

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



**SOMADORES RÁPIDOS TOLERANTES A FALHAS TRANSIENTES
IMPLEMENTADOS EM FPGA**

Eduardo Macedo Mesquita

Pelotas, 2007

EDUARDO MACEDO MESQUITA

**SOMADORES RÁPIDOS TOLERANTES A FALHAS TRANSIENTES
IMPLEMENTADOS EM FPGA**

Trabalho acadêmico apresentado ao curso de Bacharelado em 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

M582s Mesquita, Eduardo Macedo

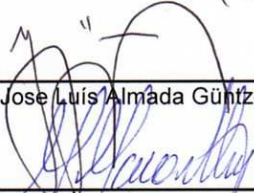
Somadores rápidos tolerantes a falhas transientes implementados em FPGA / Eduardo Macedo Mesquita; orientador Luciano Volcan Agostini ; co-orientador José Luís Almada Güntzel. – Pelotas, 2007. – 130f. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. 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.Güntzel, José Luís Almada. III.Título.

Banca examinadora:

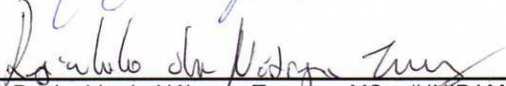


Prof. Luciano Volcan Agostini, Msc. (Orientador)



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

Prof. Marcello da Rocha Macarthy, MSc.



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



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

Agradecimentos

Agradeço, primeiramente, a meus pais por terem-me concedido a oportunidade de cursar esta universidade, e, além disto, me fornecido condições de cursá-la de maneira prioritária, sacrificando seus interesses em favor dos meus. Não existem agradecimentos suficientes a fazer a vocês. Obrigado pela presença e amor incondicional!

Aos colegas do GACI, pela amizade e aprazível ambiente de trabalho proporcionado ao longo destes últimos anos, não podendo deixar de mencionar a descontração dos churrascos de final de ano na casa do Luciano, sejam eles no Cassino ou em Pelotas. Devo-me confessar grato, em especial, a Helen e Guilherme, importantes para a execução deste trabalho.

Aos meus colegas de faculdade pelo companheirismo e amizade irrefutáveis ao longo de toda graduação, seja nos momentos de trabalho árduo (madrugadas inteiras, que com toda tensão conseguiam ser agradáveis) ou descontração. Momentos que passei com vocês jamais serão esquecidos: Reuniões no Barros, cervejas no Bar da Mara, Bauru do Eloí (hoje é festa lá no meu apê, pode aparecer, Eloí eu vou comer..) e a criação de diversas canções e vídeos, como o Vôo da Mariposa. Barros, Braga, Foguinho, Helen, Presidente e Zanetti: Vocês foram e são especiais!

À minha namorada e amiga Nathália, que sempre soube entender meus momentos de ausência. Obrigada pelo amor, amizade e apoio dispensados (e também pelas correções...). Eles foram fundamentais!

Aos professores do Curso de Ciência da Computação, que em muito contribuíram para minha formação profissional e pessoal dentro da universidade, não podendo deixar de mencionar os professores Güntzel, Luciano, Gil (proprietário do teclado mágico) e Campani.

Por fim, agradeço a meus orientadores do GACI, pela oportunidade inigualável de participar como membro deste grupo de pesquisa por 2 anos, bem como pelo auxílio constante e exitoso esforço de manter o GACI como uma verdadeira família . Em especial agradeço a meu orientador, Prof. Güntzel, pela dedicação e interesse demonstrados para a execução deste, pelo incentivo e confiança depositados no meu trabalho ao longo dos anos em que participei do GACI, pelos laços de amizade criados ao longo deste tempo (que com certeza não se desfarão), bem como pelas críticas que tanto me acrescentaram.

À todos, meu mais sincero “Obrigado” !

Resumo

MESQUITA, Eduardo Macedo. Somadores Rápidos Tolerantes a Falhas Transientes Implementados em FPGA. Monografia – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas.

FPGAs têm se tornado cada vez mais importante para implementação de sistemas eletrônicos. Isto se deve a sua alta densidade lógica, curto tempo para lançamento no mercado e baixo custo. A fim de prover esta alta densidade lógica, os FPGAs do estado-da-arte são fabricados com tecnologia CMOS nanométrica. Entretanto, devido à contínua redução das dimensões dos transistores, proporcionada pela evolução dos processos de fabricação CMOS, os circuitos integrados fabricados em tecnologias do estado-da-arte estão se tornando cada vez mais vulneráveis a falhas transientes. Dentre estas falhas encontram-se os *single-event upsets* (SEUs), que se caracterizam pela colisão de partículas energéticas com um elemento de memória, causando uma troca do valor por ele armazenado, e os *single-event transients* (SETs), que se referem à colisão de partículas energéticas com a parte combinacional do circuito, gerando um pulso transiente de tensão.

A preocupação com os efeitos da radiação sobre os FPGAs não é um tema recente, visto que, devido a sua reconfigurabilidade, os FPGAs são muito utilizados em missões espaciais, onde a atividade cósmica é mais intensa. Algumas poucas empresas têm investido na fabricação de FPGAs protegidos contra radiação. Porém, devido à pequena demanda, o preço por peça é muito elevado. Assim, a solução que tem sido proposta mais freqüentemente reside na aplicação de técnicas de tolerância a falhas às arquiteturas que serão implementadas nos FPGAs, podendo assim utilizar-se FPGAs de propósito geral, os quais possuem menores custos unitários.

Neste trabalho somadores protegidos contra SETs foram sintetizados e validados para FPGAs da família Stratix II da Altera. Três técnicas de proteção foram investigadas: *Triple Module Redundancy* (TMR), *Time redundancy* (TR) e *Duplication with Comparison combined with Time Redundancy* (DWC+RT).

Os somadores experimentados foram os *Ripple Carry* (RCA), *Carry Select* (CSA) e *Re-computing the Inverse Carry-in* (RIC), sendo este último uma nova arquitetura de somador rápido proposta neste trabalho. Os resultados mostraram que os somadores RIC, quando comparados aos CSAs, utilizaram em média uma quantidade de recursos 17% menor, apresentando um desempenho superior, no que se refere aos somadores com mais de 16 bits.

A análise dos resultados de síntese das diversas versões de somadores protegidos permitiu quantificar o acréscimo de hardware e a degradação de desempenho resultantes da aplicação de cada técnica de proteção, relevando quais técnicas são mais apropriadas para cada tipo de somador quando um dos requisitos, uso de recursos ou desempenho, precisa ser otimizado.

Palavras-chave: Microeletrônica. Falhas transientes. FPGA. VHDL. SETs. Tolerância a falhas. Arquiteturas de somadores.

Abstract

MESQUITA, Eduardo Macedo. Somadores Rápidos Tolerantes a Falhas Transientes Implementados em FPGA. Monografia – Curso de Bacharelado em Ciência da Computação. Universidade Federal de Pelotas.

FPGAs are becoming very important to implement electronic systems, due to their high logic density, fast time-to-market and low cost. In order to provide high logic densities the state-of-the-art FPGA devices are fabricated with nanometer CMOS technology. However, due to the continuous shrinking of transistor dimensions offered by the evolution of CMOS processes, integrated circuits fabricated in state-of-the-art technology are becoming more vulnerable to transient faults. Among these kind of faults are single-event upsets (SEUs), that correspond to bit flips caused by the collision of energetic particles in memory cells, and single-event transients (SETs), that are transient voltage pulses caused by the collision of energetic particles in the combinational logic.

The concern about the radiation effects on FPGAs is not recent. Due to their reconfiguration capability FPGAs have been used in spaces missions where the cosmic activity is high. Thus, some FPGA vendors have invested on hardened-by-fabrication FPGAs. However, the low demand makes the unitary cost of such devices prohibitively high. Thus, the most frequently proposed solution relies on applying mitigation techniques to architectures implemented into ordinary (non-protected) FPGAs.

In this work adders protected against SETs were synthesized and validated assuming Altera Stratix II FPGAs. Three protection techniques were investigated: Triple Module Redundancy (TMR), Time redundancy (TR) e Duplication with Comparison combined with Time Redundancy (DWC+RT).

Three adder architectures were investigated in this work: Carry Select (CSA), Ripple Carry (RCA) and Re-computing the Inverse Carry-in (RIC). The RIC adder is a novel fast adder architecture proposed in this work. The results showed that the RIC, uses in average 17% less resources than the CSA, while presenting a better performance than CSAs for adders with a bit width greater than 16 bits.

By analyzing the synthesis results of various adder versions it was possible to quantify both resource overhead and performance degradation resulting from the application of each protection technique, thus revealing which technique is the most appropriate to each type of adder when the optimization of either resource or performance is required.

Keywords: Microelectronics. Transient Faults. FPGA. VHDL. SETs. Fault tolerance. Adder Architectures.

Lista de Figuras

Figura 1– Comportamento da região sensível de um transistor, dada a ação de uma partícula carregada.	20
Figura 2 – Forma do pulso de corrente gerado após a colisão de uma partícula carregada com a região sensível do silício.	21
Figura 3 – <i>Single event upset</i> (SEU) em uma célula de memória SRAM.....	24
Figura 4 - <i>Single-event transient</i> (SET) e sua possível propagação em um bloco combinacional.	25
Figura 5 – Mascaramento lógico.	26
Figura 6 – Processo de atenuação de um pulso de tensão quando este se propaga pelo circuito.....	27
Figura 7 – Mascaramento por latching Window.	27
Figura 8 – Modelo de Messenger para um pulso de tensão gerado a partir de um SET.	28
Figura 9 – Estrutura interna geral de um FPGA.	32
Figura 10 – Estrutura geral de um bloco lógico.....	33
Figura 11 – Implementação de uma função lógica de três entradas utilizando uma LUT.	34
Figura 13 – <i>Pass-Transistor switches</i> em FPGAs.....	35
Figura 14 – Visualização de um conjunto de células SRAM organizadas com um registrador deslocador.....	36
Figura 15 – Arquitetura interna dos FPGAs da família Stratix II.....	37
Figura 16 – Estrutura interna de um LAB.	38
Figura 17 - Algumas configurações suportadas por um ALM.....	39
Figura 18 – Estrutura interna de um ALM em alto nível.	40
Figura 19 – Estrutura interna de um ALM em um nível maior de detalhamento.....	41
Figura 20 – Linhas de conexão dentro de dos dispositivos da Stratix II.....	43
Figura 21 – Extração da arquitetura de um RCA através de sua tabela-verdade.	45
Figura 22 – Somador completo de 1 bit.	46
Figura 23 – Somador <i>Ripple Carry</i> de 4 bits.	46
Figura 24 – Caminho crítico de um RCA de 4 bits.	47
Figura 25 – Somador RCA de 1 bit construído a partir de dois meio-somadores (<i>half-adders</i>).	47
Figura 26 – Somador CSA de 8 bits.....	48
Figura 27 – Caminho crítico de um CSA de 8 bits.....	51
Figura 28 – Caminho crítico de um somador CSA de 32 bits.....	51
Figura 29 – Somador RIC de 4 bits.....	52
Figura 30 – Exemplo de uma soma utilizando a propriedade 1 da soma.....	53
Figura 31 – Exemplo de uma soma utilizando a propriedade 2 da soma.....	53
Figura 32 – Processo de extração do bloco RB do RIC.....	55
Figura 33 – Caminho crítico de um somador RIC de 8 bits.....	57
Figura 34 – Caminho crítico de um somador RIC de 32 bits.....	58
Figura 35 – Comparação entre os atrasos críticos das arquiteturas de somadores.....	59

