragem com 8 multiplicadores.

Módulo	Custo em LEs
5	465
7	595
11	740
13	745
16	718
17	824
19	936
23	953
29	953
31	933
47	1250
53	1292
59	1343
61	1413
Total	13165

5.3 UNIDADE DE RETROPROJEÇÃO PARA FEIXE PARALELOS COM INTERPOLAÇÃO LINEAR

O modelo mais simples de ser implementado nas unidades de retroprojeção é aquele que trabalha com feixes paralelos e utiliza interpolação linear. A Figura 5.8 apresenta um diagrama de blocos dessa unidade. O seu funcionamento começa com recebimentos de uma projeção filtrada, que é guardada nos bancos de memória par e ímpar, e os outros valores como os de *seno* e *coseno* relacionados ao ângulo de aquisição da projeção, que vão para o gerador de endereços da projeção, conforme descrito na Figura 4.3. Não é mostrado, mas é necessário um sincronismo com o endereçamento da memória de imagem com a geração dos valores da retroprojeção.

O gerador de endereços produz um valor de s por ciclo e conseqüentemente, gera os endereços das posições de memória SM e $SM+1$, cujos conteúdos serão usados na interpolação. Para isso adotou-se a separação da memória, que guarda a projeção filtrada, em 2 blocos, par e ímpar, permitindo o acesso a 2 posições de memória em 1 ciclo. Os valores dos pesos utilizados na interpolação linear são produzidos em aritmética binária convencional, devendo ser transformados em resíduos para em seguida entrar no cálculo da interpolação. O valor do ponto retroprojetado é somado ao valor correspondente guardado na memória de imagem, cujo conteúdo é o somatório de todas as retroprojeções anteriores. O resultado dessa soma será guardado na mesma posição da memória de imagem. Repare-se que a memória de imagem é lida sempre a cada ciclo de relógio e sofre uma escrita a cada ciclo de relógio. Logo, para implementar o bloco memória de imagem de forma que trabalhe em alta freqüência são sugeridos 2 modelos:

- i) Utilizar memória do tipo *dual-port*, que possui dupla entradas de endereços e dados, permitindo simultaneamente dois acessos. Entretanto, é uma solução de custo elevado em relação ao modelo a seguir.
- ii) Utilizar dois bancos de memória onde os bancos ficam alternando seu funcionamento a cada nova retroprojeção, isto é, fornecedor dos valores acumulados até aquela projeção ou acumulador dos valores projetados.

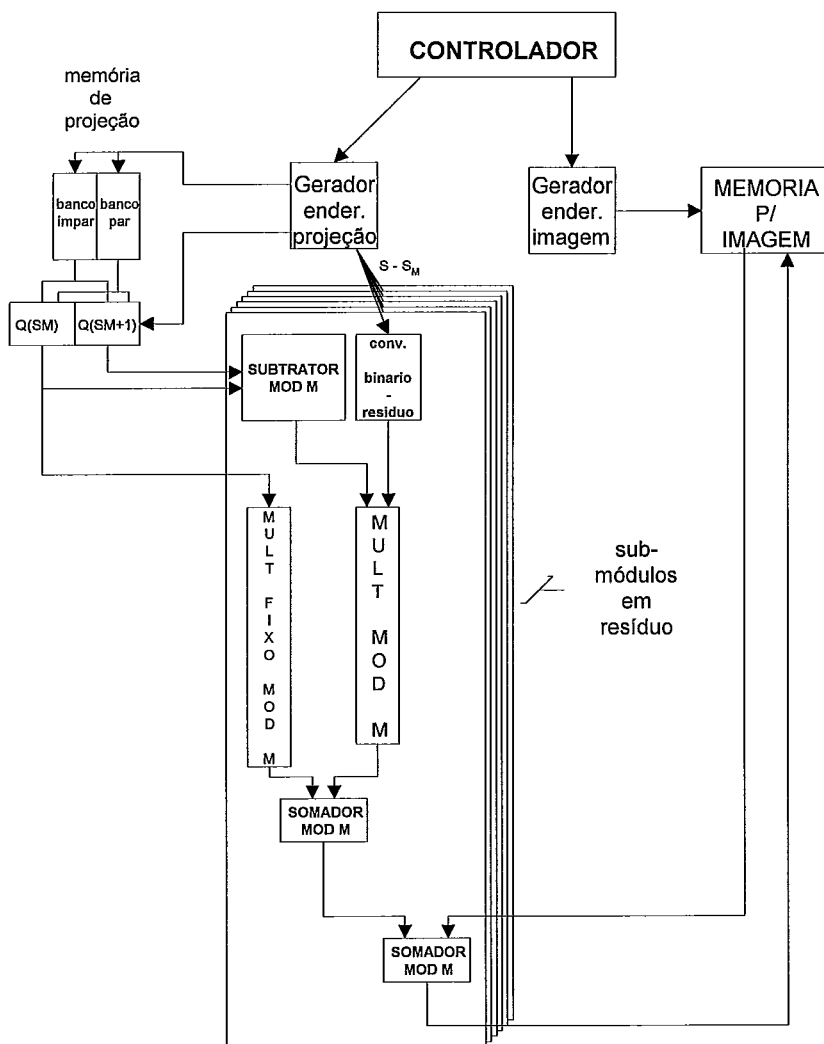


Fig. 5.8: Unidade de retroprojeção para feixes paralelos

5.3.1 Discussão

O diagrama da Figura 5.8 representa uma solução que utiliza somente uma unidade de retroprojeção. No caso da utilização de várias unidades de retroprojeção deve-se fazer algumas modificações como: substituir a ligação com a memória, pois em seu lugar essa ligação deverá ser entre as unidades de retroprojeção, com exceção da última unidade; a última unidade deverá estar ligada a memória e deverá ser acrescida de um somador modular, para somar o valor vindo da penúltima unidade de retroprojeção .

Para o caso de uma imagem de 512 x 512 pontos a memória de imagem deve possuir 18 bits de endereço por M^2 bits de largura, o que é demasiadamente grande e impraticável para ser mapeada em blocos de memória interna do FPGA. A projeção filtrada

a ser retroprojetada, cujo conteúdo já se encontra em resíduos, pode ser guardada internamente nos blocos de memória, semelhante ao que foi feito na operação de filtragem.

Internamente, cada unidade de retroprojeção contém um conjunto de sub-módulos que trabalham em aritmética de resíduos, em paralelo, que contém uma estrutura *pipeline* composta por um conversor binário-resíduo, um multiplicador modular fixo, um multiplicador modular genérico e três somadores modulares.

A Tabela 5.4 representa os custos, em LÉs, para um exemplo de implementação dos sub-módulos que realizam as operações em resíduo na retroprojeção com interpolação linear de 8 bits. Utilizou-se um sistema RNS redundante de base composta por $\{5,7,11,13,16,17,19,23,29,31,47,53,59,61\}$, onde 16 é o módulo que torna o sistema redundante. Os multiplicadores, para os resíduos com 6 bits de largura, foram implementados no modelo que usa apenas um bloco de memória (ver Seção 3.2). O multiplicador em módulo redundante também foi mapeado em blocos de memória. O número de estágios da estrutura que trabalha em resíduos foi 18 e outras latências devido ao acesso à memória e controle gasta cerca de 6 ciclos. Admitindo que a memória externa possa trabalhar a 80 MHz, o tempo total para fazer a retroprojeção dadas 100 projeções filtradas e usando apenas uma unidade de retroprojeção, tal qual mostrada na Figura 5.8 e sem levar em consideração os tempos gastos na filtragem e na configuração do sistema, mas gastando-se 512 ciclos para carregar uma projeção filtrada, é aproximadamente:

$$(512*512*100 + (512 +24) *100) /80 \times 10^6 = 0.33 \text{ segundos} .$$

Onde $512*512$ é o número de células que compõem a imagem, e 100 é o número de retroprojeções. Como a arquitetura trabalha sob a forma *pipeline* são necessários $512*512*100$ ciclos do relógio para se efetuar todas as 100 retroprojeções. Entretanto, antes de cada operação de retroprojeção é necessário carregar a projeção filtrada no FPGA, e portanto, para isso são necessários 512 ciclos do relógio por projeção, num total de $100*512$ ciclos para todas as projeções. A latência para se produzir um primeiro resultado por retroprojeção é a soma de todas as latências das unidades que compõem a estruturas, neste caso dá um total de 24 por projeção e $24*100$ para todas projeções. Assim, o tempo

total para executar as 100 retroprojeções é dada pela soma dos ciclos do relógio acima descritos, vezes o período da frequência de trabalho da estrutura.

Tabela 5.4: Custos aproximado, em LEs, da implementação de uma unidade de retroprojeção.

Módulo, (número de bits)	Conv. Binário-resíduo	Multiplicador	Somador (x. 3)	Multiplicador fixo	Total Para todos módulos
5,7 (3)	21	16	15 x 3	3	170
11,13 (4)	27	31	19 x 3	4	238
17,19,23,29,31(5)	33	70	23 x 3	33	1025
47,53,59,61 (6)	39	121	27 x 3	39	1120
Redundante	0	0	4 x 3	4	16
					2569

5.4 UNIDADE DE RETROPROJEÇÃO PARA FEIXES DIVERGENTES COM INTERPOLAÇÃO LINEAR

As unidades de retroprojeção (Figura 5.9) para feixes divergentes trabalham também em *pipeline*, de forma similar às unidades de retroprojeção para feixes paralelos. A sua estrutura é muito parecida com a de feixes paralelos, entretanto com o tamanho e número de estágios um pouco maior, devido a inclusão de um conversor binário resíduo e um multiplicador RNS aos circuitos que implementam as operações RNS. As organizações das memórias internas e externa seguem os mesmos modelos estabelecidos para feixes paralelos. O gerador de endereços, que era implementado por somadores e multiplexadores, tem nessa implementação uma complexidade maior, incluindo uma operação de divisão, e foi chamado de bloco para cálculo de U e S' , implementado segundo o diagrama mostrado na Figura 5.10. Introduziu-se um novo bloco que realiza o cálculo de $1/U^2$. Repare-se que o resultado desse novo bloco deve ser convertido em resíduo antes de entrar no processamento das operações que calcula o valor da retroprojeção.

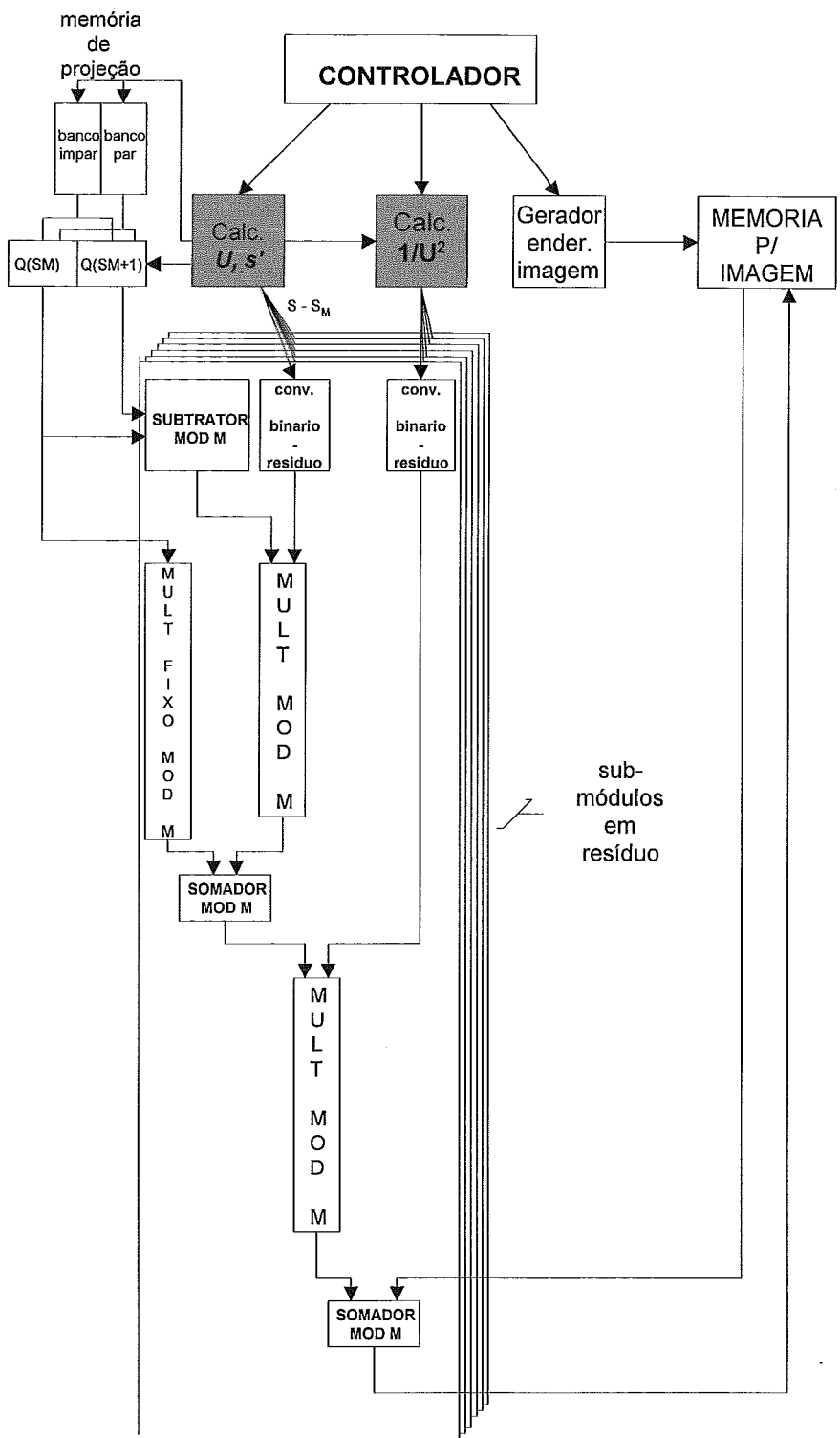


Fig 5.9: Unidade de retroprojeção para feixes divergentes.

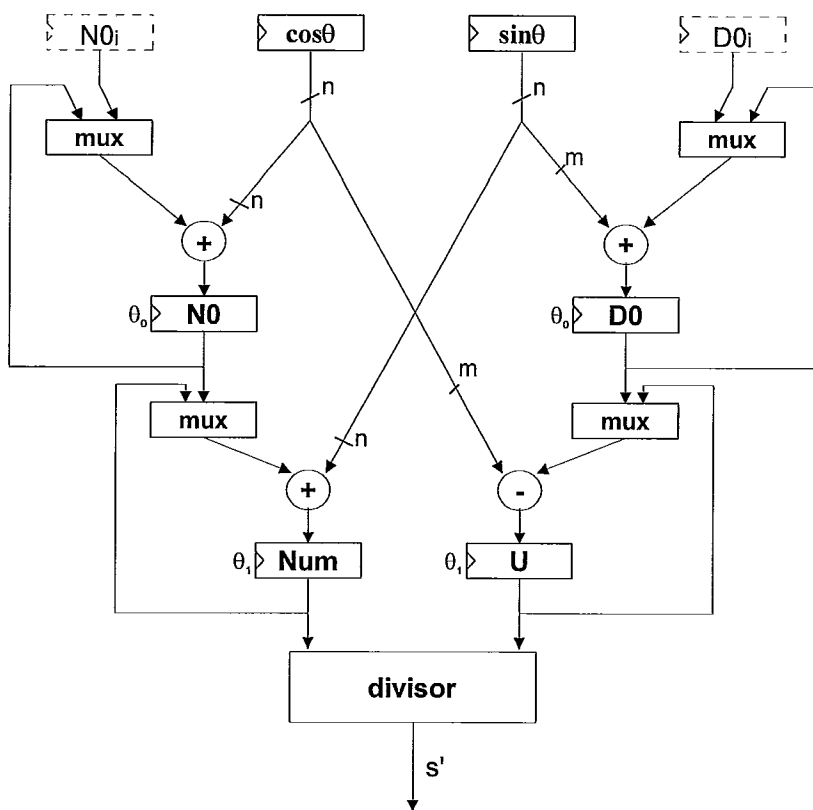


Fig. 5.10: Diagrama em blocos para geração de s' .

