

UNIVERSIDADE FEDERAL DE PELOTAS
INSTITUTO DE FÍSICA E MATEMÁTICA
DEPARTAMENTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO



**UMA INVESTIGAÇÃO SOBRE IMPLEMENTAÇÕES
TOLERANTES A FALHAS TRANSIENTES DE SOMADORES
CARRY LOOKAHEAD E RIPPLE CARRY EM FPGA**

Helen de Souza Franck

Pelotas, 2007

Helen de Souza Franck

**UMA INVESTIGAÇÃO SOBRE IMPLEMENTAÇÕES TOLERANTES A FALHAS
TRANSIENTES DE SOMADORES *CARRY LOOKAHEAD* E *RIPPLE CARRY* EM
FPGA**

Trabalho acadêmico apresentado ao Curso de Ciência da Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Luciano Volcan Agostini

Co-Orientador: Prof. Dr. José Luís Almada
Güntzel

Pelotas, 2007

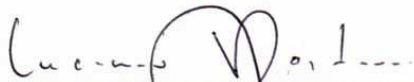
Dados de catalogação na fonte:
Ubirajara Buddin Cruz – CRB-10/901
Biblioteca de Ciência & Tecnologia - UFPel

F822i Franck, Helen de Souza
Uma investigação sobre implementações tolerantes a falhas transientes de somadores carry lookahead e ripple carry em FPGA / Helen de Souza Franck ; orientador Luciano Volcan Agostini ; co-orientador José Luís Almada. – Pelotas, 2007. – 137f : il. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2007.

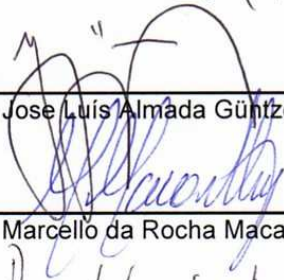
1.Informática. 2.Microeletrônica. 3.Arquiteturas de somadores. 4.Falhas transientes. 5.Tolerância a falhas. 6.SETs. 7.FPGAs. 8.VHDL. I.Agostini, Luciano Volcan. II.Almada, José Luís. III.Título.

DD: 004.2

Banca examinadora:

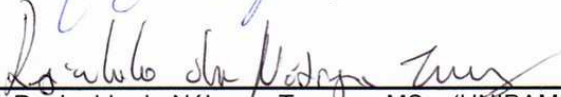


Prof. Luciano Volcan Agostini, Msc. (Orientador)



Prof. Jose Luis Almada Güntzel, Dr. (Co-Orientador)

Prof. Marcello da Rocha Macarthy, MSc.



Prof. Reginaldo da Nóbrega Tavares, MSc. (UNIPAMPA)



Prof. Vagner Santos da Rosa, MSc. (FURG)

*“Sabemos que todas as coisas
cooperam para o bem daqueles
que amam a Deus, daqueles são
chamados segundo o seu propósito.”*

Rm 8:28

Agradecimentos

Agradeço a Deus, em primeiro lugar por ter me dado como pais duas pessoas maravilhosas que me ensinaram as lições que realmente importam na vida: lições de amor, sinceridade, amizade, hospitalidade, respeito, fé, caráter e principalmente, me ensinaram o sentido da vida. É por eles e para eles que estou concluindo mais essa etapa. Amo vocês e muito obrigada por tudo.

Em segundo lugar, agradeço a Deus por sempre ter colocado na minha vida as pessoas certas, desde os primeiros anos de colégio até agora, principalmente pelo meu orientador, Güntzel, a quem vou sempre agradecer pela conversa que teve comigo no Praça XV que encorajou a entrar pro GACI (a minha decisão mais acertada de todos os anos de faculdade), e por toda a paciência e dedicação com que sempre me orientou, agradeço a ele todas as experiências ótimas que tive no GACI tanto academicamente como também pelas amizades geradas por um ambiente de trabalho agradável.

A todos os colegas do GACI, Zanetti, Guilherme, Meskita, Matheus, Fabiane, João, Rafael e Elvio, pelas ajudas, pelos cafés, chimarrão e risadas, principalmente ao Meskita e ao Guilherme que me ajudaram mais diretamente neste trabalho.

Ao professor Luciano Agostini, que mesmo à distância, foi importante para a realização deste trabalho.

Ao Matheus, minha dupla de trabalho desde o terceiro semestre. Muito obrigada por todas as explicações pacientes, pelas inúmeras caronas independentes de horário, na volta das longas jornadas de trabalhos ou estudos, pela companhia e amizade.

Ao Zanetti, por toda a amizade, preocupação e ajuda durante o semestre mais difícil. E ao Meskita por ser a melhor e mais engraçada dupla de pesquisa que eu poderia ter tido.

Ao Lucas, São Risal, amigo que sempre lembrarei com enorme carinho.

A todos meus colegas de turma que sempre tornaram as aulas mais divertidas, em especial ao grupo de estudos para as provas do Gil e slides do professor Ricardo: Rodrigo (nosso tradutor :P), Meskita, Presidente, Foguinho, Matheus e Zanetti.

Aos professores com quem tive aula ao longo do curso, em especial ao professor Gil Medeiros e Luciano Agostini.

Muito obrigada a todos!

Resumo

A evolução da tecnologia de fabricação CMOS (*Complementary Metal-Oxide Semiconductor*) tem permitido reduções drásticas das dimensões dos transistores, o que proporciona um aumento no número de transistores que podem ser integrados em um único *chip* e também frequências de operação mais elevadas. Para tornar isto possível, algumas modificações nas técnicas e materiais de fabricação são necessárias, bem como a redução na tensão de alimentação dos transistores. Em função destas modificações, e também devido à redução das dimensões dos transistores, os circuitos fabricados em tecnologia CMOS estado-da-arte estão se tornando vulneráveis a falhas transientes, sobretudo aquelas provocadas pela colisão de partículas energéticas que provêm do espaço e encontram-se presentes na atmosfera terrestre, fazendo com que a confiabilidade de tais circuitos diminua. Quando uma partícula carregada de alta energia colide com alguma região sensível de um circuito (drenos de transistores que estão desligados) ela perde energia e produz pares lacuna-elétron livres, resultando em uma trilha de ionização. A ionização gera um pulso transiente no circuito que pode ser interpretado como um sinal lógico. Em um circuito combinacional, o pulso pode propagar-se até ser armazenado em um elemento de memória. Tal fenômeno é denominado *Single-Event Transient* (SET). Atualmente, os dispositivos programáveis (FPGAs) se constituem na principal opção para a implementação física de sistemas eletrônicos integrados. Por utilizarem tecnologia CMOS VDSM, os FPGAs estão se tornando vulneráveis às falhas transientes mencionadas anteriormente. Por essa razão, várias técnicas de Tolerância a Falhas para sistemas implementados em FPGAs têm sido propostas.

Este trabalho tem como principal objetivo a análise de técnicas de proteção de circuitos somadores do tipo *Carry Lookahead* e *Ripple Carry* implementados em FPGAs, objetivando aumentar assim a confiabilidade de tais arquiteturas, em relação a falhas transientes. As técnicas de proteção investigadas foram: *Triple Module Redundancy* (TMR), *Time Redundancy* (TR) e *Duplication With Comparison combined with Time Redundancy* (DWC+RT). Os resultados obtidos permitem a quantificação do acréscimo de recursos e da degradação de desempenho resultantes da aplicação de cada técnica de proteção às arquiteturas de somadores investigadas. Em particular, os resultados indicam qual técnica de proteção é mais adequada para cada tipo de somador, quando o projeto precisa otimizar o uso de recursos ou precisa maximizar o desempenho.

Palavras-chave: Microeletrônica, Falhas transientes, SETs, FPGAs, VHDL, arquiteturas de somadores, Tolerância a Falhas.

Abstract

The evolution of the CMOS (Complementary Metal-Oxide Semiconductor) fabrication technology has allowed drastic transistor dimension reductions, thus providing an increase in the number of transistors that can be integrated in a single *chip* as well as higher operation frequencies. In order to achieve that some modifications in the fabrication process and materials, as well as the voltage supply, were needed. Due to these modifications, and also due to the reduction of the transistor dimensions, the circuits manufactured in the state-of-the-art CMOS technology are becoming vulnerable to transient faults, especially those provoked by the collision of energetic particles that come from outer space or those existing in the terrestrial atmosphere. Such phenomena, known as transient faults or soft errors, reduce circuit operation reliability. When an energetic particle of high energy hits the sensible region of a circuit (drains of transistors that are turned off) it loses energy and produces pairs whole-electron-electrons, resulting in an ionization track. Ionization generates a transient pulse in the circuit that can be interpreted as a logical signal change. In a combinational circuit the pulse can propagate down the logic and eventually, be stored in a memory element. Such phenomenon is called Single-Event Transient (SET). Currently, FPGAs are the main option for the physical implementation of integrated electronic systems. As long as FPGAs are fabricated using state-of-the-art CMOS technology, they are becoming vulnerable to the already mentioned transient faults. Therefore, several fault-tolerance techniques for systems implemented in FPGAs have been proposed. This work has as main objective the analysis of transient fault protection techniques applied to Carry Lookahead and Ripple Carry adders implemented in FPGA, aiming to increase the reliability of such architectures under the presence of transient faults.

The investigated protection techniques were: Triple Module Redundancy (TMR), Time redundancy (TR) and Duplication with Comparison combined with Time Redundancy (DWC+RT). The obtained results allow to quantify the amount of extra resources needed and the performance degradation resulting from the application of each protection technique to the investigated adder architectures. In particular, the results show which protection technique is more appropriate for each type of adder when either resource or performance is to be optimized.

Keywords: Microelectronics, Transient Fault, SET, FPGA, VHDL, Adder architectures, Fault Tolerance.

Lista de Figuras

Figura 1 – Fases da coleta da carga gerada pela colisão e o pulso gerado	24
Figura 2 – Formas de onda geradas pela equação 1	26
Figura 3 – Modelagem do mecanismo de geração de um pulso transiente	26
Figura 4 – Single event upset (SEU) em uma célula de memória SRAM.....	28
Figura 5 – NAND com uma entrada em valor controlante.....	29
Figura 6 – Mascaramento Elétrico.....	29
Figura 7 – Mascaramento por <i>Latching Window</i>	30
Figura 8 – <i>Single-event transient</i> (SET) e sua possível propagação em um bloco combinacional	31
Figura 9 – Estrutura básica de um FPGA.....	34
Figura 10 – Conexões SRAM.....	36
Figura 11 – Anti-fusível	36
Figura 12 – LUT implementada com células de memórias (a) e com multiplexadores e células de memórias (b)	37
Figura 13 – LUT de quatro entradas e Tabela-verdade	38
Figura 14 – Inclusão de um <i>flip-flop</i> em um bloco básico de um FPGA	39
Figura 15 – Diagrama de bloco de Stratix II	42
Figura 16 – Diagrama de bloco de alto nível do ALM Stratix II.....	43
Figura 17 – Estrutura do LAB de um Stratix II	44
Figura 18 – Fluxograma genérico de um projeto FPGA	46
Figura 19 – Circuito Lógico de um Meio-Somador.	48
Figura 20 – Símbolo para um Meio-Somador.....	49
Figura 21 – Circuito lógico de um Somador-Completo.....	50
Figura 22 – Símbolo para um Somador-Completo de um bit	50
Figura 23 – Somador <i>Ripple Carry</i> de n bits sem <i>carry</i> de entrada.....	51
Figura 24 – Somador <i>Ripple Carry</i> de n bits com <i>carry</i> de entrada.....	52
Figura 25 – Somador <i>Ripple Carry</i> de 8 bits construído a partir de dois blocos somadores de 4 bits	52
Figura 26 – Caminho crítico de um somador RCA de 4 bits	53
Figura 27 – Circuitos lógicos para a geração dos sinais auxiliares, <i>Carry Generate</i> e <i>Carry Propagate</i> (a) e soma (b) para o i-ésimo estágio	55
Figura 28 – Unidade <i>carry lookahead</i>	56
Figura 29 – Somador CLA de 8 bits com nível simples.....	57
Figura 30 – Unidade <i>Carry Lookahead</i> alterada para a implementação de somadores <i>Carry Lookahead</i> com níveis hierárquicos	58
Figura 31 – Somador CLA de 8 bits utilizando dois níveis	59
Figura 32 – Caminho crítico de um somador CLA de 32 bits	60
Figura 33 – Adaptação do modelo dos três universos	62
Figura 34 – Latência de Falha e Latência de Erro.....	63
Figura 35 – Redundância de <i>hardware</i> Dinâmica com S reservas.....	67
Figura 36 – Sistema híbrido com TMR e S módulos reservas	68

Figura 37 – Técnica NMR (<i>N-Modular Redundancy</i>)	72
Figura 38 – Redundância Modular Tripla (TMR)	72
Figura 39 – Circuito Votador.....	73
Figura 40 – Órgão Regenerador (<i>Restoring Organ</i>).....	74
Figura 41 – Redundância Temporal utilizada para mascarar falhas transientes.....	75
Figura 42 – Redundância Temporal usada para a detecção de falhas permanentes	76
Figura 43 – Células de Memórias protegidas contra SEUs com a técnica TRM	77
Figura 44 – Blocos Combinacionais protegidos contra SETs através da redundância de <i>hardware</i> (a) e redundância temporal (b)	77
Figura 45 – Bloco Combinacional protegido contra SETs com Duplicação e CWSP	78
Figura 46 – DWC combinada com Redundância temporal	80
Figura 47 – Bloco básico <i>Carry Lookahead</i> de 4 bits	83
Figura 48 – Bloco <i>Generate-Propagate</i> para 4 bits	83
Figura 49 – Somador <i>Carry Lookahead</i> de 4 bits não protegido	84
Figura 50 – Bloco <i>Generate-Propagate</i> para 8 bits	84
Figura 51 – Somador <i>Carry Lookahead</i> de 8 bits não protegido	85
Figura 52 – Somador <i>Ripple Carry</i> de 4 bits não protegido	85
Figura 53 – Circuito que é triplicado para a obtenção do CLA TMR de 4 bits	86
Figura 54 – Circuito votador para os somadores protegidos de 4 bits	87
Figura 55 – Somador <i>Carry Lookahead</i> de 4 bits com TMR	87
Figura 56 – Circuito triplicado para a geração de um CLA TMR de 8 bits.....	88
Figura 57 – RCA TMR de 8 bits	89
Figura 58 – Somador CLA de 4 bits juntamente com três registradores e votador	90
Figura 59 – Máquina de estados descrita para a técnica TR e sinais gerados	90
Figura 60 – Somador <i>Carry Lookahead</i> de 4 bits protegido com TR	91
Figura 61 – Detector de Erros de 1 bit	92
Figura 62 – Circuito Detector de Erros para somadores de 4 bits.....	92
Figura 63 – Circuito CLA DWC+TR de 4 bits intermediário.....	93
Figura 64 – Máquina de estados descrita para a técnica DWC+TR e sinais de controle	93
Figura 65 – Somador DWC+TR de 4 bits.....	94
Figura 66 – Código VHDL com o uso da diretiva <i>syn_keep</i>	95
Figura 67 – Atraso crítico dos somadores não-protegidos	96
Figura 68 – Recursos utilizados pelos somadores CLA e RCA.....	97
Figura 69 – Atraso crítico dos somadores protegidos com TMR e não-protegidos.....	99
Figura 70 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e não- protegidos	102
Figura 71 – Atraso crítico dos somadores protegidos com TR e não-protegidos.....	104
Figura 72 – Número de ALUTs utilizadas pelos somadores protegidos com TR e não- protegidos.	106
Figura 73 – Atraso crítico dos somadores protegidos com TMR e TR	108
Figura 74 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e TR	110
Figura 75 – Atraso crítico dos somadores protegidos com DCW+TR e não-protegidos	112
Figura 76 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e não-protegidos	115

Figura 77 – Atraso crítico dos somadores protegidos	117
Figura 78 – Número de ALUTs utilizadas pelos somadores protegidos.....	118
Figura 79 – Atraso crítico dos somadores protegidos com TR e DWC+TR	119
Figura 80 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e TR	121
Figura 81 – Atraso crítico dos somadores protegidos com TMR e DWC+TR	123
Figura 82 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e TMR	125
Figura 83 – Algoritmo de Injeção de Falhas	127
Figura 84 – Fluxograma do processo injeção e simulação automática de falhas.....	128

Lista de Tabelas

Tabela 1 – Características gerais da família Stratix II	45
Tabela 2 – Resultado da Soma e Carry para a soma do bit menos significativo.....	48
Tabela 3 – Resultado da soma e carry levando se em consideração o carry do estágio anterior.....	49
Tabela 4 – Comparação do atraso crítico e recursos utilizados entre os somadores CLA e RCA não-protegidos.....	98
Tabela 5 – Acréscimo do atraso crítico gerado pela técnica TMR para somadores CLA e RCA.....	100
Tabela 6 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e não-protegidos.	101
Tabela 7 – Comparação entre os recursos utilizados pelos somadores protegidos com TMR e não-protegidos.....	103
Tabela 8 – Comparação entre os atrasos críticos dos somadores protegidos com TR e não-protegidos.	105
Tabela 9 – Comparação entre os recursos utilizados pelos somadores protegidos com TR e não-protegidos	107
Tabela 10 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e protegidos com TR.....	109
Tabela 11 – Comparação entre os recursos utilizados pelos somadores protegidos com TMR e protegidos com TR.....	111
Tabela 12 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e não-protegidos.....	113
Tabela 13 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e não-protegidos.....	116
Tabela 14 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e com TR	120
Tabela 15 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e protegidos com TR.....	122
Tabela 16 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e protegidos com TMR.....	124
Tabela 17 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e protegidos com TMR.....	126
Tabela 18 – Resultados da injeção e simulação de falhas para CLAs de 4 bits.	129
Tabela 19 – Resultados da injeção e simulação de falhas para RCAs de 4 bits.....	129

Lista de Abreviaturas e Siglas

ALM	<i>Adaptive Logic Module</i>
ALUT	<i>Adaptive Look Up Table</i>
CLA	<i>Carry Lookahead Adder</i>
CLB	<i>Configurable Logic Block</i>
CMOS	<i>Complementary Metal-Oxide Silicon</i>
CPLD	<i>Complex Programmable Logic Device</i>
DPS	<i>Digital Signal Processing</i>
DRAM	<i>Dinamic Random Access Memory</i>
DWC	<i>Duplication with Comparison</i>
DWC+TR	<i>Duplication with Comparison combined with Time Redundancy</i>
EDAC	<i>Error Correction and Detection Code</i>
EPROM	<i>Erasable Programmable Read Only Memory</i>
EEPROM	<i>Electrically Programmable Read Only Memory</i>
FIFO	<i>First In First Out</i>
FIT	<i>Failure InTtime</i>
FPGA	<i>Field Programmable Gate Array</i>
LAB	<i>Logic Array Block</i>
LE	<i>Logic Element</i>
LET	<i>Llinear Energy Transfer</i>
LUT	<i>Look-Up Table</i>
NMOS	<i>N-channel Metal-Oxide-Semiconductor</i>
PLA	<i>Programmable Logic Array</i>
PMOS	<i>P-channel Metal-Oxide-Semiconductor</i>
RAM	<i>Random Access Memory</i>
RCA	<i>Ripple Carry Adder</i>
SEB	<i>Single Event Burnout</i>
SEE	<i>Single Event Effect</i>
SEGR	<i>Single Event Gate Rupture</i>
SEL	<i>Single Event Latchup</i>
SER	<i>Soft Error Rate</i>
SET	<i>Single-Event Transient</i>
SEU	<i>Single-Event Upset</i>
SRAM	<i>Static Random Access Memory</i>
TMR	<i>Triple Modular Redundancy</i>
IOB	<i>Input and Output Block</i>
LUT	<i>Look-Up Table</i>
PLD	<i>Programmable Logic Device</i>
RAM	<i>Random Access Memory</i>

SEL	<i>Single-Event Latchup</i>
SET	<i>Single-Event Transient</i>
SEU	<i>Single-Event Upset</i>
DSP	<i>digital signal processing</i>
LB	<i>Configurable Logic Block</i>
TMR	<i>Triple Modular Redundancy</i>
VDSM	<i>Very Deep Sub-Micron</i>

Sumário

1	Introdução	16
1.1	Organização da Monografia.....	19
2	Falhas Transientes.....	20
2.1	Fontes de Falhas Transientes	21
2.2	Geração do Pulso Transiente e Mecanismo de <i>Charge Collection</i>	23
2.3	SEUs e SETs.....	27
2.4	Propagação do SET.....	31
3	FPGA	33
3.1	Estrutura Genérica de um FPGA	34
3.2	Estrutura dos Dispositivos da Família Stratix II da Altera	41
3.3	Fluxograma Genérico do Projeto FPGA	45
4	Somadores.....	47
4.1	Meio-Somador	47
4.2	Somador-Completo.....	49
4.3	<i>Ripple Carry Adder</i>	51
4.4	<i>Carry Lookahead Adder</i>	53
5	Técnicas de Proteção contra Falhas Transientes	61
5.1	Terminologia e Conceitos	61
5.2	Tolerância a Falhas	64
5.3	Projetos de Circuitos Tolerantes a Falhas Transientes.....	70
6	Experimentos Práticos e Resultados.....	81
6.1	Arquiteturas Descritas.....	82
6.2	Diretivas de síntese da linguagem VHDL	94
6.3	Comparação entre os Somadores RCA e CLA não-protegidos.....	95
6.4	Comparação entre os somadores Protegidos com TMR e Não-Protegidos	99
6.5	Comparação entre os somadores protegidos com TR e Não-Protegidos.....	104
6.6	Comparação entre os somadores protegidos com TR e com TMR	108
6.7	Comparação entre os somadores protegidos com DWC+RT e Não-Protegidos	112
6.8	Comparação entre os somadores protegidos com DWC+RT e com TR	118
6.9	Comparação entre os somadores protegidos com DCW+RT e com TMR	123
6.10	Análise da Proteção dos Somadores Contra SETs	126
7	Conclusões.....	130
8	Referências	133

1 Introdução

Com a diminuição do tamanho dos transistores, a confiabilidade dos circuitos integrados começa a ser ameaçada por fatores antes não relevantes. Os componentes atuais, fabricados com tecnologia de 90nm ou inferior, já exibem uma maior vulnerabilidade a falhas devido à redução da carga crítica necessária para levar um transistor à condução. Tal evolução da tecnologia de fabricação CMOS (*Complementary Metal-Oxide Semiconductor*) se caracteriza por apresentar velocidades de operação mais rápidas e uma maior densidade de integração, ou seja, uma maior quantidade de transistores em um único *chip*.

Para que tal evolução possa ocorrer garantindo o funcionamento dos circuitos fabricados com tecnologias recentes, mudanças no processo de fabricação dos transistores têm sido necessárias, como por exemplo, a redução da tensão de alimentação e da tensão de limiar (*threshold*). Com isto, tem diminuído a confiabilidade dos circuitos fabricados com tecnologias estado-da-arte, os quais passam a apresentar uma baixa imunidade a ruídos e a falhas transientes causadas principalmente pela radiação proveniente do espaço.

A radiação é constituída por partículas energéticas compostas principalmente de íons provenientes de raios cósmicos, partículas alfa, nêutrons e prótons emitidos nos períodos de atividade solar intensa.

Quando uma partícula energética colide com a região sensível de um transistor, isto é, com o dreno de um transistor que se encontra em estado desligado (*off*), caso esta partícula tenha energia suficiente, ela irá penetrar na região ativa, gerando uma trilha de ionização a qual poderá se estender até o substrato (ou poço, conforme o tipo de transistor). Se esta trilha atingir o substrato, uma corrente elétrica irá

se estabelecer, podendo gerar um pulso transiente de tensão, o qual poderá ser interpretado como uma mudança de nível lógico.

Caso a colisão da partícula ocorra em uma região sensível de um elemento de memória poderá ocasionar a inversão do valor lógico nela armazenado. Este fenômeno recebe o nome de *Single-Event Upset* (SEU). Caso a colisão ocorra com uma região sensível de um circuito combinacional, um pulso transiente pode ser gerado. Neste caso, o fenômeno é denominado *Single-Event Transient* (SET). SEUs e SETs são classificadas como falhas transientes por não serem defeitos de fabricação e por não causarem nenhum dano físico ao dispositivo onde ocorrem.

Quanto mais próximo da superfície terrestre, menor é a quantidade de partículas e menor é a energia contida nestas, logo, até pouco tempo atrás, falhas transientes causadas pela radiação eram uma preocupação apenas para aplicações espaciais. Porém, com a redução das dimensões dos transistores e conseqüentemente, das capacitâncias internas ao circuito integrado, partículas de menor energia passam a poder ocasionar falhas transientes em sistemas operando na superfície terrestre.

Como a tendência da evolução da tecnologia CMOS prevê uma contínua redução das dimensões dos dispositivos, este problema tende a se agravar mais nos próximos anos. Portanto, como falhas tem se tornado cada vez mais comuns, e como as soluções tecnológicas disponíveis são economicamente inviáveis e/ou pouco eficientes, é necessário incorporar técnicas de Tolerância a Falhas ao projeto de circuitos e sistemas integrados.

Atualmente, os dispositivos programáveis tipo FPGA (*Field Programmable Gate Array*) se constituem na principal opção para a implementação física de sistemas eletrônicos integrados. Por utilizarem tecnologia CMOS estado-da-arte os FPGAs também estão se tornando suscetíveis a falhas transientes provocadas pela colisão de partículas energéticas, conforme mencionado anteriormente.

Diante disto, vários métodos de Tolerância a Falhas têm sido propostos para a implementação de sistemas eletrônicos em FPGAs, buscando tornar os sistemas confiáveis e com alto desempenho, mesmo na ocorrência de falhas. Os métodos de Tolerância a Falhas consistem basicamente na inclusão de recursos adicionais para garantir o funcionamento dos sistemas. Tais recursos normalmente têm a forma de

elementos redundantes, incluindo componentes de *hardware* ou *software*, redundância de informação ou redundância temporal.

A base deste trabalho é a comparação entre arquiteturas de somadores com e sem proteção contra falhas transientes. A implementação, em *hardware*, de somadores é essencial para qualquer projeto que possua operações aritméticas. Logo, o desempenho destes é um fator crítico para o sucesso de tais projetos. Assim, passa a ser de suma importância a análise de qual técnica de Tolerância a Falhas é mais eficiente na proteção de somadores, em função das características arquiteturais de cada tipo de somador.

Portanto, levando-se em consideração a ocorrência de um único SET em um circuito combinacional, este trabalho estuda e compara arquiteturas de somadores protegidos e não-protegidos contra falhas transientes do tipo SETs.

Foram escolhidos para serem estudados e comparados os somadores *Carry Lookahead*, cujo princípio básico é paralelizar o cálculo dos *carries*, diminuindo com isso, o atraso da cadeia de propagação do *carry*, e os somadores *Ripple Carry*, uma vez que estes últimos se constituem em referencial para comparações de desempenho (em termos de atraso crítico) e de uso de recursos de *hardware*.

As técnicas de proteção aplicadas a ambos os tipos de somadores foram as técnicas de Tolerância a Falhas que se baseiam em redundância de *hardware*, redundância temporal e a combinação de ambas. Após proteger os somadores mencionados através das técnicas de proteção, a eficiência das mesmas foi analisada, levando-se em consideração o desempenho e os recursos utilizados por tais somadores.

Como não existe nenhuma técnica capaz de garantir um projeto totalmente imune a falhas, faz-se necessária a análise de qual das técnicas aplicadas aos somadores *Ripple Carry* e *Carry Lookahead* amenizará a vulnerabilidade com uma menor penalidade em termos de custo de implementação e de desempenho.

1.1 Organização da Monografia

Esta monografia está organizada da seguinte forma. O capítulo 2 descreve as conseqüências do avanço da tecnologia CMOS nos dispositivos construídos com tal tecnologia, em relação a falhas transientes. Este capítulo também discorre sobre as fontes de tais falhas e como estas podem se propagar através dos circuitos.

No capítulo 3 é descrita a estrutura genérica de um FPGA juntamente com a descrição da família de FPGAs Stratix II da Altera, visto que os dispositivos escolhidos para a implementação das arquiteturas estudadas pertencem a esta última família. O fluxograma genérico de um projeto em FPGAs também é explicado.

O capítulo 4 apresenta os princípios do projeto de somadores, com especial atenção para os somadores, *Ripple Carry* e principalmente *Carry Lookahead*.

No capítulo 5, a questão da Tolerância a Falhas é apresentada, inicialmente de forma abrangente, explicando as técnicas de proteção existentes. Após, são apresentadas as principais técnicas de proteção de circuitos.

O capítulo 6 contém os experimentos práticos, explicando de forma detalhada o modo como cada arquitetura foi descrita. Os resultados de síntese são apresentados e analisados profundamente e de várias formas, por meio de comparações entre as técnicas de proteção implementadas, bem como com comparações entre as arquiteturas não-protegidas e protegidas dos somadores *Ripple Carry* e *Carry Lookahead*.

Finalmente, no capítulo 7 são apresentadas as conclusões obtidas com a realização deste trabalho.

2 Falhas Transientes

Com o avanço das tecnologias de fabricação dos circuitos integrados CMOS (*Complementary Metal-Oxide-Silicon*), questões referentes à confiabilidade e à robustez dos sistemas eletrônicos têm se tornado cada vez mais preocupantes. Essa preocupação se deve à redução drástica das dimensões dos transistores, a qual tem por objetivo permitir maiores densidades de integração e velocidades de operação mais altas, à redução da tensão de alimentação e da tensão de limiar (*threshold*), às menores capacitâncias e também às frequências de relógio cada vez mais elevadas.

Estes fatores têm contribuído para reduzir a imunidade dos circuitos a ruído, podendo permitir que pequenas variações de tensão sejam interpretadas como inversões do sinal lógico, de modo que está cada vez mais difícil garantir que os circuitos fabricados com tecnologias CMOS estado-da-arte irão operar de maneira confiável durante todo o período de vida útil.

Logo, uma outra consequência importante destes ajustes é o aumento da suscetibilidade dos circuitos a falhas transientes, também conhecidas como *soft errors* (COHEN, 1999; BAUMANN, 2001; IROM et al., 2002; MAVIS; EATON, 2002).

Uma falha pode ser classificada como permanente transiente ou intermitente. Uma falha permanente é aquela que existirá indefinidamente se alguma ação corretiva não for tomada. Uma falha transiente é aquela que aparece e desaparece no sistema durante um curto período de tempo e não causa danos permanentes aos dispositivos onde ocorre. Já a falha intermitente aparece, desaparece e reaparece repetidamente.

Uma falha transiente pode ser decorrente de um evento de radiação, ou seja, quando uma partícula energética colide com um dispositivo causando um distúrbio de

carga suficiente para reverter o estado lógico de um sinal ou mesmo dos dados de um elemento de armazenamento.

Atualmente, um dos efeitos induzidos por irradiação mais preocupantes para aplicações terrestres comerciais são os *single-event effects* (SEEs). Estes podem ser classificados nas seguintes categorias: *Soft SEEs* (*Single Event Transients* (SETs) e *Single Event Upsets* (SEUs)), *Hard SEEs* (*Single Hard Errors* (SHEs)), e *Destructive SEEs* (*Single Event Latchups* (SELs), *Single Event Burnouts* (SEBs) e *Single Event Gate Ruptures* (SEGRs)).

O foco do presente trabalho é a proteção de circuitos somadores contra *Soft SEEs* (que são *soft errors*), do tipo *Single Event Transients*, ou simplesmente, SETs.

2.1 Fontes de Falhas Transientes

A taxa em que os *soft errors* ocorrem é chamada de *soft error rate* (SER). A unidade de medida geralmente usada para SER é a falha no tempo (FIT-*failure in time*). Um FIT é equivalente a uma falha em 10^9 horas do dispositivo (JEDEC, 2001; BAUMANN, 2005).

O valor do distúrbio que um íon causa depende de transferência da energia linear (*linear energy transfer* ou LET) desse íon, medida em $\text{MeV}\cdot\text{cm}^2/\text{mg}$. Em um substrato de silício, um par elétron-lacuna é produzido para cada 3.6 eV da energia perdida pelo íon. O LET depende da massa e da energia da partícula e também do material em que esta penetra. Partículas mais maciças e mais energéticas, penetrando em materiais mais densos, têm um LET maior (BAUMANN, 2005).

No espaço, a atividade solar é a principal causa da radiação, a qual é constituída por partículas energéticas, tais como elétrons, prótons ou íons energéticos, e também pela radiação eletromagnética, a qual pode ser constituída por raios X, raios gama ou luz ultravioleta (BARTH, 1997; BAUMANN, 2001).

Ao nível do mar três mecanismos diferentes geram os íons energéticos responsáveis por induzir os *soft errors*: as partículas alfa, oriundas de impurezas presentes no encapsulamento e no próprio circuito integrado, nêutrons, gerados pela

interação entre raios cósmicos de alta energia com átomos da atmosfera terrestre e raios cósmicos de baixa energia (NORMAND, 1996; BAUMANN, 2005).

A principal causa de *soft errors* em dispositivos DRAM (*Dinamic Random Access Memory*) são as partículas alfa emitidas pelas impurezas presentes no encapsulamento e no próprio circuito integrado. Estas impurezas emitem partículas alfa em energias discretas específicas sobre uma escala de 4 a 9 MeV.

Quando uma partícula alfa percorre o material do dispositivo, ela perde energia cinética através de interações com os elétrons deste material, gerando uma trilha de ionização. Quanto mais elevada a energia da partícula alfa, maior será a distância percorrida por esta dentro do material. Logo, esta distância se dá em função da energia da partícula e das propriedades do material (principalmente a densidade do material) em que esta percorre. No silício, a distância para uma partícula alfa de 10 MeV é $< 100 \mu\text{m}$ (BAUMANN, 2005).

A segunda fonte significativa para o aumento do SER está relacionada aos raios cósmicos. Eles interagem com a atmosfera terrestre e produzem cascatas de partículas secundárias.

Em altitudes terrestres, menos de 1% do fluxo alcança o nível do mar - onde o fluxo é isotrópico e composto por muons, prótons, nêutrons, e pions. Os nêutrons são um dos componentes principais do fluxo, e como as reações do nêutron têm LET maior, elas são as radiações cósmicas mais prováveis de causar um *soft errors* nos dispositivos em altitudes terrestres (BAUMANN, 2005).

O fluxo dos nêutrons é dependente da altitude e da localização geográfica, visto que a intensidade do fluxo de nêutrons nos raios cósmicos aumenta com o aumento da altitude.

Os nêutrons não geram diretamente a ionização no silício, eles interagem com os materiais do *chip* criando reações nucleares que causam recombinações, as quais criam uma região de ionização podendo causar *soft errors*, contribuindo então para o aumento do SER (BAUMANN, 2005).

A terceira fonte significativa de partículas energéticas é a radiação gerada pela interação dos nêutrons dos raios cósmicos de baixa energia com o Boro. O Boro é

usado extensivamente como um dopante do tipo P, em implantes no silício e usado também na formação de camadas dielétricas.