Figura 36 – Atrasos críticos dos somadores RIC e CSA.....	59
Figura 37 - Número de ALUTs utilizadas pelos somadores RCA, RIC e CSA.	61
Figura 38 – Técnica de proteção TMR.	67
Figura 39 – Redundância dinâmica utilizando S módulos reservas.	69
Figura 40– Redundância híbrida com um bloco reserva.	70
Figura 41 – Bloco combinacional protegido com Redundância Temporal.....	71
Figura 42 – Bloco combinacional protegido com DWC+TR.	73
Figura 43 – Pontos vulneráveis de um FPGA da família Virtex da Xilinx.	76
Figura 44 - <i>SEU-Hardened D Flip-Flop</i> do FPGA da Actel.....	77
Figura 45 – CSA de 8 bits protegido com TMR.	80
Figura 46 – (a) Votador de 1 bit e (b) tabela-verdade.	81
Figura 47 – Somador CSA de 16 bits mapeado para o FPGA da família Stratix II.....	82
Figura 48 - Somadores (a) RIC de 8 bits protegido com TMR (b) RCA de 8 bits protegido com TMR.....	83
Figura 49 -(a) Diagrama em blocos de um módulo de hardware protegido com TR e (b) diagrama de estados do controle.	85
Figura 50 – RIC de 8 bits protegido com RT.	86
Figura 51– (a) Diagrama em blocos de um módulo de hardware protegido com DWC+TR e (b) diagrama de estados do controle.	87
Figura 52 – RIC de 8 bits protegido com DWC+TR.	88
Figura 53 – (a) Circuito detector de erros e (b) sua tabela-verdade.....	89
Figura 54 – (a) RCA protegido com TMR utilizando as diretivas e (b) RCA protegido com TMR sem as diretivas.....	91
Figura 55 – Atraso crítico das arquiteturas não-protegidas e protegidas com TMR.....	92
Figura 56 – Atraso crítico dos somadores rápidos não-protegidos e protegidos com TMR.	93
Figura 57 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e não- protegidos.	95
Figura 58 – Atraso crítico das arquiteturas protegidas com redundância temporal -TR e não-protegidas.	98
Figura 59 - Atraso crítico dos somadores rápidos protegidos com TR.....	98
Figura 60 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e TR.	99
Figura 61 – Número de ALUTs utilizadas pelos somadores não-protegidos e protegidas com TR.....	102
Figura 62 – Comparação entre os somadores protegidos com as técnicas TR e TMR, quanto à utilização de recursos.....	103
Figura 63 – Número médio de ALUTs demandado pelos somadores, dada a aplicação das técnicas TMR e TR.....	104
Figura 64– Atraso crítico das arquiteturas não-protegidas e protegidas com DWC+TR.	107
Figura 65 - Atraso crítico dos somadores rápidos protegidos com DWC+TR.	108
Figura 66 – Comparação entre os atrasos críticos de todos os somadores protegidos com TMR, TR e DWC+TR.....	109
Figura 67 – Atraso crítico dos somadores não-protegidos e protegidos com DWC+TR.	112

Figura 68 – Comparação entre o número de ALUTs de todos tipos de somadores protegidos com TMR, TR e DWC+TR.....	113
Figura 69 – Número médio de ALUTs demandado pelos somadores, dada a aplicação das técnicas TMR e DWC+TR.	115
Figura 70 - Fluxo de injeção e simulação de falhas.	119

Lista de Tabelas

Tabela 1 – Conversão do resultado de uma adição com <i>carry</i> de entrada igual a 0 para o resultado com <i>carry</i> de entrada igual a 1.	54
Tabela 2 – Atrasos críticos dos somadores RCA, RIC e RCA.	60
Tabela 3 - Comparação entre as arquiteturas de somadores, quanto à utilização de ALUTs.	61
Tabela 4 - Comparação entre os atrasos críticos dos somadores protegidos com TMR e não-protegidos.	94
Tabela 5 - Comparação entre as arquiteturas de somadores protegidas com TMR e não-protegidas quanto à utilização de ALUTs.....	96
Tabela 6 - Comparação entre os atrasos críticos dos somadores protegidos com TR e não-protegidos.	100
Tabela 7 - Comparação entre os atrasos críticos dos somadores protegidos com TR e TMR.	101
Tabela 8 - Comparação entre as arquiteturas de somadores protegidos com TR e não-protegidas, quanto à utilização de ALUTs.....	103
Tabela 9 - Comparação entre as arquiteturas de somadores protegidos com TR e TMR, quanto à utilização de ALUTs.	105
Tabela 10 - Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e não-protegidos.	109
Tabela 11 - Comparação entre os atrasos críticos dos somadores protegidos com TMR e DWC+TR.....	110
Tabela 12 - Comparação entre os atrasos críticos dos somadores protegidos com TR e DWC+TR.....	111
Tabela 13 - Comparação entre as arquiteturas de somadores protegidos com DWC+TR e não-protegidos, quanto à utilização de ALUTs.....	114
Tabela 14 - Comparação entre as arquiteturas de somadores protegidos com TMR e DWC+TR, quanto à utilização de ALUTs.	116
Tabela 15 - Comparação entre as arquiteturas de somadores protegidos com TR e DWC+TR, quanto à utilização de ALUTs.	116
Tabela 16 – Resultados oriundos da injeção de falhas nos circuitos somadores de 4 bits.....	120

Lista de abreviaturas e siglas

ALM	<i>Adaptive Logic Module</i>
ALUT	<i>Adaptive Look Up Table</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
CLA	<i>Carry Lookahead Adder</i>
CLB	<i>Configurable Logic Block</i>
CMOS	<i>Complementary Metal-Oxide-Semiconductor</i>
CSA	<i>Carry Select Adder</i>
CPLD	<i>Complex Programmable Logic Device</i>
DSP	<i>Digital Signal Processing</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>
FIFO	<i>First In First Out</i>
FPGA	<i>Field Programmable Gate Array</i>
FSM	<i>Finite State Machine</i>
HDL	<i>Hardware Description Language</i>
LAB	<i>Logic Array Block</i>
LE	<i>Logic Element</i>
LUT	<i>Look-Up Table</i>
MBU	<i>Multi-Bit Upsets</i>
NMOS	<i>N-channel Metal-Oxide-Semiconductor</i>
PLA	<i>Programmable Logic Array</i>
PMOS	<i>P-channel Metal-Oxide-Semiconductor</i>
PROM	<i>Programmable Read Only memory</i>
RAM	<i>Random Access Memory</i>
RB	<i>Recomputing Block</i>
RCA	<i>Ripple Carry Adder</i>
RIC	<i>Re-computing the Inverse Carry-in</i>
SEE	<i>Single Event Effect</i>
SEL	<i>Single-Event Latchup</i>
SET	<i>Single-Event Transient</i>
SEU	<i>Single-Event Upset</i>
SHE	<i>Single Hard Error</i>
SOI	<i>Silicon on Insulator</i>
SPLD	<i>Simple Programmable Logic Device</i>
SRAM	<i>Static Random Access Memory</i>
TID	<i>Total Ionization Dose</i>
TMR	<i>Triple Modular Redundancy</i>

TR	<i>Time Redundancy</i>
VDSM	<i>Very Deep Sub-Micron</i>
VHDL	<i>VHSIC Hardware Description Language</i>

Sumário

Resumo	4
Abstract.....	5
Lista de Figuras.....	6
Lista de Tabelas	9
Lista de abreviaturas e siglas	10
Sumário.....	12
1 Introdução.....	14
2 Os Efeitos da Radiação Sobre os Semicondutores	18
2.1 Os Efeitos da Radiação Sobre os Circuitos Integrados.....	20
2.2 <i>Single Event Effects (SEEs)</i>	21
2.2.1 <i>Destructive SEE</i>	22
2.2.2 <i>Hard SEE</i>	22
2.2.3 <i>Soft SEE</i>	22
2.2.3.1 <i>Single Event-Upset (SEU)</i>	23
2.2.3.2 <i>Single Event-Transient (SET)</i>	24
3 FPGAs - <i>Field Programmable Gate Arrays</i>	30
3.1 Arquitetura dos FPGAs	32
3.2 Stratix II Altera	36
3.2.1 Blocos Lógicos – LABs	38
3.2.2 ALM - <i>Adaptive Logic Module</i>	38
3.2.3 Interconexões – Stratix II.....	42
4 Somadores.....	44
4.1 Somadores <i>Ripple-Carry</i>	45
4.2 Somadores <i>Carry Select</i>	48
4.3 Somadores RIC.....	51
4.4 Resultados comparativos entre as arquiteturas de somadores	58
5 Projeto de Sistemas Tolerantes a falhas.....	63
5.1 Tolerância a falhas aplicada a Circuitos Integrados	65
5.2 Processo de Fabricação Específico.....	66
5.3 Técnicas de proteção	66

5.3.1	Técnicas de proteção baseadas em redundância de hardware.....	67
5.3.1.1	Redundância estática.....	67
5.3.1.2	Redundância Dinâmica	68
5.3.1.3	Redundância Híbrida.....	70
5.3.2	Redundância Temporal	71
5.3.3	Redundância de hardware combinada com Redundância temporal.....	72
5.3.4	Técnicas baseadas em Redundância de Informação	74
5.3.5	Técnicas de tolerância a falhas aplicadas à FPGAs	75
6	Experimentos e Resultados Práticos	79
6.1	Arquiteturas de somadores Protegidas.....	80
6.1.1	Detalhes de implementação dos somadores protegidos com - TMR.....	80
6.1.2	Detalhes de implementação dos somadores protegidos com TR	84
6.1.3	Detalhes de implementação dos somadores protegidos com DWC+TR ..	87
6.2	Diretivas de compilação da ferramenta Quartus II	90
6.3	Resultados de síntese para os Somadores Protegidos com TMR	92
6.4	Resultados de síntese para somadores protegidos com TR.....	97
6.5	Resultados de síntese para somadores protegidos com DWC+TR	106
6.6	Análise da Proteção dos Somadores Contra SETs	117
7	Conclusão.....	121
8	Referências.....	124
	ANEXO A – Código VHDL de um RCA de 4 bits Utilizando as Diretivas de Compilação.	130

1 Introdução

O desenvolvimento de sistemas computacionais tolerantes a falhas não é um tema recente, haja vista que atualmente diversos equipamentos eletrônicos já utilizam algum mecanismo de tolerância a falhas com a finalidade de aumentar a confiabilidade de seu funcionamento. Tal preocupação começa a estender-se também aos circuitos integrados fabricados com tecnologias CMOS (*Complementary Metal-Oxide Silicon*) estado-da-arte.

O constante avanço da tecnologia de fabricação CMOS VDSM (*Very Deep Sub-Micron*) é caracterizado por uma diminuição drástica das dimensões dos transistores, a qual tem por objetivo permitir maiores densidades de integração (ou seja, mais transistores por milímetro quadrado de chip) e velocidades de operação mais altas. Para isto, alguns ajustes no processo de fabricação CMOS têm sido necessários, dentre eles a redução da tensão de limiar (*threshold*) e da tensão de alimentação. Tais ajustes, aliados aos efeitos causados pela redução das dimensões dos transistores, como a diminuição das capacitâncias internas, têm tornado os circuitos cada vez mais suscetíveis a falhas transientes (COHEN, 1999; IROM et al., 2002; MAVIS; EATON, 2002), também referenciadas na literatura como *soft errors*.