5.4.1 Operações não-lineares na retroprojeção

A expressão numérica da retroprojeção para feixes divergentes requer uma maior complexidade para ser implementada do que a de feixes paralelos, principalmente por executar operações não lineares como divisões e recíprocos. Das operações matemáticas elementares a divisão é a mais complexa e custosa para ser implementada. Numa tentativa de diminuir os custos associados à implementação plena dessas funções no algoritmo de retroprojeção divergente, alguns pesquisadores exploraram aproximações dessas funções considerando que a precisão dos resultados não precisa necessariamente ser muito elevada.

Agi *et al.* [16] desenvolveram estimadores recursivos para calcular os valores de $1/U^2$ e s' . Os estimadores são baseados na expansão de 2ª ordem da série de Taylor de suas respectivas expressões. Entretanto, numa tentativa de se usar recursividade, e portanto

somente operações de adição, os termos de duas variáveis da série de Taylor foram simplificados e tabelados por aproximações. Desta forma, os valores de s' para possuir uma precisão aceitável, necessitam que o valor de D , distância entre a fonte e o centro de rotação, seja grande, logo impondo-se restrições à geometria do sistema.

O trabalho de Jain e Lin [17, 18] propõe o uso de células UNL (*Universal Nonlinear cell*) em integração do tipo MCM (*Multi Chip Module*), para implementar com precisão média funções como recíproco e seno/coseno. A UNL é baseada na expansão de 2ª ordem da série de Taylor para aproximar um polinômio que representa a função não linear desejada, porém utiliza interpolação de valores tabelados ao invés de valores exatos dos coeficientes dos termos da série, com o intuito de se diminuir o custo do tabelamento. Uma célula UNL é composta de memórias ROMs, 2 somadores/subtratores e um multiplicador rápido de 20x20 bits. Esta solução permite uma precisão de 24 bits para o cálculo de s' , contudo requer o uso de 2 células UNL, e portanto 2 multiplicadores.

É desejada uma melhor precisão do que a oferecida pelos estimadores não-linear de Agi, e o uso de multiplicadores binários convencionais é evitado neste trabalho, devido ao custo de implementação em FPGA. Buscou-se então, dentre os algoritmos de divisão disponíveis, aquele com características mais adequadas para realizar o cálculo de s' em FPGAs.

Os algoritmos de divisão podem ser agrupados em 5 classes principais [123]:

(i) recorrência de dígito, (ii) iteração funcional, (iii) base numérica elevada, (iv) baseados em tabelas e (v) latência variável. Existem implementações híbridas combinando alguns desses algoritmos. Os divisores de latência variável ou exploram o fato de algumas fases que compõem a divisão poderem ser executadas mais rapidamente de acordo com os valores dos operandos, produzindo resultados em tempos diferentes; ou reutilizam resultados prévios de outras divisões. A arquitetura proposta neste trabalho trabalha em forma *pipeline* e, portanto, requer sincronização, produzindo um resultado para a divisão sempre com o mesmo número de ciclos e, portanto, com o mesmo tempo, logo o uso de latência variável é inadequado.

Os métodos iteração funcional, base numérica elevada e baseados em tabelas utilizam uma ou mais multiplicações, uma vez que a divisão é vista como a multiplicação do dividendo pelo recíproco do divisor e, portanto, muito custosa para ser implementada em FPGAs. Esses algoritmos diferenciam-se principalmente na forma de tabelamento do recíproco e do tipo de refinamento usado para melhorar a sua precisão. Os métodos usados para aproximar o resultado final da divisão incluem expansão em série de Taylor e Newton-Rapson.

Já os algoritmos que utilizam recorrência de dígitos parecem os mais adequados para trabalhar no modelo de FPGA escolhido, pois utilizam métodos subtrativos e sua implementação, em geral, é menos custosa. No entanto, os métodos subtrativos tem convergência linear, enquanto que os de iteração funcional e de base numérica elevada tem convergência quadrática. Isto implica que a latência dos métodos que utilizam multiplicadores é pequena, enquanto que os de recorrência de dígito possuem latência elevada.

5.4.2 Implementação de s'

Foi implementado uma versão *pipeline* do algoritmo de recorrência de dígitos chamado SRT (Apêndice A) para calcular o valor de s' . Utilizou-se a base 4 com mínima redundância, onde cada etapa irá produzir um quociente numa das seguintes representações $\{-2,-1,0,1,2\}$, ou seja dois bits por iteração. Para o funcionamento correto é necessário implementar as etapas de normalização do dividendo e do divisor. Neste trabalho o dividendo deverá ser normalizado no intervalo $(-8/3, 8/3)$ e o divisor no intervalo $[0.5, 1)$. Na geração do quociente foi utilizado um número de bits sugerido em [124], isto é, selecionando 6 bits do resto parcial deslocado e 3 bits do divisor, segundo a disposição indicada na Figura 5.11.

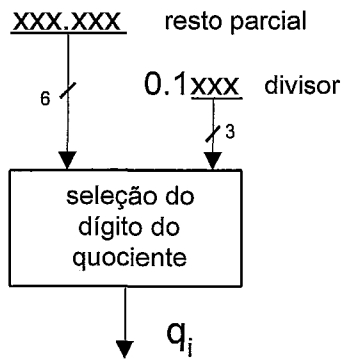


Fig. 5.11: Função de geração do dígito do quociente com 9 bits de entrada.

O valor de U , divisor da expressão s' , é sempre positivo e, através da inspeção da equação 4.11, é fácil determinar a faixa de valores que poderá assumir. Por exemplo: numa aplicação para reconstruir imagens de resolução de 512 x 512 pontos com valores de $D > 512$, a faixa dos valores de U é (0.291, 1.707). Esta faixa facilita a implementação do *hardware* que irá fazer a normalização, pois necessitará apenas de 1 ou 2 deslocamentos para atingir o intervalo [0.5, 1).

Como o objetivo de se obter um resultado de divisão por ciclo do *pipeline* foi necessária a replicação do *hardware* que implementa o algoritmo SRT. No caso da reconstrução de imagens com 512 x 512 pontos com a retroprojeção trabalhando com interpolação linear de 8 bits, o resultado da divisão, s' , requer 18 bits de precisão. Para se obter essa precisão são necessários dividendos com 36 bits e divisores com 18 bits. É necessário construir um *pipeline* com 9 estágios SRT base 4 para produzir 18 bits no último estágio.

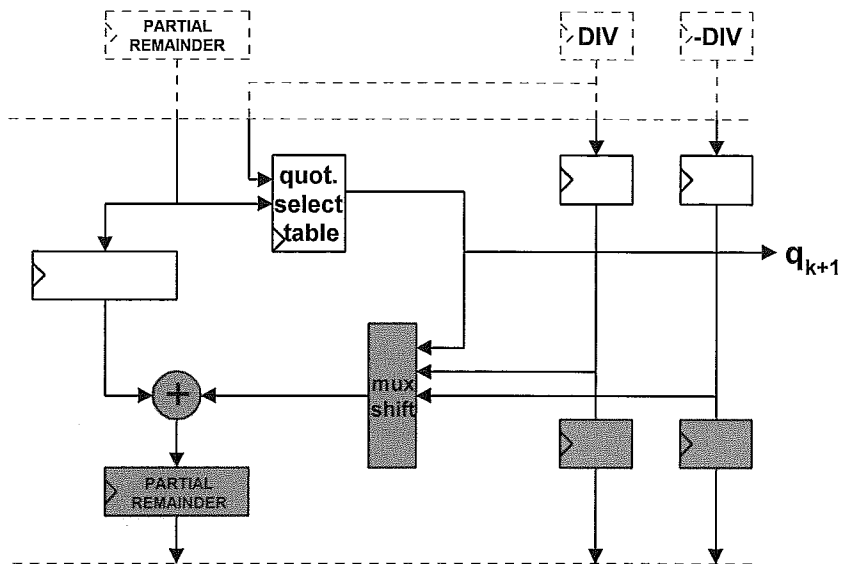


FIG. 5.12: Estágio SRT em 2 etapas

Cada estágio SRT (Figura 5.12) funciona em duas etapas:

- i) Seleção do quociente;
- i) Geração do múltiplos do divisor junto com o cálculo do resto parcial.

Note-se que cada etapa consome um ciclo do *pipeline* e portanto são necessários dois ciclos para produzir um dígito do quociente. No entanto, são produzidos 2 bits do resultado por iteração do algoritmo, assim é possível produzir 18 bits por ciclo no final do *pipeline*. O funcionamento do *pipeline* é simples. Cada estágio SRT recebe o quociente, o divisor e seu valor em complemento de 2 e o resto parcial vindos do estágio anterior; calcula um novo dígito do quociente, com 2 bits, e acumula o valor do quociente ao valor do quociente vindo do estágio anterior e envia esses valores para o próximo estágio. O registrador que acumula o quociente em cada estágio vai aumentando de tamanho até que, no último estágio, possua 18 bits. O processo utilizado para voltar o quociente para a notação convencional em complemento de dois é o “On-the_fly” [124], e é executado à medida em que um novo dígito do quociente é gerado e concatenado aos resultados parciais. As regras utilizadas na conversão “On-the_fly” são mostradas pelas Equações 5.1 e 5.2, um esboço do circuito que foi utilizado para implementá-lo pode ser visto na Figura

5.13. Observe-se que duas formas do quociente são necessárias e mantidas em registradores separados, Q e QM, e o resultado final estará contido em Q.

$$QM[k+1] = \begin{cases} (Q[k], q_{k+1} - 1) & \text{if } q_{k+1} > 0 \\ (QM[k], ((r-1) - |q_{k+1}|)) & \text{if } q_{k+1} \leq 0 \end{cases} \quad (5.1)$$

$$Q[k+1] = \begin{cases} (Q[k], q_{k+1}) & \text{if } q_{k+1} \geq 0 \\ (QM[k], (r - |q_{k+1}|)) & \text{if } q_{k+1} < 0 \end{cases} \quad (5.2)$$

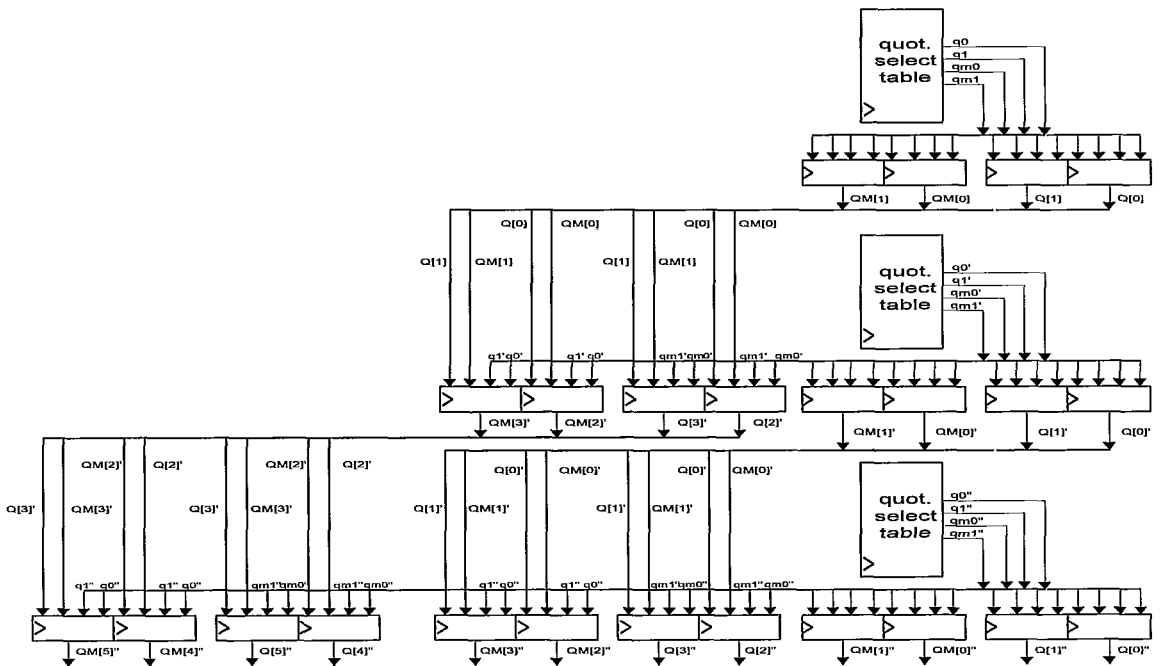


Fig. 5.13: Diagrama simplificado de conversão "ON THE FLY".

O tabelamento da função de seleção em um estágio SRT foi implementado configurando-se a memória interna do modelo de FPGA em 512 x 4 bits. Embora somente 3 bits fossem necessários para representar os dígitos do quociente, foram utilizados todos os 4 bits da memória, conforme mostra a Tabela 5.5. Essa representação simplificou algumas etapas do projeto incluindo a formação dos múltiplos do divisor e a conversão "On-the-fly".

Tabela 5.5: representação dos dígitos do quociente.

Quociente(q_j)	q_1	q_0	q_{m0}	q_{m1}
2	1	0	0	0
1	0	1	0	0
0	0	0	0	0
-1	0	0	0	1
-2	0	0	1	0

5.4.2.1 Discussão

A Figura 5.10 representa um diagrama simplificado do algoritmo recursivo (Seção 4.2.2) para obtenção de s' , onde D , distância entre a fonte e o centro de rotação, tem valor em potência de 2. O número de bits necessários para evoluir as somas nesse diagrama deve levar em consideração a precisão estabelecida pela operação de divisão. No exemplo ficou estabelecido uma precisão de 36 bits do dividendo e 18 bits do divisor, respectivamente Num e U . Adotou-se 40 bits para os somadores da recursão para obtenção do dividendo com o propósito de manter alguns bits de segurança, evitando-se assim a propagação de erros de arredondamentos. Para verificar a precisão dos valores de s' simulou-se o diagrama da Figura 5.10 para fornecer os valores dos operandos da divisão e a divisão foi simulada segundo o modelo SRT acima descrito. Comparando-se o resultados simulados aos valores de s' calculado em aritmética de ponto flutuante, para o caso de uma imagem de 512 x 512 pontos que utiliza 360 projeções na retroprojeção, obteve-se resultados muito bons onde 97% dos resultados apresentam erro menor do que o bit menos significativo utilizado na representação binária do resultado.

Cada estágio SRT gasta 110 LEs e um bloco de memória, e as simulações preliminares trabalham a 60 MHz. O custo para implementar a divisão para o cálculo de s' foi 1860 LEs e 9 blocos de memória. O circuito simulado tem uma latência de 20 ciclos.