O boro é composto de dois isótopos, ^{11}B e ^{10}B . O ^{10}B é instável quando exposto aos nêutrons. O ^{11}B também reage com os nêutrons; entretanto, sua reação é quase um milhão de vezes menor, e os produtos desta reação, os raios gama, são menos prejudiciais.

Portanto, para determinar exatamente o SER de qualquer dispositivo, devem-se somar os três mecanismos citados acima: partículas alfa emissoras das impurezas radioativas nos materiais do dispositivo, a radiação cósmica terrestre em forma de nêutrons de alta energia, e as reações de ^{10}B induzidas pelos nêutrons de baixa energia (BAUMANN, 2005).

Quanto mais próximo da superfície terrestre, menor é a energia das partículas (NORMAND; BAKER, 1993). Apesar disto, *soft errors* ocasionados por partículas energéticas em circuitos integrados operando na superfície terrestre estão tornando-se cada vez mais preocupantes. Como mencionado anteriormente, isto se deve ao fato dos circuitos integrados fabricados nas tecnologias CMOS mais recentes permitirem transistores com dimensões muito reduzidas. Logo, as capacitâncias dos nós internos a tais circuitos são muito pequenas, o que os deixa mais vulneráveis aos efeitos transientes ocasionados por partículas com menor energia (JOHNSTON, 2000; DUPONT; NICOLAIDIS; ROHR, 2002). Como a redução das dimensões dos dispositivos tende a continuar, este problema irá se agravar nos próximos anos.

2.2 Geração do Pulso Transiente e Mecanismo de *Charge Collection*

As junções P-N dos drenos dos transistores que estão desligados (em estado *off*) constituem-se nas regiões sensíveis dos transistores (BAUMANN, 2001). Quando um íon energético colide com uma destas regiões sensíveis, uma trilha cilíndrica de pares elétron-lacuna e uma elevada concentração de portadores são formados seguindo a passagem do íon, conforme ilustra a Fig. 1a. Quando a trilha de ionização resultante atravessa ou se aproxima da região de depleção (região vazia de cargas), portadores são coletados rapidamente pelo campo elétrico, que cria uma

corrente/tensão transiente nesse nó. Uma característica notável deste evento é que a região de depleção toma forma de um funil. Este funil aumenta a eficiência da coleta da carga devido ao aumento da região de depleção dentro do substrato, o que é ilustrado pela Fig. 1b. O tamanho deste funil é função da dopagem do substrato – o funil é maior para substratos menos dopados. Esta fase da coleta ocorre dentro de 1ns (BAUMANN, 2005) e é seguida pela fase onde a difusão começa a dominar o processo da coleta (Fig. 1c).

A carga adicional é coletada enquanto os elétrons se difundem na região de depleção em uma escala de tempo mais longa (centenas de ns), até todos os portadores adicionais serem coletados, recombinados, ou difundidos pela área da junção. No geral, quanto mais distante da junção o evento ocorre, menor a quantidade de carga que será coletada, sendo menos provável que a colisão cause um *soft error* (BAUMANN, 2005).

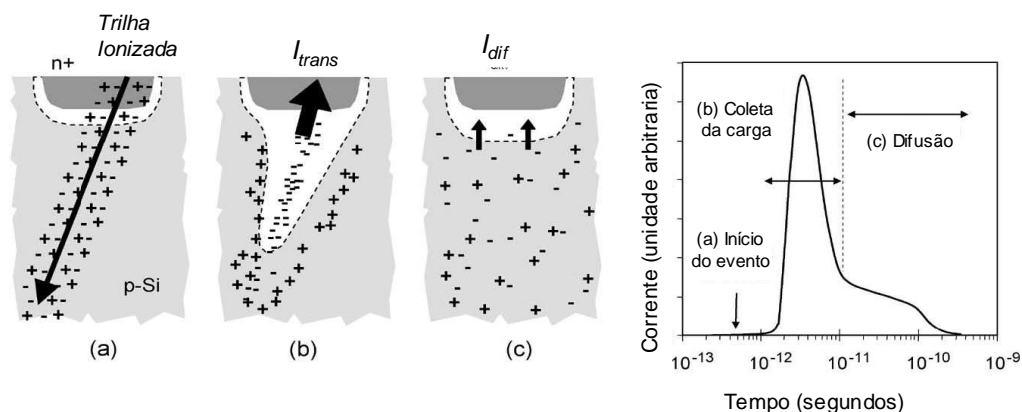


Figura 1 – Fases da coleta da carga gerada pela colisão e o pulso gerado
Fonte: BAUMANN, 2005

A coleta da carga geralmente ocorre dentro de aproximadamente $2 \mu\text{m}$ da junção do dispositivo (GADLAGE, 2004). Assim, a carga coletada (Q_{coll}) para estes eventos varia de 1 até 100 fC, dependendo do tipo de íon, sua trajetória e sua energia. O valor de Q_{coll} também depende de uma combinação de fatores, incluindo o tamanho do dispositivo, disposição dos vários nós do circuito, a estrutura do substrato, a

dopagem do dispositivo, a posição inicial do evento dentro do dispositivo e o estado do mesmo.

Além de Q_{coll} , a carga mínima coletada, por uma região sensível de um dispositivo devido à colisão de uma partícula energética capaz de causar um *soft error* precisa ser examinada. Esta carga é chamada de Carga Crítica (Q_{crit}) (JEDEC, 2001). Q_{crit} é definida primeiramente pela capacitância do nó, pela tensão de operação, e pela realimentação do transistor. Q_{crit} também depende das características do pulso da radiação e da resposta do circuito a este, dificultando a modelagem deste efeito (BAUMANN, 2005).

Em células de armazenamento do tipo DRAM (*Dinamic Random Access Memory*), um *soft error* será induzido quando uma colisão resultar em $Q_{coll} > Q_{crit}$. Inversamente, se a colisão resultar em $Q_{coll} < Q_{crit}$, então o circuito não sofrerá nenhum *soft error*. No caso de SRAM (*Static Random Access Memory*) ou *latches*, Q_{crit} aumenta com o tempo, já que existe uma realimentação ativa que faz com que velocidades mais baixas possibilitem mais tempo para o circuito restaurar um dado errôneo, reduzindo, conseqüentemente, a probabilidade de ocorrer um *soft error* (JEDEC, 2001; BAUMANN, 2005).

Na Fig. 1 a curva representa o pulso transiente real, gerado pela colisão da partícula energética com o dispositivo. Porém, para analisar a suscetibilidade dos circuitos combinacionais a falhas transientes e a propagação de tais falhas através dos circuitos, é necessário modelar matematicamente o pulso gerado.

Em 1982, Messenger modelou o pulso gerado na saída de uma porta lógica pela colisão de uma partícula através de uma fonte de corrente cujo comportamento obedece a uma dupla exponencial:

$$I(t) = I_0 (e^{-t/t\alpha} - e^{-t/t\beta}) \quad (1)$$

Na equação (1), I_0 é a corrente máxima, $t\alpha$ é a constante de tempo da junção e $t\beta$ é a constante de tempo para estabelecer o impacto inicial da partícula. A Fig. 2

mostra as formas de onda para uma tecnologia estado-da-arte (100nm), considerando diversos valores de I_0 e $t\beta$.

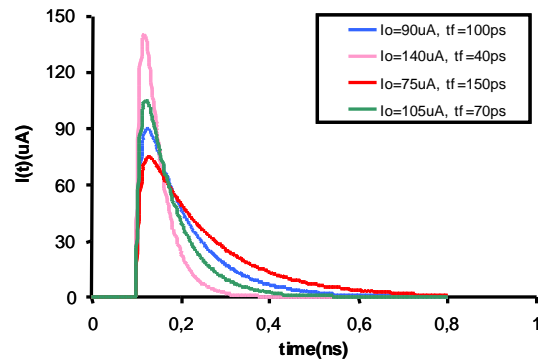


Figura 2 – Formas de onda geradas pela equação 1

A duração e amplitude desta corrente dependem de diversos fatores, tais como energia da partícula, dimensões da área ativa afetada e perfil da distribuição dos dopantes (MESSENGER, 1982). Se esta corrente tiver amplitude e/ou duração suficiente para carregar (ou descarregar) o capacitor da saída do transistor, um pulso transiente de tensão será gerado, o qual poderá inverter o nível lógico da saída da porta lógica. A Fig. 3 mostra a modelagem proposta por Messenger.

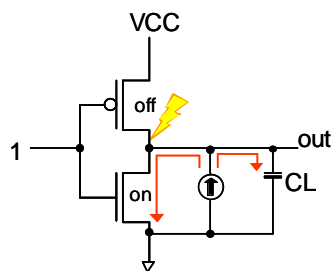


Figura 3 – Modelagem do mecanismo de geração de um pulso transiente

2.3 SEUs e SETs

Como mencionado anteriormente, um dos tipos de SEEs são os *Soft SEEs*, que podem ser divididos em *Soft Event Upset* (SEU) e *Single Event Transient* (SET).

O que define o tipo do SEE é a região em que a partícula energética colide. A partícula pode colidir com uma região sensível da lógica combinacional ou com um elemento de memória (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002).

Quando uma partícula colide com o dreno de um transistor PMOS que se encontra em estado *off* em uma célula de memória SRAM, o pulso de tensão gerado pode carregar a capacitância associada a este dreno, ligando assim o transistor por ele controlado. Em função das características de uma célula de memória SRAM, tais como laço de realimentação e dimensionamento dos transistores, esta descarga de capacitor poderá ser interpretada como uma mudança de nível lógico. Caso a partícula energética colida com o dreno do transistor NMOS que se encontra em estado *off*, o efeito é similar, exceto que a capacitância associada ao dreno será descarregada.

Em ambos casos, o pulso de tensão induzido acaba por causar a inversão indesejada do valor lógico armazenado na célula de memória. Ou seja, pode ocorrer um *bit-flip* na célula de memória.

O fenômeno da colisão de uma partícula energética em uma região sensível de um elemento de memória e a conseqüente inversão do valor lógico nela armazenado recebe o nome de *Single-Event Upset*, ou SEU (BAUMANN, 2001; DODD; MASSENGILL, 2003; NICOLAIDIS, 2005). A Fig. 4 mostra o *bit-flip* causado pela colisão de uma partícula energética no dreno de um transistor PMOS de uma célula de memória SRAM.

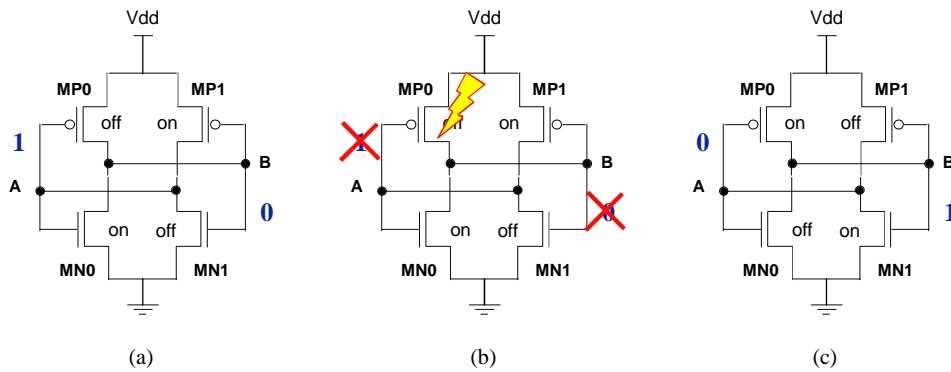


Figura 4 – Single event upset (SEU) em uma célula de memória SRAM

Quando uma partícula energética colide com uma região sensível de um circuito combinacional, um pulso transiente pode ser gerado. Neste caso, o fenômeno é denominado *Single-Event Transient* ou SET (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002; VIOLANTE, 2003). Se o SET se propagar até alguma saída primária, ele poderá ser capturado por um elemento de memória e assim, ser interpretado como um valor lógico correto, gerando então um *soft error*.

Existem três tipos de mascaramentos que podem impedir que o pulso gerado por um SET se propague pela lógica do circuito combinacional e atinja um elemento de memória: mascaramento lógico, mascaramento elétrico e mascaramento por *latching window* (KASTENSMIDT, 2003; WIRTH et al, 2005).

O mascaramento lógico está associado ao conceito de controlabilidade das portas lógicas (GOLDSTEIN, 1979; ABRAMOVICI; BREUER; FRIEDMAN, 1990; BUSHNELL; AGRAWAL, 2000). Um pulso não irá se propagar se estiver ocorrendo em uma das entradas de uma porta lógica que possua pelo menos uma outra entrada estável com seu valor controlante. O valor controlante de uma porta lógica é definido como sendo o valor que, quando aplicado a alguma das entradas da porta, determina o valor lógico na saída da porta, independentemente dos valores lógicos das demais entradas. A Fig. 5 ilustra o conceito de valor controlante para portas NAND.

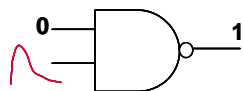


Figura 5 – NAND com uma entrada em valor controlante

O mascaramento elétrico ocorre quando um pulso é atenuado, ao ser propagado pelas portas lógicas, até ser filtrado. O atraso da porta e a capacitância a ela associada são os fatores que podem determinar o mascaramento elétrico. Com a diminuição das dimensões dos transistores, menor tende a ser o atraso das portas e ao mesmo tempo, maior tende a ser a duração dos pulsos transientes gerados por SETs. Logo, menor tende a ser o efeito do mascaramento elétrico. A Fig. 6 mostra um exemplo de mascaramento elétrico, onde o pulso transiente perde amplitude à medida que percorre o circuito. Neste exemplo, para uma melhor visualização do efeito da atenuação do nível de tensão, a lógica inversora das portas foi ignorada.

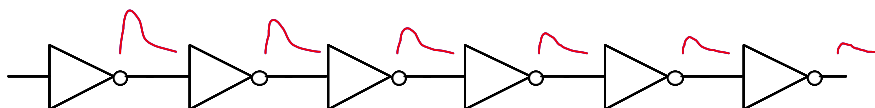


Figura 6 – Mascaramento Elétrico

O mascaramento por *latching window* (CHA et al., 1996; SHIVAKUMAR et al., 2002; ANGHEL; LEVEUGLE; VANHAUWAERT, 2005; WIRTH et al., 2005) ocorre quando o pulso atinge o elemento de memória fora da janela de amostragem, isto é, não existe transição de *clock* no momento que o pulso chega ao elemento de memória e, portanto, ele não é armazenado. Se t_{su} e t_{th} são os tempos de preparação (*setup*) e manutenção (*hold*) do registrador de saída e t é o instante em que ocorre a borda ativa do relógio, então o pulso será capturado caso tenha duração mínima entre $(t - t_{su})$ e $(t + t_{th})$. O pulso é mascarado caso termine antes de $(t - t_{su})$ ou inicie depois de $(t + t_{th})$. A Fig. 7 é mostra um exemplo de um pulso capturado e pulsos mascarados por terem

chegado às saídas primárias do circuito fora da janela de amostragem do registrador de saída.

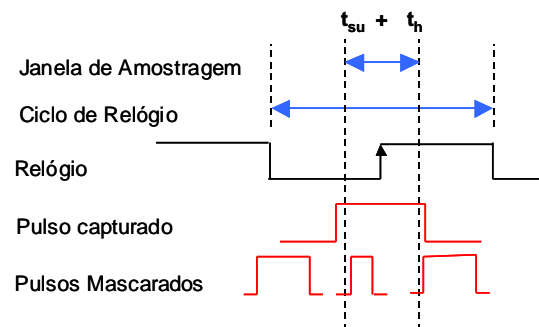


Figura 7 – Mascaramento por *Latching Window*

SEUs e SETs são classificadas como falhas transientes (*soft errors*), uma vez que não se originam de defeitos de fabricação e tampouco causam algum dano ao dispositivo onde ocorrem (BUSHNELL; AGRAWAL, 2000; HEIJMEN; NIEUWLAND, 2006).

Até pouco tempo atrás, SEUs e SETs se constituíam em problema relevante apenas para aplicações militares ou aeroespaciais, as quais precisam continuar funcionando mesmo em ambiente nos quais a radiação pode ser elevada. Porém, nos circuitos fabricados com tecnologias CMOS estado-da-arte a ocorrência de *soft errors* em memórias, causadas pela colisão de partículas energéticas, já está se tornando freqüente (NIEUWLAND; JASAREVIC; JERIN, 2006).

Embora os SEUs já tenham se tornado uma preocupação para o projeto de circuitos integrados em tecnologias VDSM – *Very Deep SubMicron* (abaixo de 100nm) há algum tempo, os SETs ainda não têm recebido grande atenção por apresentarem probabilidades menores de se converterem em *soft errors*.

Entretanto, Shivakumar e colaboradores (2002) já previram que, devido à contínua redução das dimensões dos dispositivos, em 2010 o SER (*logic SER*) para *soft errors* na lógica combinacional será igual ao SER em memórias não protegidas.

2.4 Propagação do SET

Caso um SET ocorra em uma porta que não esteja conectada diretamente a um elemento de memória, é necessário então avaliar se o pulso gerado irá ou não se propagar até o elemento de memória da saída do circuito. O circuito da Fig. 8 encontrava-se estável sob o vetor de entrada “101” no momento em que um SET ocorreu na porta NAND P_6 , gerando um pulso $1 \rightarrow 0 \rightarrow 1$ na saída desta porta.

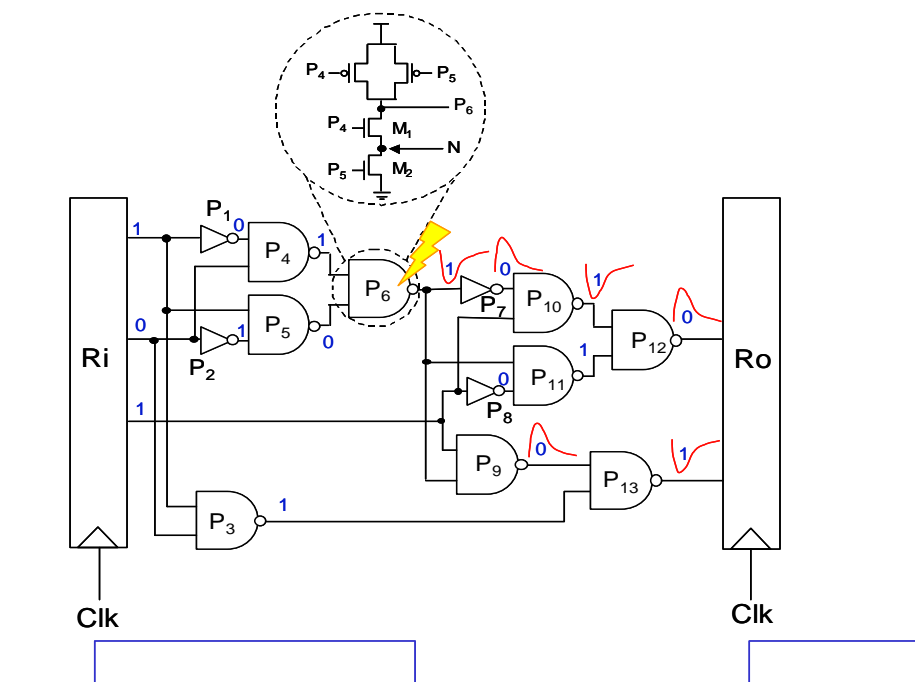


Figura 8 – *Single-event transient* (SET) e sua possível propagação em um bloco combinacional

A porta NAND P_6 tem um nó interno N , mostrado na Fig. 8. Para este nó ser sensível à colisão de uma partícula energética, o transistor M_2 deve estar em estado *off*, e para propagar o pulso gerado em N para a saída da porta P_6 , o transistor M_1 deve estar em estado *on*. Esta condição pode ser satisfeita apenas quando as saídas das portas P_4 e P_5 forem 1 e 0, respectivamente, como ilustra a Fig. 8.

Para saber a probabilidade do pulso gerado em N se propagar para a saída de P_6 é necessário conhecer todos os vetores de entrada que farão com que N seja sensível à colisão de uma partícula energética. Neste caso, os vetores “100” e “101” são os únicos vetores que tornam o nó N sensível e o pulso gerado neste pode propagar-se para a saída da porta P_6 (GILL, 2005) e através do circuito.

Considerando-se apenas os valores lógicos provocados pelo vetor de entrada “101”, o pulso irá propagar-se pelos caminhos $\{P_7, P_{10}, P_{12}\}$ e $\{P_9, P_{13}\}$. O pulso não irá se propagar pela porta P_{11} devido ao “0” presente na saída do inversor P_8 , fazendo com que o pulso seja mascarado logicamente. Apesar disto, neste exemplo, o pulso gerado por um SET em P_6 propagou-se por mais de um caminho, atingindo mais de uma saída primária do bloco combinacional. Examinando a Fig. 8 pode-se inferir que a probabilidade de um pulso se propagar até alguma saída primária tende a ser inversamente proporcional ao número médio de níveis lógicos entre o ponto em que o pulso foi gerado e as saídas primárias. Ou seja, quanto maior o número de portas lógicas que o pulso precisa atravessar, maior é a probabilidade de ele ser completamente mascarado lógica ou eletricamente.

Assim, a análise da propagação de SETs em circuitos combinacionais é útil para identificação dos pontos mais sensíveis, permitindo identificar quais partes precisam ser protegida contra radiação. Os métodos existentes para a análise da suscetibilidade de circuitos combinacionais a SETs podem ser classificados em métodos baseados em simulação e métodos baseados em propagação topológica. Porém, o estudo de tais métodos foge ao escopo deste trabalho.

3 FPGA

Dispositivos lógicos podem ser classificados em duas categorias: fixos e programáveis. Circuitos lógicos fixos são permanentes, isto é, eles executam apenas uma função ou um conjunto de funções. Uma vez construídos, eles não podem ser alterados. Já os dispositivos programáveis (*Programmable Logic Devices* - PLDs) podem ser modificados a qualquer momento para executar qualquer número de funções (DESCHAMPS; BIOUL; SUTTER, 2006).

A vantagem de utilizar PLDs é que, durante a fase do projeto, o projetista pode alterar o circuito muitas vezes até este operar satisfatoriamente. A programabilidade dos PLDs decorre do uso de memórias tipo SRAM: para modificar um projeto, o conteúdo destas memórias internas ao dispositivo precisa apenas ser reescrito. Reusabilidade é uma das vantagens adicionais dos PLDs (DESCHAMPS; BIOUL; SUTTER, 2006).

Muitos tipos de dispositivos lógicos programáveis estão disponíveis atualmente. A escala de produtos no mercado inclui desde pequenos dispositivos capazes de executar algumas equações lógicas até dispositivos maiores com capacidade para implementar um processador inteiro e mais alguns periféricos. Além da diversidade de tamanhos, numerosas arquiteturas alternativas também são oferecidas ao projetista. Os dois tipos de dispositivos programáveis mais importantes são: *Complex Programmable Logic Devices* (CPLD) e *Field Programmable Gate Arrays* (FPGA) (DESCHAMPS; BIOUL; SUTTER, 2006). Este trabalho assume o uso de FPGAs.

3.1 Estrutura Genérica de um FPGA

O FPGA é um *chip* que suporta a implementação de circuitos lógicos relativamente grandes e com certa complexidade em função do uso de tecnologia CMOS estado-da-arte. Este dispositivo não possui matrizes de portas OR ou AND (BROWN; VRANESIC, 2000). Ao invés disso, ele consiste de uma grande matriz de células lógicas configuráveis ou blocos lógicos (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000), contidos em um único circuito integrado, que podem ser utilizados para a implementação de funções lógicas definidas pelo usuário.

Cada célula contém capacidade computacional para implementar funções lógicas e realizar roteamento para comunicação entre elas. A arquitetura de um FPGA é constituída de três estruturas básicas: blocos lógicos, blocos de entrada e saída, e interconexões programáveis (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000). A Fig 9 mostra a estrutura básica de um FPGA.

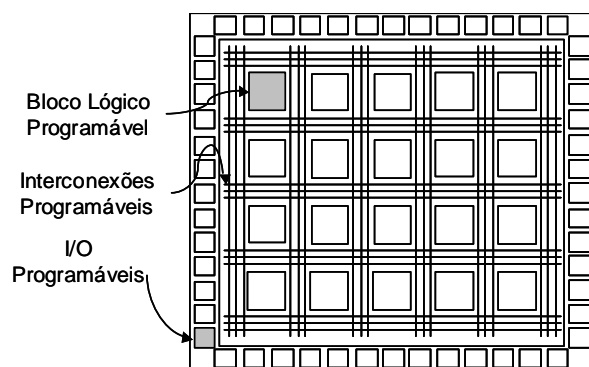


Figura 9 – Estrutura básica de um FPGA

- LB (*Logic Blocks*): Circuitos idênticos programáveis, constituídos de *flip-flops* e lógica combinacional.
- IOB (*Input/Output Blocks*): Circuitos responsáveis pelo interfaceamento das saídas dos LBs com os pinos bidirecionais de entrada e saída do encapsulamento do FPGA. São basicamente *buffers*, que funcionarão como um pino bidirecional entrada/saída do FPGA.

- Chaves Programáveis (*Programmable Switches*): Os recursos de interconexão possuem trilhas para conectar as entradas e saídas dos LBs e IOBs. O processo de escolha das interconexões é chamado de roteamento. Atualmente, o número de interconexões programáveis em um FPGA varia em cada *chip* disponível comercialmente (BROWN; VRANESIC, 2000).

Os IOBs formam uma borda ao redor do dispositivo; cada um destes fornece entrada, saída ou acesso bi-direcional aos pinos de I/O de uso geral, disponíveis na parte exterior do componente. Dentro desta borda de IOB existe uma região retangular onde estão os LBs.

Os LBs formam uma matriz bidimensional e as interconexões programáveis são organizadas como canais de roteamento horizontais e verticais entre as linhas e colunas dos LBs (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000). Os canais de roteamento possuem chaves de interligação programáveis que permitem conectar os LBs de maneira conveniente, em função das necessidades de cada projeto (BROWN; VRANESIC, 2000).

Atualmente, existe no mercado uma grande variedade de famílias de FPGAs de diversos fabricantes. Alguns exemplos dos fabricantes mais importantes são Altera, Actel, Xilinx, Lucent etc, sendo que estes oferecem famílias de dispositivos com capacidades de programação distintas, conjuntos de características específicas, tais como desempenho, consumo de energia, testabilidade, diferentes meios físicos para implementar a programação do usuário, bem como diferentes arranjos das interconexões e funcionalidades básicas distintas dos LBs (DESCHAMPS; BIOUL; SUTTER, 2006). Os FPGAs atuais podem ser usados para implementar circuitos lógicos com tamanho equivalente a mais do que um milhão de portas lógicas. Os FPGAs podem ser programados através da utilização de uma das seguintes tecnologias:

Baseados em SRAM (Xilinx e Altera): As conexões do FPGA são construídas utilizando transistor de passagem, ou *transmission gates*, ou multiplexadores, que são controlados por células SRAM (DESCHAMPS; BIOUL; SUTTER, 2006). Devido à volatilidade dessas memórias, os FPGAs que utilizam esta tecnologia precisam de uma

memória externa não-volátil para carregar a configuração do *chip*. Devido a isto, esta tecnologia ocupa muito espaço no circuito integrado, entretanto é rapidamente reprogramável e o FPGA pode ser programado um número ilimitado de vezes (DESCHAMPS; BIOUL; SUTTER, 2006).

A Fig. 10 mostra as conexões de um FPGA controlado por células de memória SRAM.

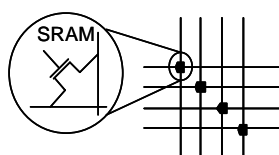


Figura 10 – Conexões SRAM

Anti-fusível (Actel e Quicklogic): essa tecnologia baseia-se num dispositivo de dois terminais, que no estado não programado apresenta uma alta impedância (circuito aberto). Aplicando-se uma tensão, por exemplo, entre 11 e 20 Vcc, o dispositivo forma um caminho de baixa impedância entre seus terminais (Fig 11). Os componentes FPGA que usam tecnologia anti-fusível não podem ser reprogramados, porém, apresentam um custo menor em relação aos FPGAs baseados em SRAM (DESCHAMPS; BIOUL; SUTTER, 2006).

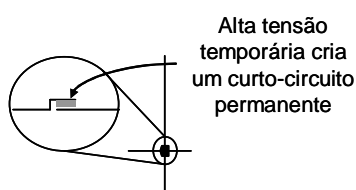


Figura 11 – Anti-fusível

EEPROM/EPROM (vários fabricantes): esta tecnologia é baseada no método de criação das memórias EPROM (*Erasable Programmable Read Only Memory*) e EEPROM (*Electrically Programmable Read Only Memory*). Sua principal vantagem é

que a configuração é armazenada dentro do dispositivo, isto é, não é necessária a utilização de memória externa (DESCHAMPS; BIOUL; SUTTER, 2006).

No interior de cada LB do FPGA existem vários modos possíveis para implementação de funções lógicas. O mais utilizado pelos fabricantes de FPGA como, por exemplo, as empresas Altera e Xilinx, é projetado com uma LUT (*Look-Up Table*) implementadas com células de memórias, ou com multiplexadores e células de memórias. A Fig. 12 mostra a implementação com LUTs (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000).

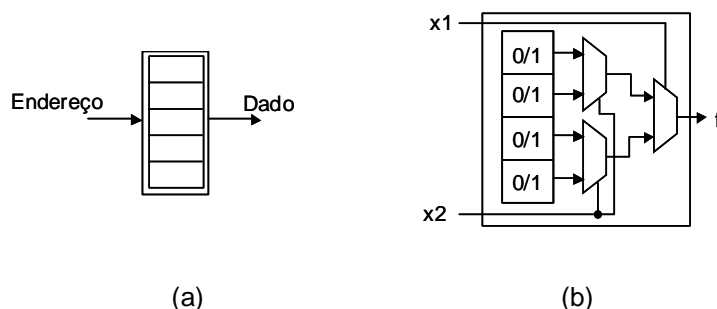


Figura 12 – LUT implementada com células de memórias (a) e com multiplexadores e células de memórias (b)

LUTs de vários tamanhos são encontradas nos FPGAs disponíveis no mercado, sendo o tamanho destas definido pelo número de entradas. Cada LB em um FPGA típico tem um pequeno número de entradas, mas apenas uma saída (BROWN; VRANESIC, 2000). A Fig. 12b mostra a estrutura de uma LUT de duas entradas, $x1$ e $x2$, e uma saída, f . Ela é capaz de implementar qualquer função lógica de duas variáveis.

Uma função lógica pode ser implementada em uma LUT diretamente a partir de sua tabela-verdade. Cada posição da memória armazena o valor da função que corresponde a uma combinação de entrada (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000). Como a tabela-verdade de duas variáveis tem quatro linhas (i.e., quatro possíveis mintermos), a LUT mostrada na Fig. 12b tem quatro células de memória, uma célula para armazenar cada um dos quatro possíveis valores de saída.

As variáveis de entrada x_1 e x_2 são usadas como entradas de controle dos multiplexadores, os quais, dependendo dos valores de x_1 e x_2 , selecionam o conteúdo de uma das quatro células de memória como a saída da LUT (BROWN; VRANESIC, 2000). O resultado é uma porta lógica "universal" de duas entradas (DESCHAMPS; BIOUL; SUTTER, 2006).

Nos FPGAs disponíveis comercialmente as LUTs possuem geralmente quatro ou cinco entradas, o que permite endereçar, respectivamente, 16 ou 32 células de memória (BROWN; VRANESIC, 2000). Quando um circuito lógico é implementado em um FPGA, os LBs são programados para realizar as funções necessárias, e os canais de roteamento são estruturados de forma a realizar a interconexão necessária entre os LBs (BROWN; VRANESIC, 2000).

Na Fig. 12a, uma LUT é implementada com uma memória $2^n \times 1$ (DESCHAMPS; BIOUL; SUTTER, 2006). O endereço da entrada seleciona uma das 2^n posições de memória. As posições de memória são carregadas normalmente com os valores do *bit-stream* (são os dados binários que deverão ser carregados no FPGA para fazer com que o *chip* execute um projeto em particular) da configuração do usuário.

A Fig 13 mostra o exemplo de uma LUT com quatro entradas implementada com células de memórias e multiplexador, juntamente com o conteúdo da tabela-verdade que a LUT implementa (DESCHAMPS; BIOUL; SUTTER, 2006).

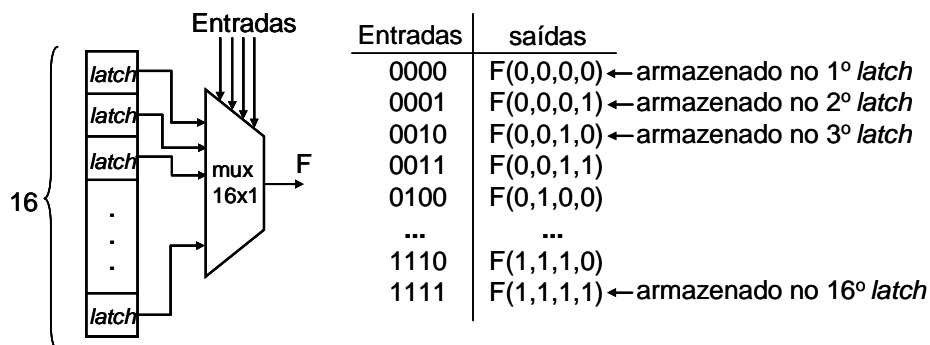


Figura 13 – LUT de quatro entradas e Tabela-verdade

Os FPGAs usualmente apresentam circuitos extras, além da LUT, em cada bloco lógico. A Fig 14 mostra como um *flip-flop* pode ser incluído em um bloco lógico de um FPGA (DESCHAMPS; BIOUL; SUTTER, 2006) (BROWN; VRANESIC, 2000).

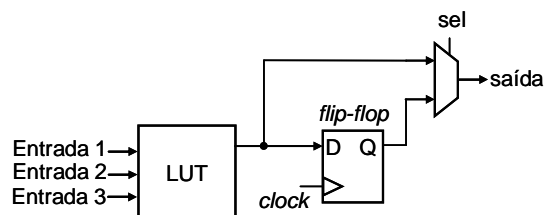


Figura 14 – Inclusão de um *flip-flop* em um bloco básico de um FPGA

O *flip-flop* pode ser usado para armazenar o valor de sua entrada D sob o controle da sua entrada de *clock*.

Conforme já mencionado, as células de memória das LUTs de um FPGA baseados em SRAM são voláteis, o que implica a perda do conteúdo armazenado, no caso de falta de suprimento de energia elétrica. Dessa forma, o FPGA deve ser reprogramado toda vez que for ligado. Geralmente, utiliza-se uma pequena memória externa cuja função é carregar automaticamente as células de memória, toda vez que o FPGA for ligado (BROWN; VRANESIC, 2000).