Atualmente, a principal fonte de falhas transientes nos circuitos integrados é a radiação (BAUMANN, 2001; BAUMANN, 2005). Partículas ionizantes, advindas principalmente da atividade solar, ao colidirem com uma região sensível de um circuito (dreno de um transistor que se encontra desligado) (BAUMANN, 2005; MESSENGER, 1982), podem ocasionar a geração de uma trilha de ionização entre o dreno e o substrato do transistor, permitindo o estabelecimento de um pulso de corrente (O'BRYAN et al., 2002). Se essa corrente possuir amplitude e duração suficientes para carregar (ou descarregar) o nodo, gerando assim um pulso de tensão transiente, este

pode vir a ser interpretado como uma mudança de nível lógico. Caso o mesmo não venha a ser mascarado logicamente, ou eletricamente, ou chegue ao registrador de saída dentro da janela de amostragem (*latching window*), uma falha não-permanente ocorrerá no circuito.

Se uma partícula energizada colidir com uma região sensível de um circuito combinacional, ocasionando a geração de um pulso transiente de tensão, tem-se o fenômeno chamado de *Single-Event Transient* ou SET (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002; VIOLANTE, 2003). Por outro lado, se uma partícula colide com uma região sensível de um elemento de memória, ocasionando uma mudança do valor lógico ali armazenado (também chamado de *bit-flip*), atribui-se o nome de *Single-Event Upset* ou simplesmente SEU (BAUMANN, 2001; DODD; MASSENGIL, 2003; NICOLAIDIS, 2005).

Atualmente, os dispositivos programáveis tipo FPGA se constituem em uma opção muito utilizada para a implementação física de circuitos e sistemas eletrônicos. Tais dispositivos, por serem fabricados com tecnologia CMOS estado-da-arte, estão vulneráveis a falhas transientes. Há alguns anos atrás tais efeitos eram uma preocupação apenas para sistemas embarcados que utilizavam FPGAs em missões espaciais, onde a atividade cósmica é mais intensa. Em função disto, algumas empresas desenvolveram famílias de FPGAs que utilizam algumas técnicas de proteção, notadamente nas células de memória, com vistas a aplicações espaciais e militares. Entretanto, em função do acréscimo de hardware e da baixa demanda, o preço por peça de tais componentes é muito elevado, podendo chegar a U\$ 1.000,00 (informação verbal)¹. Dessa forma, existe uma tendência natural dos projetistas em aplicar técnicas de projeto tolerante a falhas transientes em níveis mais altos, de modo a poder utilizar componentes FPGA convencionais, visto que técnicas no nível físico somente são aplicáveis no caso de se querer projetar matrizes de FPGAs ou no caso de projeto físico dedicado (chip configurado pelas máscaras, ou *mask-programmed*).

Atualmente, a maioria das técnicas que têm sido propostas para a proteção de arquiteturas implementadas em FPGAs se remetem aos efeitos causados por SEUs.

¹ Informação fornecida por Melanie Berg da NASA em palestra proferida na VIII escola de Microeletrônica, em Porto Alegre, em maio de 2006.

Porém, devido à contínua redução das dimensões dos transistores, está previsto que em 2010 a taxa de falhas provocadas por SETs será equivalente à taxa de falhas provocadas por SEUs (SHIVAKUMAR, 2002). Assim, uma preocupação com os efeitos causados por SETs começa despertar a atenção dos projetistas que, por sua vez, começam a propor técnicas de proteção contra tais efeitos. A maioria das técnicas de alto nível utilizadas atualmente, tanto para proteção contra SEUs quanto para proteção contra SETs, baseia-se em redundância de hardware (CARMICHAEL, 2001), redundância temporal (NICOLAIDIS, 1999) ou em uma combinação de ambas (LIMA; CARRO; REIS, 2003).

O objetivo deste trabalho é a investigação de arquiteturas de somadores rápidos protegidas contra SETs em FPGAs da Altera. Para a proteção dos somadores foram utilizadas 3 técnicas no nível arquitetural: TMR (*Triple Module Redundancy*), TR (*Time Redundancy*) e DWC+TR (*Duplication with Comparison combined with Time Redundancy*). O objetivo é analisar os impactos proporcionados pela aplicação destas técnicas aos somadores quanto à utilização de recursos e quanto à velocidade de operação, bem como analisar o nível de proteção proporcionado pela aplicação das técnicas. Para a análise do nível de proteção foram realizadas campanhas de injeção de falhas nos somadores.

As arquiteturas de somadores investigadas neste trabalho foram *Ripple Carry* (RCA), *Carry Select* (CSA) e *Re-computing the Inverse Carry-in* (RIC), sendo que a proposta e avaliação desta última arquitetura se constitui em uma das principais contribuições deste trabalho.

Dada a implementação de um novo tipo de somador rápido, é também um objetivo deste trabalho realizar uma análise do desempenho e dos recursos exigidos por ele, a fim de que o novo somador possa ser avaliado comparativamente com as arquiteturas de somadores consolidadas, como RCAs e CSAs.

A presente monografia está organizada como segue:

O capítulo 2 apresenta os efeitos causados pela radiação nos materiais semicondutores, decorrentes do constante avanço da tecnologia CMOS.

O capítulo 3 apresenta um estudo sobre as principais características dos dispositivos FPGAs, com especial atenção aos FPGAs da família Stratix II da Altera, utilizados neste trabalho para implementação das arquiteturas.

O capítulo 4 apresenta as características dos somadores RCA e CSA e também da nova arquitetura de somador implementada neste trabalho, o RIC. Ao final do capítulo são apresentados resultados comparativos entre os somadores implementados, com relação ao uso de recursos e desempenho.

O capítulo 5 apresenta um estudo das principais técnicas de tolerância a falhas utilizadas para a proteção de sistemas eletrônicos, bem como algumas técnicas utilizadas em FPGAs.

No capítulo 6 são apresentados os detalhes de implementação dos somadores protegidos com as 3 técnicas propostas neste trabalho: TMR, TR e DWC+TR. Na seqüência, são analisados os resultados referentes à aplicação das técnicas aos somadores, quanto ao desempenho e ao uso de recursos. Por fim, são apresentados os resultados de validação referentes à injeção de falhas nos somadores protegidos.

Finalmente, com base nos resultados obtidos, o capítulo 7 apresenta as conclusões referentes à nova arquitetura de somador proposta, bem como as conclusões referentes à aplicação das técnicas de proteção aos somadores.

2 Os Efeitos da Radiação Sobre os Semicondutores

O constante avanço da tecnologia CMOS – *Complementary Metal-Oxide-Semiconductor*, que tem por objetivo aumentar a densidade de integração e a velocidade de operação dos circuitos digitais, tem causado uma diminuição drástica das dimensões dos transistores. Diante disso, alguns ajustes no processo de fabricação CMOS têm sido necessários, dentre eles a redução da tensão de limiar (*threshold*) e da tensão de alimentação. Tais ajustes, aliados aos efeitos causados pela redução das dimensões dos transistores, como a diminuição das capacitâncias dos nós internos, têm tornado os circuitos cada vez mais suscetíveis a falhas transientes (COHEN, 1999; IROM et al., 2002; MAVIS; EATON, 2002), também referenciadas na literatura como *soft errors*.

Atualmente, a principal fonte de falhas transientes nos circuitos integrados é a radiação (BAUMANN, 2001; BAUMANN, 2005). No espaço, a atividade solar é a principal responsável pela radiação, a qual é constituída por partículas carregadas (ou energéticas), tais como elétrons, prótons ou íons pesados, ou de radiação eletromagnética, a qual pode ser constituída por raios X, raios gama ou luz ultravioleta (BARTH, 1997; BAUMANN, 2001).

A influência da radiação nos materiais é medida através da energia e do fluxo de partículas. O fluxo corresponde ao número de partículas que em um segundo atravessam um material em uma área de 1 cm^2 . Quanto mais próximo da superfície da terra, menor é o fluxo e a energia das partículas (NORMAND; BAKER, 1993).

Levando-se em conta o que foi explicado no parágrafo anterior, a preocupação com o funcionamento de circuitos integrados operando na superfície da terra parece um

tanto distante. Porém, o constante avanço da tecnologia CMOS tem tornado o tamanho dos transistores cada vez menor. Dessa forma, as capacitâncias internas a estes também têm reduzido seus valores. O resultado deste processo é uma maior vulnerabilidade dos circuitos integrados a falhas provocadas por partículas de menor intensidade, justamente as mais freqüentes na superfície da terra (JOHNSTON, 2000; DUPONT; NICOLAIDIS; ROHR, 2002).

Atualmente, na superfície da terra, três tipos de mecanismos são responsáveis pela geração de partículas energéticas. Dentre elas podemos citar as partículas alfa, raios cósmicos de alta energia e raios cósmicos de baixa energia (TOSAKA et al., 1998; BAUMANN; SMITH, 2000; BAUMANN, 2001; LERAY et al., 2004; BAUMANN, 2005; BORKAR, 2005). As partículas alfa, em se tratando de circuitos digitais encapsulados, são emitidas por impurezas tais como urânio e Tório presentes no próprio encapsulamento do circuito. Já a interação entre raios cósmicos de alta energia e átomos na atmosfera, é responsável pela geração de nêutrons de alta e baixa energia. Os nêutrons de alta energia produzem partículas carregadas quando interagem com os componentes elásticos e inelásticos do material atingido. Já os nêutrons de baixa energia, geram partículas energéticas quando interagem com o boro, elemento muito utilizado na dopagem de material tipo P no silício.

Vários fatores devem ser levados em consideração para que a ação de uma partícula venha a se tornar efetivamente uma falha. Uma delas é a coleta de carga (*Charge collection*) ou Q_{coll} , que se refere à quantidade de carga depositada no silício quando da ação de uma partícula. Outro importante fator é chamado carga crítica (*Critical Charge*) ou Q_{crit} , que corresponde à sensibilidade do silício quanto ao excesso de cargas depositadas. Esta última assim como o Q_{coll} , depende também de vários outros fatores (BAUMANN, 2005). Para que um *soft-error* realmente se caracterize como uma falha transiente, Q_{coll} deve ser maior que Q_{crit} . Para que se tenha uma idéia, a colisão de um íon com o substrato de silício produz um par elétron lacuna para cada 3.6 eV (*eleto Volt*) de energia perdida pelo íon. Normalmente, utiliza-se o termo LET – *Linear Energy Transfer* para medir a quantidade de energia perdida pela partícula por unidade de comprimento, quando esta atravessa um material (DODD; MASSENGILL, 2003; BAUMANN, 2005). A unidade de medida do LET é MeV/cm²/mg.

2.1 Os Efeitos da Radiação Sobre os Circuitos Integrados

Diversos efeitos podem ser observados quando da colisão de partículas com materiais semicondutores dos circuitos integrados. Os efeitos variam de simples falhas no funcionamento dos circuitos até mesmo a danificação destes.

A junção de polarização reversa do dreno do transistor que se encontra desligado (estado *off*) é considerada a região mais sensível do transistor (MESSENGER, 1982; GADLAGE et al., 2004; BAUMANN, 2005). Isto se deve ao fato de suas próprias características internas de polarização. A Fig. 1 ilustra o comportamento do silício quando da presença de uma partícula carregada.

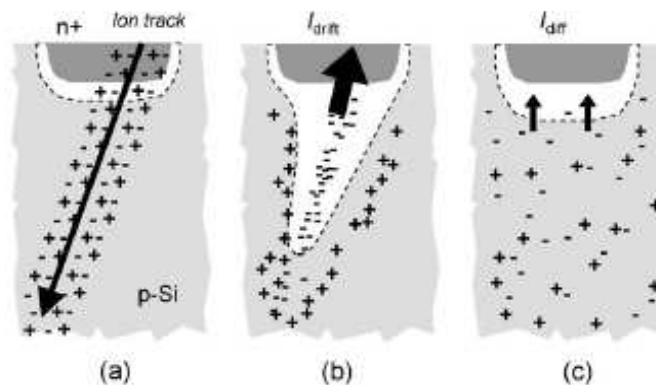


Figura 1– Comportamento da região sensível de um transistor, dada a ação de uma partícula carregada.

