UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

FÁBIO PIRES ITTURRIET

Exploração Adaptativa de Paralelismo sob Restrições Físicas e de Tempo Real em Sistemas Embarcados Tolerantes a Falhas

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Microeletrônica

Prof. Dr. Luigi Carro
Orientador

Porto Alegre, Novembro de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Itturriet, Fábio Pires

Exploração Adaptativa de Paralelismo sob Restrições Físicas e de Tempo Real em Sistemas Embarcados Tolerantes a Falhas / Fábio Pires Itturriet – Porto Alegre: Programa de Pós-Graduação em Microeletrônica, 2012.

90p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica. Porto Alegre, BR – RS, 2012. Orientador: Luigi Carro.

1. Arquiteturas Adaptativas 2. Tolerância a Falhas 3. Sistemas Embarcados I. Carro, Luigi. II. Título.

AGRADECIMENTOS

Agradeço a Deus por ter me dado força e ânimo durante este período para exercer a dupla e árdua jornada de professor e mestrando.

Agradeço a minha família por sempre priorizar meus estudos.

Agradeço a minha companheira, Vanessa, por ter estado ao meu lado e compreendido meu constante mau humor, principalmente nos últimos meses.

Agradeço a todos os colegas de laboratório, em especial, aos colegas Ronaldo Ferreira e Gabriel Nazar pelas discussões técnicas que muito enriqueceram este trabalho e contribuíram para minha formação.

Agradeço a meu orientador Luigi Carro por ter confiado no meu potencial.

Agradeço a Capes pelo suporte financeiro a este trabalho.

SUMÁRIO

SUMÁRIO **5**
LISTA DE ABREVIATURAS E SIGLAS **7**
LISTA DE FIGURAS **9**
LISTA DE TABELAS **11**
RESUMO **12**
ABSTRACT **13**
1 INTRODUÇÃO **14**
1.1 Motivação **14**
1.1.1 Impactos na confiabilidade de circuitos associados com o *scaling* 14
1.1.2 Limitadores do consumo de potência para os próximos nós tecnológicos 17
1.1.3 Exploração Adaptativa de Paralelismo baseado na Banda de Memória 18
1.1.4 Exploração Adaptativa de Paralelismo aplicado em sistemas de tempo real 18
1.1.5 Proteção do algoritmo de Multiplicação de Matrizes 19
1.1.6 Utilização de Aceleradores de Hardware em aplicações Multimídia e de Tempo Real 20
1.2 Objetivos **21**
1.3 Organização do texto **22**
2 PRINCIPAIS FONTES DE RADIAÇÃO E SEUS EFEITOS NO SILÍCIO E CIRCUITOS ELETRÔNICOS **23**
2.1 Partículas *alpha* **23**
2.2 Raios Cósmicos **24**
2.2.1 Nêutrons de alta energia 25
2.2.2 Nêutrons de baixa energia 25
2.3 Interação de partículas *Alpha* e Nêutrons com cristais de silício **26**
2.4 Impacto dos efeitos transientes nos circuitos eletrônicos **27**
2.5 Classificação dos *Single Event Effects* (SEEs) **27**
2.6 Métricas para a avaliação da vulnerabilidade de circuitos a *Soft Errors* **29**
2.6.1 Métodos de estimação da SER 29
2.6.2 FIT (do ingles *Failure in Time*) 30
2.6.3 MTTF (do ingles *Mean Time To Failure*) 30
3 TÉCNICAS DE MITIGAÇÃO DE *SOFT ERRORS* **31**
3.1 Técnicas relacionadas ao processo de fabricação **31**
3.2 Técnicas baseadas em hardware **32**
3.2.1 Redundância Temporal 32

3.2.2 Redundância Espacial 33
3.2.3 Redundância de Informação 33
3.3 Técnicas baseadas em software 34
3.3.1 Tolerância a falhas baseadas em Algoritmo (ABFT) 34
3.3.2 A técnica de Freivalds e suas evoluções 35
3.3.3 Minimizando o custo de recomputação do elemento corrompido 39
3.3.4 Paralelização da técnica para implementação em *hardware* 40

4 IMPLENTAÇÃO DA TÉCNICA DE FREIVALDS MODIFICADA EM HARDWARE 43

4.1 Unidade Protegida 45
4.2 Unidade de Detecção de Erros 47
4.2.1 Primeira fase da geração dos vetores codificados de entrada 48
4.2.2 Finalização dos vetores codificados de entrada 52
4.2.3 Geração dos vetores codificados de saída 54
4.2.4 Comparador de Linhas e Subtrator de colunas 56
4.2.5 Geração do endereço do elemento corrompido 58
4.3 Unidade de Correção de Erros 60
4.4 Modelo Analítico de Desempenho do RA^3 61
4.5 Tornando adaptativos os módulos Multiplicador Matriz-Matriz e Matriz-Vetor 62

5 FERRAMENTA DE SIMULAÇÃO DE FALHAS 64

5.1 Funcionamento 65
5.2 Geração do conjunto de Falhas 66
5.3 Processo de Simulação de Falhas 68
5.4 Relatórios dos resultados 69

6 RESTRIÇÕES FÍSICAS E DE TEMPO REAL E APLICAÇÕES AFINS 72

6.1 ESTUDO DE CASO I – Decodificação MIMO 73
6.2 ESTUDO DE CASO II – Sistemas VoIP e Filtragem Adaptativa 74

7 RESULTADOS EXPERIMENTAIS 76

7.1 Análise de Área da arquitetura RA^3 77
7.2 Análise do consumo de potência da arquitetura RA^3 78
7.3 Análise de custos da técnica de *Freivalds Modificada* implementada na arquitetura RA^3 79
7.4 Máximo Desempenho frente a um *Memory Wall* 80
7.5 Atingindo restrições de tempo real com mínima potência 81
7.6 Análise da cobertura de falhas da arquitetura RA^3 83
7.7 Comparação do tempo de execução com uma GPU 84

8 CONCLUSÕES E TRABALHOS FUTUROS 85

REFERÊNCIAS 86

APÊNCICE A REFERÊNCIAS DAS PUBLICAÇÕES OBTIDAS COM ESTE TRABALHO E OUTRAS CONTRIBUIÇÕES 90

LISTA DE ABREVIATURAS E SIGLAS

ABFT	Algorithm Based Fault-Tolerance
ASIC	Application Specific Integrated Circuit
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
DWC	Duplication With Comparison
FIT	Failure In Time
FPGA	Field-programmable gate array
GR	Geomagnetic Rigidity
LDTF	Long Duration Transient Faults
LET	Linear Energy Transfer
MBU	Multiple Bit Upset
MCU	Multiple Cell Upset
MPSOC	Multiprocessor System-On-Chip
MTTF	Mean Time To Failure
PG	Power Gating
PVP	Pipelined Vision Processor
RA^3	Resilient Adaptive Algebraic Architecture
RTL	Register Transfer Level
SBU	Single Bit Upset
SEE	Single Event Effects
SEL	Single Event Latchup
SER	Single Event Rate

SET	Single Event Transient
SEU	Single Event Upset
SOI	Silicon On Insulator
SRAM	Static Random Access Memory
TCL	Tool Command Language
TMR	Triple Modular Redundancy
VHDL	VHSIC Hardware Description Language

LISTA DE FIGURAS

Figura 1.1: Projeção das dimensões dos transistores para os próximos anos (ITRS, 2009). 14
Figura 1.2: Aumento da taxa relativa de falhas com o *scaling*..... 15
Figura 1.3: Relação entre a largura dos SETs e a redução na frequência de *clock* dos circuitos. 15
Figura 1.4: Tempo de Ciclo e largura dos efeitos transientes com o scaling (LISBÔA, 2009). 16
Figura 1.5: *Single Event Rate* (SER) de circuitos individuais (SHIVAKUMAR, KISTLER, *et al.*, 2002). 17
Figura 1.6: Uso da técnica de *power gating* para o desligamento de unidades visando a redução de consumo de potência e temperatura dos chips. 18
Figura 1.7: *Gap* de desempenho entre memórias e processadores. 18
Figura 1.8: Exploração de recursos visando redução do consumo de potência baseadas no *deadline* de um tarefa. 19
Figura 1.9: Exemplo de um MPSOC heterogêneo e sua ampla gama de aplicações com requisitos de tempo real envolvendo multiplicação de matrizes. 20
Figura 1.10: Principais fabricantes de DSPs com seus respectivos coprocessadores. ... 21
Figura 2.1: LET crítico e *threshold* SEU e SET como função do scaling para tecnologias *bulk* e SOI CMOS. 24
Figura 2.2. Fluxo simulado de partículas em função da altitude (LEI, CLUCAS, *et al.*, 2004). 25
Figura 2.3. Fissura do Boro induzida por um nêutron de baixa energia. 26
Figura 2.4. Interação de uma partícula *alpha* ou nêutron com cristais de silício. 26
Figura 2.5: Impactos dos efeitos de SETs nos circuitos eletrônicos. 27
Figura 2.6. Classificação resumida dos *single event effects* (*SEE's*) (YU, XIAOYA e NICOLAIDIS, 2008). 28
Figura 3.1. Níveis de abstração onde técnicas de mitigação de *soft-errors* são geralmente aplicadas. 31
Figura 3.2: Exemplo da aplicação da técnica de redundância temporal. 32
Figura 3.3: Diagrama em blocos com descrição da técnica de DWC. 33
Figura 3.4: Diagrama em blocos com a descrição da técnica de TMR. 33
Figura 3.5: Demonstração do problema relacionado a escolha aleatória do vetor *r*. 35
Figura 3.6: Esquema proposto capaz apenas de detectar erros. 36
Figura 3.7: Técnica de Freivalds Modificado acoplado ao algoritmo de multiplicação de matrizes capaz de encontrar linha e coluna do elemento corrompido. 37

Figura 3.8: Análise do comportamento da técnica sob a presença de uma falha no elemento $C_{(1,1)}$. 39
Figura 3.9: Interação entre o algoritmo de multiplicação de matrizes e a técnica de *Freivalds Modificada*. 41
Figura 4.1: Estrutura organizacional da implementação da arquitetura em VHDL. 44
Figura 4.2: Fluxo de dados do RA^3. 44
Figura 4.3: Exemplo do circuito *Multiplicador Matriz-Matriz* sintetizado com os parâmetros informados pelo usuário. 46
Figura 4.4: Exemplo de funcionamento do algoritmo de multiplicação de matrizes. 47
Figura 4.5: Circuito projetado para cálculo do vetor r^TA conectado ao *Multiplicador Matriz-Matriz*. 49
Figura 4.6: Circuito projetado para o cálculo do vetor B*r* conectado ao *Multiplicador Matriz-Matriz*. 50
Figura 4.7: Exemplo de uma configuração do circuito r^TA acoplado ao circuito *Multiplicador Matriz-Matriz*. 51
Figura 4.8: Primeira penalidade imposta ao algoritmo de multiplicação de matrizes.... 53
Figura 4.9: Segunda penalidade imposta ao algoritmo de multiplicação de matrizes.... 54
Figura 4.10: Operação e circuito para cálculo do vetor C*r*. 55
Figura 4.11: Circuito para cálculo do vetor codificado de saída r^TC. 56
Figura 4.12: Diagrama do circuito comparador de linhas. 56
Figura 4.13: Diagrama do circuito Comparador/Subtrator de Colunas. 57
Figura 4.14: Diagrama de uma memória de dados armazenando a matriz C de dimensões 3×3. 58
Figura 4.15: Exemplo do cálculo do endereço de leitura do elemento corrompido. 60
Figura 4.16: Diagrama do circuito responsável por corrigir o elemento corrompido. ... 61
Figura 4.17: Módulo Multiplicador Matriz-Matriz Modificado. 63
Figura 5.1: Interação entre a descrição do circuito e o injetor de falhas. 65
Figura 5.2: Geração dos vetores de tempo de injeção 67
Figura 5.3: Histograma com a distribuição dos tempos de falha 67
Figura 5.4: Exemplo da criação de classes de unidades 68
Figura 5.5: Demonstração da estrutura do processo de injeção de falhas da ferramenta. 68
Figura 5.6: Demonstração da utilização dos vetores de tempo e sinais no processo de injeção de falhas. 69
Figura 5.7: Exemplo de um relatório gerado após a simulação de uma campanha de injeção de falhas. 70
Figura 7.1: Utilização da arquitetura RA^3 como coprocessador embarcado. 76
Figura 7.2: Diagrama em blocos do TMR aplicado na unidade protegida (a) e da arquitetura RA^3 (b). 77
Figura 7.3: Comparação de Área entre TMR e RA^3 78
Figura 7.4: *Overhead* (%) imposto pela técnica de *Freivalds Modificada* na arquitetura RA^3. 79
Figura 7.5: Latência de acesso a memória e tempo de execução. 81
Figura 7.6: Restrições de tempo de execução e desempenho para decomposição QR. . 82
Figura 7.7: Restrições de tempo de execução e desempenho para o AEC. 83
Figura 7.8: Comparação do tempo de execução do RA^3 com um GPU. 84

LISTA DE TABELAS

Tabela 3.1: Resumo das operações da técnica de *Freivalds Modificada* para detecção e indicação da linha e coluna do elemento corrompido. 39
Tabela 4.1: Sinais de Controle para o cálculo do vetor $r^{T}A$. 49
Tabela 4.2: Sinais de Controle para o cálculo do vetor Br. 51
Tabela 4.3: Valores armazenados em cada um dos acumuladores ao final da multiplicação de matrizes. 55
Tabela 4.4: Resumo do funcionamento do modulo *Comparador de Linhas.* 57
Tabela 4.5: Resumo do funcionamento do módulo Comparador/Subtrator de colunas. 57
Tabela 4.6: Atualização do *Endereço_Linha* do elemento corrompido na memória. 59
Tabela 4.7: Atualização do endereço que aponta a coluna do elemento corrompido. ... 59
Tabela 4.8: Exemplo do controle Liga/Desliga das unidades de hardware. 63
Tabela 7.1: Consumo de potência total (mW) 22nm-1GHz. 79
Tabela 7.2: Resultado de injeção de falhas na arquitetura RA^{3} 83

RESUMO

A constante redução nas dimensões dos transistores foi o principal combustível capaz de manter o crescente desempenho exigido por aplicações. Ao mesmo tempo, as tensões de alimentação dos circuitos também são reduzidas a cada novo nó tecnológico, fazendo com que partículas como nêutrons e partículas *alpha*, portando quantidades de energia cada vez menores sejam capazes de gerar os chamados *soft errors*, que impactam diretamente na redução da confiabilidade dos sistemas embarcados atuais. Isto faz com que a implementação de técnicas de tolerância a falhas se tornem praticamente obrigatórias para tecnologias atuais e futuras. Estes mesmos sistemas embarcados, como *smartphones*, devem apresentar alto poder de processamento, visando atender um crescente conjunto de aplicações de natureza heterogênea, consumindo a mínima potência possível. Nestes sistemas, algumas dessas principais aplicações como codec GSM, cancelamento de eco acústico, processamento de áudio e vídeo apresentam em comum a necessidade de multiplicar matrizes de diferentes dimensões em determinados intervalos de tempo. Pensando nestas demandas, será proposta a arquitetura RA^3, cujo objetivo é executar o algoritmo de multiplicação de matrizes em paralelo com a técnica de tolerância a falhas conhecida na literatura como ABFT, visando a aumentar a confiabilidade da mesma. Além disso, a RA^3 possui uma estrutura adaptativa que permite que unidades internas como memórias, multiplicadores e somadores sejam ligadas ou desligadas através da aplicação da técnica de *power gating* em tempo de execução, conforme restrições impostas pela largura da banda de memória, *power budgets* e *deadlines* impostos por aplicações de tempo real, visando executar tarefas consumindo a mínima potência possível. Para avaliar as funcionalidades propostas, dois estudos de caso reais são apresentados e o comportamento da arquitetura é avaliado sobre diversos aspectos como desempenho, área, consumo de potência e cobertura de falhas. Finalmente é possível comprovar que a adaptabilidade proposta pela arquitetura RA^3 permite que seja encontrada, em diversos cenários, a quantidade exata de recursos necessários para executar determinadas aplicações sem comprometer as restrições impostas principalmente no consumo de potência e por aplicações com *deadlines* críticos, mantendo ainda altas taxas de cobertura de falhas.

Palavras-Chave: arquiteturas adaptativas, tolerância a falhas, sistemas embarcados

Adaptive Parallelism Exploitation under Physical and Real-Time Constraints for Fault Tolerant Embedded Systems

ABSTRACT

The continuous reduction of transistors' dimensions was the main drive capable of maintaining the performance increase required by applications. At the same time, supply voltages of the circuits are also reduced with each new technology node, causing particles such as neutrons or alpha particles, even with reduced amounts of energy, to generate so-called soft errors that directly impact on the reliability of embedded systems. This scenario makes the implementation of techniques for fault tolerance mandatory for current and future technologies. Still, embedded systems, such as smartphones, must provide high processing power to execute a growing set of applications of heterogeneous nature, consuming the least possible power. In these systems, applications like GSM codec, acoustic echo cancellation, audio and video processing have in common the need for matrix multiplication operations of different dimensions at certain time intervals. To efficiently support the aforementioned scenario, this dissertation proposes the RA^3 architecture whose goal is run the matrix multiplication algorithm in parallel with the fault tolerance technique know in the literature as ABFT, aiming to support software execution with high reliability. Furthermore, the RA^3 architecture provides adaptive internal units such as memories, multipliers and adders with adaptive powering on or off by applying power gating at runtime. Runtime power gating enables to meet restrictions imposed by real-time applications or memory bandwidth with minimum power. To evaluate the proposed architecture, two case studies are presented and the behavior of the architecture is evaluated in terms of performance, area, power consumption and fault coverage. Finally, a comprehensive design space exploration shows that the adaptability provided by the RA^3 architecture allows the system designer to find, in many scenarios, the exact amount of resources needed to run a set of applications without compromising the restrictions imposed mainly in power consumption and real-time deadlines, while still maintaining a high fault coverage rate.

Keywords: Adaptive Architectures, Fault Tolerance, Embedded systems

1 INTRODUÇÃO

1.1 Motivação

1.1.1 Impactos na confiabilidade de circuitos associados com o *scaling*

Um dos maiores desafios para projetistas de circuitos integrados preocupados com questões relacionadas com confiabilidade consiste em qual será a tendência deste comportamento para tecnologias futuras. Para iniciar esta análise é preciso avaliar o que está acontecendo com as dimensões dos transistores para as próximas gerações de tecnologias e entender quais os impactos que estas mudanças irão trazer à tona. Como podemos avaliar na Figura 1.1, uma projeção é feita até o ano de 2024 permitindo avaliar que o *scaling* continuará ocorrendo com um fator praticamente constante em cada novo nó tecnológico.

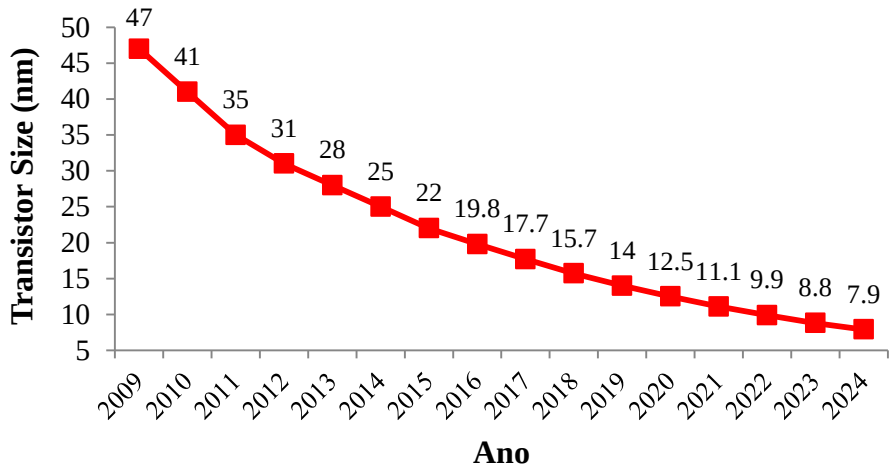


Figura 1.1: Projeção das dimensões dos transistores para os próximos anos (ITRS, 2009).

Com a diminuição constante das dimensões dos transistores, projetistas são obrigados a reduzir a tensão de alimentação dos transistores visando redução da densidade e consumo de potência em circuitos integrados. Estes dois fatos impactam num aumento da sensibilidade dos circuitos a efeitos de radiação, uma vez que reduz a quantidade de carga mínima necessária para gerar um efeito transiente (*glitch*) na forma de um pulso de tensão ou corrente nos nós dos circuitos integrados. Estes pulsos são denominados SETs (do inglês *Single Event Transients*) e podem ser causados quando

uma única partícula carregada atinge determinados nós dentro do circuito e muda temporariamente e de maneira indesejada o comportamento de circuitos combinacionais. Quando estes efeitos transientes (temporários) são capturados por elementos de memória, como por exemplo *flip-flops*, estas perturbações são registradas dando origem aos chamados SEUs (do inglês *Single Event Upsets*), podendo originar erros no funcionamento dos circuitos integrados.

Estes efeitos impactam na diminuição da confiabilidade em circuitos integrados devido a efeitos de radiação, e sua relação com o *scaling* é mostrada na Figura 1.2, onde é feita uma projeção da taxa de falhas para os próximos nós tecnológicos. A curva indica um aumento na taxa relativa de falhas de 8% de degradação/bit/geração de tecnologia confirmando que circuitos irão falhar cada vez mais à medida que forem reduzidas as dimensões dos transistores.

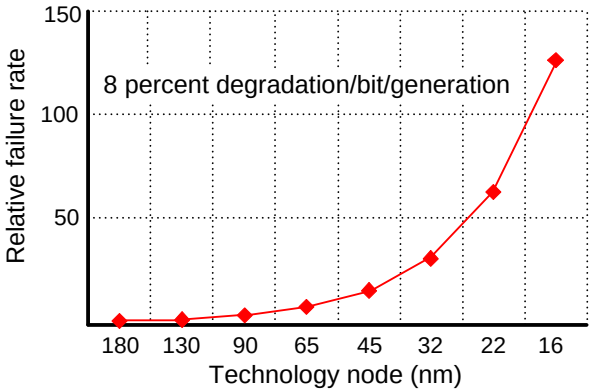


Figura 1.2: Aumento da taxa relativa de falhas com o *scaling*.

Por outro lado, em busca de desempenho, projetistas continuam aumentando constantemente a frequência de *clock* dos circuitos integrados. Isto significa que em tecnologias passadas, onde os *chips* operavam com frequências de *clock* mais baixas, a janela de tempo dada pelo período de *clock*, era muito maior do que a largura dos SETs, fazendo com que a probabilidade dos mesmos tornarem-se SEUs ser muito pequena ou praticamente nula. Entretanto, como pode ser visto na Figura 1.3, o aumento da frequência de *clock* reduz a janela de captura, e SETs agora possuem probabilidades cada vez maiores de tornarem-se SEUs. Este fenômeno foi denominado na literatura como LDTF (do inglês *Long Duration Transient Faults*), e a probabilidade destes efeitos serem capturadas por elementos de memória aumenta linearmente com o aumento da frequência de *clock* nos circuitos integrados (LISBÔA, 2009).

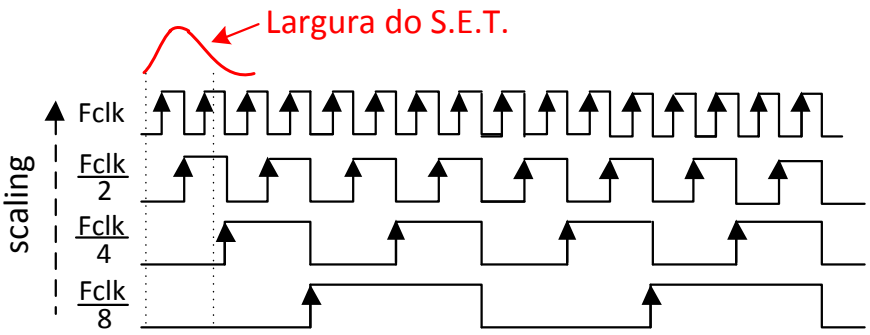


Figura 1.3: Relação entre a largura dos SETs e a redução na frequência de *clock* dos circuitos.

No gráfico da Figura 1.4, são mostrados mais dados que reforçam a ideia do problema relacionado a redução da frequência de *clock* com os SETs. Neste gráfico, as linhas relacionam a largura do período de *clock* com os atrasos de cadeias com diferentes números de inversores (10, 8, 6 e 4) em diferentes tecnologias. Na mesma figura as barras correspondem à largura dos pulsos transientes previstos para partículas com LET (do inglês *Linear Energy Transfer*) de $10Mev - cm^2/mg$ (amarelo) e $20Mev - cm^2/mg$ (laranja). O LET significa, de maneira simplificada, a quantidade de energia depositada num determinado material quando uma partícula ionizante atinge e atravessa o mesmo e têm como objetivo quantificar os efeitos da radiação ionizante nos dispositivos eletrônicos. O que pode ser observado, é que para tecnologia de 180nm, apenas as partículas com LET de 20MeV teriam transientes com largura suficiente para preocupar circuitos com ciclos de *clock* de largura equivalente a uma cadeia com 4 inversores da mesma tecnologia. Para tecnologias abaixo de 180nm, partículas de LET cada vez menores são capazes de gerar pulsos transientes com largura maior do que o período de *clock* dos circuitos, aumentando assim a chance destes transientes serem capturados pelos elementos de memória.

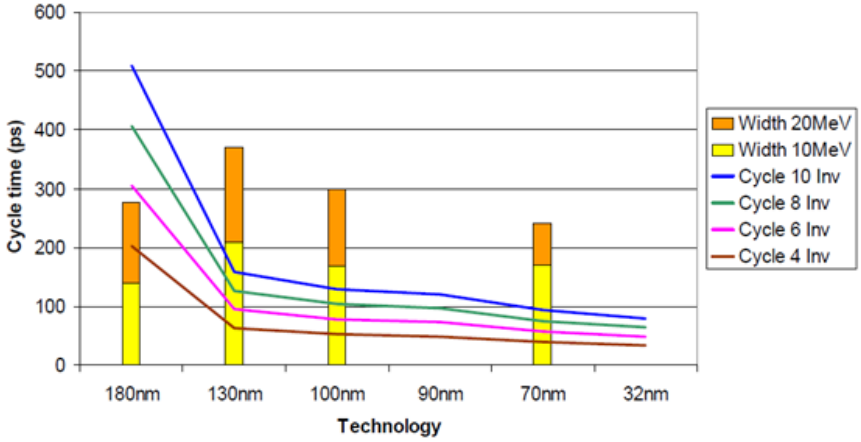


Figura 1.4: Tempo de Ciclo e largura dos efeitos transientes com o scaling (LISBÔA, 2009).

Até aqui, foram discutidos alguns aspectos preocupantes trazidos pelo *scaling* e seus reflexos na confiabilidade dos circuitos. Também foi discutido sobre os SETs e SEUs, que são efeitos gerados em partes distintas dos circuitos digitais. Os SETs atingem a parte lógica e podem ou não se tornar SEUs, enquanto que esses em sua essência, ocorrem quando partículas atingem diretamente os elementos de memória. Sendo assim, projetistas se perguntam sobre quais as partes que devem ser efetivamente protegidas, a parte lógica (combinacional) ou os elementos de memória dos sistemas digitais.

Existe um número muito maior de trabalhos na literatura focando na proteção de dispositivos de memórias, como DRAMs, SRAMs e *latchs* do que parte na lógica dos circuitos. Entretanto, conforme podemos observar na Figura 1.5, é possível notar que a SER (do inglês *Single Event Rate*) ocasionada diretamente em elementos de memória, no caso da figura em SRAMs e *latchs* que em tecnologias passadas era predominante, para tecnologias atuais e futuras será ultrapassado pelas falhas geradas na parte lógica do circuito. Isto serve para destacar que técnicas de proteção direcionadas a parte lógica (combinacional) serão obrigatórias para os próximos nós de tecnologia.

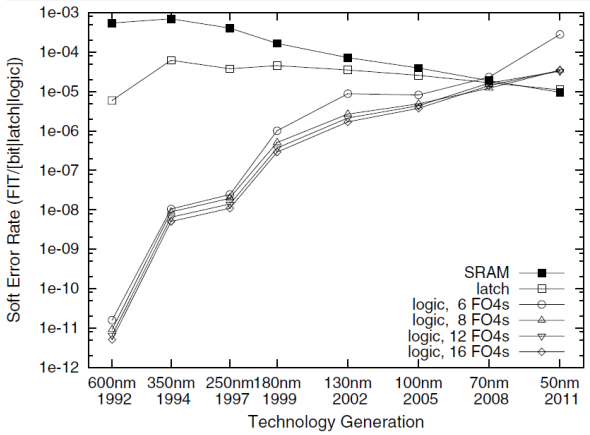


Figura 1.5: *Single Event Rate* (SER) de circuitos individuais (SHIVAKUMAR, KISTLER, *et al.*, 2002).

1.1.2 Limitadores do consumo de potência para os próximos nós tecnológicos

O mesmo *scaling* discutido na subseção anterior que permite uma maior integração e aumento no número de transistores por unidade de área dentro dos chips, também ocasiona um aumento da densidade de potência dentro dos circuitos. Isto se dá pelo fato de que a quantidade de corrente elétrica necessária para que os transistores funcionem não decai na mesma proporção em que as dimensões dos mesmos, fazendo com que circuitos no futuro dissipem cada vez mais potência. Este aumento na densidade de corrente é um dos principais desafios dos projetistas de sistemas embarcados, pois obrigará a indústria a trabalhar sob os chamados orçamentos de potência (*power budgets*) quando projetarem novos circuitos. Em outras palavras, no futuro os chips não poderão permanecer com 100% de suas unidades ligadas durante todo o tempo. Até mesmo em cenários onde não existam restrições críticas de energia, a temperatura máxima na junção dos transistores deve ser respeitada levando a um aumento nos custos relacionados com encapsulamento e resfriamento dos chips.

Para resolver estes problemas, projetistas de hardware aplicam técnicas como DVFS (do inglês *Dynamic Voltage and Frequency Scaling*). O problema desta técnica é que a redução da tensão de alimentação reduz a carga crítica dos nós do circuito afetando diretamente a confiabilidade dos circuitos. Outra forma muito utilizada para reduzir o consumo de potência é aplicar técnicas de PG (do inglês *Power Gating*), de forma a desligar unidades que não estiverem sendo utilizadas em determinados momentos. A aplicação de PG será mandatória em tecnologias futuras: uma recente estimativa mostra que para a tecnologia de 22nm, em torno de 21% do circuito deverá estar desligado para respeitar o *power budget* enquanto que para tecnologia de 8nm a taxa de unidades desligadas está por volta de 50% da área do chip, sendo esta grande área desligada chamada de silício negro (*dark silicon)* (ESMAEIZADEH, BLEM, *et al.*, 2011).

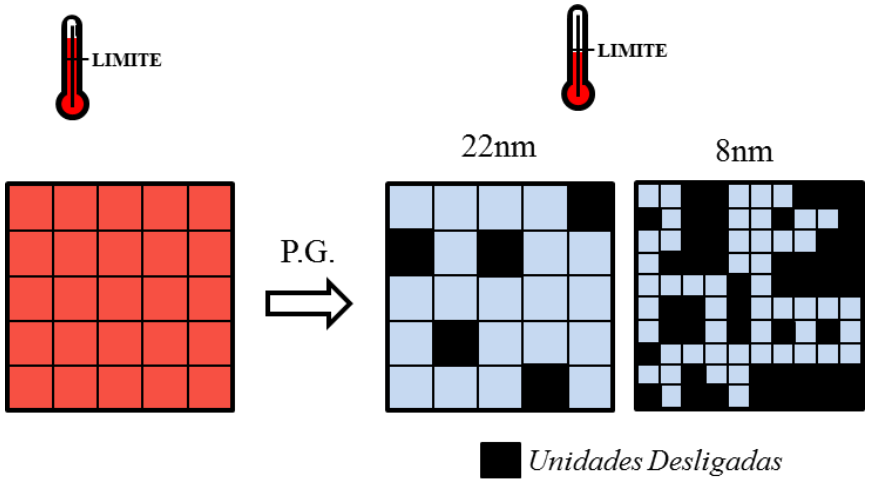


Figura 1.6: Uso da técnica de *power gating* para o desligamento de unidades visando a redução de consumo de potência e temperatura dos chips.

Na Figura 1.6, é mostrado um diagrama que visa ilustrar o impacto da aplicação da técnica de *power gating* nos chips e o aumento da área denominada silício negro com o *scaling*. Sabendo que unidades deverão estar desligadas dentro dos chips, devem ser estabelecidos critérios que determinem quais e quando estas unidades deverão ser desligadas.

1.1.3 Exploração Adaptativa de Paralelismo baseado na Banda de Memória

Sabendo do aumento dos níveis de dissipação de potência e visando o aumento de desempenho, os projetos de microprocessadores atuais focam na exploração máxima de paralelismo possível, o que inclui mais recursos de *hardware* para isto. Um dos principais problemas na exploração do paralelismo está em como alimentar este mesmo *hardware* com dados suficientes, sendo que a largura de banda é limitada e não escala na mesma proporção que a vazão dos dados da parte lógica, dando origem ao chamado *memory gap*, cujo diagrama é mostrado na Figura 1.7.

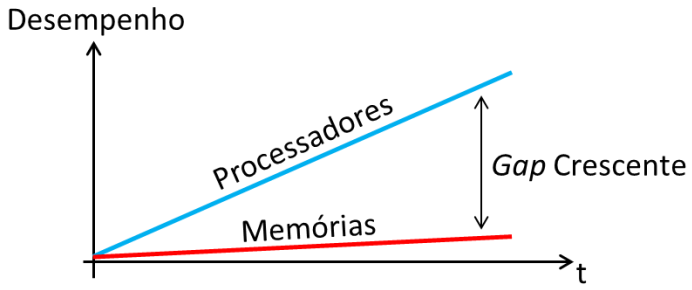


Figura 1.7: *Gap* de desempenho entre memórias e processadores.