Os FPGAs podem ser utilizados para a implementação de praticamente qualquer projeto de *hardware*. Um dos usos mais comuns é a utilização do FPGA para a prototipação de componentes que poderão, no futuro, ser implementados por meio de configuração por meio de todas as máscaras (DESCHAMPS; BIOUL; SUTTER, 2006). No entanto, os FPGAs estão cada vez mais sendo utilizados como produto final. Esta decisão requer uma análise do desempenho desejado, do desenvolvimento, do custo de produção etc.

A característica da reconfigurabilidade dos FPGAs baseados em SRAM faz com que estes dispositivos sejam uma ótima opção para aplicações espaciais, onde correções nos projetos podem ser necessárias durante a execução de uma missão. Porém, por estarem fora da atmosfera terrestre, estes dispositivos ficam expostos à

ação intensa de partículas energéticas. Em função disto, alguns fabricantes, como a Xilinx e a Actel, desenvolveram famílias de FPGAs que utilizam algumas técnicas de proteção nas células de memória para serem utilizadas em aplicações espaciais e militares. Porém, em função do acréscimo de *hardware* e da pequena demanda, o preço por peça destes FPGAs é muito elevado, podendo chegar a U\$ 1.000,00 (BERG, 2006).

Conforme mencionado anteriormente, as famílias mais recentes de FPGAs fazem uso de tecnologias CMOS estado-da-arte, e por isto, são suscetíveis a falhas transientes mesmo operando na superfície terrestre. Com isso torna-se necessário projetar circuitos tolerantes a SEUs e a SETs também para sistemas em FPGAs operando ao nível do mar.

Logo, como os FPGAs baseados em SRAM são os dispositivos programáveis mais usados na atualidade, é natural que sejam o foco de interesse, no que se refere à proteção contra falhas transientes.

A colisão de partícula energética em parte sensível de um FPGA irá causar efeitos específicos, os quais dependem do tipo de recurso atingido, conforme explicado no capítulo 2. Se a partícula colide com algum circuito combinacional da matriz do FPGA, o efeito será a geração de um SET que poderá se propagar e ser capturado por um elemento de memória, causando uma inversão de dado (*bit flip*). Se a partícula colide com algum elemento de memória disponível para o usuário, um SEU poderá ser gerado, causando também um *bit flip*.

Porém, se a partícula colidir com uma célula SRAM, invertendo seu conteúdo, a programação do FPGA será afetada. O resultado disto é que a funcionalidade do circuito será alterada, o que se constitui em um erro grave. Para a correção deste erro é necessária a utilização de algumas técnicas de restauração da programação ou então que a própria matriz FPGA utilize células SRAM protegidas contra SEUs, conforme já mencionado.

No entanto, as técnicas de restauração da programação não são capazes de corrigir os erros provocados por SEUs nas memórias disponíveis para o usuário. Como a quantidade de memória disponíveis para o usuário nos FPGAs é significativa e como os FPGAs mais novos são fabricados em tecnologias CMOS, as chances destes serem

atingidos por partículas energéticas estão aumentando, de modo que o uso de técnicas de projeto tolerante a falhas transientes precisará, cada vez mais, ser aplicadas ao projeto de sistemas a serem implementados com FPGAs.

3.2 Estrutura dos Dispositivos da Família Stratix II da Altera

Os dispositivos da família Stratix II da Altera (2007) são baseados em tecnologia de 1,2 V e 90nm, possuindo SRAM com camadas de cobre, caracterizando assim uma nova estrutura que maximiza o desempenho e permite dispositivos com densidades de aproximadamente 180.000 elementos lógicos equivalentes (LEs) (ALTERA, 2007).

A família Stratix II contém uma arquitetura bidimensional baseada em linhas e colunas para implementar a lógica do usuário, como mostrado anteriormente na estrutura genérica de um FPGA. Uma série de colunas e linhas de interconexões de vários comprimentos e velocidades provêm sinais de interconexão entre os *arrays* de blocos lógicos (*logic array blocks* - LABs), estruturas dos blocos de memórias (blocos M512 RAM, M4K RAM e M-RAM) e blocos DSP (*Digital Signal Processing*) (ALTERA, 2007).

Um ALM (*Adaptive Logic Module*) é o bloco básico da família Stratix II (equivalente ao LB explicado anteriormente na estrutura genérica de um FPGA), ou seja, é o bloco que implementa as funções lógicas dos usuários. Cada ALM contém uma variedade de recursos baseados em LUT que podem ser divididos entre duas LUTs adaptativas (ALUTs). A ALUT é a célula usada no *software* Quartus II para a síntese lógica. Com até oito entradas das duas ALUTs, uma ALM pode implementar várias combinações de duas funções (ALTERA, 2007).

Além dos recursos baseados em ALUTs, cada ALM contém dois registradores programáveis, dois somadores completos dedicados, uma cadeia de *carry*, uma cadeia aritmética compartilhada e uma cadeia de registradores, podendo com isso implementar várias funções aritméticas e registradores de deslocamento. Cada ALM utiliza todo o tipo de interconexão: local, linha, coluna, cadeia de *carry*, cadeia aritmética compartilhada, cadeia de registradores e interconexões diretas (ALTERA, 2007).

A Fig. 15 apresenta uma visão geral do dispositivo Stratix II da Altera (2007).

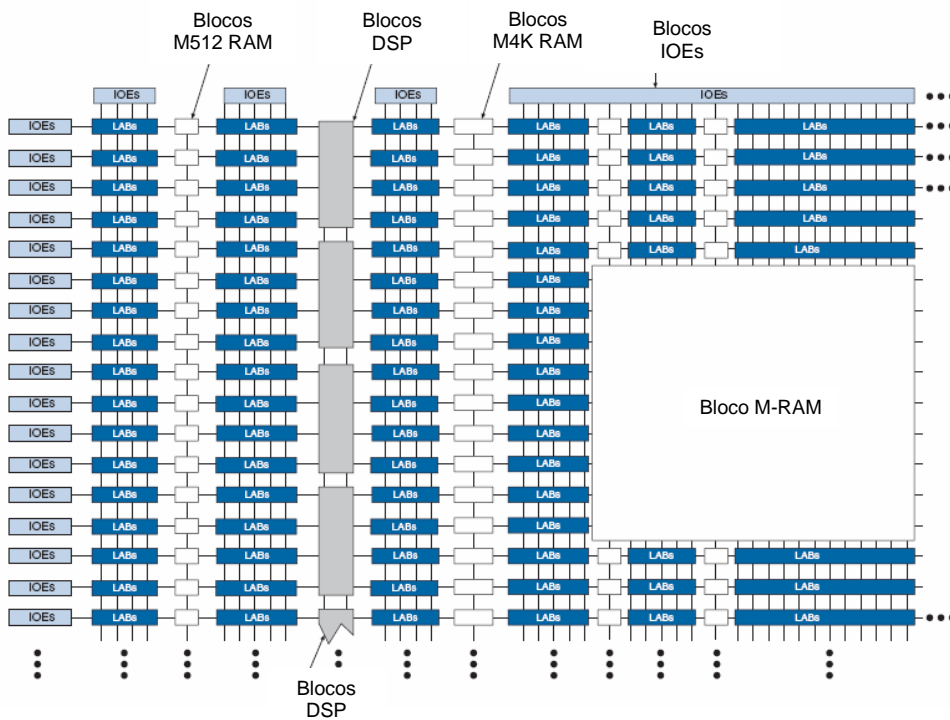


Figura 15 – Diagrama de bloco de Stratix II
 Fonte: Altera, 2007.

Cada LAB é constituído de oito ALMs, cadeias de *carry*, cadeias aritméticas compartilhadas, sinais de controle do LAB, interconexão local e linhas de conexões de cadeias de registradores. A interconexão local transfere sinais entre as ALMs na mesma LAB. As conexões das cadeias de registradores transferem a saída de um registrador ALM para o registrador ALM adjacente em um LAB. LABs são agrupadas em linhas e colunas no dispositivo (ALTERA, 2007).

A Fig. 16 mostra o diagrama em blocos de alto nível de um ALM da família Stratix II.

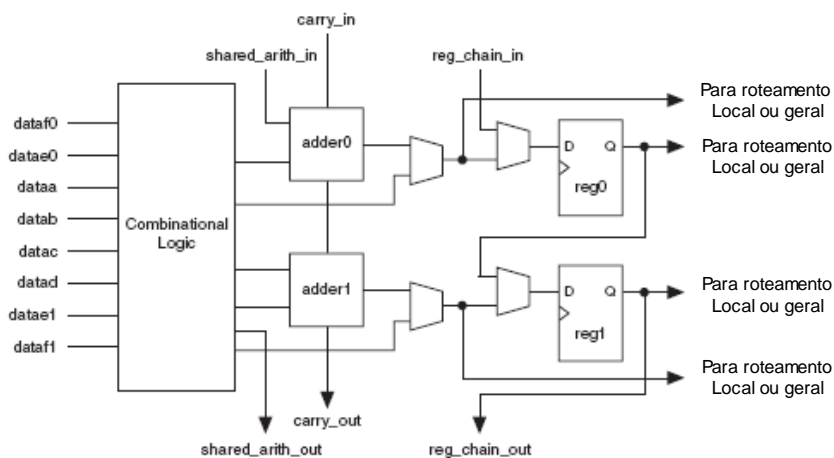


Figura 16 – Diagrama de bloco de alto nível do ALM Stratix II
 Fonte: Altera, 2007.

O compilador do Quartus II posiciona a lógica associada em um LAB ou em um LAB adjacente, permitindo o uso de cadeias aritméticas locais e compartilhadas, além das conexões de cadeias de registradores, para eficiência no desempenho e no uso de área (ALTERA, 2007). A Fig. 17 mostra a estrutura de um LAB do Stratix II.

Os blocos DSP providenciam implementação dedicada de multiplicadores, funções de multiplicação acumulada e também contêm registradores de deslocamento para aplicações de processamento digital de sinais - DSP (*digital signal processing*). Os blocos DSP são agrupados em colunas através do dispositivo e operam em até 450 MHz.

Cada pino de I/O do dispositivo Stratix II é alimentado por um elemento de entrada/saída (IOE) localizado na extremidade das linhas e colunas de LABs em torno do dispositivo. Cada IOE contém *buffers* bidirecionais e seis registradores para armazenar entradas, saídas e sinais de ativação de saída. Quando usados com *clock* dedicado, esses registradores oferecem suporte de interface com dispositivos de memória externa, tais como dispositivos SDRAM DDR e DDR2, RLDRAM II, e SRAM QDR II (ALTERA, 2007). Os IOEs equivalem aos IOBs mencionados anteriormente.

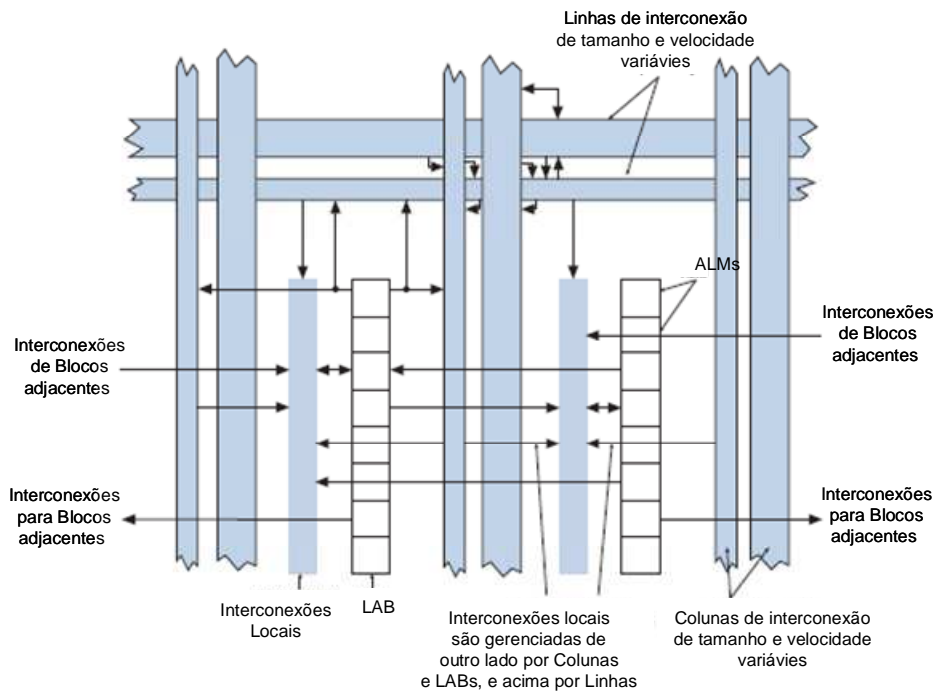


Figura 17 – Estrutura do LAB de um Stratix II

Fonte: Altera, 2007.

Os dispositivos da família Stratix II possuem memória TriMatrix que consiste de três tamanhos de blocos de memórias RAM (M512 RAM, M4K RAM e M-RAM) para implementar memórias de duas portas e *buffers* com fila (*first-in first-out* - FIFO), sendo que o número de blocos M512 RAM, M4K RAM, e DSP varia de dispositivo pra dispositivo, juntamente com o número de linhas e colunas e blocos de M-RAM (ALTERA, 2007).

A Tab. 1 mostra as características gerais dos FPGAs da Altera pertencentes à família Stratix II, lembrando que um ALM contém duas ALUTs, e os multiplicadores são implementados utilizando os blocos DSP.

O número de LEs equivalentes mostrado na Tab. 1 refere-se a arquiteturas baseadas em LUTs de quatro entradas.

Tabela 1 – Características gerais da família Stratix II

CARACTERISTICAS	EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180
ALMs	6,240	13,552	24,176	36,384	53,016	71,760
LUT Adaptativa	12,480	27,104	48,352	72,768	106,032	143,520
LEs Equivalentes	15,600	33,880	60,440	90,960	132,540	179,400
Blocos M512 RAM	104	202	329	488	699	930
Blocos M4K RAM	78	144	255	408	609	768
Blocos M-RAM	0	1	2	4	6	9
Total de bits RAM	419,328	1,369,728	2,544,192	4,520,488	6,747,840	9,383,040
Blocos DSP	12	16	36	48	63	96
Multiplicador 18x18 bits	48	64	144	192	252	384
Pinos I/O	366	500	718	902	1,126	1,170

Fonte: Altera, 2007.

3.3 Fluxograma Genérico do Projeto FPGA

A Fig. 18 mostra as sucessivas fases do fluxograma do projeto FPGA que serão explicadas a seguir (DESCHAMPS; BIOUL; SUTTER, 2006):

- Entrada do projeto: criação de um arquivo do projeto usando um editor esquemático ou uma linguagem de descrição de *Hardware* (Verilog, VHDL, Abel).
- Síntese do projeto: um processo que começa em alto nível de abstração da lógica, utilizando-se alguma linguagem de descrição de *hardware* (e.g., Verilog ou VHDL) (IEEE, 1994) e automaticamente cria um nível mais baixo de abstração da lógica usando uma biblioteca de primitivas.
- Divisão (ou Mapeamento): um processo que atribui a cada elemento da lógica um elemento físico específico que execute verdadeiramente a função lógica no dispositivo configurável.
- Posicionamento: mapeia a lógica em posições específicas no *chip* FPGA-alvo
- Roteamento: escolha das rotas para conectar a lógica mapeada.

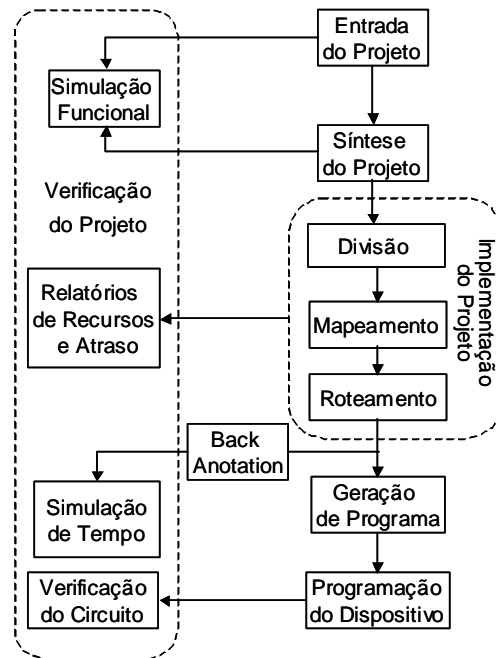


Figura 18 – Fluxograma genérico de um projeto FPGA

- Geração de programa: um arquivo *bit-stream* é gerado para programar o dispositivo.
- Programação do dispositivo: o *bit-stream* é carregado no FPGA.
- Verificação do projeto: a simulação é usada para verificar funcionalidades. A simulação pode ser concluída em níveis diferentes. A simulação funcional ou comportamental não considera o componente ou o atraso das interconexões. A simulação de tempo é usada para extrair a informação de atraso do circuito.

Outros relatórios são gerados para verificar outros resultados da implementação, tais como a frequência e atraso máximos e a utilização de recursos.

A Divisão (ou mapeamento), Posicionamento, e Processos de Roteamento são gerados geralmente durante a implementação do projeto (DESCHAMPS; BIOUL; SUTTER, 2006).

4 Somadores

A implementação de somadores de alto desempenho é essencial não apenas para adição, mas também para a subtração, multiplicação e divisão. Assim, a adição é essencial para qualquer projeto que envolva operações aritméticas e por isso, o desempenho destes operadores é fator crítico para um bom desempenho dos sistemas digitais.

Fatores como uso de recursos, frequência de operação e consumo de energia, devem ser previamente analisados para que os somadores projetados satisfaçam às restrições impostas ao projeto. Diversas soluções para aumentar o desempenho dos somadores mediante a aceleração da propagação do *carry* têm sido propostas, destacando-se entre elas o *Carry Lookahead*, o *Carry Select*, o *Carry Skip*, etc (HWANG, 1979; OKLOBDZIJA, 2001).

Estas soluções envolvem transformações na arquitetura do somador, onde as equações lógicas são manipuladas, objetivando a redução do atraso de propagação do *carry*. Porém, apesar de existirem muitos tipos de somadores rápidos, este trabalho terá como foco o somador *Carry Lookahead*.

4.1 Meio-Somador

Observando o exemplo da soma de quatro bits $A = 1011$ e $B = 1110$, cujo resultado é $S = 1001$ e *carries* = 1111, percebe-se que os bits do resultado são gerados da direita para a esquerda, ou seja, do bit menos significativo para o mais significativo. Portanto, para a geração do bit menos significativo do resultado, S_0 , apenas interessam os bits menos significativos de cada um dos operandos, A_0 e B_0 .

Neste caso, é possível avaliar todas as combinações possíveis dos valores de entrada com os correspondentes valores para o resultado, S_0 , e *carry*, C_0 , como pode ser visto na Tab. 2.

Tabela 2 – Resultado da Soma e *Carry* para a soma do bit menos significativo

A_0	B_0	C_0	S_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Através da Tab. 2 é possível concluir que o bit menos significativo da soma pode ser obtido pela função XOR dos bits menos significativos dos operandos de entrada, enquanto que o bit de *carry* menos significativo é uma AND dos dois operandos de entrada. A Fig. 19 mostra o circuito lógico que implementa estas funções, o qual é normalmente denominado de Meio-Somador (*Half-Adder*).

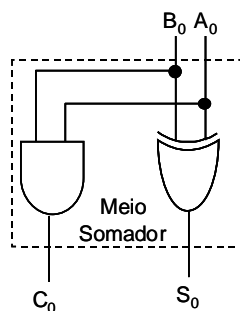


Figura 19 – Circuito Lógico de um Meio-Somador.

Note-se, entretanto, que o Meio-Somador não admite um *carry* de entrada. Logo, ele só pode ser usado para a soma dos bits menos significativo dos operandos. Um Meio-Somador pode ser representado como mostra a Fig. 20.

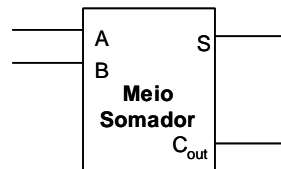


Figura 20 – Símbolo para um Meio-Somador

4.2 Somador-Completo

Para o cálculo dos demais bits da soma (S_{n-1}, \dots, S_1), além de considerar os respectivos bits dos operandos A_i e B_i , é necessário considerar também o *carry* gerado no estágio anterior, C_{i-1} , ou seja, o *carry* gerado na soma dos operandos A_{i-1} e B_{i-1} . Para isso utiliza-se um circuito Somador-Completo (*Full-Adder*). A tabela-verdade para a soma dos demais bits dos operandos de entrada é representada na Tab. 3, onde A_i , B_i e C_{i-1} são as entradas de i -ésimo estágio do Somador-Completo, e S_i e C_i são as saídas da soma e *carry* para o i -ésimo estágio, respectivamente.

Tabela 3 – Resultado da soma e *carry* levando-se em consideração o *carry* do estágio anterior

A_i	B_i	C_{i-1}	C_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Através da tabela-verdade acima e de simplificações booleanas é possível extrair as seguintes equações lógicas:

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1} \quad (2)$$

$$S_i = A_i \oplus B_i \oplus C_{i-1} \quad (3)$$

O circuito lógico que implementa as equações para C_i e S_i acima é mostrado na Fig. 21(OKLOBDZIJA, 2001).

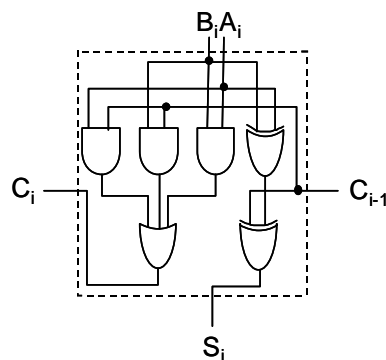


Figura 21 – Circuito lógico de um Somador-Completo.

Para esta implementação, o atraso entre A_i ou B_i , até S_i é o atraso de 2 portas Xor e o atraso de C_{i-1} até C_i é o atraso de uma porta Xor, uma porta AND e uma porta OR.

De um modo geral, em um Somador-Completo, ao bit de *carry* de entrada é dado o nome de C_{in} e ao bit de saída C_{out} . Um Somador-Completo pode ser representado como ilustrado na Fig. 22.

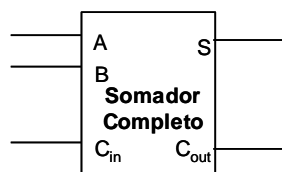


Figura 22 – Símbolo para um Somador-Completo de um bit

4.3 Ripple Carry Adder

O somador *Ripple Carry* (RCA) de n bits pode ser facilmente construído a partir de um circuito Meio-Somador, utilizado para a adição dos bits menos significativos, e de $n-1$ circuitos Somadores-Completos para os bits restantes, ou até mesmo utilizando n Somadores Completos (OKLOBDZIJA, 2001). Para isso, basta conectar a saída de *carry* de um dado bit à entrada do *carry* do somador do estágio subsequente (OKLOBDZIJA, 2001). A Fig. 23 mostra um somador *Ripple Carry* de n bits implementado a partir de um Meio-Somador e de $n-1$ Somadores-Completos.

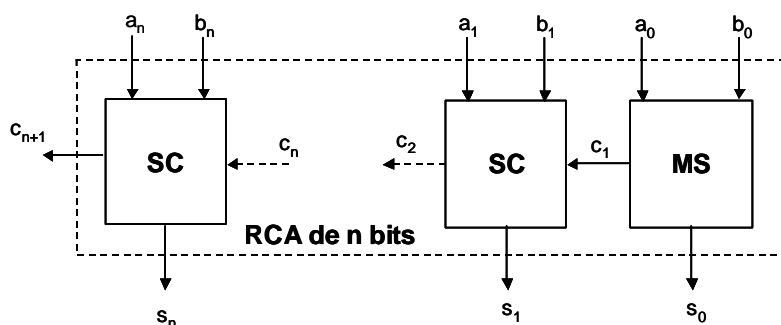


Figura 23 – Somador *Ripple Carry* de n bits sem *carry* de entrada

O somador mostrado na Fig. 23 é chamado de *Ripple Carry* devido ao fato do sinal de *carry* se propagar desde o bit menos significativo até o mais significativo, passando por cada um dos estágios. Com esta estrutura é possível, teoricamente, construir um somador com qualquer número de bits, bastando para isso interligar tantos Somadores-Completos quantos forem necessários. O bit de *carry* do Somador-Completo mais significativo pode ser interpretado como bit mais significativo do resultado ou como excedente da soma (*overflow*), conforme o tipo de aplicação.

A Fig. 24 mostra o mesmo somador de n bits, com a diferença que a adição dos bits menos significativos utiliza um Somador-Completo. Esta alteração acrescenta ao somador um bit de *carry* de entrada, C_{in} . Com isso é possível interligar somadores *Ripple Carry* de n bits para se obter somadores de mais alta ordem através do mesmo tipo de interligação que foi utilizado para os Somadores Completos de um bit.

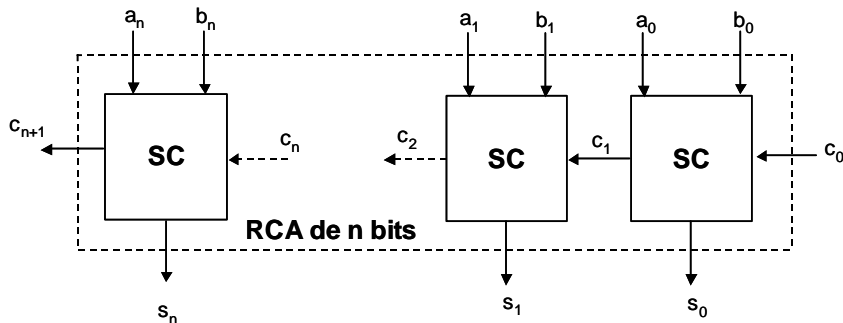


Figura 24 – Somador *Ripple Carry* de n bits com *carry* de entrada

A Fig. 25 mostra um exemplo de um somador de 8 bits construído pela interligação de 2 blocos somadores de 4 bits como o da Fig. 24. A saída C_{out} do somador menos significativo (bits 0 a 3) é conectada ao bit de *carry* de entrada, C_{in} do somador de 4 bits mais significativo, (bits 4 a 7). Por outro lado, a saída C_{out} do somador de 4 bits mais significativo serve como o bit mais significativo do resultado, S_8 .

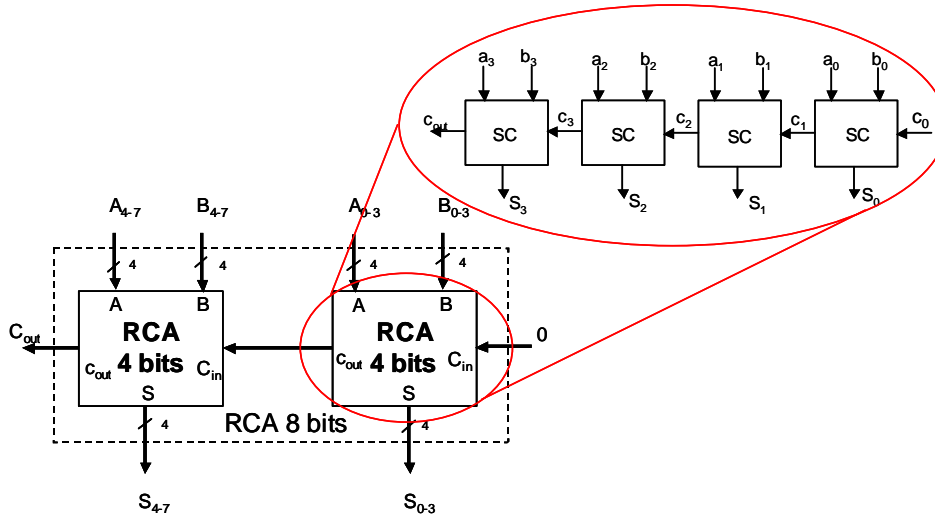


Figura 25 – Somador *Ripple Carry* de 8 bits construído a partir de dois blocos somadores de 4 bits

Como pode ser visto na Fig. 25, ao bit de *carry* de entrada do primeiro somador é atribuído de maneira fixa o valor lógico 0.

Os somadores RCA são os mais simples de serem construídos e estão entre os mais econômicos, em termos de *hardware*. No entanto, são também os mais lentos, pois para cada estágio da soma é necessário utilizar o *carry* do estágio anterior, o que gera um grande atraso devido à propagação de *carry* por todos os estágios.

Logo, o caminho crítico do circuito é definido pela propagação do sinal de *carry*, ou seja, pela soma dos atrasos de todas as portas entre o bit menos significativo de um dos operandos (A_0 ou B_0) e o bit mais significativo da saída, S_n (OKLOBDZIJA, 2001) ou o *carry* de saída C_{out} . No RCA corresponderá à soma do atraso de n portas AND mais n portas OR (para a propagação do bit de *carry* pelos n somadores completos), o que resultará em um atraso de $2n$ (OKLOBDZIJA, 2001) ou $2n+1$ (no caso da utilização de uma porta XOR para a verificação de *overflow*). Portanto, para operandos com um número de bits elevado, por exemplo, 32 bits, a utilização de um RCA poderá ser inaceitável para muitas aplicações que tenham o desempenho como restrição.

A Fig. 26 mostra o caminho crítico de um somador RCA de 4 bits.

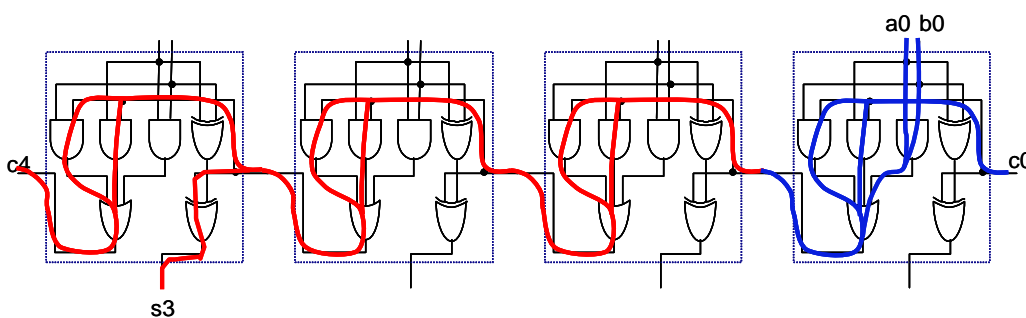


Figura 26 – Caminho crítico de um somador RCA de 4 bits

4.4 Carry Lookahead Adder

O somador *Carry Lookahead* (CLA) tem como objetivo a geração rápida do *carry*, isto é, visa o aumento da velocidade de propagação do *carry* (HWANG, 1979; OKLOBDZIJA, 2001) em relação aos somadores RCA. Logo, para isso ser possível, o somador *Carry Lookahead* utiliza uma técnica para acelerar a propagação do *carry* que se baseia em consultar todos os estágios de entrada do somador e, simultaneamente,

gerar os *carries* referentes a cada um destes estágios. Em outras palavras, todos os *carries* são calculados ao mesmo tempo, em paralelo (HWANG, 1979; OKLOBDZIJA, 2001).

Estes *carries* que entram em todos os estágios do somador em paralelo são gerados simultaneamente por um circuito lógico adicional (HWANG, 1979). Isso resulta em um tempo de adição constante e independente do tamanho dos operandos de entrada do somador (HWANG, 1979; OKLOBDZIJA, 2001).

Para possibilitar a geração simultânea dos *carries* de cada estágio da soma são utilizados dois sinais auxiliares: *Carry Generate* e *Carry Propagate*, definidos como (HWANG, 1979):

$$g_i = A_i \cdot B_i \quad (4)$$

$$p_i = A_i \oplus B_i \quad (5)$$

onde: A_i e B_i são os operandos de entrada do i -ésimo estágio do somador *Carry Lookahead*

O sinal *Carry Generate* indica, quando verdadeiro, que foi gerado um *carry* no i -ésimo estágio e, portanto, o *carry* de saída do estágio anterior, C_{i-1} , não precisará ser considerado. O sinal *Carry Propagate* indica, quando verdadeiro, que o i -ésimo estágio irá propagar para sua saída o seu *carry* de entrada, C_{i-1} , independente do valor que este possua (HWANG, 1979; OKLOBDZIJA, 2001).

A Fig. 27 mostra o circuito que implementa os sinais auxiliares do somador CLA e o circuito que implementa a soma propriamente dita, ambos para o i -ésimo estágio (HWANG, 1979).



Figura 27 – Circuitos lógicos para a geração dos sinais auxiliares, *Carry Generate* e *Carry Propagate* (a) e soma (b) para o i -ésimo estágio

Substituindo p_i e g_i nas equações de soma e de *carry*, obtém-se, respectivamente (HWANG, 1979):

$$S_i = (A_i \oplus B_i) \oplus C_{i-1} = p_i \oplus C_{i-1} \quad (6)$$

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1} = (A_i B_i) + (A_i \oplus B_i) C_{i-1} = g_i + p_i C_{i-1} \quad (7)$$

Através das equações (4) e (5) nota-se que os sinais p e g de todos os estágios podem ser gerados simultaneamente a partir das entradas A e B (HWANG, 1979), como mostra a Fig. 27. A equação (6) mostra que todos os bits da soma S_i podem ser gerados em paralelo se todas as entradas de *carry* forem disponibilizadas simultaneamente. Logo, como todos os *carries* devem ser calculados paralelamente, as referências aos *carries* anteriores na equação (7) precisam ser eliminadas, o que pode ser feito eliminando-se a recursividade existente nestas equações. Deste modo, para um somador CLA de 3 bits, por exemplo, se obtém o seguinte conjunto de equações de propagação do *carry*, em termos dos sinais p , g e *carry* inicial, C_{in}

$$C_0 = g_0 + p_0 C_{in} \quad (8)$$

$$C_1 = g_1 + p_1 g_0 + p_1 p_0 C_{in} \quad (9)$$

$$C_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_{in} \quad (10)$$

Generalizando para um somador CLA de n bits, a geração do i-ésimo *carry* seria dada pela equação abaixo.

$$C_{n-1} = g_{n-1} + g_{n-2}p_{n-1} + \dots + p_0p_1\dots p_{n-1}C_{in} \quad (11)$$

Formatado: Inglês (EUA)

Conforme já mencionado, as equações de *carry* de um somador CLA mostradas anteriormente podem ser implementadas por um circuito lógico adicional, conhecido como unidade *carry lookahead* de n bits, sendo que o caso mais utilizado é n=4, como mostra a Fig. 28 (HWANG, 1979).