Fonte: BAUMANN, 2005.

Quando a partícula choca-se com a região sensível do transistor, uma trilha de ionização, constituída por pares elétron-lacuna e uma alta concentração de portadores, é formada por onde passa a partícula, conforme ilustra a Fig. 1 (a). Devido à proximidade da trilha com a região de depleção² da junção, é gerado então um campo magnético. Este campo por sua vez, faz com que a região de depleção tome a forma de um funil (MESSENGER, 1982; BAUMANN, 2005) e as cargas negativas sejam atraídas rapidamente para a região positiva do silício (dreno), conforme ilustrado na Fig. 1 (b). Após, a região de depleção toma seu formato natural novamente e por difusão, as cargas negativas continuam a se deslocar para a região positiva (mas desta vez de

² Região na qual há ausência de cargas.

maneira mais lenta), conforme ilustrado na Fig. 1 (c). Como resultado deste fenômeno, é gerado um pulso de corrente entre o dreno do transistor e o substrato (ou poço, conforme o tipo de transistor) (O'BRYAN et al., 2002). Caso essa corrente possua amplitude e duração suficientes para carregar (ou descarregar) um nodo de saída, um pulso de tensão transiente é gerado. Tal pulso de tensão pode vir a ser interpretado como uma mudança de nível lógico, caso este tenha uma amplitude maior que a margem de ruído da porta subsequente (DODD; MASSENGILL, 2003). A quantidade de ionização e a intensidade da corrente gerada pela colisão da partícula carregada com o silício é diretamente proporcional à quantidade de carga perdida pela partícula (LIMA, 2003; KARNIK; HAZUCHA; PATEL, 2004).

A Fig. 2 ilustra a forma do pulso de corrente gerado pela ação da partícula no silício. É possível verificar de maneira transparente as três fases da ação da partícula no silício.

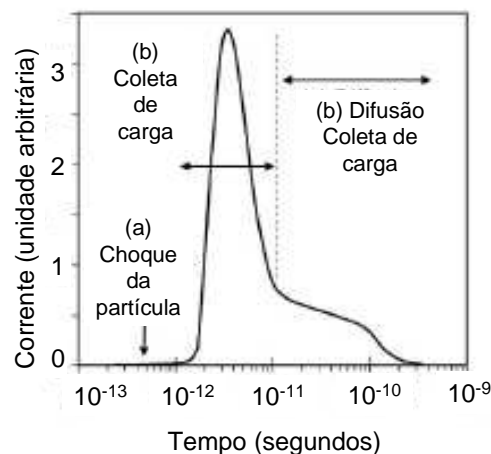


Figura 2 – Forma do pulso de corrente gerado após a colisão de uma partícula carregada com a região sensível do silício.

Fonte: BAUMANN, 2005.

2.2 *Single Event Effects (SEEs)*

Fenômenos causados pela ação da colisão de partículas em circuitos integrados são também referenciados na literatura com SEEs – *Single Event Effects*. Dependendo

das conseqüências causadas quando da ação de uma partícula, estes podem ser divididos em três categorias (O' BRYAN et al, 1998; LIMA, 2000): *Destructive SEE*, *Hard SEE* e *Soft SEE*.

2.2.1 Destructive SEE

Caracteriza um dano físico permanente no circuito causado pela ação de uma partícula. O mais comum SEE é o chamado *Single Event Latchup* –SEL. Dada a ação de uma partícula, se a quantidade de energia despendida pela partícula ultrapassar os limites de corrente suportados pelo transistor, por exemplo, uma falha permanente poderá se deflagrar no circuito tornando-o defeituoso (KOLASINSKI et al., 1979).

2.2.2 Hard SEE

Também referenciados como *hard errors*, caracterizam um desvio de comportamento permanente em uma das funcionalidades do circuito. O fenômeno neste caso caracteriza-se como SHE – *Single Hard Error*. Um bit de memória com um valor fixo em “1” ou “0” pode ser um exemplo neste caso (LABEL et al., 2000).

2.2.3 Soft SEE

São aquelas falhas provocadas pela colisão de partículas energizadas com as regiões sensíveis dos circuitos que não danificam o circuito. A ação da partícula apenas causa um mau funcionamento momentâneo do circuito, de forma que instantes depois o circuito continua a funcionar normalmente. Esta característica de não permanência de falhas é a responsável por caracterizar os SEEs como *soft errors* (KARNICK; HASUCHA; PATEL 2004). A proteção contra *soft errors* se constitui no principal objetivo deste trabalho. Assim, nas subseções seguintes serão apresentados os dois principais tipos de *soft errors*: *Single Event Transient* – SET e *Single Event Upset* – SEU. Levando-se em conta que a proteção contra SETs é o principal alvo deste trabalho este será abordado em um nível de detalhes maior.

2.2.3.1 *Single Event-Upset (SEU)*

Uma partícula carregada pode colidir diretamente com uma região sensível de um elemento de memória. Caso isto venha a acontecer, conforme a quantidade de energia da partícula, o valor armazenado na célula de memória pode vir a ser invertido, efeito este também chamado de *bit-flip*. A este fenômeno é atribuído o nome de *Single Event-Upset* ou SEU (BAUMANN, 2001; DODD; MASSENGIL, 2003; NICOLAIDIS, 2005).

Considerando as tecnologias submicrônicas atuais, a probabilidade de ocorrência de SEUs é bem maior do que a probabilidade de ocorrência de SETs. Isto se deve principalmente ao fato das células de memória serem projetadas com transistores mínimos ou próximos dos mínimos (BAZE; BUCHNER, 1997). Assim, mecanismos de geração de SEUs e também técnicas para proteção contra SEUs têm sido muito investigados nos últimos anos. Entretanto, conforme já explicitado anteriormente, com a evolução da tecnologia CMOS, os circuitos tem se tornado cada vez mais vulneráveis aos efeitos de colisões de partículas carregadas, de modo que a frequência de SETs tem aumentado significativamente nos últimos anos.

O comportamento de uma célula de memória SRAM (*Static Random Access Memory*) quando da colisão de uma partícula carregada é mostrado Fig. 3. Note que inicialmente o circuito está estável (Fig. 3 (a)). Após, uma partícula carregada então colide com o dreno do transistor PMOS (*P-channel Metal-Oxide-Semiconductor*) (MP0) gerando um pulso de tensão na saída do transistor, conforme a Fig. 3 (b). Isto faz com que seja carregada, mesmo que momentaneamente, a capacitância associada a este dreno, ligando assim o transistor controlado por ele. Levando-se em conta as características de projeto, típicas de qualquer tipo de elemento de memória, que utilizam dimensionamentos diferentes dos transistores e laços de realimentação, o resultado da carga do capacitor pode ser interpretado erroneamente como uma mudança de nível lógico. Dessa forma, será efetuada uma troca do valor armazenado na célula de memória caracterizando um SEU (Fig. 3 (c)).

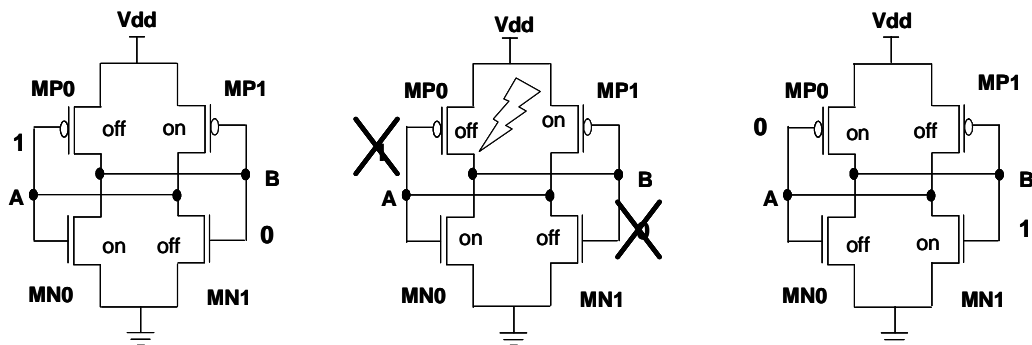


Figura 3 – *Single event upset* (SEU) em uma célula de memória SRAM.

Um outro fenômeno que não pode deixar de ser citado, são os chamados MBU – *Multibit Upset* (DODD; MASSENGILL, 2003; LIMA, 2003). Dependendo da quantidade de energia despendida pela partícula incidente, mais de um bit pode vir a ser atingido causando a troca de mais de um valor armazenados em células de memória. Vale ressaltar que a possibilidade da ocorrência de tal fenômeno possui proporções bem pequenas comparadas com SEU (BAUMANN, 2005).

2.2.3.2 *Single Event-Transient* (SET)

Se uma partícula carregada colide com uma região sensível de um circuito combinacional, ocasionando a geração de um pulso transiente de tensão, o fenômeno associado recebe o nome de *Single-Event Transient* ou SET (ALEXANDRESCU; ANGHEL; NICOLAIDIS, 2002; VIOLANTE, 2003). Se o pulso de corrente gerado possuir amplitude e duração suficientes para carregar (ou descarregar) uma capacitância de saída, gerando assim um pulso de tensão transiente, tal pulso de tensão pode vir a ser interpretado como uma mudança de nível lógico. Em média, a duração de um pulso de tensão provocado por SET varia entre 1 ps e 100 ps (ANGHEL; NICOLAIDIS, 2000; HEIJMEN; NIEUWLAND, 2006). Embora a geração de um pulso de tensão no circuito não se caracterize como algo previsto por si só, este não necessariamente irá provocar uma falha transiente no circuito. Esta somente se concretizará de fato, caso o pulso venha a se propagar pelo circuito e por sua vez, venha a ser capturado (erroneamente) por um elemento de memória, caracterizando

um *soft error*. A Fig. 4 apresenta o comportamento de um circuito, dada a colisão de uma partícula energizada com uma região sensível de um transistor pertencente ao bloco combinacional. Cabe aqui salientar que tal fato se caracteriza como uma falha transiente no circuito, haja vista que na próxima borda de relógio, o valor errôneo armazenado pelo registrador será sobrescrito, fazendo com que o circuito continue a operar normalmente.

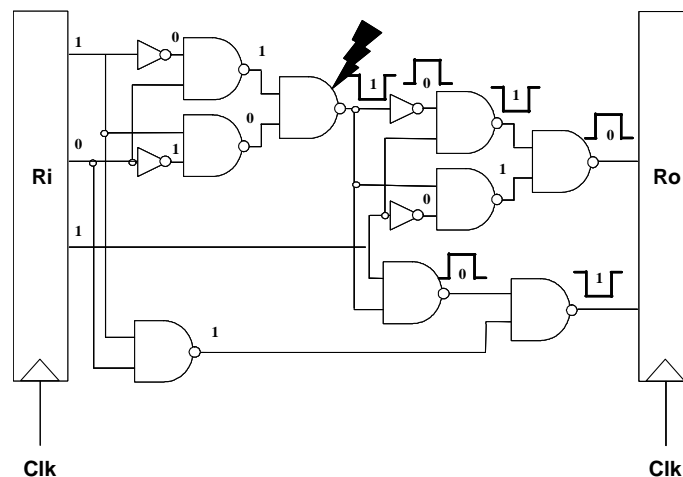


Figura 4 - *Single-event transient* (SET) e sua possível propagação em um bloco combinacional.