Visto isso, é razoável pensar em desligar unidades funcionais de forma adaptativa dentro de um processador de forma a explorar o grau máximo de paralelismo possível para uma determinada aplicação, considerando a banda de memória disponível.

1.1.4 Exploração Adaptativa de Paralelismo aplicado em sistemas de tempo real

Outro critério para exploração adaptativa de recursos abordada neste trabalho é direcionada a sistemas embarcados de tempo real, partindo da premissa que sistemas deste tipo trabalham com determinados *deadlines*. O *deadline* é o limite máximo de

tempo para a execução de uma determinada tarefa. Conhecendo estes *deadlines* e a quantidade de recursos disponíveis é possível desligar unidades até encontrar o número de unidades que permitem o atendimento do *deadline* imposto gastando a menor quantidade de potência possível.

Na Figura 1.8, foi criado um cenário onde uma tarefa genérica apresenta um *deadline* de 10µs. Para executar esta tarefa, foram criadas quatro configurações (*a*, *b*, *c* e *d*) que atendem o *deadline* com quantidades diferentes de unidades ligadas e consequentemente gastando quantidades diferentes de potência. Do ponto de vista energético, o gasto é o mesmo em todas as configurações, entretanto na configuração *d*, embora o circuito leve 8 vezes mais tempo do que a configuração *a* para ser executado, ele consome 8× menos potência do que na mesma configuração, não excedendo o consumo de 1,25W de potência. Sistemas mais complexos, como MPSOCs (do inglês *Multi-Processor Systems on Chip*) heterogêneos, que delegam *power budgets* para suas unidades de processamentos, são exemplos de como estruturas adaptativas podem ser bem aplicadas.

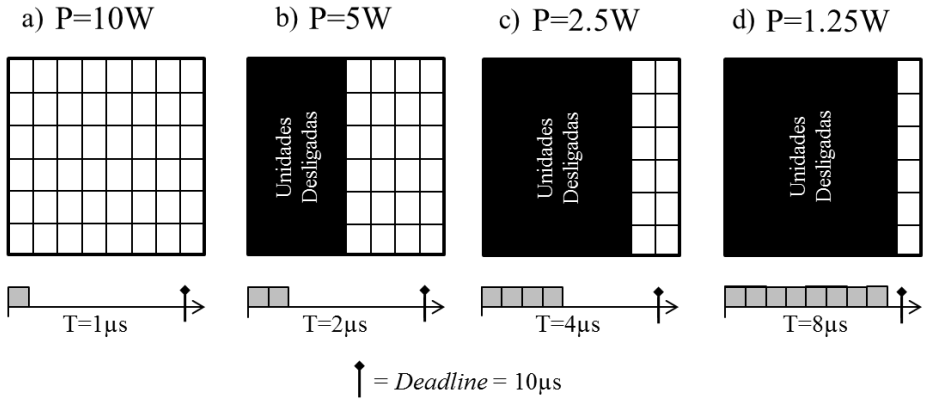


Figura 1.8: Exploração de recursos visando redução do consumo de potência baseadas no *deadline* de um tarefa.

1.1.5 Proteção do algoritmo de Multiplicação de Matrizes

Uma realidade dentro do mercado de sistemas embarcados são os chamados MPSOCs. Estes componentes são compostos por múltiplos processadores de natureza geralmente heterogênea haja vista a variada gama de aplicações com características diferentes que o mesmo deve executar. Estas aplicações são concorrentes e geralmente possuem seus próprios *deadlines*. Na Figura 1.9 é mostrado um *smartphone* como exemplo de MPSOC heterogêneo e uma vasta gama de aplicações que utilizam a operação de multiplicação de matrizes. Estas operações possuem *deadlines* próprios e por serem executadas em sistemas embarcados alimentados por bateria, possuem restrições quanto ao consumo de potência.

Figura 1.9: Exemplo de um MPSOC heterogêneo e sua ampla gama de aplicações com requisitos de tempo real envolvendo multiplicação de matrizes.

Além de sistemas embarcados, a operação de multiplicação de matrizes é uma operação muito utilizada em diversas áreas da engenharia, como:

- Computação Gráfica;
- Processamento Digital de Sinais;
- Sistemas de Controle;
- Álgebra Linear;
- Modelos numéricos de previsão meteorológica.

Em determinadas aplicações, matrizes com grandes dimensões são utilizadas, sendo que em casos onde estes tipos de operações são feitas em grande número e sequencialmente, um possível erro indesejado poderia se propagar para as demais operações matriciais e seria capaz de prejudicar horas e até dias de processamento. Isto serve para ressaltar a importância do uso de técnicas de detecção e correção de erros em matrizes.

1.1.6 Utilização de Aceleradores de Hardware em aplicações Multimídia e de Tempo Real

Como discutido anteriormente, as aplicações multimídia são dominantes nos sistemas embarcados atuais, que por sua vez demandam grande capacidade de processamento. Visando este fato, as grandes fabricantes do mercado de DSPs (do inglês *Digital Signal Processors*) estão a algum tempo focando na utilização de unidades de hardware dedicadas para execução de serviços específicos. Para tal, estes fabricantes inserem os chamados coprocessadores visando aliviar a carga de trabalho dos *cores* internos dos DSPs. A Figura 1.10 mostra três DSPs das maiores fabricantes do mercado, o modelo ADSP-BF609 (ANALOG DEVICES, 2012) da Analog Devices®, o modelo TMS320C6670 (TEXAS INSTRUMENTS, 2012) da Texas Instruments® e o modelo MSC8126 (FREESCALE) da fabricante Freescale® e seus respectivos coprocessadores embarcados.

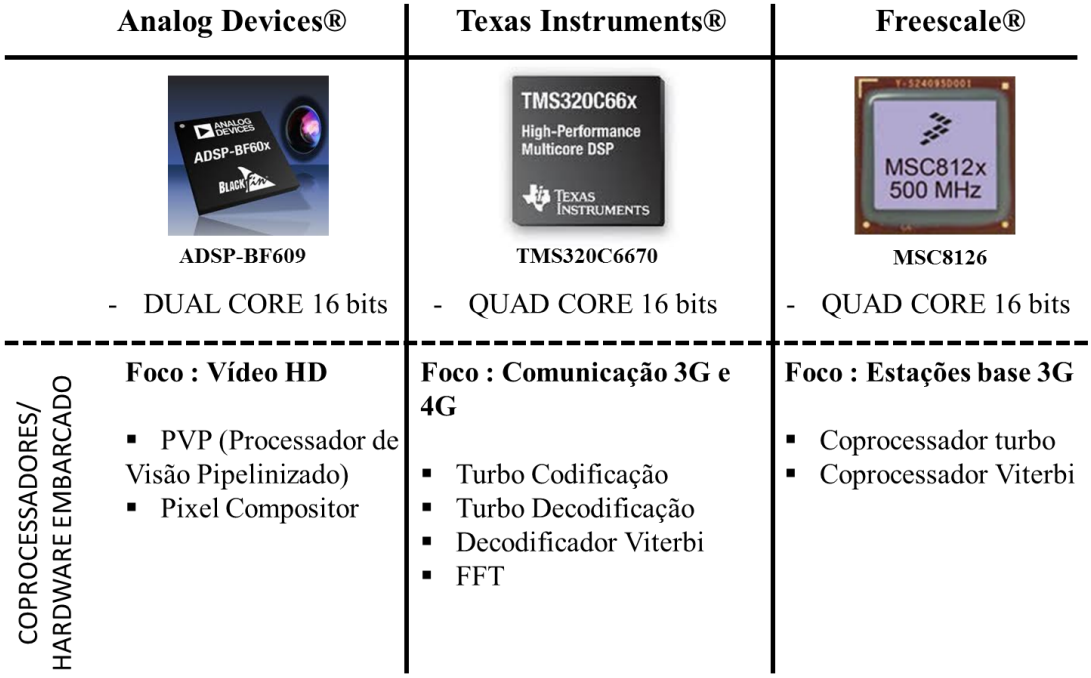


Figura 1.10: Principais fabricantes de DSPs com seus respectivos coprocessadores.

Conforme podemos observar na figura, o DSP ADSP-BF609 conta com o coprocessador PVP (do inglês *Pipelined Vision Processor*) utilizado para dispositivos com visão embarcada, como no uso de câmeras em veículos para detecção de pedestres e outros veículos, capaz de acelerar até cinco algoritmos de imagem concorrentes. Já os DSPs modelo TMS320C6670 e MSC8126 contam com uma gama de coprocessadores como Turbo Codificador e Turbo Decodificador que são utilizados para acelerar estes processos em canais de comunicação 3G e 4G. Estes dois modelos contam também com os coprocessadores de decodificação Viterbi, utilizado para acelerar a decodificação de canais de voz e dados em sistemas de telefonia 3G, que necessitam de decodificação de dados codificados com código convolucional. Entretanto apenas o modelo da Texas Instrument® conta com hardware dedicado de FFT utilizado para a aceleração da execução da transformada rápida de Fourier.

Apesar de se perceber que os DSPs apresentados anteriormente são projetados especificamente para acelerar as aplicações alvo de determinados segmentos, muito pouco tem se visto em termos de tolerância a falhas nestes dispositivos. Com a crescente preocupação com relação à confiabilidade para os próximos nós tecnológicos, demonstrado nas últimas subseções, dispositivos que atendam as necessidades de desempenho e confiabilidade simultaneamente podem em breve se tornar uma realidade.

1.2 Objetivos

O objetivo deste trabalho é desenvolver em VHDL uma arquitetura capaz de efetuar operações de multiplicação de matrizes em paralelo com uma técnica de mitigação de soft erros genuinamente aplicada em software, chamada de ABFT (do inglês *Algorithm-based fault tolerance*) embarcada ao sistema, para detecção e correção de erros em

matrizes. Esta arquitetura deve ser capaz de permitir a exploração dos recursos *hardware*, desde a escolha do número máximo de unidades disponíveis em tempo de projeto até o controle que permita ligar ou desligar unidades funcionais internas em tempo de execução de acordo com requisitos pré-estabelecidos. Fazendo assim com que a arquitetura seja capaz de se adaptar de forma a explorar o máximo paralelismo de uma aplicação sob determinados limitadores de potência e largura de banda de memória.

Outro objetivo deste trabalho está em criar uma plataforma que permita simular o procedimento de injeção de falhas na descrição da arquitetura e permita avaliar a eficácia do funcionamento da técnica de ABFT em paralelo com a execução do algoritmo de multiplicação de matrizes. Esta plataforma deverá ser capaz de simular os efeitos de SETs na parte lógica dos circuitos e permitir a visualização dos valores corrigidos nas matrizes de resultado.

1.3 Organização do texto

O restante deste trabalho está organizado como segue. O capítulo 2 apresenta as principais partículas radioativas responsáveis pela geração de *soft erros* e o impacto que a incidência destas partículas causa nos transistores e como estes efeitos se propagam até circuitos eletrônicos mais complexos. Depois disso, no capítulo 3, são discutidas técnicas diferentes para combater estes efeitos em diferentes níveis de abstração no projeto de circuitos integrados. No capítulo 4 é apresentada a metodologia de implementação da arquitetura proposta. A forma e critérios utilizados no processo de injeção de falhas na arquitetura, que serve para validar a estrutura são apresentados no capítulo 5. Após, no capítulo 6, são apresentados dois estudos de casos baseados em aplicações reais com características que vão ao encontro da arquitetura proposta. No capítulo 7, são mostrados e discutidos os resultados da arquitetura em relação a diversos eixos de exploração. Finalmente no capítulo 8, são apresentadas as conclusões e possíveis trabalhos futuros.

2 PRINCIPAIS FONTES DE RADIAÇÃO E SEUS EFEITOS NO SILÍCIO E CIRCUITOS ELETRÔNICOS

Falhas transientes em dispositivos semicondutores podem ser induzidas por diversas fontes, como variabilidade dos transistores, radiação externa ao chip, etc. Existem duas fontes principais de radiação capazes de induzir tais falhas: partículas *alpha* provenientes do encapsulamento de circuitos integrados (CI's) e nêutrons atmosféricos, gerando os chamados pares elétron-lacuna (direta ou indiretamente) na medida em que atravessam um dispositivo semicondutor (BAUMANN, 2005).

2.1 Partículas *alpha*

São partículas compostas por dois nêutrons e dois prótons provenientes de um átomo de hélio duplamente ionizado. No final da década de 70 foram responsáveis pela primeira observação de um SEU numa memória RAM na superfície terrestre (MAY e WOODS, 1978). Estas partículas são provenientes de impurezas de natureza radioativa contidas nos materiais utilizados no encapsulamento de circuitos integrados, especificamente Urânio (U) e Tório (Th), sendo assim muito difíceis de serem completamente eliminadas. Pequenas quantidades de epóxi ou chumbo não radioativo conseguem reduzir a sensibilidade às partículas *alpha* fornecendo uma espécie de blindagem contra esta fonte de radiação (BAUMANN, 2005). Outro fator que influencia na incidência destas partículas é o material utilizado no processo de solda dos dispositivos, usualmente feitas de chumbo (Pb) e estanho (Sn), os quais são extraídos de minérios que podem conter traços de Urânio (U) e Tório (Th). Sabendo-se disso, é aconselhável que projetistas não posicionem pontos de solda próximos aos nós mais sensíveis dos circuitos (VELASCO e FRANCO, 2007).

A Figura 2.1 mostra a tendência de SEUs e SETs em relação ao *scaling* para as tecnologias baseadas em *bulk* silício e SOI (do inglês *Silicon On Insulator*). Nesta figura, os valores limite de LET para SRAMs é plotado com relação à tecnologia e junto com os limites para propagação de SETs. A primeira observação sobre esta figura é sobre os limites para SEUs e SETs, que como já comentado no capítulo anterior, nitidamente caem com o *scaling*, ou seja, colisões de partículas com LETs cada vez menores começam a se tornar preocupantes. A segunda observação diz respeito ao tipo da tecnologia e suas sensibilidades, por exemplo, para a tecnologia de *bulk* CMOS, as memórias SRAM se tornam sensitivas a *upsets* causados por partículas *alpha* em

tecnologias inferiores a 250nm, enquanto que para tecnologias SOI apenas por volta de 90nm (DODD, SHANEYFELT, *et al.*, 2010).

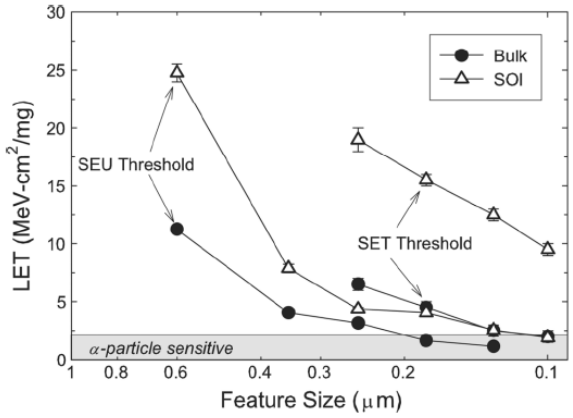


Figura 2.1: LET crítico e *threshold* SEU e SET como função do scaling para tecnologias *bulk* e SOI CMOS.

2.2 Raios Cósmicos

A magnetosfera é constantemente bombardeada por um fluxo isotrópico de partículas carregadas, principalmente pelos núcleos dos átomos que tiveram seus elétrons capturados. Este fluxo de partículas energéticas é proveniente tanto da atividade solar quanto do núcleo da galáxia sendo composto de 85% de prótons, 14% de partículas *alpha* ou núcleo de hélio e 1% de materiais mais pesados como, por exemplo, núcleos de carbono e ferro (DYER e RODGERS, 1998).

Em altitudes terrestres (diferente de altitudes de satélite e de aviões), menos do que 1% do fluxo primário atinge o nível do mar, onde o fluxo é isotrópico e composto de múons, prótons, nêutrons e pions. A primeira observação de um SEU na superfície terrestre devido a raios cósmicos ocorreu no ano de 1979 (ZIEGLER e LANFORD, 1981). Os nêutrons apresentam um dos componentes com o maior fluxo e reações com o maior LET, eles são a fonte de radiação mais provável de causar transtornos em dispositivos eletrônicos em altitudes terrestres (BAUMANN, 2005).

Os dois principais fatores que influenciam no fluxo de nêutrons são a altitude e a latitude. A Figura 2.2 mostra a variação do fluxo de algumas partículas com a altitude. Pode-se notar que o fluxo aumenta aproximadamente dez vezes a cada 3 km com um ponto de saturação que ocorre por volta de 15-20 km com um valor de fluxo 100-200 vezes maior do que o nível do mar. (VELASCO e FRANCO, 2007).

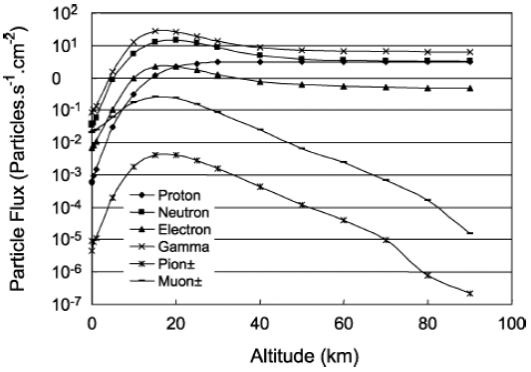


Figura 2.2. Fluxo simulado de partículas em função da altitude (LEI, CLUCAS, *et al.*, 2004)**.**

Outro fator, menos discutido do que a variação do fluxo com a altitude, mas que também influencia no fluxo de nêutrons é a localização na superfície da terra onde o fluxo incide. O campo magnético terrestre consegue curvar as partículas cósmicas primárias e secundárias e refleti-las de volta para o espaço. O momento mínimo necessário para que uma partícula normalmente incidente vença a intensidade magnética da terra e alcance o nível do mar é chamado de GR (do inglês *geomagnetic rigidity*) de um ponto na terra. Quanto maior a GR de um ponto, menor é o fluxo de nêutrons neste ponto. O GR é maior próximo da linha do equador (por volta de 17 Gev), por isso apresenta um fluxo menor de nêutrons. Em contrapartida, o GR é menor em localidades próximas aos pólos norte e sul (em torno de 1 Gev), onde o fluxo de elétrons é maior (MUKHERJEE, 2008). Os nêutrons podem ainda ser classificados em nêutrons de alta e baixa energia e serão explicados nas duas próximas subseções.

2.2.1 Nêutrons de alta energia

Os nêutrons de alta energia não geram diretamente os chamados pares elétron-lacuna nos semicondutores. Esta interação se dá através de colisões elásticas e inelásticas com o núcleo de silício (Si). As colisões elásticas depositam apenas pequenas quantidades de energia no substrato de silício, já nas colisões inelásticas grandes quantidades de energia são trocadas.

Ao contrário das partículas *alpha*, o fluxo de nêutrons cósmicos não pode ser reduzido significativamente com blindagem ao nível do chip. Por outro lado, no nível de ambiente, concreto tem mostrado bons resultados para conseguir blindar contra radiação cósmica numa taxa de 1,4× a cada 30 cm de espessura de concreto (DIRK, NELSON e ZIEGLER, 2003). Além disso, estas partículas inclinam mais o centro do rastro de ionização, produzindo um rastro com maior diâmetro. Rastros com maior diâmetro produzem mais MBUs (do inglês *Multiple Bit Upsets*) e uma taxa menor de recombinação elétron-lacuna.

2.2.2 Nêutrons de baixa energia

A terceira fonte de ionização de partículas em dispositivos eletrônicos resulta da interação de nêutrons de baixa energia (<< 1 Mev) com o Boro (B). O Boro é um elemento químico muito utilizado na fabricação de circuitos integrados, principalmente como dopante do tipo *p*. O ^{10}B é um material instável quando exposto a nêutrons, e no caso de absorção de um nêutron, seu núcleo sofre uma fissura liberando um núcleo excitado de ^{7}Li (Lítio) e uma partícula *alpha* conforme mostrado na Figura 2.3. Tanto a partícula *alpha* quanto o lítio são capazes de induzir *soft errors* em dispositivos

eletrônicos, principalmente com tecnologias que operam com tensões de alimentação reduzidas (BAUMANN, 2005).

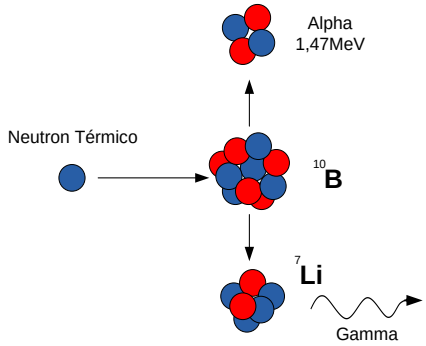


Figura 2.3. Fissura do Boro induzida por um nêutron de baixa energia.

2.3 Interação de partículas *Alpha* e Nêutrons com cristais de silício

As interações entre partículas *alpha* ou nêutrons com cristais de silício possuem algumas diferenças. Partículas *alpha* carregadas interagem diretamente com os elétrons enquanto que com os nêutrons a interação se dá através de colisões elásticas ou inelásticas. Resultados experimentais mostram que as colisões inelásticas são a maior causa de *soft errors* devido a nêutrons (MUKHERJEE, 2008).

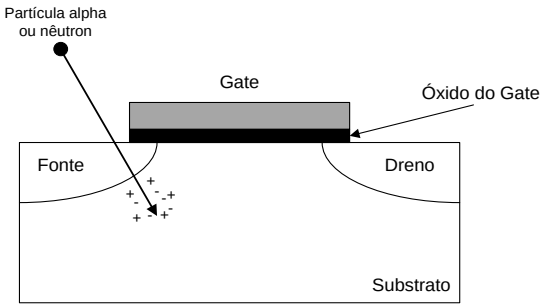


Figura 2.4. Interação de uma partícula *alpha* ou nêutron com cristais de silício.

Quando partículas *alpha* penetram num cristal de silício, fortes perturbações são geradas, criando os chamados pares elétron-lacuna nos substratos dos transistores. O campo elétrico próximo da junção *p-n*, interface entre substrato e difusão, pode ser alto suficiente para prevenir que pares elétron-lacuna se recombinem. Sendo assim, estes portadores de carga em excesso serão varridos para dentro das regiões de difusão e eventualmente aos contatos do dispositivo.

Um dos principais conceitos para o bom entendimento dos efeitos de radiação em cristais de silício chama-se potência de parada (do inglês *stopping power*). A potência de parada é definida como a perda de energia por unidade de comprimento do rastro deixado pela partícula, a qual mede a troca de energia média entre a partícula *alpha* incidente e os elétrons. Isto é o mesmo que o LET, assumindo que toda energia absorvida na média é utilizada para produção de pares elétrons-lacuna.

Nêutrons não causam diretamente uma falha transiente, pois eles não criam diretamente os chamados pares elétrons-lacuna em cristais de silício, portanto sua potência de parada é zero. Ao invés disso, estas partículas colidem com o núcleo do semicondutor, resultando na emissão de fragmentos nucleares secundários. A probabilidade da ocorrência de colisões que produzem estes fragmentos secundários é

extremamente pequena, consequentemente é necessário um número de nêutrons por volta de 10^5 vezes maior do que partículas *alpha* para produzir o mesmo número de falhas transientes num dispositivo semicondutor (MUKHERJEE, 2008).

2.4 Impacto dos efeitos transientes nos circuitos eletrônicos

A análise anterior foi feita com relação às características físicas da interação dos cristais de silício quando atingidos por partículas ionizantes. Entretanto é importante entender como este efeito se propaga para outros níveis de abstração, como por exemplo, os níveis elétrico e lógico. Na Figura 2.5, é mostrado uma cadeia com dois inversores entre dois registradores, no nível lógico (em cima) e elétrico (em baixo). Este diagrama visa facilitar o entendimento do efeito que é gerado quando estas partículas atingem os transistores que operam dentro destes circuitos.

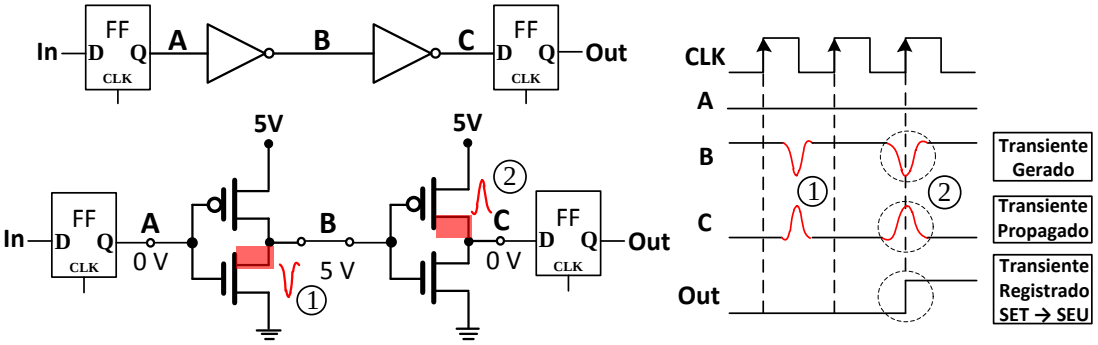


Figura 2.5: Impactos dos efeitos de SETs nos circuitos eletrônicos.

No esquema elétrico mostrado na Figura 2.5, a área em vermelho representa as áreas dos transistores sensíveis a falhas, e que varia de acordo com o estado lógico em que o circuito se encontra. Em ambos os inversores da figura, o transistor que não está em condução é o que apresenta o nó sensível, no caso do inversor com a entrada em nível lógico baixo, o nó sensível pertence ao transistor NMOS e no caso do inversor com nível lógico alto na entrada o nó sensível pertence ao transistor PMOS. Na mesma figura foram criadas duas situações identificadas pelos números 1 e 2, onde SETs são gerados em tempos diferentes no circuito. No lado direito da figura, pode ser visto que o primeiro efeito transiente (1) se propaga até a saída *C*, entretanto não é capturado pelo flip-flop, pois o mesmo não ocorre no instante de tempo onde ocorre a borda de subida do sinal de *clock*, este efeito é conhecido como mascaramento de tempo, e será mostrado com mais detalhes na subseção 2.5. Já no caso do segundo efeito transiente (2), o SET atinge o mesmo nó sensível no exato instante em que ocorre a borda de subida no sinal de *clock*, e ao contrário do transiente (1) onde o efeito havia sido mascarado, o mesmo será registrado e deixará de ser um SET e passará a ser um SEU.

2.5 Classificação dos *Single Event Effects* (SEEs)

De forma geral, os efeitos gerados pelas partículas anteriormente apresentadas, são conhecidos como SEEs (do inglês *Single Event Effects*) e estão associados a mudanças de estados ou transientes em dispositivos induzidos pela passagem de um único íon através ou próximo de um nó sensível de um circuito. Os SEEs podem ser classificados basicamente quanto ao tipo de dano causado em dispositivos CMOS, como mostra a Figura 2.6.

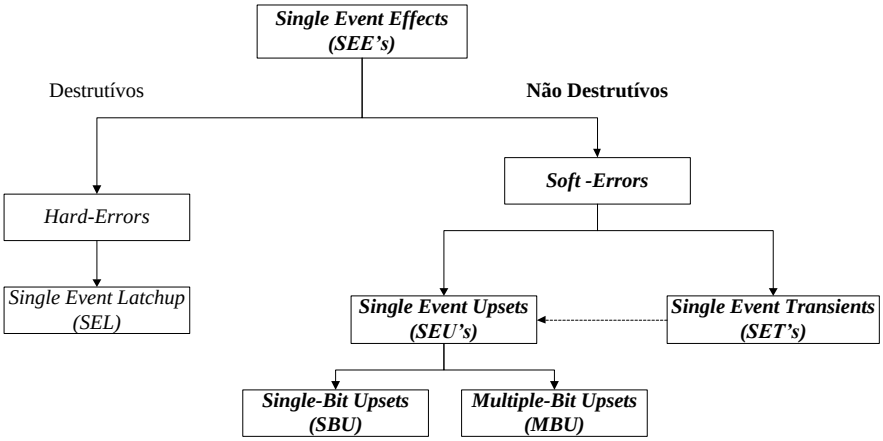


Figura 2.6. Classificação resumida dos *single event effects* (*SEE's*) (YU, XIAOYA e NICOLAIDIS, 2008)**.**

Os *soft errors* são uma classe de erros que tem como principal característica não danificar permanentemente circuitos ou dispositivos. Sendo assim, quando ocorrem podem ser corrigidos através da restauração do valor correto. A fonte mais conhecida de *soft errors* é o *Single Event Upset* (SEU), ocasionado quando uma única partícula colide com uma célula de um elemento memória. Estas células podem ser do tipo SRAM, DRAM, latch ou registrador. Os SEUs podem ainda ser classificados em *upsets* de um único bit denominado de SBU (do inglês *Single Bit Upset*) ou *upsets* de múltiplos bits ou MBUs (do inglês *Multiple Bit Upsets*) ou ainda upsets de múltiplas células chamado de MCU (do inglês *Multiple Cell Upsets*). Os MCUs ocorrem quando uma partícula modifica o estado de células adjacentes de memória. Já os MBUs modificam mais de um único bit dentro de uma mesma palavra, sendo assim um efeito mais difícil e mais caro de ser corrigido em memórias, uma vez que muitos circuitos correção são baseados em paridade e códigos de detecção. Memórias atuais já utilizam técnicas que entrelaçam bits de células diferentes, de forma a evitar que bits de uma mesma palavra estejam fisicamente adjacentes no *layout* do circuito (LADBURY, 2007).

Dentro dos *soft erros* existem ainda os chamados SETs, que já foram discutidos anteriormente, que são pulsos transitórios que causam perturbações em circuito combinacionais. Os principais fatores físicos que influenciam na largura e amplitude deste pulso de tensão gerado são a velocidade da coleta de cargas e a capacitância de carga. Quando estes pulsos são capturados por elementos de memória se transformam em SEUs. Nem todo SET se tornará um SEU, pois existe ainda a possibilidade dos SETs serem mascarados. As três principais fontes de mascaramento de SETs são: (SEIFERT, 2010) (DIXIT, HEALD e WOOD)

1. **Mascaramento elétrico:** pulsos serão atenuados antes de chegar a um elemento de memória;
2. **Mascaramento lógico:** propagação de pulsos está bloqueada por portas lógicas;
3. **Mascaramento de tempo:** pulsos propagados não serão registrados ao menos que cruzem com o tempo de *setup* ou *hold* da janela de tempo do receptor.

Uma vez que um SET tenha sido capturado por um elemento de memória (l*atch*, *flip-flop*, etc.), se torna indistinguível de um SEU causado diretamente por uma colisão de partícula com um elemento de memória. Os três efeitos de mascaramento dependem de propriedades elétricas, lógicas e de temporização do circuito. Para que o pulso se

propague do nó atingido através de um caminho combinacional, até a entrada de um elemento de memória, este caminho deve estar logicamente sensibilizado e a largura do pulso deve ser maior do que o atraso de propagação da porta mais lenta do caminho.

Finalmente temos os erros do tipo *hard*, que podem tanto danificar permanentemente apenas um único nó de um circuito como até mesmo o dispositivo todo. Um exemplo de *hard error*, mostrado na Figura 2.6, é o SEL (do inglês *Single Event Latchup*), gerado por efeitos parasitas inerentes da tecnologia CMOS que podem gerar um curto virtual entre *Vdd* e terra, resultando no efeito chamado de *latchup* (BRUGUIER e PALAU, 1996). Existem ainda outros tipos de *hard errors* que não serão mencionados, pois erros destrutivos não fazem parte do escopo deste trabalho.

2.6 Métricas para a avaliação da vulnerabilidade de circuitos a *Soft Errors*

O conhecimento do grau de vulnerabilidade a soft erros de um determinado circuito é fundamental ao usuário final, uma vez que quanto mais vulnerável maior probabilidade deste circuito apresentar erros desta natureza. Algumas empresas informam os dados de vulnerabilidade de acordo com algumas métricas conhecidas e que serão apresentadas a seguir. Esta vulnerabilidade é também conhecida como SER (do inglês *Single Event Rate*) e corresponde a taxa com que *soft errors* acontecem num determinado dispositivo para um dado ambiente.

2.6.1 Métodos de estimação da SER

Existem várias maneiras de se estimar a SER de um determinado circuito. Algumas delas são baseadas na simulação dos efeitos de radiação através do conhecimento de parâmetros físicos inerentes a tecnologia empregada e do ambiente de teste. Esta metodologia acaba se tornando interessante, pois é uma maneira rápida e com razoável precisão de orientar projetistas sobre o grau de sensibilidade apresentada pelo seu *design*. De acordo com os resultados apresentados nas simulações, torna-se possível ao projetista optar por implementar ou não algum mecanismo de tolerância a falhas, de forma a aumentar a confiabilidade e sucesso do produto final.

Estas simulações podem ser executadas no nível elétrico e apresentam resultados aparentemente independentes de mecanismos ligados à tecnologia aplicada, como por exemplo, simulações de geração e coleta de cargas. Entretanto, o processo de coleta de carga ocorrido quando uma partícula atinge o nó de um circuito, é simulado através de uma fonte de corrente fixada ao mesmo. Esta abordagem foi utilizada em (FREEMAN, 1996) para descrever a carga crítica (Q_{CRIT}) baseada em metodologias de simulação no nível de circuito para SRAMs de tecnologia bipolar. Entretanto esta abordagem pode ser utilizada para qualquer tipo de circuito e tecnologia. A forma de onda da corrente gerada por esta fonte é uma exponencial conforme mostra a equação 1:

$$I_{coll}(t) = KQ_{coll}\sqrt{t^{\frac{e^{-t}}{L}}} \tag{1}$$

O procedimento de simulação consiste na aplicação destes pulsos em forma de corrente em determinados nós do circuito e no monitoramento destes efeitos na saída do mesmo.