5.4.3 COMPUTAÇÃO DE $1/U^2$

Estabeleceu-se na Seção 4.2.3 que as aproximações da função $1/U^2$ apresentam um resultado muito bom para trabalhar no algoritmo de retroprojeção para feixes divergentes em aritmética de resíduos, quando se utilizam de 8 a 10 bits para constante multiplicadora aplicada aos valores de $1/U^2$ antes de ser convertido em resíduos. Considerando-se que os valores de U estão na faixa de (0.2910, 1.707) tem-se os resultados de $1/U^2$ no intervalo (0.3431, 11.809). A utilização da constante multiplicadora com 8 bits implica numa saída inteira com 12 bits, já a constante multiplicadora com 10 bits implica numa saída com número de 14 bits. Logo, o mapeamento direto da função $1/U^2$, numa visão simplificada, poderia ser feito através de mapeamento com 12 bits entrada por 12 bits de saída ou 14 bits de entrada por 14 bits de saída, o que é extremamente custoso para ser implementado em FPGAs, mesmo que se utilizem os blocos de memória. Analisando-se a função do ponto de vista de intervalos da entrada, pode-se interpretar o resultado da função $1/U^2$ da seguinte forma:

- i) 0.2910 a 0.5 implica numa saída binária do tipo: xxxxxxxxxx;
- ii) 0.5 a 1 implica numa saída binária do tipo: 00xxxxxxx;
- iii) 1 a 1.707 implica numa saída binária do tipo: 0000xxxxxxx;

Observando esses intervalos cuja saídas estão numa representação com 14 bits, decidiu-se implementar a função $1/U^2$ com 10 bits de entrada por 10 bits de saída, para entradas normalizadas no intervalo [1, 2). Assim, a computação $1/U^2$ será formada por 3 etapas: a primeira etapa é normalização da entrada que está no intervalo (0.2910, 1.707), a etapa intermediária é o cálculo da função normalizada e a última etapa é a desnormalização. Logo a saída para esses intervalos nessa abordagem dará os seguintes resultados por faixa:

- i) 0.2910 a 0.5 que implica numa saída binária do tipo: xxxxxxxx0000, com precisão menor do que a indicada pela simulações;
- ii) 0.5 a 1 que implica numa saída binária do tipo: 00xxxxxxx00, com precisão igual a utilizada com a constante multiplicadora de 8 bits.

iii) 1 a 1.707 que implica numa saída binária do tipo: 0000.xxxxxxxxxx, com precisão igual a utilizada com a constante multiplicadora de 10 bits.

A normalização para o intervalo dos valores possíveis nesse trabalho irá requerer no máximo 2 deslocamentos, e assim a desnormalização poderá ter nenhum deslocamento, dois deslocamentos ou 4 deslocamentos. Logo, as implementações do normalizador e desnormalizador são muito simples. Já o mapeamento direto em blocos de memória para realizar uma função com 10 bits de entrada e 10 bits de saída consome 5 blocos de memória. Como o uso direto de blocos de memória só deve ser feito quando não houver mais alternativas, aplicou-se o recente método *Symmetric Bipartite tables* [125-127] (Apêndice B) para calcular a função $1/U^2$ com o intuito de reduzir a quantidade de memória utilizada em sua implementação. Esse método foi desenvolvido para aproximar funções com precisão média entre 16 a 24 bits, utilizando tabelas. Comparados às outras técnicas que visam também reduzir a quantidade de memória necessária para aproximar funções elementares (recíproco, seno, cosseno, raiz quadrada, \log_2), os métodos baseados em tabelas bipartidas têm a vantagem de não utilizar multiplicadores [125-130]. A redução de quantidade de memória quando comparada ao mapeamento convencional de funções em tabelas pode chegar entre 15 a 41 vezes para operandos com 16 bits e entre 99 a 270 vezes para operandos de 24 bits. A redução de memória será maior quanto maior for o número de bits do operando.

5.4.3.1 Aplicando-se o método *Symmetric Bipartite Tables* em $1/U^2$:

Nesse método a função é aproximada pela soma de duas funções a_0 e a_1 derivadas da expansão de segunda ordem da série de Taylor (Apêndice B). Contudo, essas funções usam somente parte dos bits de entrada, diminuindo assim a quantidade de memória para seu mapeamento. A simetria produzida pelo método permite reduzir a quantidade de memória utilizada no mapeamento da função a_1 (Equação 5.4), o que pode ser observado na Tabela 5.6. Inicialmente, supõe-se a entrada dada por 1.xxxxxxxxxx, com a parte inteira igual a 1, e parte fracionária com 10 bits. Divide-se a parte fracionária em 3 partes x_0 , x_1 , x_2 respectivamente 4, 4 e 2 bits (Figura 5.14) tal que seus valores são dados por:

$$\begin{aligned}
 0 &\leq x_0 \leq 1 - 2^{-4} \\
 0 &\leq x_1 \leq 2^{-4} - 2^{-8} \\
 0 &\leq x_2 \leq 2^{-8} - 2^{-10}
 \end{aligned}
 \tag{5.3}$$

$$f(x) \approx a_0(x_0, x_1) + a_1(x_0, x_2) \tag{5.4}$$

$$\begin{aligned}
 a_0(x_0, x_1) &= 1/U^2 \\
 \text{onde } U &= 1 + x_0 + x_1 + 2^{-9} - 2^{-11}.
 \end{aligned}
 \tag{5.5}$$

$$\begin{aligned}
 a_1(x_0, x_2) &= -2 \cdot (x_2 - 2^{-9} + 2^{-11}) / W^3 \\
 \text{onde } W &= 1 + x_0 + 2^{-5} - 2^{-9} + 2^{-9} - 2^{-11}.
 \end{aligned}
 \tag{5.6}$$

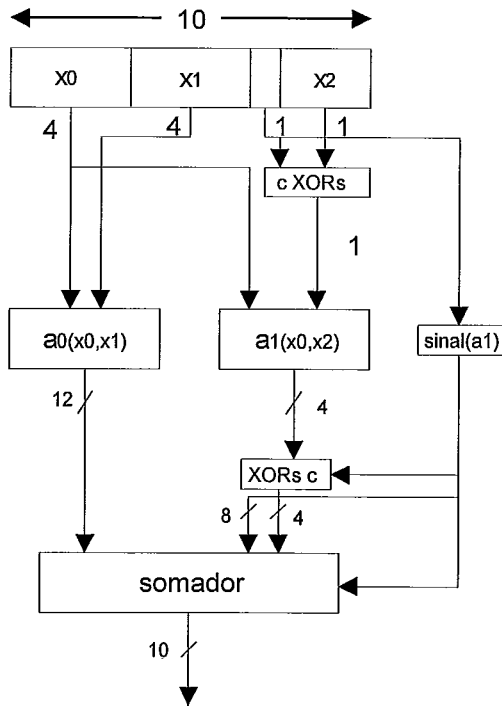


Fig. 5.14: Diagrama em blocos para calcular $1/U^2$ utilizando-se o método *Symmetric bipartite Tables*.

O método para produzir 10 bits com erro menor do que 2^{-10} requer o uso de bits de segurança para evitar a propagação de erros, implicando nesse caso o uso de 12 bits de largura para a saída de a_0 . A saída da função a_1 precisa somente de 4 bits para ser

representada, pois os demais bits, isto é, os 8 bits mais significativos, seguem o valor do sinal de a_1 , isto é, 0 para número par e 1 para número ímpar.

Tabela 5.6: Valores da função $a_1(x_0, x_2)$ multiplicados por 2^{12}

X0\X2	00	01	10	11
0000	11	4	-4	-11
0001	9	3	-3	-9
0010	8	3	-3	-8
0011	7	2	-2	-7
0100	6	2	-2	-6
0101	5	2	-2	-5
0110	4	1	-1	-4
0111	4	1	-1	-4
1000	3	1	-1	-3
1001	3	1	-1	-3
1010	3	1	-1	-3
1011	2	1	-1	-2
1100	2	1	-1	-2
1101	2	1	-1	-2
1110	2	1	-1	-2
1111	2	1	-1	-2

5.4.3.2 Discussão

De fato, o uso do método *Symmetric Bipartite Tables* permitiu a redução do número de blocos de memória na implementação da função $1/U^2$ quando comparada à implementação direta. Nessa solução precisou-se de dois blocos de memória para mapear a_0 , e a_1 foi diretamente mapeado em LUTs. O custo total da implementação foi 40 LEs e dois blocos de memória. Essa função implementada sob forma *pipeline* e atinge uma frequência de aproximadamente 90 MHz.

É importante lembrar que o ideal é mapear no mesmo FPGA os blocos que calculam U , s' e $1/U^2$. A solução proposta para calcular s' gasta muitos blocos de memória devido à implementação da operação de divisão. A economia de blocos de memória no cálculo de $1/U^2$ contribui na redução dos requisitos de memória total para escolha do FPGA que implementará as funções descritas acima.

A solução para implementar retroprojeção para feixes divergentes requer um número de estágios muito maior que a solução para feixes paralelos. São necessários cerca de 44 estágios para produzir o primeiro valor a ser retroprojetado. Mesmo assim, comparado ao tempo total para realizar a retroprojeção para imagem de 512 x 512 pontos, esse valor é muito pequeno. Os custos, em LEs, relacionados aos sub-módulos que trabalham em RNS são aproximadamente o dobro das respectivas implementações para feixes paralelo. O custo de implementação da divisão é alto, e da mesma ordem que custo de implementação dos sub-módulos em RNS. Portanto, muito mais caro que a implementação visando feixes paralelos. O desempenho preliminar, isto é, a frequência máxima de operação atingida foi de 60 MHz. Esse valor é menor do que o atingido no caso de feixes paralelos, 80MHz, isto é devido principalmente aos problemas de roteamento e não à eficiência da implementação dos sub-módulos que compõem a solução final. Logo, confirma a possível busca de soluções mais rápidas num outro momento.

Simplificadamente, pode-se supor que a retroprojeção de 200 projeções filtradas, utilizando-se somente uma unidade de retroprojeção e o mesmo raciocínio adotado para feixes paralelos, requer:

$$(512*512*200 + (512 + 44)*200) / 60 \times 10^6 = 0.87 \text{ segundos} .$$

Onde $512*512$ é o número de células que compõem a imagem, e 200 é o número de retroprojeções. Como a arquitetura trabalha sob a forma *pipeline* são necessários $512*512*200$ ciclos do relógio para se efetuar todas as 200 retroprojeções. Entretanto, antes de cada operação de retroprojeção é necessário carregar a projeção filtrada no FPGA, e portanto, para isso são necessários 512 ciclos do relógio por projeção, num total de $200*512$ ciclos para todas as projeções. A latência para se produzir um primeiro resultado

por retroprojeção é a soma de todas as latências das unidades que compõem a estruturas, neste caso dá um total de 44 por projeção e $44 \cdot 200$ para todas projeções. Assim, o tempo total para executar as 200 retroprojeções é dada pela soma dos ciclos do relógio acima descritos, vezes o período da frequência de trabalho da estrutura.

5.5 ANÁLISE DOS RESULTADOS

Para ilustrar o desempenho esperado da proposta, estimou-se o tempo para reconstruir imagens de 512×512 pontos para o algoritmo de retroprojeção filtrada (Tabela 5.8), considerando-se os tempos de configuração para alguns dispositivos conforme apresentado na Tabela 5.7 e os resultados obtidos nas Seções anteriores e no Capítulo 3. A tabela 5.9 apresenta os tempos para reconstruir imagens utilizando máquinas comerciais baseadas em alguns modelos do Processador PENTIUM, cujos programas foram escritos em linguagem C, Microsoft Visual C++.

Tabela 5.7: Alguns parâmetros de FPGAs com densidade pequena, média e grande.

Dispositivo	Número de elementos lógicos	Equivalência em portas lógicas	Tempo de configuração em milisegundos
EPF10K20	1152	20000	22.5
EPF10K50	2880	50000	60.9
EPF10K100	4992	100000	117.2

Tabela 5.8: Tempo estimado para reconstrução de imagens para a arquitetura proposta.

Tipo de geometria	Número de projeções	Tempo de filtragem [s]	Tempo de Configuração [s]	Tempo de inversão RNS/Binário [s]	Tempo de retroprojeção [s]	Frequência de Trabalho	Tempo total [s]
Paralela	100	0.026	0.0225	0.009	0.33	80MHz	0.38
Paralela	100	0.026	0.1172	0.009	0.33	80MHz	0.48
Divergente	200	0.052	0.0609	0.009	0.87	60MHz	0.99
Divergente	200	0.052	0.1172	0.009	0.87	60MHz	1.05

Tabela 5.9: Desempenho do algoritmo de retroprojeção filtrada em máquinas comerciais.

Máquina/freqüência	Tipo de Geometria	Número de Projeções	Tempo (segundos)
PENTIUM I-166MHz	Paralela	100	45
PENTIUM I-166MHz	Divergente	200	116
PENTIUM III-500MHz	Paralela	100	24
PENTIUM III-500MHz	Divergente	200	52

Comparando-se os tempos estimados da nossa proposta com os tempos obtidos em máquinas comerciais, conclui-se que a utilização configuração composta de uma unidade para fazer filtragem associada a apenas uma unidade retroprojeção atinge desempenho de 1 a 2 ordens de grandeza mais rápido que algumas das mais difundidas máquinas comerciais.

Nos dispositivos utilizados, o tempo de reconfiguração dos FPGAs cresce linearmente com o tamanho do dispositivo. A utilização de pequenos dispositivos implica numa pequena parcela do tempo de configuração em relação ao tempo total da reconstrução, mas o uso de grandes dispositivos implica que uma parcela significativa do processamento é gasto configurando-se o dispositivo. A utilização de várias unidades de retroprojeção tem sua eficiência limitada pelo tempo de configuração, entretanto este tempo de configuração pode ser sobreposto, em parte, ao tempo de retroprojeção, configurando-se a unidade de filtragem para fazer a conversão resíduo-binário para a imagem reconstruída, logo após ser filtrada a última projeção.

Felizmente, a diminuição do tempo de configuração tem sido bastante investigada e alguns fabricantes de FPGA possuem mecanismos de configuração trabalhando cerca de 6 vezes mais rápido do que os dispositivos utilizado em nossas simulações [131].

CAPÍTULO 6

TRABALHOS CORRELATOS

A seguir são apresentados os principais trabalhos relacionados à exploração de paralelismo nos algoritmos de retroprojeção filtrada. A tabela 6.1 lista as características importantes desses trabalhos. Por fim, são também apresentadas as semelhanças e diferenças entre esses trabalhos e a proposta deste trabalho.

1) *Sprint* (Systolic processor with a reconfigurable interconnection network of transputer)

É um sistema de multiprocessadores composto por 64 nós de processamento, usando processadores Transputer T414 e T800, que estão associados a um computador hospedeiro modelo *MicroVaxII* [9]. Este sistema pode operar como SIMD ou MIMD. Os nós de processadores são interconectados por uma rede reconfigurável que realiza diversas topologias como: malha de 2 dimensões, malha triangular, árvore binária, *shuffle-exchange*. O cálculo da retroprojeção filtrada é feito sob forma *sistólica* e são feitas a partir de um dos dois métodos:

i) Método das projeções residentes: No cálculo da retroprojeção filtrada cada processador recebe uma ou mais projeções. A primeira etapa é aplicar um filtro na projeção, em seguida é feito a retroprojeção obtendo-se uma imagem parcial. A imagem final é feita somando-se todas as imagens parciais vindas de todos elementos processadores

ii) Método da imagem residente: Cada elemento processador irá reconstruir uma região da imagem. Inicialmente, as projeções são distribuídas entre todos os elementos processadores, que realizam a filtragem e as retornam ao hospedeiro. Para computar a retroprojeção, as projeções filtradas são difundidas uma a uma para todos os processadores. Cabendo a todos processadores realizar as operações de retroprojeção, somente para as respectivas sub-áreas da imagem, para todas as projeções filtradas.

Em ambos os métodos, para formar a imagem final, deve-se realizar a soma das retroprojeções que se encontram distribuídas entre os elementos processador. Como a topologia é em malha esta soma é realizada num primeiro instante nos elementos da mesma

linha da malha, acumulando-se os resultados nos elementos de uma mesma extremidade na malha, depois faz-se a soma na outra direção da malha utilizando-se somente esses elementos da extremidade.

2) O trabalho de Zapata *et al.* [11] trata o problema de reconstrução de um objeto em 3 dimensões a partir das sua projeções divergentes em 2 dimensões (*Cone Beam*). Neste trabalho a reconstrução é feita por uma máquina hipercúbica, que implementa o algoritmo retroprojeção filtrada por FFT. O modelo computacional escolhido foi SIMD, com memória não compartilhada. A linguagem de programação utilizada foi uma extensão da linguagem C, que possui algumas facilidades como: operadores e tipos de dados que permitem o controle de operações intra e inter elementos processadores; armazenamento de dados tanto no hospedeiro quanto nos elementos processadores. O algoritmo de reconstrução de imagem foi dividida em 3 etapas: i) Filtragem das projeções, que envolve cálculo de FFT em duas dimensões, cálculo da função peso, e inversa da FFT 2D; ii) Retroprojeção de cada projeção e a filtragem do objeto tridimensional, que envolve o cálculo da FFT 3D; iii) Cálculo do filtro e a inversa da FFT 3D. Para cada etapa acima foi feita uma análise antes de realizar a sua paralelização

A eficiência da implementação do algoritmo como um todo é devido aos seguintes fatores: distribuição da imagem e dados usando um modelo de armazenamento consecutivo, na forma de indexação dos elementos processadores e no uso de diferentes partições do hipercubo para várias etapas do algoritmo.