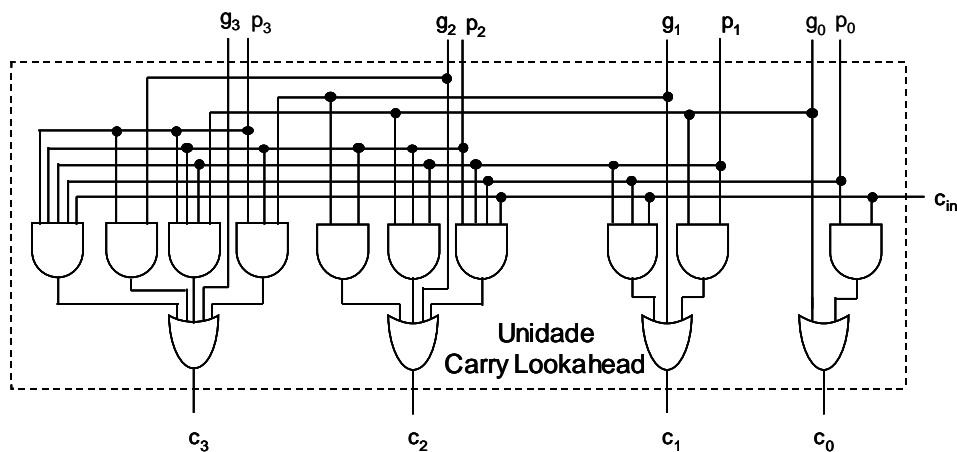


Figura 28 – Unidade *carry lookahead*

Através das equações de *carry* do CLA e do circuito que implementa estas equações, mostrado na Fig. 28, pode-se perceber que a implementação em *hardware* de um CLA utiliza uma maior quantidade de recursos que a implementação de um RCA.

Através da união de circuitos mostrados nas Figs. 27 e 28 é possível construir um somador CLA de 8 bits, como ilustrado na Fig. 29 (HWANG, 1979).

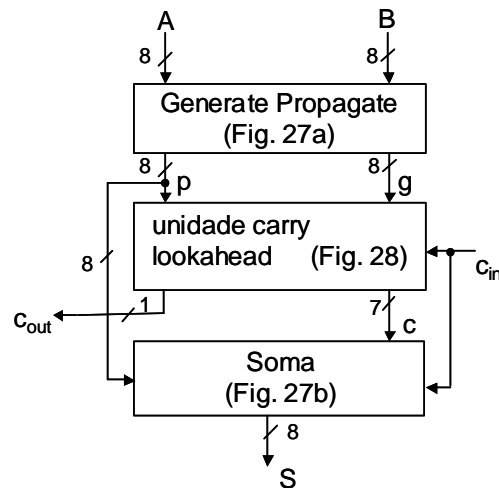


Figura 29 – Somador CLA de 8 bits com nível simples

Também se pode perceber, através do conjunto das equações de *carry* mostrado acima, que a complexidade das equações de *carry* cresce à medida que o número de bits dos operandos aumenta. Logo, é necessário o uso de níveis hierárquicos para facilitar a implementação de CLAs de mais alta ordem (HWANG, 1979). Para isso ser possível, a unidade *carry lookahead* mostrada na Fig. 28 precisa ser alterada, de modo que, além de gerar os *carries* de cada estágio da soma simultaneamente, gere outros dois sinais chamados *Carry Generate* do bloco, *gen*, e *Carry Propagate* do bloco, *prop*, como mostra a Fig. 30. Estes sinais indicam se o bloco como um todo gera ou propaga o *carry* de entrada (ou um dos *carries* internos, caso seja gerado um), respectivamente, possibilitando assim a utilização de níveis hierárquicos (HWANG, 1979). A unidade *carry lookahead* alterada gera três *carries* de saída, *c0*, *c1* e *c2*, e os sinais *gen* e *prop*, que são definidos, respectivamente, como:

$$\mathbf{gen_j = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3} \quad (12)$$

$$\mathbf{prop_j = p_0 p_1 p_2 p_3} \quad (13)$$

Onde j representa a j -ésima unidade *carry lookahead* utilizada para a implementação de somadores com um número de bits maior, através do uso de níveis.

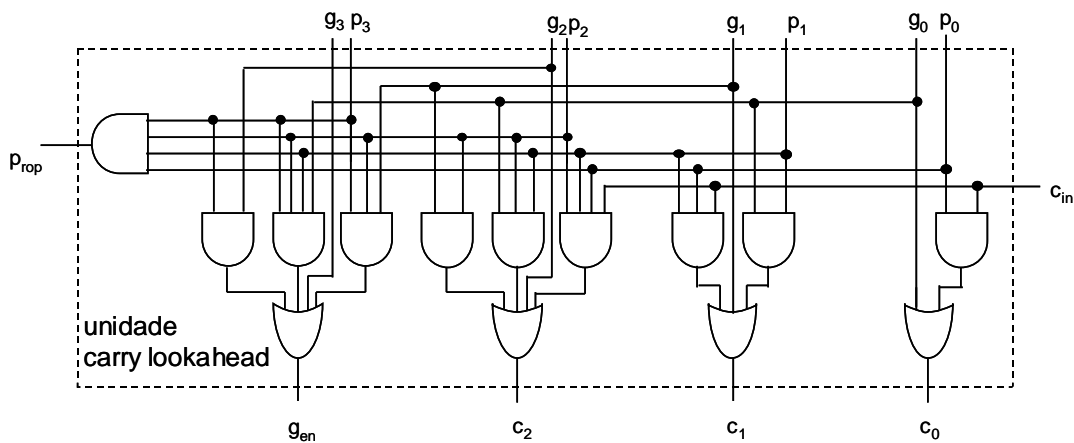


Figura 30 – Unidade *Carry Lookahead* alterada para a implementação de somadores *Carry Lookahead* com níveis hierárquicos

Deste modo, o somador é dividido em blocos de 4 bits, os quais são agrupados para gerarem somadores maiores. Como consequência, tem-se um somador com níveis hierárquicos equivalente a um CLA sem hierarquia, porém mais rápido. Os sinais *prop* e *gen* das diversas unidades *carry lookahead* são utilizados por um bloco, chamado Bloco *Generate-Propagate* (BGP), que está no próximo nível da hierarquia e que será responsável pela geração do *carry* de entrada para os blocos intermediários do somador e pela geração do *carry* de saída do somador.

As equações de geração dos *carries* de entrada para os blocos é similar àquela utilizada pela unidade *carry lookahead*. A diferença é que, ao invés de considerar os sinais p e g de cada bit da soma, são considerados os sinais *prop* e *gen* de cada um dos blocos básicos da hierarquia e os *carries* de entrada dos blocos intermediários são gerados pelo bloco BGP, ou seja, para um somador CLA de 8 bits o bloco BGP gera duas equações de *carry*, uma para o bloco que realiza a soma dos bits mais significativos e o *carry* de saída do somador.

$$C_4 = gen_0 + prop_0 0C_{in} \quad (14)$$

$$C_{out} = gen_1 + prop_1 gen_0 + prop_1 prop_0 C_{in} \quad (15)$$

A Fig. 31 ilustra o diagrama de blocos de um somador CLA de 8 bits, dividido em dois blocos de 4 bits, utilizando dois níveis.

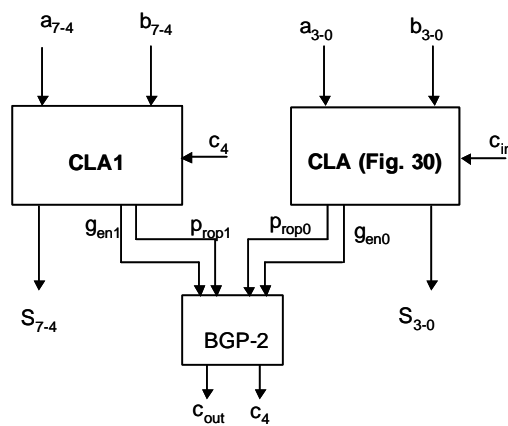


Figura 31 – Somador CLA de 8 bits utilizando dois níveis

A utilização de níveis hierárquicos tem como propósito aumentar ainda mais a velocidade de propagação do *carry* em relação a um CLA sem hierarquia, diminuindo a complexidade das equações de *carry* produzidas quando se usa a técnica *carry lookahead* em um somador. Logo, o atraso do CLA com níveis não é diretamente proporcional ao tamanho N do somador, mas ao número de níveis utilizados (OKLOBDZIJA, 2001). Devido ao fato dos somadores CLA utilizando níveis hierárquicos assemelharem-se com uma estrutura de árvore, o atraso do CLA é proporcional à função log do tamanho dos operandos de entrada (OKLOBDZIJA, 2001).

O atraso do CLA é avaliado através de um somador com um nível simples de *lookahead* (palavra de 4 bits) que contém um atraso de 3 portas no caminho do *carry*. Cada nível adicional de *lookahead* aumenta o tamanho máximo da palavra em um fator de k e adiciona o atraso de 2 portas, já que um somador CLA de um grupo de k bits introduz um atraso de 3 portas por nível.

Esta dependência do log torna o CLA teoricamente uma das estruturas mais rápidas para a adição (OKLOBDZIJA, 2001).

A Fig. 32 mostra um somador CLA de 32 bits usando dois níveis, 8 unidades de somadores de 4 bits no primeiro nível e um bloco BGP de 8 bits no segundo nível. As entradas para o BGP de 8 bits vêm das saídas prop e gen dos oito blocos de 4 bits. Os *carries* de saída do segundo nível são conectados nos *carries* de entrada dos blocos do primeiro nível.

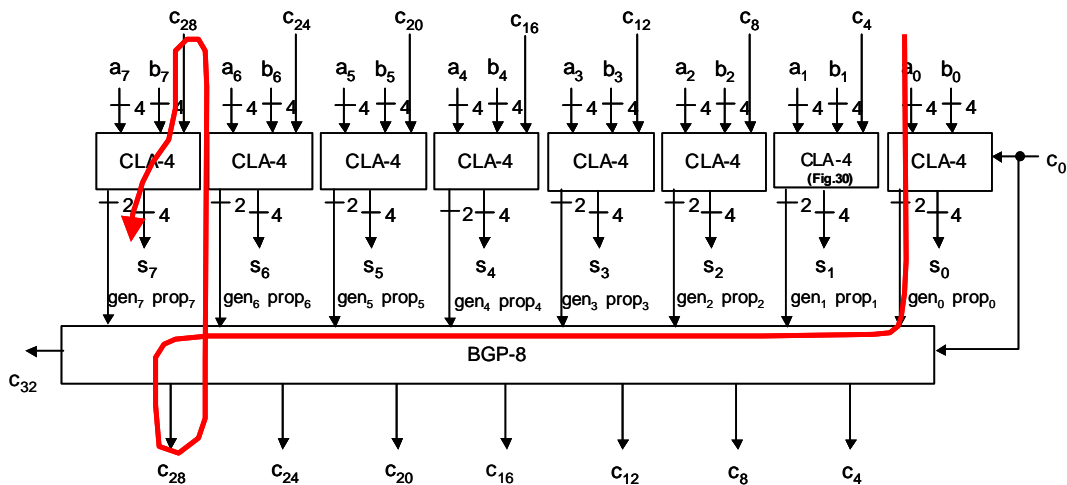


Figura 32 – Caminho crítico de um somador CLA de 32 bits

A mesma idéia pode ser estendida para projetar somadores maiores, com mais de dois níveis (HWANG, 1979).

5 Técnicas de Proteção contra Falhas Transientes

Atualmente, sistemas computacionais são cada vez mais utilizados por automatizarem e acelerarem várias tarefas do cotidiano. Porém, ao mesmo tempo em que facilitam algumas atividades, geram problemas que anteriormente não existiam.

Sistemas tolerantes a falhas eram, até pouco tempo atrás, uma preocupação apenas em projetos de sistemas críticos, como aviões, sondas espaciais e sistemas de tempo real. Porém, com o aumento da utilização da computação pela maioria da população, confiabilidade e disponibilidade estão, cada vez mais, sendo exigidos de qualquer tipo de sistema.

5.1 Terminologia e Conceitos

Para um estudo adequado de sistemas tolerantes a falhas é importante a apresentação da nomenclatura e de conceitos básicos utilizados na área de Tolerância a Falhas.

Três termos importantes na comunidade de Tolerância a Falhas são **falha**¹ (*fault*), **erro** (*error*) e **falha do sistema** (*failure*) (PRADHAN, 1996). Existe uma relação de causa e efeito entre falha, erro e falha do sistema. Falhas são as causas dos erros, e os erros são as causas das falhas do sistema. A **falha** é defeito físico ou imperfeição que ocorre dentro de algum componente de *hardware* ou *software*. Um **erro** é a manifestação de uma falha. Finalmente, se um erro resulta em uma ação não esperada,

¹ Embora amplamente utilizada pela comunidade de Tolerância a Falhas e pela comunidade de Teste, a palavra "falha" não parece ser a melhor tradução para *fault*. Talvez, uma tradução mais adequada fosse "imperfeição".

então ocorreu uma **falha do sistema**, ou seja, uma falha do sistema é qualquer desvio do comportamento desejado ou esperado do sistema (PRADHAN, 1996).

Já na comunidade de teste são usados os termos **defeito** (*defect*), **erro** (*error*) e **falha** (*fault*), onde **defeito** é definido como sendo a diferença não planejada entre o *hardware* implementado e o seu projeto pretendido, o **erro** é uma saída incorreta produzida por um sistema defeituoso e a **falha** é a representação de um defeito em um nível de abstração mais alto (BUSHNELL; AGRAWAL, 2000).

Neste trabalho os termos explicados acima serão unidos, assumindo que **defeitos** são imperfeições no *hardware* do sistema e **falhas** são a modelagem física dos defeitos. Os conceitos de defeito, falha, erro e falha do sistema podem ser melhor representados usando um modelo de três universos (neste caso, modificado para compatibilizar as nomenclaturas da comunidade de teste e da comunidade de Tolerância a Falhas). O primeiro universo é o universo físico, onde ocorre o defeito que é modelado pela falha. O universo físico contém componentes semicondutores, elementos mecânicos, fonte de energia, e outras entidades físicas que o sistema possui. O segundo universo é o universo da informação, que é onde ocorre o erro. Erros afetam unidades de informações, como palavras de dados internas do computador, informações de imagens etc. O último universo é o universo do usuário, ou externo (Fig. 33). O universo externo é onde o usuário do sistema percebe o efeito da falha e do erro, ou seja, o universo externo é onde o defeito se manifesta.

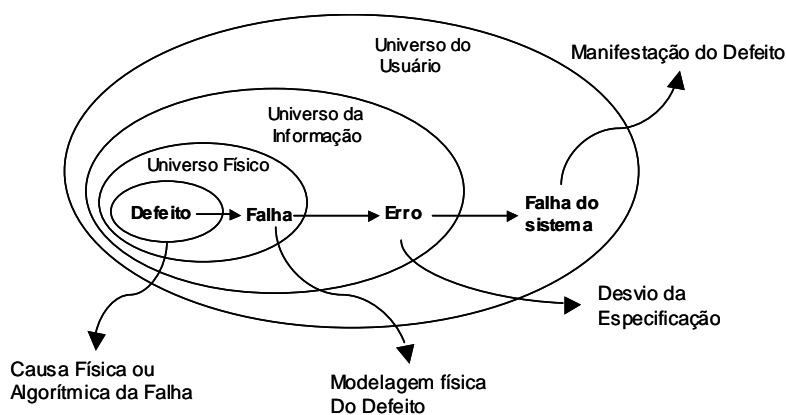


Figura 33 – Adaptação do modelo dos três universos

Falhas podem resultar em erros no universo da informação, e erros podem finalmente resultar em falhas do sistema sentidas no universo do usuário. Porém, é importante ressaltar que uma falha não obrigatoriamente causa um erro e um erro também pode não conduzir obrigatoriamente a uma falha do sistema. Falhas que não se manifestam no sistema são chamadas de falhas latentes.

A partir do modelo de três universos pode se obter dois parâmetros importantes: Latência de Falha e Latência de Erro. O período de tempo desde a ocorrência de uma falha até a manifestação do erro causado devido a esta falha é definido como Latência de Falha.

Já o período de tempo desde a ocorrência do erro até a manifestação do defeito é definido como Latência de Erro (Fig. 34). Então, o tempo total entre a ocorrência de uma falha física e o aparecimento no sistema do defeito modelado por esta falha será a soma da latência de falha e da latência de erro.

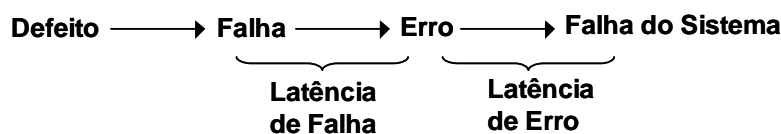


Figura 34 – Latência de Falha e Latência de Erro

As principais causas de falhas são erros de especificação (algoritmos incorretos e erros de interface entre os componentes do sistema), erros de implementação (práticas de projeto e escolha de componentes incorretas ou erros na codificação do *software*), componentes defeituosos (imperfeições de fabricação e/ou desgastes dos componentes físicos) e distúrbios externos (radiação, interferência eletromagnética, variações ambientais como temperatura, pressão e umidade, e também problemas de operação) (PRADHAN, 1996).

Além da causa, para uma falha ser definida adequadamente, outras características são necessárias:

- Natureza: especifica o tipo de falha, ou seja, indica se esta ocorreu no *software* ou no *hardware* do sistema.

- Duração: indica se a falha é permanente, transiente ou intermitente. Uma falha permanente se mantém indefinidamente, ou até que uma ação corretiva seja tomada, enquanto que uma falha transiente pode aparecer e desaparecer dentro de um curto período de tempo. Já a falha intermitente aparece, desaparece e então reaparece repetidamente.
- Extensão: indica se os efeitos da falha estão localizados em um determinado componente do sistema, ou se afetam outros módulos de *hardware* e *software*.
- Valor: pode ser determinado ou indeterminado.

Existem três técnicas principais para melhorar ou manter o funcionamento normal do sistema em ambiente onde falhas são preocupantes: Intolerância a falhas (*fault avoidance*), Mascaramento de Falhas e Tolerância a Falhas.

Intolerância a Falhas é composta por todas as técnicas usadas para evitar a ocorrência de falhas, incluindo revisões dos projetos, componentes selecionados, testes, e outros métodos de controle de qualidade. O Mascaramento de Falhas é todo o processo que impede que as falhas em um sistema produzam erros, ou seja, garante uma resposta correta mesmo na presença de falhas. A falha não se manifesta como erro, então o sistema não entra em estado errôneo. A técnica de Tolerância a Falhas é a habilidade do sistema de continuar funcionando corretamente, durante a ocorrência de uma falha e depois da ocorrência desta (PRADHAN, 1996). Tolerância a Falhas é a técnica abordada neste trabalho.

5.2 Tolerância a Falhas

Tolerância a Falhas pode ser realizada de por meio de várias técnicas que garantam o funcionamento correto do sistema, mesmo na ocorrência de falhas. Mascaramento é uma das abordagens de Tolerância a Falhas. Outra abordagem é a Detecção e Localização de Falhas, com a Reconfiguração do sistema.

O objetivo da Tolerância a Falhas é garantir a confiança e a qualidade de serviços oferecidos por sistemas computacionais, ou seja, a Tolerância a Falhas objetiva o alcance da dependabilidade (*dependability*). As principais medidas de

dependabilidade são confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), manutenibilidade, testabilidade e comprometimento do desempenho (*performability*) (PRADHAN, 1996).

Se em um sistema algum tipo de Tolerância a Falhas ou Detecção de Falhas for necessário, então alguma forma de redundância também será necessária. O conceito de redundância se baseia na adição de informações, recursos ou tempo além do necessário para a operação normal do sistema. Ou seja, a redundância pode ser obtida de várias formas, incluindo redundância de *hardware*, redundância de informação, redundância de *software* e redundância temporal. Porém, deve-se compreender que redundância pode ter um impacto importante no sistema, em termos de área, peso, desempenho, consumo de potência, confiabilidade, e outros.

A replicação física do *hardware* é a forma mais comum de redundância usada nos sistemas. O aumento da densidade de integração dos componentes semicondutores e a redução de seu custo favorecem o uso desta técnica. Existem três tipos básicos de redundância de *hardware*: redundância passiva (ou estática), dinâmica (ou ativa) e híbrida.

A redundância estática ou passiva, em sua forma mais básica, não provê a detecção da falha. Ela simplesmente mascara a falha, ou seja, ela utiliza a técnica de Mascaramento de Falhas, ao invés da Detecção de Falhas, para evitar a ocorrência e a propagação de erros no sistema (o efeito da falha é mascarado imediatamente). É projetada para realizar a Tolerância a Falhas sem requerer nenhuma ação por parte do sistema ou do usuário.

Todos os módulos do sistema executam a mesma tarefa e o resultado é determinado por meio de uma votação por maioria. NMR (*n-modular redundancy*) é uma das técnicas que utiliza redundância de *hardware* estática, e sua aplicação mais comum é a TMR (*Triple Modular Redundancy*), que será explicada posteriormente. (PRADHAN, 1966; LALA, 2001).

Um sistema utilizando Redundância de *Hardware* Dinâmica é constituído de vários módulos, porém com somente um módulo em operação. A redundância empregada neste caso não provê o mascaramento. Se uma falha é detectada no módulo em funcionamento, este é substituído por um dos módulos reserva. Desta

forma, Redundância de *Hardware* Dinâmica requer consecutivas ações de Detecção e Recuperação de Falhas, ou seja, a Tolerância a Falhas é realizada através da aplicação das técnicas de Detecção, Localização e Recuperação de Falhas (PRADHAN, 1966; LALA, 2001).

Detecção de Falhas é o processo de reconhecer que uma falha está ocorrendo e a Localização de Falhas é a determinação de onde a falha está ocorrendo, para que então possa ser aplicada uma Recuperação adequada. Já o Confinamento de Falhas é o processo de isolar a falha, impedindo os efeitos da propagação desta através do sistema. A Recuperação de Falhas ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas.

Reconfiguração, no contexto de Tolerância a Falhas, é o processo de eliminação do módulo defeituoso do sistema e a conseqüente restauração do sistema para alguma condição ou estado operacional. Se a técnica de reconfiguração for usada, então o projeto deve se preocupar com a detecção, localização, confinamento e recuperação de falhas.

Definições equivalentes podem ser feitas para o universo da informação, onde Detecção de Erros é o processo de reconhecer que um erro ocorreu, a Localização de Erros é o processo de determinar qual o módulo específico produziu o erro, o Confinamento de Erros é o processo de impedir que erros se propaguem pelo sistema e a Recuperação de Erros, como na Recuperação de falhas, é a troca do estado atual incorreto para um estado livre de falhas (PRADHAN, 1996).

Assumindo que as técnicas de Detecção de falhas e Reconfiguração do sistema são perfeitas, um sistema utilizando Redundância de *Hardware* Dinâmica com S módulos reservas tem uma confiabilidade de

$$R = 1 - (1 - R_m)^{(S+1)} \quad (16)$$

onde: R_m é a confiabilidade de cada módulo ativo ou reserva no sistema. (Lala, 2001).

Esta técnica normalmente implica em uma interrupção do funcionamento do sistema. Logo, é mais utilizada em aplicações nas quais resultados errôneos são aceitáveis durante um breve período de tempo (como por exemplo, os satélites) (PRADHAN, 1996), tempo esse necessário para a detecção do erro e recuperação para um estado livre de falhas. Exemplos de técnicas que utilizam Redundância Dinâmica são Duplicação com Comparação (*Duplication with Comparison*) e Módulos Reservas (*Standby Sparing*) (Fig.35) (PRADHAN, 1996).

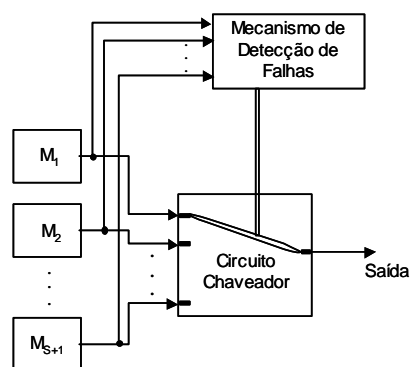


Figura 35 – Redundância de *hardware* Dinâmica com S reservas

A redundância híbrida procura combinar as vantagens das redundâncias de *hardware* estática e dinâmica. O mascaramento de falhas é utilizado para impedir que o sistema gere um resultado errôneo e a Detecção, Localização e Recuperação são utilizadas para reconfigurar o sistema na ocorrência de uma falha (PRADHAN, 1996). A maior parte dos sistemas empregando redundância híbrida se baseia no conceito de NRM com reserva (*N-modular redundancy with spares*), que combina as técnicas NMR e *Standby Sparing*, ou seja, consiste em compor um conjunto de módulos operando em NMR, os quais, na ocorrência de falhas, podem ser substituídos por módulos de reserva (PRADHAN, 1966; LALA, 2001).

Um sistema com quatro módulos redundantes poderia ser configurado da seguinte maneira: três módulos operam (TMR) até a ocorrência da primeira falha. Nesta ocasião, o módulo defeituoso é retirado do conjunto e substituído pelo módulo de reserva. Como resultado, o sistema com quatro módulos redundantes passa a ter

capacidade para tolerar até duas falhas; caso fosse empregada somente a redundância estática, seriam necessários cinco módulos para atingir o mesmo resultado.

A implementação física de um sistema utilizando Redundância de *Hardware* Híbrida com três módulos em funcionamento e com S módulos reservas é mostrada na Fig. 36. O circuito detector de discordância detecta se a saída de qualquer um dos três módulos é diferente da saída do circuito votador por maioria (*majority voter*). Se um dos módulos discorda do votador, o circuito chaveador substitui o módulo defeituoso por um dos módulos reserva (LALA, 2001).

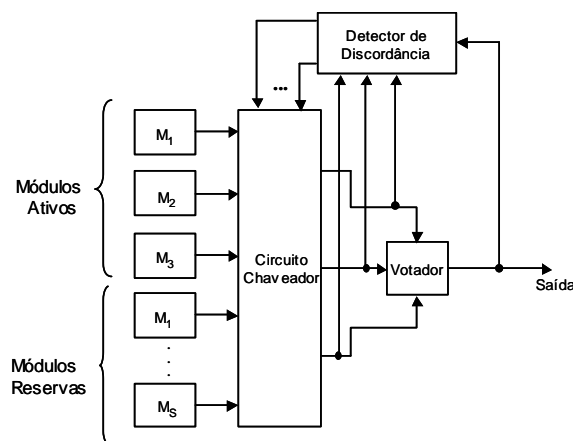


Figura 36 – Sistema híbrido com TMR e S módulos reservas

Assumindo que os circuitos votador, detector de discordância e chaveador estão livres de falhas e que a confiabilidade de todos os módulos, tanto os em funcionamento como os reservas, são equivalentes, a confiabilidade do sistema híbrido com TMR e S módulos reservas é:

$$R(3, S) = 1 - (1 - R_m)^{(S+2)} \cdot \{1 + R_m^{(S+2)}\} \quad (17)$$

Onde: R_m é a confiabilidade de cada um dos módulos e S é o número de módulos reservas. (LALA, 2001).

O principal problema na aplicação de redundância é o custo extra de *hardware* para a implementação de várias técnicas. Tanto a Redundância de *hardware* como a Redundância de *software* necessita uma quantidade extra de *hardware* para a sua implementação. Redundância temporal é uma opção de implementação que busca diminuir o *hardware* necessário para realizar a Detecção de Falhas ou Tolerância a Falhas, utilizando tempo adicional. Em muitas aplicações o tempo é menos importante do que o *hardware*, por este último ser uma entidade física que gera impactos no tamanho, peso, energia consumida e custo. Logo, o tempo pode ser explorado nestas aplicações.

Redundância de Tempo é normalmente usada para detectar e corrigir erros causados por falhas temporárias e envolve a repetição de instruções, segmentos de programas ou programas inteiros (LALA, 2001). A principal desvantagem desta técnica é que ela requer mais tempo para obter o resultado verificado. Logo, não é ideal para ser utilizada em aplicações onde o tempo é um fator importante (LALA, 2001).

Portanto, é necessário saber que a escolha do uso de algum tipo de redundância depende do tipo de aplicação onde esta será utilizada. Logo, a utilização de redundância deve ser realizada analisando o que cada aplicação necessita e avaliando qual das técnicas pode melhor suprir suas necessidades (PRADHAN, 1996).

Muitas técnicas de Detecção de Falhas e de Tolerância a Falhas podem ser implementadas em *Software*. Redundância de *Software* pode ser implementada de várias formas. Na redundância de *software* diferentes versões do mesmo *software* são necessárias. A simples replicação de módulos idênticos é uma técnica inútil em *software*, uma vez que módulos idênticos de *software* irão apresentar erros idênticos. Assim, não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes, em tempos diferentes. Redundância de *Software* pode aparecer, por exemplo, em linhas extras de códigos em um programa utilizado para verificar a magnitude de um sinal, ou como uma rotina para testar periodicamente posições específicas de leitura e escrita da memória (PRADHAN, 1996).

A desvantagem da Redundância de *Software* é que não é possível determinar a melhora esperada na confiabilidade do sistema que utiliza esta técnica (LALA, 2001).

Programação com N-Versões (*N-version programming*), Blocos de Recuperação (*Recovery Blocks*), Verificação de Consistência (*Consistency Checks*) e Verificação de Capacidade (*Capability Checks*) são exemplos de algumas técnicas que utilizam Redundância de *Software* (PRADHAN, 1966; LALA, 2001).

A redundância de informação consiste na duplicação dos dados ou armazenamento de informação redundante que permite a detecção e mascaramento de falhas e possibilita a Tolerância a Falhas. Redundância de informação é obtida por meio de Códigos de Detecção e de Correção de Erros (*Error Correction and Detection Codes*). Existem vários Códigos de Detecção e Correção de Erros. Dentre eles os principais são: Duplicação, Paridade, *Checksum*, *Hamming* e *Reed-Solomon* (PRADHAN, 1966; LALA, 2001).

Redundância de *Software* e Redundância de Informação fogem ao escopo deste trabalho.

5.3 Projetos de Circuitos Tolerantes a Falhas Transientes

Atualmente, um dos principais fatores que tem gerado demanda para o desenvolvimento de projetos tolerantes a falhas é o aumento da vulnerabilidade dos Circuitos Integrados fabricados com tecnologias CMOS nanométricas a falhas transientes causadas pela colisão de partículas energéticas, conforme mencionado no capítulo 2. Além disso, a grande variabilidade de parâmetros de processo de tais tecnologias e o estresse dos materiais são fatores que também contribuem para o surgimento de falhas de funcionamento.

A blindagem (*shielding*) foi a primeira solução encontrada para amenizar a vulnerabilidade dos circuitos integrados e aumentar a tolerância destes a falhas transientes. Porém, esta solução não eliminava completamente a incidência das partículas energéticas nos circuitos integrados, mas gerava um alto custo de projeto.

Nos dias de hoje, o projeto de sistemas que possuem como objetivo a Tolerância a Falhas transientes pode ser realizado através das seguintes técnicas: processos de fabricação específicos, restauração da programação (que é uma técnica

aplicada somente a sistemas implementados em FPGAs, pois estes podem ser reprogramados) e técnicas de projeto dedicadas.

Processos de fabricação específicos são processos diferentes do processo de fabricação CMOS padrão e podem diminuir alguns dos efeitos da radiação a níveis aceitáveis. Porém, tais processos apresentam altos custos de fabricação, além de não eliminarem completamente a vulnerabilidade dos circuitos a SETs e a SEUs.

A técnica que utiliza a restauração da programação, por ser aplicada em FPGAs, usa a reconfiguração dinâmica de sistemas (ABRAMOVICI et al., 1999; CARMICHAEL; CAFFREY; SALAZAR, 2000; GOKHALE et al., 2004). Nesta técnica, a memória de configuração que mantém a programação do FPGA é periodicamente lida, com o objetivo de detectar alguma possível inversão de bit. Se uma inversão de bit for detectada, duas técnicas podem ser aplicadas: *scrubbing* ou reconfiguração parcial.

Scrubbing consiste em recarregar completamente a programação do FPGA. Já na reconfiguração parcial, a programação do FPGA é parcialmente refeita, recarregando apenas o *frame* da região onde ocorreu a inversão (CARMICHAEL; CAFFREY; SALAZAR, 2000). Porém esta técnica de restauração da programação pode ser usada somente em FPGAs baseados em SRAM. Além disso, ela apresenta a desvantagem de apenas poder detectar e corrigir falhas que afetam o *bit-stream*, ou seja, os conteúdos dos bits SRAM que configuram o FPGA.

As técnicas de tolerância a falhas transientes baseadas em projetos dedicados podem ser aplicadas tanto nos circuitos seqüenciais como nos blocos combinacionais dos sistemas. A aplicação de tais técnicas requer o uso de redundância de *hardware*, redundância de tempo ou uma combinação de ambas, ou seja, é necessária a utilização de algum tipo de redundância. A seguir, serão explicadas as técnicas de tolerância baseadas em projetos dedicados mais comumente utilizadas.

A técnica NMR (*n-modular redundancy*) consiste na utilização de *n* módulos iguais que realizam a mesma computação. Para a determinação da resposta correta é utilizado um votador (Fig. 37). Essa técnica considera que a probabilidade de mais de um módulo apresentar falhas durante a computação é muito pequena e desse modo, a confiabilidade do sistema seria aumentada. Essa consideração exige que os módulos sejam independentes no que diz respeito às falhas.

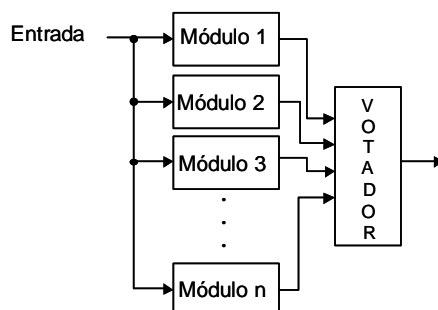


Figura 37 – Técnica NMR (*N-Modular Redundancy*)

A Redundância Modular Tripla ou TMR (*Triple Modular Redundancy*) é a forma mais comum de redundância estática de *hardware*, ou seja, é a aplicação da redundância de *hardware* pura (PRADHAN, 1996). Segundo Lala (2001, p 162), o conceito de TMR foi originalmente proposto por von Neumann em 1956.

O conceito básico da TMR consiste em triplicar um módulo que se deseja proteger e aplicar suas saídas a um votador (Fig. 38). Este votador reproduz em sua saída o valor lógico correspondente a pelo menos duas de suas entradas. Assim, mesmo que um dos módulos redundantes apresente alguma falha, os resultados dos outros dois módulos irão mascarar-la na saída do votador.

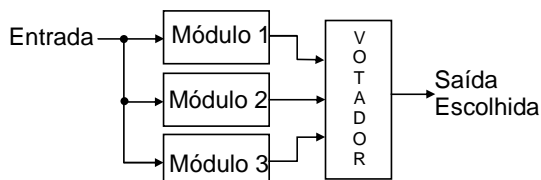


Figura 38 – Redundância Modular Tripla (TMR)

O votador não tem a função de detectar qual módulo redundante que realizou a computação de forma errônea. Se o sistema necessitar desta detecção, um circuito detector de falhas deverá ser implementado, ou seja, a simples aplicação da técnica TMR não será suficiente. A Fig. 39 mostra o circuito votador

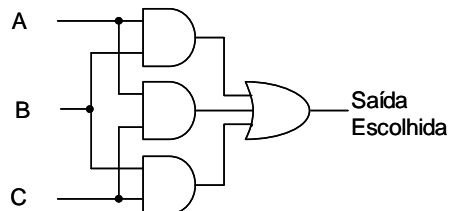


Figura 39 – Circuito Votador

As principais desvantagens desta técnica são o significativo acréscimo de *hardware*, o qual é normalmente estimado como sendo maior que 200%.