Existem também metodologias de estimação de SER de circuitos baseadas em modelos matemáticos. Nesta abordagem é necessário conhecer uma ampla gama de características físicas vinculadas à tecnologia utilizada. Todas as simulações envolvendo

modelos são executadas no nível de circuito após os impactos tecnológicos tenham sido calibrados. Estes modelos são chamados de modelos compactos de SER e consistem de funções analíticas que tem como principais variáveis a carga crítica (Q_{crit}), áreas de difusão dos nós sensíveis e o tipo de difusão utilizada. Um modelo compacto clássico de SER foi apresentado por (HAZUCHA e SVENSSON, 2000) conforme mostrado na equação 2:

$$SER = K \times F \times A \times e^{-\frac{Q_{crit}}{Q_s}} \quad (2)$$

Como podemos observar na equação 2, a taxa nominal de *soft errors* de um nó atingido é proporcional à probabilidade da carga coletada (Q_s) ser maior do que a carga crítica (Q_{crit}). O parâmetro Q_{crit} é uma variável específica do circuito, e pode ser estimada através de simulações em ferramentas, como por exemplo, o SPICE, enquanto que a carga coletada é um parâmetro específico da tecnologia. Existem também outros parâmetros utilizados neste modelo como o *cross section* (A) que equivale à área da junção reversamente polarizada do nó atingido, a probabilidade de uma particula depositar sua carga no volume sensível do chip (K) e o fluxo de partículas (F). Nas próximas subseções serão apresentadas algumas unidades em que o SER pode ser encontrado.

2.6.2 FIT (do ingles *Failure in Time*)

Quando um ambiente em particular é conhecido, o SER pode ser dado em FIT. A grandeza de 1 FIT equivale a 1 falha em 10^9 horas de operação. Em memórias semicondutoras, a sensibilidade é geralmente dada em FIT/Mb ou em FIT/dispositivo. O valor do FIT pode ser tanto previsto através de simulações quanto através de medidas experimentais, como por exemplo, submeter um determinado circuito a um acelerador de partículas. Este tipo de procedimento permite alcançar medidas mais precisas, haja vista que se torna possível gerar efeitos mais próximos da realidade (GAILLARD, 2011).

Uma forma complementar para a estimação do FIT, sem a utilização de modelos de falhas exponenciais mais complexos, se dá através do conhecimento de parâmetros conhecidos como *cross section* (σ) e o fluxo de partículas (ϕ) de um ambiente real expresso em $n/cm^2/h$ e pode ser calculado conforme a equação 3:

$$FIT = \sigma \times \phi \times 10^9 \quad (3)$$

2.6.3 MTTF (do ingles *Mean Time To Failure*)

O MTTF é outra medida muito usada para mensurar a confiabilidade de sistemas. Esta medida indica o tempo médio esperado até a primeira falha de uma determinada parte do circuito. O MTTF possui uma relação inversa com o FIT, que pode ser expressa por:

$$MTTF_{(HORAS)} = \frac{10^9}{FIT} \quad (4)$$

3 TÉCNICAS DE MITIGAÇÃO DE *SOFT ERRORS*

Como visto no final do capítulo anterior, projetistas buscam por metodologias capazes de avaliar a vulnerabilidade de determinados circuitos a *soft erros*. Dentre elas, as baseadas em modelos se mostram capazes de prever esta vulnerabilidade através de simulações da SER. Contudo, estas simulações exigem um conhecimento de parâmetros físicos de fabricação dos chips e fatores externos referentes ao ambiente. Estas simulações são capazes de apontar além quão susceptível está o circuito a estes efeitos como também quais as partes críticas e que de alguma forma devem ser protegidas. É por isso, que os mesmos projetistas preocupados com aspectos referentes à tolerância a *soft errors* podem tomar a decisão de utilizar técnicas capazes de mitigar (abrandar, amenizar) estes efeitos. Com o passar do tempo, muitas técnicas de mitigação foram desenvolvidas nos mais variados níveis de abstração, como poder ser observado no diagrama da Figura 3.1, que representa a classificação destes níveis.

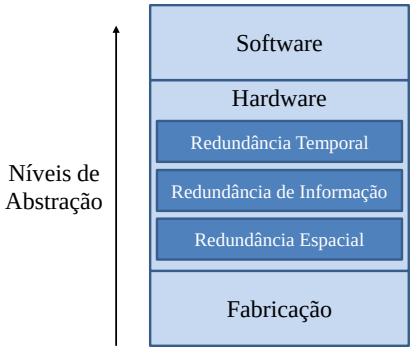


Figura 3.1. Níveis de abstração onde técnicas de mitigação de *soft-errors* são geralmente aplicadas.

3.1 Técnicas relacionadas ao processo de fabricação

Em termos de melhorias efetuadas na tecnologia de fabricação, basicamente as duas que apresentaram os melhores resultados visando reduzir a taxa de *soft errors* foram o poço triplo (do inglês *triple-well*) e o SOI (do inglês *Silicon On Insulator*).

Um circuito de tecnologia CMOS pode ser fabricado basicamente de duas formas, com *double well* (poço duplo) ou *triple well* (poço triplo) (CHATTERJEE, NARASIMHAM, *et al.*, 2011). A técnica de *triple well* consiste na criação de um terceiro poço, chamado “poço profundo”, local para onde são varridas as cargas geradas quando uma partícula *alpha* ou um nêutron atingem o dispositivo, evitando acúmulo de

carga no dreno do transistor. Esta técnica é muito utilizada em memórias SRAMs, pois melhora consideravelmente a isolação dos transistores com o substrato.

Já a tecnologia de silício como isolante (SOI) é uma técnica consolidada e consiste na introdução de oxido entre o dreno e substrato e a fonte e substrato. Desta forma, a capacitância de junção é reduzida entre a fonte (ou dreno) e o substrato melhorando o desempenho. Além disso, a tecnologia reduz significativamente o volume sensível dos circuitos, o que resulta numa redução da quantidade coletada de carga da partícula que gerou a colisão, reduzindo assim a vulnerabilidade a *soft errors* (MUKHERJEE, 2008).

3.2 Técnicas baseadas em hardware

São técnicas de mitigação implementadas durante a fase de projetos do sistema a ser protegido. Este grupo de técnicas não é direcionada ao mercado de GPPs (do inglês *General Purpouse Processors*) e têm sua aplicação restrita a ASICs (do inglês *Application Specific Integrated Circuits*) ou projetos baseados em FPGAs (do inglês *Field-Programmable Gate Array*).

Estas técnicas se caracterizam pela necessidade da inserção de novos elementos de hardware, que permitam alguma forma de redundância, acarretando num aumento de área e consequentemente no consumo de potência dos circuitos. Podem ser divididas em redundância temporal, redundância espacial e redundância de informação.

3.2.1 Redundância Temporal

Nesta técnica a redundância se dá na repetição da computação ou no armazenamento do resultado desta computação em elementos de memória em instantes de tempo diferentes. Na Figura 3.2, é mostrado um exemplo de um circuito combinacional que deve ter sua saída protegida, para que eventuais SETs não se propaguem até a saída e sejam registrados, tornando-se SEUs. Para isto, sua saída é conectada na entrada de três flip-flops diferentes com sinais de *clock* também diferentes, que na verdade são versões atrasadas do sinal de *clock* principal. Isto permite uma amostragem do sinal de saída em momentos diferentes. Na ocorrência de um SET, sabendo que este pulso possui um intervalo de tempo de vida, supõe-se que ele seria capturado apenas por um dos três *flip-flops*. Como as saídas destes *flip-flops* estão ligadas a um votador de maioria, a captura do SET por apenas um *flip-flop* seria mascarada pelo mesmo, uma vez que as outras duas entradas estariam com o valor correto.

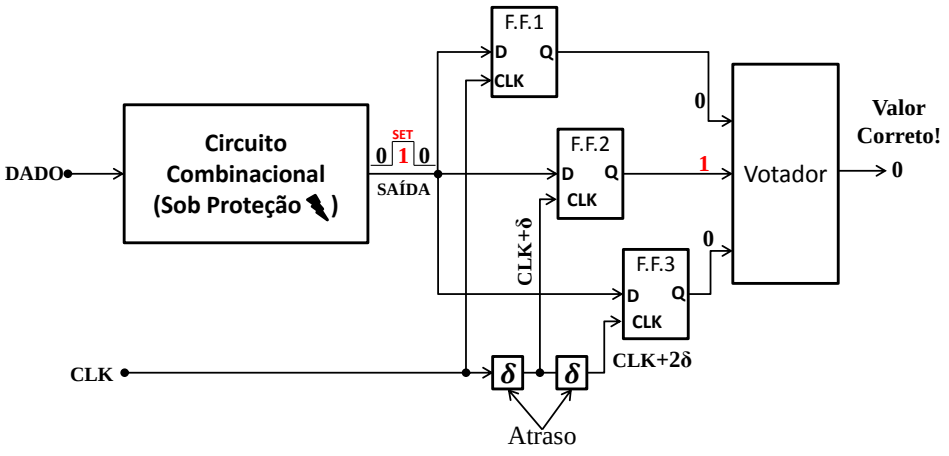


Figura 3.2: Exemplo da aplicação da técnica de redundância temporal.

3.2.2 Redundância Espacial

Este grupo de técnicas baseia-se na replicação dos circuitos para apenas detectar ou detectar e corrigir erros. Dentro do grupo de técnicas baseadas na redundância espacial, as duas principais são a DWC (do inglês *Duplication With Comparison*) e o TMR (do inglês *Triple Modular Redundancy*).

A técnica de DWC, conforme o diagrama mostrado na Figura 3.3, é uma técnica que permite apenas a detecção de erros através da comparação da saída do circuito com a saída da réplica do circuito (Saída 2). Além da duplicação do circuito, a técnica exige ainda a inserção de um comparador que apontará os possíveis erros quando ocorrerem divergências entre os valores das saídas.

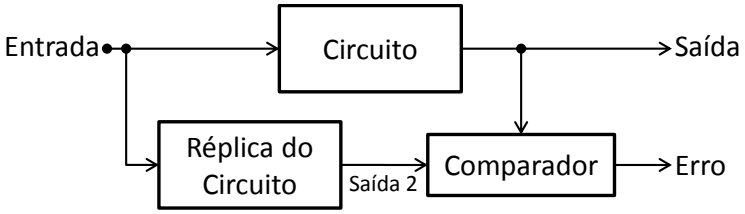


Figura 3.3: Diagrama em blocos com descrição da técnica de DWC.

Já a técnica de TMR, é capaz de mascarar as falhas em um componente de hardware através da triplicação dos componentes e votação entre os resultados das saídas para a determinação do resultado correto. É capaz de detectar e corrigir erros desde que seja considerado o modelo de falhas simples, onde apenas uma falha ocorra de cada vez. A Figura 3.4 mostra um diagrama em blocos com a estrutura da técnica de TMR.

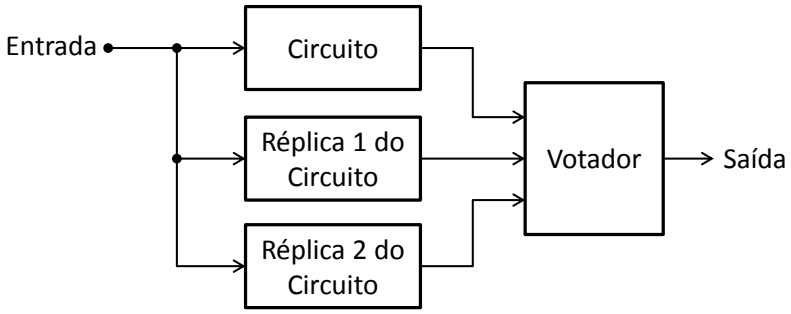


Figura 3.4: Diagrama em blocos com a descrição da técnica de TMR.

O principal problema referente a este grupo de técnicas está relacionado ao alto custo em área e principalmente no consumo de potência imposto. Isto acaba se tornando um grande problema principalmente para projetistas de hardware de sistemas embarcados alimentados por bateria, que são obrigados a buscar por diferentes alternativas para prover sistemas mais confiáveis.

3.2.3 Redundância de Informação

Neste grupo de técnicas, a redundância encontra-se na adição de informação extra, na forma de bits ou sinais, junto aos dados de interesse. Esta informação extra é verificada de forma a indicar um possível erro nos dados a serem protegidos. São técnicas muito utilizadas para proteger principalmente as memórias contra os SEUs.

Os códigos de paridade são exemplos clássicos deste tipo de técnica, onde para cada *n* bits, são armazenados *n+1* bits. O bit extra indica a paridade da palavra, em outras

palavras, se o número de bits em 1 dentro da palavra é par ou ímpar. Códigos de paridade servem apenas para detecção de falhas simples, que são falhas que possuem apenas um bit modificado dentro da palavra de cada vez.

Existem também códigos que são capazes de detectar e corrigir erros, um exemplo é o ECC (do inglês *Error Correction Code*). Este tipo de código é muito utilizado em memórias e na transferência de dados entre memórias e processadores. Exemplos de ECCs são os códigos de Hamming, que são formados por um conjunto de bits de paridade que permitem a detecção e correção de erros (WEBER, 2002).

3.3 Técnicas baseadas em software

Sistemas baseados em GPPs não permitem a utilização de técnicas de tolerância em hardware. Nestes casos uma solução atrativa é embutir mecanismos de *software* para lidar com os problemas de *soft errors*. Este mecanismo se baseia na inserção de redundância no código ou nos dados de forma que seja possível detectar e até corrigir possíveis falhas.

Foi pensando principalmente na redução de custos e em atingir altos índices de dependabilidade, que foi criado um novo paradigma conhecido como SIHFT (do inglês *Software-Implemented Hardware Fault Tolerance*) para desenvolvimento de sistemas baseados em processadores onde o usuário possa contar com níveis aceitáveis de confiabilidade. A técnica de SIHFT permite a mitigação de *soft erros* usando redundância de informação e temporal, evitando que se utilize a redundância de hardware, que como discutido anteriormente, apresenta altos custos referentes à área e consumo de potência.

A técnica de SIHFT pode ser dividida em redundância temporal em nível de instrução ou em nível de tarefas. O foco da redundância temporal em nível de instruções está na adição de instruções que visam exclusivamente replicar dados ou replicar a computação. Esta inserção de instruções pode ser feita em programas escritos tanto em linguagem C, assembly e até em níveis mais baixos. A redundância temporal em nível de instruções pode ser dividida ainda em:

- **Técnicas orientadas a dados**: O código fonte da aplicação é modificado pela replicação de cada dado armazenado, de cada operação e na checagem da consistência dos dados. Pode ser aplicado em linguagens de alto nível como C, *assembly* e até no código intermediário gerado pelo compilador.
- **Técnicas orientadas ao controle**: Focam em detectar falhas que modificam o fluxo correto de execução dos programas. Todas as técnicas no nível de instrução são baseadas na divisão do código do programa em BBs (do inglês *Basic Blocks*), construção de grafos e a checagem em tempo de execução sobre a correta transição entre os vértices deste grafo.

Existe ainda a técnica de SIHFT que utiliza a redundância temporal aplicada ao nível de tarefas onde a checagem de consistência é feita ao final da execução de cada tarefa. Apresenta as vantagens de necessitar de um número menor de checagens e não necessitar desabilitar as otimizações do compilador.

3.3.1 Tolerância a falhas baseadas em Algoritmo (ABFT)

Esta técnica foi proposta por (HUANG e ABRAHAM, 1984) e foi dirigida para a detecção de erros em alto nível devido a falhas internas em sistemas com múltiplos processadores. Nesta técnica, os dados de entrada são codificados em nível de sistema e

o algoritmo é modificado para operar com estes dados e assim gerar os dados de saída também codificados. A redundância nos dados de saída codificados é usada na verificação do sistema para detectar e localizar a presença de falhas. A técnica de ABFT não duplica a computação para prover tolerância a falhas, ao contrário de diversas outras técnicas apresentadas anteriormente.

A técnica de ABFT é muito utilizada em aplicações baseadas em matrizes e processamento de sinais, como multiplicação de matrizes, inversão de matrizes, decomposição LU e a transformada rápida de Fourier (KOREN e KRISHNA, 2007).

3.3.2 A técnica de Freivalds e suas evoluções

A técnica de tolerância a falhas utilizada neste trabalho, com foco na proteção de multiplicação de matrizes, é uma evolução da técnica proposta originalmente por Freivalds (FREIVALDS, 1979). Esta técnica utiliza a multiplicação de matrizes por vetores de forma a reduzir o tempo de computação no processo de verificação de resultados gerados pelo algoritmo de multiplicação de matrizes. Freivalds provou que criando vetores compostos apenas por zeros e uns de maneira aleatória, denominados pela letra *r*, e multiplicando estes mesmos vetores pela matriz C (C*r*=C × *r*) e pelo produto entre as matrizes AB ($ABr = AB \times r$) em caso de igualdade destes novos vetores gerados, a probabilidade do produto $C = AB$ estar correto é maior do que ½. Outro atrativo da técnica é que enquanto o algoritmo de multiplicação de matrizes apresenta uma complexidade $O(n^3)$, a técnica baseada na multiplicação de matriz por vetor apresenta a complexidade $O(n^2)$.

A escolha do vetor *r* de forma aleatória, faz com que a técnica possua o inconveniente de não garantir que todos os erros sejam sempre detectados. Isto se dá pelo fato de que existe a mesma probabilidade de ½ para que os elementos aleatórios que compõe o vetor *r* sejam zero ou um. O problema, em sí, ocorre apenas para os elementos do vetor *r* que forem iguais a zero. Um exemplo disso, pode ser visto na Figura 3.5, onde que o vetor *r*, gerado aleatoriamente, possui dois elementos iguais a zero e dois iguais a um.

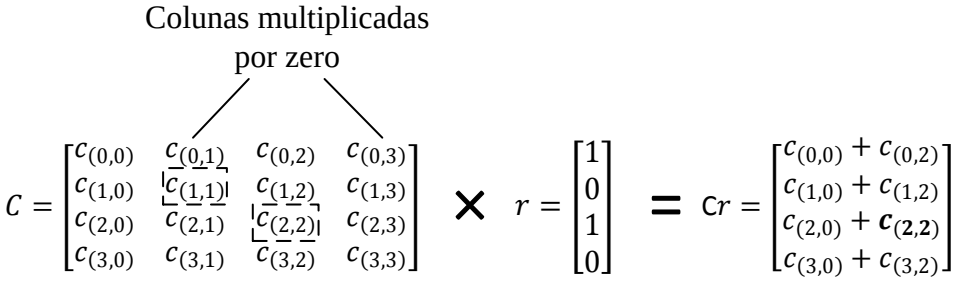


Figura 3.5: Demonstração do problema relacionado a escolha aleatória do vetor *r*.

O fato de que os elementos iguais a zero estão posicionados nas linhas 1 e 3 do vetor *r*, faz com que os elementos da colunas 1 $(c_{(0,1)}, c_{(1,1)}, c_{(2,1)}\ e\ c_{(3,1)})$ e da coluna 3 $(c_{(0,3)}, c_{(1,3)}, c_{(2,3)}\ e\ c_{(3,3)})$ da matriz C sejam sempre multiplicados por zero, impossibilitando que sejam detectados erros em elementos pertencentes a estas colunas. Ainda na Figura 3.5, estão destacados dois elementos (retângulo tracejado) na matriz C em duas situações diferentes, primeiro o elemento $c_{(1,1)}$ pertencente a coluna 1, que por ser multiplicado por zero não aparece na construção do vetor Cr, enquanto que o outro elemento $c_{(2,2)}$ pertence a coluna dois que será multiplicado por um elemento igual a

um do vetor r, e por causa disso é contabilizado no cálculo do vetor Cr, permitindo que um possível erro neste elemento seja detectável.

Visto isso, algumas melhorias foram propostas visando aumentar a cobertura de falhas proposta na técnica original. Uma delas se baseia no teorema apresentado em (LISBOA e CARRO, 2007), que prova que após a geração aleatória do vetor *r* para técnica, seja gerado um vetor denominado r_c, contendo elementos que sejam o complemento binário dos elementos do vetor *r*. Após a geração destes dois vetores, a técnica é aplicada duas vezes, uma com o vetor *r* e outra com o vetor r_c, aumentando a probabilidade de detecção de erros de ½ para 1. O dobro na taxa de cobertura de falhas acarreta no dobro do tempo de execução da técnica de detecção de erros. Seguindo o mesmo raciocínio, concluiu-se que não existe a necessidade da geração de elementos iguais a zero no vetor *r* ou ainda um segundo vetor r_c complementar. Sendo assim necessário a utilização de apenas um único vetor *r*, composto apenas por elementos iguais a 1, permitindo que seja possível obter uma probabilidade de detecção de erros igual a um com a execução da técnica apenas uma única vez.

Como discutido anteriormente, é possível comparar os vetores Cr e $A(Br)$ e em caso de igualdade afirmar que o produto $A \times B = C$ está correto. Em caso de divergência entre pelo menos um destes elementos que compõe os vetores, pode-se afirmar que existe um elemento corrompido dentro da matriz C, entretanto não é possível determinar qual é este elemento. Desta forma, a forma de correção possível é através da recomputação completa da multiplicação de matrizes em questão. Foi provado também em (LISBOA, ERIGSON e CARRO, 2007), que para matrizes pequenas, o overhead da técnica é maior do que recomputar a multiplicação, entretanto à medida que a matriz aumenta, o custo se torna cada vez menor comparado com a recomputação completa da matriz. A Figura 3.6 mostra um diagrama em blocos que utiliza a técnica de Freivalds apenas para a detecção de erros.

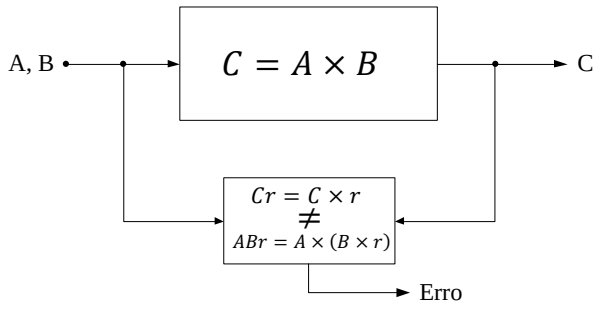


Figura 3.6: Esquema proposto capaz apenas de detectar erros.

Para que seja possível reduzir o tempo de recomputação, dado pelo tempo total da multiplicação de matrizes, foi proposto em (LISBÔA, 2009) uma maneira de reduzir o tempo de recomputação através da descoberta e recomputação apenas do elemento corrompido dentro da matriz C. Para que isto seja possível, é necessário utilizar também um vetor denominado r^T, que é apenas uma versão transposta do vetor r. Assim é possível através da comparação dos vetores Cr com $A(Br)$ e os vetores $r^T C$ com $(r^T A)B$ descobrir a linha e a coluna em que o elemento corrompido está localizado dentro da matriz C. Esta última versão da técnica é a utilizada no trabalho e será referenciada neste trabalho como "*Freivalds Modificada*".

Para facilitar o entendimento de como a técnica de *Freivalds Modificada* opera junto com a operação de multiplicação matrizes foi feito um diagrama em blocos mostrado na

Figura 3.7. Pode ser observado que a técnica codifica as matrizes de entrada e opera com estes dados codificados através do módulo denominado "Multiplicação Matriz-Vetor". A matriz resultado C, também é codificada de forma a permitir que seja feita uma comparação entre os vetores codificados de entrada e saída. Através desta comparação é possível, além de detectar uma falha, encontrar a linha e coluna onde o elemento corrompido está posicionado dentro da matriz C. A linha e a coluna do elemento corrompido apontado pela técnica significam também qual a linha da matriz A e a coluna da matriz B que devem ser buscadas para que seja possível recomputar apenas o elemento corrompido e sobrescrevê-lo.

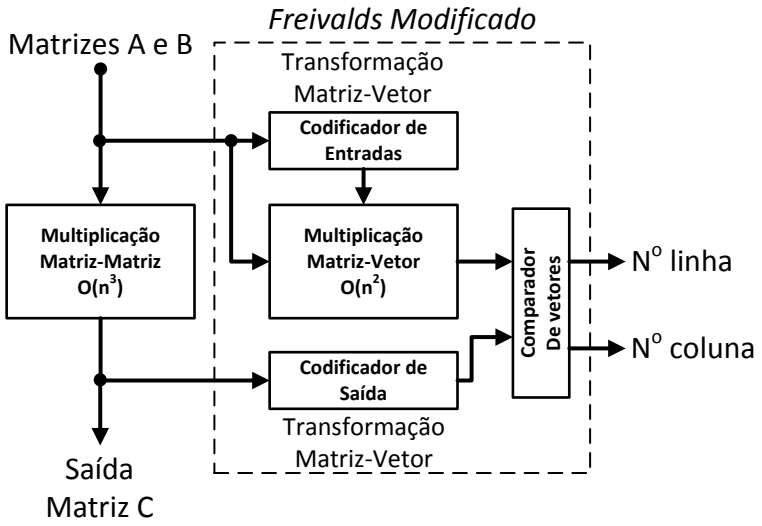


Figura 3.7: Técnica de Freivalds Modificado acoplado ao algoritmo de multiplicação de matrizes capaz de encontrar linha e coluna do elemento corrompido.

Conhecendo a estrutura da técnica de *Freivalds Modificada* e a forma como é acoplada ao circuito de multiplicação de matrizes, uma explicação mais aprofundada sobre cada um dos blocos será feita. Para isto, é considerada a operação C=A×B, onde A e B são matrizes 3×3 com os valores numéricos mostrados abaixo, e todos os cálculos utilizados na técnica serão executados em cima destes valores.

$$A = \begin{bmatrix} 45 & 83 & -30 \\ -18 & -11 & 62 \\ 5 & -73 & 22 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} -44 & -9 & 27 \\ 8 & 34 & 91 \\ -23 & 51 & -63 \end{bmatrix}$$

Como resultado correto do produto entre as matrizes A e B temos a matriz C igual a:

$$C = \begin{bmatrix} -626 & 887 & 10658 \\ -722 & 2950 & -5393 \\ -1310 & -1405 & -7894 \end{bmatrix}$$

- **Codificação das matrizes de entrada A e B:** A codificação das matrizes de entrada é feita através da multiplicação da matriz B pelo vetor r e pela multiplicação do vetor r^T pela matriz A. Estas duas operações dão origem aos vetores $Br_{(3\times1)}$ e $r^TA_{(1\times3)}$, que na Figura 3.7 são representadas pelo bloco "Codificador de entradas".

$$B = \begin{bmatrix} -44 & -9 & 27 \\ 8 & 34 & 91 \\ -23 & 51 & -63 \end{bmatrix} \times r = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \mathrm{B}r = \begin{bmatrix} -26 \\ 133 \\ -35 \end{bmatrix}$$

$$r^T = [1 \quad 1 \quad 1] \times A = \begin{bmatrix} 45 & 83 & -30 \\ -18 & -11 & 62 \\ 5 & -73 & 22 \end{bmatrix} = r^T A = [32 \quad -1 \quad 54]$$

- **Finalização dos vetores codificados de entrada:** A segunda parte da codificação dos vetores de entrada se dá através da multiplicação da matriz A pelo vetor B*r* e do vetor r^TA pela matriz B. Estas duas operações dão finalmente origem aos vetores codificados AB$r_{(3\times1)}$ e r^TAB$_{(1\times3)}$ e são executadas pelo módulo representado na Figura 3.7 como "Multiplicação Matrix-Vetor".

$$A = \begin{bmatrix} 45 & 83 & -30 \\ -18 & -11 & 62 \\ 5 & -73 & 22 \end{bmatrix} \times \mathrm{B}r = \begin{bmatrix} -26 \\ 133 \\ -35 \end{bmatrix} = A(Br) = \begin{bmatrix} 10919 \\ -3165 \\ -10609 \end{bmatrix}$$

$$r^T A = [32 \quad -1 \quad 54] \times B = \begin{bmatrix} -44 & -9 & 27 \\ 8 & 34 & 91 \\ -23 & 51 & -63 \end{bmatrix} = r^T AB = [-2658 \quad 2432 \quad -2629]$$

- **Codificação da matriz da saída C:** A matriz C de saída é codificada de duas maneiras, através da multiplicação com o vetor *r* e pela multiplicação do vetor r^T pela matriz C. Estas duas operações que dão origem aos vetores C$r_{(3\times1)}$ e r^TC$_{(1\times3)}$ e são executadas pelo bloco denominado "Codificador de Saída".

$$C = \begin{bmatrix} -626 & 887 & 10658 \\ -722 & 2950 & -5393 \\ -1310 & -1405 & -7894 \end{bmatrix} \times r = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = Cr = \begin{bmatrix} 10919 \\ -3165 \\ -10609 \end{bmatrix}$$

$$r^T = [1 \quad 1 \quad 1] \times C = \begin{bmatrix} -626 & 887 & 10658 \\ -722 & 2950 & -5393 \\ -1310 & -1405 & -7894 \end{bmatrix} = r^T C = [-2658 \quad 2432 \quad -2629]$$

- **Comparação entre os vetores codificados:** A comparação se dá entre os vetores AB$r_{(3\times1)}$ com C$r_{(3\times1)}$ e r^TAB$_{(1\times3)}$ com r^TC$_{(1\times3)}$. Como pode ser observado os vetores codificados de entrada possuem exatamente os mesmos valores dos vetores codificados de saída. Isto indica que o circuito que executa o algoritmo de multiplicação de matrizes não foi atingido por nenhum *soft error.*

$$ABr_{(3\times1)} = \begin{bmatrix} 10919 \\ -3165 \\ -10609 \end{bmatrix} = \begin{bmatrix} 10919 \\ -3165 \\ -10609 \end{bmatrix} = Cr_{(3\times1)}$$

$$r^T AB_{(1\times3)} = [-2658 \quad 2432 \quad -2629] = [-2658 \quad 2432 \quad -2629] = r^T C_{(1\times3)}$$

Após a explicação do funcionamento da técnica com a suposição de que nenhuma falha tenha ocorrido, será modificado propositalmente o elemento $C_{(1,1)}$ da matriz C de modo a simular uma possível falha, alterando o valor correto +2950 para o valor -589, permitindo assim uma análise do comportamento da técnica nesta situação. Para isto, os cálculos da técnica foram refeitos considerando o elemento $c_{(1,1)}$ corrompido, conforme podemos observar na Figura 3.8.

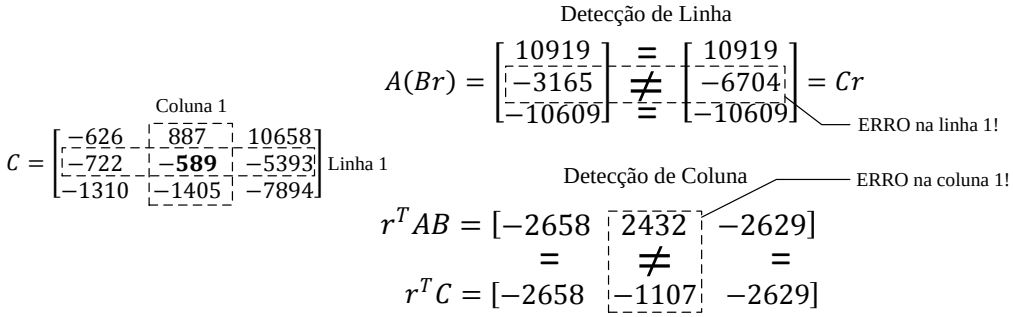


Figura 3.8: Análise do comportamento da técnica sob a presença de uma falha no elemento $C_{(1,1)}$.

Na Figura 3.8, pode ser verificar que a modificação do valor impactou diretamente na mudança dos elementos C$r_{(0,1)}$ e r^TC$_{(1,0)}$ dos vetores codificados de saída. Outro fato importante é que podemos concluir que a comparação entre os vetores ABr com Cr indica o número exato da linha onde o elemento corrompido está posicionado dentro da matriz C, da mesma forma que a comparação entre so vetores r^TAB com r^TC indica o número da coluna. Desta forma, podemos concluir que a técnica de *Freivalds Modificada* é capaz de detectar todo e qualquer erro simples, que modifica apenas um único elemento, dentro da matriz C. A Tabela 3.1 mostra um resumo das operações da técnica de *Freivalds Modificada* necessárias para a detecção da linha e coluna do elemento corrompido.

Operações	**Detecção de Linha**	**Detecção de Coluna**
Codificação das matrizes de entrada	Br	r^TA
Finalização dos vetores codificados de entrada	ABr	r^TAB
Codificação da matriz de saída	Cr	r^TC
Comparação entre os vetores codificados	ABr com Cr	r^TAB com r^TC

Tabela 3.1: Resumo das operações da técnica de *Freivalds Modificada* para detecção e indicação da linha e coluna do elemento corrompido.

3.3.3 Minimizando o custo de recomputação do elemento corrompido

Conforme o exemplo apresentado na subseção anterior, à técnica de *Freivalds Modificada* é capaz de detectar erros simples, além de indicar a linha e a coluna exata que apontam para o elemento corrompido. Uma vez conhecida esta posição do elemento corrompido dentro da matriz resultado, o próximo passo é a correção deste elemento. Uma forma de correção deste elemento se dá através da simples recomputação do mesmo, dada pela multiplicação de todos os elementos da linha apontada pela técnica por todos os elementos da coluna também apontada na etapa de detecção.

Entretanto, em (LISBÔA, 2009) foi mostrado uma forma de adaptar a técnica de *Freivalds Modificada* e torná-la capaz de corrigir o elemento corrompido da matriz com a execução de apenas duas operações de subtração. O primeiro passo é o cálculo da subtração entre os elementos divergentes dos vetores codificados de entrada e saída. Esta primeira operação de subtração é denominada de cálculo de *resíduo* e pode ser dividida em resíduo de linha e resíduo de coluna. O resíduo de linha leva este nome por ser gerado através da subtração entre os vetores codificados de detecção de linha e o resíduo de coluna por ser gerado pela subtração entre os vetores de detecção de coluna. Como pode ser observado, eles apresentam exatamente o mesmo valor.