3) O trabalho de Lin *et al.* [10] descreve a implementação de algoritmos para processamento paralelo numa topologia configurável para aplicação em tomografia computadorizada, utilizando fonte de radiação divergente e algoritmo de reconstrução por retroprojeção filtrada. O processamento tomográfico foi agrupados em 3 etapas de computação: i) Pré-convolução; ii) Convolução e retroprojeção; iii) Pós-processamento. A retroprojeção neste modelo é trabalhada em coordenadas polares.

Através da observação do fluxo de dados nas diversas etapas da computação do algoritmo, direcionou-se a pesquisa para uma arquitetura multiprocessada de topologia

flexível que pudesse acomodar eficientemente a comunicação entre os elementos de processamento nessas diversas etapas.

O desempenho do sistema foi devido a exploração do paralelismo inerente ao algoritmo de retroprojeção filtrada e a redução do número de computações trigonométricas através da utilização de propriedades de simetria trigonométrica.

A exploração de paralelismo para etapa de retroprojeção foi feita dividindo as projeções entre os elementos processadores. A retroprojeção total da imagem é feita através do envio das retroprojeções parciais por uma conexão em anel, onde elementos processadores acumulariam suas contribuições e as transferiam para o processador vizinho.

Neste trabalho os pontos da imagem a serem reconstruídos são aqueles que se encontram sobre um reticulado retangular, tornando a retroprojeção orientada somente para estes pontos. Observe-se que o uso deste reticulado permite explorar as propriedades trigonométricas acima citadas, implicando na divisão do reticulado em octângulos. Para realizar os cálculos da retroprojeção foi utilizado uma topologia *malha* com oito elementos. Como resultado, o cálculo de alguns valores utilizados na retroprojeção para um ponto do reticulado é aproveitado para outros pontos, diminuindo-se o trabalho computacional. Para aumentar eficientemente o tamanho da topologia em malha é proposto acrescentar sempre um conjunto com oito elementos.

Levando em consideração as diversas topologias necessárias para atender eficientemente os algoritmos de processamento de sinal e a distribuição de dados entre os processadores nas diversas etapas de computação, o trabalho propôs a topologia MOT (*Modified Orthogonal Trees*). O MOT pode ser entendido como uma topologia reconfigurável capaz de realizar tanto estruturas em anel quanto em árvores e combinações destas.

Conforme abordado no Capítulo 1 de Introdução, onde é feita uma descrição do P^3E [8], diversos trabalhos foram baseados no modelo de multiprocessador P^3E , com o propósito de realizar o algoritmo de retroprojeção filtrada, entre eles tem-se:

4) Os Trabalhos de Shieh *et al.* [12, 14] estuda o uso de DSP como elemento de processamento para uma arquitetura derivada do P^3E para fazer retroprojeção filtrada para

feixes paralelos. Para se obter boa qualidade nas imagens reconstruídas é necessário utilizar interpolação durante o processo de retroprojeção. Neste trabalho são analisadas as seguintes técnicas de interpolação: NN (*nearest neighbor*), LI (*linear interpolation*) e LL (*line length*). No trabalho são confrontados dois modelos DSPs, bastante difundidos, que internamente trabalham em aritmética de ponto-fixado, o *TMS32020* e o *DSP16*. Para estes DSPs foram desenvolvidos os algoritmos de retroprojeção filtrada para feixes paralelos, onde o filtro foi implementado utilizando-se FFT com aritmética em ponto-flutuante. Assim, a filtragem das projeções mantém um nível de ruído baixo, evitando-se sucessivos arredondamentos, caso fosse utilizado aritmética de ponto fixo. As demais operações são feitas em ponto-fixado. Entre os resultados obtidos, mostrou-se que a qualidade da imagem reconstruída varia em função do tipo de interpolação e a razão entre o tamanho do pixel da imagem e a largura do detetor. No caso em que o tamanho do pixel é igual à largura do detetor, a imagem obtida a partir de projeção com interpolação NN apresentava artefatos e a imagem com projeção obtida com os outros dois métodos apresentaram resultados superiores a interpolação NN e de qualidade bem próxima.

5) A implementação de um circuito dedicado capaz de fazer a transformada direta da transformada de Radon e a operação de retroprojeção, para feixes paralelos, foi realizada por Agi *et al.* [13] com o propósito de ser o elemento processador de uma arquitetura pipeline semelhante ao P³E. O C.I. foi desenvolvido para fazer interpolação linear e trabalhar com aritmética de ponto-fixado e projeções paralelas. A concepção deste ASIC é baseada na premissa que os pontos, tanto da imagem como da projeções, são fornecidos com o formato vindo de um *raster-scan*. Neste formato os elementos que compõem a imagem (ou a projeção), são elementos de um vetor unidimensional sequenciados a partir da leitura de uma matriz, feita elemento a elemento de uma linha na ordem crescente e em seguida os elementos da linha abaixo. A partir do formato *raster scan* foi possível desenvolver equações recursivas para a transformada discreta direta de Radon e para a retroprojeção.

6) O IPE, de Agi *et al.* [15, 16] é um ASIC sucessor ao anterior que utiliza aritmética de ponto-fixado e trabalha com feixes divergentes ou feixes paralelo. A implementação do

algoritmo é feita através de acumuladores recursivos evitando-se ao máximo o uso de multiplicadores e divisores. Para computar s' e $1/U^2$, recursivamente, foi necessário expandir essas funções em séries de Taylor, em seus termos de 1ª e 2ª ordens. Os termos de duas variáveis da série de Taylor foram simplificados e tabelados por aproximações, uma vez que seus mapeamentos de forma direta teriam custos muito elevado. Assim, os valores de s' para se ter uma precisão aceitável necessitam que o valor de D , distância entre a fonte e o centro de rotação, seja grande. Um valor grande D implica num pequeno peso dos coeficientes de duas variáveis na solução do algoritmo. Logo, nesta solução para se ter imagens de melhor qualidade impõem-se restrições à geometria do sistema.

7) Lin e Jain [17, 18] propõem uma alternativa com maior precisão do que o IPE de Agi, trabalhando com feixes divergentes, usa também interpolação linear e recursividade em alguns passos do algoritmo. Esse trabalho foi desenvolvido utilizando-se a técnica MCM, a partir da associação de duas células básicas de alto desempenho: célula para funções não lineares (UNL) e célula com multiplicador e acumulador (UMSA). Uma UNL é capaz de realizar 4 funções não-lineares: raiz quadrada, recíproco, seno/coseno, arco-tangente. A realização de UNL é baseada na expansão de 2ª ordem da série de Taylor, para aproximar um polinômio que representa a função não linear desejada, porém utiliza interpolação de valores tabelados ao invés de valores exatos dos coeficientes dos termos da série, com o intuito de se diminuir o custo do tabelamento. Uma célula UNL é composta de memórias ROMs, 2 somadores/subtratores e um multiplicador rápido de 20x20 bits. As funções não lineares, $1/U^2$ e s' , são implementadas com a utilização de algumas dessas células. Esta solução permite uma precisão de 24 bits para o cálculo de s' , contudo requer o uso de duas células UNL. O custo de implementação de um módulo que realize retroprojeção para imagem com 512 x 512 pontos requer 3 UNLs e 11 UMSAs. Uma das principais vantagens deste trabalho é permitir num mesmo encapsulamento MCM a colocação de 8 módulos de reconstrução.

8) O trabalho de Rajan *et al* [19] descreve um sistema baseado num *cluster* formado por DSPs que trabalham internamente em aritmética de ponto flutuante, o ADSP21062, conectados em um barramento hierárquico. Os nós que formam o *cluster* estão conectados

numa topologia hipercúbica que utiliza portas de comunicação de alto desempenho existentes nesses DSPs. Neste sistema foi implementado o algoritmo de retroprojeção filtrada para feixes paralelos, com interpolação linear e Lagrange com 6 pontos. A filtragem foi feita utilizando-se FFT. Também foi implementada a reconstrução a partir da inversão direta Fourier, através do Teorema da Fatia, usando um método especial de interpolação. A implementação utilizando-se 8 DSPs para retroprojeção filtrada com interpolação linear tem um desempenho cerca de 100 vezes mais rápido do que a implementação numa estação de trabalho IBM-Risc6000/340.

9) O processamento paralelo do algoritmo de retroprojeção filtrada para feixes paralelos, utilizando convolução numa rede de estações de trabalho Sun, foi feito por Vista [20]. Neste trabalho foi experimentado o uso de até 4 estações utilizando-se a biblioteca MPI (*Message Passing Interface*) para troca de mensagens. Um programa executado sob MPI consiste de processos autônomos executando seus próprios códigos num estilo MIMD. O resultado do trabalho demonstra a alta eficiência atingida neste sistema multiprocessador com um aceleração de 95 % em relação a implementação seqüencial.

6.1 Discussão:

O trabalho proposto nesta tese a princípio pode ser classificado com uma variação da arquitetura P³E, que apresenta uma forma bastante conveniente de explorar o paralelismo do algoritmo de retroprojeção filtrada, isto é, usar varias unidades idênticas para realizar a retroprojeção tal como as variações do P³E, retira a realização da filtragem das unidades de retroprojeção. Entretanto, ao se explorar o paralelismo implícito na Aritmética de Resíduos, mostrou-se um lado ainda não explorado que permite um desempenho muito alto para o algoritmo, mesmo utilizando um número pequeno de unidades que realizam retroprojeção. Diferentemente de todas as aplicações dedicadas, que não tratam da filtragem, foi apresentada uma unidade para fazer a filtragem com alto desempenho, que possui aspectos ainda inovadores sob o ponto de vista da tecnologia FPGA. Mais ainda, os resultados são apresentados sob um enfoque atual que é Computação Configurável, o que permitirá a empreender no futuro novas configurações do sistema proposto para realizar, provavelmente com grande desempenho, outras etapas necessárias

ao processamento tomográfico. Além da filtragem, outra característica única proposta neste trabalho é o uso do algoritmo clássico SRT de divisão, em base 4 para FPGA, no cálculo de s' , que embora tenha um custo elevado, em LEs, e alta latência, não compromete as expectativas de alto desempenho da solução. E por fim, a utilização da recente técnica *Symmetric bipartite tables* para mapear a função $1/U^2$ com sucesso em FPGA mostra toda a potencialidade da solução em FPGAs proposta.

Tabela 6.1: Resumo das características de alguns dos principais trabalhos que exploram o paralelismo do algoritmo retroprojeção filtrada

Autor/ caracter	GROOT	ZAPATA	W.T.LIN	HINKLE (P ³ E)	SHIEH	AGI	AGI (IPE)	L.LIN	RAJAN	VISTA	Proposta
TOPOLOG.	MALHA	HIPERCÚB.	ANEL/ ÁRVORE	PIPELINE	PIPELINE	PIPELINE	PIPELINE	PIPELINE	HIPERCUB.	DISTRIB.	PIPELINE
NÓ DE PROCESS.	MICRO- PROCESSA DOR	MICRO- PROCESSA DOR	MICRO- PROCESSA DOR	ALU, DSP, LUT.	DSP (ponto fixo)	DEDICADO	DEDICADO	DEDICADO	DSP (ponto flutuante)	MICRO- PROCESSA DOR	DEDICADO
FILTRAG.	FFT	FFT	FFT	CONVOL/ FFT	FFT	NÃO DISPON.	NÃO DISPON.	NÃO DISPON.	FFT	CONVOL.	CONVOL.
TIPO DE FEIXES	PARALELO	DIVERG.	DIVERG.	PARALELO	PARALELO	PARALELO	DIVERG./ PARALELO	DIVERG.	PARALELO	PARALELO	DIVERG./ PARALELO
ARITMÉTI- CA	PONTO- FLUT.	PONTO- FLUT.	PONTO- FLUT.	PONTO FIXO	PONTO FIXO	PONTO FIXO	PONTO FIXO	PONTO FIXO	PONTO- FLUT.	PONTO- FLUT.	Resíduos
INTERPOL.	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR/ VIZINHO PRÓXIMO	LINEAR/ VIZINHO PRÓXIMO	LINEAR	LINEAR	LINEAR/ LAGRANG.	LINEAR	LINEAR
FUNÇÕES S', 1/U ²	-	BIBILIO- TECA	BIBILIO- TECA	-	-	-	RECURSIV. TOTAL	Tabelas e multiplic.	BIBILIO- TECA	-	SRT Tabelas bipartidas

CAPÍTULO 7

CONCLUSÃO E TRABALHOS FUTUROS

Esta tese explora o paralelismo oferecido pela aritmética de resíduos e sistemas reconfiguráveis para diminuir o tempo para reconstruir imagens tomográficas. O uso de sistemas reconfiguráveis mostrou ser bastante eficaz para o algoritmo de retroprojeção filtrada por convolução produzindo imagens com 1 a 2 ordens de grandeza mais rápido que os computadores de uso geral.

O uso de aritmética de resíduo como uma alternativa ao uso de aritmética de ponto flutuante no algoritmo de Retroprojeção Filtrada permitiu obter imagens, do ponto vista numérico, muito próximas aos resultados obtidos por ponto-flutuante. Esses resultados são genéricos e podem ser estendidos a qualquer outra tecnologia diferente de FPGA com intuito de reduzir o tempo de reconstrução.

A implementação da solução para retroprojeção de feixes paralelos pode ser mapeada em pequenos dispositivos. No entanto, a implementação da solução para feixes divergentes requer o uso de dispositivos de tamanho médio, devido ao custo de implementação da operação de divisão para obtenção de s' que para atingir o máximo de desempenho deve estar mapeado em um único dispositivo.

O método *Symmetric Bipartite Tables* utilizado no cálculo $1/U^2$ permitiu ter uma noção da potencialidade desse método quando do seu uso em FPGAs. Acredita-se que em outras etapas do processamento tomográfico, como visualização volumétrica onde o cálculo de funções transcendentais é intenso, esse método possa ser amplamente aplicado.

Os algoritmos de recorrência de bits também podem ser aplicados na extração de raiz quadrada, que é por exemplo necessária em processamento de sinal ou imagem em aplicações como: filtragem adaptativa, computação de gradientes na detecção de bordas.

Logo, com algumas modificações no algoritmo SRT feito para divisão, pode-se implementar uma função que extrai a raiz quadrada.

O uso de convolução na filtragem utilizando uma subestrutura com 8 multiplicadores permitiu a realização de uma filtragem muito rápida, com tempo cerca de 10 vezes menor do que a retroprojeção. Logo, essa estrutura é adequada para o uso em sistemas com mais de uma unidade de retroprojeção. Encontra-se na literatura especializada sistemas baseados em FPGAs, com o mesmo modelo adotado neste trabalho, onde a FFT com 1024 pontos leva cerca de 10 milissegundos. Um dos modelos de DSP mais difundidos, o SHARC ADSP-21065L é capaz de fazer FFT de 1024 pontos em 0.31 milissegundos. Lembrando-se que a filtragem por FFT para projeção com 512 pontos deve ser realizada com 1024 pontos e que ainda é necessário fazer multiplicações e a FFT inversa para realizar a filtragem de uma projeção. Esta comparação superficial revela que o uso da convolução realizada em resíduo levando-se em consideração características dos coeficientes do filtro, permitiu realizar em FPGA uma solução tão rápida quanto o uso de FFT em DSP e ainda mais rápida do que FFT usando aritmética convencional em FPGA. Entretanto, em termos de custos o DSP hoje ainda é mais vantajoso.