Na técnica TMR uma falha que ocorre em qualquer um dos módulos redundantes poderá ser mascarada. Porém, uma falha no votador não poderá ser mascarada, pois nesta parte do sistema não existe nenhum tipo de redundância, resultando em uma falha em todo o sistema. Logo, o votador é um ponto único de falhas (*single-point of failure*) da técnica TMR. Além disto, se dois módulos apresentarem o mesmo erro, o votador escolherá um resultado incorreto.

Assumindo que os módulos redundantes possuam a mesma confiabilidade e que falhas em tais módulos são independentes umas das outras, e assumindo ainda o votador como sendo um elemento livre de falhas, a confiabilidade da técnica TMR será: R_{TMR} = probabilidade de todos os 3 módulos funcionando corretamente + probabilidade de quaisquer dois módulos funcionando corretamente.

$$R_{TMR} = R_{M3} + 3R_{M2}(1 - R_M)$$

$$R_{TMR} = 3R_{M2} + 2R_{M2} \quad (18)$$

Onde: R_{TMR} é a confiabilidade do sistema protegido pela TMR e R_M é a confiabilidade de cada módulo.

A confiabilidade da TMR é usualmente melhor que a sugerida pela equação (18), desde que o sistema possa continuar funcionando corretamente se dois módulos também falharem.

Assumindo que o votador não é um elemento livre de falhas, sua confiabilidade, R_V , deve também ser considerada e então, o valor de R_{TMR} deve ser multiplicado por esta.

Então, a confiabilidade da técnica TMR será:

$$R_{TMR} = (3R_{M2} + 2R_{M2})R_V \quad (19)$$

Onde: R_{TMR} é a confiabilidade do sistema protegido pela TMR, R_M é a confiabilidade de cada módulo e R_V é a confiabilidade do votador.

Através da equação (19) pode-se concluir que a confiabilidade do votador limita a confiabilidade do sistema protegido pela TMR. Logo, para aumentar a confiabilidade do votador outras soluções precisam ser implementadas, tais como construir votadores com componentes de alta confiabilidade, triplicar o votador ou realizar a votação por meio de *software*.

A Fig. 40 mostra o uso de votadores triplicados. Desta forma, uma falha em qualquer componente afetará apenas uma das saídas redundantes, e poderá ser corrigida no próximo bloco do sistema. Esta estrutura é comumente chamada de órgão regenerador (*restoring organ*), pois possibilita a obtenção de três saídas corretas independentes mesmo na presença de uma entrada incorreta (PRADHAN, 1996).

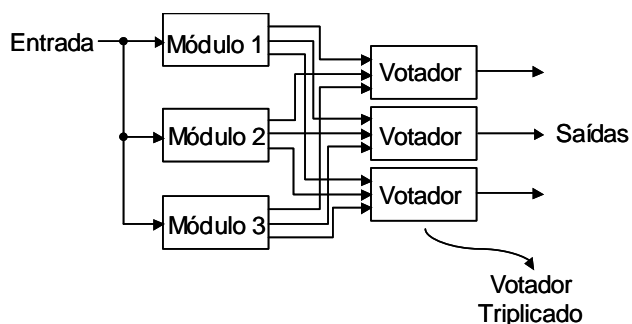


Figura 40 – Órgão Regenerador (*Restoring Organ*)

TR (*Time Redundancy*) corresponde à aplicação da redundância temporal pura, ou seja, em vez de triplicar o bloco a ser protegido, faz-se uma amostragem do resultado em três momentos diferentes. A diferença de tempo entre as amostras do resultado deve garantir o desaparecimento da falha, caso esta ocorra. Para a implementação desta técnica, mostrada na Fig. 41, é necessária a triplicação de elementos de armazenamento para a obtenção das três amostras do resultado, que serão usadas como entradas do votador. Nesta técnica o aumento de *Hardware* não é tão grande quanto na técnica TMR. Porém, esta técnica apresenta certa complexidade devido ao fato dos elementos de memória possuírem diferentes momentos de disparo. Logo, esta técnica possui uma desvantagem em relação ao tempo necessário para realização da computação, o qual será aproximadamente igual a $clk+2d+tp$, onde tp é o atraso do votador (LIMA-KASTENSMIDT, 2003).

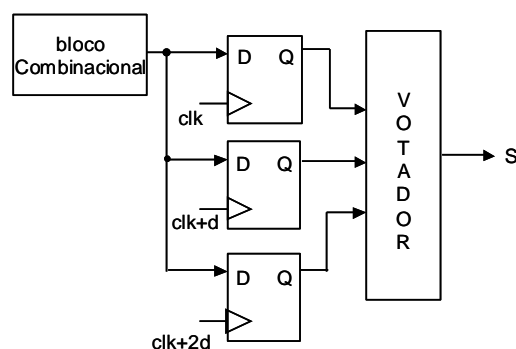


Figura 41 – Redundância Temporal utilizada para mascarar falhas transientes

Redundância temporal pode ser utilizada também para detectar falhas permanentes, como mostra a Fig. 42. Esta técnica utiliza um mínimo de *hardware* extra para a realização da computação em tempos diferentes. Durante uma primeira computação, realizada no instante t_0 , os operandos são usados normalmente e os resultados gerados são armazenados em registradores. Em uma segunda computação, realizada no instante $t_0 + d$, os operandos são codificados de alguma maneira, utilizando funções de codificação. Depois que a operação é realizada sobre os

operandos codificados, os resultados desta segunda computação são então decodificados e comparados com os resultados gerados na primeira computação.

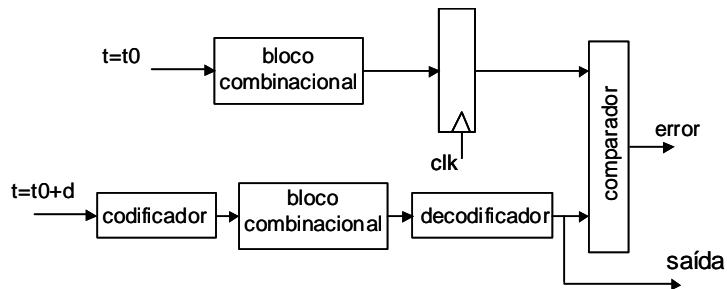


Figura 42 – Redundância Temporal usada para a detecção de falhas permanentes

Uma outra solução possível para se obter Tolerância a Falhas e amenizar o acréscimo de *hardware* é proteger apenas os pontos mais vulneráveis dos circuitos integrados, ou seja, triplicar apenas as regiões mais sensíveis do circuito.

Como as células de memória se constituem em um dos pontos mais sensíveis dos circuitos integrados, a técnica TMR pode ser aplicada a tais componentes, para aumentar a tolerância destes a SEUs. Neste caso, a célula de memória é triplicada e suas saídas são aplicadas nas entradas de um votador (Fig. 43). Porém, caso ocorra uma falha no votador, este pode realizar uma votação incorreta e pode ocorrer uma acumulação de SEUs, com duas ou mais células armazenando um valor incorreto.

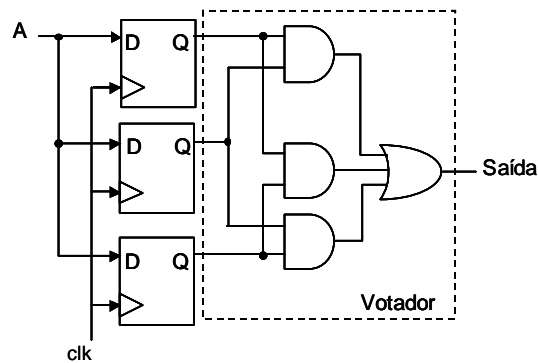


Figura 43 – Células de Memórias protegidas contra SEUs com a técnica TRM

Além de proteger as células de memória contra SEUs, também é necessária a proteção dos blocos combinacionais contra SETs, devido às razões explicitadas no capítulo 2.

A proteção dos blocos combinacionais também se dá através do uso de redundância, ou seja, pode ser baseada no uso de redundância de *hardware* pura, através da aplicação de técnicas como TMR, redundância temporal, ou através da combinação de ambas (Fig. 44).

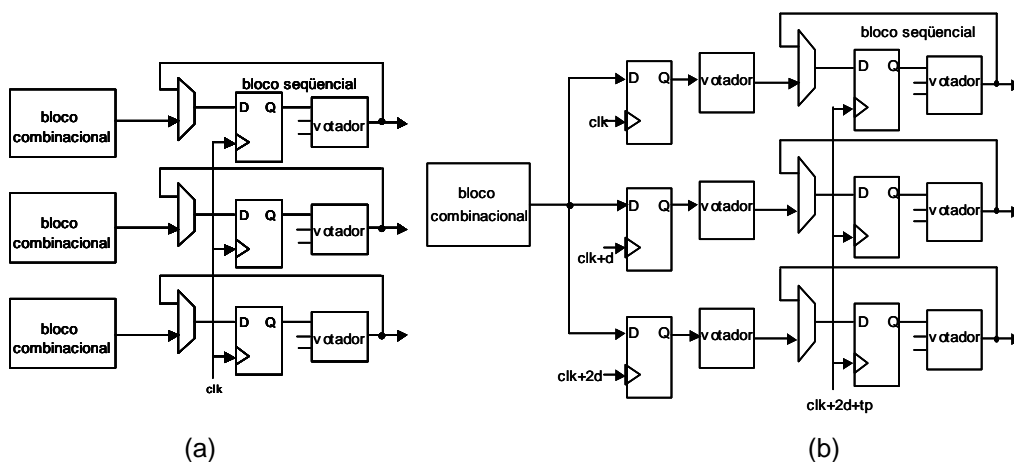


Figura 44 – Blocos Combinacionais protegidos contra SETs através da redundância de *hardware* (a) e redundância temporal (b)

A combinação da redundância de *hardware* e da redundância temporal é uma alternativa que tenta amenizar as desvantagens em relação ao aumento de *hardware* provocado pela redundância de *hardware* pura e melhorar o desempenho que é reduzido pela aplicação da redundância temporal pura. Um exemplo da aplicação desta técnica é o método que procura proteger um bloco combinacional contra SETs realizando a duplicação deste e utilizando um estágio CWSP (*Code Word State Preserving*) encontrado em (Anghel; Alexandrescu; Nicolaidis, 2000) (Fig. 45). Este método não necessita de votador e a duplicação dos circuitos pode ser substituída por redundância temporal, o que diminui o *hardware* utilizado, mas gera uma perda no desempenho.

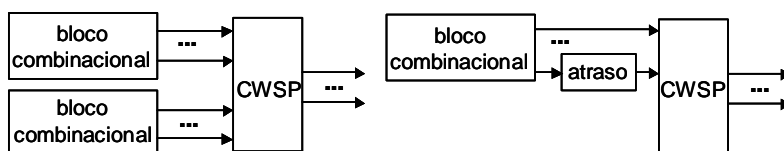


Figura 45 – Bloco Combinacional protegido contra SETs com Duplicação e CWSP

Também é possível alcançar a tolerância a falhas transientes utilizando técnicas para a proteção baseadas em projeto dedicado que fazem o uso de códigos de Detecção e Correção de Erros (EDAC), os quais foram mencionados anteriormente. Os códigos mais usados são os de *Hamming* e os de *Reed-Solomon* e necessitam de bits extras para codificar paridade (usados principalmente para proteger memórias).

Uma outra forma de se alcançar a tolerância a falhas transientes é a utilização de células de memórias protegidas (*hardened memory cells*) no processo de fabricação. Alguns exemplos de células protegidas são: Célula protegida da IBM (ROCKET, 1988) , Célula “Hit” (CALIN; NICOLAIDIS; VELAZCO, 1996), Célula “Canaris” (CANARIS; WHITAKER, 1995), Células DICE (CANARIS; WHITAKER, 1995) e Célula NASA (LIU; WHITAKER, 1992). Porém, um estudo mais aprofundado sobre células de memória protegidas foge ao escopo deste trabalho.

As técnicas de tolerância a falhas transientes explicadas anteriormente são aplicadas tanto para projetos a serem fabricados com o uso de máscaras quanto para

projetos baseados em FPGAs. Porém, a seguir serão explicadas técnicas de Tolerância a Falhas no nível arquitetural a serem aplicados ao projeto com FPGAs.

Primeiramente, as técnicas de proteção com FPGAs no nível arquitetural utilizaram a técnica TMR combinada com restauração da programação (CARMICHAEL; CAFFREY; SALAZAR, 2000; CARMICHAEL, 2001). Outras técnicas arquiteturais para projetos com FPGAs propostas mais recentemente foram DWC-CED, que é encontrada em (LIMA; CARRO; REIS, 2003), e DWC+TR (DWC - *duplication with comparison*, TR – *Time redundancy*).

DWC combinada com redundância temporal é baseada na utilização da redundância de *hardware* juntamente com a redundância temporal. A redundância temporal pode apenas detectar falhas transientes na lógica combinacional (NICOLAIDIS, 1999; ANGHEL; ALEXANDRESCU; NICOLAIDIS, 2000). O mesmo acontece com a Duplicação com Comparação - DWC. Entretanto, a combinação da redundância temporal com DWC provê uma avaliação interessante das falhas, pois pode não somente detectar a presença de falhas, como também reconhecer em qual dos blocos redundantes a falha ocorreu (KASTENSMIDT, 2003).

Esta técnica utiliza dois blocos combinacionais redundantes. Desta forma, uma falha em um dos blocos pode ser detectada e votada antes de ser armazenada por algum elemento de memória. Quatro amostras do resultado são armazenadas em *latches* auxiliares, dois para cada bloco redundante, que armazenam em tempos diferentes. Dois *latches* armazenam em clk e os outros dois *latches* armazenam em clk somado a um atraso, $clk + d$. Este atraso deve ser grande o suficiente para que, caso uma falha ocorra, esta não seja armazenada por nenhum dos *latches* (KASTENSMIDT, 2003).

Estas quatro amostras são usadas como entradas do bloco de detecção de erro, que é responsável por verificar se um erro ocorreu em um dos dois blocos. Se este for o caso, o detector do erro entrega o valor correto a ser usado como a terceira entrada do votador. O votador recebe também a saída de cada bloco redundante. Sua saída é armazenada em um registrador no instante $clk+2d$. A fig. 46 mostra a técnica DWC+TR.

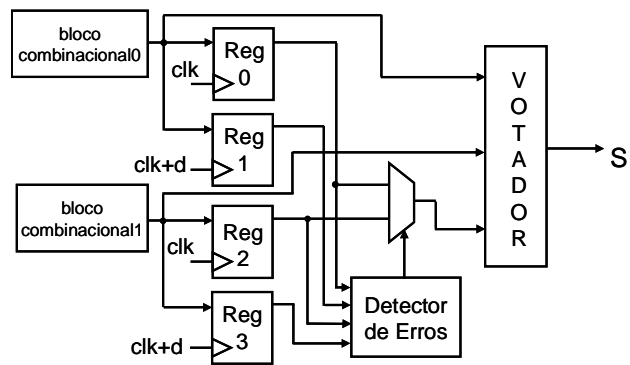


Figura 46 – DWC combinada com Redundância temporal

Não existe técnica de tolerância que consiga manter o projeto totalmente tolerante a ocorrência de falhas transientes: elas apenas aumentam a imunidade dos sistemas a falhas, diminuindo a probabilidade de SETs e SEUs se transformarem em erros.

6 Experimentos Práticos e Resultados

Este capítulo apresenta a aplicação das técnicas de proteção aos somadores *Carry Lookahead* e *Ripple Carry*. Também são analisados detalhadamente os resultados gerados para as arquiteturas descritas.

O somador *Ripple Carry* é um somador básico e de fácil implementação, sendo por isso utilizado como referência para comparação com o somador *Carry Lookahead*. As comparações entre os somadores foram feitas em termos de recursos utilizados do FPGA e de desempenho, medidos através do número de ALUTs utilizadas e do atraso crítico, respectivamente.

Todas as arquiteturas foram descritas em VHDL e sintetizadas utilizando o *software* Quartus II, na versão 7.0, da Altera (2007). Foram descritos somadores de 4, 8, 16, 32, 64 e 128 bits.

As arquiteturas foram sintetizadas para o dispositivo EP2S15F484C3 da família Stratix II da Altera, que foi escolhida por ser a família da Altera mais utilizada atualmente. Além disso, os dispositivos desta família são fabricados em tecnologia CMOS de 90nm, sendo, portanto suscetíveis aos SETs.

A validação do funcionamento foi feita por meio de simulação funcional com atrasos também utilizando a ferramenta Quartus II da Altera, na versão 7.0.

A validação da proteção foi feita por meio de campanhas de injeção de falhas utilizando a ferramenta de simulação ModelSim da Mentor Graphics (versão para Altera) (MENTOR, 2007).

As arquiteturas foram descritas utilizando dois registradores, um registrador conectado à entrada do somador, que recebe como entradas os operandos e o *carry* de entrada, e outro registrador conectado à saída do circuito. Esses registradores foram

utilizados para auxiliar na obtenção do atraso crítico das arquiteturas, de modo a neutralizar a influência dos atrasos dos pinos de entrada e saída do dispositivo.

6.1 Arquiteturas Descritas

A fim de realizarem-se os experimentos, os somadores considerados neste trabalho foram descritos em quatro versões: não protegida, protegida com TMR (*Triple Modular Redundancy*), protegida com TR (*Time Redundancy*) e protegida com Duplicação com Comparação combinada com Redundância Temporal - DWC+TR (*DWC - duplication with comparison*).

Os somadores não-protegidos foram descritos utilizando dois registradores, o bloco somador e, no caso do *Carry Lookahead*, o bloco BGP (Bloco *Generate-Propagate*), que serve para gerar os *carries* de entrada para os blocos básicos *Carry Lookahead* de 4 bits, no caso da utilização de hierarquia, e o *carry* de saída do somador.

O bloco básico *Carry Lookahead* de 4 bits foi descrito contendo a lógica que gera os sinais *Generate* e *Propagate* (GP), a lógica que realiza a soma propriamente dita (SOMA) e a lógica que gera os *carries* para a soma de forma simultânea, chamada de Unidade *Carry Lookahead*. O circuito que implementa o bloco básico de 4 bits é mostrado na Fig.47.

Este bloco é utilizado como base para todos os somadores *Carry Lookahead* (CLA) descritos, ou seja, é a partir deste que são gerados os somadores cujos operandos de entrada possuem mais bits (8, 16, 32, 64 e 128 bits). Isso se deve ao fato do número de portas necessárias em um somador CLA crescer conforme crescem as equações dos *carries*, pois estas são recursivas, tornando-se assim inviável a construção de somadores CLA sem a utilização de hierarquia.

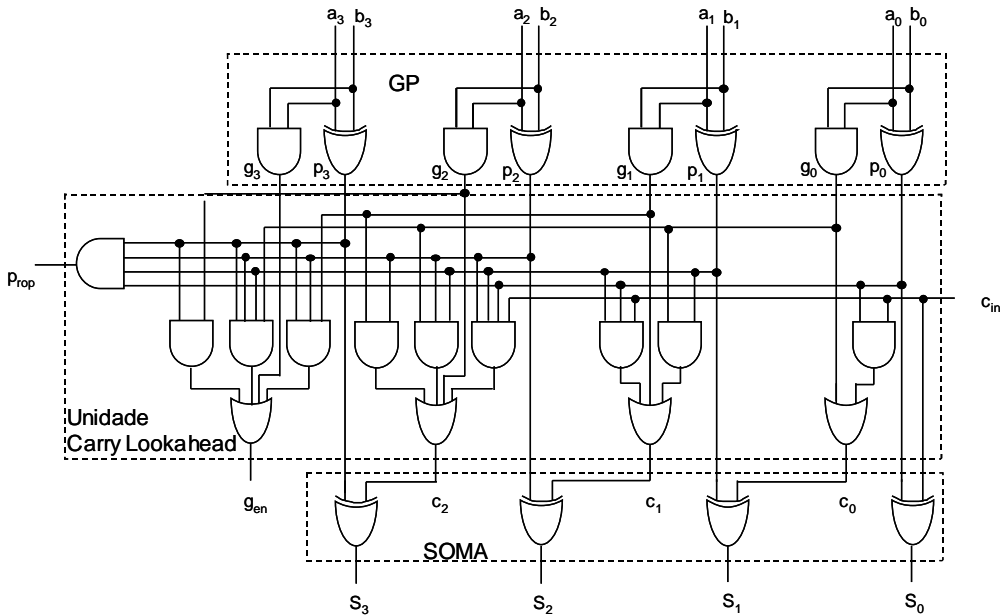


Figura 47 – Bloco básico *Carry Lookahead* de 4 bits

As Figs. 48 e 49 mostram, respectivamente, como foram descritos o circuito do bloco BGP para um somador de 4 bits e o CLA de 4 bits completo. O circuito BGP recebe como entradas os sinais *propagate* e *generate* vindos do bloco básico *carry lookahead* de 4 bits e gera o *carry* de saída do somador CLA de 4 bits. Para 4 bits, o bloco BGP gera apenas um *carry* que é o *carry* de saída do somador como um todo.

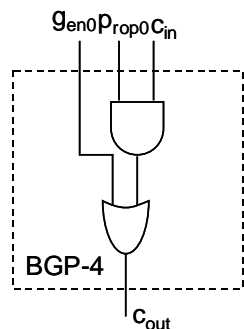


Figura 48 – Bloco *Generate-Propagate* para 4 bits

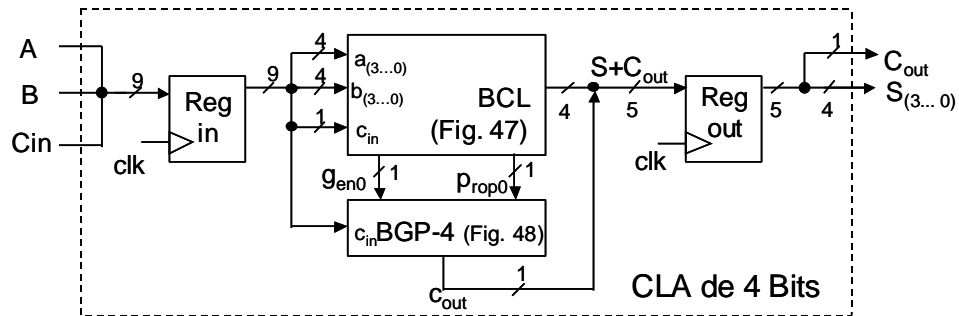


Figura 49 – Somador *Carry Lookahead* de 4 bits não protegido

Para somadores cujos operandos possuem um número maior de bits, o bloco BGP, além de gerar o *carry* de saída, gera também os *carries* de entrada dos blocos básicos *carry lookahead* de 4 bits que são instanciados para formar o somador. Por exemplo, em um CLA de 8 bits são usados dois blocos básicos *carry lookahead* 4 bits. Logo o bloco BGP para um CLA de 8 bits gera dois *carries*, um *carry* de entrada para o bloco básico *carry lookahead* dos bits mais significativos e o *carry* de saída do somador propriamente dito. Para um CLA de 16 bits, o bloco BGP gera 4 *carries*, e assim sucessivamente.

A Fig. 50 mostra um bloco BGP para um CLA de 8 bits. Como pode ser visto através das Figs. 48 e 50, a lógica do bloco BGP cresce de acordo com o número de bits dos operandos de entrada.

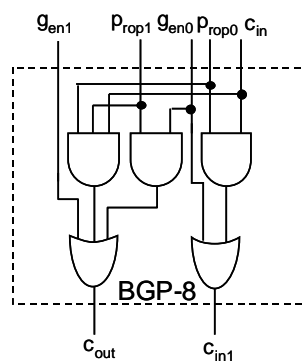


Figura 50 – Bloco *Generate-Propagate* para 8 bits

A Fig. 51 mostra como foi descrito o CLA de 8 bits, utilizando dois blocos básicos *carry lookahead* de 4 bits. Os demais somadores não-protegidos de 16, 32, 64 e 128 bits foram descritos da mesma forma, sempre através da instanciação do bloco básico *carry lookahead* de 4 bits.

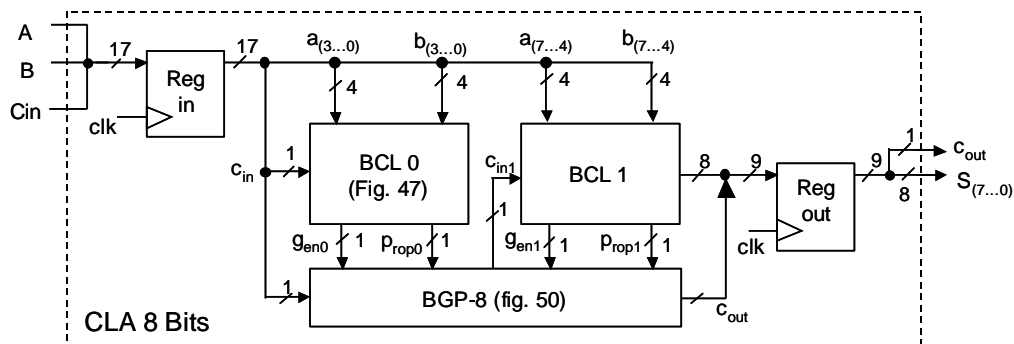


Figura 51 – Somador *Carry Lookahead* de 8 bits não protegido

Os somadores *Ripple Carry* (RCA) não-protegidos foram descritos da mesma maneira que os somadores *Carry Lookahead*, ou seja, utilizando o bloco somador e dois registradores, um de entrada e outro de saída. A diferença é que os blocos dos somadores RCAs não foram descritos utilizando hierarquia, ou seja, não existe um bloco básico a partir do qual são construídos os RCAs de mais bits. A Fig. 52 mostra o RCA de 4 bits.

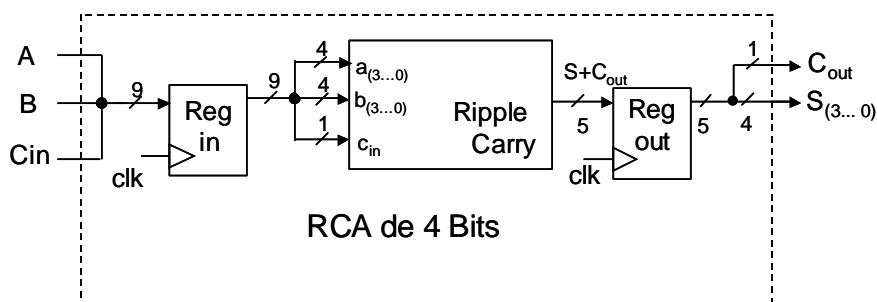


Figura 52 – Somador *Ripple Carry* de 4 bits não protegido

A técnica de proteção TMR é a aplicação pura da redundância de *hardware* estática explicada no capítulo 5.

Os somadores protegidos com TMR foram descritos, como em todas as arquiteturas implementadas, contendo dois registradores, um na entrada e outro na saída do circuito.

No caso, por exemplo, de um somador de 4 bits, na entrada foi utilizado um registrador de 9 bits para o armazenamento dos operandos e do *carry* de entrada. Já na saída foi utilizado um registrador de 5 bits, sendo armazenados o resultado da soma e o *carry* de saída. Isso foi realizado tanto para o RCA como também para o CLA

Como mencionado no capítulo 5, a técnica TMR utiliza três exemplares do bloco a ser protegido e um circuito votador por maioria (*majority voter*) para a eleição do resultado correto. Logo, no caso do somador CLA de 4 bits, o elemento triplicado é formado pelo bloco básico *carry lookahead* e pelo bloco BGP.

A Fig. 53 mostra como foi descrito o circuito de 4 bits que é triplicado para a geração de um CLA TMR de 4 bits.

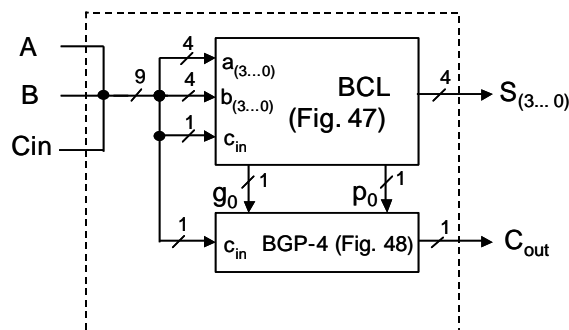


Figura 53 – Circuito que é triplicado para a obtenção do CLA TMR de 4 bits

O circuito Votador, tanto para o RCA como para o CLA, é descrito através de instâncias de um circuito votador de 1 bit. A Fig. 54 mostra o circuito votador utilizado para a descrição das arquiteturas protegidas de 4 bits.

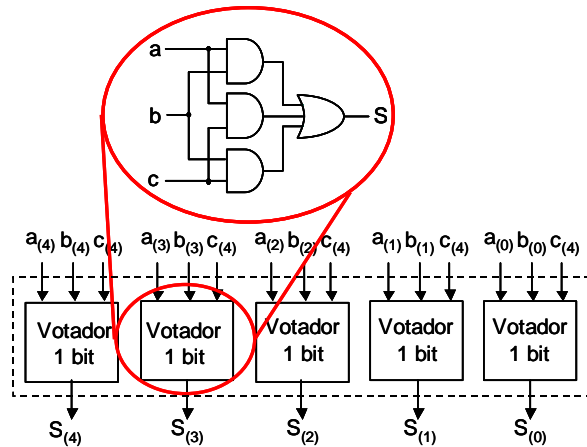


Figura 54 – Circuito votador para os somadores protegidos de 4 bits

Pode ser visto na Fig. 54 que o votador para os somadores protegidos de 4 bits utiliza cinco instâncias do votador de 1 bit. Isso ocorre para que, além dos quatro bits do resultado, o bit de *carry* também seja votado.

A Fig. 55 mostra como foi descrito um CLA TMR de 4 bits.

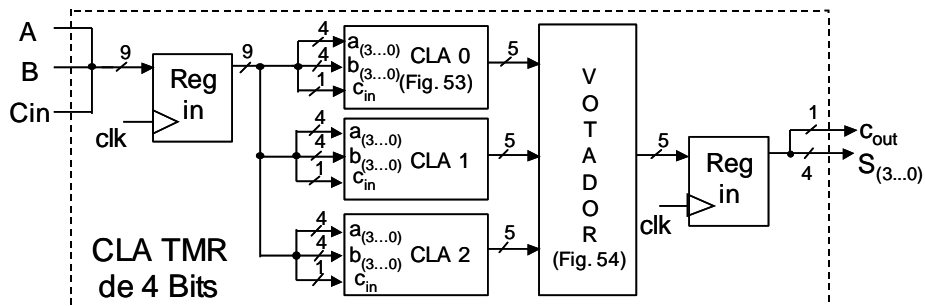


Figura 55 – Somador *Carry Lookahead* de 4 bits com TMR

A diferença para os CLAs de mais alta ordem reside no bloco triplicado, onde o bloco básico *carry lookahead* é instanciado várias vezes para a composição de somadores de mais bits, conforme mencionado anteriormente.

A Fig. 56 mostra o elemento triplicado em um CLA TMR de 8 bits, composto por dois blocos básicos *carry lookahead* e o bloco BGP que gera 2 *carries*, um *carry* de entrada do bloco básico *carry lookahead* que calcula a soma dos bits mais significativos e o *carry* de saída do CLA de 8 bits.

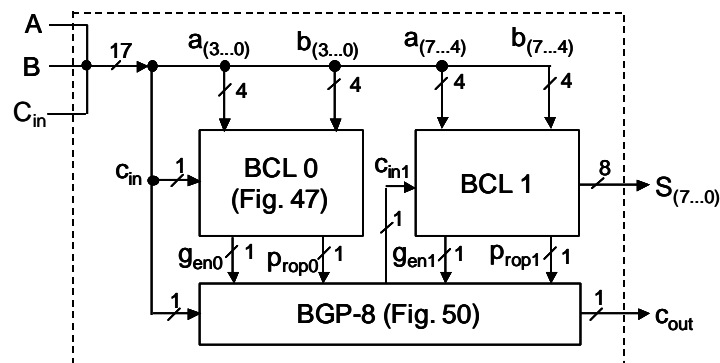


Figura 56 – Circuito triplicado para a geração de um CLA TMR de 8 bits

No caso do RCA, não foi utilizada hierarquia, ou seja, para a construção de um somador RCA TMR de 8 bits foram instanciados três blocos simples RCA de 8 bits. Já para um RCA TMR de 16 bits, foram instanciados três RCAs de 16 bits, e de maneira semelhante para os demais somadores com um número maior de bits.

A Fig. 57 mostra como foi descrito um RCA TMR de 8 bits. Os RCAs TMR com mais bits foram descritos da mesma forma.

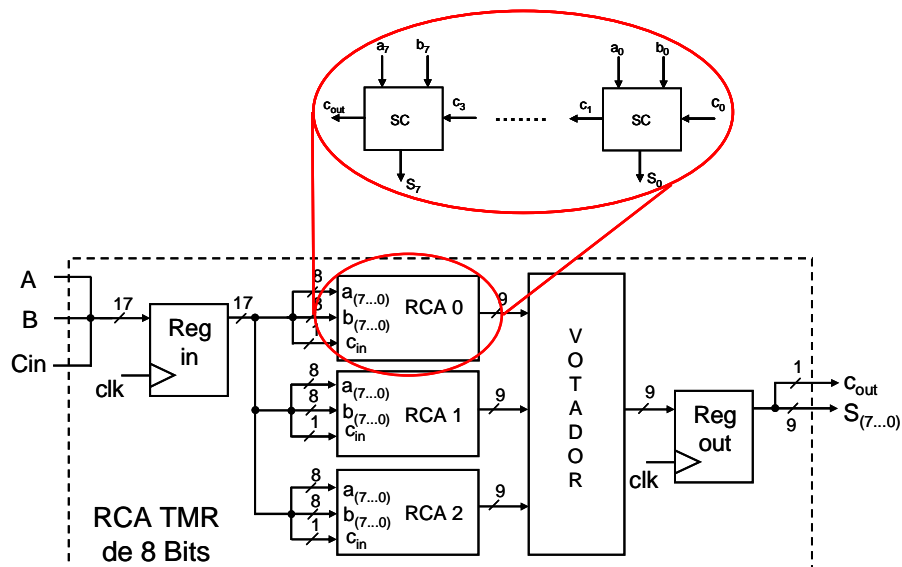


Figura 57 – RCA TMR de 8 bits

A técnica de proteção TR é a aplicação da redundância de tempo pura. Para isso o bloco somador não sofreu replicação. Porém, houve a necessidade da utilização de recursos extras para o armazenamento do resultado em diferentes tempos.

Conforme já mencionado, as arquiteturas protegidas com TR também foram descritas utilizando dois registradores, um de entrada e outro de saída.

Porém, esta técnica de proteção exige a utilização de mais registradores, para o armazenamento de amostras do resultado em tempos diferentes. Assim, o resultado do cálculo é armazenado em três tempos diferentes para que o votador possa votar o resultado correto mesmo na presença de uma falha.