1º Cálculo do resíduo:

$$resíduo\_linha = Cr_{(1,0)} - A(Br)_{(1,0)} = -6704 - (-3165) = -3539$$
$$resíduo\_coluna = r^T C_{(0,1)} - (r^T A)B_{(0,1)} = -1107 - (2432) = -3539$$

A segunda operação de subtração leva em consideração o resíduo para recalcular o elemento corrompido. Na verdade, basta subtrair o elemento corrompido pelo resíduo de linha ou de coluna para a perfeita restauração do elemento corrompido.

2º Subtração do elemento corrompido pelo resíduo previamente calculado:

$$Novo\_C_{(1,1)} = C_{(1,1)} - resíduo(linha\ ou\ coluna) = -589 - (-3539) = +2950$$

Como pode ser observado, apenas duas subtrações são suficientes para a restauração do valor corrompido, desde que sejam conhecidos a linha e coluna do elemento corrompido dentro da matriz C e que este mesmo elemento seja trazido da memória.

3.3.4 Paralelização da técnica para implementação em *hardware*

Com o entendimento da técnica de *Freivalds Modificada* e todas as operações matemáticas necessárias envolvidas no processo, que foram amplamente discutidas anteriormente neste capítulo, torna-se necessário fazer um levantamento de como as mesmas seriam executadas em conjunto com o algoritmo de multiplicação de matrizes a ser protegido. Um importante tópico de projeto consiste em decidir se a técnica implementada em hardware será executada de forma sequencial ou paralela com a execução do algoritmo. No caso de uma implementação sequencial, primeiramente seria necessário aguardar toda a operação do algoritmo de multiplicação de matrizes para que somente depois disso, sejam efetuadas as operações da técnica de *Freivalds modificada*. Obviamente, isto acarretaria num aumento no tempo total de execução de toda estrutura em comparação com a execução apenas do algoritmo de multiplicação de matrizes. Foi por este motivo, que um dos principais objetivos deste trabalho está em propor uma estrutura capaz de executar as operações da técnica de *Freivalds modificada* totalmente (ou o máximo possível) em paralelo com a execução da multiplicação de matrizes, visando assim evitar que a técnica comprometa o desempenho global do sistema.

De forma a facilitar o entendimento do processo de paralelização entre algoritmo e técnica de *Freivalds Modificada*, um diagrama é apresentado na Figura 3.9, cujo principal objetivo permitir uma análise das operações de multiplicação de matrizes e da técnica de uma forma lógica e temporal e da mecânica de interação e dependência entre elas.

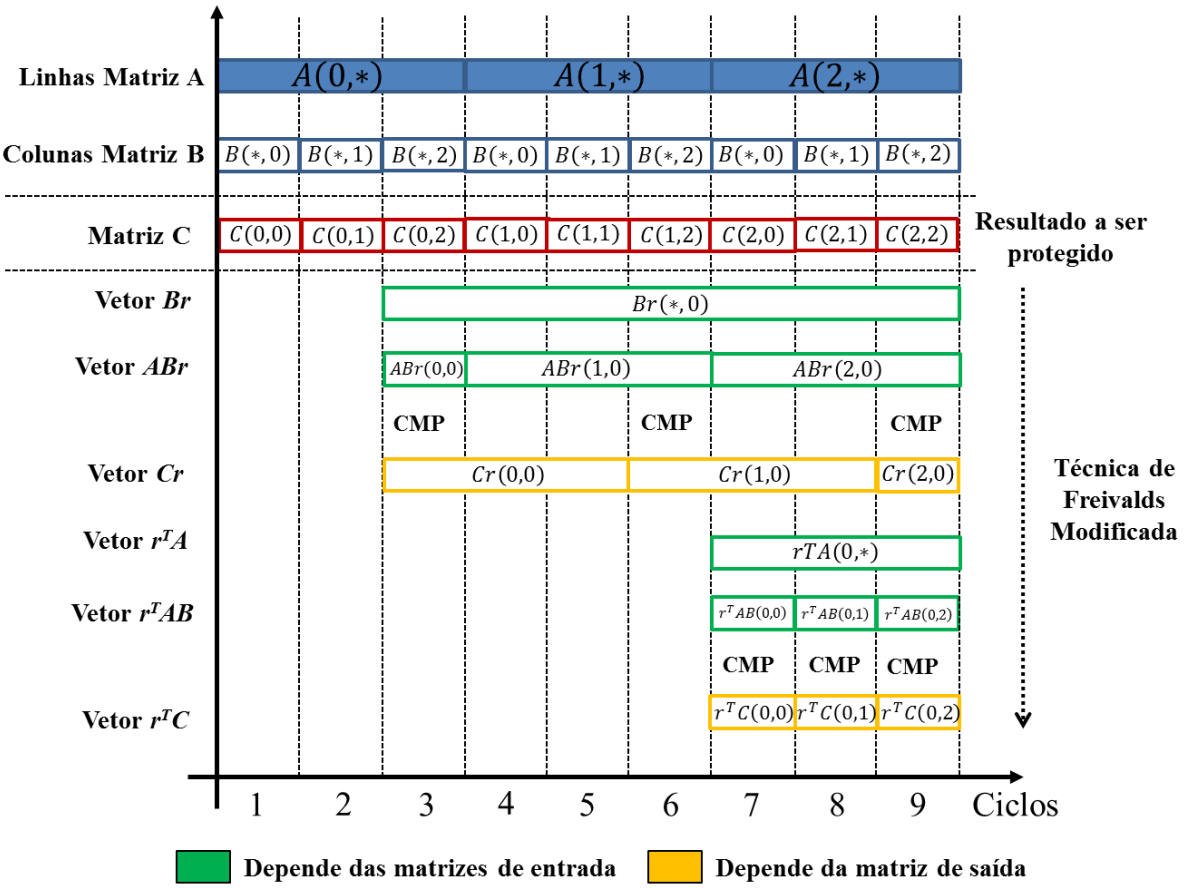


Figura 3.9: Interação entre o algoritmo de multiplicação de matrizes e a técnica de *Freivalds Modificada*.

No exemplo do diagrama, foi considerado uma multiplicação de duas matrizes A e B de dimensões 3×3, resultando na matriz C de mesmas dimensões e que deve ter seus elementos protegidos pela técnica proposta. Além disso, são mostrados como são calculados cada um dos vetores codificados de entrada e saída utilizados pela técnica, assim como os instantes onde são executadas as comparações entre os elementos destes vetores. A ideia por trás deste processo de paralelização das operações baseia-se em projetar um circuito responsável pelos cálculos da técnica que aproveite os dados da mesma maneira com que são repassados da memória ao algoritmo da multiplicação de matrizes. Para tal, considera-se que tanto os elementos das linhas da matriz A quanto os elementos das colunas da matriz B podem ser acessados simultaneamente. Neste caso, é necessário uma memória com 6 portas de leitura, 3 para fornecer os elementos das linhas da matriz A e 3 para fornecer os elementos das colunas da matriz B, e apenas 1 porta de escrita. No primeiro ciclo, por exemplo, considera-se que são carregados da memória os 3 elementos da primeira linha da matriz A ($A(0,*)$) e os 3 elementos da primeira coluna da matriz B ($B(*,0)$), que após multiplicados geram o primeiro elemento da matriz C ($C(0,0)$).

Na parte da técnica é possível perceber que o cálculo dos três elementos do vetor B*r* é finalizado no terceiro ciclo de *clock*, uma vez que para isto basta ser feita apenas uma única passagem por todas as colunas da matriz B, o que ocorre naturalmente devido ao algoritmo de multiplicação de matrizes. Depois de finalizado, os valores dos elementos deste vetor permanecem inalterados até o final do processo de multiplicação de

matrizes, apenas servindo de base para a geração da etapa de finalização do vetor codificado de entrada AB*r*. O vetor AB*r*, por sua vez têm seus elementos calculados a medida com que as linhas da matriz A mudam junto com o algoritmo de multiplicação de matrizes.

Os elementos que compõe o vetor Cr, por sua vez, dependem dos dados da matriz C gerado do produto entra as matrizes A e B. O primeiro elemento deste vetor $Cr_{(0,0)}$ é finalizado apenas quando os 3 elementos da primeira linha da matriz C são calculados. Após isto, deve ser feita a primeira comparação entre os elementos $ABr_{(0,0)}$ e $Cr_{(0,0)}$ no ciclo 3. Esta comparação, que na figura é indicada pela sigla CMP, em caso de igualdade garante a integridade dos 3 elementos que compõe a primeira linha da matriz C. Como pode ser observado também, os ciclos em que são realizadas estas comparações devem ser precisamente definidos, sob pena de comparar os valores errados e indicar erros inexistentes, comprometendo a eficácia da técnica empregada. As comparações entre os elementos destes vetores, como mostrado na Tabela 3.1, fazem parte processo de detecção de linha.

A parte das operações detecção de coluna, que consiste no cálculo e comparação do vetor codificado de entrada r^{T}AB com o vetor codificado de saída r^{T}C, é inicializada apenas nos últimos ciclos de *clock*. Isto ocorre pelo fato de que a primeira etapa de codificação da matriz A, através do cálculo do vetor r^{T}A, e codificação da matriz de saída C, através do cálculo do vetor r^{T}C, podem ser finalizados apenas quando o algoritmo de multiplicação de matrizes estiver utilizando a última linha da matriz A. A medida que são calculados, estes valores são comparados para verificar a integridade dos elementos das colunas da matriz C.

Como pôde ser observado, nesta análise simplificada, foram necessários 9 ciclos de *clock* para executar a multiplicação de duas matrizes de dimensões 3×3 além de executar em paralelo a técnica de detecção capaz de localizar algum elemento corrompido dentro da matriz C.

4 IMPLENTAÇÃO DA TÉCNICA DE FREIVALDS MODIFICADA EM HARDWARE

Como pôde ser visto na subseção 3.3.4, é possível a implementação de uma arquitetura que execute a técnica de *Freivalds modificada*, pelo menos teoricamente, de maneira totalmente paralela com a execução do algoritmo de multiplicação de matrizes (operação sob proteção). Entretanto, nesta análise que foi apresentada, é feita uma abordagem teórica e temporal que considera a hipótese de que existissem recursos infinitos para a execução completa do algoritmo e da técnica, como se fosse um grande circuito combinacional realizando muitas operações em paralelo em cada um dos ciclos de *clock*. Na prática, principalmente na área de sistemas embarcados, sabe-se que seria necessário pagar um alto preço em área e consequentemente potência para usufruir desta infinidade de recursos. A solução proposta, que será apresentada neste capítulo, é baseada na redução da quantidade de recursos de hardware (área e potência) disponíveis pagando um preço na redução do desempenho do sistema como um todo.

Neste capítulo, serão apresentados cada um dos circuitos digitais projetados para executar tanto o algoritmo de multiplicação de matrizes quanto as operações matemáticas da técnica de *Freivalds Modificada* apresentadas no capítulo 3. A descrição destes circuitos foi feita em VHDL, separada em módulos de acordo com a função que executam, num total de mais de 2500 linhas de código. A

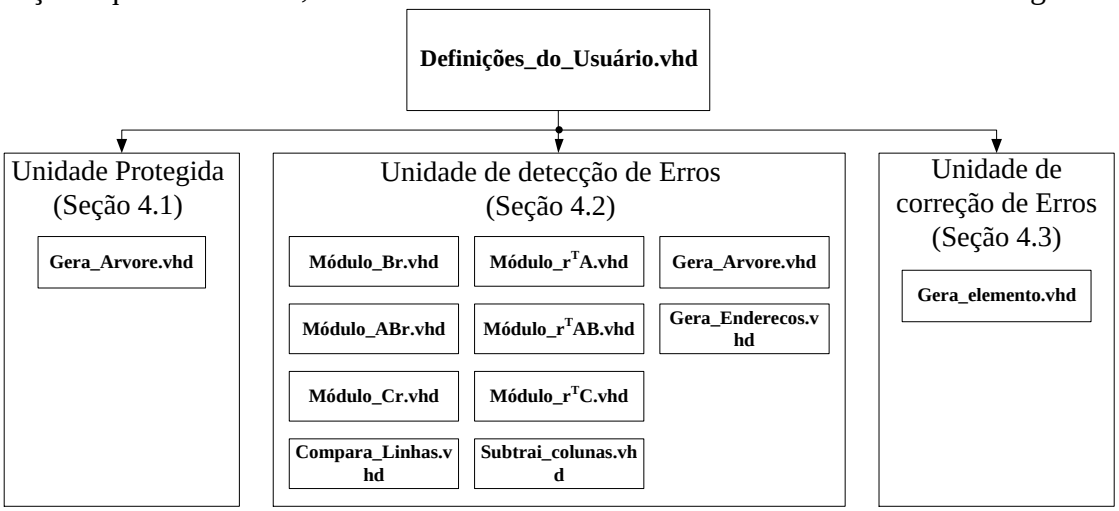


Figura 4.1, mostra um diagrama em blocos com a forma em que estes módulos descritos em VHDL estão divididos e organizados. A funcionalidade de cada um dos módulos e da estrutura completa foi testada através de simulações na ferramenta

Modelsim®. Esta divisão separa a arquitetura em três blocos principais, a unidade protegida que consiste na operação de multiplicação de matrizes que será mostrada na seção 4.1, a unidade de detecção de erros na seção 4.2 e finalmente a unidade de correção de erros na seção 4.3.

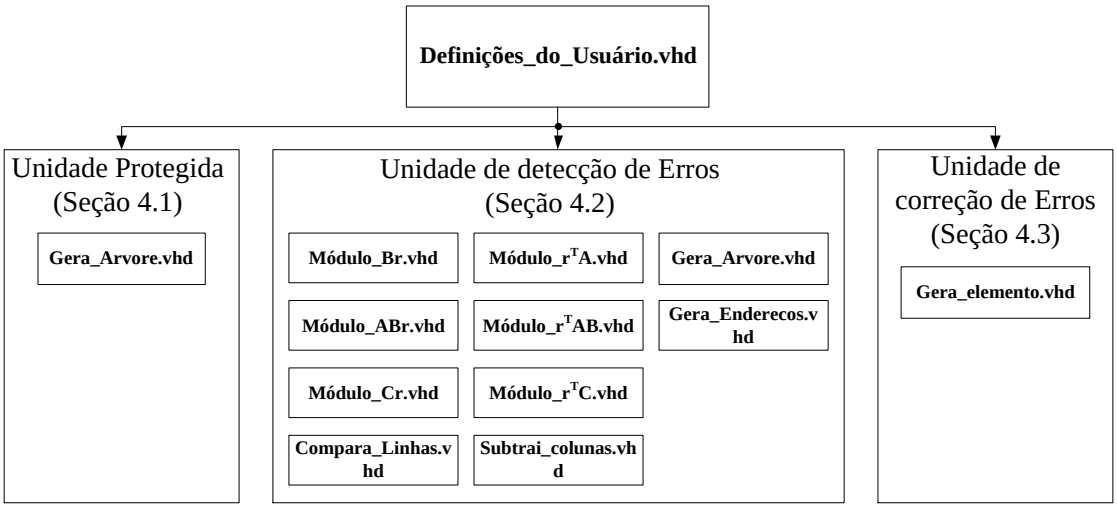


Figura 4.1: Estrutura organizacional da implementação da arquitetura em VHDL.

Uma vez tenha sido definida a forma com que os módulos descritos serão separados, é necessário definir a maneira com que estes módulos devem estar conectados entre sí. Pensando nisto, a implementação da arquitetura completa foi dividida em blocos conforme a função que executam dentro do sistema. Para esta arquitetura completa, foi dado o nome de RA^3 (do inglês *Resilient Adaptive Algebraic Architecture*), devido suas características funcionais que serão apresentadas. Na Figura 4.2 é mostrado um diagrama em blocos do fluxo de dados da estrutura completa com os três blocos principais em destaque.

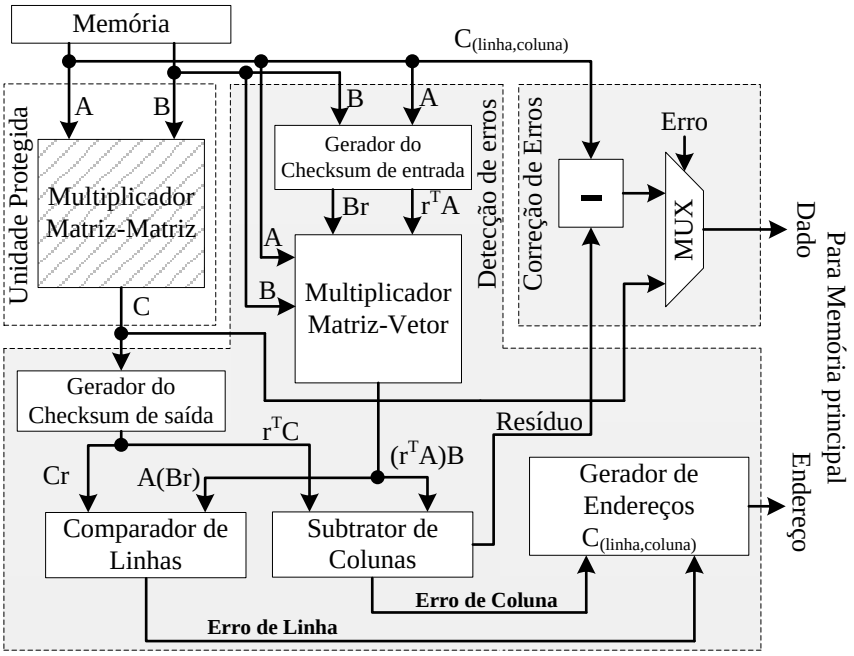


Figura 4.2: Fluxo de dados do RA^3.

4.1 Unidade Protegida

Este bloco é composto apenas pelo circuito denominado *Multiplicador Matriz-Matriz*, que têm como função executar o algoritmo de multiplicação de matrizes que estará em proteção. Este circuito tem conectado em suas entradas uma memória que possui uma ou mais portas específicas para cada matriz. No caso de uma estrutura com apenas um multiplicador no bloco *Multiplicador Matriz-Matriz*, a memória terá duas portas, uma para entregar apenas elementos pertencentes à matriz A e a outra somente para fornecer os elementos que compõe a matriz B.

A descrição deste circuito encontra-se dentro do arquivo *Gera_Árvore.vhd*, e tem como principal característica ser totalmente parametrizável em tempo de síntese. Dentro do arquivo *Definições_do_Usuário.vhd*, o usuário pode definir alguns parâmetros que serão utilizados no processo de síntese do RA^3, sendo que alguns deles são utilizados na geração do bloco *Multiplicador Matriz-Matriz*, como por exemplo:

- NUMERO_DE_MULTIPLICADORES: Define o número total de multiplicadores que serão sintetizados em paralelo na estrutura. Este número total, equivale ao número de multiplicadores dentro do módulo *Multiplicador Matriz-Matriz* somado ao número de multiplicadores dentro do módulo *Multiplicador Matriz-Vetor*. Estes dois módulos são exatamente idênticos, por isso ambos módulos apresentam o mesmo número de multiplicadores. Sendo assim, no caso do usuário decidir utilizar 16 multiplicadores no RA^3, o mesmo estará selecionando 8 multiplicadores para o algoritmo em proteção e 8 multiplicadores para técnica de *Freivalds*. Esta definição implica também no número de somadores que também serão sintetizados automaticamente em forma de árvore nos dois módulos.
- LARGURA_DOS_DADOS: Define a largura da palavra de dados dentro da arquitetura. Impacta no tamanho dos circuitos multiplicadores, somadores, registradores, entre outros.
- SOMADORES_EM_SERIE: Uma vez definido o número de multiplicadores, a árvore de somadores é automaticamente gerada. Em casos onde muitos multiplicadores são selecionados (visando exploração de paralelismo), são geradas árvores com grande profundidade, ou seja, com um grande número de somadores em série. Isto faz com que estas árvores acabem se tornando o caminho crítico da arquitetura. Pensando nisso, o projeto permite ao usuário definir o número máximo de somadores que serão colocados em série, inserindo de forma automática registradores de forma que não ultrapasse este valor.

Na Figura 4.3 pode ser observado o impacto que estes parâmetros causam na geração da arquitetura. Para que seja possível a automatização deste processo de síntese, utiliza-se o comando GENERATE em conjunto com os parâmetros selecionados. Ainda na figura, foi criado um cenário com a seguinte configuração no módulo *Multiplicador Matriz-Matriz*:

- NUMERO_DE_MULTIPLICADORES= 16
- LARGURA_DOS_DADOS= 32
- SOMADORES_EM_SERIE = 3

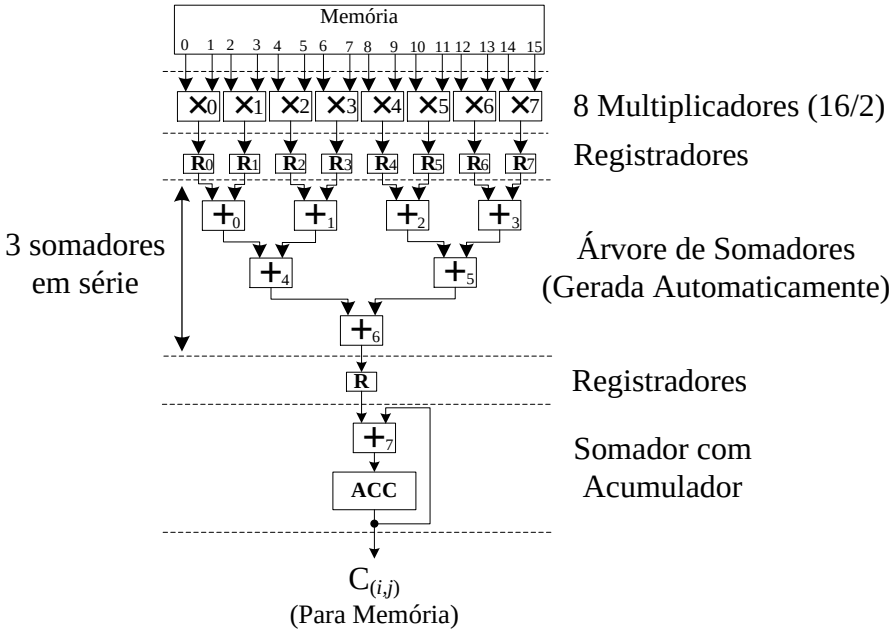


Figura 4.3: Exemplo do circuito *Multiplicador Matriz-Matriz* sintetizado com os parâmetros informados pelo usuário.

Para que fosse possível avaliar o atraso dos circuitos multiplicadores e somadores, foram realizados testes na ferramenta *Design Compiler* desenvolvida pela *Synopsys*® utilizando uma biblioteca da tecnologia de 90nm. Nestes testes foi possível comprovar que o atraso de 3 somadores em série são equivalentes ao atraso de 1 multiplicador para esta tecnologia. Mesmo assim, o parâmetro SOMADORES_EM_SERIE foi deixado configurável, para casos onde bibliotecas de outras tecnologias ou circuitos com outras topologias sejam utilizados e modifiquem está relação de 3 para 1 nos caminhos críticos destes circuitos.

Existe ainda uma unidade de controle associado a este módulo que controla principalmente os sinais de *RESET* dos registradores (R) e do acumulador (ACC) de acordo com as dimensões das matrizes envolvidas na operação. Existem basicamente três situações possíveis no uso do circuito *Multiplicador Matriz-matriz*, a primeira ocorre quando as matrizes envolvidas nas operações possuem dimensões menores do que o número de multiplicadores sintetizados, a segunda quando as matrizes envolvidas possuem as dimensões iguais ao número de multiplicadores e a terceira e última ocorre quando as dimensões das matrizes são maiores do que o número de multiplicadores sintetizados. Nestes três casos o que varia é basicamente a unidade de controle e forma com que envia os sinais a cada uma das unidades.

Em casos onde o número de multiplicadores é maior do que as dimensões das matrizes, como por exemplo, na multiplicação de duas matrizes de dimensões 4×4 utilizando o circuito da Figura 4.3, os multiplicadores (×) e registradores (R) 4, 5, 6 e 7 e os somadores 2, 3 e 5 permanecem ociosos durante toda a operação. Outra questão é que metade das portas de leitura da memória (portas 8 a 15), neste caso, não estão fornecendo dados de interesse na operação. Neste caso específico, a unidade de controle mantém os registradores 4, 5, 6 e 7 zerados para que a segunda entrada do somador 6, proveniente das unidades funcionais não operantes, esteja em zero de forma a não influenciar no resultado do somador 6.

No caso onde as dimensões das matrizes são iguais ao número de multiplicadores, por exemplo, na multiplicação de duas matrizes 8×8, todos os multiplicadores e somadores estarão em atividade constante, todas as portas de memória estarão fornecendo dados úteis ao algoritmo, e por fim, nenhum registrador estará sendo *resetado* pelo controle. Neste caso o acumulador estará exercendo apenas a função de um registrador sem receber o sinal de acumulação por parte do controle.

Finalmente nos casos em que as dimensões são maiores do que o número de multiplicadores, todos os recursos estão sempre sendo utilizados, com a diferença que neste caso o acumulador (ACC) será colocado no modo de acumulação para que seja capaz do somar os produtos parciais gerados pela estrutura. Considerando, por exemplo, a operação C=A×B onde as dimensões das matrizes A e B são 16×16, conforme mostrado na Figura 4.4, sendo executada pelo circuito mostrado na Figura 4.3, teremos que os cálculos dos elementos da matriz C serão executados em duas etapas distintas.

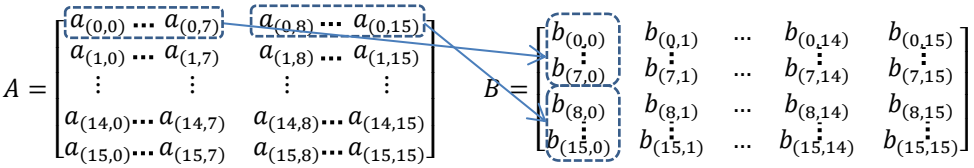


Figura 4.4: Exemplo de funcionamento do algoritmo de multiplicação de matrizes.

Analisando o cálculo específico do elemento $c_{(0,0)}$, na equação 5, podemos observar que o mesmo é calculado através da multiplicação da linha *zero* da matriz A ($a_{(0,0)}$ até $a_{(0,15)}$) pela coluna *zero* da matriz B ($b_{(0,0)}$ até $b_{(15,0)}$). Podemos assim concluir que, para matrizes destas dimensões, são necessárias dezesseis multiplicações seguido de quinze operações de soma para o cálculo de cada um dos elementos da matriz C.

$$c_{(0,0)} = \left[\left(a_{(0,0)} \times b_{(0,0)}\right) + \left(a_{(0,1)} \times b_{(1,0)}\right) + \cdots + \left(a_{(0,15)} \times b_{(15,0)}\right)\right] \qquad (5)$$

Sendo assim, pensando no circuito da Figura 4.3, é possível verificar que para o cálculo de cada um dos elementos da matriz C, é necessário utilizar duas vezes a estrutura deste circuito, somando os dois resultados intermediários, de forma a gerar o elemento $c_{(0,0)}$. Para este fim, é utilizado o acumulador (ACC), que armazena a primeira versão do resultado $c'_{(0,0)}$ com a segunda versão $c''_{(0,0)}$, finalmente gerando o elemento $c_{(0,0)}$. As equações abaixo demonstram esta geração do elemento $c_{(0,0)}$ em duas parcelas.

$$c'_{(0,0)} = \left(a_{(0,0)} \times b_{(0,0)}\right) + \cdots + \left(a_{(0,7)} \times b_{(7,0)}\right) \rightarrow ACC = c'_{(0,0)}$$

$$c''_{(0,0)} = \left(a_{(0,8)} \times b_{(8,0)}\right) + \cdots + \left(a_{(0,15)} \times b_{(15,0)}\right) \rightarrow ACC = ACC + c''_{(0,0)} = c_{(0,0)}$$

4.2 Unidade de Detecção de Erros

Esta unidade foi acoplada a unidade a ser protegida com o objetivo de supervisionar, em paralelo com a execução da multiplicação de matrizes, eventuais erros que possam vir a ocorrer devido a partículas ionizantes. A unidade recebe como entrada os elementos das matrizes A e B, da mesma forma em que os mesmos são enviados a unidade protegida. Esta unidade também recebe como entrada os elementos da matriz C, que são produtos gerados pela unidade protegida.

A unidade de detecção pode ser dividida em quatro grupos principais, de acordo com a função que executam dentro desta unidade. Um breve resumo de cada um destes grupos é apresentando abaixo:

- **Geração dos Vetores Codificados de Entrada**: Consiste na codificação dos dados das matrizes A e B em vetores. Esta codificação é feita em duas etapas, até a geração dos vetores ABr e r^{T}AB.
- **Geração dos vetores codificados de Saída**: Consiste na codificação da matriz resultado C em dois vetores denominados Cr e r^{T}C.
- **Comparadores de Linha e Coluna e Gerador de Resíduo**: Possui a função de comparar os vetores codificados de entrada com os de saída para averiguar a ocorrência de um possível elemento corrompido dentro da matriz C. O resíduo de linha também é calculado de forma a permitir que o cálculo de correção do elemento corrompido seja feito através de uma única operação de subtração.
- **Geração do endereço do elemento corrompido**: À medida que os elementos da matriz C são calculados, os mesmos são diretamente escritos na memória. Ao final do algoritmo, no caso em que tenha sido detectado algum elemento corrompido, é necessário que este elemento seja trazido da memória para então seja feita a correção pela unidade de detecção de erros.

A seguir, será apresentado o funcionamento de cada uma das unidades que compõe a unidade de detecção de erros.

4.2.1 Primeira fase da geração dos vetores codificados de entrada

O objetivo desta etapa está em gerar automaticamente circuitos capazes de calcular os vetores Br e r^{T}A, de forma que possam ser acoplados a unidade protegida utilizando os dados provenientes da memória da mesma forma com que são enviados a unidade protegida. Isto faz com que não seja necessário modificar em nada o algoritmo de multiplicação de matrizes em proteção. As descrições de cada um destes circuitos podem ser encontradas nos arquivos *Modulo_Br.vhd* e *Modulo_r^{T}A.vhd*.

Como foi mostrado na seção 3.3.2, o calculo do vetor Br pode ser executado através da soma de todos os elementos que compõe cada uma das linhas da matriz B, assim como o cálculo de r^{T}A pode ser feito através da soma dos elementos de cada uma das colunas da matriz A. Isto permite que sejam utilizados circuitos somadores ao invés de multiplicadores para realização desta tarefa. Na Figura 4.5, é mostrado como é o circuito projetado para executar a soma dos elementos das colunas da matriz A. No exemplo da figura, é considerado que o usuário tenha definido que seriam utilizados apenas dois multiplicadores na estrutura, fazendo que com o algoritmo de multiplicação de matrizes possua apenas um único multiplicador ($\times_{\mathbf{0}}$). Sendo assim, a memória utilizada deverá apresentar duas portas de leitura, uma dedicada aos elementos da matriz A (porta 0) e a outra dedicada apenas a fornecer os elementos da matriz B (porta 1). No mesmo exemplo, considera-se uma operação onde a matriz A possui 3 colunas. Neste caso é necessário a colocação de 3 acumuladores, um para acumular os elementos de cada uma das colunas. É importante destacar que o número de acumuladores colocados nesta estrutura não depende do número de multiplicadores escolhido para a arquitetura e sim das dimensões máximas permitida para as matrizes em operação. Estas dimensões máximas podem também ser configuradas em tempo de síntese da arquitetura RA^{3} pelo usuário no arquivo *Definições_do_Usuário.vhd*.

A medida com que os elementos da matriz A são enviados ao algoritmo de multiplicação de matrizes, os mesmos também são enviados ao circuito de cálculo do vetor r^{T}A. Como pode ser observado, o fato de que a porta 0 da memória fornece exclusivamente os elementos da matriz A, faz com que os dados provenientes desta porta sejam a entrada deste circuito. Estes elementos à medida que são enviados devem

ser somados apenas com os outros elementos da mesma linha, que por sua vez se encontram em cada um dos acumuladores (ACC) do circuito. Para que isto seja possível, tantos os sinais de habilitação dos acumuladores (en0, en1 e en2) quanto os bits de seleção do multiplexador (MUX) devem a cada ciclo serem controlados corretamente. Para isto existe uma unidade de controle para este circuito, de forma a gerenciar a correta colocação dos sinais em cada uma das unidades.

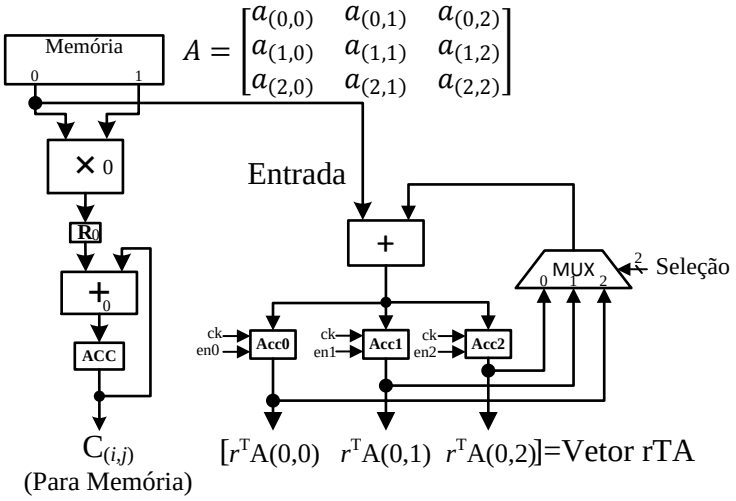


Figura 4.5: Circuito projetado para cálculo do vetor r^TA conectado ao *Multiplicador Matriz-Matriz*.