Um importante ponto a ser investigado é a aplicação do tipo extensão da base RNS, isto é, aumentar a faixa dinâmica a partir de um determinado ponto do algoritmo empregado. À primeira vista, esta operação poderá ser feita após a projeção ser filtrada, uma vez que a faixa dinâmica, somente para a filtragem das projeções, requer cerca de 36 bits. Contudo, conforme o ocorrido na inversão resíduo binário, esta é também uma operação não modular que necessita de todos os módulos empregados na base RNS. Estima-se que o tempo gasto na extensão da base, deverá ser pequeno em relação ao tempo gasto para se fazer a retroprojeção. O emprego de escalonamento antes da expansão da base também deve ser investigado com o intuito de se reduzir a faixa dinâmica total.

O ideal em termos de custos de implementação é manter os resíduos trabalhando numa faixa dinâmica em que se utilize no máximo módulos com 5 bits para o resíduo, se possível evitando-se utilizar módulos com resíduos de largura maior ou igual 6 bits. O

custo do mapeamento das tabelas de índices, empregadas nas operações de multiplicação, cresce exponencialmente com o número de bits do resíduo.

A solução apresentada para reconstrução de feixes divergentes pode ser facilmente adaptada para o método de reconstrução em 3 dimensões conhecido *Cone Beam* pelo método de Feld Kamp [21] e pode vir a ser um imediato próximo trabalho. A implementação de outras etapas do processamento tomográfico como segmentação de imagens, visualização e filtragens também devem ter soluções mapeadas no sistema configurável proposto.

A incorporação de novos recursos ao FPGAs deve ser investigada e pode ter o seguinte impacto em relação a proposta deste trabalho:

i) Rápida reconfiguração [133], quando o dispositivo tem uma rápida mudança de contexto, isto é, o FPGA é capaz de guardar algumas configurações e trocá-la em poucos ciclos de *clock*, o imediato benefício é a diminuição do tempo de configuração e conseqüentemente um desempenho mais rápido da solução proposta. A redução do total de *hardware* é possível com alguma redução de desempenho, configurando-se por exemplo o sub-módulos que trabalham em paralelo e com aritmética de resíduo nas unidades de retroprojeção. A filtragem também poderia talvez se beneficiar pela personalização dos multiplicadores que poderiam ser modificados por novas configurações. Entretanto, os atuais modelos que permitem este tipo de configuração não são adequados para a tradução imediata da solução proposta.

ii) Alguns pesquisadores e fabricantes de FPGAs têm proposto a incorporação de multiplicadores de tamanho pequeno à estrutura interna do FPGA [119, 134, 135]. O uso de multiplicadores de poucos bits pode ser incorporado trabalhando em conjunto com o propósito de formar um multiplicador com um número de bits médio, por exemplo 25 bits. E esse multiplicador poderia ser utilizado na substituição do algoritmo SRT empregado na divisão, fazendo-se a divisão através da multiplicação do recíproco do divisor pelo dividendo e sendo a função recíproco mapeada pelo método *Symmetric Bipartite Tables*.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.
- [2] A.C. Kak e M. Slaney, *Principles of Computerized Tomographic Imaging*, IEEE PRESS, 1988.
- [3] G.T. Herman, *Image Reconstruction from Projections*, Academic Press, 1980.
- [4] H.P. Hiriyannaiah, “*X-ray Computed Tomography for Medical Imaging*”, IEEE Signal processing Magazine, Vol. 14, NO. 2, pp. 42-59, Março de 1997.
- [5] S.W. Roland, “*Computer Implementation of Image Reconstruction Formulas*”, pp. 9-78, em *Image Reconstruction from Projections, Implementation and Applications*, editor G.T. Herman, Topics in Applied Physics, VOL. 32.
- [6] R.N. Bracewell, “*Strip Integration in Radio Astronomy*”, Aust. J. Phys., 9, pp. 198-217.
- [7] P. Reimers, J. Goebbels, H.-P. Weise e K. Wilding, “*Some Aspects of Industrial Non-Destructive Evaluation by X and γ Ray Computed Tomography*”, Nuclear Instruments and Methods in Physics research, VOL. 221, pp. 201-206, 1984.
- [8] E.B. Hinkel, J.L.C. Sanz e A.K. Jain, “*P³E: New Life for Projection-Based Image Processing*”, Journal of Parallel and Distributed Computing, VOL.4, pp. 45-78, 1987.
- [9] A.J. Groot., S.G. Azevedo, D.J. Schneberk, E.M. Johansson e S.R. Parker, “*A Systolic Array for Efficient Execution of the Radon and Radon Inverse Transforms*”, SPIE VOL.975 Advanced Algorithms and Architectures for Signal Processing III, pp. 145-153, 1988.

- [10] W.T. Lin, C.Y. Ho e C.Y. Chin, "*Parallel Computation of Fan Beam Back-projection Reconstruction Algorithm in Computed Tomography*", pp. 113-127, em *Parallel Algorithms and Architectures for DSP Applications*-Kluwer Academic Publishers, editor Magdy Bayoumi, 1991.
- [11] E.L. Zapata, I. Benavides, F.F. Rivera, J.D. Bruguera, T.F. Pena e J.M. Carazo, "*Image reconstruction on hypercube computers: Application to electron microscopy*", *Signal Processing*, Vol.27, pp. 51-64, 1992.
- [12] E. Sheih, K.W. Current, P.J. Hurst e I. Agi, "*Radon Transform Computations using DSP Chips: An Evaluation and Comparison*", in *Proceedings of ISCAS'89*, pp. 1899-1902, Portland, OR, USA, 1989.
- [13] I. Agi, P.J. Hurst, E. Shieh, S. Azevedo e G. Ford, "*A VLSI architecture for high-speed image reconstruction: considerations for a fixed-point architecture*", *SPIE*, VOL.1246, *Parallel Architectures for image Processing*, pp. 11-24, 1990.
- [14] E. Sheih, K.W. Current, P.J. Hurst e I. Agi, "*High-Speed Computation of the Radon Transform and Backprojection Using an Expandable Multiprocessor Architecture*", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 2, NO.4, pp. 347-360, Dezembro de 1992.
- [15] I. Agi, P.J. Hurst e K.W. Current, "*A pipelined VLSI Chip Architecture for Real-Time Computed Tomography of Fan-Beam Data*" in *Proceedings of ISCAS'92*, San Diego, CA, USA, pp. 661-663, 1992 .
- [16] I. Agi, P.J. Hurst e K.W. Current, "*An Image Processing IC for Backprojection and Spatial Histogramming in a Pipelined Array*", *IEEE Journal of Solid-State Circuits* , VOL.28, NO.3, pp. 210-221, Março de 1993.
- [17] L. Lin e V.K. Jain, "*Parallel Architectures for Computing The Hough Transform and C.T. Image Reconstruction*", in *Proceedings of The International Conference on Application Specific Array Processors (ASAP'94)*, pp. 152-163.

- [18] L. Lin e V.K. Jain, "*Image Processing Using a Universal NonLinear Cell and the Warp wafer*", IEEE Trans. On Components, Packaging, and Manufacturing Technology- part B, vol. 17, NO. 3, pp. 342-349, Agosto de 1994.
- [19] K. Rajan, L.M. Patnaik e J. Ramakrishna, "*High-speed Image reconstuction based on CBP and Fourier Inversion Methods*", in Proceedings of High Performance Computing on the Information Superhighway, HPCA-ASIA 97, pp. 401-406.
- [20] R.C. Vista, "*Processamento Paralelo na Reconstrução de Imagens Tomográficas*", Tese de Mestrado, IPRJ-UERJ, Friburgo-RJ, 72p. 1999.
- [21] L.A. Feldkamp, L.C. Davis e J.W. Kress, "*Practical cone-beam algorithm*", Journal of Optical Society of. America, VOL. 1, NO. 6, pp. 612-619, Junho de 1984.
- [22] A.S. Trimberger, *Field-Programmable Gate Array Technology* , Kluwer Academic Publishers, 1994.
- [23] A. Sangiovanni-Vincentelli, A.E. Gamal e J. Rose, "*Synthesis Methods for Field Programmable Gate Arrays*", in Proceedings of the IEEE, vol. 81, NO.7, pp. 1057-1083, Julho de 1993.
- [24] P. Bertin, D. Roncin e J. Vuillemin, "*Introduction to Programmable Active Memories*", Resarch Report 3 , Digital Paris Research Laboratory, 1989
- [25] M. Shand e J. Vuillemin, "*Fast Implementations of RSA Cryptography*", in Proceedings of IEEE 11th Symposium on Computer Arithmetic, pp. 252-259, de 29 de junho a 2 de Julho 1993, Windsor, Canada.
- [26] M.E. Louie e M.D. Ercegovac, "*On digit-Recurrence Implementations for Field Programmable Gate Arrays*", in Proceedings of IEEE 11th Symposium on Computer Arithmetic, pp. 202-209, de 29 de junho a 2 de Julho 1993, Windsor, Canada.

- [27] J. Babb, R. Tessier e A. Agarwal, “*Virtual Wires: Overcoming pin Limitations in FPGA- based Logic Emulators*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp.142-151, 1993.
- [28] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E.Lam, P. Athanas, H. Silverman e S. Gosh, “*PRISM II: Compiler and Architecture*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 9-16, 1993.
- [29] H.J. Herpel, N. Wehn, M. Gasteier e M. Glesner, “*A Reconfigurable Computer for Embedded control Applications*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 111-120, 1993.
- [30] C. Iseli e E. Sanchez, “*Spyder: A Reconfigurable VLIW Processor using FPGAs*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 17-24, 1993.
- [31] J.M. Arnold, “*The Splash 2 software Environment*”, IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 88-93, 1993.
- [32] S. Casselman, “*Virtual Computing and the Virtual Computer*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 43-48, 1993.
- [33] D.T. Hoang, “*Searching Genetic Databases on Splash2*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 185-151, 1994.
- [34] M.E. Louis e M.D. Ercegovac, “*A Digit-Recurrence Square Root Implementation for Field Programmable Gate Arrays*”, in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM’93, pp. 178-183, 1993.

- [35] S. Monaghan e C.P. Cowen, "*Reconfigurable Multi-Bit Processor for DSP Applications in Statistical physics*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, pp. 103-110, 1993.
- [36] S.A. Cuccaro e C.F. Reese, "*The CM-2X: A Hybrid CM-2/Xilinx Prototype*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, pp. 121-130, 1993.
- [37] X.P. Ling e H. Amano, "*WASMII: A Data Driven Computer on a Virtual Hardware*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, pp. 33-42, 1993.
- [38] M. Dahl, J. Babb, R. Tessier, S. Hanono, D. Hoki e A. Agarwal "*Emulation of the Sparcle Microprocessor with the MIT Virtual Wires Emulation System*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, pp. 14-32, 1994.
- [39] J. Darnauer, P. Garay, T. Isshiki, J. Ramirez e W.W.M. Dai, "*A Field Programmable Multi-Chip Module (FPMCM)*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, pp. 1-10, 1994.
- [40] N.W. Bergmann e J.C. Mudge, "*Comparing the Performance of FPGA-Based Custom Computers with General-Purpose Computers for DSP Applications*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, pp. 164-171, 1994.
- [41] A.L. Abbot, P.M. Athanas, L. Chen e R.L. Elliot, "*Finding Lines and Building Pyramids with Splash 2*", in Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, pp. 155-163, 1994.
- [42] G. Goslin e B. Newgard, "*16-Tap, 8-bit FIR Filter Application Guide*", Xilinx Application Notes, Novembre de 1994.

- [43] J.D. Hadley e B.L. Hutchings, "*Design Methodologies for Partially Reconfigured systems*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 78-108, 1995.
- [44] M.J. Wirthlin e B.L. Hutchings, "*A Dynamic instruction Set Computer*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 95-108, 1995.
- [45] R. Amerson, R.J. Carter, W.B. Culbertson, P. Kuekes e G. Snider, "*Teremac-Configurable Custom Computing*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 32-38, 1995.
- [46] N.K. Ratha, A.K. Jain e D.T. Rover, "*Convolution on Splash 2*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 204-213, 1995.
- [47] N. Shirazi, A. Walters e P. Athanas, "*Quantitative Analysis of Floating Point Arithmetic on FPGA-Based custom computing machines*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 155-162, 1995.
- [48] H.A. Chow, H. Alnuweiri e S. Casselman, "*FPGA-Based Transformable Computers for fast Digital Signal processing*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'95, pp. 197-203, 1995.
- [49] P. Bakkes, J. du Plessis e B. Hutchings, "*Mixing fixed and Reconfigurable logic for array processing*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'96, pp. 118-125, 1996.
- [50] L. Louca, T.A. Cook e W.H. Johnson, "*Implementation of IEEE single precision Floating point addition and multiplication on FPGAS*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'96, pp.107-117, 1996.

- [51] J. Villasenor, B. Schoner, K.-N. Chia, C. Zapata, H.J. Kim, C. Jones S. Lansing e B.M. Smith, “*Configurable Computing Solutions for Automatic Target Recognition*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM’96, pp. 70-79, 1996.
- [52] Altera Corporation, “*Implementing FIR Filters in FLEX Devices*”, Application Notes 073, Janeiro de 1996.
- [53] K. Chapman, “*Building High Performance FIR filters using KCM’s*”, Xilinx Application Notes, Julho de 1996.
- [54] N.W. Bergmann, Y.Y. Chung e B.K. Gunther, “*Efficient Implementation of the DCT on Custom Computers*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM’97, pp. 244-245, 1997.
- [55] M. Rencher e B.L. Hutchings, “*Automated Target Recognition on Splash2*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM’97, pp. 192-200, 1997.
- [56] L. Mintzer, “*The Role of Distributed Arithmetic in FPGA-based Signal processing*”, Xilinx Application Notes, 1997.
- [57] R.D. Hudson, D.I. Lehn e P.M. Athanas, “*A run-time Reconfigurable Engine for image Interpolation*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM’98, pp. 88-95, 1998.
- [58] H. Lee e G.E. Sobelman, “*Digit-Serial DSP Library for Optimized FPGA Configuration*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM’98, pp. 322-323, 1998.

- [59] K.M. Gajjala Purna and D. Bathia, "*Temporal Partitioning and Scheduling for Reconfigurable Computing*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'98, pp. 329-330, 1998.
- [60] M.J. Wirthlin e B.L. Hutchings, "*Improving Functional Density using Run-Time Circuit Reconfiguration*", IEEE Trans. On VLSI, pp. 247-256, Junho de 1998.
- [61] A. Tsusui e M. Miyazaki, "*ANT-on-YARDS: FPGA/MPU Hybrid Architecture for Telecommunication Data Processing*", IEEE Trans. On VLSI, pp. 199-211, Junho de 1998.
- [62] B. Von Hezen, "Signal Processing at 250 MHz Using High-Performance FPGA's", IEEE Trans. On VLSI, pp-238-246, Junho de 1998.
- [63] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer e A. Perez-Uribe, "*Static and Dynamic Configurable Systems*", IEEE Trans. On Computer, VOL.48, NO. 6, pp.556-564, Junho de 1999.
- [64] N.S. Szabó e R.I. Tanaka, *RESIDUE ARITHMETIC AND ITS APPLICATIONS TO COMPUTER TECHNOLOGY*, McGraw-HILL New York, 1963.
- [65] W.K. Jenkins e B.J. Leon, "*The use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters*", IEEE Trans. on Circuits and Systems, VOL.24, NO.4, pp. 191-201, Abril de 1977.
- [66] G.A. Jullien, "*Residue Number Scaling and Other Operations using ROM Arrays*", IEEE Trans. on Computers, VOL.27, NO.4, pp. 325-336, Abril de 1978.
- [67] W.K. Jenkins, "*Recent Advances in Residue Number Techniques for Recursive Digital Filtering*", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL.27, NO.1, pp. 19-30, Fevereiro de 1979.