A Fig. 58 mostra como foi descrito o circuito composto pelo somador CLA de 4 bits, pelos três registradores que armazenam o resultado em diferentes tempos e pelo votador.

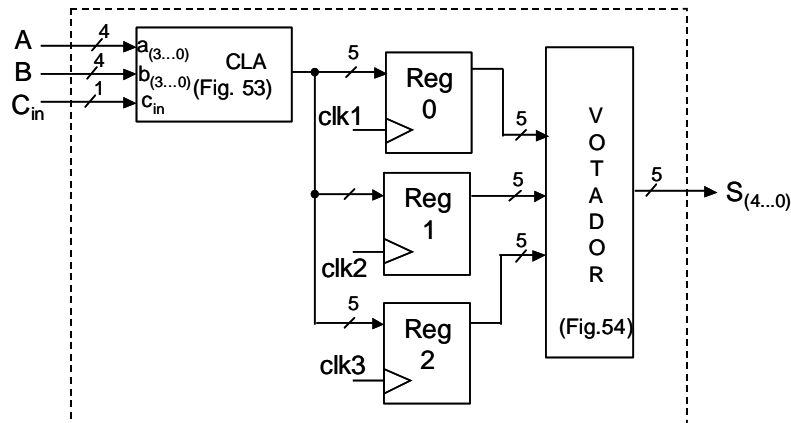


Figura 58 – Somador CLA de 4 bits juntamente com três registradores e votador

Para ser possível o armazenamento do cálculo em três tempos diferentes foi descrito um bloco de controle que faz com que, em cada ciclo de relógio, o resultado seja armazenado por um registrador diferente. Este bloco de controle gera 4 sinais de saída, clk0, clk1, clk2 e clk3 a partir de um sinal de relógio mestre (clk). Os sinais gerados são usados para controlar os registradores de entrada e saída, e os registradores reg0, reg1, e reg2. A Fig. 59 mostra a máquina de estados juntamente com o diagrama dos quatro sinais de saída gerados por esta.

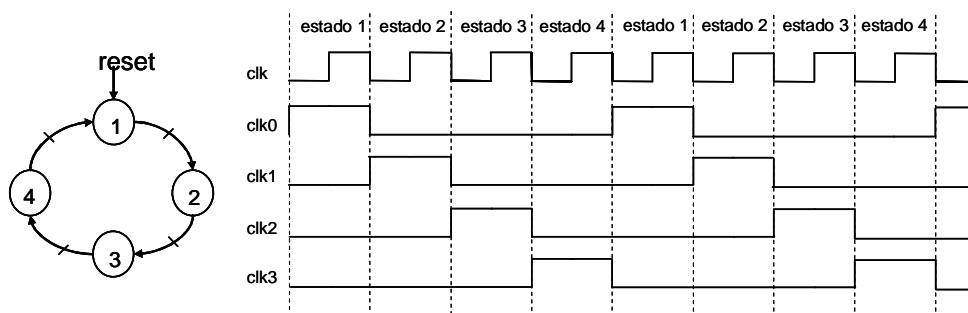


Figura 59 – Máquina de estados descrita para a técnica TR e sinais gerados

A Fig. 60 mostra a organização de um somador CLA de 4 bits protegido com TR, incluindo as conexões entre o bloco de controle e os demais elementos já descritos anteriormente.

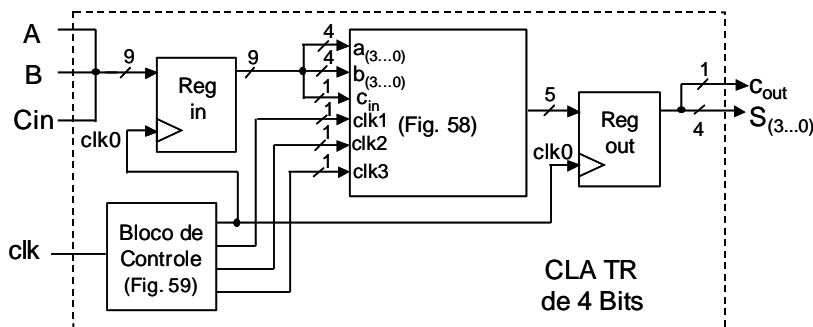


Figura 60 – Somador *Carry Lookahead* de 4 bits protegido com TR

Os somadores CLA de mais bits protegidos com TR foram descritos da mesma forma, utilizando, a partir de 8 bits, instâncias do bloco básico *carry lookahead* de 4 bits.

Os somadores RCA protegidos com TR foram descritos da mesma maneira, ou seja, um bloco somador juntamente com três registradores, que foram unidos com um bloco de controle e com o votador para a obtenção da técnica. Porém, a diferença dos somadores RCA é que estes não foram descritos utilizando hierarquia.

A técnica de proteção por Duplicação com Comparação (*DWC - Duplication with Comparison*) combinada com redundância temporal (*Time Redundancy*) - *DWC+TR* consiste na duplicação do bloco somador e no armazenamento, em dois tempos diferentes, de duas amostras do resultado calculado por cada bloco. Ou seja, a saída de cada bloco somador serve de entrada para dois registradores.

Desta forma, são obtidas quatro amostras do resultado, que servem como entrada para o Circuito Detector de Erros. A técnica também utiliza um multiplexador e um votador.

O Circuito Detector de Erros foi descrito utilizando-se instâncias de um Circuito Detector de Erros para um bit, conforme mostra a Fig. 61.

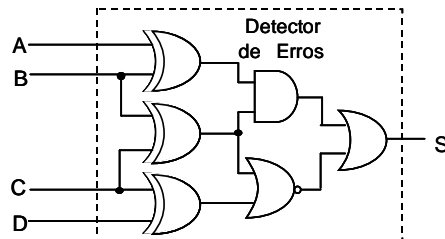


Figura 61 – Detector de Erros de 1 bit

A fig 62 mostra como foi descrito um Circuito Detector de Erros utilizado para proteger somadores de 4 bits.

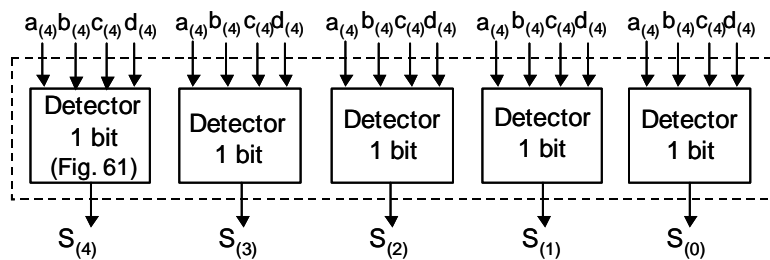


Figura 62 – Circuito Detector de Erros para somadores de 4 bits

Pode ser visto na Fig. 62 que o Detector de Erros para os somadores de 4 bits utilizará cinco instâncias do Detector de Erros de 1 bit. Isso ocorre para serem detectados erros também no bit de *carry* do resultado. Os Detectores de Erros utilizados pelas arquiteturas de mais bits foram descritos da mesma forma, utilizando instâncias do Detector de Erros de 1 bit.

Primeiramente, como pode ser visto na Fig. 63, foi descrita a duplicação dos somadores, juntamente com os registradores responsáveis pelo armazenamento das quatro amostras do resultado e com o votador. Também foi utilizado o multiplexador que escolhe a terceira entrada do votador.

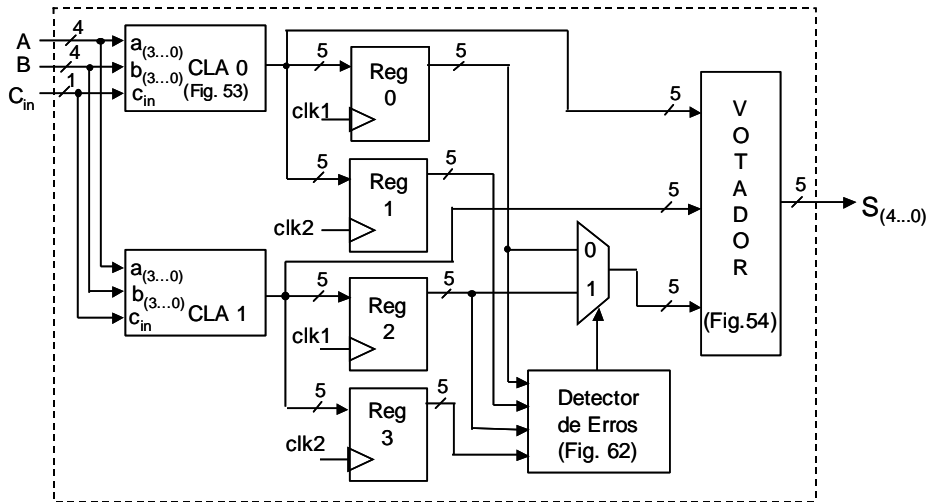


Figura 63 – Circuito CLA DWC+TR de 4 bits intermediário

Para ser descrito o somador CLA DWC+TR de 4 bits propriamente dito, ao circuito mostrado na Fig. 63 foram acrescentados os registradores de entrada e saída do somador e o bloco de controle. Porém, o bloco de controle descrito para a técnica DWC+TR gera apenas três sinais de saída: um para os registradores de entrada e saída, um segundo para os registradores 0 e 2 e um terceiro para os registradores 1 e 3. A Fig. 64 mostra a Máquina de estados juntamente com o diagrama de sinais de saída por esta.

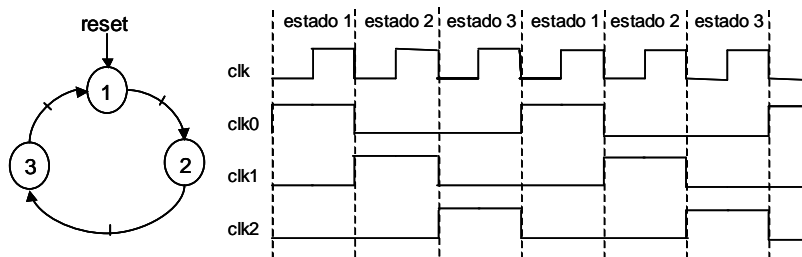


Figura 64 – Máquina de estados descrita para a técnica DWC+TR e sinais de controle

A Fig. 65 mostra como foi descrita a união de todos os circuitos mostrados anteriormente, para a geração do CLA DWC de 4 bits.

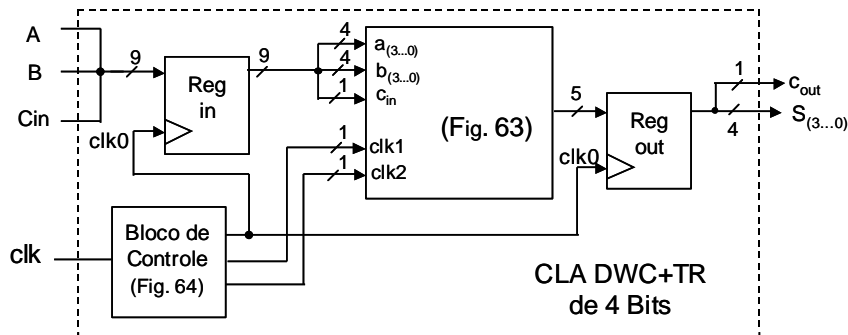


Figura 65 – Somador DWC+TR de 4 bits

Os demais somadores CLA com DWC+TR foram descritos da mesma forma, utilizando, a partir de 8 bits, instâncias do bloco básico *carry lookahead* de 4 bits.

Os somadores RCA protegidos com DWC+TR também foram descritos conforme explicado anteriormente, ou seja, utilizando dois blocos somadores cujas saídas serviram de entrada para dois registradores, juntamente com um bloco Detector de Erros e um votador. A este circuito foram acrescentados o bloco de controle e os registradores de entrada e saída. Porém, a diferença dos somadores RCA com DWC+TR é que os blocos somadores RCAs não foram descritos utilizando hierarquia, conforme já mencionado.

6.2 Diretivas de síntese da linguagem VHDL

Para todas as arquiteturas protegidas descritas na seção anterior foi necessário o uso de algum tipo de redundância para a realização da proteção. Porém, a redundância de *Hardware* gerou uma grande dificuldade, pois para a ferramenta de síntese do Quartus II, circuitos idênticos, com as mesmas entradas, são desnecessários, uma vez que produzirão o mesmo resultado. Logo, a ferramenta realiza simplificações que retiram a redundância de *hardware* que se desejava manter.

Para solucionar este problema e impedir a remoção da lógica redundante foram utilizadas algumas diretivas de síntese da linguagem VHDL, conforme proposto por Jasinski (2004) O atributo *syn_keep* indica ao compilador que um determinado sinal deve ser mantido na compilação durante os processos de otimização.

Considerando um somador CLA TMR de 4 bits, os valores para número de ALUTs utilizadas e para atraso crítico do somador foram, respectivamente, 9 e 1,629 ns sem a utilização da diretiva. Com a utilização da diretiva *syn_keep*, os resultados para o número de ALUTs e para atraso crítico foram alterados para, 169 e 5,75 ns, respectivamente.

A Fig. 66 mostra um trecho de código VHDL com o uso da diretiva *syn_keep*.

```
18  end add4cla;
19
20  architecture behavior of add4cla is
21
22      attribute syn_keep : boolean;
23      |
24      signal c0, c1, c2 : std_logic;
25
26      attribute syn_keep of c0: signal is true;
27      attribute syn_keep of c1: signal is true;
28      attribute syn_keep of c2: signal is true;
29
```

Figura 66 – Código VHDL com o uso da diretiva *syn_keep*

6.3 Comparação entre os Somadores RCA e CLA não-protégidos

Antes de realizar a análise das técnicas de proteção escolhidas para serem investigadas, é conveniente que se analise os somadores não-protégidos RCA e CLA, quando implementados em PFGAs Stratix II da Altera. O gráfico da Fig. 67 apresenta o atraso crítico dos somadores CLA e RCA não-protégidos de 4, 8, 16, 32, 64 e 128 bits.

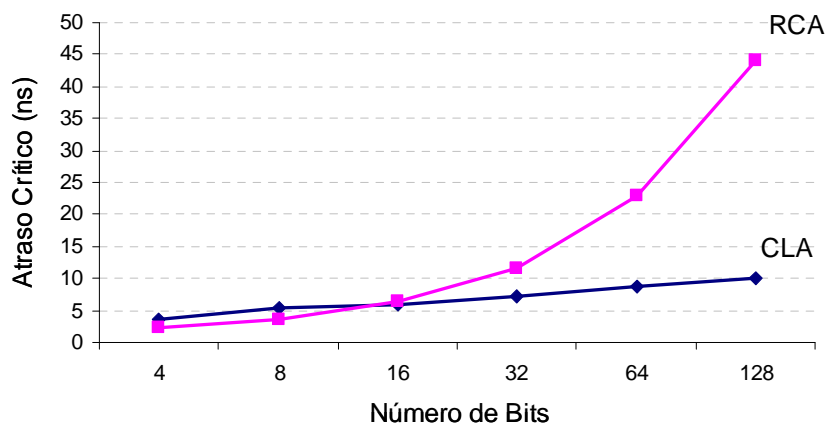


Figura 67 – Atraso crítico dos somadores não-protegidos

Pode ser visto na Fig. 67 que o atraso crítico do somador CLA possui um comportamento praticamente linear, ou seja, com o aumento do número de bits, o atraso crítico do somador CLA tem um acréscimo aproximadamente constante.

Já o atraso crítico do somador RCA apresenta um comportamento não linear, ou seja, com o aumento do número de bits, o somador RCA possui um acréscimo não proporcional em seu atraso crítico.

Considerando os somadores de 4 e 8 bits, pode ser visto que o somador RCA apresentou um melhor desempenho que o somador CLA. A partir de 16 bits, o RCA passou a apresentar um desempenho inferior ao do CLA. Isso acontece devido ao fato de que no somador do tipo RCA o *carry* se propaga de bit em bit, já que existe uma dependência entre os estágios de cálculo da soma.

Assim, o aumento de bits da soma resulta em um aumento do atraso de propagação do *carry* que é proporcional ao número de estágios, pois existirão mais estágios por onde o *carry* devará se propagar até o cálculo do bit mais significativo da soma.

Já no somador do tipo CLA, os *carries* são calculados em paralelo, dentro do bloco básico *carry lookahead* de 4 bits, de maneira que os estágios não são

dependentes entre si. A única dependência que existe está relacionada aos blocos básicos *carry lookahead* de 4 bits, quando usada hierarquia.

Em relação aos recursos do FPGA utilizados pelos somadores, acontece o contrário: o CLA apresentou valores mais elevados que os do RCA. Porém, a quantidade de recursos necessários para a implementação de ambos os somadores aumenta de maneira não-linear. O gráfico da Fig. 68 apresenta o número de ALUTs utilizadas na implementação dos somadores CLA e RCA não-protégidos.

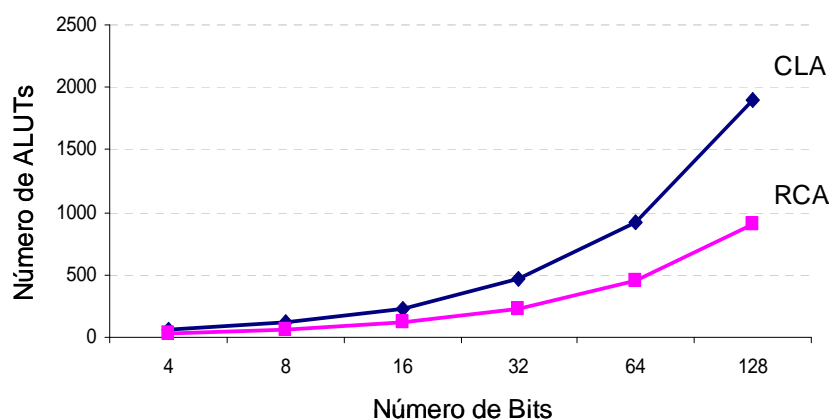


Figura 68 – Recursos utilizados pelos somadores CLA e RCA

Observando-se as curvas da Fig. 68 nota-se que os somadores do tipo RCA apresentam curvas não-lineares tanto para atraso crítico, como para número de ALUTs. Com isso comprova-se na prática a vantagem dos somadores CLA sobre os somadores RCA.

A Fig. 68 mostra que os somadores CLA necessitam de uma quantidade de ALUTs consideravelmente maior que os somadores RCA, como era esperado. Isso se deve ao fato do CLA possuir uma lógica extra para o cálculo simultâneo dos *carries* de cada estágio da soma, como mencionado no capítulo 4.

A Tab. 4 mostra os resultados de atraso crítico e de número de ALUTs para os somadores CLA e RCA. Também são mostrados os valores percentuais de redução do atraso crítico e de acréscimo de recursos, tomando-se o RCA como referência.

Tabela 4 – Comparação do atraso crítico e recursos utilizados entre os somadores CLA e RCA não-protegidos.

Bits	Atraso Crítico-ns			ALUTs		
	RCA (1)	CLA (2)	Redução de atraso $[(1)-(2)/(1)]*100$	RCA (1)	CLA (2)	Acréscimo de recursos $[(2)/(1)-1]*100\%$
4	2,28	3,63	-59%	33	63	+91%
8	3,60	5,48	-52%	61	120	+97%
16	6,34	5,94	+6%	117	230	+97%
32	11,59	7,29	+37%	229	463	+102%
64	22,94	8,88	+61%	453	920	+103%
128	44,15	10,00	+77%	901	1.905	+111%

Considerando os somadores de 4 e 8 bits, o CLA apresentou um atraso crítico 56% maior, em média, que o atraso crítico do RCA. Embora possa surpreender em uma primeira análise, este comportamento é normal, visto que os somadores CLA são compostos pelos blocos básicos *carry lookahead* de 4 bits, os quais possuem um caminho crítico maior que o caminho crítico do RCA de 4 bits.

Porém, a partir de 16 bits a aceleração provida pelos cálculos dos *carries* em paralelo começa a ter impacto no caminho crítico dos somadores CLA, de modo que estes passam a ser muito mais rápidos que os RCAs. Portanto, a partir de 16 bits, o CLA passou a apresentar um atraso crítico menor que o RCA, variando entre 6% (para 16 bits) a 77% (para 128 bits).

Em relação aos recursos utilizados pelo RCA e pelo CLA, a Tab. 4 mostra que o acréscimo percentual de recursos foi quase constante, sendo que o CLA utiliza, em média, 100% mais recursos que o RCA. Note-se que o melhor caso foi apresentado pelo CLA de 4 bits, que utilizou 91% mais ALUTs que o RCA de 4 bits. Já o CLA de 128 bits apresentou o pior caso, tendo utilizado 111% mais ALUTs que o RCA de 128 bits.

6.4 Comparação entre os somadores Protegidos com TMR e Não-Protegidos

O gráfico da Fig. 69 apresenta o atraso crítico dos somadores CLA e RCA não-protegidos, juntamente com o atraso crítico que estes apresentaram, quando protegidos com TMR.

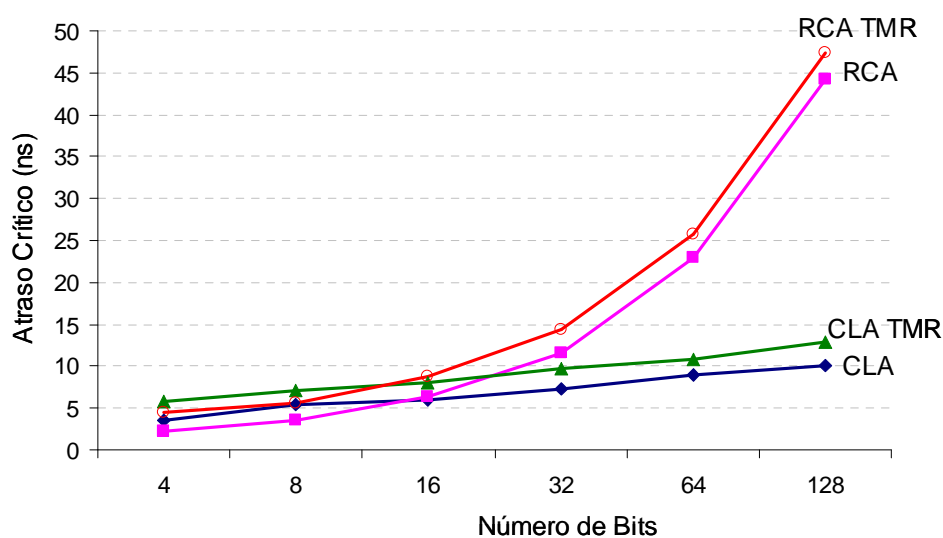


Figura 69 – Atraso crítico dos somadores protegidos com TMR e não-protegidos.

Como pode ser visto na Fig. 69, as tendências dos somadores CLA e RCA mostradas na seção 6.3 foram mantidas, quando ambos somadores são protegidos com TMR. Isto é, o atraso crítico do CLA TMR aumenta de forma quase linear, enquanto que o atraso do RCA TMR aumenta de forma não-linear.

Como era esperado, os somadores protegidos com TMR apresentaram atrasos críticos maiores que suas respectivas versões não protegidas. Pode ser visto no gráfico da Fig. 69 e também na Tab. 5 que este acréscimo no atraso crítico ocasionado pela aplicação da técnica TMR é praticamente constante.

Tabela 5 – Acréscimo do atraso crítico gerado pela técnica TMR para somadores CLA e RCA.

Bits	RCA (1) (ns)	RCA TMR (2) (ns)	Dif. entre atrasos (2)-(1) (ns)	CLA (1) (ns)	CLA TMR (2) (ns)	Dif. entre atrasos (2)-(1) (ns)
4	2,28	4,50	2,22	3,63	5,75	2,12
8	3,60	5,53	1,93	5,48	7,06	1,58
16	6,34	8,70	2,36	5,94	8,03	2,09
32	11,59	14,36	2,77	7,29	9,62	2,33
64	22,94	25,71	2,77	8,88	10,91	2,03
128	44,15	47,45	3,30	10,00	12,84	2,84

Isto se deve ao fato da utilização do circuito votador na técnica TMR inserir uma queda no desempenho que é praticamente igual para todas as arquiteturas. Apesar do bloco somador ser triplicado, os três resultados são gerados em paralelo. Logo, o atraso crítico dos somadores protegidos com TMR será igual ao atraso dos somadores não-protegidos, mais o atraso do votador.

Semelhantemente ao que ocorreu com as versões não protegidas, o RCA TMR apresentou um desempenho melhor que o CLA TMR para os somadores de 4 e 8 bits. Conforme o número de bits aumenta, ocorreu uma degradação no desempenho do RCA. Isto aconteceu tanto para os somadores não-protegidos como também para os somadores protegidos com TMR.

A Tab. 6 apresenta, de maneira detalhada, os resultados referentes ao atraso crítico dos somadores não-protegidos e protegidos através da aplicação da técnica TMR, bem como os resultados da comparação entre as versões não protegidas com as versões protegidas.

A Tab. 6 mostra que o somador RCA apresentou um acréscimo médio de 39% no atraso, ao passo que o somador CLA apresentou um acréscimo médio de 34% com a aplicação da técnica de proteção TMR. Logo, os somadores protegidos com TMR ficaram mais lentos, apresentando um atraso crítico, em média, 36% maior que o atraso crítico dos somadores não-protegidos.

Tabela 6 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e não-protegidos.

Bits	Não-protegido (1)		TMR (2)		Acréscimo de Atraso [[2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	2,28	3,63	4,50	5,75	+98%	+59%
8	3,60	5,48	5,53	7,06	+54%	+29%
16	6,34	5,94	8,70	8,03	+37%	+35%
32	11,59	7,29	14,36	9,62	+24%	+32%
64	22,94	8,88	25,71	10,91	+12%	+23%
128	44,15	10,00	47,45	12,84	+7%	+28%

No caso do RCA, conforme o número de bits dos somadores aumenta, a diferença entre o desempenho do somador não-protegido e o desempenho do protegido com TMR diminui. Isso se deve ao fato do atraso inserido pelo votador tornar-se menos importante diante do acréscimo do atraso crítico para somadores de mais bits. Assim, o atraso crítico do RCA TMR de 4 bits apresentou a maior diferença em relação ao RCA de 4 bits, sendo 98% maior. Já os somadores de 128 bits apresentaram a menor diferença, ficando o atraso crítico do RCA TMR 7% maior que o RCA não protegido.

No caso do CLA, o somador de 4 bits apresentou a maior queda no desempenho, sendo o atraso do CLA TMR 59 % maior que o atraso do CLA não-protegido. Já o somador CLA TMR de 64 bits apresentou a menor queda de desempenho em relação ao CLA não-protegido com o mesmo número de bits, apresentando um aumento no atraso crítico de 23%.

Em relação aos somadores protegidos, o RCA TMR apresentou um melhor desempenho que o CLA TMR para 4 e 8 bits, sendo o atraso crítico do CLA TMR 28% maior que o atraso do RCA TMR, para ambos os casos.

A partir de 16 bits, o atraso crítico do RCA TMR passou a crescer consideravelmente em relação ao atraso crítico do CLA TMR. O acréscimo de atraso variou entre 8% (para 16 bits) e 270% (para 128 bits). Isso se deve ao fato do CLA ser um somador rápido.

O gráfico da Fig. 70 apresenta o número de ALUTs utilizadas pelos somadores protegidos através da aplicação da técnica TMR e pelos somadores não-protegidos.

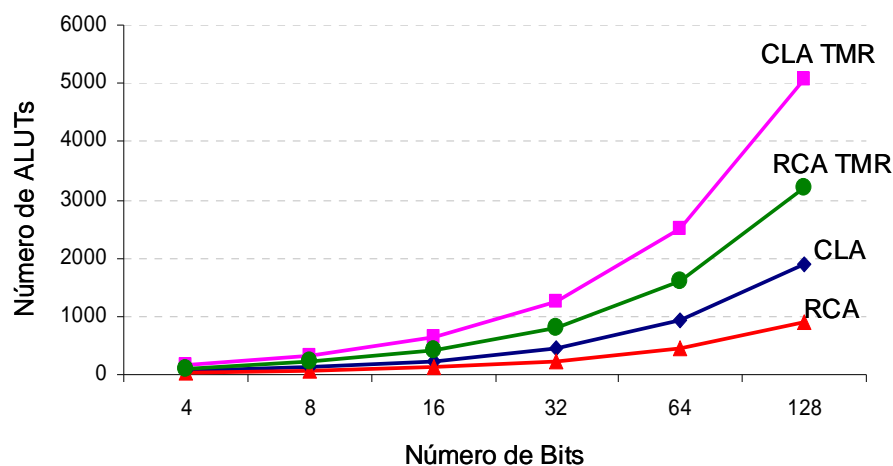


Figura 70 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e não-protegidos

Ao serem protegidos com TMR, os somadores continuam apresentando um acréscimo de recursos não-linear, e utilizam mais recursos do FPGA que os não-protegidos, o que era esperado, já que a técnica consiste na triplicação do bloco que se deseja proteger, neste caso, o somador, e no uso do votador

Como mencionado anteriormente, o somador CLA não-protegido utilizou mais recursos que o RCA não-protegido. Logo, era esperado que o CLA TMR demandasse mais recursos que o RCA TMR.

A Tab. 7 mostra os resultados referentes aos recursos do FPGA utilizados pelos somadores não-protegidos e protegidos com TMR, bem como o acréscimo de recursos necessário para as versões TMR.

Tabela 7 – Comparação entre os recursos utilizados pelos somadores protegidos com TMR e não-protegidos.

Não-protegido (1)			TMR (2)		Acréscimo de Recursos [(2)/(1)-1]*100 %	
Bits	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	33	63	104	169	+215%	+168%
8	61	120	216	327	+254%	+173%
16	117	230	416	637	+256%	+177%
32	229	463	816	1.260	+256%	+172%
64	453	920	1.616	2.504	+257%	+172%
128	901	1.905	3.216	5.054	+257%	+165%

Analisando-se a Tab. 7, percebe-se que para ambos tipos de somadores, a aplicação da técnica TMR gerou um acréscimo no número de ALUTs utilizadas, sendo que o RCA TMR utilizou, em média, 249% mais ALUTs que o RCA, ao passo que o CLA TMR utilizou, em média, 171% mais ALUTs que o CLA.

No caso do somador RCA TMR, o acréscimo de recursos variou entre 215% (para 4 bits) e 257% (para 64 e 128bits). Já no caso do CLA TMR, a arquitetura de 16 bits apresentou o maior acréscimo de recursos, sendo 177% maior que o somador CLA de 16 bits. Por outro lado, o menor acréscimo ocorreu para o caso do CLA TMR de 128 bits, com 165%.

Em relação apenas às arquiteturas protegidas, o somador CLA utilizou mais recursos que o RCA para todos os casos, o que era esperado, já que para as versões não protegidas o CLA também necessitou mais recursos que o RCA. O somador que apresentou o maior acréscimo de recursos foi o CLA TMR de 4 bits, que utilizou 63% mais ALUTs que o RCA TMR de 4 bits. Além disso, para os somadores de mais bits, o CLA TMR demandou, em média, 54% mais recursos que o RCA TMR.

Através da análise dos resultados pode-se concluir que a aplicação da técnica TMR gerou um aumento médio de 210% no uso de recursos pelos somadores protegidos em relação aos não-protegidos, não gerando, porém, um decréscimo significativo em termos de desempenho. Logo, TMR pode ser considerada uma boa técnica de proteção a ser utilizada em aplicações onde o tempo é um fator importante.

6.5 Comparação entre os somadores protegidos com TR e Não-Protegidos

O gráfico da Fig. 71 apresenta o atraso crítico dos somadores CLA e RCA não-protegidos, juntamente com o atraso crítico que estes apresentaram quando protegidos com TR.

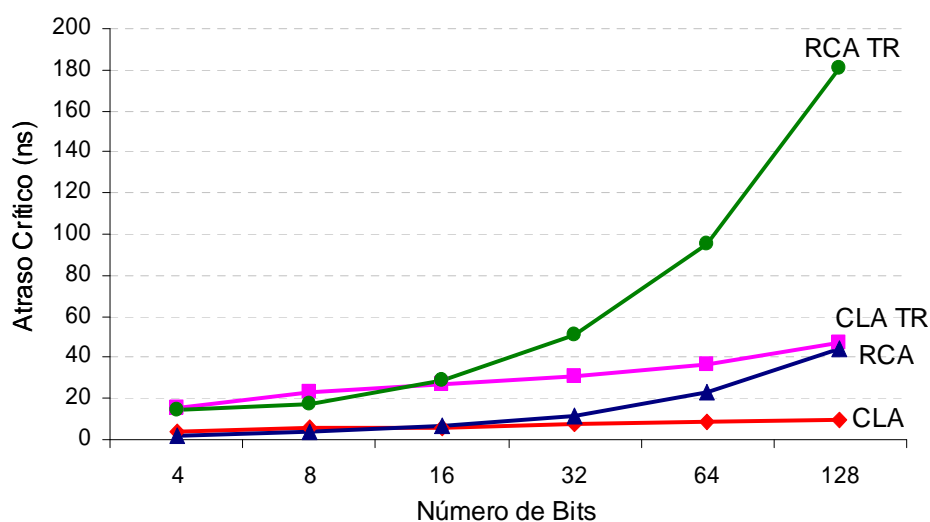


Figura 71 – Atraso crítico dos somadores protegidos com TR e não-protegidos

Analisando-se a aplicação da técnica TR, pode-se perceber que os somadores apresentam tendências similares aos somadores não-protegidos e protegidos com TMR, conforme mostrado anteriormente, ou seja, o atraso do CLA TR aumenta quase linearmente, enquanto que o atraso do somador RCA TR aumenta de forma não-linear.

Uma outra característica que se mantém é que, para somadores de 4 e 8 bits, o atraso do CLA TR é maior que o atraso do RCA TR. Porém, com o aumento do número de bits, o RCA TR apresenta uma queda considerável de desempenho.

Pode-se perceber pela Fig. 71 que, caso fossem descritos somadores de mais bits, a tendência seria que o somador RCA não-protegido apresentasse um

desempenho inferior até mesmo que a versão protegida do CLA TR, o que se constitui em um dado bastante interessante para os projetistas de circuitos tolerantes a SETs.

A Tab. 8 apresenta, de maneira detalhada, os resultados referentes ao atraso crítico dos somadores não-protegidos e protegidos com TR, bem como os acréscimos de atraso das versões protegidas com TR em relação às não-protegidas.

Tabela 8 – Comparação entre os atrasos críticos dos somadores protegidos com TR e não-protegidos.

Bits	Não-protegido (1)		TR (2)		Acréscimo de Atraso [(2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	2,28	3,63	14,60	15,71	+542%	+333%
8	3,60	5,48	17,68	23,33	+392%	+326%
16	6,34	5,94	28,40	27,04	+348%	+355%
32	11,59	7,29	51,06	31,11	+341%	+327%
64	22,94	8,88	94,83	36,89	+313%	+315%
128	44,15	10,00	180,50	46,91	+309%	+369%

Como pode ser observado na Tab. 8, a técnica TR gerou uma grande queda no desempenho das arquiteturas dos somadores. Isso acontece devido ao fato desta técnica amostrar o resultado em três tempos diferentes, cada um deles com a duração igual ao atraso crítico do somador em questão.