A unidade de controle do módulo r^TA, se baseia na coluna em que o elemento que está instantaneamente está na porta de memória pertence. Sabendo isto, é possível habilitar o acumulador correto e os bits de seleção do multiplexador para receber o valor atualizado. A Tabela 4.1 mostra como são controlados os sinais de *enable* de cada um dos acumuladores (ACC) e dos bits de seleção do multiplexador (sel0 e sel1) para que os acumuladores funcionem de maneira correta. Os dados desta tabela são baseados numa multiplicação de duas matrizes de dimensões 3×3 sendo executada no circuito da Figura 4.5, totalizando exatos 27 ciclos de clock. Apenas os ciclos onde são efetuadas as modificações nos sinais de controle dos registradores e multiplexador são mostrados na tabela.

Ciclo	**en0**	**en1**	**en2**	**Sel 0**	**Sel 1**	**Saídas do Módulo r^TA**		
						r^TA(0,0)	**r^TA(0,1)**	**r^TA(0,2)**
1	1	0	0	1	0	**Acc0+=*a*(0,0)**	Acc1=0	Acc2=0
2	0	1	0	0	1	Acc0=Acc0	**Acc1+=*a*(0,1)**	Acc2=0
3	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*a*(0,2)**
10	1	0	0	1	0	**Acc0+=*a*(1,0)**	Acc1=Acc1	Acc2=Acc2
11	0	1	0	0	1	Acc0=Acc0	**Acc1+=*a*(1,1)**	Acc2=Acc2
12	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*a*(1,2)**
19	1	0	0	1	0	**Acc0+=*a*(2,0)**	Acc1=Acc1	Acc2=Acc2
20	0	1	0	0	1	Acc0=Acc0	**Acc1+=*a*(2,1)**	Acc2=Acc2
21	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*a*(2,2)**

Tabela 4.1: Sinais de Controle para o cálculo do vetor r^TA.

Nos três primeiros ciclos de *clock*, são lidos da memória os três elementos da primeira linha da matriz A. Como o objetivo é somar os elementos das colunas desta matriz, no primeiro ciclo de *clock* apenas o primeiro acumulador (Acc0) deverá estar

habilitado, no segundo ciclo apenas o segundo acumulador (Acc1) e no terceiro ciclo apenas o terceiro acumulador (Acc2). Os bits de seleção do multiplexador são controlados para que seja possível somar o valor atual dos acumuladores com os valores passados. Dos ciclos 4 a 9, a mesma primeira linha continua a ser utilizada pelo algoritmo de multiplicação de matrizes, que como já teve seus elementos guardados nos acumuladores não é necessário guarda-los novamente. Nos ciclos 10,11 e 12 os três elementos da segunda linha da matriz A estarão na porta de memória respectivamente. Nestes ciclos, é preciso efetuar o mesmo controle que foi executado nos ciclos 1, 2 e 3 para que então seja feito a soma correta entre elementos de mesma coluna. Finalmente nos ciclos 19, 20 e 21 começarão a ser buscados da memória os elementos da terceira linha onde o mesmo ciclo de controle deve ser reiniciado de forma a finalizar o processo de geração do vetor r^{T}A.

O processo necessário para o cálculo do vetor B*r* é muito parecido com o processo do cálculo r^{T}A apresentado anteriormente. O circuito é exatamente igual, como pode ser visto na Figura 4.6, entretanto a diferença se dá no fato que no cálculo do vetor B*r* são os elementos de cada uma das linhas da matriz B é que devem ser somados. O circuito para o calculo do vetor B*r* conectado ao módulo Multiplicador *Matriz-Matriz* é mostrado na Figura 4.6.

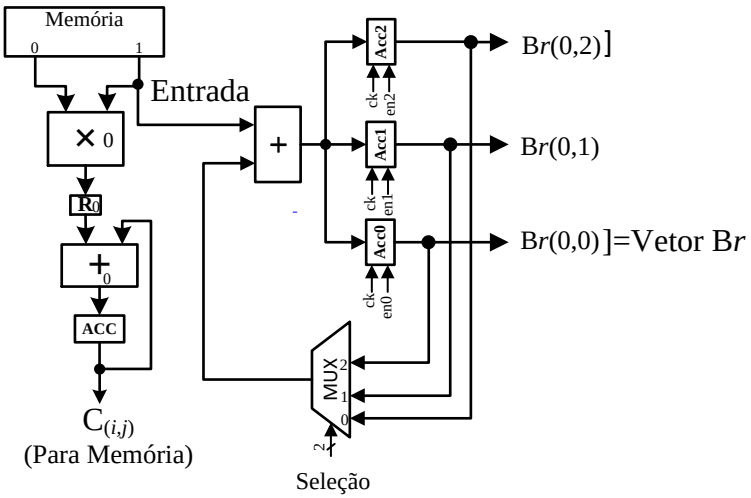


Figura 4.6: Circuito projetado para o cálculo do vetor B*r* conectado ao *Multiplicador Matriz-Matriz*.

A real diferença entre o cálculo do vetor B*r* com relação ao calculo do vetor r^{T}A se dá no módulo de controle. Como já discutido anteriormente, o algoritmo de multiplicação de matrizes mantém a primeira linha da matriz A enquanto que percorre todas as colunas da matriz B, fazendo com que todos os elementos da matriz B sejam lidos nos primeiros 9 ciclos de *clock*. Após o nono ciclo de *clock*, o vetor B*r* está pronto e já pode ser repassado para segunda fase da geração dos vetores codificados de entrada.

Ciclo	**en0**	**en1**	**en2**	**Sel 0**	**Sel 1**	**Saídas do Módulo B*r***		
						B*r*(0,0)	**B*r*(1,0)**	**B*r*(2,0)**
1	1	0	0	1	0	**Acc0+=*b*(0,0)**	Acc1=0	Acc2=0
2	0	1	0	0	1	Acc0=Acc0	**Acc1+=*b*(1,0)**	Acc2=0
3	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*b*(2,0)**
4	1	0	0	1	0	**Acc0+=*b*(0,1)**	Acc1=Acc1	Acc2=Acc2
5	0	1	0	0	1	Acc0=Acc0	**Acc1+=*b*(1,1)**	Acc2=Acc2

6	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*b*(2,1)**
7	1	0	0	1	0	**Acc0+=*b*(0,2)**	Acc1=Acc1	Acc2=0
8	0	1	0	0	1	Acc0=Acc0	**Acc1+=*b*(1,2)**	Acc2=0
9	0	0	1	0	0	Acc0=Acc0	Acc1=Acc1	**Acc2+=*b*(2,2)**

Tabela 4.2: Sinais de Controle para o cálculo do vetor Br.

A geração automática dos circuitos que calculam os vetores r^{T}A e B*r*, assim como a geração do módulo *Multiplicador Matriz-Matriz*, devem respeitar os parâmetros informados pelo usuário dentro arquivo *Definições_do_Usuario.vdh*. Podemos observar que o número de somadores e multiplexadores necessários na geração automática destes dois módulos é exatamente o mesmo que o número de multiplicadores escolhido pelo usuário no módulo *Multiplicador Matriz-Matriz*. Já o número de entradas dos multiplexadores depende de uma divisão do número de acumuladores pelo número de multiplicadores. O número de acumuladores, por sua vez, também deve ser informado no processo de síntese da arquitetura, sabendo que este número delimitará as dimensões máximas em número de linhas e de colunas que a arquitetura será capaz de processar. Por este motivo, foi criado um parâmetro denominado DIMENSOES_MAXIMAS dentro do arquivo *Definições_do_Usuario.vdh*, que permite que o usuário escolha em tempo de síntese o número máximo de linhas e colunas permitido para as matrizes envolvidas. Caso o usuário defina que vai operar com matrizes de dimensões máximas iguais a 32×32, ele deverá selecionar o parâmetro DIMENSOES_MAXIMAS para 32, fazendo com que sejam automaticamente inseridos 32 acumuladores para o circuito r^{T}A (32 colunas da matriz A) mais 32 acumuladores para o circuito B*r* (32 linhas da matriz B). Uma vez sintetizado o circuito, matrizes de dimensões menores poderão operar normalmente na arquitetura, uma vez que o módulo de controle destes circuitos controla tanto os sinais de controle dos multiplexadores quando os sinais dos acumuladores de acordo com as dimensões das matrizes em operação.

A Figura 4.7, mostra um circuito para o cálculo do vetor r^{T}A de acordo com os parâmetros também mostrados na figura. Como visto anteriormente, o número de somadores e multiplexadores é o mesmo que número de multiplicadores. O número de acumuladores necessários para o circuito r^{T}A, neste exemplo foi selecionado para oito, fazendo com que cada multiplexador tenha quatro entradas.

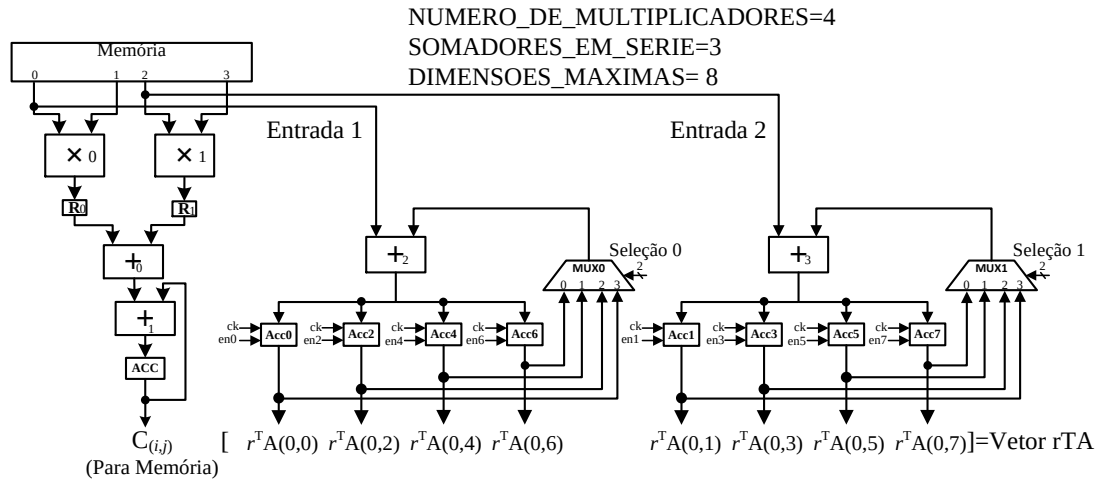


Figura 4.7: Exemplo de uma configuração do circuito r^{T}A acoplado ao circuito *Multiplicador Matriz-Matriz*.

4.2.2 Finalização dos vetores codificados de entrada

O processo de finalização da codificação dos vetores de entrada envolve duas operações de multiplicação entre matrizes e vetores, mais especificamente, entre a matriz A com o vetor Br e entre o vetor r^{T}A com a matriz B. Como os vetores envolvidos não são compostos apenas por elementos iguais a um, estas operações de multiplicação não podem ser substituídas por operações de somas de linhas e colunas. Embora a operação de multiplicação de matrizes por vetores sejam operações que demandem um tempo menor do que multiplicações entre matrizes, o circuito que as executa é exatamente o mesmo. Sendo assim, o circuito utilizado neste módulo será uma cópia exata do circuito utilizado pela unidade protegida, descrito no arquivo *Gera_Arvore.vhd*. Por este motivo, quando definimos o número total de multiplicadores da arquitetura no parâmetro NUMERO_DE_MULTIPLICADORES, estamos na verdade definindo o número de multiplicadores da unidade *Multiplicador Matriz-Matriz* mais o número de multiplicadores na unidade *Multiplicador Matriz-Vetor* utilizado nesta etapa.

Nesta etapa são introduzidas as duas primeiras penalidades em termos de desempenho pela técnica na execução completa da arquitetura. Estas penalidades ocorrem por que o algoritmo de multiplicação de matrizes deve ser parado duas vezes por determinados intervalos de tempo para que alguns cálculos da técnica consigam ser finalizados. Uma vez estes cálculos geradores das penalidades tenham sido finalizados, o algoritmo de multiplicação de matrizes continua com sua execução normalmente.

A primeira penalidade ocorre no momento da transição da primeira para a segunda linha da matriz A como pode ser verificado na Figura 4.8. A primeira linha da matriz A deve ser lida uma vez a mais do que seria necessário na execução apenas do algoritmo de multiplicação de matrizes.

- **Passo 1:** O algoritmo de multiplicação é congelado momentaneamente, ou seja, nenhum elemento da matriz C será gerado. Neste instante apenas os elementos da matriz A serão lidos da memória.
- **Passo 2:** Ao invés de buscar o primeiro elemento da segunda linha da matriz A visando dar continuidade na execução da multiplicação, os elementos da primeira linha serão utilizados uma última vez.
- **Passo 3:** Neste passo a primeira linha da matriz A é lida pela última vez para seja então realizada a operação $ABr_{(0,0)}=a_{(0,0)}\times Br_{(0,0)}+ a_{(0,1)}\times Br_{(1,0)}+ a_{(0,2)}\times Br_{(2,0)}$ no módulo *Multiplicador Matriz-Vetor*.

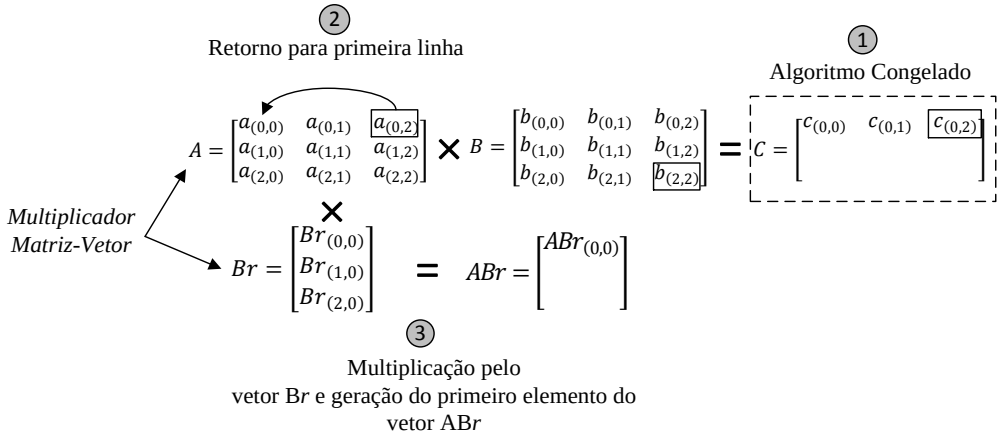


Figura 4.8: Primeira penalidade imposta ao algoritmo de multiplicação de matrizes.

Apenas após a execução dos três passos demonstrados anteriormente que a segunda linha da matriz pode ser buscada para que o algoritmo de multiplicação de matrizes continue normalmente.

Esta primeira penalidade pode ser modelada em número de ciclos de *clock* baseado nos dados da arquitetura e do número de colunas da matriz A, conforme mostrado na equação 6.

$$Penalidade\ 1 = \frac{M_A}{\left(\frac{NUMERO\_DE\_MULTIPLICADORES}{2}\right)} \tag{6}$$

Na equação 6, o parâmetro M_A equivale ao número de colunas da matriz A e o campo NUMERO_DE_MULTIPLICADORES é dividido por dois, pois apenas metade dos multiplicadores estão no módulo *Multiplicador Matriz-Vetor* utilizado nesta etapa.

A segunda penalidade inserida pela técnica ao algoritmo de multiplicação de matrizes, em proteção, se dá pela necessidade de se calcular o primeiro elemento do vetor $r^T AB_{(0,0)}$. Como é sabido, este primeiro elemento é calculado através da multiplicação do vetor $r^T A$ pela primeira coluna da matriz B.

Como podemos observar na Figura 4.9, foi criado um conjunto de passos para facilitar o entendimento da causa da segunda penalidade gerada pela técnica ao desempenho global do sistema.

- **Passo 1:** Parar a execução do algoritmo de multiplicação de matrizes após a geração do primeiro elemento da última linha da matriz C, no caso do exemplo da figura após a geração do elemento $c_{(2,0)}$. A partir detse instante apenas os elementos da matriz B serão utilizados.

- **Passo 2:** Após a geração do elemento $c_{(2,0)}$, o algoritmo de multiplicação de matrizes naturalmente estaria buscando da memória os elementos da segunda coluna da matriz B. Entretanto, neste instante a mesma primeira coluna da matriz B é trazida da memória para uma última vez.
- **Passo 3:** Utilizando novamente os elementos da primeira coluna da matriz B é possível calcular o elemento $r^{T}AB_{(0,0)} = b_{(0,0)} \times r^{T}A_{(0,0)} + b_{(1,0)} \times r^{T}A_{(0,1)} + b_{(2,0)} \times r^{T}A_{(0,2)}$ utilizando o módulo *Multiplicador Matriz-Vetor*.

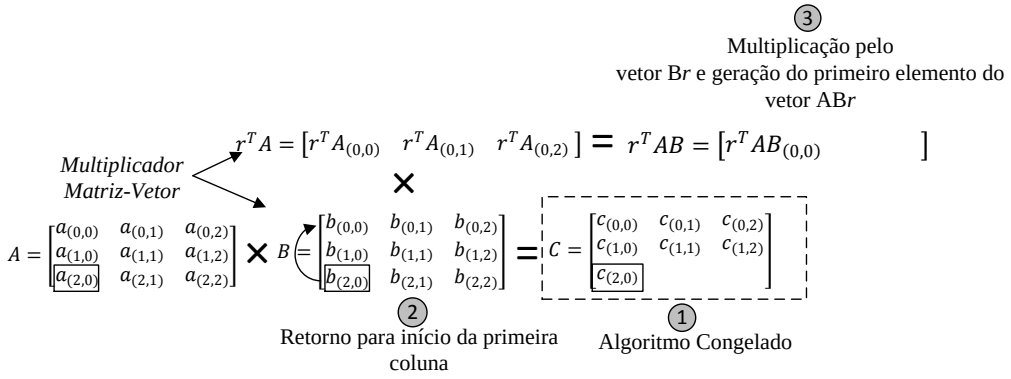


Figura 4.9: Segunda penalidade imposta ao algoritmo de multiplicação de matrizes.

A segunda penalidade pode ser modelada em número de ciclos de *clock* baseado nos dados da arquitetura e do número de linhas da matriz B, conforme mostrado na equação 7.

$$Penalidade\ 2 = \frac{N_B}{\left(\frac{NUMERO\_DE\_MULTIPLICADORES}{2}\right)} \tag{7}$$

Uma análise mais aprofundada dos custos impostos pela técnica, da maneira com que foi implementada no trabalho, será apresentada na seção 4.4. Nesta seção será apresentada uma equação que permite avaliar estes custos de acordo com os parâmetros que foram escolhidos pelo usuário da arquitetura e levando em consideração as duas penalidades aqui mostradas.

4.2.3 Geração dos vetores codificados de saída

Como discutido anteriormente, os vetores codificados de saída C*r* e $r^{T}C$ são gerados pelo bloco "Codificador dos vetores de Saída", que por sua vez, recebem os elementos resultantes do algoritmo de multiplicação de matrizes diretamente do módulo *Multiplicador Matriz-Matriz*. A geração do vetor C*r* é realizada através de um circuito composto apenas por um único somador conectado a um registrador. A simplicidade da implementação deste circuito é devido às características do algoritmo de multiplicação de matrizes, que entrega os resultados (elementos da matriz C) linha após linha. Como o cálculo de cada um dos elementos do vetor C*r* consiste na soma de todos os elementos que compõe cada uma das linhas da matriz C, basta o circuito acumular os resultados recebidos até o último elemento de cada uma das linhas. Quando o primeiro elemento de uma nova linha entrar no codificador dos vetores de saída (elementos $C_{(*,0)}$), o registrador (acumulador) deverá ser zerado para que resultados de linhas diferentes não se misturem na geração dos elementos do vetor C*r*. Este sinal que zera o acumulador C*r*, é comandado pela unidade de controle local deste bloco, e varia de acordo com a configuração do RA^{3} (número de multiplicadores escolhidos) e com as dimensões das

matrizes envolvidas na multiplicação. A necessidade pelo uso de apenas um único registrador, e não um registrador para cada uma das linhas da matriz C, se justifica pelo fato de que ao finalizar o calculo de um elemento do vetor C*r*, o mesmo é instantaneamente comparado com o elemento de mesmo índice do vetor AB*r*. Esta comparação é suficiente para garantir que os elementos de uma linha, que já estão armazenados dentro da memória, não estão corrompidos e por este motivo não é necessário que sejam armazenados elementos do vetor C*r* após a comparação.

A Figura 4.10, mostra (à esquerda) os resultados da matriz C sendo recebidos e acumulados interativamente com o algoritmo de multiplicação de matrizes para à geração do vetor C*r*. Na mesma figura (à direita), é mostrado o circuito para cálculo do vetor C*r*, composto por um somador seguido de um registrador para acumular os resultados das linhas.

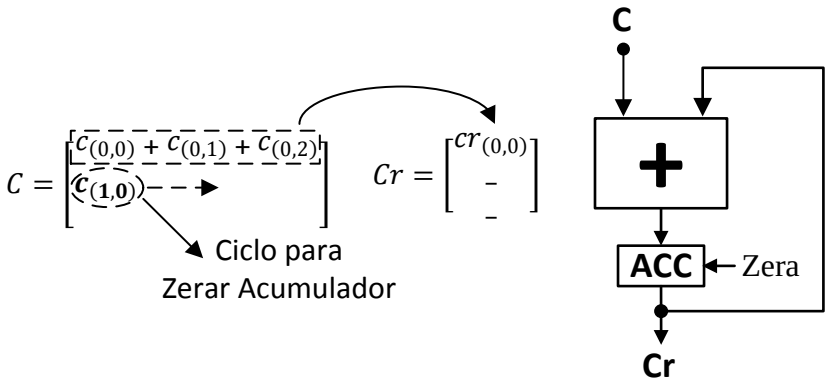


Figura 4.10: Operação e circuito para cálculo do vetor C*r*.

O segundo vetor codificado de saída r^TC, é calculado da mesma forma que o vetor r^TA. Isto acontece pelo fato de que tanto a matriz de entrada A quanto a matriz de saída C são lidas e escritas, respectivamente, linha por linha enquanto que são os elementos das colunas que devem ser acumulados. Como pode ser observado na Figura 4.11, é necessário apenas um somador conectado a três acumuladores (ACC), que por sua vez têm suas saídas conectadas a um multiplexador. O sinal *seleção* é comandado pelo controle, e seleciona qual o acumulador que deve ser somado com o elemento da matriz C atualmente da entrada do circuito. A mesma unidade de controle dedicada deste módulo também comanda os sinais de *enable (en)* de cada um dos acumuladores individualmente. Ao final do cálculo da matriz C, cada um dos acumuladores conterão os valores mostrados na Tabela 4.3.

Acumulador	**Elemento do vetor r^TC**	**Valor Acumulado**
Acc0	$r^TC_{(0,0)}$	$C_{(0,0)}+ C_{(1,0)}+ C_{(2,0)}$
Acc1	$r^TC_{(0,1)}$	$C_{(0,1)}+ C_{(1,1)}+ C_{(2,1)}$
Acc2	$r^TC_{(0,2)}$	$C_{(0,2)}+ C_{(1,2)}+ C_{(2,2)}$

Tabela 4.3: Valores armazenados em cada um dos acumuladores ao final da multiplicação de matrizes.

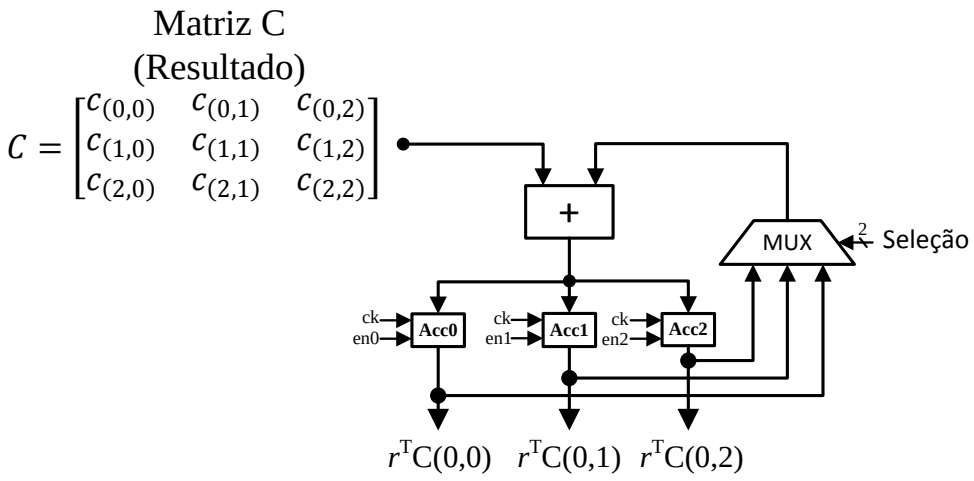


Figura 4.11: Circuito para cálculo do vetor codificado de saída $r^T C$.

4.2.4 Comparador de Linhas e Subtrator de colunas

Neste processo de apresentação da arquitetura foram apresentados até aqui, os circuitos geradores dos vetores codificados de entrada e saída e a forma com que operam em paralelo com a execução do algoritmo de multiplicação de matrizes. Interativamente com estes cálculos, o processo de comparação entre estes elementos deve estar ocorrendo. Será nos casos em que houver discordância de valores é que os erros poderão ser detectados.

O bloco denominado de *Comparador de Linhas* é responsável pela comparação dos elementos de mesmo índice dos vetores AB*r* e C*r*. Os instantes de tempo onde estas comparações devem ser executadas devem ser sincronizados pelo controle deste módulo através do sinal *Compara*. Isto é necessário devido ao fato de que os valores destes dois vetores ficam armazenados apenas durante um curto intervalo de tempo até serem sobrescritos.

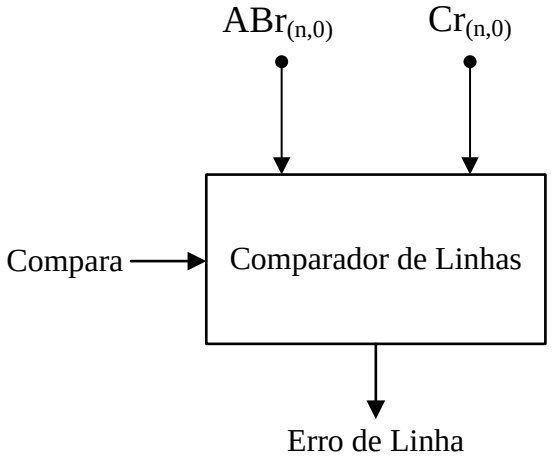


Figura 4.12: Diagrama do circuito comparador de linhas.

n	**Compara**	**Elementos a Comparar**		**Situação**
X	0	X	X	Nada será comparado
0	1	$ABr_{(0,0)}$	$Cr_{(0,0)}$	Verifica Integridade da **primeira** linha

1	1	$ABr_{(1,0)}$	$Cr_{(1,0)}$	Verifica Integridade da **segunda** linha
2	1	$ABr_{(2,0)}$	$Cr_{(2,0)}$	Verifica Integridade da **terceira** linha

Tabela 4.4: Resumo do funcionamento do modulo *Comparador de Linhas*.

O controle deste bloco comanda as operações através do sinal *Compara*, que é colocado em nível lógico alto apenas nos ciclos em que ambos os elementos estão prontos para ser comparados. Quando uma comparação apontar uma divergência entre elementos, significa que a linha contendo o elemento corrompido foi encontrada e que novas comparações de linha não são mais necessárias. Neste mesmo ciclo, o sinal *Erro de Linha* é colocado em nível lógico alto, que por sua vez é repassado ao módulo *Gerador de Endereços*.

Uma vez tenham sido testadas as linhas, as colunas também deverão ser testadas. Para isto, além de uma comparação, como é feito nas linhas, deverá ser feita uma subtração entre os elementos de mesmo índice dos vetores $r^{T}AB$ e $r^{T}C$. Esta subtração representa o cálculo de resíduo de linha, apresentado na subseção 3.3.3, que tem como finalidade ser utilizado no processo de correção do elemento corrompido. Como foi abordada nesta mesma subseção, existem os resíduos de linha e coluna, e que ambos quando calculados apresentam o mesmo valor. Sendo assim, não existe a necessidade do resíduo ser calculado de duas formas diferentes, por isso, apenas o resíduo de linha é calculado pela arquitetura RA^{3}.

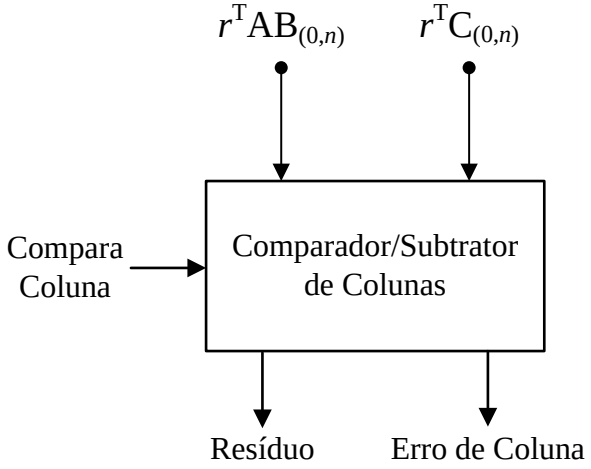


Figura 4.13: Diagrama do circuito Comparador/Subtrator de Colunas.

A Tabela 4.5 mostra de maneira simplificada, o funcionamento do controle da unidade "Comparador/Subtrator de Colunas" no cálculo de duas matrizes de dimensões 3×3.

n	**Compara Coluna**	**Elementos a Comparar**		**Resíduo**	**Situação**
X	0	X	X	X	Nada será subtraído
0	1	$r^{T}AB_{(0,0)}$	$r^{T}C_{(0,0)}$	$r^{T}AB_{(0,0)} - r^{T}C_{(0,0)}$	Testa Coluna 0
1	1	$r^{T}AB_{(0,1)}$	$r^{T}C_{(0,1)}$	$r^{T}AB_{(0,1)} - r^{T}C_{(0,1)}$	Testa Coluna 1
2	1	$r^{T}AB_{(0,2)}$	$r^{T}C_{(0,2)}$	$r^{T}AB_{(0,2)} - r^{T}C_{(0,2)}$	Testa Coluna 2

Tabela 4.5: Resumo do funcionamento do módulo Comparador/Subtrator de colunas.

4.2.5 Geração do endereço do elemento corrompido

No final de uma determinada operação de multiplicação de matrizes, o bloco *Gerador de Endereços*, pertencente à unidade de detecção de erros do RA[3] tem como objetivo entregar calculado o endereço de memória do elemento corrompido da matriz C em sua saída, para que então seja possível a busca deste elemento na memória para que seja repassado à unidade de correção de erros. Este cálculo é realizado à medida que os elementos que compõe os vetores codificados de entrada (ABr e r^{T}AB) são comparados com vetores codificados de saída (Cr e r^{T}C) e só será utilizado no caso em que um erro for detectado. Quando uma comparação entre elementos de mesmo índice do vetor ABr com o vetor Cr resultar em igualdade, é um indício de que a linha da matriz resultado com mesmo índice encontra-se sem elementos corrompidos. Neste mesmo ciclo, o bloco comparador de linhas envia nível lógico baixo através do sinal *erro_de_linha* ao bloco gerador de endereços, fazendo com que o mesmo atualize seu endereço de forma a apontar para o endereço de memória da próxima linha da matriz C. Esta atualização no endereço é feita através da soma do endereço da linha que acabou de ser testada com o número de colunas na matriz C, e é denominado de *"Endereço_Linha"*. Na Figura 4.14, é mostrado um digrama de uma memória de dados contendo os elementos da matriz C, resultantes de uma multiplicação de duas matrizes A e B de dimensões 3×3. Nesta figura, foi considerado o endereço base da matriz C igual a 0x1000 e que os elementos da matriz C são de 32 bits, além de um suposto elemento corrompido $c_{(1,2)}$ com o objetivo de facilitar o entendimento.

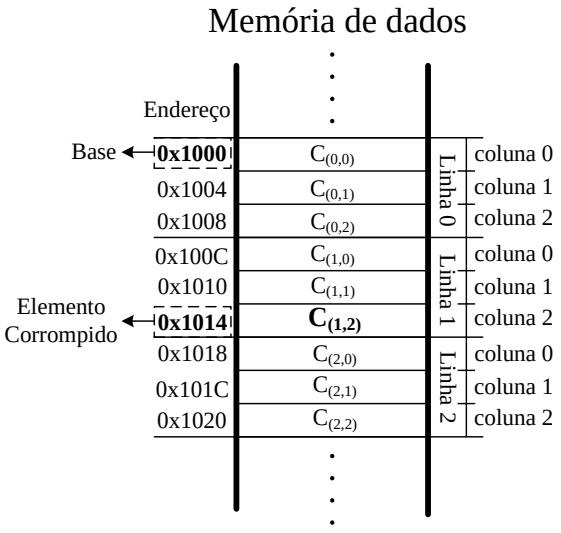


Figura 4.14: Diagrama de uma memória de dados armazenando a matriz C de dimensões 3×3.

Conforme podemos observar na Tabela 4.6, quando elementos dos vetores ABr e Cr de mesmo índice forem divergentes, significa que o elemento corrompido pertence à linha testada, não havendo a necessidade da atualização do *Endereço_Linha* por intermédio da soma com o número de colunas. No exemplo da Figura 4.14, a divergência ocorreria apenas na comparação entre os elementos AB$r_{(1,0)}$ e C$r_{(1,0)}$ indicando que o elemento corrompido está localizado na 2ª linha da matriz C.