- [68] C.H. Huang e F.J. Taylor, "*A Memory Compression Scheme for Modular Arithmetic*", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL.27, NO.6, pp. 608-611, Dezembro de 1979.
- [69] M.H. Etzel e W.K. Jenkins, "*Redundant Residue Number Systems for Error Detection and Correction in Digital Filters*", IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL.28, NO. 5, pp 538-544, Outubro de 1980.
- [70] F.J. Taylor, "*An autoscale Residue Multiplier*", IEEE Trans. on Computers, VOL. 31, NO. 4, pp. 321-325, Abril de 1982.
- [71] F.J. Taylor, "*A VLSI Residue Arithmetic Multiplier*", IEEE Trans. on Computers, VOL. 31, NO. 6, pp. 540-546, Junho de 1982.
- [72] F.J. Taylor, "*Residue Arithmetic: A Tutorial with Examples*", IEEE COMPUTER, pp. 50-62, Maio de 1984.
- [73] M.A. Bayoumi, G.A. Jullien e W.C. Miller, "*A VLSI Implementation of Residue Adders*", IEEE Trans. on Circuit and Systems, VOL.34, NO.3, pp-284-288, Março de 1987.
- [74] M.A. Bayoumi, G.A. Jullien e W.C. Miller, "*A Look-up Table VLSI Design Methodology for RNS Structures used in DSP Applications*", IEEE Trans. on Circuit and Systems, VOL.34, NO.6, pp. 604-615, Junho de 1987.
- [75] M. Taheri, G.A. Jullien e W.C. Miller, "*High-Speed Signal Processing Using Systolic Array Over Finite Rings*", IEEE Journal On Selected Areas in Communications, VOL. 6, NO. 3, pp. 504-512, Abril de 1988.
- [76] P. Burrascano, G.C. Cardarilli, R. Lojacono, G. Martinelli e M. Salerno, "*Properties and Synthesis of RNS Digital Circuits*", IEEE Trans. on Circuit and Systems, VOL. 37, pp. 903-911, Julho de 1990.

- [77] G. Alia e E. Martinelli, "*A VLSI Modulo m Multiplier*", IEEE Trans. on Computers, VOL. 40, NO. 7, pp. 873-878, Julho de 1991.
- [78] M. Dugdale, "*VLSI Implementation of Residues Adders Based on Binary Adders*", IEEE Trans. on Circuits and Systems- II: Analog and Digital Signal Processing, VOL. 39, NO. 5, pp. 325-329, Maio de 1992.
- [79] T. Stouratis, S.K. Kim e A. Skvantzoz, "*Full Adder-Based Arithmetic Units for Finite Integer Rings*", IEEE Trans. on Circuit and Systems- II: Analog and Digital Signal Processing, VOL. 40, No. 1, pp. 740-745, Novembro de 1993.
- [80] M.K. Ibrahim, "*Novel digital filter implementations using RNS-binary arithmetic*", SIGNAL PROCESSING, VOL. 40, pp. 287-294, 1994.
- [81] E.D. Di Claudio, G. Orlandi e F. Piazza, "*A Systolic Redundant Residue Arithmetic Error Correction Circuit*", IEEE Trans. on Computers, VOL.42, No. 4, pp. 427-432, Abril de 1993.
- [82] G.A. Jullien, "*Implementation of Multiplication, Modulo a Prime Number, with Applications to Number theoretic Transforms*", IEEE Trans. On Computers, Vol. 29, NO. 10, pp. 899-905, Outubro de 1980.
- [83] D. Radhakrishnan e Y. Yuan, "*Novel Approaches to Design of VLSI RNS Multipliers*", IEEE Trans. on Circuit and Systems- II: Analog and Digital Signal Processing, VOL. 39, NO. 1, pp. 52-57, Janeiro de 1992.
- [84] V. Piuri, M. Berziera, A. Bisaschi e A. Fabi, "*Residue Arithmetic For a Fault-Tolerant Multiplier: The Choice of the Best triple of Bases*", Microprocessing and Microprogramming, VOL. 20, pp. 15-24, 1987.
- [85] A. Wrzyszc, D. Caban e E.L. Dagless, "*Design of a Discrete Cosine Transform Circuit Using the Residue Number System*", in Proceedings of European Conf. Design Automation, pp. 584-588, Paris, de 22 a 25 de Fevereiro, 1993.

- [86] S.J. Piestrak, "*Design of Residue Generators and Multioperand Modular Adders Using Carry Save Adders*", IEEE Trans. On Computer, VOL.43, NO. 1, pp. 68-77, Janeiro de 1994.
- [87] F.J. Taylor e A.S. Ramnarayanan, "*An Efficient Residue-to-Decimal Converter*", IEEE Trans. on Circuit and Systems, VOL.28, NO. 12, pp. 1164-1169, Dezembro de 1981.
- [88] M.A. Soderstrand, C. Vernia e J.-H. Chang, "*An Improved Residue Number System Digital-to-Analog Converter*", IEEE Trans. on Circuit and Systems, VOL. 30, pp. 903-907, Dezembro de 1983
- [89] G. Alia, F. Barsi e E. Martinelli, "*A Fast VLSI Conversion between binary and residue systems*", Information Processing Letters, VOL.18, NO.3, pp. 141-145, Março de 1984.
- [90] C.H. Huang, "*A Fully Parallel Mixed-Radix Conversion Algorithm for Residue Number Application*", IEEE Trans. On Computers, Vol. 32, No. 4, pp. 398-402, Abril de 1985.
- [91] T.V. Vu, "*Efficient Implementations of the Chinese Remainder Theorem for Sign Detection and Residue Decoding*", IEEE Trans. On Computers, Vol. 34, No. 7, pp. 646-651, Julho de 1985.
- [92] A.P. Shenoy e R. Kumaresman, "*Residue to Binary conversion for RNS Arithmetic using only modular look-up tables*", IEEE Trans. on Circuit and Systems, VOL. 35, NO. 9, pp. 1158-1162, Setembro de 1988.
- [93] K.M. Ibrahim e S.N. Saloun, "*An Efficient Residue to Binary Converter Design*", IEEE Trans. on Circuits and Systems, VOL.35, pp. 1156-1158, Setembro de 1988.

- [94] S. Andraos e H. Ahmad, "A New Efficient Memoryless residue to Binary Converter", IEEE Trans. on Circuits and Systems, VOL.35, pp. 1441-1444, Novembro de 1988.
- [95] C.N. Zhang, B. Shirazi e D.Y.Y. Yun, "Parallel Designs for Chinese Remainder Conversion", in Proceedings of International Conference Parallel Processing, pp. 557-559, 1988.
- [96] R.M. Capocelli e R. Giancarlo, "Efficient VLSI Networks for Converting an Integer from Binary System to Residue Number System and Vice Versa", IEEE Trans. on Circuits and Systems, VOL.35, NO. 11, pp. 1425-1430, Novembro de 1988.
- [97] A.P. Shenoy e R. Kumaresman, "Fast Base Extension Using a Redundant Modulus in RNS", IEEE Trans. on Computers, VOL.38, No. 2, pp. 292-297, Fevereiro de 1989
- [98] S.J. Meehan, S.D. O'neil e J.J. Vaccaro, "An Universal Input and Output RNS Converter", IEEE Trans. on Circuits and Systems, VOL.37, No.6, pp 799-803, Junho de 1990.
- [99] G. Alia e E. Martinelli, "VLSI Binary-Residue Converters for Pipelined Processing", The Computer Journal, VOL. 33, NO. 5, pp. 473-475, 1990.
- [100] A.B. Premkumar, "An RNS to Binary Converter in $2n+1$, $2n$, $2n-1$ Moduli Set", IEEE Trans. on Circuits and Systems- II: Analog and Digital Signal Processing, VOL. 39, No. 7, pp 480-482, Julho de 1992.
- [101] F. Pourbigharaz e H.M. Yassine, "A Signed-digit Architecture for Residue to Binary Transformation", IEEE Trans. On Computers, Vol. 46, NO. 10, pp. 1146-1150, Julho de 1997.

- [102] B.D. Tseng, G.A. Jullien, W.C. Miller, “*Implementations of FFT Structures using the Residue Number System*”, IEEE Trans. on Computers, VOL.28, NO. 11, pp. 831-844, Novembro de 1979.
- [103] A.Z. Baraniecka e G.A. Jullien, “*Residue Number System Implementation of Number Theoretic Transforms in Complex Residue Rings*”, IEEE Trans. on Acoustics, Speech, and Signal Processing, VOL. 28, NO. 3, pp. 285-291, Junho de 1980
- [104] J.V. Krogmeier e W.K. Jenkins, “*Error Detection and Correction in Quadratic Residue Number Systems*”, in Proceedings of 26 Midwest Symp. Circuit Syst., pp. 408-411, Puebla, Mexico, Agosto de 1983.
- [105] F.J. Taylor, G. Papadourakis, A. Skavantzios e A. Stouraitis, “*A Radix-4 FFT Using complex RNS Arithmetic*”, IEEE Trans. On Computer, VOL.34, NO. 6, pp. 573-576, Junho de 1985.
- [106] L. Maltar C.B., F.M.G. França, V.C. Alves e C.L. Amorim, “*Implementation of RNS addition and RNS multiplication into FPGAs*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines FCCM'98, pp. 330-331, 1998.
- [107] S. Brown e J. Rose, “*FPGA and CPLD Architectures: A Tutorial*”, IEEE Design & Test of Computers, pp. 42-57, Verão de 1996.
- [108] I-Cube, Inc. <http://www.icube.com/prodpage.html>
- [109] Aptix, Field programmable interconnect data book.
- [110] D. Bursky, “*Efficient RAM-Based FPGAs Ease System Design*”, Electronic Design, pp. 53-62, 22 de Janeiro de 1996.

- [111] B. Dipert, “*Shattering the programmable logic speed barrier*”, EDN Magazine, pp. 37-60, 22 de Maio de 1997.
- [112] K.M. Gajjala Purna e D. Bathia, “*Temporal Partitioning and Scheduling for Reconfigurable Computers*”, IEEE Trans. On Computers, VOL.48, NO. 6, pp. 579-590, Junho de 1999.
- [113] J.M.P. Cardoso e M.P. Véstias, “*Architectures and Compilers to Support Reconfigurable Computing*”, CROSSROADS The ACM Student Magazine, pp. 15-21, primavera de 1999.
- [114] A.R. Omondi, *Computer Arithmetic Systems*, Prentice Hall, 1994.
- [115] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, 1993.
- [116] D. Goldberg, “*What Every Computer Scientist Should Know About Floating-Point Arithmetic*”, ACM Computing Surveys, Vol. 32, No. 1, pp. 5-48, Março de 1991.
- [117] S.D. Haynes e P.Y.K. Cheung “*A Reconfigurable Multiplier Array for Video Image Processing Tasks, Suitable for Embedding in an FPGA Structure*”, in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'98, pp. 226-234, 1998.
- [118] R. Hartley e P. Corbett, “*Digit-Serial Processing Techniques*”, IEEE Trans on Circuits and Systems, Vol. 37, No. 6, pp. 707-719, Junho de 1990.
- [119] S.C. Coutinho, *Números Inteiros e Criptografia RSA*, Rio de Janeiro, IMPA/SBM, 1997.
- [120] Altera Corporation, “*Ripple-carry Adders in FLEX 8000 Devices*”, Application Brief 118, Maio de 1994.

- [121] Altera Corporation, Design with the MAX-plus II software using AHDL.
- [122] L. Maltar C.B., F.M.G. França, V.C. Alves e C.L. Amorim, “*Reconfigurable Hardware for Tomographic Processing*”, in Proceedings of SBCCI’98 XI Brazilian Symposium on Integrated Circuit Design, pp. 19-24, 1998.
- [123] S.F. Oberman and M. J. Flynn, “*Division Algorithms and Implementations*”, IEEE Trans. On Computers, vol.46, no. 8, pp. 833-854, Agosto de 1997.
- [124] M.D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Boston: Kluwer Academic, 1994.
- [125] M.J. Schulte e J.E. Stine, “*Symmetric Bipartite Tables for Accurate Function Approximation*”, in Proceedings of IEEE 13th Symposium on Computer Arithmetic, pp. 174-183, de 6 a 9 de Julho 1997, Asilomar, CA ,USA.
- [126] M.J. Schulte e J.E. Stine, “*Accurate Function Approximation by Symmetric Table Look and Addition*”, in Proceedings of IEEE International Conference on Application Specific Systems, Architectures, and Processors (ASAP’97), pp.144-153, 1997.
- [127] M.J. Schulte e J.E. Stine, “*Approximating Elementary Functions with Symmetric Bipartite Tables*”, IEEE Trans. On Computers, Vol. 48, pp. 842-847, Agosto de 1999.
- [128] H. Hassler e N. Takagi, “*Function Evaluation by Table Look-up and Addition*”, in Proceedings of IEEE 12th Symposium on Computer Arithmetic, pp. 10-16, de 19 a 21 de Julho 1995, Bath, Inglaterra
- [129] D.D. Sarma e D.W. Matula, “*Faithful Bipartite ROM Reciprocal Tables*”, in Proceedings of IEEE 12th Symposium on Computer Arithmetic, pp. 17-28 ,de 19 a 21 de Julho 1995, Bath, Inglaterra

- [130] W.F. Wong e E. Goto, "*Fast Evaluation of the Elementary Functions in Single Precision*", IEEE Trans. On Computers, Vol. 44, pp. 453-457, Março de 1995.
- [131] Xilinx Corporation, "*Virtex Configuration Architecture Advanced Users Guide*", Application Notes XAPP151, Maio de 2000.
- [132] L. Maltar C.B., F.M.G. França, V.C. Alves e C.L. Amorim, "*An FPGA-based Fan Beam Image Reconstruction Module*", in Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'99, 1999.
- [133] "*Dynamically Reconfigurable Devices & Technology*", in <http://dec.bournemouth.ac.uk/drhwh-lib/technology.html>, Julho de 1999.
- [134] S.D. Haynes, A.B. Ferrari e P.Y.K. Cheung "*Algorithms and Structures for Reconfigurable Multiplication Units*", in Proceedings of SBCCI'98 XI Brazilian Symposium on Integrated Circuit Design, pp. 13-18, 1998.
- [135] K. Rath, S. Tangirala, P. Friel, S. Vayuvegula, P. Balsara, J. Flores e J. Walley, "*Reconfigurable Array Media Processor (RAMP)*", in Proceedings of PACT'99 Workshop on Reconfigurable Computing, Newport Beach, CA, USA, Outubro de 1999.
- [136] D.E. Atinks, "*Higher Radix Division Using Estimates of the Divisor and Partial Remainders*", IEEE Trans. On Computers, Vol. 17, NO. 10, pp. 925-934, Outubro de 1968.
- [137] G.S. Taylor, "*Radix 16 SRT Dividers with Overlapped Quotient Selection Stages*", in Proceedings of IEEE 7 Symposium on Computer Arithmetic, pp. 64-81, 1985.
- [138] J. Fandrianto, "*Algorithm for High Speed Share Radix 4 Division and Radix 4 Square-Root*", in Proceedings of IEEE 8th Symposium on Computer Arithmetic, pp. 73-79, 1987.

- [139] M.L. Anido, "*Design and Implementation of a RISC Processor for Computer Image Generation Geometric Computation*", Ph. D. Thesis, University of Southampton, 1990.
- [140] D.M. Mandelbaum, "*Some Results on a SRT Type Division Scheme*", IEEE Trans. On Computers, vol.42, no. 1, pp. 102-106, Janeiro de 1993.
- [141] P. Montuschi e L. Ciminiera, "*Over-Redundant Digit Sets and the Design of Digit-by-Digit Division Units*", IEEE Trans. On Computers, VOL. 43, NO.. 63, pp. 269-277, Março de 1994.
- [142] H.R. Srinivas e K.K. Parhi, "*A Fast Radix-4 Division Algorithm and its Architecture*", IEEE Trans. On Computers, vol.44, no. 6, pp. 826-831, Junho de 1995.
- [143] N. Burgess e T. Willians, "*Choices of Operand Truncation in the SRT Division Algorithm*", IEEE Trans. On Computers, vol.44, no. 7, pp. 933-938, Julho de 1995.
- [144] E.M. Schartz, L. Sigal e T.J. McPherson, "*CMOS floating-point unit for the S/390 Parallel Enterprise Server G4*", IBM J.", IBM J. RES. DEVELOP. , VOL. 41, NO. 4/5, pp.475-488, Julho/Setembro de 1997.