Conforme mostrado na Fig. 4, um efeito causado por um SET somente se caracterizará como uma falha no funcionamento do circuito caso o pulso de tensão se propague pelo circuito e posteriormente, seja armazenado por um elemento de memória. Neste ponto, é possível perceber que a probabilidade de um pulso se propagar até um elemento de memória, ou até mesmo a uma saída primária do circuito, 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 registradores propriamente ditos. Assim, quanto maior o número de portas lógicas que um pulso transiente precisa atravessar, maior a probabilidade deste ser mascarado antes de ser armazenado por elemento de memória ou ser enviado para uma saída primária (NIEUWLAND; JASAREVIC; JERIN, 2006).

Dentre os mecanismos que contribuem para que haja uma diminuição da probabilidade de que um pulso venha a se propagar pelo circuito podemos citar o mascaramento lógico, o mascaramento elétrico e o mascaramento por *latching window*.

O mascaramento lógico está associado a controlabilidade das portas lógicas (GOLDSTEIN, 1979; ABRAMOVICI; BREUER; FRIEDMAN, 1990; BUSHNELL; AGRAWAL, 2000). Se um pulso ao se propagar pelo circuito atingir uma das entradas de uma porta lógica, este pode vir a ser mascarado logicamente caso esta porta possua o valor controlante em ao menos uma de suas outras entradas. A Fig. 5 mostra um exemplo desta situação. Note que, independente do valor que esteja presente na entrada B, o valor da saída da porta NAND permanece imutável.

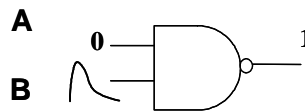


Figura 5 – Mascaramento lógico.

O mascaramento elétrico se dá devido às propriedades elétricas das portas lógicas no caminho do pulso. Quanto maior a duração de um pulso causado por um SET maior é a probabilidade deste conseguir se propagar pelo circuito e eventualmente, ser capturado por um elemento de memória.

Se a duração de um pulso, provocado por SET, possuir uma largura maior que o tempo de propagação de uma porta, este por sua vez, não será atenuado. Porém, se a largura do pulso possuir um valor menor que o tempo de propagação da porta, este tenderá a ser atenuado. Caso o pulso venha a ter uma largura menor que metade do tempo de propagação de uma porta, então o pulso será completamente atenuado caracterizando um mascaramento elétrico (NICOLAIDIS, 1999; ALEXANDRESCU; ANGHEL; NICOLAIDS, 2004). A Fig. 6 (pictoricamente) apresenta o processo de atenuação de um pulso quando da passagem por uma seqüência de inversores.

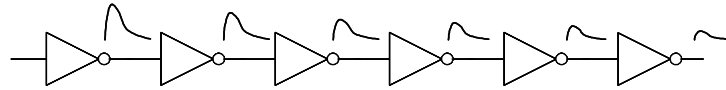


Figura 6 – Processo de atenuação de um pulso de tensão quando este se propaga pelo circuito³.

E por fim, além dos tipos de mascaramentos citados acima, existe também o mascaramento por *latching window*. Este, por sua vez, ocorre quando o pulso atinge o elemento de memória fora da janela de amostragem. Se t_{su} e t_h 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 (com probabilidade 1) caso tenha duração entre $(t - t_{su})$ e $(t + t_h)$, no mínimo. Caso o pulso termine antes de $(t - t_{su})$ ou inicie depois de $(t + t_h)$, diz-se que ocorre mascaramento por *latching window* (SHIVAKUMAR et al., 2002; ANGHEL; LEVEUGLE; VANHAUWAERT, 2005; WIRTH et al., 2005). Caso a duração de um pulso venha a possuir uma largura maior que o período de relógio do circuito, este será capturado por elemento de memória com probabilidade 1 (GADLAGE, 2004).

A Fig. 7 ilustra o exato momento em que um pulso é capturado por um elemento de memória, pois chega no registrador dentro da janela de amostragem do registrador e também alguns exemplos de pulso mascarados.

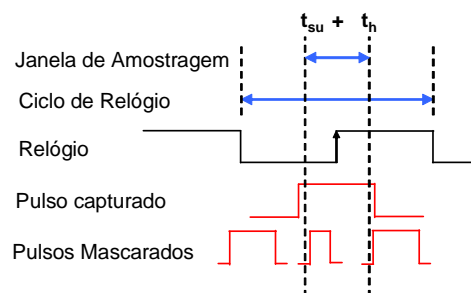


Figura 7 – Mascaramento por latching Window.

Um outro fator importante com relação à probabilidade de um SET tornar-se uma falha transiente é a frequência de operação dos circuitos digitais. À medida que esta se

³ A lógica inversora foi ignorada a fim de pôr em evidência a atenuação do pulso.

torna maior, maiores são as chances de uma falha ser capturada por elemento de memória, haja vista o aumento de janelas de amostragem proporcionadas pela alta frequência (GADLAGE, 2004).

Com intuito de analisar computacionalmente os efeitos causados pela ação das partículas no silício, mais precisamente os efeitos causados pelo pulso gerado por um SET, alguns autores propuseram alternativas de modelagem de um pulso de tensão. A Fig. 8 ilustra o modelo proposto por Messenger (1982). O modelo caracteriza a fenômeno de deposição de carga na saída de uma porta lógica como uma fonte de corrente cujo comportamento obedeceria a dupla exponencial mostrada na equação (1) a seguir:

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

Na equação (1), I_0 é 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. 8 mostra a modelagem proposta por Messenger e as formas de onda típicas para uma tecnologia estado-da-arte (100nm), considerando diversos valores de I_0 e $t\beta$.

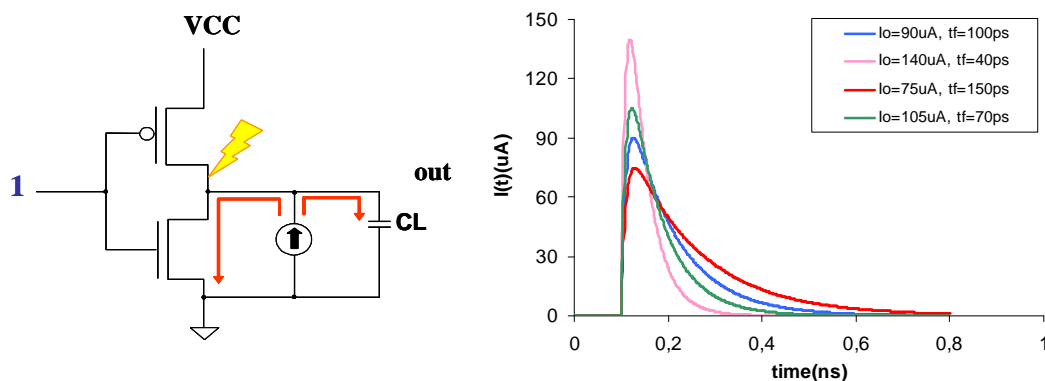


Figura 8 – Modelo de Messenger para um pulso de tensão gerado a partir de um SET.

A modelagem de um pulso de tensão transiente é de suma importância quando se deseja realizar a análise da sua propagação, em um circuito elétrico. Embora a equação de Messenger modele de forma razoavelmente precisa os efeitos causados pela ação de partículas nos circuitos, sua utilização para estes fins é bastante difícil,

dada a complexidade para modelagem de cada porta. Assim, vários modelos têm sido propostos e utilizados com este mesmo fim (DAHLGREN; LIDÉN, 1995; OMAÑA et al., 2003; ALEXANDRESCU; ANGHEL; NICOLAIDS, 2004; WIRTH et al., 2005). Embora, os vários modelos propostos sejam de valia importância quando se trata da análise da propagação de SETs, por fugirem ao escopo deste trabalho, estes não serão abordados.

Historicamente, a preocupação com falhas em circuitos integrados provocadas por partícula energizadas, assim como já explicitado na seção anterior, era apenas com relação a SEU, ou seja, partículas que se chocam diretamente com elementos de memória. Isto se deve principalmente à densidade espacial dos elementos de memória e à quantidade de informação que estes podem armazenar (MAHESHWARI; KOREN; BURLESON, 2003). Porém, o constante avanço da tecnologia CMOS tem tornada substancial a preocupação com os efeitos causados por partículas que se chocam com a parte combinacional dos circuitos integrados. Como resultado da redução das dimensões dos transistores e conseqüente diminuição das capacitâncias internas, a probabilidade de que um pulso consiga se propagar através de um circuito tem aumentado, haja vista que mascaramentos elétricos tem se tornado cada vez menos freqüentes. É previsto que no ano de 2010, a taxa de *soft errors* provocados por SETs seja equivalente à taxa de falhas provocadas SEUs em memórias não protegidas (SHIVAKUMAR, 2002).

3 FPGAs - *Field Programmable Gate Arrays*

Os FPGAs surgiram com o intuito de solucionar um *gap* que estava sendo enfrentado na década de 1980 quanto à implementação de sistemas digitais. Dispositivos tais como SPLDs (*Simple Programmable Logic Device*), CPLDs (*Complex Programmable Logic Device*), PROMs (*Programmable Read Only memorys*), PLAs (*Programmable Logic Arrays*) etc. lançados no mercado na década de 1970, embora tivessem a característica de reconfigurabilidade, não podiam descrever funções lógicas complexas. Isto ocorre devido ao fato das funções lógicas internas a estes componentes serem pré-definidas pelo fabricante (MAXFIELD, 2004). Já os chamados projetos dedicados ou configuráveis por máscaras (*Full-custom, Standard-cells, Gate arrays* etc), fabricados desde a década de 1960, podiam fazer uso de milhões de portas lógicas e definir qualquer função lógica. Porém, o alto custo do projeto, aliado ao alto custo da confecção das máscaras para a fabricação, inviabilizava o uso desta opção para baixas demandas de peças por ano. Além disto, qualquer falha de projeto no desenvolvimento de um circuito integrado configurado por máscaras é irreparável.

Assim, em 1984, a Xilinx lançou o primeiro FPGA no mercado. Este, por sua vez, era baseado na tecnologia CMOS e utilizava células de memória SRAM para propósitos de configuração. Inicialmente, os FPGAs eram utilizados apenas como interconexões entre blocos lógicos maiores, máquinas de estado de média complexidade e processamento de algumas pequenas tarefas (MAXFIELD, 2004). Com o aprimoramento dos FPGAs, estes passaram a ocupar espaço em outras áreas tais como telecomunicações e redes de interligação de computadores. Diante do fato de tais dispositivos poderem ser reconfigurados de maneira prática e ainda possuírem um curto tempo para lançamento no mercado, estes passaram a concorrer diretamente com os

ASICs (*Application-Specific Integrated Circuits*). Hoje em dia, dispositivos FPGAs utilizam dezenas de milhões de portas lógicas, interface de entrada e saída de alta velocidade e muitos deles possuem pequenos processadores embutidos, com intuito de aumentar o desempenho (MAXFIELD, 2004).

Atualmente, estão disponíveis no mercado três principais tipos de tecnologias de FPGAs: baseados em SRAM ("*SRAM-based*"), baseados em anti-fusível ("*anti-fuse-based*") e baseados em memória FLASH ("*FLASH-based*") (ROOSTA, 2004).

Os FPGAs baseados em memórias SRAM caracterizam-se principalmente pela reconfigurabilidade. Toda configuração destes dispositivos é armazenada em memórias SRAM, o que possibilita que os FPGAs possam ser reconfigurados quantas vezes for necessário. Além disso, a fabricação de memórias SRAM utiliza a mesma tecnologia CMOS aplicada ao resto do dispositivo. Assim, nenhum processo especial se faz necessário quando da criação das memórias nestes FPGAs.