Comparação		**Resultado Comparação**	***Endereço_Linha***
AB$r_{(0,0)}$	C$r_{(0,0)}$	Diferente	0x1000 (Erro está na 1ª linha)
		Igual	0x100C (Atualiza Endereço)

$ABr_{(1,0)}$	$Cr_{(1,0)}$	Diferente	0x100C (Erro está na 2ª linha)
		Igual	0x1018 (Atualiza Endereço)
$ABr_{(2,0)}$	$Cr_{(2,0)}$	Diferente	0x1018 (Erro está na 3ª linha)
		Igual	Matriz Sem Erros!

Tabela 4.6: Atualização do *Endereço_Linha* do elemento corrompido na memória.

De posse do endereço do primeiro elemento da linha (*Endereço_Linha*= 0x100C) que contem o elemento corrompido, resta ainda apontar o endereço exato do elemento corrompido dentro dessa linha, ou seja, descobrir a coluna do elemento. Isto será possível através das comparações feitas entre os elementos de mesmo índice dos vetores $r^{T}AB$ e $r^{T}C$, ou seja, em caso de igualdade o endereço deverá ser atualizado, só que desta vez através da soma com o número de *bytes* da palavra de dados. Esta segunda etapa gera o exato endereço do elemento corrompido dentro da matriz C, como podemos observar na Tabela 4.7.

Comparação		**Resultado Comparação**	**Endereço_Elemento_Corrompido**
$r^{T}AB_{(0,0)}$	$r^{T}C_{(0,0)}$	Diferente	0x100C (Erro na 1ª coluna)
		Igual	0x1010 (Atualiza Endereço)
$r^{T}AB_{(0,1)}$	$r^{T}C_{(0,1)}$	Diferente	0x1010 (Erro na 2ª coluna)
		Igual	0x1014 (Atualiza Endereço)
$r^{T}AB_{(0,2)}$	$r^{T}C_{(0,2)}$	Diferente	0x1014 (Erro na 3ª coluna)
		Igual	Não existem erros na coluna

Tabela 4.7: Atualização do endereço que aponta a coluna do elemento corrompido.

Como mencionado anteriormente, a geração deste endereço é feita através de alguns cálculos que são feitos durante a execução da técnica de *Freivads Modificada*. Estes cálculos dependem de três parâmetros diferentes, o primeiro é configurado pelo usuário em tempo de síntese, denominado LARGURA_DOS_DADOS, referente ao número de bits dos dados das matrizes em operação, o segundo é o número de colunas da matriz resultante C, que pode ser deduzido de acordo com as dimensões das matrizes em operação, conforme propriedades matemáticas da operação de multiplicação de matrizes, e finalmente, o endereço base onde se inicia a escrita dos elementos da matriz C na memória de dados.

Na Figura 4.15, podemos observar o circuito que realiza os cálculos do endereço do elemento corrompido, que será explicado a seguir. Para facilitar o entendimento, será o utilizado o mesmo exemplo mostrado na Figura 4.14, onde a matriz C resultante possui 3 linhas e 3 colunas e o elemento corrompido encontra-se na intersecção da linha 1 com a coluna 2 (elemento $C_{(1,2)}$). O endereço base, utilizado como uma das entradas do circuito é o 0x1000, os elementos das matrizes são de 32 bits e o número de colunas da matriz C é igual a 3. Neste mesmo exemplo, a comparação entre $ABr_{(0,0)}$ e $Cr_{(0,0)}$ resulta em igualdade uma vez que o elemento corrompido $C_{(1,2)}$ não pertence a primeira linha. Neste instante o endereço base deve ser somado com um valor que faça com que o endereço aponte para o primeiro elemento da próxima linha (Linha 1), que no caso do exemplo, é o endereço 0x100C. Para tal, o endereço deve ser somado com o número de bytes de todos os elementos da primeira linha. Como cada linha apresenta 3 colunas, e cada elemento é composto por 4 bytes (LARGURA_DOS_DADOS=32), o valor somado ao endereço base é 3 (número de colunas) × 4 (número de bytes por elemento) que resulta em 12 bytes, ou na base hexadecimal 0x000C. Afim de evitar a utilização de mais um multiplicador em *hardware* apenas para multiplicar o número de colunas da matriz C pelo número de bytes de cada elemento, este que é um valor fixo definido em

tempo de síntese, foi utilizado um circuito de deslocamentos de bit a esquerda (SHIFT) fazendo o mesmo papel da multiplicação. O número de bits que devem ser deslocados é definido em tempo de síntese e pode ser calculado através da equação abaixo.

$$SHIFT = log_2\ Palavra\ em\ bytes$$

Considerando o mesmo exemplo, onde a matriz C possui 3 colunas e os dados possuem 4 bytes (32 bits), o número de bits que devem ser deslocados é igual a 2, fazendo com que o número 3 após dois deslocamento se torne o número 12. Após a descoberta que o elemento corrompido pertence a segunda linha, o calculo do endereço passa para a segunda fase, onde o objetivo é a descoberta da coluna do elemento corrompido.

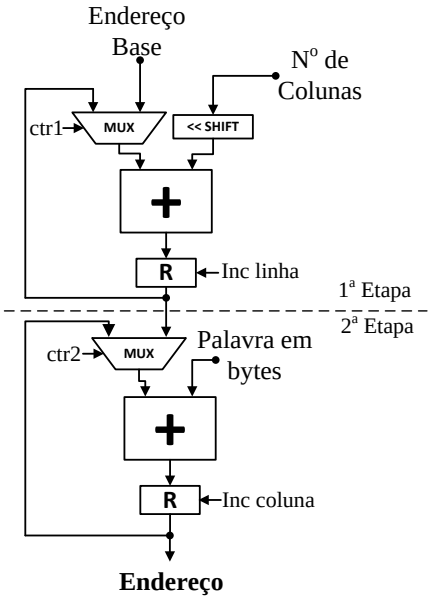


Figura 4.15: Exemplo do cálculo do endereço de leitura do elemento corrompido.

A segunda etapa de geração do elemento corrompido é executada através da soma do endereço da linha do elemento corrompido com o número de bytes de cada palavra. Aproveitando o exemplo anterior, podemos observar que os elementos $r^{T}AB_{(0,0)}$ e $r^{T}C_{(0,0)}$ são iguais, logo o elemento corrompido não está na primeira coluna (coluna 0), sendo assim o endereço da linha 0x100C será somado com 4 (número de bytes da palavra de dados), gerando o endereço 0x1010 que aponta diretamente ao elemento $C_{(1,1)}$, que será o próximo elemento a ser testado.

4.3 Unidade de Correção de Erros

Por fim, quando um erro é detectado pela arquitetura durante a execução do algoritmo de multiplicação de matrizes, inicia-se o processo de correção do elemento corrompido dentro da matriz C. Este processo começa com a busca do elemento corrompido da memória, através do endereço calculado durante a multiplicação de matrizes. De posse do elemento corrompido, basta ser feita a subtração pelo resíduo também já calculado durante a execução do algoritmo. Neste instante o sinal do multiplexador (ERRO!) é setado pelo controle para selecionar o resultado da saída do circuito subtrator, fazendo com que o valor corrigido seja sobrescrito na mesmo posição de memória de onde foi buscado o elemento corrompido. Após isto, a arquitetura está

pronta para uma nova operação de multiplicação de matrizes. Na Figura 4.16, é mostrado um diagrama em blocos com o circuito responsável pela correção do elemento corrompido.

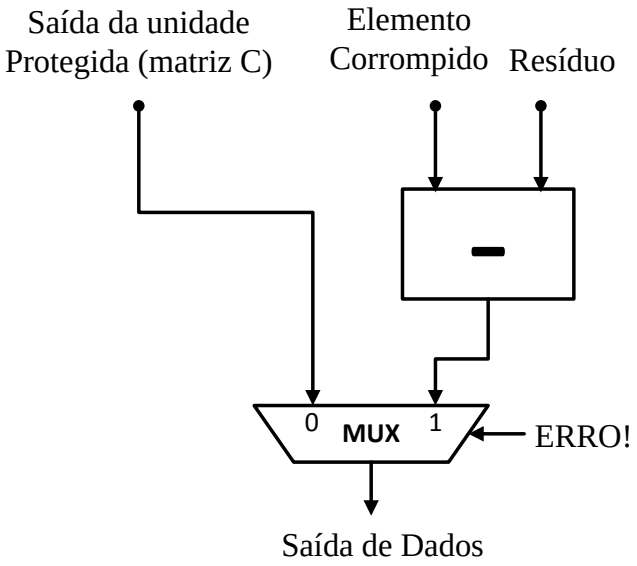


Figura 4.16: Diagrama do circuito responsável por corrigir o elemento corrompido.

4.4 Modelo Analítico de Desempenho do RA^3

Após a apresentação e descrição de cada um dos módulos que compõe a arquitetura RA^3, será mostrado um modelo analítico que permite calcular o número de ciclos de *clock* necessários para realizar uma determinada multiplicação de matrizes. Para utilizar o modelo, é necessário conhecer as dimensões das matrizes envolvidas na multiplicação, assim como a configuração (Número de Multiplicadores) previamente selecionada pelo usuário para a síntese do RA^3, entre outros, conforme mostrado na equação 3.

$$\#Ciclos = \left(\frac{N_{MA} \times M_{MA} ou N_{MB} \times M_{MB}}{K/2}\right)^3 + 2 \times \left(\frac{M_{MA} ou N_{MB}}{K/2}\right) + p \qquad (3)$$

No modelo, o parâmetro N_{MA} equivale ao número de linhas da matriz A enquanto que M_{MA} e M_{MB} equivalem ao número de colunas das matrizes A e B respectivamente. A constante *K* corresponde ao número total (Multiplicação + Freivalds) de multiplicadores sintetizados na arquitetura RA^3, e têm seu valor dividido por 2 pois apenas os multiplicadores da unidade protegida influenciam no número de ciclos. Por fim, o parâmetro *p* equivale à profundidade do *pipeline* utilizado na unidade de multiplicação de matrizes, ou seja, quanto mais multiplicadores forem colocados em paralelo, mais ciclos de *clock* serão necessários para a geração do primeiro valor de saída da matriz C. Os valores de *p* são definidos conforme mostrado abaixo:

$$p = \begin{cases} 2, & se\ K < 16 \\ 3, & se\ 16 \leq K < 128 \\ 4, & se\ K \geq 128 \end{cases}$$

Para a demonstração do modelo, será considerada a operação C=A×B, onde a matriz A possui as dimensões N_{MA}×M_{MA} e a matriz B possui dimensões N_{MB}×M_{MB}. No exemplo da figura abaixo, podemos observar que as dimensões da matriz A são N_{MA}=2 e M_{MA}=2 enquanto que as dimensões da matriz B são N_{MB}=2 e M_{MB}=2.

$$C = A \times B = \begin{bmatrix} a_{(0,0)} & a_{(0,1)} \\ a_{(1,0)} & a_{(1,1)} \end{bmatrix} \times \begin{bmatrix} b_{(0,0)} & b_{(0,1)} \\ b_{(1,0)} & b_{(1,1)} \end{bmatrix}$$

Considerando que o usuário tenha escolhido 4 multiplicadores (2 para o módulo Multiplicador Matriz-Matriz e 2 para o módulo Multiplicador Matriz-Vetor) temos o seguinte cálculo:

$$\#Ciclos = \left(\frac{2 \times 2 \times 2}{2}\right) + 2 \times \left(\frac{2}{2}\right) + 2 = 9\ ciclos$$

No capítulo 7, será mostrada uma figura que permite uma comparação em termos de desempenho entre a simples multiplicação de matrizes (sem proteção) e a arquitetura RA^3, variando as dimensões das matrizes e o número de multiplicadores sintetizados na estrutura. Esta figura permite avaliar o custo real da implementação da técnica junto com a operação de multiplicação de matrizes.

4.5 Tornando adaptativos os módulos Multiplicador Matriz-Matriz e Matriz-Vetor

A adaptabilidade proposta neste trabalho consiste na aplicação da técnica de *Power Gating* de forma que seja possível ligar ou desligar unidades de hardware, como multiplicadores, memórias, somadores, etc. A aplicação desta técnica é necessária para que seja possível a exploração adaptativa de recursos baseadas nos critérios anteriormente apresentados, como largura máxima da banda de memória e *deadlines* em aplicações de tempo real. Para tal, algumas modificações foram feitas no módulo Multiplicador Matriz-Matriz e consequentemente no módulo Multiplicador Matriz-Vetor previamente apresentados na Figura 4.3. O diagrama do circuito contendo as modificações é mostrado na Figura 4.17, e apresenta "interruptores" (ON/OFF) que representam, na realidade, o controle dos *sleep transistors* necessários para a aplicação da técnica de *Power Gating*. Outra consideração é sobre a colocação de memórias do tipo *scratchpad* dedicadas em cada uma das entradas dos multiplicadores da estrutura, que são abastecidas pela memória principal do tipo LPDDR2 através do barramento de DMA (do inglês Direct Memory Access). A regularidade com que estes dados são acessados pela arquitetura, favorece a utilização deste tipo de memória ao invés do uso das tradicionais memórias do tipo cache que apresentam como inconveniente um alto consumo de potência relacionado ao controle lógico.

O exemplo da Figura 4.17, conta também com uma configuração onde foram colocados 4 multiplicadores em paralelo para o algoritmo de multiplicação de matrizes e 4 multiplicadores para a técnica de Freivalds Modificada. Os sinais de controle (S0, S1, S2 e S3) foram projetados para desligar ou ligar unidades de acordo com o número de multiplicadores desejados pelo projetista.

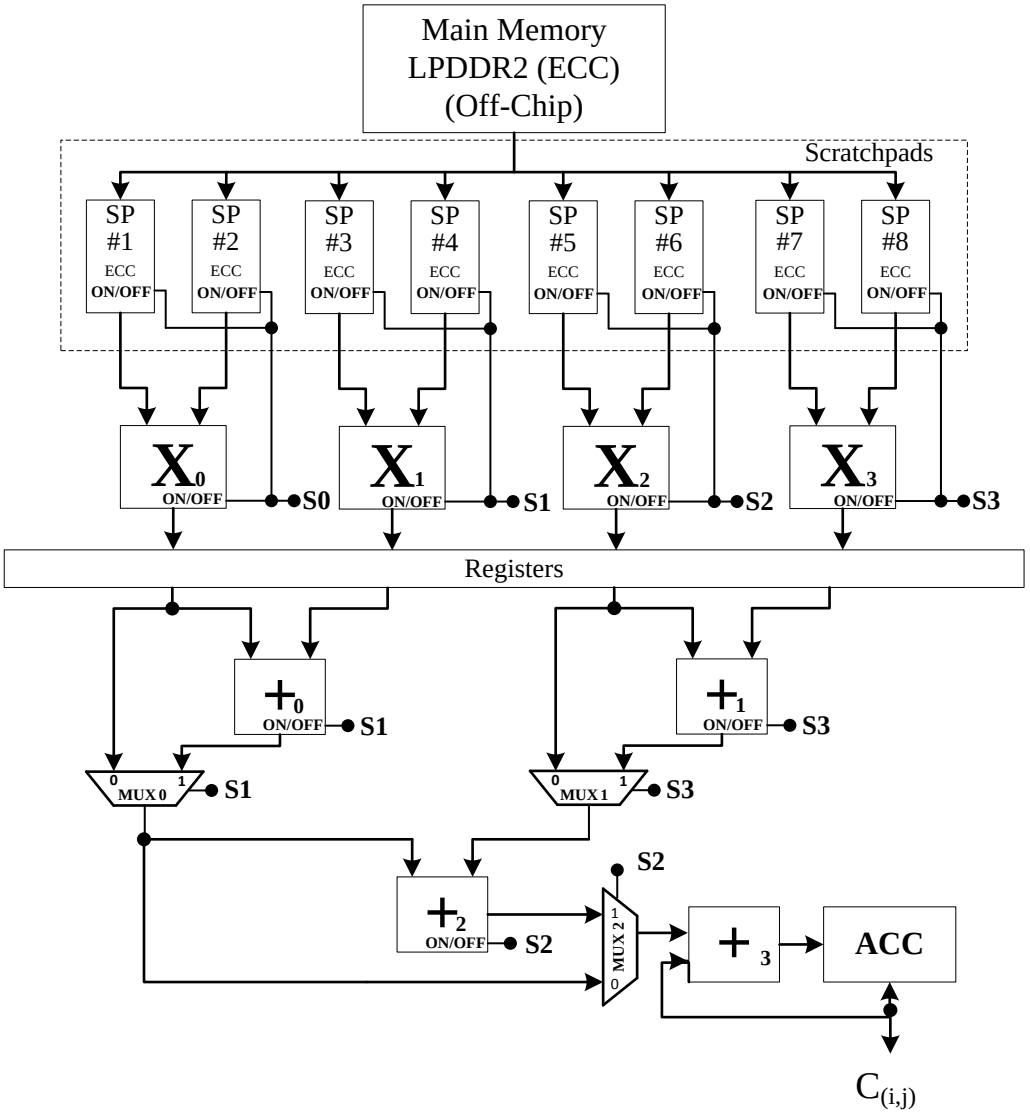


Figura 4.17: Módulo Multiplicador Matriz-Matriz Modificado.

Na Tabela 4.8 é possível verificar que no caso onde apenas 2 dos 4 multiplicadores devem permanecer ligados para execução de uma determinada tarefa, os sinais S0 e S1 devem estar em nível lógico alto enquanto que os sinais S2 e S3 devem estar em nível baixo. Os mesmos sinais que ligam e desligam unidades também controlam o fluxo correto dos dados dentro da estrutura através dos multiplexadores. Vale ressaltar que nível lógico alto liga unidades enquanto que unidades que recebem nível lógico baixo estão sendo desligadas.

Número de Multiplicadores Desejados	S0	S1	S2	S3
1 (X0)	1	0	0	0
2 (X0 e X1)	1	1	0	0
3 (X0, X1 e X2)	1	1	1	0
4 (X0, X1, X2 e X3)	1	1	1	1

Tabela 4.8: Exemplo do controle Liga/Desliga das unidades de hardware.

5 FERRAMENTA DE SIMULAÇÃO DE FALHAS

Todo projeto focado em técnicas de tolerância a falhas, nos mais diversos níveis de abstração, necessita de uma etapa de teste que permita avaliar o desempenho das mesmas. Em casos onde o circuito a ser testado foi fabricado, estes testes podem ser realizados através de medição direta, ou seja, enviando o circuito a grandes altitudes para que os efeitos sejam observados num ambiente real, através de testes a laser ou com testes de radiação em aceleradores de partículas. Esta última opção, consiste na exposição do dispositivo a feixes compostos por partículas conhecidas, geralmente prótons, nêutrons e partículas *alpha*, e a resposta deste dispositivo é observada em tempo real. Este feixe é gerado através de um acelerador de partículas e tem como objetivo simular ambientes com características espaciais (nêutrons atmosféricos) e terrestres (partículas *alpha* e nêutrons). A regulamentação JESD89A (JEDEC STANDARD, 2006) define os requisitos e procedimentos que regulamentam os testes terrestres de SER para circuitos integrados. Embora sejam efetivos, estes métodos apresentam inconvenientes como alto custo e a necessidade por instalações e equipamentos adequados.

Por outro lado, existem métodos alternativos para injeção de falhas em circuitos que apresentam um baixo custo de implementação e podem ser aplicados ainda na fase de projeto. A técnica de injeção de falhas por emulação em *hardware*, por exemplo, possibilita injetar muitas falhas num pequeno intervalo de tempo. Entretanto, o circuito a ser testado e o injetor devem ser sintetizados numa plataforma FPGA que nem sempre está disponível ou é de interesse dos projetistas (GRINSCHGL, KRIEG, *et al.*, 2011).

Projetistas que buscam por baixo custo de implementação e não desejam sintetizar suas descrições arquiteturais em FPGAs, contam ainda com a opção da injeção de falhas por simulação. A simulação destas falhas pode ser feita em qualquer ferramenta de simulação de códigos descritos em VHDL, que no caso deste trabalho, foi o ModelSim desenvolvido pela empresa *Mentor Graphics*®. Esta ferramenta possui um console que permite executar comandos e scripts na linguagem TCL (do inglês *Tool Command Language*). Dentro dos scripts, são colocadas sequências de comandos TCL que podem ser usados tanto para compilação e simulação dos módulos descritos em VHDL da arquitetura RA^3 quanto para simular os efeitos das falhas. A simulação das falhas é feita no nível RTL (do inglês *Register Transfer Level*) que mesmo sendo executada antes da síntese lógica, apresenta boas estimativas da taxa de cobertura de falhas.

5.1 Funcionamento

A ferramenta de injeção de falhas, é na prática, um programa capaz de gerar os scripts TCL conforme parâmetros que podem ser escolhidos pelo usuário e outros baseados nos dados do circuito em teste. No presente trabalho, o injetor de falhas foi baseado na ferramenta desenvolvida em (AZAMBUJA, 2010), entretanto apresenta diferenças em termos da linguagem de desenvolvimento e metodologia de implementação. Enquanto que na versão original a ferramenta de injeção foi feita em Java, na versão utilizada neste trabalho foi totalmente desenvolvida no Matlab®, por oferecer uma ampla gama de recursos estatísticos e gráficos que serão posteriormente mostrados.

Para a simulação de falhas, a linguagem TCL conta um comando chamado *force*, que é capaz de modificar o valor de qualquer bit (sinal) dentro dos nós que compõe o circuito em teste. A ideia é utilizar este comando para que sinais tenham seus valores momentaneamente modificados, simulando assim os efeitos de SETs. Além disso, alguns parâmetros também devem ser definidos, como o número de falhas, o instante de tempo em que estas falhas serão injetadas, quais os sinais que serão afetados durante a campanha, o tempo de duração das falhas, etc.

Na Figura 5.1, é mostrado um diagrama em blocos que demonstra o fluxo de funcionamento da ferramenta de injeção proposta neste trabalho e a interação entre circuito e injetor de falhas. A figura permite analisar também a sequência de passos que um projetista de circuitos que contém apenas a descrição de seu circuito em VHDL e deseja injetar falhas deve seguir para utilizar a ferramenta.

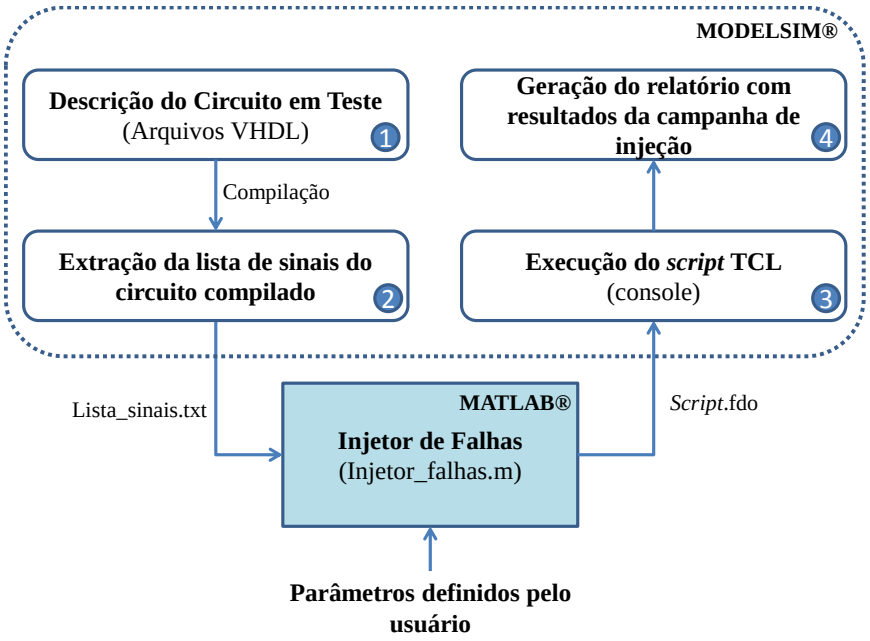


Figura 5.1: Interação entre a descrição do circuito e o injetor de falhas.

Como pode ser visto, o processo se inicia através da compilação dos arquivos com a descrição do circuito em VHDL dentro do Modelsim®. Após a compilação bem sucedida, é possível acessar a lista contendo os sinais (nós) do circuito através da janela *Objects* ainda no Modelsim®. Após definida a lista completa dos nós do circuito, é necessário configurar o injetor para a campanha de injeção de falhas. Esta configuração

é realizada através da edição de alguns parâmetros sobre o processo de injeção de falhas no arquivo *Injetor_Falhas.m*. Alguns dos principais parâmetros são:

- **Número de falhas a injetar:** Serve para definir o número de falhas a injetar numa campanha, este número geralmente está na casa de milhares de falhas. Será injetada uma falha em cada simulação.
- **Tempo de Execução:** Neste parâmetro o usuário deve informar o tempo total que o circuito em teste leva para realizar todo o processamento. Este parâmetro é o valor limitador na geração da lista com os tempos de falha. Tempos de injeção são escolhidos de maneira pseudoaleatória, entretanto não podem ultrapassar este parâmetro.
- **Frequência de *clock* do circuito:** Definido para a simulação do circuito. Serve de base para a geração do tempo de duração das falhas.
- **Tempo de duração das falhas:** Aqui é definido o tempo em que os sinais contidos na lista de sinais do circuito terão seu valor modificado (simulação da falha). Geralmente este tempo é igual ao período de *clock* do circuito definido pelo usuário.
- **Lista de Tempos de Falha:** São valores de tempo gerados de forma pseudoaleatória, apresentando como restrição não ultrapassar o parâmetro **Tempo de Execução**.
- **Lista de pontos de falha:** É uma lista contendo todos os sinais (nós) do circuito em teste. Esta lista é retirada após a compilação dos arquivos VHDL do circuito.
- **Localização da Memória:** Circuitos que necessitam utilizar memórias no seu processamento, o caminho do arquivo contendo estas informações deve ser informado.
- **Nome do Relatório com os resultados:** Neste parâmetro é possível definir o nome e a pasta em que o relatório (resultados.txt) deve ser direcionado.

5.2 Geração do conjunto de Falhas

Uma das diferenças da versão apresentada aqui para a versão original do injetor de falhas está na geração da lista contendo os tempos de falhas. Como discutido anteriormente, esta geração consiste na geração de números pseudoaleatórios, referentes ao tempo da injeção da falha, que não ultrapassem o tempo total de execução do processamento. Na Figura 5.2, é mostrado uma parte do *script* TCL gerado pelo injetor, onde o usuário definiu que seriam injetadas 10 falhas com o tempo de execução igual a 5300ns. Como pode ser observado na figura, o injetor gerou 10 valores de tempo, que serão os tempos exatos para a injeção de cada uma das 10 falhas.

```
set tempo_execucao 5300
set Numero_Falhas  10

array Lista_Tempos {
1 2537.0
2 4658.0
3 2727.0
4 3138.0
5 3619.0
6 1485.0
7 42.0
8 884.0
9 980.0
10 3168.0
}
```

Figura 5.2: Geração dos vetores de tempo de injeção

Na ferramenta Matlab® existem várias funções para geração de números pseudoaleatórios, como por exemplo, *rand*, *randn*, *randi* que podem ser escolhidas pelo usuário de acordo com a distribuição desejada. Em campanhas com elevado número falhas a serem injetadas é importante saber como estes tempos de injeção estão distribuídos dentro do tempo total de execução. Após a configuração do injetor, no momento da compilação no Matlab®, algumas informações em forma de gráficos são geradas e mostradas ao usuário, dentre elas está um histograma mostrando como os tempos de falhas foram distribuídos dentro do tempo total de execução, conforme visto na Figura 5.3.

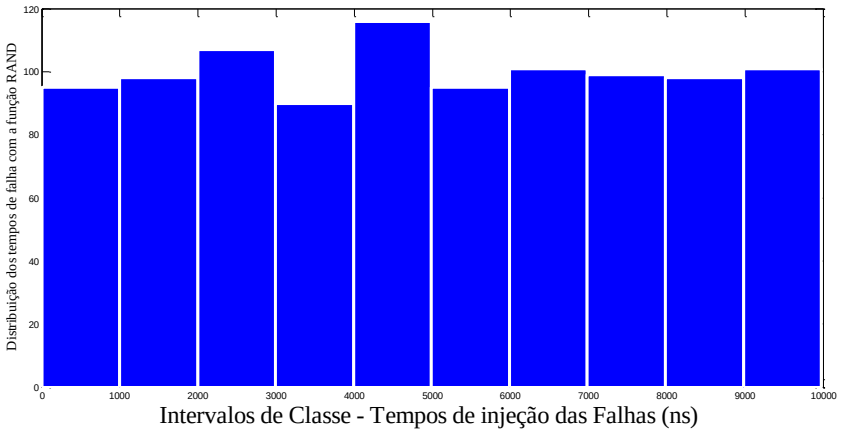


Figura 5.3: Histograma com a distribuição dos tempos de falha.

No exemplo da Figura 5.3, o usuário configurou o injetor para 1000 falhas e o tempo de execução total é de 10000ns. Como a função *rand* possui uma distribuição uniforme pode ser observado que dentro destas 1000 falhas injetadas, uma quantidade muito parecida de falhas será injetada ao longo do tempo de execução. Existem ainda outras funções com distribuições diferentes que são capazes de concentrar as falhas para que ocorram mais no início do processamento ou mais ao final, de acordo com a exigência do usuário.

Existe ainda outro parâmetro que diferencia o injetor desenvolvido neste trabalho do projeto do injetor original que diz respeito à seleção dos sinais onde serão injetadas as falhas. Isto significa que o injetor de falhas aqui apresentado leva em consideração a área ocupada pelas unidades de *hardware* que compõe um circuito sob teste e fazer com que este percentual influencie na seleção dos sinais. Para tal, o usuário deve dividir o

circuito em grupos, dentro da arquitetura ou do circuito em teste e informar ao injetor a área percentual que estas unidades ocupam dentro do circuito. Este parâmetro é levado em consideração na hora da seleção da seguinte forma, grupos que ocupam áreas maiores dentro do circuito terão probabilidades (pesos) maiores de terem seus sinais (nós) escolhidos para injeção. No exemplo da Figura 5.4, é dado um exemplo de um circuito que será testado, cuja área total foi dividida em grupos simbólicos A, B, C e D com suas respectivas áreas percentuais. Seguindo a distribuição de áreas da figura e considerando que 1000 falhas devem ser injetadas, pode-se concluir que em torno de 500 falhas deve ser injetada sobre o grupo A, uma vez que sua área corresponde a 50% da área total do circuito. As outras 500 falhas estarão distribuídas entre os grupos B, C e D conforme a distribuição do percentual de áreas mostrada na Figura 5.4.

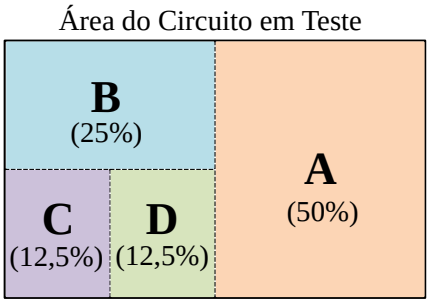


Figura 5.4: Exemplo da criação de classes de unidades

5.3 Processo de Simulação de Falhas

De posse da lista de sinais e tempos e com os parâmetros do injetor definidos e configurados, o próximo passo é voltar ao Modelsim® para a execução do script TCL através do console. Na Figura 5.5, é mostrado um diagrama em blocos para demonstrar o funcionamento da ferramenta de injeção de falhas. A primeira vez que o circuito é simulado, nenhuma falha é injetada, dando origem ao resultado denominado *Golden.* Depois disso, o número de simulações será igual ao número de falhas que o usuário definiu injetar, onde apenas uma única falha será injetada em cada simulação. Isto ocorre devido ao modelo de falha simples utilizado neste trabalho, que considera que apenas uma única falha poderá vir a ocorrer em cada execução.

Toda a simulação finalizada terá seu valor de saída comparado ao resultado *Golden*. Em casos de divergência entre estes resultados, é possível verificar que a falha injetada foi capaz de gerar um erro no funcionamento do circuito.

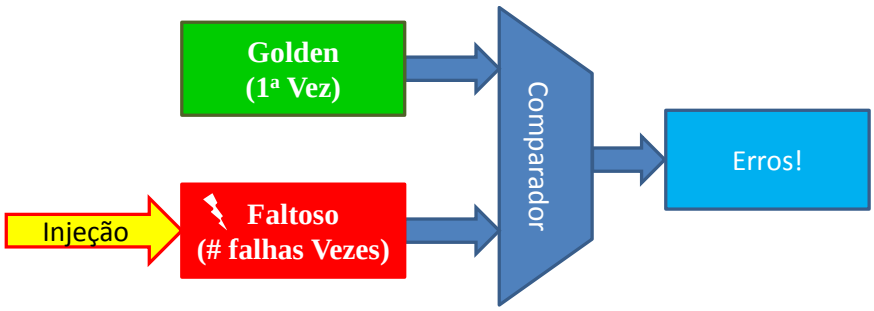


Figura 5.5: Demonstração da estrutura do processo de injeção de falhas da ferramenta.

Na Figura 5.6, foi criado um exemplo para demonstração do processo de injeção de falhas baseados na consulta dos vetores de tempo de injeção e dos sinais do circuito. O tempo total de execução da tarefa em questão é igual a 1000ns. No exemplo, foi

considerado que apenas duas falhas serão injetadas, visando facilitar a explicação. Após a primeira simulação, e geração do resultado *Golden*, as simulações seguintes serão baseadas nos dados consultados dos vetores de tempo e de sinais mostrados na figura. Na primeira falha, a simulação ocorrerá normalmente até o tempo 200ns, onde a mesma é parada e o sinal (bit) na posição 1 indicado no vetor de sinais será modificado (invertido) pelo tempo definido pelo usuário. Decorrido o tempo da falha, o sinal (bit) é trazido ao estado anterior e a simulação prossegue até o final normalmente. Na próxima simulação, o procedimento será o mesmo, com a diferença que agora o próximo valor dos vetores é que serão utilizados.