Divisão SRT

A divisão do tipo SRT, cujo nome é formado a partir das iniciais dos nomes dos pesquisadores Sweeney, Robertson, Tocher que desenvolveram simultaneamente e independentemente o método, pertence a classe dos algoritmos de divisão por recorrência de dígitos. Algoritmos baseados em recorrência de dígitos usam métodos subtrativos para calcular um dígito do quociente por iteração. Os algoritmos baseados em recorrência podem ser agrupados em 2 categorias: com restauração ou sem restauração do quociente. Os algoritmos que utilizam restauração se comportam como os métodos manuais de divisão quando da escolha do quociente: testando vários valores para o quociente, os quais formam os múltiplos do divisor. Esses valores são subtraídos do dividendo sendo escolhido aquele que produz o menor resto cujo sinal deve ser igual ao do dividendo. Se o resto desta subtração possuir sinal diferente do resto parcial deslocado, implica que o valor escolhido para o quociente não é adequado, pois o múltiplo do divisor tem seu valor em módulo maior do que o valor do qual havia sido subtraído e portanto o valor correto do quociente deverá ser menor do que o valor testado. Portanto, o quociente deve ser restaurado e devem-se testar outros valores até atingir o dígito correto. Nos algoritmos sem restauração não existem ciclos de restauração e, portanto, estes gastam menos tempo para realizar divisão. A invés de testar todos os múltiplos do divisor no processo de escolha do dígito do quociente, é feita uma consulta a uma tabela cuja entrada são alguns dos bits mais significativos do divisor e do resto parcial deslocado (ou dividendo no primeiro passo). A partir do quociente obtido da tabela seleção gera-se um múltiplo do divisor, o qual deverá ser subtraído do resto parcial deslocado. É necessário que os dígitos gerados para o quociente possam assumir individualmente tanto valores positivos como negativos. Desta forma, pequenos erros introduzidos numa iteração, isto é, valores cujo restos parciais possuem em módulo valor pequeno e tenham sinais diferente do dividendo, não invalida o dígito selecionado e esses erros serão corrigidos nas iterações subsequentes. A expressão da recorrência pode ser vista como:

dividendo = quociente x Divisor + resto

ou resto = dividendo - quociente x Divisor ou :

$$P_i = r P_{i-1} - q_i \text{DIV} \quad (1)$$

onde

$$r = 2^b = \text{base}$$

$$r P_0 = \text{dividendo}$$

P_i = resto parcial no i -ésimo ciclo

$r P_{i-1}$ = resto parcial deslocado no i -ésimo ciclo

q_i = i th dígito do quociente

DIV = divisor

Após a n -ésima iteração o quociente final é dado por:

$$Q[n] = \sum_{i=0}^{n-1} q_i r^{i-1} \quad (2)$$

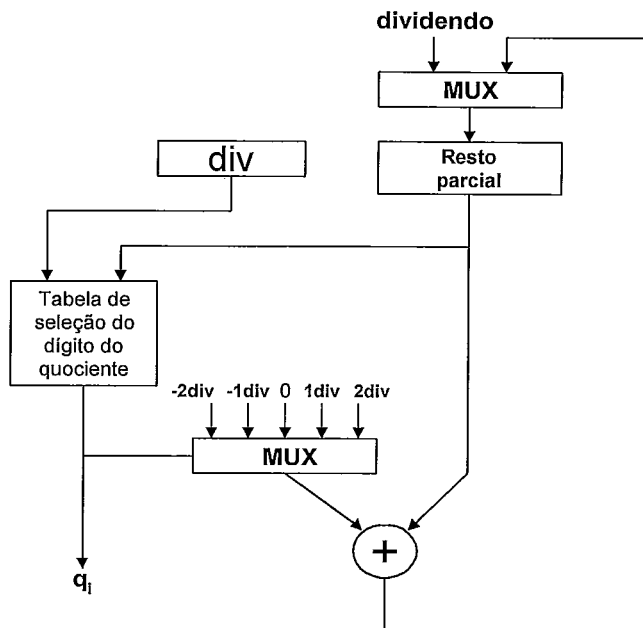


Fig. A1: Modelo simplificado sequencial de divisão SRT com base 4 de redundância mínima.

Um ciclo do algoritmo SRT é composto por 3 etapas:

- i) Seleção do quociente, a partir do divisor e do resto parcial do ciclo anterior;
- ii) Formação dos múltiplos do divisor ;
- iii) Cálculo do resto parcial;

A Figura A.1 apresenta um diagrama simplificado do algoritmo SRT, em base 4, onde cada dígito é obtido seqüencialmente. Diversas técnicas foram propostas para incrementar o desempenho da divisão SRT[136-144], por exemplo: uso de *carry save adders* no cálculo do resto parcial implicando em soma mais rápida; replicação de *hardware* quando usar base numéricas pequenas; sobreposição dos estágios do algoritmo; uso combinado de pequenas bases numéricas com altas bases numéricas, entre outras.

A escolha da base numérica indica o número de bits a ser produzido por iteração. Por exemplo: utilizando-se base 2 produz 1 bit por iteração, enquanto utilizando-se base 4 se produzem 2 bits do quociente por iteração. Logo, o número de ciclos diminui a medida em que se trabalha com base maiores. Entretanto, isto não significa uma diminuição proporcional no tempo de execução do algoritmo, pois aumenta-se a complexidade do circuito que gera a seleção do quociente e portanto aumenta-se o tempo de execução de um ciclo.

O algoritmo SRT necessita de uma representação dos dígitos do quociente que permita ser um a um individualmente sinalizado. Uma forma não convencional para representar números de base fixa adequada para divisão e também para multiplicação é conhecida com sistema *signed digit*.

A representação convencional de sistemas de base fixa, r , utiliza um conjunto restrito de dígitos igual a $\{0, \dots, r-1\}$. Exemplos: um sistema base 2 possui o conjunto de dígitos formado por dois elementos $\{0, 1\}$; um sistema de base 10 seu conjunto de dígitos é $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ com dez elementos. Num sistema SD para uma base r , onde seus dígitos possuem sinal, o conjunto de dígitos com número máximo de elementos é dado por

{ (r-1), (r-2), ..., 1, 0, 1, ... (r-1) }, onde a notação sublinhada significa sinal negativo. Logo, um número formado por 2 ou mais dígitos num sistema SD pode possuir mais de uma forma de representá-lo, isto é, apresenta redundância de representação. Exemplo: num sistema de base 2, com representação SD, e com 2 dígitos, o número 1 poderá ter as seguintes representações: 01 ou 11. A conversão de um sistema SD para um sistema sem redundância é feita diminuindo-se a parte negativa da positiva, logo $1\bar{1} = 10_b - 01_b = 2 - 1$. O nível de redundância pode ser estabelecido de forma a restringir o conjunto de dígitos na representação SD:

$$\{ (\underline{a}), (\underline{a-1}), \dots, 1, 0, 1, \dots, a \} \text{ com } \lceil (r-1)/2 \rceil \leq a \leq r-1 \quad (3)$$

Estabeleceu-se a denominação fator de redundância ρ , definido por:

$$\rho = a / (r-1) \quad (4)$$

No algoritmo SRT a utilização de redundância na representação do quociente estabelece uma relação que limita o valor para os restos parciais:

$$| P_i | < \rho \times \text{DIVISOR} \quad (5)$$

A construção de diagramas P-D facilita a visualização da função de seleção dos dígitos do quociente. Esse diagrama fornece as regiões onde dígito do quociente pode ser seguramente escolhido de forma que o resto parcial deslocado se mantenha dentro do limite estabelecido por (5). Cada região está compreendida entre o limite superior e inferior, para um dado valor de K para quociente:

$$\text{limite_superior} = (\rho + K) \text{ divisor} \quad (6)$$

$$\text{limite_inferior} = (-\rho + K) \text{ divisor} \quad (7)$$

onde K é um dos possíveis dígitos que o quociente pode assumir.

As Figuras A.2 e A.3, apresentam uma parte do diagrama P-D, resto parcial \times divisor, para o caso em que o divisor se encontra no intervalo $(0.5, 1]$ e os dígitos pertencem ao conjunto $\{-2, -1, 0, 1, 2\}$, obtido da redundância mínima para base 4. As coordenadas DIVISOR e Resto Parcial deslocado estão truncados estabelecendo-se regiões de incertezas no gráfico P_D. Verifica-se a existência de áreas em que os valores corretos para o dígito do quociente se sobrepõe, logo dois valores para o dígito atendem corretamente o critério de limitação para resto parcial. Se as regiões de incertezas são muito grandes, elas podem se espalhar por uma área em que não é possível ter um único valor para o quociente, como pode ser visto na Figura A.2, onde duas regiões de incerteza não retornam um valor único para o dígito do quociente. Pode-se citar um critério utilizado para verificar se o truncamento escolhido é adequado: pega-se o limite superior e inferior da região de incerteza que visualmente é dado pelo canto superior esquerdo e canto inferior direito no caso de resto parcial positivo. Entra-se o gráfico P-D ambos devem retornar com o mesmo valor para q . Nos casos em que houver superposição de regiões são retornados dois valores para um dos limites, nesse caso um dos valores deverá ser igual ao do outro canto. Testam-se todas as regiões de incerteza do gráfico P-D. Havendo uma única falha, significa que o intervalo não foi suficientemente truncado, devendo-se aumentar o número de bits do truncamento. Na verdade, o gráfico P_D é apenas uma ferramenta visual que facilita o entendimento do funcionamento da criação da função seleção dígito do quociente. Na prática, o número de entradas na tabela pode ser calculado numericamente por métodos de tentativas e erros [115].

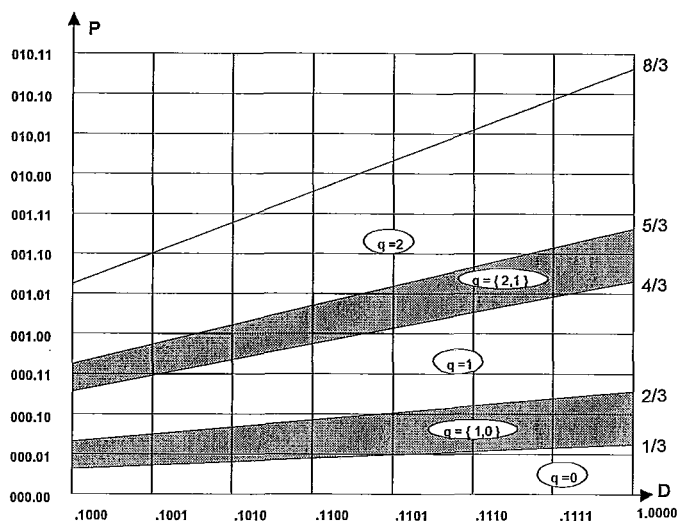


Fig. A.2: Diagrama P x D, com número de bits insuficiente para determinação do quociente.

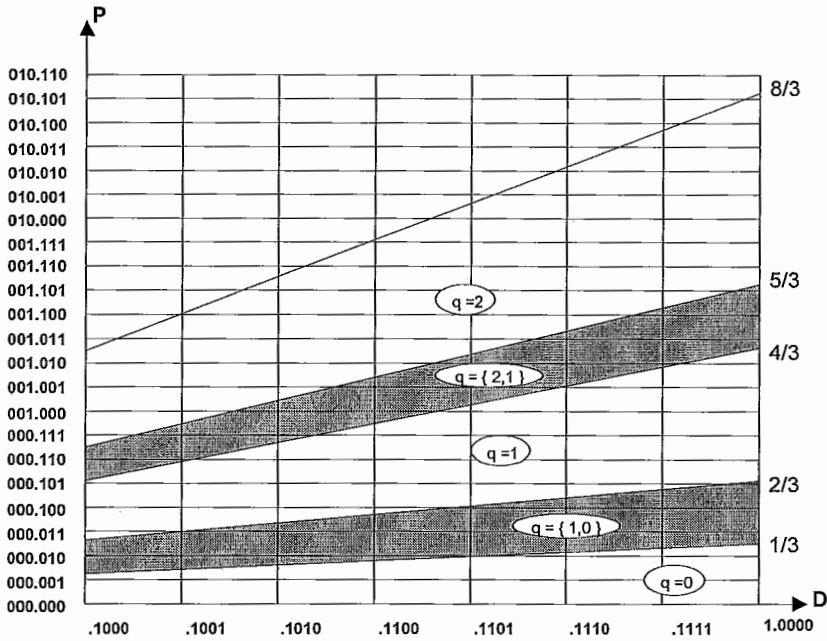


Fig. A.3: Diagrama P x D, com 6 bits para o resto parcial e 4 bits para o divisor, sem problemas para determinar o valor do quociente.

Demonstra-se que a utilização de um alto nível de redundância implica numa diminuição da quantidade de bits necessário para o truncamento do DIVISOR e do Resto Parcial. Diminuindo a complexidade para se implementar a função de seleção de dígitos do quociente. No entanto, aumenta-se a complexidade para gerar os múltiplos do divisor. Por exemplo: a geração dos múltiplos do divisor para o caso de base 4, com redundância mínima, pode ser feita por complemento do divisor, deslocamentos do divisor ou de seu complemento. Utilizando-se base 4 com redundância máxima, o conjunto de dígitos permitidos é $\{-3, -2, -1, 0, 1, 2, 3\}$, onde a geração $(\pm) 3 \times \text{DIVISOR}$ é mais complexa do que simples deslocamentos de $(\pm) \text{DIVISOR}$.

APÊNDICE B

Introdução ao método *Symmetric Bipartite tables*

Este método é baseado na aproximação da função desejada por dois termos série de Taylor, expandindo-a em torno de um valor próximo aos valores dos operandos de entrada, cuja parte dos bits que o compõe é conhecido e fixo. Assim, diminuindo a quantidade de memória necessária para tabelar os dois termos da série de Taylor. Inicialmente dividi-se o operando em três partes, fig. B.1, de forma que um operando x de n bits pode ser visto como $x = x_0 + x_1 + x_2$ e $n = n_0 + n_1 + n_2$. Supondo x normalizado no intervalo $[0,1)$ tem-se:

$$\begin{aligned}
 0 \leq x_0 &\leq 1 - 2^{-n_0} \\
 0 \leq x_1 &\leq 2^{-n_0} - 2^{-(n_0 + n_1)} \\
 0 \leq x_2 &\leq 2^{-(n_0 + n_1)} - 2^{-(n_0 + n_1 + n_2)}
 \end{aligned} \tag{b.1}$$

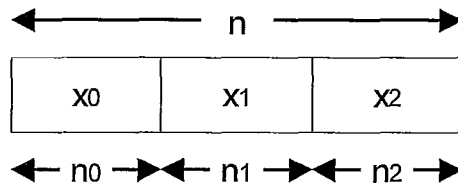


Fig: B.1: divisão do operando em 3 partes

Seja $f(x) = f(x_0 + x_1 + x_2)$ a função a ser aproximada por:

$$f(x) \approx f(x_0 + x_1 + \delta_2) + f'(x_0 + x_1 + \delta_2)(x_2 - \delta_2) \tag{b.2}$$

A equação b.2 é obtida aproximando-se a série de Taylor, em dois termos, na vizinhança do operando de valor $x_0 + x_1 + \delta_2$, onde δ_2 é um número muito pequeno e seu valor é igual a $2^{-(n_0 + n_1 + 1)} - 2^{-(n_0 + n_1 + n_2 + 1)}$, que é exatamente a metade do intervalo permitido para x_2 . Logo a primeira parcela da equação só necessita de $n_0 + n_1$ bits de

entrada. Substituindo x_1 por δ_1 de valor igual $2^{-(n_0+1)} - 2^{-(n_0+n_1+1)}$, que é a metade do intervalo permitido para de x_1 , tem-se a nova aproximação dada por :

$$f(x) \approx f(x_0 + x_1 + \delta_2) + f'(x_0 + \delta_1 + \delta_2)(x_2 - \delta_2) \quad (b.3)$$

ou

$$f(x) \approx a_0(x_0, x_1) + a_1(x_0, x_2)$$

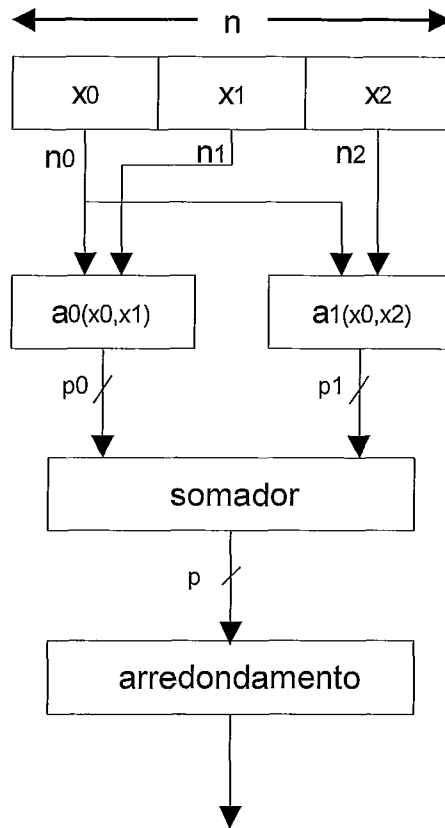


Fig. B2: Diagrama de blocos para evolução de uma da função usando 2 tabelas e um somador.