Por outro lado, FPGAs baseados em memórias SRAM possuem algumas desvantagens. Dada a característica de volatilidade de uma memória RAM (*Random Access Memory*), os FPGAs necessitam ser reconfigurados toda vez que o dispositivo é desligado. Desta forma, estes FPGAs necessitam ainda fazer uso de um elemento de memória externo para armazenar a configuração do dispositivo.

Os FPGAs baseados em fusíveis, ao contrário dos FPGAs baseados em SRAM, utilizam fusíveis para a configuração dos dispositivos. Isto significa que estes FPGAs, uma vez programados, não podem mais ser modificados. Esta característica de não reconfigurabilidade faz com que estes sejam menos utilizados que os FPGAs baseados em SRAM. Alguns autores referem-se às interconexões destes dispositivos como sendo *Rad-Hard*, ou imunes aos efeitos da radiação (MAXFIELD, 2004). Tal característica credencia que tais FPGAs sejam muito utilizados em missões espaciais e militares, necessitando apenas a proteção das memórias.

Por fim, existem ainda os FPGAs baseados em memória FLASH. Tais dispositivos são muito similares aos FPGAs baseados em memória SRAM. A sua principal vantagem com relação a este último é a não volatilidade deste tipo de dispositivo, ou seja, sua configuração não é perdida, dada a ausência de energia. Esta característica possibilita que estes dispositivos possam ser configurados mesmo

desligados. A grande desvantagem dos FPGAs baseados em memória *FLASH* refere-se ao processo de fabricação destes dispositivos. Em se tratando da tecnologia CMOS padrão, são necessárias 5 etapas extras para fabricá-los (MAXFIELD, 2004). Esta característica implica em um maior custo, fazendo com que estes não sejam muito utilizados.

A seção 3.1 apresenta a arquitetura padrão interna de FPGAs baseados em memória SRAM enquanto que na seção 3.2 são apresentados alguns detalhes da arquitetura interna dos FPGAs da família Stratix II, família escolhida para a implementação das arquiteturas protegidas desenvolvidas neste trabalho.

3.1 Arquitetura dos FPGAs

A estrutura interna de um FPGA é constituída principalmente por três recursos de hardware: blocos lógicos que configuram a lógica do usuário, os blocos de entrada e saída, responsáveis por interconectar a estrutura interna do FPGA aos pinos de saída, e as interconexões configuráveis, que têm o objetivo de realizar a ligação entre as estruturas de hardware internas ao dispositivo. A Fig.9 ilustra de maneira superficial a estrutura geral interna de um FPGA.

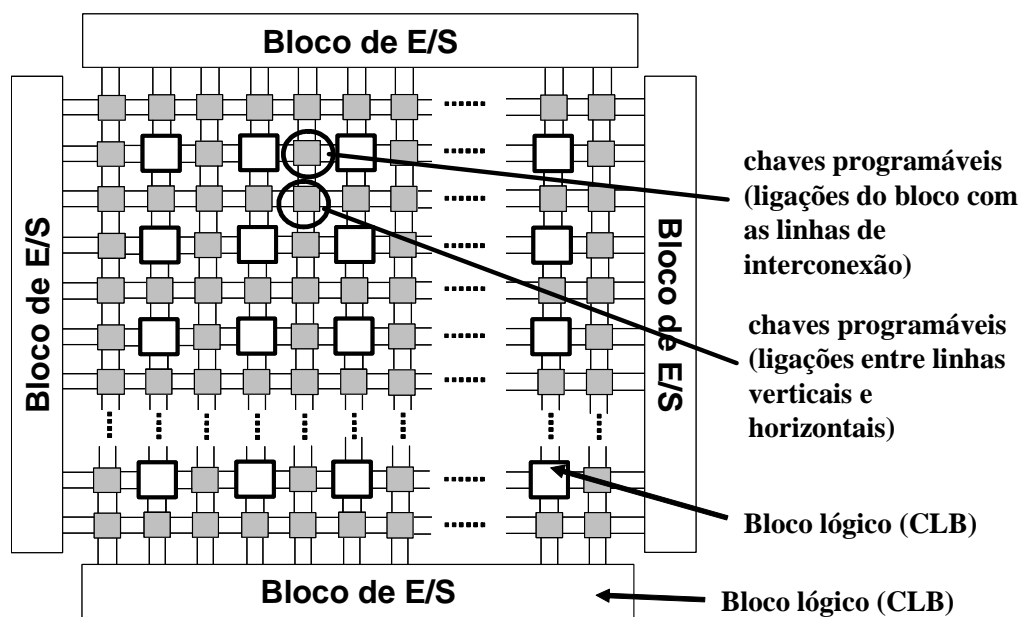


Figura 9 – Estrutura interna geral de um FPGA.

Percebe-se na Fig.9 que os blocos lógicos estão distribuídos em forma de vetores (*arrays*) formando uma enorme matriz. Em cinza mais claro estão representados os blocos de matrizes de chaveamento, que também possuem características de reconfigurabilidade. Estas matrizes têm por objetivo interligar os diversos recursos de hardware dentro do FPGA.

A estrutura responsável por implementar as funções lógicas dentro dos FPGAs são os blocos lógicos. A arquitetura interna destes blocos varia de acordo com o fabricante. Porém, geralmente esses blocos apresentam algumas estruturas básicas. Dentre elas podemos citar a LUT (*Look Up Table*) que utiliza células de memória SRAM para implementação de pequenas funções lógicas, um registrador que pode funcionar tanto como *flip-flop* quanto *latch* e um multiplexador. A Fig.10 apresenta a estrutura de um bloco lógico que faz uso dos recursos de hardware citados anteriormente.

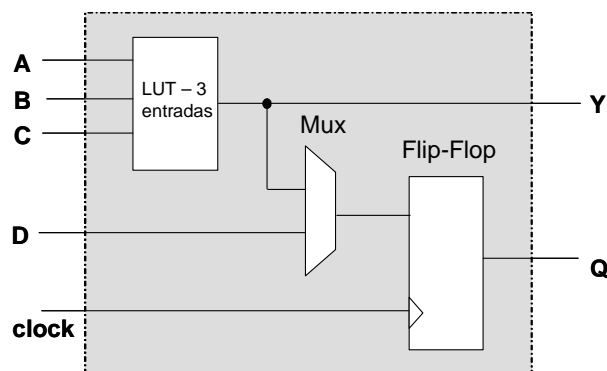


Figura 10 – Estrutura geral de um bloco lógico.

Conforme a Fig. 10, foi utilizado um bloco lógico com uma LUT de três entradas, o que significa que o bloco possui a capacidade de implementar qualquer função lógica de três entradas. O princípio de funcionamento de uma LUT é simples. Um conjunto de valores de entrada é utilizado como um índice para o acesso aos dados armazenado nas células de memória SRAM da LUT. Com o intuito de melhor esclarecer o princípio de funcionamento de uma LUT, a Fig.11 apresenta um fluxo intuitivo de como uma LUT poderia ser configurada para implementar a equação (1) abaixo. Basicamente, os valores possíveis para a saída X serão armazenados em células SRAM.

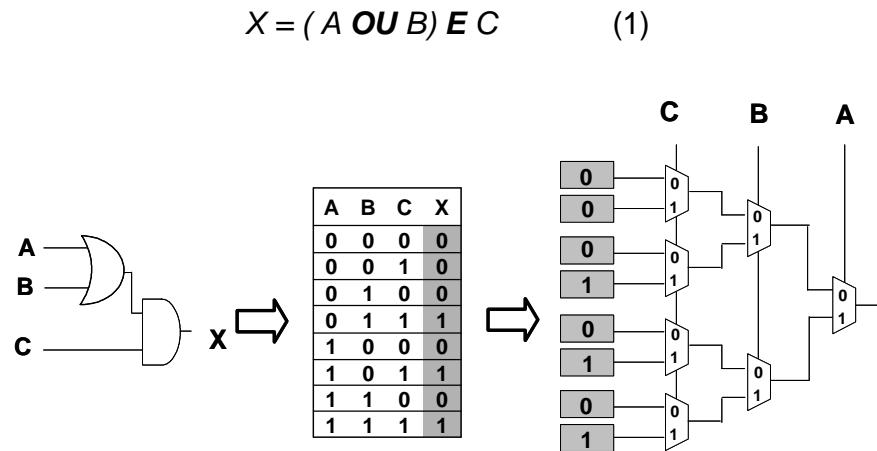


Figura 11 – Implementação de uma função lógica de três entradas utilizando uma LUT.

Embora a utilização de LUTs de três entradas seja uma realidade, existe um consenso entre os fabricantes de FPGAs que LUTs de quatro entradas suprem a maioria das aplicações e portanto, são as mais utilizadas (MAXFIELD, 2004). A arquitetura normalmente utilizada para a implementação de uma LUT de quatro entradas pode ser observada na Fig.12. Note que os multiplexadores são construídos com transistores de passagem, com intuito de diminuir o custo de hardware.

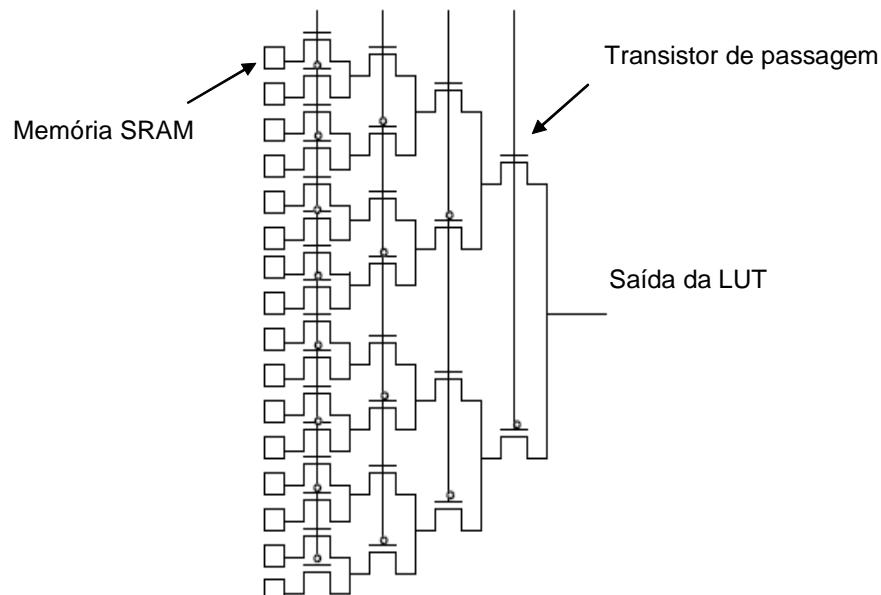


Figura 12 – LUT de 4 entradas implementada com transistores de passagem.

Fonte: LIMA, 2003.

Com relação às matrizes de chaveamento, a configurabilidade dessas conexões, assim como nas ALUTs, é realizada utilizando-se memórias SRAM. Uma das implementações de matriz de chaveamento mais simples é a que utiliza um transistor NMOS (*N-channel Metal-Oxide-Semiconductor*) com o terminal *gate* controlado por uma célula de memória SRAM. Essa implementação é conhecida como *pass-transistor switch* (BROWN, 2005). A Fig.13 apresenta um exemplo da aplicação de uma matriz de chaveamento utilizando as características acima citadas. Um bloco lógico produz uma saída f_1 que é enviada para a linha de conexão horizontal adjacente. O sinal produzido pela saída f_1 pode ser enviado para uma linha vertical utilizando a chave de conexão programável. Se o valor armazenado na célula de memória SRAM possuir um valor 0, o transistor NMOS associado a esta célula de memória estará desligado e o caminho para a coluna correspondente a este estará fechado. Caso contrário, o transistor poderá conectar uma linha horizontal a uma linha vertical.