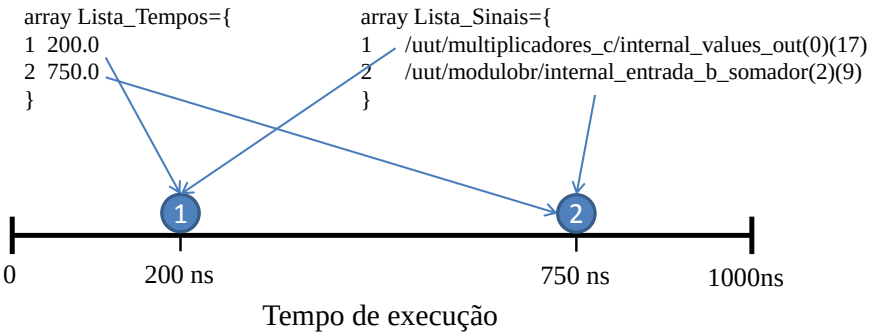


Figura 5.6: Demonstração da utilização dos vetores de tempo e sinais no processo de injeção de falhas.

5.4 Relatórios dos resultados

Por fim, os resultados provenientes deste processo de injeção de falhas devem de alguma maneira ser registrados, de forma a permitir que o usuário verifique o desempenho do circuito em teste. A linguagem TCL, permite a leitura e escrita em arquivos texto dentro dos scripts, o que permite que memórias sejam simuladas e relatórios para coleta de resultados sejam gerados. Na Figura 5.7, é mostrado o exemplo de um relatório gerado contendo os principais dados capazes de permitir uma avaliação do desempenho do injetor e principalmente da arquitetura RA^3.

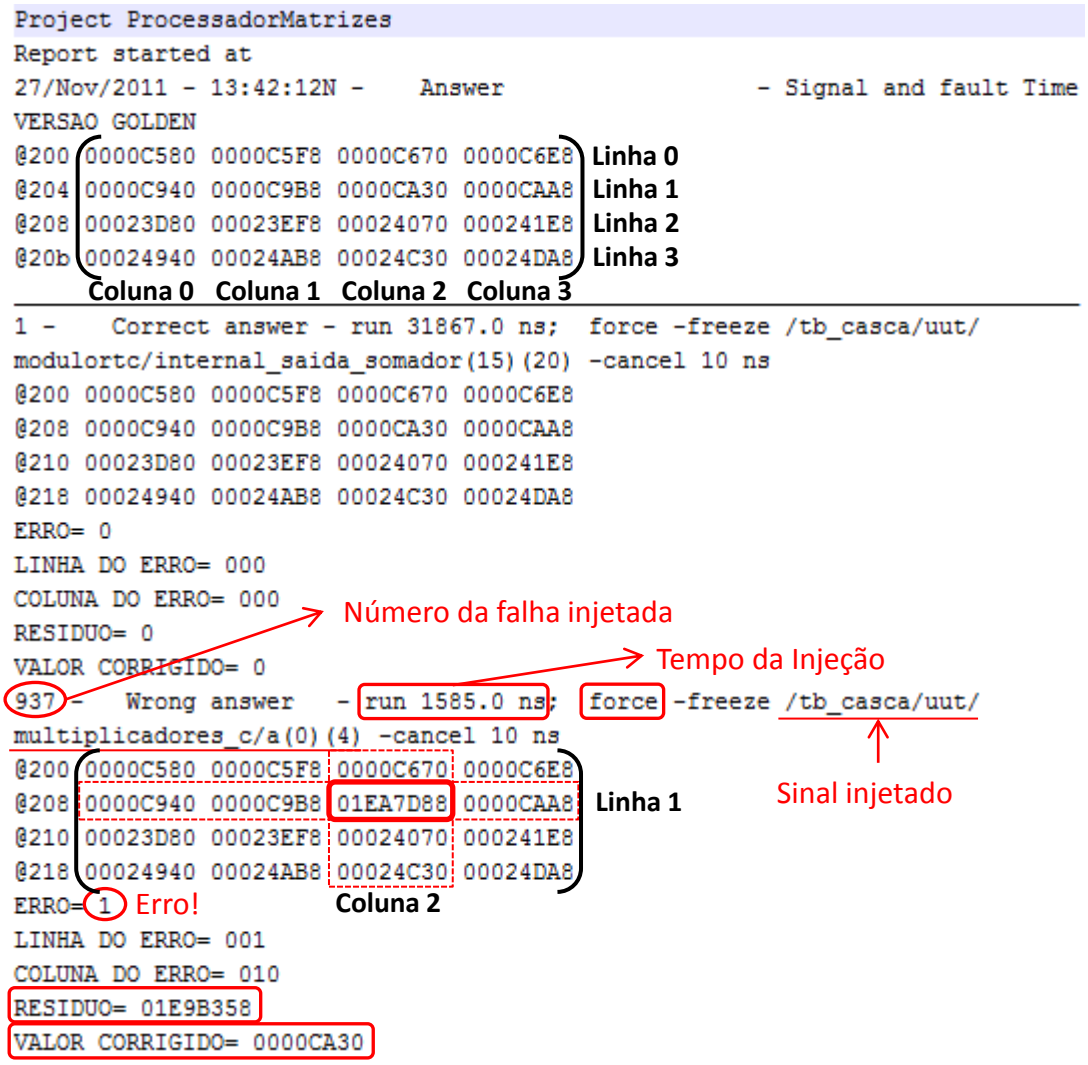


Figura 5.7: Exemplo de um relatório gerado após a simulação de uma campanha de injeção de falhas.

Podemos dividir a Figura 5.7 em três blocos principais, versão *Golden* (acima), uma simulação onde a falha injetada não causou nenhum erro (centro), e finalmente uma simulação onde a falha foi capaz de gerar um erro (abaixo). A operação que está sendo protegida neste exemplo é uma multiplicação de duas matrizes de dimensões 4×4, e a matriz resultado contendo os elementos corretos (na base hexadecimal) tem sua escrita organizada no formato de matricial.

Como discutido anteriormente, na primeira simulação nenhuma falha é injetada, dando origem ao resultado *Golden* mostrado na parte de cima da figura. A partir deste momento é que as falhas são injetadas, uma em cada simulação. Ao final de uma simulação é feita uma comparação entre a matriz resultado gerada na atual simulação com a matriz resultado da versão *Golden*. Se os resultados forem iguais, é impresso no ralatório, ao lado do número da falha, a mensagem *Correct answer* (Resposta Correta) e no caso onde a matriz gerada possua alguma divergência com o resultado *Golden* a mensagem *Wrong answer* (Resposta Errada) é impressa. Abaixo da matriz gerada durante a simulação são impressos também alguns dados da arquitetura RA^3 para que seja possível constatar a eficácia da estrutura para detectar e corrigir erros. Foi colocado na arquitetura um pino denominado ERRO, que é colocado em zero quando nenhum

erro é detectado pela arquitetura e colocado em um quando um erro foi detectado. Foram colocados também alguns pinos no RA^3 para indicar a linha e coluna do elemento corrompido que são impressos no relatório com o nome de LINHA DO ERRO e COLUNA DO ERRO. Para finalizar, foram colocados ainda em pinos os valores do resíduo calculado e do valor do elemento corrompido corrigido que será novamente escrito na memória, que são impressos com os nomes RESIDUO e VALOR CORRIGIDO respectivamente.

Na primeira falha injetada o resultado simulado é exatamente igual ao resultado *Golden* e a mensagem *Correct Answer* é impressa. A arquitetura RA^3 por sua vez confirma que nenhuma falha foi detectada colocando o sinal ERRO=0, além dos valores RESIDUO e VALOR CORRIGIDO iguais a zero. Logo abaixo é mostrada a falha de número 937, selecionada dentro do relatório, que acusa que o resultado da simulação diverge do resultado Golden. Realmente, comparando as duas matrizes é possível verificar que o elemento 0x01EA7D88 na intersecção da linha 1 com a coluna 2 difere do elemento 0x0000CA30 no resultado *Golden* na mesma posição. Isto é um indício que a falha injetada foi capaz de corromper o cálculo de um dos elementos da matriz C, mais especificamente o elemento $C_{(1,2)}$. Ao mesmo tempo é possível também observar que a arquitetura RA^3 foi capaz de detectar o erro (ERRO=1) além de indicar precisamente a linha (LINHA DO ERRO= 001) e a coluna (COLUNA DO ERRO=010) do elemento corrompido dentro da matriz C. Vale observar que os números da linha e da coluna são impressos na base binária.

Uma vez o erro tenha sido detectado e o elemento corrompido tenha sido identificado dentro da matriz C, torna-se necessário iniciar o processo de correção do mesmo. Para tal, é necessário executar a subtração do elemento corrompido $C^*_{(1,2)}$ pelo resíduo previamente calculado pela arquitetura RA^3 e impresso também no relatório. O cálculo mostrado abaixo demonstra a subtração necessária para restauração do elemento corrompido e prova e eficácia da estrutura de detecção e correção de erros da arquitetura RA^3 .

$$C_{(1,2)} = C^*_{(1,2)} - Resíduo = 0X01EA7D88 - 0X01E9B358 = \mathbf{0X0000CA30}$$

O valor corrigido é então sobrescrito na mesma posição de memória de onde foi trazido o elemento corrompido $C^*_{(1,2)}$ fazendo com que a matriz C esteja livre de erros.

6 RESTRIÇÕES FÍSICAS E DE TEMPO REAL E APLICAÇÕES AFINS

Neste capítulo será apresentado o espaço de projeto explorado, considerando as restrições encontradas no estado da arte de sistemas embarcados. As considerações aqui descritas são feitas levando em conta que os sistemas embarcados frequentemente enfrentam rigorosas restrições de tempo real e de confiabilidade. Serão apresentados também neste capítulo, os dois estudos de caso escolhidos para este trabalho. Eles foram cuidadosamente selecionados por apresentar todas as características de interesse para a exploração da arquitetura RA^3.

Um dos maiores desafios enfrentados em sistemas computacionais é o aumento limitado na largura da banda de memória, quando comparado com a evolução da parte lógica dos circuitos. Este *gap* entre o tempo necessário para realizar uma determinada computação e o tempo necessário para ler um dado da entrada e escrever uma saída na memória principal geralmente é o limitador do tempo total de computação.

A velocidade da computação da parte lógica aumenta à medida que os transistores tornam-se mais rápidos e menores, permitindo frequências de operação mais altas e uma exploração de paralelismo maior com a mesma área. Tanto a frequência de operação quanto a exploração de paralelismo podem ser atenuadas através de técnicas como DVFS e PG para limitar o *throughput* dos dados e manter o desempenho compatível com a largura da banda de memória disponível, minimizando a potência dissipada. Todavia, projetistas devem manter em mente que a redução da tensão de alimentação afeta quadraticamente a confiabilidade geral contra *soft erros* (SALEHI, AZARPEYVAND, *et al.*, 2010), assim como SEUs induzidos por radiação. Por isso, neste trabalho, nos consideramos o uso da técnica de *power gating* para o desligamento de partes do circuito que não são utilizados, enquanto o tempo de computação é mantido junto ao limites impostos pelo *memory wall*.

Algoritmos que executam cálculos de baixa complexidade sobre uma grande quantidade de dados estão provavelmente tendo seu *throughput* limitado pelo tempo levado para ler as entradas e escrever as saídas, ainda que técnicas eficientes de forem empregadas. No caso do algoritmo de multiplicação de matrizes, a localidade temporal varia favoravelmente com as dimensões das matrizes. Considerando a operação AB = C sobre matrizes quadradas de dimensões $n \times n$. Enquanto que a quantidade mínima de acessos a memória aumenta em $O(n^2)$, o tempo de computação (utilizando o algoritmo tradicional) aumenta em $O(n^3)$. Em outras palavras, cada elemento lido é utilizado em n operações de multiplicação, significando que mais paralelismo deve ser explorado por

matrizes grandes se é para manter o tempo de computação limitado pela largura da banda de memória ao invés da latência lógica. Para exemplificar isto, vamos comparar os custos para a execução de 100 multiplicações de matrizes 10 × 10 e para uma única multiplicação de matrizes 100 × 100. Para ambos os casos, 20000 valores devem ser lidos e 10000 devem ser escritos na memória. No primeiro caso são executadas somente 100.000 multiplicações escalares, enquanto no segundo caso são necessárias 1.000.000, um *gap* de dez vezes. Isto significa a configuração de hardware necessária a finalizar a computação dentro da latência de acesso à memória varia de acordo com as dimensões matrizes a ser multiplicadas.

A dificuldade crescente encontrada no *scaling* da tensão de alimentação juntamente com as dimensões dos transistores leva a um aumento na densidade de potência e a necessidade de estrangular o *throughput* de forma a evitar a danificação do circuito. Além disso, o custo total do sistema relacionado com o encapsulamento e refrigeração estão diretamente relacionados com a potência máxima do chip esperada. A manutenção de uma baixa dissipação de potência total reduz os custos de projeto e aumenta a vida útil dos dispositivos.

Respeitar *power budgets* no nível de sistema, vem se tornando uma tarefa cada vez mais exigente, principalmente considerando a tendência atual dos MPSOCS. Nestes sistemas, diversas aplicações podem estar rodando em paralelo, portanto compartilhando fatias do mesmo *power budget* total, criando a necessidade de técnicas de gerenciamento de potência (NELSON, MOLNOS e GOOSSENS, 2011) (REN, KROGH e MARCULESCU, 2005). Como o comportamento em tempo de execução é, muitas vezes imprevisível, técnicas de gerenciamento dinâmico de potência, apresentadas em (REN, KROGH e MARCULESCU, 2005), são capazes de encontrar o melhor *power budget* disponível para um componente específico. Para aplicações sem rígidas restrições de tempo real, executar uma computação com menor esforço de forma a respeitar o *power budget* disponível significa manter o máximo desempenho garantindo ainda a integridade do dispositivo.

Além disso, quando consideramos aplicações com restrições de tempo real, assim com em (NELSON, MOLNOS e GOOSSENS, 2011), respeitando o *deadline* imposto junto com o *power budget* alocado pode se tornar crítico. Como alternativa, ao assumir que uma aplicação com um tempo real rígido possui alta prioridade dentro do sistema, o fato de garantir que o deadline será alcançado com potência mínima deixa uma grande porção de potência disponível para os componentes restantes, levando a uma melhora global de desempenho. Para este propósito, o tempo de computação determinístico, permite ao RA^3 determinar precisamente o tempo de computação necessário e encontrar a configuração ótima para a carga de trabalho atual e o tempo disponível antes do próximo deadline.

6.1 ESTUDO DE CASO I – Decodificação MIMO

Sistemas MIMO consistem em N_T antenas transmissoras e N_R antenas receptoras que utilizam a mesma banda de comunicação. Desta forma, o vetor y de N_R símbolos recebidos a cada ciclo é o produto do vetor x de N_T símbolos transmitidos e o caminho físico específico em cada transmissão para cada antena receptora. As mudanças de amplitude e fase causadas pelo canal formam uma matriz de resposta ao impulso H de dimensões $N_R \times N_T$. O vetor y é dado pelo modelo do canal $y = H.x + n$, onde n é o vetor de ruído. O receptor tenta recuperar x de y recebido, com o conhecimento da matriz H de forma a minimizar a taxa de bits de erro.

Existem diversos algoritmos de decodificação MIMO, preenchendo um espaço de projeto entre esforço computacional × *BER/SNR*. Muitos deles dependem da decomposição QR da matriz H, o que frequentemente aparece como gargalo do processo de decodificação (LUETHI, BURG, *et al.*, 2007) (NAZAR, GIMMLER e WHEN, 2010) (SALMELA, BURIAN, *et al.*, 2008). Como discutido anteriormente, as dimensões da matriz H são dadas pelo número de antenas no sistema. Sabe-se também que a matriz H é composta de números complexos, entretanto pode ser convertida para uma matriz real $2N_R \times 2N_T$ sem afetar o resto do sistema (FISCHER e WINDPASSINGER, 2003). Além disso, a matriz H varia de acordo com a posição relativa das antenas. Considerando que no mínimo, um dos lados da comunicação está se movendo, a matriz H será modificada com o passar do tempo e novas operações de decomposição QR serão necessárias. O tempo no qual a matriz H pode ser considerada como invariante é chamado de tempo de coerência $t_{coh} = c / vr.fc$ (SALMELA, BURIAN, *et al.*, 2008) (NAZAR, GIMMLER e WHEN, 2010), onde *c* é a velocidade da luz, *vr* é a velocidade do receptor e *fc* é a frequência da portadora. Dentro deste *time frame*, a decomposição QR de todos os dados das sub-portadoras em uso (assumindo um sistema MIMO-OFDMA) deve ser calculado. Por exemplo, assumindo um cenário crítico com *vr* = 400km/h e *fc* =5.8GHz (uma das frequências utilizadas no MIMO-WiMAX (IEEE , 2009)), o t_{coh} resultante é de 0.466ms. A quantidade *k* de sub-portadoras alocadas pode variar de usuário para usuário. Assumindo, por exemplo, *k* =96 (2 sub-canais de 48 subportadores cada (IEEE , 2009)) o sistema precisará realizar cada decomposição QR em aproximadamente 4.85µs. Operando com outras frequências portadoras ou quantidades diferentes de sub-portadoras leva a uma variação dos requisitos em tempo real, permitindo assim uma variação ótima da quantidade de recursos necessários visando minimização do consumo de potência.

A decomposição QR por si só, não é diretamente uma operação de multiplicação de matrizes, entretanto pode ser executada através de rotações de *Given's*, e componentes de hardware dedicado têm sido propostos para a execução desta tarefa (LUETHI, BURG, *et al.*, 2007). Estas rotações consistem em sucessivas multiplicações de matrizes para produzir as matrizes Q e R. As dimensões destas matrizes de acordo com a norma 802.16 (IEEE , 2009), por exemplo, assume no máximo N_T =4 e N_R=4, levando a matrizes H complexas 4×4 ou matrizes reais 8×8 da forma que serão utilizadas aqui. Cada decomposição QR de uma matriz 8×8 pode ser computada através de 28 multiplicações de matrizes (GOLUB e VAN LOAN, 1996). Assumindo nosso exemplo, cada multiplicação deve ser computada em 173ns.

Além dos requisitos de tempo real, a confiabilidade no processo de decomposição QR é de crítica relevância. Uma vez que a matriz resultado será usada subsequentemente para decodificar os frames de dados recebidos, a ocorrência de erros neste processo acarreta numa sobrecarga em outras camadas da pilha do protocolo de rede. Os aumentos nos requisitos de mecanismo de correção de erros e retransmissão de pacotes levam a um aumento do uso de energia e atraso na comunicação.

6.2 ESTUDO DE CASO II – Sistemas VoIP e Filtragem Adaptativa

Em sistemas VOIP (do inglês *Voice Over IP*) um dos maiores desafios está em melhorar a QOS (do inglês *Quality of service*) oferecida ao usuário em redes chaveadas por pacotes (GOODE, 2002). Os parâmetros mais impactantes, em termos de QOS, no transporte efetivo do tráfego VOIP nestas redes são a largura de banda, atraso, *jitter*, eco e perda de pacotes (OHRTMAN, 2002). Se precauções não forem tomadas estes

parâmetros podem comprometer o QOS global de sistemas VOIP, como gateways, roteadores, telefones e centrais IP (do inglês *Internet Protocol*). Neste trabalho foi escolhido o cancelamento de eco como estudo de caso, haja vista que sua complexidade de implementação e atraso imposto é determinante para o desempenho do sistema (BORGH, SCHULDT, *et al.*, 2011). Existem dois tipos principais de eco nos sistemas VOIP: eco de linha e eco acústico.

O problema do eco acústico se torna realmente um problema em aplicações como telefonia *hands-free*, vídeo conferência e necessitam de uma técnica para o cancelamento de eco acústico (AEC) para eliminar a realimentação acústica do *loudspeaker* para o microfone. Já o cancelamento de eco de linha (LEC) é necessário devido a um descasamento de impedâncias ocasionado nas híbridas, dispositivos destinados a conversão entre o circuito telefônico a dois fios que conecta as centrais aos telefones dos assinantes e o circuito telefônico a 4 fios utilizado entre as centrais. Tanto no caso do AEC quando no LEC filtros adaptativos são amplamente utilizados, e tem como principal objetivo estimar a resposta ao impulso e subtrair esta estimativa do sinal transmitido.

A resposta ao impulso da rede possui um comprimento típico de 64 a128ms (LIN, KHONG, *et al.*, 2008), enquanto que a resposta ao impulso acústica tem um comprimento típico de aproximadamente 300ms (ZOIA, STURZENEGGER e HOCHREUTINER, 2007). Considerando longas respostas impulsivas, assim com AEC, são necessários filtros adaptativos com um número grande de *taps*, que por consequência demandam um esforço computacional maior. Uma maneira muito utilizada para lidar com este problema é a utilização de filtros adaptativos no domínio da transformada (TDAF), devido a sua característica de processamento em blocos de dados, exatamente o cenário de sistemas VOIP.

Além dos requisitos de tempo real, também existem requisitos de confiabilidade importantes em sistemas VOIP, principalmente se algum erro ocorrer na etapa de atualização dos *taps* do filtro. Este erro pode fazer com que o filtro que está atuando para o cancelamento de eco entre em instabilidade prejudicando completamente a inteligibilidade da chamada VoIP.

7 RESULTADOS EXPERIMENTAIS

Neste capítulo serão mostrados e discutidos os resultados referentes à arquitetura RA^3 proposta no trabalho sob vários aspectos. Além dos tradicionais resultados de área, potência e desempenho, serão mostrados também resultados focados nos estudos de caso apresentados no capítulo 6 sob a ótica dos tópicos motivadores deste trabalho, que por sua vez foram apresentados no capítulo 1. Estes tópicos são:

- Utilização de coprocessadores dedicados para aceleração de aplicações específicas;
- Confiabilidade de circuitos digitais;
- Técnica de *Power Gating* para desligamento de unidades de *hardware*, ociosas temporariamente, para redução do consumo de potência;
- Exploração de paralelismo limitado pela banda de memória;
- Exploração de paralelismo em sistemas de tempo real;

A ideia a seguir é apresentar, de forma objetiva, resultados que sustentem e permitam uma análise da arquitetura RA^3 desenvolvida neste trabalho. Sendo assim, inicialmente podemos observar um diagrama em blocos da estrutura montada para a realização dos testes que originaram os resultados deste trabalho. Como poder ser observado, a arquitetura RA^3 será utilizada na forma de um coprocessador dedicado de *hardware*, auxiliando um ou mais processadores, no exemplo da figura um DSP *dual core*, na tarefa de acelerar a execução do algoritmo de multiplicação de matrizes dada sua grande utilização em aplicações atuais. Além da aceleração proporcionada pela arquitetura RA^3, não menos importante é sua capacidade intrínseca de detecção e correção de erros, que impacta positivamente na confiabilidade da estrutura.

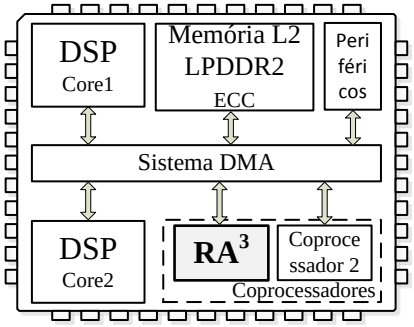
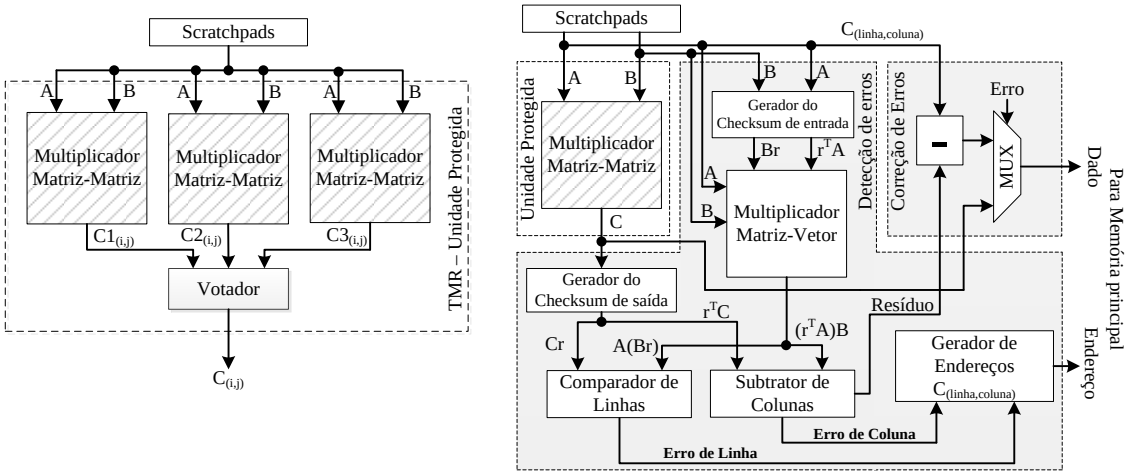


Figura 7.1: Utilização da arquitetura RA^3 como coprocessador embarcado.

7.1 Análise de Área da arquitetura RA^3

Nesta primeira seção de resultados, o objetivo é apresentar os dados referentes à área da arquitetura RA^3 e, além disso, compará-la com a reconhecida técnica de TMR. Para tal, a unidade protegida da RA^3 (Multiplicador Matriz-Matriz) foi triplicada, seguido da colocação de um votador de maioria para selecionar o resultado correto na saída dos módulos. Na Figura 7.2 é mostrado o diagrama em blocos das duas estruturas que terão suas áreas comparadas. Os módulos internos não estão em escala compatível com a área respectiva ocupada por cada um.



(a) (b)

Figura 7.2: Diagrama em blocos do TMR aplicado na unidade protegida (a) e da arquitetura RA^3 (b).

Na Figura 7.3, são mostrados os resultados desta comparação para tecnologia de 90nm obtidos na ferramenta *Design Compiler* da Synopsys®. No eixo vertical é apresentada a área em μm^2 para cada uma das configurações, enquanto que no eixo horizontal são apresentadas configurações com quantidades diferentes de multiplicadores. No primeiro caso (Número Multiplicadores/Unidade Protegida =1), a estrutura com TMR utilizará 3 multiplicadores (um para cada unidade protegida) enquanto que a arquitetura RA^3 utilizará apenas 2 (um na unidade Multiplicador Matriz-Matriz e o outro na unidade Multiplicador Matriz-Vetor). Isto faz com que em todas as configurações a estrutura TMR tenha 50% mais área de multiplicadores do que a arquitetura RA^3, impactando positivamente a favor da mesma. Por outro lado existem módulos de hardware, como geradores de vetor codificado de entrada e saída, comparadores e gerador de endereços que são específicos na RA^3 e que logo não existem na estrutura TMR. A questão é que estes módulos de hardware não escalam com o número de multiplicadores e sim com as dimensões máximas das matrizes envolvidas, que por sua vez são também mostradas no eixo horizontal (128×128, 256×256, 512×512, 1024×1024, 2048×2048 e 4096×4096) da Figura 7.3. Vale ressaltar que as dimensões máximas permitidas são definidas pelo usuário através do parâmetro DIMENSAO_MAXIMA, presente no arquivo *Definições_do_Usuario.vhd.*

Na figura podemos observar também que com 1 Multiplicador/Unidade Protegida é o único caso onde a área da técnica de TMR é menor do que qualquer valor de DIMENSAO_MAXIMA definida na arquitetura RA^3. A partir de 16

Multiplicadores/Unidade Protegida (32× RA^3 e 48× TMR), a técnica de TMR terá área superior a qualquer valor de dimensão máxima mostrada na figura.

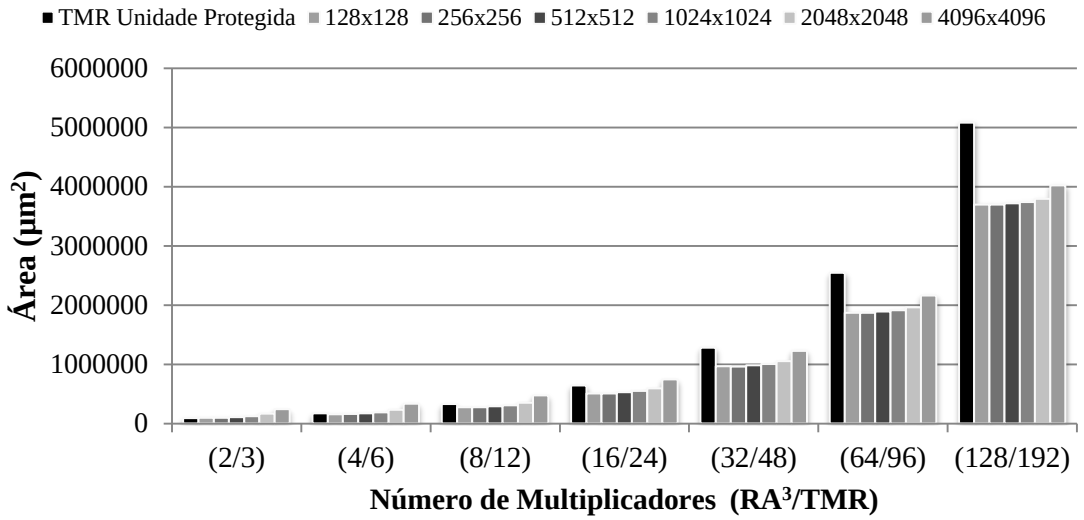


Figura 7.3: Comparação de Área entre TMR e RA^3.

7.2 Análise do consumo de potência da arquitetura RA^3

Como foi visto na seção 4.5 deste trabalho, a técnica de *Power Gating* foi aplicada em algumas das principais unidades de *hardware*, como somadores e multiplicadores, com o objetivo de permitir que as mesmas sejam ligadas e desligadas em tempo de execução, visando reduzir o consumo de potência global da arquitetura em momentos oportunos. Esta abordagem focada no controle de quais unidades devem estar funcionando para cada aplicação a cada instante de tempo dão o caráter adaptativo apresentado como uma das principais características da arquitetura RA^3.

A partir da colocação deste controle nas unidades, algumas simulações foram executadas visando encontrar a variação do consumo de potência da estrutura de acordo com o número de multiplicadores ligados dentro da estrutura. Para isto, foi gerada uma configuração do coprocessador RA^3 com 64 multiplicadores (32 no módulo *Multiplicador Matriz-Matriz* e 32 para o módulo *Multiplicador Matriz-vetor*) com a estrutura expandida da versão apresentada na Figura 4.17. Após isso, testes variando o número de multiplicadores ligados foram executados e os resultados dos mesmos podem ser vistos na Tabela 7.1. Estes resultados foram obtidos para tecnologia de 22nm operando na frequência de 1GHz com o auxílio das ferramentas *Design Compiler*, *Power Compiler* e *HSpice* utilizando os modelos PTM (NIMO GROUP) (do inglês *Predictive Technology Model*) dos transistores.

Componente	**Quantidade de Multiplicadores no RA^3**					
	2	**4**	**8**	**16**	**32**	**64**
Scratchpad	6.2	12.38	24.76	49.52	99.03	198.06
Multiplicadores	2.66	5.32	10.64	21.28	42.56	85.12
Somadores	0.30	0.4	0.7	1.11	2.07	3.99
Registradores	0.28	0.37	0.49	0.93	1.68	3.12
Controle	0.1	0.1	0.1	0.1	0.1	0.1
Total (mW)	9.54	18.57	36.69	72.94	145.44	290.39

Tabela 7.1: Consumo de potência total (mW) 22nm-1GHz.

Baseado nos dados da Tabela 7.1, é possível reparar que praticamente toda a potência é consumida pelas memórias *scratchpad* e pelos multiplicadores. Sendo assim, é possível observar a importância da adaptabilidade proporcionada pela arquitetura RA^3, que permite que seja encontrada e ligada a quantidade certa de recursos para a execução de uma determinada tarefa. Isto se torna principalmente importante para tecnologias atuais e futuras haja vista a crescente preocupação com o aumento da densidade de potência dos circuitos.

7.3 Análise de custos da técnica de *Freivalds Modificada* implementada na arquitetura RA^3

O objetivo desta seção é apresentar alguns dados que sejam capazes de permitir a avaliação do impacto causado no desempenho global da arquitetura imposto pela técnica de *Freivalds Modificada* ao algoritmo de multiplicação de matrizes que está em proteção. Como discutido em capítulos anteriores, a técnica não é completamente executada em paralelo com a execução do algoritmo de multiplicação de matrizes, haja vista as duas penalidades impostas pela técnica. Para isto foram executadas multiplicações de matrizes quadradas de diferentes dimensões na arquitetura RA^3, configurada com uma quantidade variada de multiplicadores. A figura mostra o *overhead* percentual de desempenho proporcionado pela execução da técnica de *Freivalds Modificada* em paralelo com a multiplicação de matrizes.