Por ser um somador rápido, a queda do desempenho do somador CLA foi menor que a queda do RCA, tendo o atraso crítico do CLA TR ficado, em média, 337% maior que o atraso crítico do CLA não-protegido. Já o RCA TR apresentou um atraso crítico, em média, 374% maior que o atraso do RCA não-protegido.

Em se tratando do somador RCA TR, o acréscimo do atraso crítico variou entre 542% (para 4 bits) e 309% (para 128bits). Já no caso do CLA TR, não ocorreu uma variação significativa no acréscimo do atraso crítico, entre as versões do CLA protegidas com TR, tendo o CLA TR de 128 bits apresentado a maior queda de desempenho, com atraso 369% maior que o atraso do CLA não-protegido de 128 bits. Já o CLA TR de 64 bits foi o somador com a menor queda de desempenho, tendo apresentado um atraso 315% maior que o CLA de 64 bits.

Comparando-se apenas os somadores protegidos com TR, percebe-se que, para os somadores de 4 e 8 bits, o RCA apresentou um melhor desempenho que o CLA, sendo que o CLA TR de 4 bits apresentou um atraso crítico 8% maior que o RCA TR de 4 bits, e o CLA TR de 8 bits apresentou um atraso crítico 32% maior que o atraso do RCA TR de 8 bits.

A partir de 16 bits, o desempenho do RCA TR passou a ser inferior ao desempenho do CLA TR, tendo o RCA TR apresentado um atraso crítico, em média, 128% maior que o atraso crítico do CLA TR. O acréscimo de atraso crítico do RCA TR variou entre 5% (para 16 bits) e 285% (para 128 bits).

Através dos resultados conclui-se que a técnica TR gerou um acréscimo no atraso crítico dos somadores de 356%, em média. Logo, nesta técnica os somadores apresentaram um comportamento inverso ao ocorrido na técnica TMR, onde os somadores não apresentaram uma queda muito significativa de desempenho.

O gráfico da Fig. 72 apresenta o número de ALUTs utilizadas pelos somadores protegidos através da técnica TR e pelos somadores não-protetidos.

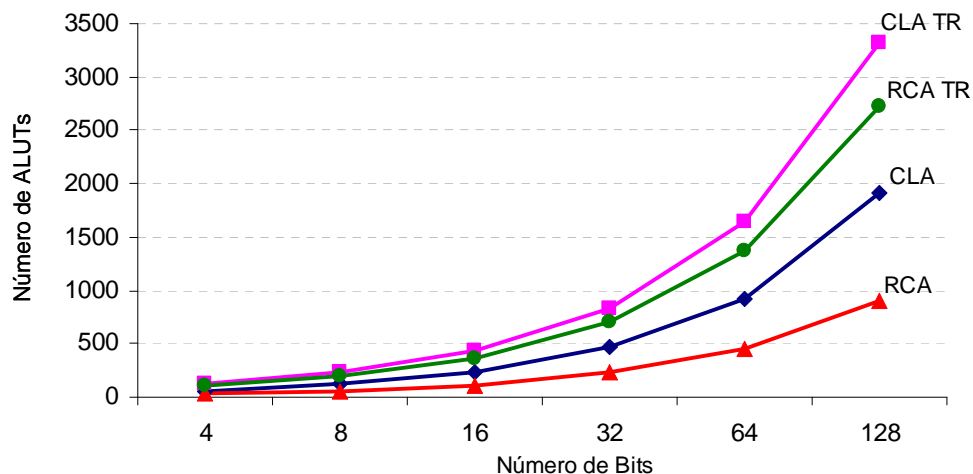


Figura 72 – Número de ALUTs utilizadas pelos somadores protegidos com TR e não-protetidos.

Novamente, os somadores mantêm a tendência apresentada anteriormente, ou seja, o acréscimo no número de ALUTs ocorre de maneira não-linear. Além disso, os

somadores protegidos utilizaram mais recursos que os não-protegidos. Isso ocorre porque mesmo a técnica TR não triplicando o *hardware*, ela utiliza recursos extras devido aos registradores necessários para o armazenamento do resultado nos três instantes diferentes. Além destes registradores extras, TR utiliza também o votador para a eleição da resposta correta. Assim como no caso da TMR, o CLA protegido com TR utilizou mais recursos do FPGA que o RCA com TR.

A Tab. 9 apresenta detalhadamente os resultados da utilização de recursos do FPGA pelos somadores protegidos com TR e não-protegidos, juntamente com os acréscimos percentuais de ALUTs gerados pela aplicação desta técnica.

Tabela 9 – Comparação entre os recursos utilizados pelos somadores protegidos com TR e não-protegidos

Bits	Não-protegido (1)		TR (2)		Acréscimo de Recursos [[2]/(1)-1]*100 %	
	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	33	63	109	129	+230%	+105%
8	61	120	197	231	+223%	+93%
16	117	230	365	433	+212%	+88%
32	229	463	701	838	+206%	+81%
64	453	920	1.373	1.646	+203%	+79%
128	901	1.905	2.717	3.327	+202%	+75%

Como esperado, a aplicação da técnica TR também resultou em um aumento no uso de recursos, em relação aos somadores não-protegidos.

O somador RCA TR apresentou um acréscimo médio de recursos de 213% em relação ao RCA, enquanto que o somador CLA TR utilizou 87%, em média, mais recursos que o CLA. Isso mostra que a aplicação da técnica TR resultou em um acréscimo no número de ALUTs, em média, 159% maior que o número de ALUTs utilizadas pelas versões não-protegidas.

O RCA TR apresentou um acréscimo na utilização dos recursos que variou entre 230% (para 4 bits) e 202% (para 128 bits). Já o CLA TR possuiu um acréscimo no número de ALUTs que variou entre 105% (para 4 bits) e 75% (para 128 bits).

Com isso, percebe-se que, conforme aumenta-se o número de bits, o acréscimo percentual de recursos diminui. Isso se dá devido ao fato dos recursos extras necessários para a aplicação da técnica se tornarem menos representativos face ao acréscimo dos recursos utilizados pelos somadores de mais bits.

Comparando-se apenas as arquiteturas protegidas, o CLA TR utilizou, em média, 19% mais recursos do que o RCA TR. O CLA TR de 128 bits é a arquitetura que utilizou mais recursos: 22% a mais que o RCA TR de 128 bits. Já o CLA TR de 8 bits apresentou o menor acréscimo em relação ao RCA TR de 8 bits, utilizando 17% mais recursos.

6.6 Comparação entre os somadores protegidos com TR e com TMR

A Fig. 73 mostra o gráfico que compara os resultados, em termos de atraso crítico, gerados pela aplicação das técnicas de proteção TMR e TR.

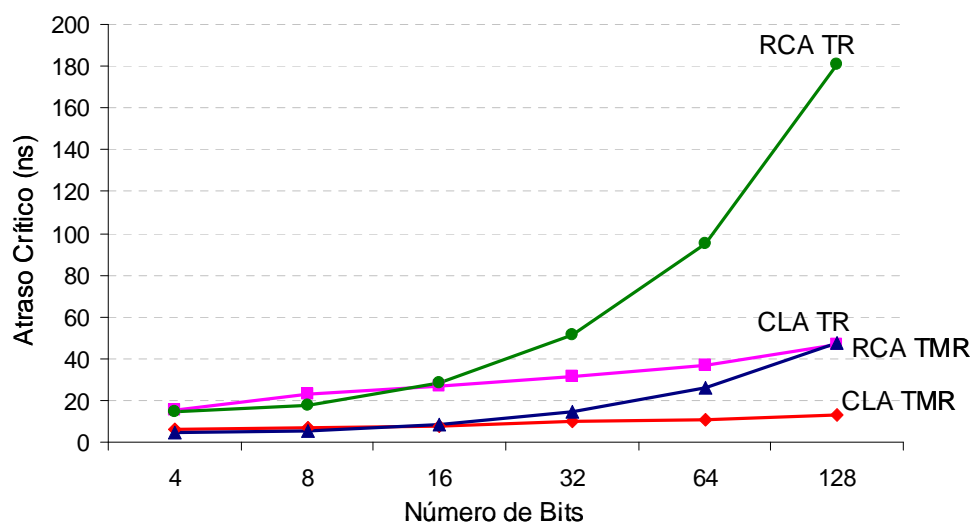


Figura 73 – Atraso crítico dos somadores protegidos com TMR e TR

Como pode ser visto na Fig. 73, a tendência dos somadores se manteve, ou seja, tanto o CLA com TMR como o CLA com TR apresentou um aumento

praticamente linear. Já o RCA possuiu um aumento não linear para ambas as técnicas comparadas, apresentando atrasos cada vez maiores, conforme aumenta-se o número de bits.

Outra característica mantida é que para somadores de 4 e 8 bits, o RCA, em ambas as técnicas, apresentou um desempenho superior ao CLA.

A partir de 16 bits ocorreu uma queda importante no desempenho do RCA, que apresentou um atraso crítico maior que o CLA, para ambas técnicas. E o CLA TMR passou a apresentar o melhor desempenho entre todos os somadores protegidos.

A Tab. 10 mostra os resultados do atraso crítico para as arquiteturas dos somadores protegidos com as técnicas TR e TMR, juntamente com os acréscimos de atraso das versões TR em relação às versões TMR.

Tabela 10 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e protegidos com TR

Bits	TMR (1)		TR (2)		Acréscimo de Atraso [(2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	4,50	5,75	14,60	15,71	+225%	+173%
8	5,53	7,06	17,68	23,33	+220%	+231%
16	8,70	8,03	28,40	27,04	+226%	+237%
32	14,36	9,62	51,06	31,11	+256%	+223%
64	25,71	10,91	94,83	36,89	+269%	+238%
128	47,45	12,84	180,50	46,91	+280%	+265%

Analisando-se a Tab. 10, percebe-se que os somadores protegidos com TRM apresentam um melhor desempenho que os protegidos com TR.

No caso do RCA TR, o atraso foi, em média, 246% maior que o atraso do RCA TMR e no caso do CLA TR, o atraso foi aproximadamente 228% maior que o CLA TMR. Logo, a técnica TR apresentou um atraso, em média, 237% maior que o atraso da técnica TMR.

O atraso do CLA TR ficou entre 173% e 265% maior que o atraso do CLA TMR, para 4 bits e 128 bits, respectivamente.

Comparando-se os somadores RCA TR e RCA TMR, a menor queda de desempenho ocorreu para os somadores de 8 bits, onde o RCA TR apresentou um atraso, em média, 220% maior que o atraso do RCA TMR. Já o RCA TR de 128 bits foi 280% maior que o RCA TMR de 128 bits, apresentando a maior queda de desempenho.

O gráfico da Fig. 74 apresenta o número de ALUTs utilizadas pelos somadores protegidos através da aplicação das técnicas TR e TMR.

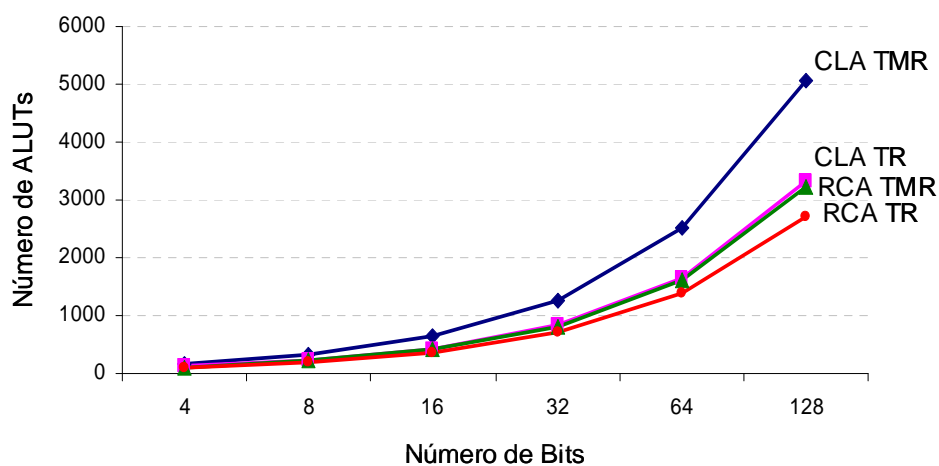


Figura 74 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e TR

Novamente, as tendências dos somadores são mantidas, ou seja, o acréscimo no número de ALUTs ocorre de maneira não-linear. Através da observação da Fig. 74 conclui-se que a técnica TMR utilizou mais recursos que a técnica TR e que, apesar da aplicação das técnicas de proteção, os somadores CLA continuam utilizando mais recursos que os somadores RCA.

Para somadores de 4 e 8 bits a utilização de recursos foi similar, ou seja, os somadores utilizaram aproximadamente a mesma quantidade de recursos. A partir de 16 bits o CLA TMR passou a utilizar mais recursos que os demais, sendo que esta diferença cresce com o aumento dos bits. Vale ainda observar que o CLA TMR foi o somador que utilizou mais recursos entre todos os somadores protegidos com TMR e TR.

Já o RCA TR é o que utilizou menos ALUTS dentre os somadores protegidos com TMR ou com TR, o que era esperado, já que o RCA não-protégido também utilizou menos recursos que o CLA não-protégido. O mesmo acontece com a técnica TR, que necessita de menos recursos que a TMR.

Os somadores que utilizaram uma quantidade intermediária de recursos do FPGA são os RCA TMR e CLA TR, tendo estes utilizado aproximadamente o mesmo número de ALUTs.

A Tab. 11 mostra os resultados dos recursos do FPGA utilizados pelas arquiteturas dos somadores protegidos com TR e com TMR, juntamente com os acréscimos de recursos das versões TMR em relação às versões TR.

Tabela 11 – Comparação entre os recursos utilizados pelos somadores protegidos com TMR e protegidos com TR

Bits	TR (1)		TMR (2)		Acréscimo de Recursos [[2]/(1)-1]*100 %	
	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	109	129	104	169	-5%	+31%
8	197	231	216	327	+10%	+42%
16	365	433	416	637	+14%	+47%
32	701	838	816	1.260	+16%	+50%
64	1.373	1.646	1.616	2.504	+18%	+52%
128	2.717	3.327	3.216	5.054	+18%	+52%

Ao contrário do que ocorre em relação ao desempenho, em se tratando dos recursos utilizados, a técnica TMR gerou um acréscimo de recursos maior do que a técnica TR.

Apenas o RCA TR de 4 bits apresentou um maior uso de recursos em relação ao RCA TMR, utilizando 5% mais recursos que o RCA TMR de 4 bits. A partir de 8 bits o RCA TMR passou a utilizar 15% mais recursos que o RCA TR, tendo este acréscimo variado entre 10% (para 16 bits) e 18% (para 128 bits).

Em relação ao somador CLA, todas as arquiteturas protegidas com TMR utilizaram mais recursos que as arquiteturas protegidas com TR, sendo que o

acréscimo de recursos destas variou entre 31% (para 4 bits) e 52% (para 64 e 128 bits).

Analisando-se os valores da Tab. 11, conclui-se que a técnica TMR utilizou, em média, 30% mais recursos do FPGA que a técnica TR.

6.7 Comparação entre os somadores protegidos com DWC+RT e Não-Protegidos

O gráfico da Fig. 75 apresenta o atraso crítico para os somadores CLA e RCA não-protegidos e para os somadores protegidos com a técnica de tolerância DWC+TR.

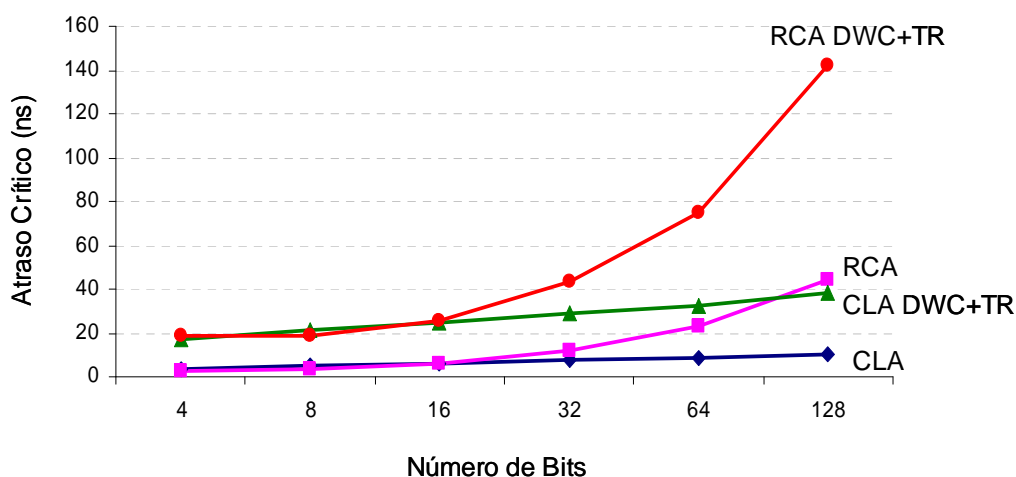


Figura 75 – Atraso crítico dos somadores protegidos com DCW+TR e não-protegidos

Como pode ser visto na Fig. 75, a tendência dos somadores mais uma vez se manteve, ou seja, tanto o CLA não-protegido como o CLA protegido com DWC+TR apresentou um aumento praticamente linear. Já os somadores RCA apresentaram um aumento não linear, tanto para o somador protegido, como para o não-protegido.

À medida que o número de bits dos somadores aumenta, o desempenho do RCA diminui, tanto para a arquitetura protegida como também para a arquitetura não

protegida, chegando até mesmo a apresentar um desempenho inferior à versão do CLA protegido com a técnica DCW+TR (para somadores de 128 bits).

Comparando-se os somadores protegidos, o CLA DWC+TR de 8 bits apresentou um desempenho inferior ao RCA DWC+TR de 8 bits. Porém, à medida que o número de bits aumenta, o desempenho do RCA DWC+TR diminui consideravelmente em relação ao CLA DWC+TR e também em relação aos demais somadores.

Comparando-se todos os somadores, tanto os protegidos com DWC+TR, como os somadores não-protegidos, nota-se que, a partir de 16 bits o CLA possui melhor desempenho.

A Tab. 12 mostra os resultados detalhados do atraso crítico para os somadores protegidos com DWC+TR e somadores não-protegidos, juntamente com os acréscimos percentuais gerados pela aplicação desta técnica de proteção.

Tabela 12 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e não-protegidos

Bits	Não-protegido (1)		DWC+TR (2)		Acréscimo de Atraso [(2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	2,28	3,63	18,33	17,20	+705%	+374%
8	3,60	5,48	18,89	21,30	+425%	+289%
16	6,34	5,94	25,70	25,03	+305%	+321%
32	11,59	7,29	43,52	28,98	+275%	+297%
64	22,94	8,88	75,00	32,14	+227%	+262%
128	44,15	10,00	142,29	37,90	+222%	+279%

A aplicação da técnica DWC+TR ocasionou uma queda considerável no desempenho dos somadores. No caso do somador RCA DWC+TR, o acréscimo de atraso foi, em média, de 360% em relação ao atraso do RCA. Já o atraso do CLA DWC+TR foi, em média, 304% maior que o atraso do CLA.

Com isso conclui-se que a aplicação da técnica DWC+TR produziu um acréscimo no atraso dos somadores, em média, de 332% em relação ao atraso dos somadores não-protegidos.

Isso se deve ao fato da técnica de proteção DWC+TR possuir, além do somador e do votador, dois registradores para o armazenamento do resultado em dois tempos diferentes, um circuito detector de erros e um multiplexador para a escolha da terceira entrada do votador, no caso de um erro ser detectado. Todo este *hardware* extra faz com que o desempenho dos somadores protegidos com esta técnica apresente um decréscimo considerável de desempenho.

Em se tratando do somador RCA DWC+TR, o acréscimo do atraso crítico variou entre 705% (para 4 bits) e 222% (para 128 bits). Percebe-se que, com o aumento dos bits dos somadores, o acréscimo percentual de atraso diminui. Isso ocorre devido ao fato do atraso inserido pelos circuitos extras utilizados para a aplicação da técnica ser mais significativo para somadores de menos bits.

No caso do somador CLA DWC+TR, o acréscimo do atraso variou entre 374% (para 4 bits) e 262% (para 64 bits). Com isso percebe-se que a tendência observada no RCA DWC+TR se mantém, ou seja, à medida que o número de bits aumenta, a queda de desempenho dos somadores protegidos com DCW+TR em relação aos somadores não-protegidos diminui.

Comparando-se apenas o somadores protegidos com DWC+TR, o RCA DWC+TR apresentou um pior desempenho que o CLA DWC+TR, com exceção do somador de 8 bits, onde o atraso do CLA DWC+TR foi 13% maior que o atraso do RCA DWC+TR. Para os demais somadores, o RCA DWC+TR apresentou um atraso, em média, 94% maior que o atraso do CLA DWC+TR.

Para a arquitetura de 128 bits, o atraso do RCA DWC+TR foi 275% maior que o CLA DWC+TR de 128 bits, comprovando a tendência do RCA DWC+TR em apresentar uma queda de desempenho cada vez maior com o aumento do número de bits.

O gráfico da Fig. 76 apresenta o número de ALUTs utilizadas pelos somadores protegidos pela técnica DWC+TR e pelos somadores não-protegidos.

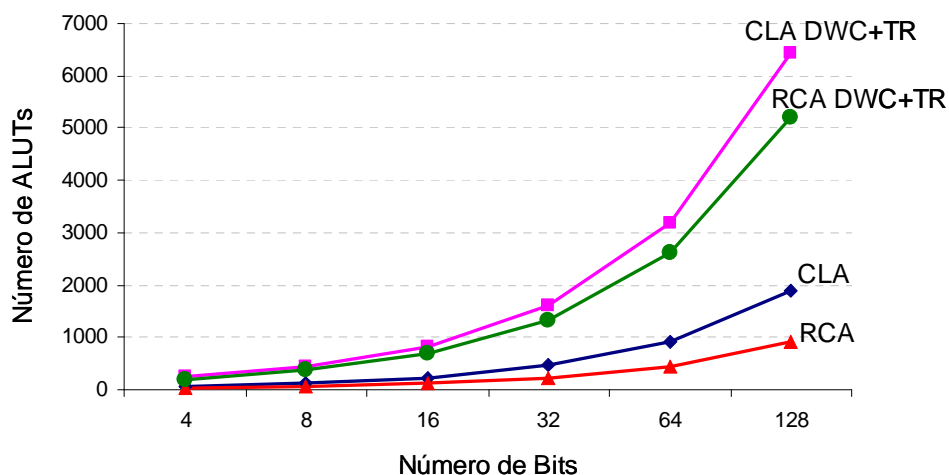


Figura 76 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e não-protegidos

Como mencionado anteriormente, a técnica DWC+TR utiliza vários circuitos extras além do somador, como os circuitos Detector de Erros e Votador. Logo, era esperado que a técnica DWC+TR aumentasse consideravelmente a utilização dos recursos do FPGA pelos somadores, e como o CLA utiliza mais recursos que o RCA, o CLA com DWC+TR também utiliza um número maior de ALUTs que o RCA com DWC+TR.

Apesar disto, a tendência dos somadores é mantida, ou seja, o acréscimo no número de ALUTs ocorre de maneira não-linear e os somadores protegidos utilizaram mais recursos que os não-protegidos.

A Tab. 13 mostra os resultados dos recursos do FPGA utilizados pelos somadores protegidos com DWC+TR e somadores não-protegidos, juntamente com os acréscimos percentuais gerados pela aplicação da técnica de proteção DWC+TR.

Tabela 13 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e não-protegidos

Não-protegido (1)			DWC+TR (2)		Acréscimo de Recursos [[2]/(1)-1]*100 %	
Bits	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	33	63	199	237	+503%	+276%
8	61	120	368	436	+503%	+263%
16	117	230	690	826	+490%	+259%
32	229	463	1.334	1.609	+483%	+248%
64	453	920	2.622	3.171	+479%	+245%
128	901	1.905	5.199	6.418	+477%	+237%

Pode ser observado na Tab. 13 que a aplicação da técnica DWC+TR aumentou consideravelmente o uso de recursos pelos somadores, principalmente no caso do RCA, tendo o RCA DWC+TR utilizado, em média, 489% mais recursos que o RCA não-protegido. Em se tratando do CLA, os somadores protegidos com DWC+TR utilizaram 255% mais recursos que os não protegidos. Pode ser concluído, portanto, que com a aplicação da técnica DCW+RT, os somadores utilizaram, em média, 372% mais recursos que os não-protegidos.

No caso do RCA DWC+TR, o acréscimo na utilização dos recursos variou entre 503% (para 4 e 8 bits) e 477% (para 128 bits). Já o CLA DWC+TR apresentou um acréscimo no número de ALUTs que variou entre 276% (para 4 bits) e 237% (para 128 bits).

Logo, percebe-se novamente que quanto maior o número de bits dos somadores, menor o acréscimo percentual de recursos em relação aos somadores não-protegidos. Isso ocorre devido ao fato do número de ALUTs utilizadas pelos circuitos extras da técnica DWC+TR não aumentar de maneira expressiva com o aumento dos bits.

Comparando-se apenas as arquiteturas protegidas, o CLA DWC+TR utilizou 20% mais recursos que o RCA DWC+TR, sendo que o CLA DWC+TR de 8 bits apresentou o menor acréscimo, tendo utilizado 18% mais ALUTs que o RCA DWC+TR

de 8 bits e o CLA DWC+TR de 128 bits apresentou o maior acréscimo, utilizando 23% mais recursos que o RCA DWC+TR de 128 bits.

Antes de realizar a comparação da técnica DWC+TR com as demais técnicas, para uma melhor visualização dos resultados, os gráficos a seguir apresentam as curvas de atraso e número de ALUTs para todos os somadores protegidos.

O gráfico da Fig. 77 mostra o atraso crítico dos somadores CLA e RCA protegidos.

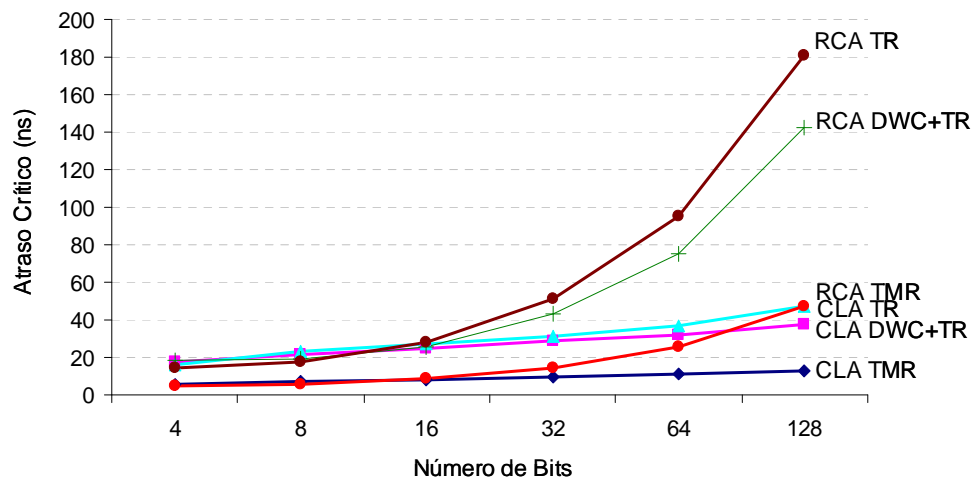


Figura 77 – Atraso crítico dos somadores protegidos

Através da Fig. 77 pode ser visto que o CLA manteve a tendência de crescimento quase linear para todos os somadores protegidos. O RCA também manteve seu aumento não-linear para todos os somadores, apresentando um crescimento considerável a partir de 16 bits.

O gráfico da Fig. 78 mostra o número de ALUTs utilizadas pelos somadores CLA e RCA protegidos.

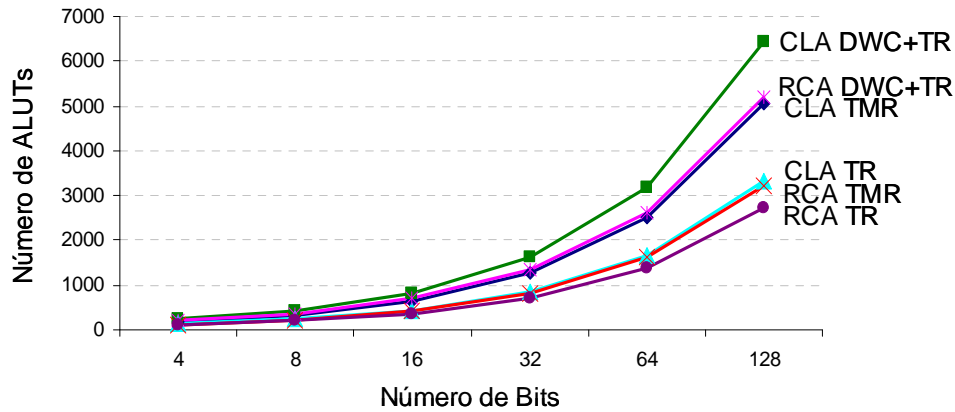


Figura 78 – Número de ALUTs utilizadas pelos somadores protegidos

Pode ser visto na Fig. 78 que o aumento do número de ALUTs utilizadas foi não-linear para todos os somadores protegidos, sendo que o CLA utilizou mais recursos que o RCA em todas as técnicas. Pode ser visto também que a técnica TR necessitou de menos recursos que as demais. Já a DWC+TR é a técnica que demandou mais recursos.

6.8 Comparação entre os somadores protegidos com DWC+RT e com TR

O gráfico da Fig. 79 apresenta o atraso crítico dos somadores protegidos com as técnicas TR e DWC+TR.

Como pode ser visto na Fig. 79, os somadores de até 16 bits apresentaram desempenhos similares para todas as versões. Porém, a tendência dos somadores se mantém, ou seja, os somadores CLA TR e CLA DWC+TR apresentam um aumento quase linear do atraso e os somadores RCA TR e RCA DCW+TR apresentam um comportamento não-linear.

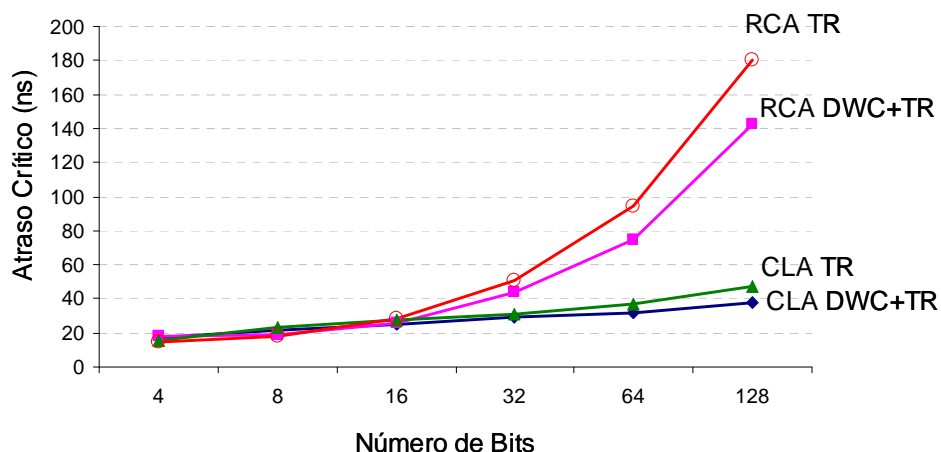


Figura 79 – Atraso crítico dos somadores protegidos com TR e DWC+TR

O RCA TR apresentou o melhor desempenho para os somadores de 4 e 8 bits. Porém, com o aumento do número de bits, este passou a apresentar, já a partir de 16 bits, o pior desempenho em relação a todos os outros somadores.

Comparando-se as técnicas DWC+TR e TR, o CLA DCW+TR, a partir de 16 bits, passou a apresentar o melhor desempenho que os demais somadores protegidos. O somador RCA apresentou, a partir de 16 bits, um desempenho inferior ao do CLA, para ambas as técnicas.

O somador CLA apresentou desempenho similar para ambas as técnicas de proteção, apenas com um desempenho inferior para a técnica TR, como era esperado, já que esta utiliza a redundância temporal pura.

A Tab. 14 mostra os resultados detalhados do atraso crítico dos somadores protegidos com DWC+TR e com TR, juntamente com os acréscimos percentuais de atraso das versões TR em relação às versões DWC+TR.

Analisando-se os valores apresentados na Tab. 14 pode-se concluir que a partir de 16 bits a técnica TR gerou uma queda no desempenho dos somadores em relação à

técnica DWC+TR, deixando o atraso crítico dos somadores com TR, em média, 17% maior que o atraso dos somadores com DWC+TR.

Tabela 14 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e com TR

Bits	DWC+TR (1)		TR (2)		Acréscimo de Atraso [(2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	18,33	17,20	14,60	15,71	-20%	-9%
8	18,89	21,30	17,68	23,33	-6%	+10%
16	25,70	25,03	28,40	27,04	+11%	+8%
32	43,52	28,98	51,06	31,11	+17%	+7%
64	75,00	32,14	94,83	36,89	+26%	+15%
128	142,29	37,90	180,50	46,91	+27%	+24%

Para somadores RCA TR este acréscimo variou entre 11% (para 16 bits) e 27% (para 128 bits). Já o CLA TR não apresentou variações crescentes do acréscimo de atraso, sendo o melhor caso apresentado pelo somador de 32 bits, onde o CLA TR apresentou um atraso 7% maior que o atraso do CLA DWC+TR. O CLA TR de 128 bits apresentou o pior caso, tendo um atraso 24% maior que o atraso do CLA DWC+TR.

Tanto para o CLA como também para o RCA, as arquiteturas de 4 bits protegidas com DWC+TR apresentaram uma queda no desempenho dos somadores maior que a queda apresentada pela aplicação da técnica TR. No caso do RCA, isso também ocorreu para a arquitetura de 8 bits.

O CLA DWC+TR de 4 bits apresentou um atraso 10% maior que o atraso do CLA TR de 4 bits e o RCA DWC+TR de 4 bits apresentou um atraso 25% maior que o atraso do RCA TR de 4 bits. Já o RCA DWC+TR de 8 bits apresentou um atraso 7% maior que o atraso do RCA TR de 8 bits.

Percebe-se, então, que com o aumento do número de bits dos somadores, o atraso das arquiteturas protegidas com TR passou a ser maior que as protegidas com DWC+TR.

A partir de 16 bits, o RCA TR ficou 20% mais lento que o RCA DWC+TR. Já no caso do CLA, a partir de 8 bits os somadores com TR ficaram 13% mais lentos que os somadores com DWC+TR.

O gráfico da Fig. 80 apresenta o número de ALUTs utilizadas pelos somadores protegidos através da aplicação das técnicas TR e DWC+TR.