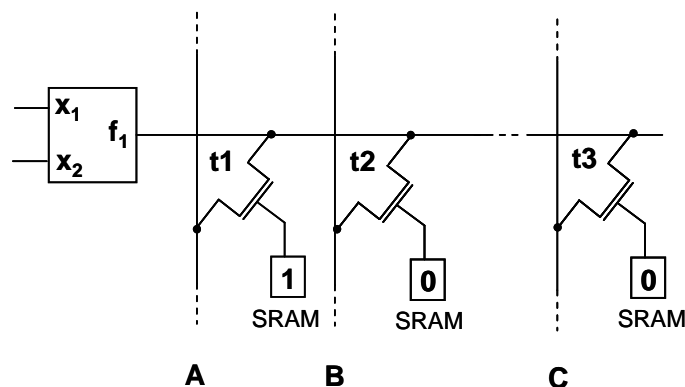


Figura 13 – *Pass-Transistor switches* em FPGAs.

Note na Fig.13 que o transistor “t1” está ligado, estabelecendo um caminho entre a saída f_1 e o ponto A.

Outra característica importante dos FPGAs é a maneira como estes são configurados. Conforme foi mostrado anteriormente, os FPGAs baseados em SRAM possuem uma grande quantidade de células de memória utilizadas para configurar a lógica do usuário. Assim, existem diversas ferramentas computacionais que auxiliam o projeto de sistemas digitais baseado no uso de FPGAs. Tais ferramentas permitem a realização das etapas de síntese e verificação necessárias, gerando ao final um arquivo

bit-file (ou *bit-stream*) que irá configurar a lógica do usuário no FPGA escolhido. O processo de carregamento do arquivo (*upload*) é conhecido como configuração do *bitstream* (MAXFIELD, 2004). Neste trabalho, como veremos mais adiante, foi utilizada a ferramenta Quartus II da Altera (ALTERA, 2007).

Em se tratando de FPGAs baseados em SRAM, as células de configuração podem ser visualizadas como um único e longo registrador deslocador, conforme mostra a Fig. 14.

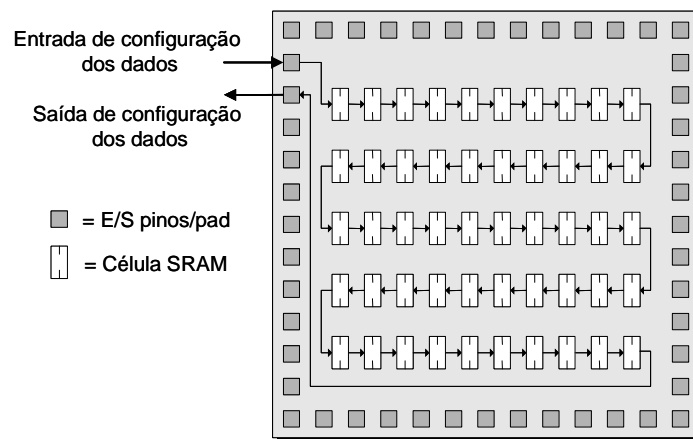


Figura 14 – Visualização de um conjunto de células SRAM organizadas com um registrador deslocador.

Note que neste caso, são utilizados apenas um pino de entrada e um de saída para configuração do FPGA. Tal característica é comum à maioria dos FPGAs comerciais, e tem por objetivo evitar o desperdício de pinos.

3.2 Família Stratix II, Altera

Levando-se em conta que todos os experimentos práticos deste trabalho foram realizados utilizando os dispositivos da família Stratix II, a presente seção descreve as principais características dessa família.

Os dispositivos da família Stratix II são baseados em tecnologia SRAM, utilizam tecnologia de fabricação CMOS de 90 nm com tensão de alimentação de 1.2-V e fazem uso de um novo tipo de estrutura de bloco lógico. Essa estrutura, quando comparada a outras famílias de FPGAs, possui a característica de maximizar o desempenho,

podendo ainda atingir densidades de integração de até 180.000 elementos Lógicos (LEs – *Logic elements*), blocos lógicos equivalentes utilizados por outras famílias (ALTERA, 2005a).

Os dispositivos da família Stratix II apresentam uma arquitetura padrão assim como a referenciada na seção anterior. Mas esta, por se tratar de uma arquitetura real e atual (seu lançamento no mercado se deu em 2004), apresenta alguns blocos de hardware a mais com o intuito de aumentar o desempenho do FPGA. A Fig. 15 mostra a arquitetura de um FPGA da família Stratix II. Os blocos lógicos responsáveis por implementar a lógica dentro do FPGA, também chamados de LABs (*Logic Array Blocks*), ficam organizados em duas dimensões: linhas e colunas, formando uma enorme matriz. Além disso, entre os LABs existem outros blocos de hardware, tais como estruturas de memórias RAM (M-512 RAM, M-4K RAM, e M-RAM) e os chamados blocos DSP - *Digital Signal Processing*. Em se tratando das estruturas de memórias, estas podem ser utilizadas com intuito de implementar blocos de memórias maiores e também outras estruturas como FIFOs (*First In First Out*), por exemplo. Já os blocos DSP podem ser utilizados para implementar operações de multiplicação e filtros.

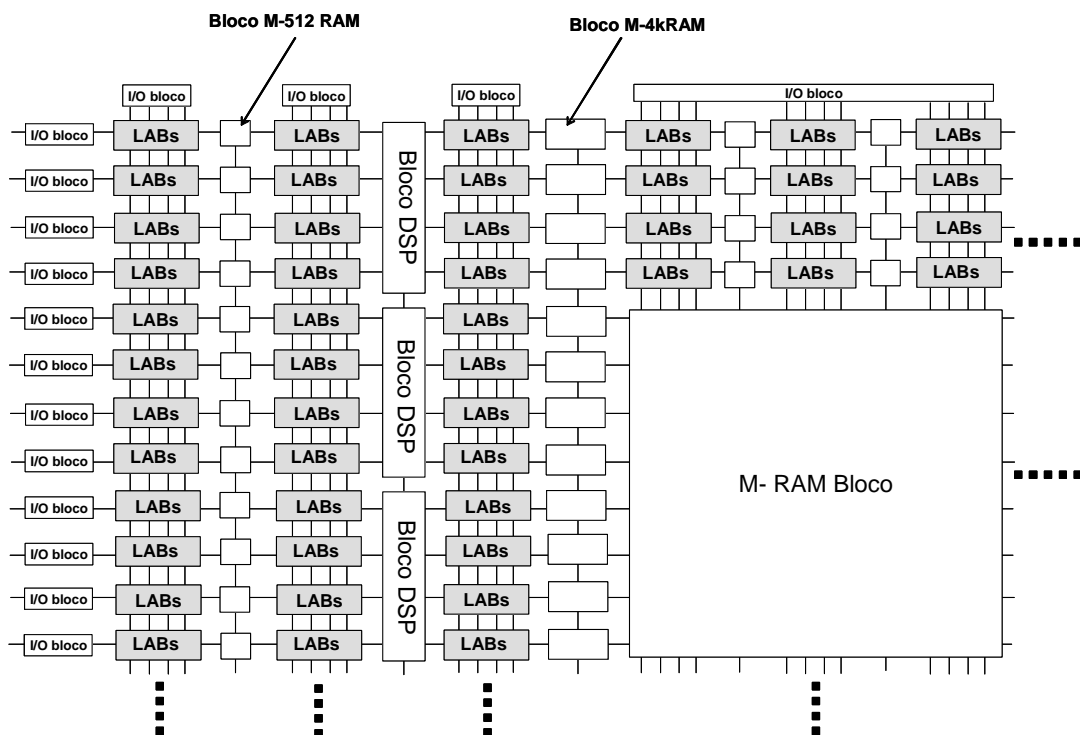


Figura 15 – Arquitetura interna dos FPGAs da família Stratix II.

3.2.1 Blocos Lógicos – LABs

Na família Stratix II cada LAB consiste de oito ALMs (*Adaptive Logic Modules*), blocos aritméticos compartilhados, sinais de controle do LAB, conexões locais e conexões entre registradores. As conexões locais são responsáveis por fazerem a ligação entre ALMs que pertencem ao mesmo LAB. Utilizando uma conexão denominada *direct link interconnection* é possível também a comunicação direta entre LABs adjacentes utilizando também as conexões locais. Já as linhas de conexões entre registradores conectam a saída de um registrador, pertencente a um ALM, a outro registrador pertencente a um ALM adjacente. A Fig. 16 mostra a estrutura interna de um LAB.

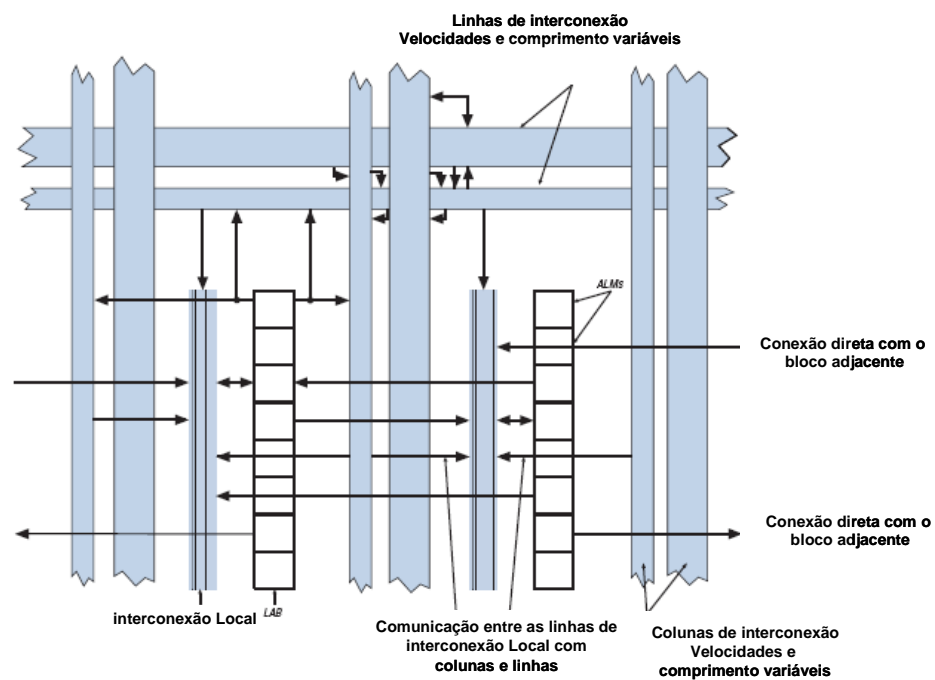


Figura 16 – Estrutura interna de um LAB.

Fonte : ALTERA, 2005a.

3.2.2 ALM - *Adaptive Logic Module*

A estrutura ALM provida pela ALTERA é a principal característica da família Stratix II. O ALM consiste em um bloco combinacional de 8 entradas aliado a uma LUT

de tamanho variável que pode ser dividida em duas ALUTs – *adaptive LUTs*. De posse destas 8 entradas, um ALM pode implementar várias combinações de duas funções. Além disso, um ALM pode implementar de maneira eficiente qualquer função lógica de seis entradas e algumas funções lógicas de sete entradas. Esta habilidade de dividir a LUT é o que o torna adaptativo. Um sumário de algumas configurações que um ALM pode alcançar é apresentado na Fig. 17.