A figura B.2 representa um diagrama de blocos para evoluir a equação b.3, onde p_0 e p_1 são os números de bits necessários para representar, respectivamente, as saídas das tabelas de $a_1(x_0, x_1)$ e $a_1(x_0, x_2)$. Seja f^{max} o maior valor que a derivada da função pode assumir dentro do intervalo normalizado da entrada, dos n bits necessários para representar a saída da função $a_1(x_0, x_2)$, aproximadamente os $n_0 + n_1 + 1 - \log_2(f^{max})$ mais significativos bits possuem valor iguais a zero quando a função tem valor positivo ou 1 quando a função tem valor negativo, assumindo a extensão do sinal. Esses valores zeros ou

uns necessitam apenas de 1 bit para representá-los, assim diminuindo-se a quantidade de bits a ser armazenado para a função $a_1(x_0, x_2)$. Isto ocorre porque a parcela $|x_2 - \delta_2|$ de $a_1(x_0, x_2)$ tem um valor muito pequeno, equação b.4, limitando o valor de $a_1(x_0, x_2)$, equação b.5, logo tem-se $p_0 > p_1$.

$$|x_2 - \delta_2| < 2^{-(n_0 + n_1 + 1)} \quad (\text{b.4})$$

$$|a_1(x_0, x_2)| < |f'(x_0 + \delta_1 + \delta_2)| 2^{-(n_0 + n_1 + 1)} \quad (\text{b.5})$$

Um aspecto importante para o tabelamento de $a_1(x_0, x_2)$ é a simetria de seus valores, permitindo a redução de metade da quantidade de memória para guardá-lo. Essa simetria pode ser observada pelo intervalo de valores permitidos na parcela $(x_2 - \delta_2)$. Aplicando-se estas reduções para o tabelamento de $a_1(x_0, x_2)$, o novo diagrama de blocos para este método pode ser visto na figura B.3.

Se é desejado que a aproximação da função tenha pf bits para a parte fracionária com erro menor do que 2^{-pf} . A parte fracionária das saídas das funções $a_0(x_0, x_1)$ e $a_1(x_0, x_2)$ deverão possuir $pf + g$ bits, onde os g bits são bits de segurança que limitam os erros de arredondamento. É sugerido, conforme usado em [129], arredondar os coeficientes de $a_0(x_0, x_1)$ com $pf + g$ bits e atribuir o valor zero ao $pf + g + 1$ -ésimo bit da parte fracionária; para $a_1(x_0, x_2)$ é sugerido truncar a parte fracionária com $pf + g$ bits e atribuir o valor 1 ao $pf + g + 1$ bit. Após a soma das parcelas vindas de $a_0(x_0, x_1)$ e $a_1(x_0, x_2)$, deve-se fazer um arredondamento para o número mais próximo com pf bits. Demonstra-se, que ao utilizar esse método de arredondamento é possível manter o erro controlado dentro do limite da unidade da última posição da representação do número (ULP), se a seguinte restrição deve ser atendida:

$$2n_0 + n_1 \geq pf - 1 + \log_2 \{ [|f''_{max}| \cdot (1 + 2^{-n_1})] / (1 - 2^{-g}) \} \quad (\text{b.6})$$

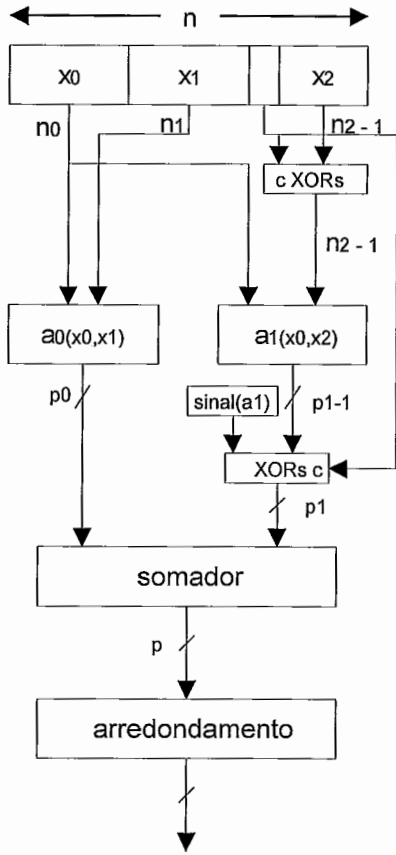


Fig. B.3: Diagrama em bloco para o método *Symmetric bipartite tables*.

APÊNDICE C

Neste Apêndice são apresentados a Transformada de Radon e o Teorema da Fatia, que são necessários na demonstração da obtenção das equações utilizadas na reconstrução de imagens para os algoritmos analíticos do tipo retroprojeção filtrada.

1. Transformada de Radon

A transformada de Radon de uma função $f(x,y)$ no plano $X-Y$ é definida como suas integrais de linha ao longo de uma eixo inclinado de um ângulo θ em relação ao eixo X a distância s da origem, Figura C.1, e tem denominação $g(s, \theta)$ e é matematicamente definida como:

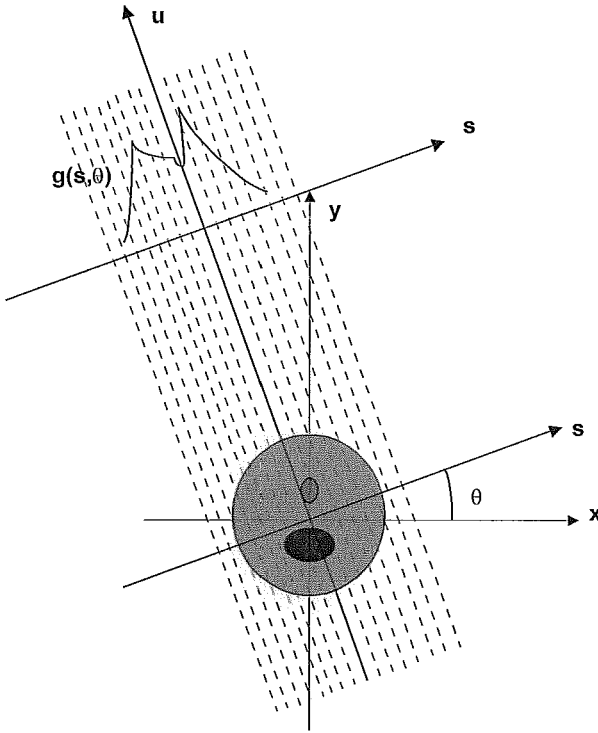


Figura C.1: Transformada de Radon para um ângulo θ

$$g(s, \theta) = \int_{-\infty}^{\infty} f(x,y) du \quad (c.1),$$

$$g(s, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) \delta(x \cdot \cos\theta + y \cdot \sin\theta - s) dx dy \quad (c.2)$$

$$-\infty < s < \infty, 0 \leq \theta < \pi$$

$$g(s, \theta) = \int_{-\infty}^{\infty} f(s \cdot \cos\theta - u \cdot \sin\theta, s \cdot \sin\theta + u \cdot \cos\theta) du \quad (c.3)$$

$$\begin{aligned} \text{onde, } s &= x \cdot \cos\theta + y \cdot \sin\theta & \text{ou } x &= s \cdot \cos\theta - u \cdot \sin\theta \\ u &= -x \cdot \sin\theta + y \cdot \cos\theta & y &= s \cdot \sin\theta + u \cdot \cos\theta \end{aligned}$$

A função $g(s, \theta)$ é uma projeção unidimensional de $f(x,y)$ relativo ao ângulo θ cujos valores se encontram sobre um sistema $s-u$ girado de θ em relação ao eixo $X-Y$. Cada ponto da função $g(s, \theta)$ é também conhecido como raio soma, pois representa o somatório ao longo de uma reta à distância s do sistema $s-u$.

2 Teorema da Fatia

A transformada de Fourier em 1 dimensão em relação ao eixo s da projeção $g(s, \theta)$ é igual a fatia central, deslocada de θ em relação ao eixo ω , da transformada de Fourier em duas dimensões da imagem $f(x,y)$:

$$g(s, \theta) \xrightarrow{\text{FOURIER}} G(\omega, \theta) \quad (c.4)$$

$$f(x,y) \xrightarrow{\text{FOURIER}} F(\omega_1, \omega_2) \quad (c.5)$$

$$G(\omega_1, \theta) = \int_{-\infty}^{\infty} g(s, \theta) e^{-j2\pi\omega_1 s} ds \quad (c.6)$$

Com as equações acima é possível demonstrar que, filtrando-se as projeções e retroprojetando-as, o resultado é uma imagem muito próxima da original.

Substituindo c.3 em c.6 tem se:

$$G(\omega_1, \theta) = \int_{-\infty}^{\infty} f(s \cdot \cos \theta - u \cdot \sin \theta, s \cdot \sin \theta + u \cdot \cos \theta) e^{-j2\pi\omega_1 s} ds du \quad (c.7)$$

Fazendo mudanças de coordenadas de (s, u) para (x, y) obtem-se:

$$G(\omega_1, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(x \cdot \omega_1 \cos \theta + y \cdot \omega_1 \sin \theta)} dx dy = F(\omega_1 \cos \theta, \omega_1 \sin \theta) \quad (c.8)$$

Aplicando-se a transformada inversa de Fourier em (c.8) tem-se:

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\omega_1, \omega_2) e^{j2\pi(\omega_1 x + \omega_2 y)} d\omega_1 d\omega_2 \quad (c.9)$$

substituindo por coordenadas polares: $\omega_1 = \omega \cos \theta$ e $\omega_2 = \omega \sin \theta$

$$d\omega_1 d\omega_2 = \omega d\omega d\theta$$

$$f(x, y) = \int_0^{2\pi} \int_{-\infty}^{\infty} F(\omega, \theta) e^{j2\pi\omega(x \cos \theta + y \sin \theta)} \omega d\omega d\theta \quad (c.10)$$

usando se a propriedade $F(\omega, \theta + \pi) = F(-\omega, \theta)$, para modificar os limites de integração:

$$f(x, y) = \int_0^{\pi} \int_{-\infty}^{\infty} |\omega| F(\omega, \theta) e^{j2\pi\omega(x \cos \theta + y \sin \theta)} d\omega d\theta \quad (c.11)$$

$$f(x, y) = \int_0^{\pi} \left\{ \int_{-\infty}^{\infty} |\omega| G(\omega, \theta) e^{j2\pi\omega(x \cos \theta + y \sin \theta)} d\omega \right\} d\theta \quad (c.12)$$

$$f(x, y) = \int_0^{\pi} Q(s, \theta) d\theta \quad (c.13)$$

$$\text{onde} \quad Q(s, \theta) = \int_{-\infty}^{\infty} |\omega| G(\omega, \theta) e^{j2\pi\omega s} d\omega \quad (\text{c.14})$$

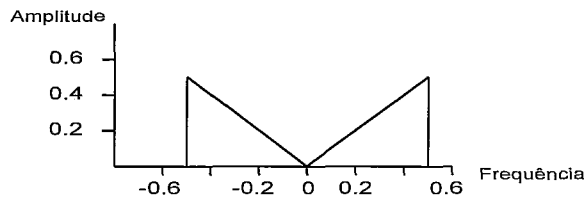
Substituindo $s = x.\cos\theta + y.\sin\theta$ em c.13

$$f(x,y) = \int_0^{\pi} Q(x.\cos\theta + y.\sin\theta, \theta) d\theta \quad (\text{c.15})$$

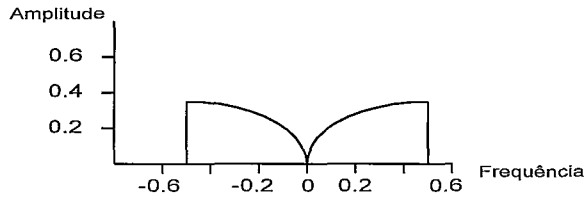
A equação c.15 representa o somatório de todas retroprojeções de $Q(s, \theta)$, projeção filtrada, assim cada o ponto no plano $X-Y$ que contém a imagem corresponde a um valor de $s = x.\cos\theta + y.\sin\theta$ para dado um valor de ângulo θ , recebendo uma contribuição da correspondente projeção filtrada.

Sendo ω a dimensão de frequência espacial, $Q(s, \theta)$ é chamada de projeção filtrada. Logo, a reconstrução de imagem pode ser feita primeiro filtrando-se cada projeção e depois reprojetoando-as sob o eixo $X-Y$.

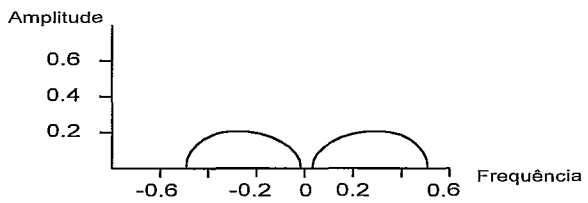
O filtro $|\omega|$ derivado na inversão da transformada de Radon acentua as altas frequências, o que pode resultar em amplificação de ruídos nos casos práticos. No sentido de limitar a resposta em frequência deste filtro, são utilizadas diversas funções janelas, $W(\omega)$, tal que o filtro a ser utilizado na filtragem das projeções tenha a seguinte representação genérica: $H(\omega) = |\omega|W(\omega)$. Entre as diversos modelo de filtros utilizados em tomografia os mais utilizados são: Ram_Lak, Shepp_Logan, Hamming, cujas resposta em frequência se encontram nas Figuras C.2 abaixo.



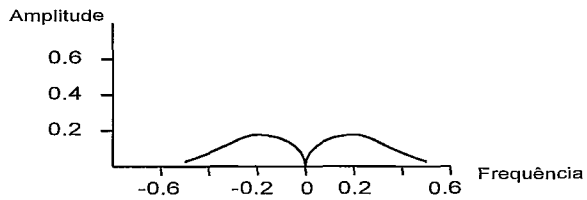
a) RAM-LAK



b) Shepp Logan



c) Coseno passa-baixa



d) Hamming genralizado

Figuras C.2: Resposta em frequência dos filtros utilizados no algoritmo de retroprojeção filtrada.