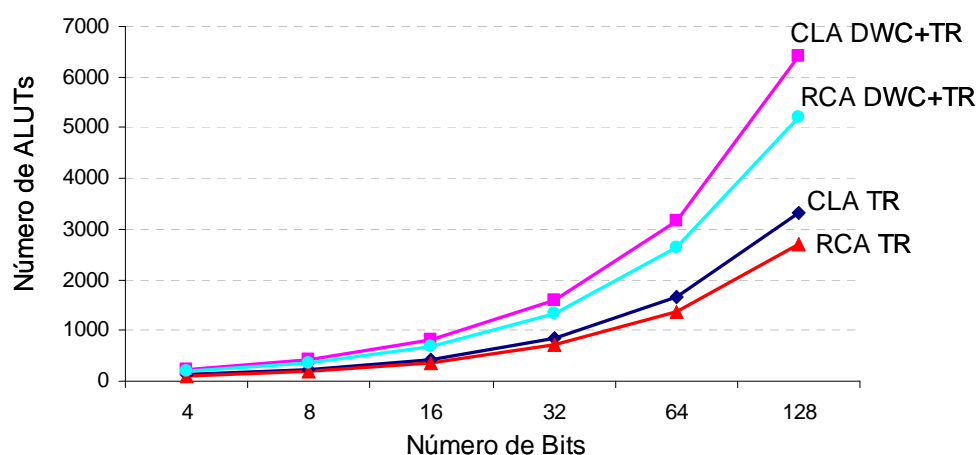


Figura 80 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e TR

Os somadores protegidos com a técnica DWC+TR utilizaram uma quantidade maior de recursos em relação aos somadores protegidos com TR, tanto para o somador CLA, como para o RCA. Este resultado era esperado, já que a técnica DWC+TR gerou um aumento considerável na utilização de ALUTs por parte dos somadores, o que não ocorreu quando os somadores foram protegidos com TR.

A Tab. 15 mostra de maneira detalhada os resultados dos recursos utilizados pelos somadores protegidos com as técnicas DWC+TR e com RT, juntamente com os acréscimos percentuais de recursos.

Em se tratando de recursos utilizados, quando se aplicam as técnicas de proteção TR e DWC+TR aos somadores, o maior acréscimo de ALUTs foi gerado pela técnica DWC+TR, ao contrário do que ocorreu em relação ao desempenho, onde os

somadores protegidos com TR tornaram-se mais lentos que os protegidos com DWC+TR.

Tabela 15 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e protegidos com TR

Bits	TR (1)		DWC+TR (2)		Acréscimo de Recursos [[2]/(1)-1]*100 %	
	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	109	129	199	237	+83%	+84%
8	197	231	368	436	+87%	+89%
16	365	433	690	826	+89%	+91%
32	701	838	1.334	1.609	+90%	+92%
64	1.373	1.646	2.622	3.171	+91%	+93%
128	2.717	3.327	5.199	6.418	+91%	+93%

No caso do RCA, os somadores protegidos com DWC+TR utilizaram, em média, 89% mais recursos que os somadores protegidos com TR, tendo este acréscimo variado entre 83% (para 4 bits) e 91% (para 64 e 128 bits).

Já para o CLA, os somadores com DWC+TR utilizaram, em média, 90% mais ALUTs que os somadores com RT. Tal este acréscimo variou entre 84% (para 4 bits) e 93% (para 64 e 128 bits) .

Com isso, percebe-se que a técnica DWC+TR utilizou, em média, 89% mais recursos que a técnica RT e que, à medida que o número de bits dos somadores aumenta, o acréscimo de recursos utilizados pelos somadores com DWC+TR torna-se maior em relação aos protegidos com TR. Isso ocorre devido ao fato da técnica DWC+TR utilizar dois exemplares do bloco somador, ao passo que a TR utiliza apenas um exemplar, e com o aumento dos bits, o acréscimo de recursos gerado pelos demais elementos que constituem a técnica torna-se menos significativo que o acréscimo gerado pelos exemplares do somador.

6.9 Comparação entre os somadores protegidos com DCW+RT e com TMR

O gráfico da Fig. 81 apresenta o atraso crítico dos somadores protegidos com as técnicas TMR e DWC+TR.

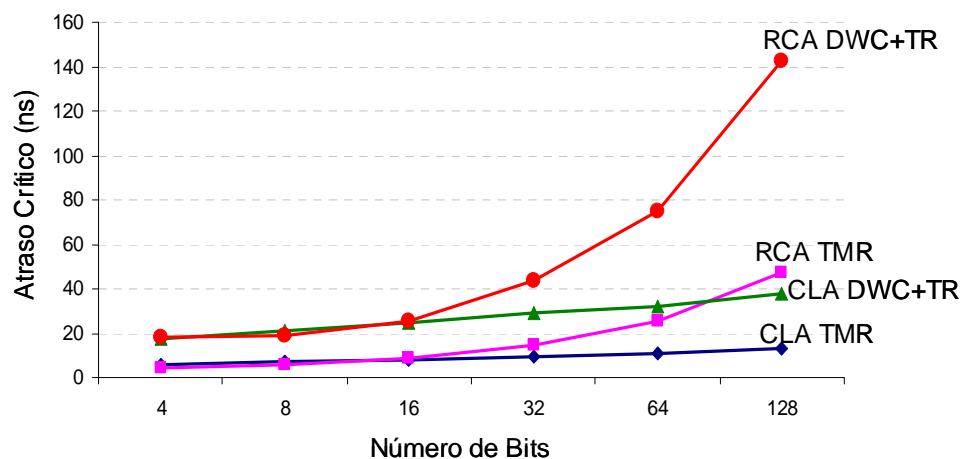


Figura 81 – Atraso crítico dos somadores protegidos com TMR e DWC+TR

Pode-se observar no gráfico que o desempenho dos somadores com DWC+TR foi inferior ao desempenho dos somadores com TMR e que a tendência dos somadores se mantém, ou seja, os somadores CLA TMR e CLA DWC+TR apresentam um aumento quase linear, enquanto que os somadores RCA TMR e RCA DCW+TR apresentam um aumento não-linear.

Para os somadores com TMR e DWC+TR, o RCA possuiu um desempenho inferior ao CLA, sendo que a partir de 16 bits o CLA TMR apresentou um melhor desempenho que o RCA TMR, e também melhor do que todos os demais somadores. O RCA TMR, a partir de 128 bits, passou a ter um desempenho inferior ao CLA com DWC+TR.

A Tab. 16 mostra os resultados do atraso crítico das arquiteturas dos somadores protegidos com DWC+TR e com TMR, juntamente com os resultados do acréscimo percentual entre as técnicas.

Comparando-se as técnicas de proteção TMR e DWC+TR percebe-se que os somadores protegidos com DWC+TR tornam-se mais lentos do que os somadores protegidos com TMR. Isso se deve ao fato da técnica DWC+TR inserir o atraso do circuito detector de erros e o atraso do votador, além do atraso do somador, enquanto que no caso da técnica TMR apenas o atraso do votador é inserido.

Tabela 16 – Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e protegidos com TMR

Bits	TMR (1)		DWC+TR (2)		Acréscimo de Atraso [(2)/(1)-1]*100 %	
	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)	RCA (ns)	CLA (ns)
4	4,50	5,75	18,33	17,20	+307%	+199%
8	5,53	7,06	18,89	21,30	+242%	+202%
16	8,70	8,03	25,70	25,03	+195%	+212%
32	14,36	9,62	43,52	28,98	+203%	+201%
64	25,71	10,91	75,00	32,14	+192%	+195%
128	47,45	12,84	142,29	37,90	+200%	+195%

No caso do RCA, os somadores protegidos com DWC+TR apresentaram atrasos, em média, 223% maiores que os atrasos dos somadores com TMR, sendo que este acréscimo variou entre 307% (para 4 bits) e 192% (para 64 bits).

No caso específico do CLA, os somadores com DWC+TR apresentaram um atraso, em média, 201% maior que o atraso dos somadores com TMR, sendo que o pior caso foi o CLA DWC+TR de 16 bits, que apresentou um atraso 212% maior que o atraso do CLA TMR. O melhor caso corresponde aos somadores CLA DWC+TR de 64 e 128 bits, que apresentaram um atraso “apenas” 195% maior que o atraso dos CLA TMR de 64 e 128 bits.

O gráfico da Fig. 82 apresenta a quantidade de recursos do FPGA utilizados pelos somadores protegidos através da aplicação das técnicas TMR e DWC+TR.

Como pode ser visto na Fig. 82 a técnica DWC+TR utilizou mais recursos que a técnica TMR. Logo, conclui-se que, em termos de recursos, a técnica DWC+TR demanda a maior quantidade de ALUTs dentre todas as técnicas.

Comparando-se os somadores protegidos com TMR e com DWC+TR, o RCA TMR utilizou menos recursos entre todos os somadores protegidos, enquanto que o CLA DWC+TR fez uso de uma maior quantidade de ALUTs entre os somadores.

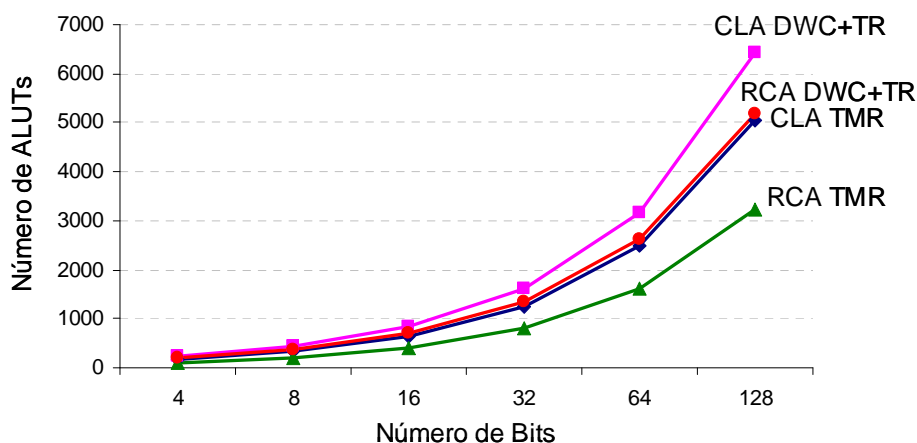


Figura 82 – Número de ALUTs utilizadas pelos somadores protegidos com DCW+TR e TMR

O CLA TMR e o RCA DWC+TR utilizaram uma quantidade de recursos intermediária, ambos com aproximadamente o mesmo número de ALUTs. Porém, como a técnica DWC+TR utiliza mais recursos que a TMR, mesmo o CLA tendo utilizado mais ALUTs que o RCA, o CLA quando protegido com TMR utilizou menos ALUTs que o RCA protegido com DWC+TR.

A Tab. 17 mostra os resultados do número de ALUTs utilizadas pelos somadores protegidos com DWC+TR e com TMR, juntamente com os resultados do acréscimo percentual entre as técnicas.

Comparando-se os somadores protegidos com DWC+TR e com TMR em termos de recursos utilizados, pode-se observar que, assim como os somadores com

DWC+TR foram mais lentos que os somadores com TMR, eles também utilizaram mais recursos que os protegidos com TMR.

Em se tratando do RCA, os somadores protegidos com DWC+TR utilizaram, em média, 69% mais recursos que os somadores com TMR, tendo este acréscimo variado entre 91% (para 4 bits) e 62% (para 64 e 128 bits).

Tabela 17 – Comparação entre os recursos utilizados pelos somadores protegidos com DWC+TR e protegidos com TMR

TMR (1)			DWC+TR (2)		Acréscimo de Recursos [[2]/(1)-1]*100 %	
Bits	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)	RCA (ALUTs)	CLA (ALUTs)
4	104	169	199	237	+91%	+40%
8	216	327	368	436	+70%	+33%
16	416	637	690	826	+66%	+30%
32	816	1.260	1.334	1.609	+63%	+28%
64	1.616	2.504	2.622	3.171	+62%	+27%
128	3.216	5.054	5.199	6.418	+62%	+27%

Em relação ao CLA, os somadores protegidos com DWC+TR utilizaram, em média, 31% mais ALUTs que os somadores com TMR, tendo este acréscimo variando entre 40% (4 bits) e 27% (para 64 e 128 bits).

Portanto, percebe-se que, com o aumento dos bits dos somadores, o acréscimo percentual de recursos entre os somadores com DWC+TR e TMR diminui.

6.10 Análise da Proteção dos Somadores Contra SETs

A fim de avaliar a eficiência das técnicas de proteção contra SETs aplicadas aos somadores RCA e CLA, campanhas de injeção de falhas foram realizadas para todos os CLAs e RCAs por meio de simulações no nível lógico usando o simulador ModelSim da Mentor Graphics (versão para Altera) (MENTOR, 2007).

A injeção de falhas e o método de simulação consistem em injetar na lógica combinacional uma falha simples por vez e analisar sua propagação usando simulação no nível lógico com atraso. Uma vez que a propagação é função da lógica do circuito, a

simulação deve levar em consideração cada possível vetor de entrada, a fim de determinar se o SET atinge uma das saídas primárias do circuito. A falha transiente é modelada no simulador ModelSim forçando a inversão de determinado sinal durante um intervalo de tempo, que corresponde à duração do SET. Para usar tal modelagem, um algoritmo de injeção de falhas clássico pode ser usado, conforme ilustrado na Fig. 83.

```

1  para cada falha
2  para cada vetor de entrada
3    simular o circuito com falha
4    comparar resultado c/ o resultado do circuito s/ falha
5    se (falha propagada)
6      incrementa lista de falhas
7    novo vetor de entrada
8  nova falha

```

Figura 83 – Algoritmo de Injeção de Falhas

Para tornar possível a simulação de um número significativo de falhas para todas as arquiteturas de CLAs e RCAs analisadas, foi necessário desenvolver três *scripts* TCL (OUSTERHOUT, 1994) para automatizar o controle do processo de injeção e simulação de falhas.

Estes *scripts* são: “Gerador de Arquivos de Simulação”, “Gerador de Resultados” e “Comparador”. A Fig. 84 mostra o fluxograma do processo de injeção e simulação de falhas e sua relação com a síntese com o Quartus II. A partir da síntese com o Quartus II é possível acessar linhas internas do circuito, mapeadas para o dispositivo FPGA Stratix II. Para cada combinação de linha interna e vetor de entrada, o Gerador de Arquivos de Simulação gera um arquivo contendo uma lista de comandos de Simulação ModelSim.

Uma vez que tenham sido gerados todos os arquivos de simulação necessários, o *script* Gerador de Resultados controla a execução da simulação. Para cada simulação são carregados os arquivos de simulação no *prompt* do ModelSim, juntamente com a informação de atraso obtida a partir da síntese com o Quartus II.

Ao término de cada simulação, um arquivo com os resultados é gerado. Após todas as simulações terem sido executadas, o *script* Comparador analisa todos os

resultados para uma dada falha, comparando-os com o resultado esperado para o circuito livre de falhas, verificando se a falha se propagou até as saídas primárias e foi capturada por um registrador de saída.

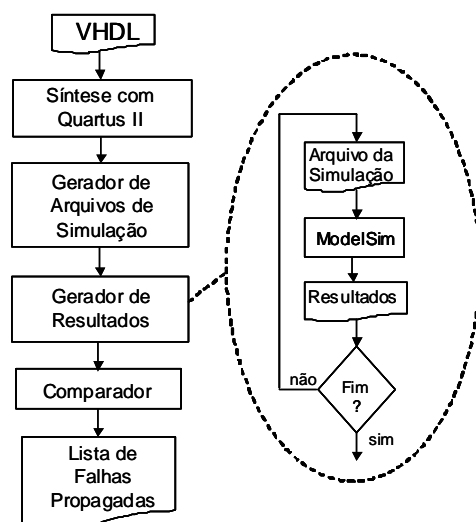


Figura 84 – Fluxograma do processo injeção e simulação automática de falhas.

Considerando que a análise da robustez de todos os CLAs e RCAs implementados necessita de um longo tempo de execução, a avaliação da robustez foi realizada apenas para os somadores de 4 bits.

Assim, falhas foram injetadas em todas as linhas internas cujo acesso foi proporcionado pelo Quartus II, por meio dos arquivos extraídos que serviram de entrada para o simulador ModelSim.

A Tab. 18 mostra os resultados para injeção e simulação de falhas para as quatro arquiteturas de 4 bits dos CLAs, enquanto que a Tab. 19 mostra os resultados para as quatro arquiteturas dos RCAs.

Tabela 18 – Resultados da injeção e simulação de falhas para CLAs de 4 bits.

CLA de 4 bits	Pontos de Injeção	Simulações	Falhas Propagadas
Não-Protegido	30	7680	232
TMR	180	46080	0
TR	65	16640	0
DWC+TR	202	51712	0

Tabela 19 – Resultados da injeção e simulação de falhas para RCAs de 4 bits.

RCA de 4 bits	Pontos de Injeção	Simulações	Falhas Propagadas
Não-Protegido	26	6656	192
TMR	78	19968	0
TR	47	12032	0
DWC+TR	178	45568	0

É importante notar que o número de falhas injetadas nas versões protegidas do CLA e do RCA é maior que o número de falhas injetadas nas versões não-protegidas devido ao acréscimo de recursos de *hardware* requeridos para realizar a proteção. Conseqüentemente, o número de simulações necessárias é maior, tornando longo o tempo de execução. Porém, as Tabs. 18 e 19 mostram que as técnicas de proteção são eficientes em proteger CLAs e RCAs de 4 bits contra SETs. Como CLAs de mais alta ordem são construídos a partir do módulo de 4 bits, é esperado que estes também sejam eficientemente protegidos com as técnicas de proteção analisadas.

7 Conclusões

O presente trabalho investigou o comportamento de somadores *Carry Lookahead* e *Ripple Carry* ao serem protegidos com as técnicas de tolerância TMR (*Triple Modular Redundancy*), TR (*Time Redundancy*) e DWC+TR (*duplication with comparison with Time Redundancy*), em implementações voltadas para dispositivos FPGAs da família Stratix II da Altera.

Primeiramente foram analisados os resultados de uso de recursos de *hardware* e desempenho, em termos de atraso crítico, dos somadores sem a aplicação das técnicas de tolerância. Tal análise serviu para determinar o comportamento dos somadores *Carry Lookahead* e *Ripple Carry* quando implementados em FPGAs Stratix II da Altera, para a posterior comparação com os resultados obtidos através dos somadores protegidos.

Os somadores RCA e CLA não-protegidos apresentaram um aumento não linear e quase linear, respectivamente, no atraso crítico. Já em relação ao uso de recursos de *hardware*, ambos somadores não-protegidos apresentaram aumento não linear, sendo que o somador CLA utilizou uma maior quantidade de ALUTs devido à lógica extra utilizada para o cálculo dos *carries* em paralelo.

Em relação ao atraso crítico, quando aplicadas as técnicas de proteção, os somadores apresentaram as mesmas tendências já apresentadas pelos somadores não-protegidos, ou seja, o somador RCA apresentou aumento não linear no atraso e o CLA apresentou um aumento quase linear. Porém, como era esperado, com

desempenho inferior aos somadores não-protegidos, sendo a queda no desempenho provocada por diferentes motivos, dependendo da técnica aplicada.

Na técnica TMR, a queda no desempenho é provocada pela inserção do circuito votador, enquanto que na técnica TR, é provocada pelo tempo extra utilizado para a obtenção das três amostras do resultado e pelo atraso do votador. Já na DWC+TR a queda no desempenho é devida ao tempo extra para obtenção das duas amostras do resultado de cada bloco somador, pelo atraso do circuito detector de erros e pelo atraso do votador.

Logo, entre somadores de mesmo tipo, a técnica de proteção que apresentou menor desempenho foi a TR. A técnica TMR apresentou o melhor desempenho entre todas as técnicas aplicadas, sendo que a DWC+TR apresentou um desempenho intermediário.

No geral, considerando apenas os somadores de 4 e 8 bits, os RCAs, tanto protegidos, como não-protegidos, apresentaram um desempenho superior aos CLAs, apesar destes últimos serem somadores rápidos. Isto se deve ao fato do atraso provocado pela cadeia de propagação do *carry* do RCA ter impacto representativo em relação ao CLA para somadores com um número maior de bits. Porém, para somadores de 128 bits, os RCAs protegidos apresentaram um desempenho inferior aos CLAs protegidos e o RCA não-protegido foi mais lento até mesmo que o CLA DWC+TR, o que não deixa de ser um resultado surpreendente.

A se julgar pelo comportamento dos resultados, se forem implementados somadores maiores de 128 bits, o RCA não-protegido tenderá a apresentar um desempenho inferior ao do CLA TR. Isso também ocorre com o RCA TMR. Apesar da técnica TR apresentar um desempenho inferior à técnica TMR, com o aumento do número de bits o RCA TMR passa a apresentar um desempenho inferior ao CLA TR, visto que o CLA é um somador rápido, compensando assim o fato da TR ser mais lenta que a TMR.

Em relação ao uso de recursos, como era de se esperar, todos os somadores protegidos utilizaram um número maior de ALUTs do que seus pares não-protegidos. Porém, a tendência com relação ao acréscimo de recursos de ambos os tipos de

somadores manteve-se a mesma, ou seja, o acréscimo de recursos evoluiu com o número de bits de forma não linear.

A técnica que apresentou uma maior utilização de recursos foi a DWC+TR, visto que duplica o bloco somador, utiliza quatro registradores e o circuito detector de erros, e ainda um votador. Porém, este é utilizado em todas as técnicas de proteção aplicadas aos somadores. A técnica TMR utilizou uma quantidade de ALUTs intermediária, pois o bloco somador é triplicado. Já a técnica TR apresentou o menor uso de recursos, devido ao fato desta técnica utilizar apenas um bloco somador e uma quantidade de registradores maior. Porém, estes não representam um grande aumento na utilização de recursos.

Entre todos os somadores protegidos, o CLA DWC+TR apresentou o maior uso de recursos, o que era esperado já que a técnica DWC+TR utilizou mais recursos em relação às demais técnicas e o CLA utilizou uma quantidade maior de ALUTs que o RCA. Já o RCA TR utilizou menos ALUTs entre todos os somadores protegidos, visto que a técnica TR utilizou menos recursos que as demais e o RCA usou menos ALUTs que o CLA.

Em relação à robustez das arquiteturas protegidas, as campanhas de injeção de falhas mostraram que os SETs não se propagam através da lógica dos somadores protegidos, logo, nenhum valor errôneo atingiu as saídas primárias dos mesmos, comprovando a eficiência das técnicas de proteção.

O presente trabalho proporcionou um avanço nos estudos realizados no GACI (Grupo de Arquiteturas e Circuitos Integrados)-UFPEL sobre Falhas transientes geradas por SETs, Técnicas de Tolerância, projeto com FPGAs da Altera e uso de linguagem VHDL.

8 Referências

ABRAMOVICI, M. et al. Using Roving STARS for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications”, In: INTERNATIONAL TEST CONFERENCE, 1999. p. 973-982.

ABRAMOVICI, M.; BREUER, M.; FRIEDMAN, A. **Digital Systems Testing and Testable Design**. Piscataway, NJ: IEEE Press, 1990. 652p.

ALEXANDRESCU, D.; ANGHEL, L.; NICOLAIDIS M. Simulating Single Event Transients in VDSM ICs for Ground Level Radiation. **Journal of Electronic Testing: theory and applications**, v. 20, p.413-421, 2004.

ALEXANDRESCU, D.; ANGHEL, L.; NICOLAIDIS M. **New Methods for Evaluating the Impact of Single-Event Transients in VDSM ICs**. In: IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 17. Vancouver, Canada, 2002, p. 99-107.

ALTERA. Altera Corporation. Stratix II Handbook. V. 2. December, 2005a. Disponível em: < http://www.altera.com/literature/hb/stx2gx/stxiigx_handbook.pdf>.

ALTERA. Stratix II vs. Virtex-4 Density Comparison. White Paper. USA, Aug. 2005b. Disponível em: < <http://www.altera.com/products/devices/stratix2/features/density/st2-vir-density-compare.html> >.

ALTERA. Altera Corporation: Quartus II Reference Documentation. Disponível em: <<http://www.altera.com/literature/quartus2/lit-qts-related.jsp>>. Acesso em fev. 2007.

ANGHEL, L.; ALEXANDRESCU, D.; NICOLAIDIS, M. Evaluation of a Soft Error Tolerance Technique Based on Time and/or Space Redundancy. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 13., 2000. **Proceedings...** Los Alamitos : IEEE Computer Society, 2000. p. 237-242.

ANGHEL, L.; LEVEUGLE, R.; VANHAUWAERT, P. Evaluation of SET and SEU Effects at Multiple Abstraction Levels. In: IEEE INTERNATIONAL ON-LINE TESTING

SYMPOSIUM, 11 (IOLTS'05). Saint Raphaël (France), July 6-8, 2005. **Proceedings...** Los Alamitos (California), IEEE Computer Society, 2005. p. 309-312.

BARTH, J. Applying Computer Simulation Tools to Radiation Effects Problems. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 1997. **Proceedings...** [S.l.]: IEEE Computer Society, 1997. p. 1-83.

BAUMANN, R. C. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. **IEEE Transactions on Devices and Materials Reliability**, New York, v.5, n.3, p.305-316, September, 2005.

BAUMANN, R. C. The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction. In: INTERNATIONAL ELECTRON DEVICES MEETING, 2002. **Digest of Papers...**(S.l.), 2002. p.329-332.

BAUMANN, R. C. Soft Errors in Advanced Semiconductor Devices - Part I: The Three Radiation Sources. **IEEE Transactions on Device and Materials Reliability**, v. 1, n. 1, March 2001.

BERG, Melanie. Comunicação Pessoal. 2006.

BROWN, Stephen; VRANESIC, Zvonko. **Fundamentals of Digital Logic With VHDL Design**. New York: McGraw Hill, 2005. 2ª edição

BUSHNELL, M. L.; AGRAWAL, V. D. **Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits**. Kluwer Academic Publishers, 2000.

CALIN, T.; NICOLAIDIS, M.; VELAZCO, R. Upset Hardened Memory Design for Submicron CMOS Technology. **IEEE Transactions on Nuclear Science**, New York, v.43, n.6, p. 2874 -2878, Dec. 1996.

CANARIS, J.; WHITAKER, S. Circuit Techniques for the Radiation Environment of Space. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, 1995. **Proceedings...** Los Alamitos: IEEE Computer Society, 1995, p. 77-80.

CARMICHAEL, C.; CAFFREY, M.; SALAZAR, A. **Correcting Single-Event Upsets Through Virtex Partial Configuration**. Xilinx Application Notes 216. San Jose, USA: Xilinx, 2000.

CARMICHAEL, C. **Triple Module Redundancy Design Techniques for Virtex FPGA**. Xilinx Application Notes 197. San Jose, USA: Xilinx, 2001.

CHA, H et al. A Gate-Level Simulation Environment for Alpha-Particle-Induced Transient Faults. **IEEE TRANSACTIONS ON COMPUTERS**. Vol. 45, No. 11, Nov 1996.

CIRCUITS AND SYSTEMS DESIGN, 2005, Florianópolis. **Proceedings...** New York (USA): ACM: Association for Computing Machinery, 2005. p. 121-126.

COHEN, N. et al. Soft error Considerations for Deep-Submicron CMOS Circuit Applications. In: INTERNATIONAL ELECTRON DEVICES MEETING. **Technical Digest...** [s.l.]: July 1999, p. 315-318.

DAHLGREN, P.; LIDÉN, P. A Switch-Level Algorithm for Simulation of Transients in Combination Logic. In: IEEE FAULT TOLERANT COMPUTING SYMPOSIUM, 1995, pp 207-216.

DESCHAMPS, Jean-Pierre; BIOUL, Géry J. A.; SUTTER, Gustavo D. **Synthesis of Arithmetic Circuits FPGA, ASIC and Embedded Systems**. New Jersey: John Wiley e Sons, 2006.

DODD, P. E.; MASSENGILL, L. W. Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics **IEEE Transactions on Nuclear Science**, v. 50, n. 3, p. 583-602, June, 2003.

DUPONT, E.; NICOLAIDIS, M.; ROHR, P. Embedded Robustness IPs for Transient Error-Free ICs. **IEEE Design & Test of Computers**, New York, v.19, n.3, p. 54-68, May-June 2002.

GADLADE, M et al. Single Event Transient Pulses in Digital Microcircuits. **IEEE Transactions on Nuclear Science**, Vol. 51, No. 6, Dec 2004.

GILL, B et al. Node Sensitivity Analysis for Soft Errors in CMOS Logic. IEEE INTERNATIONAL TEST CONFERENCE. 2006.

GOKHALE, M. et al. Dynamic Reconfiguration for Management of Radiation-Induced Faults in FPGAs. In: INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 18. p. 145-150, 2004.

GOLDSTEIN, L. H. Controllability/Observability Analysis of Digital Circuits. **IEEE Transactions on Circuits and Systems**, v. CAS-26, n.9, p. 685-693, September, 1979.

HEIJMEN, T.; NIEUWLAND, A. Soft-Error Rate Testing of Deep-Submicron Integrated Circuits. In: IEEE EUROPEAN TEST SYMPOSIUM, 11 (ETS'06) Southampton (England), May 21-24, 2006. **Proceedings...** Los Alamitos (California), IEEE Computer Society, 2006. p.247-252.

HOUGHTON, A. D. **The Engineer's Error Coding Handbook**. London: Chapman & Hall, 1997.

HWANG, Kai. Fast Two-Operand Adders/Subtractors. In: **Computer Arithmetic Principles, Architecture, and Design**. New York: John Wiley e Sons, 1979. p. 69-95.

IEEE Standard VHDL Language Reference Manual: IEEE Std 1076-1993. IEEE Press: [S.l.], 1994.

IROM, F. et al. Single-event upset in commercial silicon-on-insulator PowerPC microprocessors. In: IEEE INTERNATIONAL SILICON-ON-INSULATOR CONFERENCE, 2002. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.203-204.

JEDEC STANDARD JESD89. Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices. [S.l.]: Jedec Solid State Association, 2001.

JOHNSTON, A. Scaling and Technology Issues for Soft Error Rates. In: RESEARCH CONFERENCE ON RELIABILITY, 4., 2000. **Proceedings...** Palo Alto: Stanford University, 2000

LALA, Parag K.. Fault-Tolerant Design. In: **Self-Checking and Fault-Tolerant Digital Design**. San Francisco: Academic Press, 2001. p. 161-201.

LIMA-KASTENSMIDT, Fernanda G. de. **Designing Single Event Upset Mitigation Techniques for Large SRAM-based FPGA Components**. Tese de Doutorado. Porto Alegre: PPGC da UFRGS, 2003.

LIMA, F.; CARRO, L; REIS, R. Techniques for reconfigurable logic applications: Designing fault tolerant systems into SRAM-based FPGAs. In: INTERNATIONAL DESIGN AUTOMATION CONFERENCE, DAC, 2003. **Proceedings...** New York: ACM, 2003. p. 650-655.

LIU, M.N.; WHITAKER, S. Low Power SEU Immune CMOS Memory Circuits. **IEEE Transactions on Nuclear Science**, New York, v.39, n.6, p. 1679-1684, Dec. 1992.

MAVIS, D.G.; EATON, P.H. Soft Error Rate Mitigation Techniques for Modern Microcircuits. In: INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 2002. p. 216-225.

MENTOR Graphics Corporation, "Technology Support for Altera Devices", Disponível em: <<http://www.mentor.com>>, Acesso em Jan. 2006.

MESSENGER, C. G. Collection of Charge on Junction Nodes from Ion Tracks. **IEEE Transactions on Nuclear Science**, v. NS-29, p. 2024-2031, December 1982.

NEVES, C. **Análise Automática da Propagação de Single Event Transients em Circuitos Combinacionais CMOS**. Pelotas: Bacharelado em Ciência da Computação da UFPel, abril de 2006. Trabalho de Conclusão.

NICOLAIDIS, M. Design for Soft Error Mitigation. **IEEE Transactions on Devices and Materials Reliability**, New York, v.5, n.3, p.405-418, September, 2005

NICOLAIDIS, M. Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies. In: IEEE VLSI TEST SYMPOSIUM, 17., 1999. **Proceedings...** Los Alamitos: IEEE Computer Society, 1999. p. 86-94.

NIEUWLAND, A.; JASAREVIC, S.; JERIN, G. Combinational Logic Soft Error Analysis and Protection. In: IEEE INTERNATIONAL ON-LINE TESTING SYMPOSIUM, 12 (IOLTS'06). Lake of Como (Italy), July 10-12, 2006. **Proceedings...** Los Alamitos (California), IEEE Computer Society, 2006. p.99-104.

NORMAND, E.; BAKER, T.J. Altitude and Latitude Variations in Avionics SEU and Atmospheric Neutron Flux. **IEEE Transactions on Nuclear Science**, New York, v.40, n.6, p. 1484-1490, December. 1993.

NORMAND, E. Single event upset at ground level. **IEEE Transactions on Nuclear Science**, New York, v.43, n.6, p. 2742 -2750, December, 1996.

O'BRYAN, M. et al. Current Single Event Effects and Radiation Damage Results for Candidate Spacecraft Electronics. In: IEEE RADIATION EFFECTS DATA WORKSHOP, 2002. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p. 82-105.

OKLOBDZIJA, Vojin G.. High-Speed VLSI Arithmetic Units: Adders and Multipliers. In: A. Chandrakasan, W.J. Bowhill, F. fox, (Editores), **Design of High-Performance Microprocessor Circuits.**, New Jersey: IEEE Press, 2001. p. 181-204.

OMAÑA, M. et al. A Model for Transient Fault Propagation in Combinatorial Logic. In: 9th IEEE INTERNATIONAL ON-LINE TESTING SYMPOSIUM, 2003. **Proceedings..** Los Alamitos: IEEE: , 2003, p. 111-115.

OUSTERHOUT, J. K. Tcl and the Tk Toolkit. Reading, MA: Addison-Wesley Publishing Company, 1994. 458p.

PETERSON, W. Wesley. **Error-correcting codes.** 2nd. Edition. Cambridge, EUA: The Mit Press, 1980. 560p.

PRADHAN, Dhiraj K.. An Introduction To Design and Analysis of Fault-Tolerant Systems. In: **Fault-Tolerant Computer System Design.** New Jersey: Prentice Hall PTR, 1996. p. 1-84.

ROCKETT, L. R. An SEU-Hardened CMOS Data Latch Design. **IEEE Transactions on Nuclear Science**, New York, v.35, n.6, p. 1682-1687, December, 1988.

SHIVAKUMAR, P. et al. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2002. P.389-398.

WEAVER, H.; et al. An SEU Tolerant Memory Cell Derived from Fundamental Studies of SEU Mechanisms in SRAM. **IEEE Transactions on Nuclear Science**, New York, v.34, n.6, Dec. 1987.

WIRTH, G.; VIEIRA, M.; HENES NETO, E.; KASTENSMIDT, F. Single Event Transients in Combinatorial Circuits. In: 16th INTERNATIONAL SYMPOSIUM ON INTEGRATED

VIOLANTE, M. Accurate Single-Event-Transient via Zero-Delay Logic Simulation. **IEEE Transactions on Nuclear Science**. Vol. 50, No. 6, Dec. 2003, pp. 2113-2118.