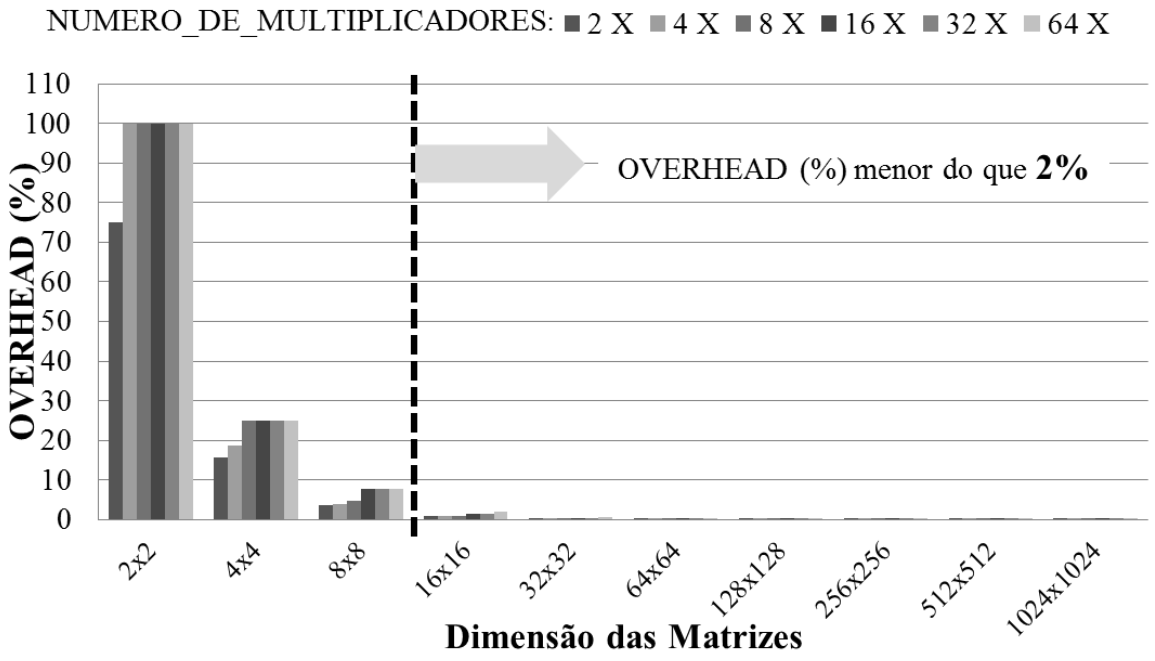


Figura 7.4: *Overhead* (%) imposto pela técnica de *Freivalds Modificada* na arquitetura RA^3.

Conforme podemos observar na Figura 7.4, o tempo de execução para multiplicação de matrizes 2×2 na arquitetuta RA^3 é, em quase todos os casos, o dobro do tempo para executar apenas o algoritmo de multiplicação de matrizes. Entretanto, à medida que as

dimensões das matrizes envolvidas aumentam, o *overhead* imposto pela técnica de *Freivalds Modificada* na arquitetura RA^3 é reduzido drasticamente. Para matrizes 4×4, por exemplo, o *overhead* cai, no pior caso para 25% e para multiplicação de matrizes 8×8 este número cai para abaixo dos 10%. Para multiplicação de matrizes com dimensões a partir de 16×16, o overhead se mantém abaixo dos 2%. Isto significa que quanto maior as dimensões das matrizes envolvidas, menor o impacto no tempo de execução total imposto pela técnica de detecção e correção proposta. Isso se dá pelo fato de que a complexidade do algoritmo de multiplicação de matrizes é $O(n^3)$ enquanto que as penalidades impostas pela técnica apresentam uma complexidade $O(n)$, fazendo com que a medida com que as dimensões das matrizes aumentem os custos do algoritmo de multiplicação de matrizes se sobreponha as penalidades impostas pela técnica.

7.4 Máximo Desempenho frente a um *Memory Wall*

Nesta subseção serão apresentados os resultados considerando as limitações impostas na exploração de paralelismo pela largura da banda de memória. Para isto, foi considerado um padrão de memória amplamente utilizado em sistemas embarcados, conhecido como LPDDR2 (SAMSUNG, 2011). A taxa de transferência destas memórias chega a 1066 Mbps por pino. Uma vez que a quantidade de pinos impacta diretamente no custo total, foram consideradas duas versões de LPDDR2: uma versão de 16 e outra de 32 pinos, chegando a larguras de banda totais de 16.66 Gbps e 33.31 Gbps respectivamente. Com estes valores é possível determinar o tempo mínimo necessário para buscar as matrizes de entrada e escrever a matriz resultante de volta.

Na Figura 7.5, são mostradas, em escala logarítmica, as latências de acesso à memória para diferentes tamanhos de matrizes, para memórias LPDDR2 com 16 e 32 pinos. São também mostradas as latências associadas com a multiplicação de tais matrizes, assumindo uma frequência de operação de 1GHz e também variando a quantidade de multiplicadores disponíveis de 2 a 64. Os resultados na figura permitem encontrar a quantidade mínima de recursos necessária para executar uma determinada computação dentro do tempo limite imposto pela latência da memória, atingindo assim, o desempenho máximo com mínima potência. Para matrizes 32 × 32, por exemplo, a escolha intuitiva seria em usar todos os recursos de forma a explorar o paralelismo máximo permitido pelo *hardware*. Entretanto, devido ao *memory wall*, 8 e 16 multiplicadores são suficientes para alcançar o desempenho máximo para casos de memórias com 16 e 32 pinos respectivamente.

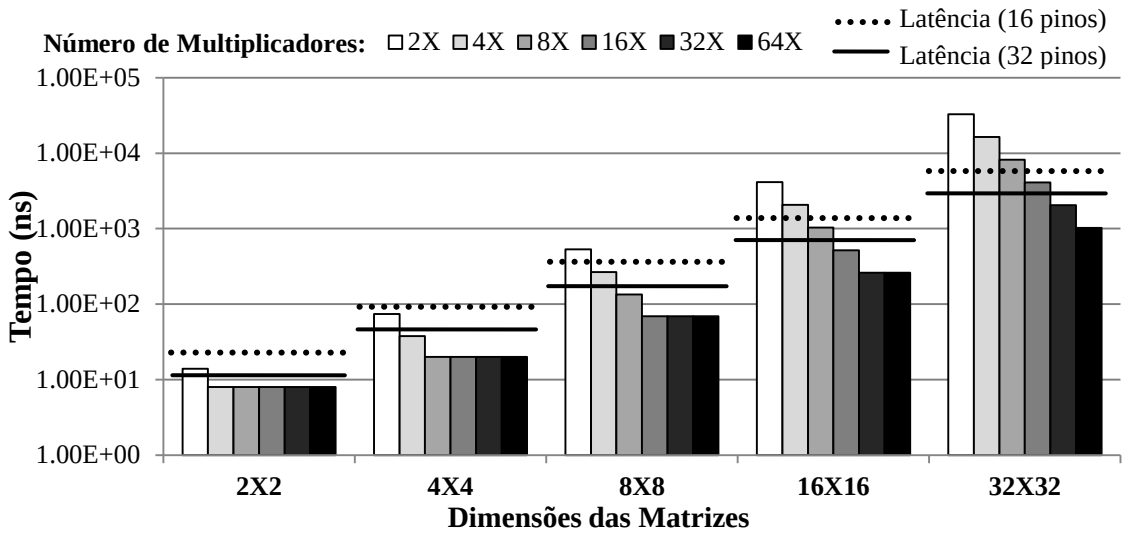


Figura 7.5: Latência de acesso a memória e tempo de execução.

7.5 Atingindo restrições de tempo real com mínima potência

O sistema MIMO proposto como estudo de caso no capítulo 6, requer 28 operações de multiplicação de matrizes de 8 × 8 para cada sub portadora de dados dentro do tempo de coerência. Para o cenário apresentado, cada multiplicação deve ser efetuada em menos de 173ns. Na Figura 7.6, o eixo horizontal apresenta o tempo de computação necessário para realizar as operações exigidas com quantidades diferentes de multiplicadores, enquanto que a linha tracejada indica a latência máxima aceitável de 173ns. Para este caso, podemos concluir que 8 é quantidade mínima de multiplicadores necessária para executar as 28 operações antes de atingir o deadline de 173ns, levando a uma redução de 7.9 vezes em termos de consumo de potência, comparado ao sistema completo com 64 multiplicadores. Se considerarmos um sistema completo menor, por exemplo, com 16 multiplicadores, uma redução de 1.9 vezes em potência é atingida. Desta forma, é mostrado que a arquitetura adaptativa permite atender determinados *deadlines* impostos pelas aplicações, além de permitir uma redução no consumo de potência total. Vale ressaltar que este é apenas um exemplo de *deadline*, ou seja, outros *deadlines* podem ser gerados para esta aplicação, uma vez que o número de antenas, velocidade, frequência da portadora e das subportadoras são parâmetros que mudam a todo instante e influenciam diretamente nesse tempo.

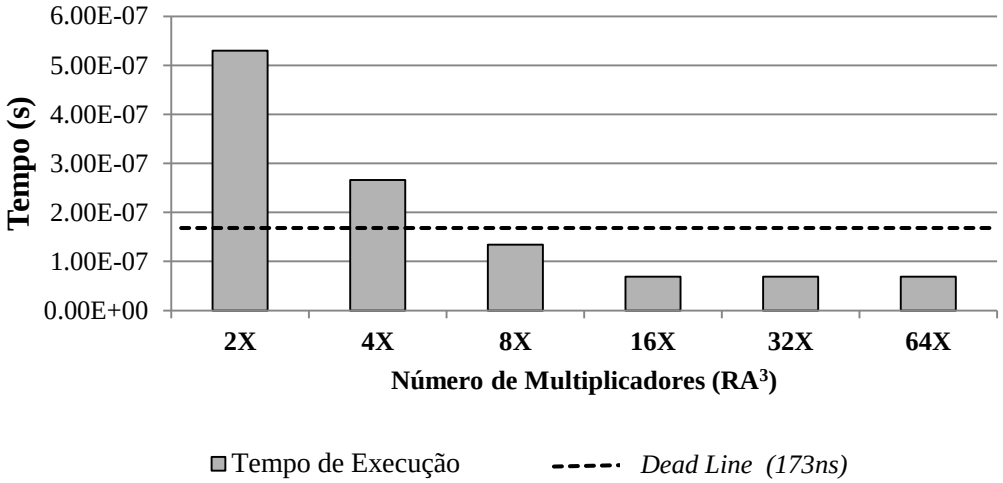


Figura 7.6: Restrições de tempo de execução e desempenho para decomposição QR.

No caso de sistemas VoIP, foi identificado que é necessário executar 16 operações de multiplicação de matrizes a cada intervalo de 20ms. Além disso, as dimensões destas matrizes estão associadas com o comprimento do filtro adaptativo aplicado ao cancelador de eco, que diz respeito ao número de *taps* do filtro. Esse parâmetro é geralmente definido pelo usuário em sistemas VoIP, e depende principalmente do ambiente em que ocorre a chamada, desta forma, foram criadas duas situações, a primeira considera o tempo de computação para filtros de 1024 *taps* e a segunda 2048 *taps*. Também é sabido que este *deadline* de 20ms é o tempo necessário para processar outros algoritmos VoIP além do AEC, por isso foi considerado também uma fatia de tempo equivalente a 50% do tempo total para o processamento de outros algoritmos. A Figura 7.7, mostra que a quantidade mínima de multiplicadores que devem permanecer ligados de forma a não ultrapassar o *deadline* de 20ms no filtro de 1024 *taps* é de 2 multiplicadores, enquanto que no caso do filtro de 2048 taps esta quantidade aumenta para 8 multiplicadores. Já para o *deadline* de 10ms (50% do tempo total de processamento) 4 multiplicadores é o número mínimo de multiplicadores no filtro de 1024 *taps*, enquanto que para um filtro de 2048 *taps* são necessários no mínimo 16 multiplicadores ligados para atender o *deadline* imposto. Esses mesmos dados demonstram que os *deadlines* das aplicações de tempo real podem ser utilizados como parâmetro de decisão no controle da quantidade de recursos que devem permanecer ligados, visando sempre minimizar o consumo de potência da arquitetura.

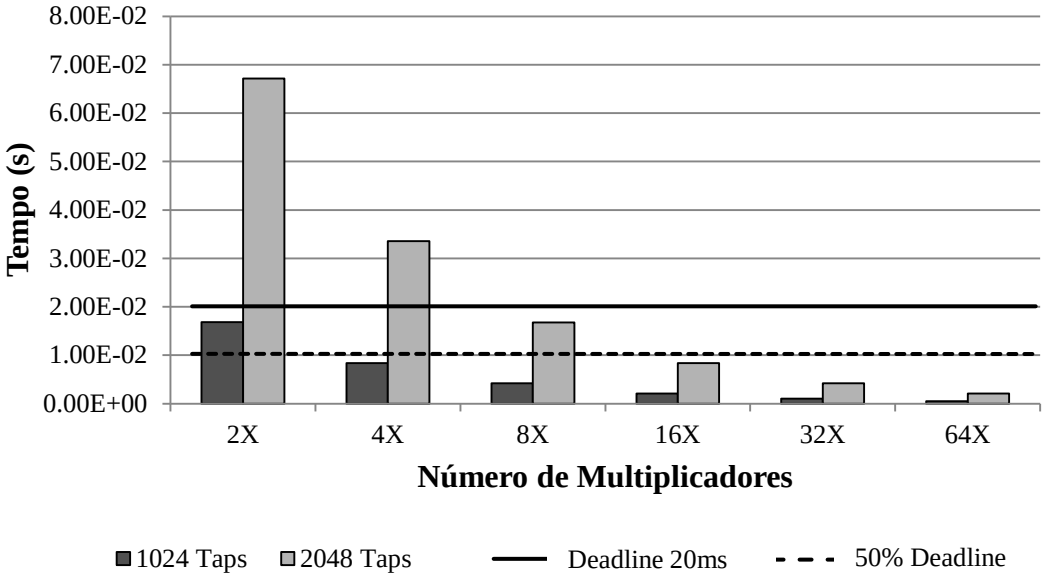


Figura 7.7: Restrições de tempo de execução e desempenho para o AEC.

7.6 Análise da cobertura de falhas da arquitetura RA^3

Nos experimentos de injeção de falhas, cujos resultados são mostrados na Tabela 7.2, foi utilizado o injetor de falhas apresentado no capítulo 5 deste trabalho. Para este processo, foi usada a mesma configuração da arquitetura RA^3, com o número total de multiplicadores igual a 64, permitindo multiplicações de matrizes de no máximo 1024×1024. Após isso, as unidades de *hardware* (incluindo multiplicadores, memórias e somadores) foram sendo desligadas dando origem a 6 configurações para o processo de injeção. Em cada uma destas 6 configurações, foi injetado um total de 50000 falhas somente nas áreas ligadas, e na linha referente ao número de erros (# Erros) é possível verificar o número de erros corrigidos/gerados. Em cima destes dados é possível determinar a taxa de cobertura de falhas proporcionada em cada uma das configurações.

Número Total de Multiplicadores na arquitetura RA^3						
Config.	**2×**	**4×**	**8×**	**16×**	**32×**	**64×**
# Erros	900/948	1563/1614	2727/2782	4345/4414	6693/6753	9236/9289
Cobertura	94.94%	96.84%	98.02%	98.44%	99.11%	99.43%

Tabela 7.2: Resultado de injeção de falhas na arquitetura RA^3.

Ainda de acordo com os dados da Tabela 7.2, podemos concluir que a taxa de cobertura de falhas aumenta junto com o número de multiplicadores. Isto ocorre pelo fato de que quando utilizamos configurações da arquitetura RA^3 com um número maior de multiplicadores, aumentamos a área da parte operativa dos dados da estrutura, onde a técnica de *Freivalds Modificada* possui uma proteção extremamente efetiva. Já a área da parte de controle permanece praticamente a mesma para todas as configurações, ou seja, proporcionalmente menor em relação à área total. Vale ressaltar que isto se torna possível devido à metodologia empregada no injetor de falhas utilizado neste trabalho, que leva em consideração a área das unidades de *hardware* no processo de injeção.

7.7 Comparação do tempo de execução com uma GPU

Esta seção apresenta uma comparação da arquitetura RA^3 contra uma GPU, executando a técnica de ABFT em software apresentada em (DING, KARLSSON, *et al.*, 2011). A comparação foi realizada tomando os tempos de execução originais apresentados também em (DING, KARLSSON, *et al.*, 2011) com os gerados através do modelo analítico de desempenho da RA^3 considerando a frequência de operação de 1GHz.

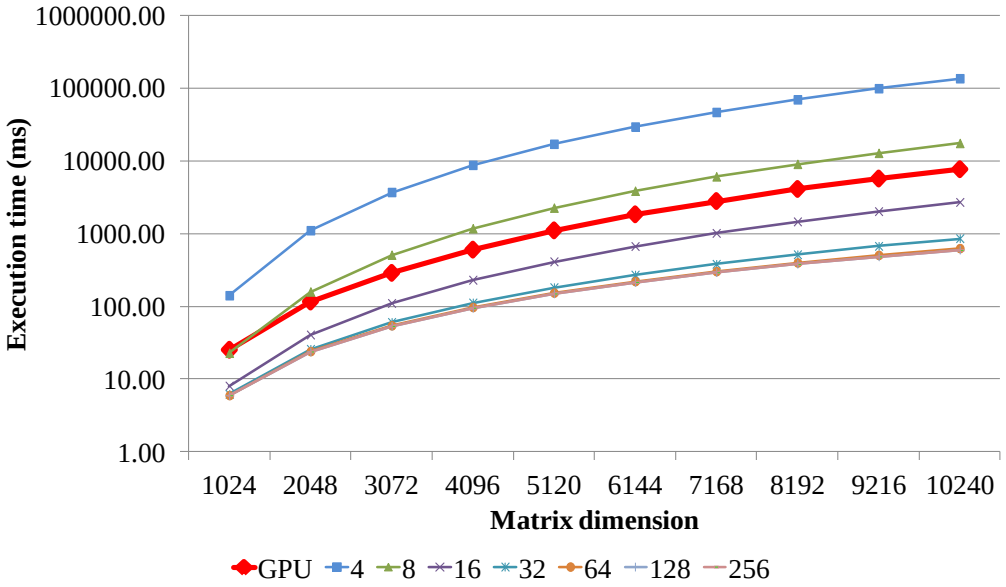


Figura 7.8: Comparação do tempo de execução do RA^3 com um GPU.

Conforme pode ser visto na Figura 7.8, a GPU apresenta melhor desempenho do que a arquitetura RA^3 com 4 multiplicadores para qualquer tamanho de matriz. Já com 8 multiplicadores, apenas com matrizes 1024×1024 o tempo de execução da RA^3 é ligeiramente inferior, entretanto a partir das matrizes de dimensões 2048×2048 o desempenho da GPU se torna superior. Com 16 ou mais multiplicadores o desempenho da arquitetura se torna superior ao da GPU para todas as dimensões das matrizes.

Os ganhos significativos apresentados pela arquitetura RA^3 quando comparada a uma GPU, podem ser principalmente creditados a maneira com que foram distribuídas as memórias *scratchpads*, permitido que a arquitetura conte com ampla largura de banda de memória e também a natureza dedicada do caminho de dados computacional da mesma. No caso da GPU, a hierarquia completa de memória e infraestrutura de comunicação foi construída para um amplo conjunto de aplicações, levando a um menos favorecido desempenho para o algoritmo de ABFT. Vale salientar novamente que a arquitetura conta com 2k multiplicadores, onde k multiplicadores são utilizados para multiplicar matrizes enquanto que os outros k multiplicadores são utilizados para implementar especificamente a técnica de ABFT.

8 CONCLUSÕES E TRABALHOS FUTUROS

Nesta dissertação foi apresentada a arquitetura RA^3 capaz de explorar paralelismo de maneira adaptativa visando respeitar restrições de tempo real ou limites impostos pela largura da banda de memória com mínima potência. A arquitetura permite também detecção e correção de erros com baixo custo e tempos de execução determinísticos, características desejáveis para sistemas críticos. Os resultados apresentados mostram a habilidade da arquitetura para computar com os limitadores de banda de memória e com a quantidade ótima de recursos para diferentes *workloads*. Duas aplicações reais e relevantes foram apresentadas como estudo de caso visando mostrar como a RA^3 é capaz de respeitar as restrições impostas e como diferentes configurações são relevantes para atender determinadas tarefas, gastando a menor quantidade de potência possível.

Foi também desenvolvido um injetor de falhas, que leva em consideração a área das unidades de hardware tornado este processo de injeção de falhas por simulação um pouco mais realista. Esta ferramenta de injeção de falhas permitiu também avaliar e confirmar e eficácia da técnica de *Freivalds Modificada* colocada na arquitetura.

E por fim, como trabalhos futuros, podemos citar:

- Prototipação em FPGA que permita uma injeção de falhas por radiação, possibilitando uma caracterização real da taxa de cobertura de falhas.
- Aplicação de técnicas de tolerância a falhas da parte de controle da arquitetura, uma vez que esta foi a parte responsável por praticamente todas as falhas não corrigíveis;
- Expandir o número de operações matriciais de forma a ampliar o número de aplicações que podem ser utilizadas pela arquitetura;
- Aplicação da técnica de DVFS e avaliar o impacto causado na confiabilidade da arquitetura.

REFERÊNCIAS

ANALOG DEVICES. ADSP-BF609: BLACKFIN DUAL-CORE PROCESSOR UP TO 1GHZ WITH HARDWARE SUPPORT FOR HD VIDEO ANALYTICS, 2012. Disponivel em: <http://www.analog.com/en/processors-dsp/blackfin/adsp-bf609/products/product.html>. Acesso em: 20 ago. 2012.

AZAMBUJA, J. R. F. **Análise de Técnicas de Tolerância a Falhas Baseadas em Software para a Proteção de Microprocessadores**. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática - UFRGS. Porto Alegre, p. 89. 2010.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 305-316, September 2005.

BORGH, M. et al. Low-Complexity Network Echo Cancellation Approach for Systems Equipped with External Memory. **IEEE Transactions on Audio, Speech and Language Processing**, v. 19, n. 8, p. 2506-2515, Novembro 2011.

BRUGUIER, G.; PALAU, J.-M. Single Particle Induced Latchup. **IEEE Transaction Nuclear Science**, v. 43, n. 2, p. 522-532, April 1996.

CALIN, T.; NICOLAIDIS, M.; VELAZCO, R. Upset Hardened Memory Design for Submicron CMOS Technology. **IEEE Transaction on Nuclear Science**, 6, Dezembro 1996. 2874-2878.

CHATTERJEE, I. et al. **Single-Event Charge Collection and Upset in 40-nm Dual and Triple-Well Bulk CMOS SRAMs**. IEEE Nuclear and Plasma Sciences Society. [S.l.]: [s.n.]. 2011. p. 2761-2767.

DING, C. et al. **Matrix Multiplication on GPUs with On-Line Fault Tolerance**. ISPA'11: IEEE International Symposium on Parallel and Distributed Processing with Applications. [S.l.]: IEEE. 2011. p. 311-317.

DIRK, J. D.; NELSON, M. E.; ZIEGLER, J. F. Terrestrial Thermal Neutrons. **IEEE TRANSACTIONS ON NUCLEAR SCIENCE**, v. 50, n. 6, p. 2060-2064, Dezembro 2003.

DIXIT, A.; HEALD, R.; WOOD, A. **Trends from ten years of soft error experimentation**. IEEE Workshop on Silicon Errors in Logic - System Effects. [S.l.]: [s.n.].

DODD, P. E. et al. Current and Future Challenges in Radiation Effects. **IEEE TRANSACTIONS ON NUCLEAR SCIENCE**, v. 57, n. 4, p. 1747-1762, August 2010.

DYER, C.; RODGERS, D. **Effects on Spacecraft & Aircraft Electronics**. Esa Workshop on Space Weather. Noordwijk, Holanda: [s.n.]. 1998.

ESMAEIZADEH, H. et al. **Dark Silicon and the end of multicore scaling**. ISCA'11. [S.l.]: ACM. 2011. p. 365-376.

FISCHER, R. F. H.; WINDPASSINGER, C. Real vs. Complex-Valued Equalization in V-BLAST Systems. **IEEE Electronics Letters**, v. 39, n. 5, p. 470-471, Março 2003.

FREEMAN, L. B. Critical charge calculations for a bipolar SRAM array. **IBM Journal of Research Development**, v. 40, n. 1, p. 119-129, 1996.

FREESCALE. MSC8126: Quad Core 16-bit DSP with Ethernet, TCOP and VCOP. Disponivel em: <http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MSC8126>. Acesso em: 20 ago. 2012.

FREIVALDS, R. **Fast probabilistic algorithms**. Mathematical Formulations of CS. Nova York, USA: Springer-Verlag. 1979. p. 57-69.

FUGITSU, L. http://www.fujitsu.com. **Sparc64 V Processor for Unix Server**, 2004. Disponivel em: <http://www.fujitsu.com/downloads/PRMPWR/sparc64_v_e.pdf>. Acesso em: 10 Janeiro 2012.

GAILLARD, R. Single Event Effects: Mechanisms and Classification. In: NICOLAIDIS, M. **Soft Errors in Modern Electronic System**. 1. ed. New York: Springer, 2011. Cap. 2, p. 27-54.

GOLLA, R. **Niagara 2:** A Highly Threaded Server-on-a-Chip, 2006. Disponivel em: <http://www.opensparc.net/pubs/preszo/06/04-SunGolla.pdf>.

GOLUB, G. H.; VAN LOAN, C. F. **Matrix Computations**. 3rd. ed. [S.l.]: The Johns Hopkins University Press, 1996.

GOODE, B. Voice Over Internet Protocol (VOIP). **Procedings of the IEEE**, v. 90, n. 9, p. 1495-1517, Setembro 2002.

GRINSCHGL, J. et al. **Automatic Saboteur Placement for Emulation-Based Multi-Bit Fault Injection**. International Workshop Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). MontPellier: Proceedings of. 2011. p. 1 - 8.

HAZUCHA, P.; SVENSSON, C. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. **IEEE Transactions on Nuclear Science**, v. 47, n. 6, p. 2586 - 2594, Dezembro 2000.

HUANG, K.; ABRAHAM, J. A. Algorithm-Based Fault Tolerance for Matrix Operations. **IEEE Transactions on Computers**, c-33, n. 6, Junho 1984. 518-528.

IEEE. **IEEE Std. 802.16-2009TM**. IEEE Computer Science and IEEE Microwave Theory and Tecniques Society. New York / USA. 2009.

ITRS. **International Technology Roadmap for Semiconductors**. [S.l.]. 2009.

JEDEC STANDARD. **Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors inSemiconductor Devices**. JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. Arlington, p. 84. 2006. (JESD89A).

KOREN, I.; KRISHNA, C. **Fault-Tolerant Systems**. San Francisco: Morgan Kaufmann, 2007.

LADBURY, R. **Radiation Hardening at the System Level**. Nasa Goddard Space Flight Center. Honolulu, Hawaii. 2007.

LEI, F. et al. An Atmospheric Radiation Model Based on Response Matrices Generated by Detailed Monte Carlo Simulations of Cosmic Ray Interactions. **IEEE transactions on Nuclear Science**, 51, n. 6, Dezembro 2004. 3442-3451.

LIN, X. et al. Frequency-Domain Adaptive Algorithm for Network Echo Cancellation in VoIP. **EURASIP Journal on Audio, Speech, Music Processing**, v. 2008, Janeiro 2008.

LISBÔA, C. A. L. **Dealing with Radiation Induced Long Duration Transient Faults in Future Technologies**. Tese (Doutorado em Computação) Instituto de Informática - UFRGS. Porto Alegre, p. 113. 2009.

LISBOA, C.; CARRO, L. **System level approaches for mitigation of long duration transient faults in future technologies**. IEEE European Test Symposium ETS'07. Freiburg: IEEE computer Society. 2007. p. 165 - 172.

LISBOA, C.; ERIGSON, M.; CARRO, L. **A low cost checker for Matrix Multiplication**. IEEE Latin-American Test Workshop, LATW. Cusco, Per: IEEE Computer Science Test Technology Technical Council. 2007.

LUETHI, P. et al. **VLSI Implementation of a high-speed iterative sorted MMSE QR decomposition**. IEEE International Symposium on Circuits and Systems. [S.l.]: [s.n.]. 2007. p. 1421-1424.

MAHATME, N. N. et al. **Analysis of soft error rates in combinational and sequential logic and implications of hardening for advanced technologies**. IEEE International Reliability Physics Symposium. [S.l.]: [s.n.]. 2010. p. 1031-1035.

MAIZ, J. et al. **Characterization of Multi-Bit Soft Error Events in Advanced SRAMs**. IEEE International Electron Devices Meeting. [S.l.]: [s.n.]. 2003. p. 21.4.1-21.4.4.

MAY, T. C.; WOODS, M. H. A New physical Mechanism for soft errors in Dynamic Memories. **Reliability Physics Symposium**, Abril 1978. 33-40.

MUKHERJEE, S. **Architecture Design for Soft Errors**. Burlington: Morgan Kaufmann, 2008.

NAZAR, G. L.; GIMMLER, C.; WHEN, N. **Implementation comparisons of the QR decomposition for MIMO detection**. 23rd Symp. on Int. Circuits and System Design. [S.l.]: ACM. 2010. p. 210-214.

NELSON, A.; MOLNOS, A.; GOOSSENS, K. **Composable Power Management with Energy and Power Budgets per Application**. SAMOS '11: International Conference on Embedded Computer Systems. [S.l.]: IEEE. 2011. p. 396-403.

NIMO GROUP. PTM- Predictive Technology Model. Disponivel em: <http://ptm.asu.edu/>. Acesso em: 11 maio 2012.

OHRTMAN, F. **Softswitch:** Architecture for VoIP. 1st. ed. [S.l.]: McGraw-Hill Professional, 2002.

REN, Z.; KROGH, B. H.; MARCULESCU, R. Hierarchical Adaptive Dynamic Power Management. **IEEE Transactions on Computers**, v. 54, n. 4, p. 409-420, 2005.

SALEHI, M. E. et al. **Reliability considerations in dynamic voltage and frequency scaling schemes**. DTIS '10: 5th International Conference on Design and Technology of Integrated Systems in Nanoscale Era. [S.l.]: [s.n.]. 2010. p. 1-4.

SALMELA, P. et al. **Complex-valued QR decomposition implementation for MIMO receivers**. International Conference on Acoustics, speech and signal processing. [S.l.]: IEEE. 2008. p. 1433-1436.

SAMSUNG. Samsung Green LPDDR2, 2011. Disponivel em: <http://www.samsung.com/global/business/semiconductor/minisite/Greenmemory/Products/LPDDR2/LPDDR2_Features.htm>. Acesso em: 22 nov. 2011.

SEIFERT, N. Radiation-induced Soft Errors: A Chip-level Modeling Perspective. **Foundations and Trends in Electronic Design Automation**, v. 4, n. 2-3, p. 99-221, 2010.

SEIFERT, N. et al. Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices. **IEEE International Reliability Physics Symposium**, March 2006. 217-225.

SHIVAKUMAR, P. et al. **Modeling the effect of technology trends on the soft error rate of combinational logic**. IEEE Dependable Systems and Networks Conference. [S.l.]: [s.n.]. 2002. p. 389-398.

TEXAS INSTRUMENTS. TMS320C6670, 2012. Disponivel em: <http://www.ti.com/product/tms320c6670>. Acesso em: 20 ago. 2012.

TÍPTON, A. D. et al. Multiple-Bit Upset in 130 nm CMOS Technology. **IEEE Transactions on Nuclear Science**, p. 53:3259-3264, December 2006.

VELASCO, R.; FRANCO, F. J. **Single Event Effects on Digital Integrated Circuits:** Origins and Mitigation Techniques. ISIE. [S.l.]: [s.n.]. 2007. p. 3322- 3327.

WEBER, T. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. UFRGS. Porto Alegre. 2002.

YU, H.; XIAOYA, F.; NICOLAIDIS, M. Design Trends and Challenges of logic Soft Errors in Future Nanotechnologies Circuits Reliability. **International conference on Solid-State and Integrated-Circuit Technology**, October 2008. p. 651-654.

ZIEGLER, J. F.; LANFORD, W. A. The effect of sea level cosmic rays on electronic devices. **Journal of Applied Physics**, v. 52, n. 6, p. 4305-4318, 1981.

ZOIA, G.; STURZENEGGER, A.; HOCHREUTINER, O. **Audio Quality and Acoustic Echo Issues for VOIP on portable Devices**. PORTABLE '07: International Conference on Portable Information Devices. [S.l.]: IEEE. 2007. p. 1-5.

APÊNCICE A REFERÊNCIAS DAS PUBLICAÇÕES OBTIDAS COM ESTE TRABALHO E OUTRAS CONTRIBUIÇÕES

ITTURRIET, F. P. ; FERREIRA, RR ; GIRAO, G. ; NAZAR, G. ; Moreira, AF ; CARRO, L. . **Resilient Adaptive Algebraic Architecture for Parallel Detection and Correction of Soft-Errors**. In: 15th EUROMICRO Conference on Digital System Design (DSD 2012), 2012, Cesme, Turkey. Proceedings of the 15th EUROMICRO Conference on Digital System Design. Los Alamitos: IEEE, 2012.

ITTURRIET, F. P. ; NAZAR, G. ; FERREIRA, RR ; Moreira, AF ; CARRO, L. . **Adaptive Parallelism Exploitation under Physical and Real-Time Constraints for Resilient Systems**. In: 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC 2012), 2012, York. Proceedings of the 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip. Los Alamitos: IEEE, 2012. p. 1-8.

* Este artigo foi indicado para nova submissão para a ACM Transactions on Reconfigurable Technology and Systems (TRETS), com deadline para 17/12/2012.

ITTURRIET, F. P. ; FERREIRA, RR ; CARRO, L. . **Fault-Tolerant Algebraic Architecture for radiation induced soft-errors.** In: IEEE 17th European Test Symposium (ETS 2012), 2012, Annecy. Proceedings of the 17th IEEE European Test Symposium. Los Alamitos: IEEE, 2012. p. 1.

FERREIRA, R.R. ; ITTURRIET, F. P. ; AGUIAR, C. Z. ; Moreira, AF ; CARRO, L. . **Single-Instruction HW/SW Unified Stack for Accelerated and Resilient Application Execution.** In: Architectural Support for Programming Languages and Operating Systems (ASPLOS '12), 2012, Londres. ASPLOS 2012 Provocative Ideas Session. New York: ACM, 2012.

JUNQUEIRA, A. ; RUTZIG, Mateus Beck ; ITTURRIET, F. P. ; CARRO, L. . **A Reconfigurable Fabric Supporting Full C/C++ Input**. In: International Workshop on Reconfigurable Communication-centric Systems-on-Chip, 2011, Montpellier. Proceedings of., 2011.