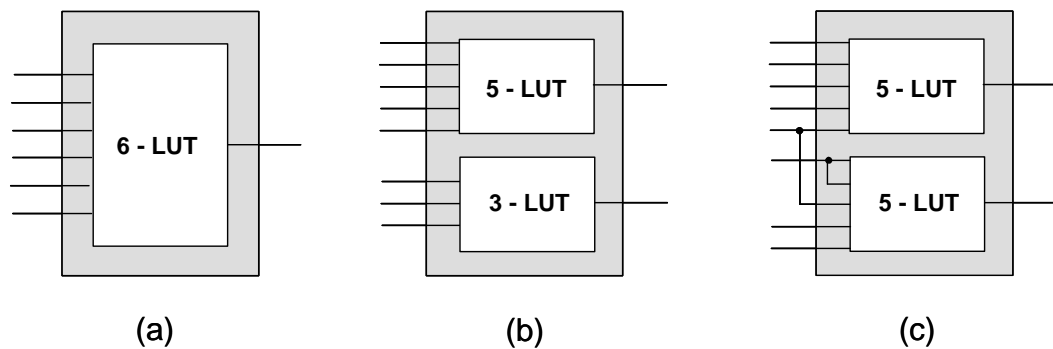


Figura 17 - Algumas configurações suportadas por um ALM.

A Fig. 17 (a) apresenta um ALM configurado para implementar qualquer função de 6 entradas. Já na Fig. 17 (b), o ALM apresenta uma configuração de forma a implementar uma função de 3 entradas e uma função de 5 entradas. Isto só é possível devido ao fato de em um ALM, as entradas serem independentes umas das outras. E por fim, a Fig. 17 (c) apresenta um ALM configurado de forma a implementar duas funções de 5 entradas. Para isso, duas entradas necessitam ser compartilhadas (ALTERA, 2005a). Essa capacidade de se adaptar de acordo com a função a ser implementada, provido pelos dispositivos da família Stratix II, é responsável pelo ganho em área e eficiência, quando comparado com outras arquiteturas de FPGAs (ALTERA, 2005b).

Um bloco ALM possui ainda blocos de hardware dedicados, tais como dois registradores programáveis, dois blocos de somadores *full-adder*, a cadeia de *carry*, uma cadeia para cálculos aritméticos compartilhada e a cadeia de registradores. Através destes blocos dedicados, o ALM pode implementar várias funções aritméticas e registradores de deslocamento. A arquitetura interna de um ALM, em um alto nível de abstração, é mostrada na Fig. 18.

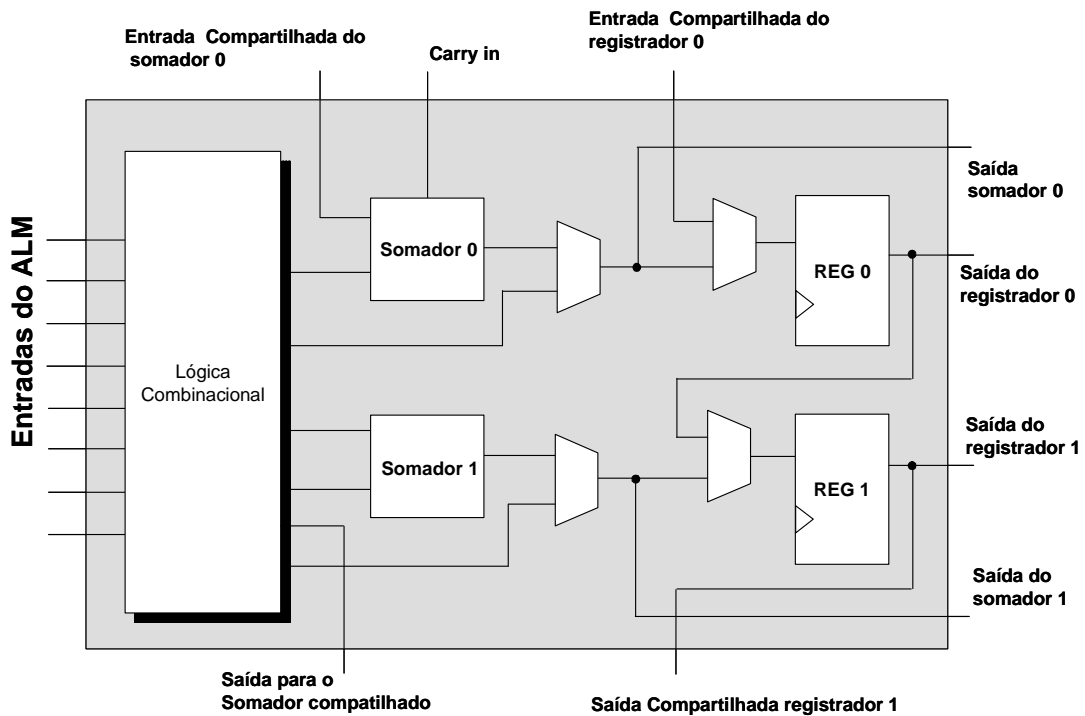


Figura 18 – Estrutura interna de um ALM em alto nível.

Com relação à Fig. 18, é importante notar que a maneira como irá se comportar cada ALM está relacionada à forma como a ferramenta de síntese irá sintetizar a arquitetura descrita em VHDL. Embora a Fig. 18 não mostre nada a respeito de sinais de controle, é importante que se saiba que vários sinais de controle são utilizados dentro da estrutura do ALM, mais precisamente oito sinais. A forma como esses sinais são configurados é que vai determinar a maneira como os blocos lógicos irão se comportar. A Fig. 19 mostra a arquitetura interna de um ALM em maior grau de detalhamento. Através desta é possível verificar os sinais de controle e como são feitas as interligações entre os dispositivos de hardware disponibilizados dentro de um ALM.

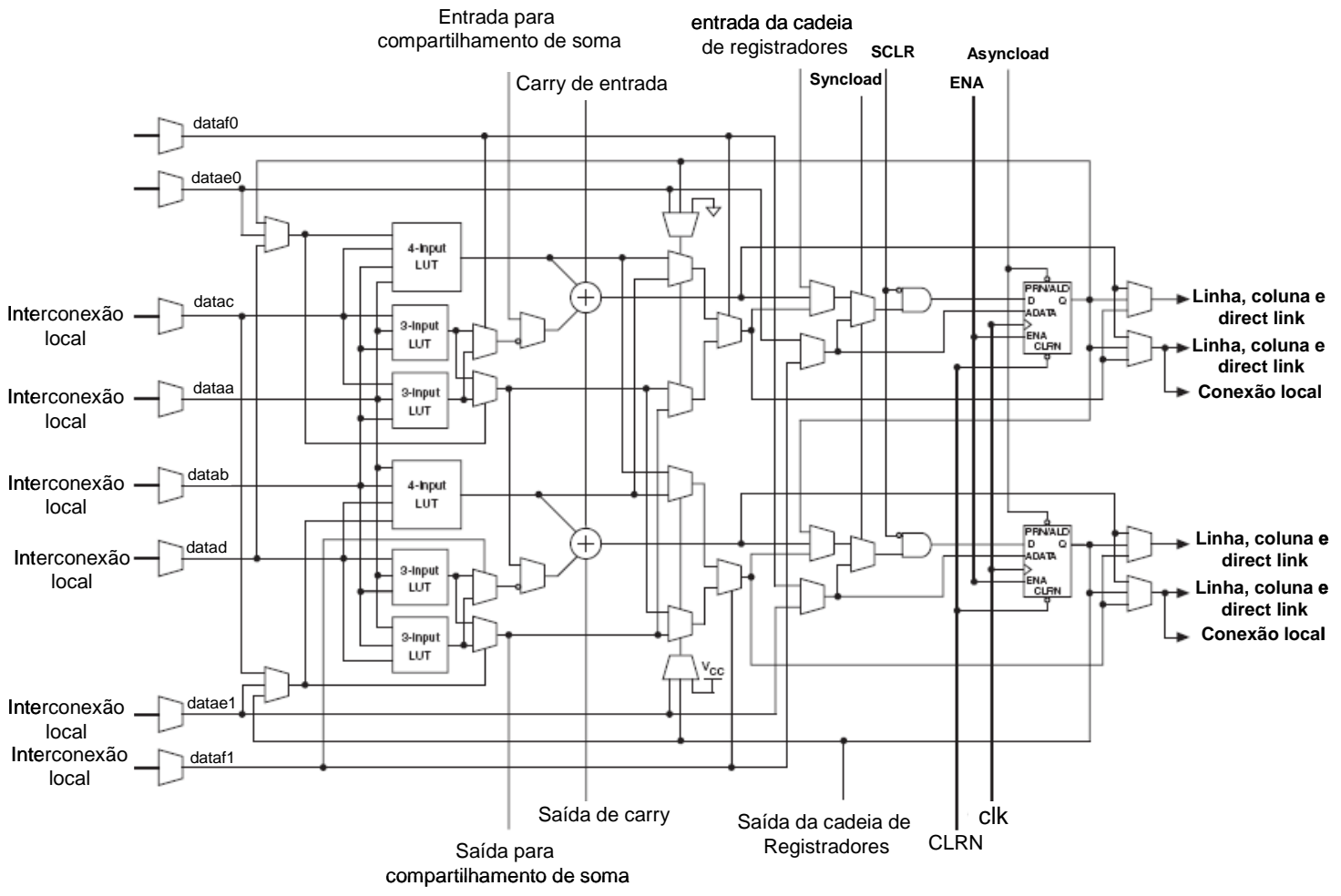


Figura 19 – Estrutura interna de um ALM em um nível maior de detalhamento.

Fonte: ALTERA, 2005a.

É possível observar na Fig. 19 que cada ALM possui dois conjuntos de saídas que interagem com conexões locais, linhas ou colunas de interconexão. Uma LUT, um somador ou um registrador podem utilizar estas saídas de maneira independente. Isto permite que um registrador utilize uma das saídas enquanto que um somador, por exemplo, utilize a outra saída. Esta característica melhora o aproveitamento de recursos dentro do FPGA, haja vista que tanto um registrador quanto um bloco combinacional podem ser utilizados para fins diferentes dentro de um mesmo ALM (ALTERA, 2005a).

3.2.3 Interconexões – Stratix II

A ligação entre os blocos de hardware dentro do FPGA é realizada por uma série de interconexões de comprimentos e velocidades diferentes, organizadas em forma de linhas e colunas, assim como os blocos lógicos. Essas estruturas, por sua vez, são chamadas de *MultiTrack interconnect* e utilizam a técnica *DirectDrive*. As conexões podem ser classificadas de duas maneiras: linhas de conexão e colunas de conexão.

Dentre as linhas de conexão estão as chamadas *Direct Link interconnect* (linhas de interconexão direta), R4 e R24. As linhas de interconexão direta permitem comunicação diretas entre ALMs adjacentes. Já linhas de conexão R4 são responsáveis por realizarem a comunicação entre conjuntos formados por no máximo 4 LABs pertencentes a mesma linha. Para isto utilizam linhas de interconexão de alta velocidade. E por fim as interconexões R24 que abrangem distâncias de até 24 LABs. Em se tratando do R24, a cada 4 LABs existe um conexão com uma linha ou coluna, porém o R24 não possui nenhuma ligação direta com nenhum LAB. A comunicação neste caso deve ser realizada através de conexões R4, C4. Para que se tenha uma idéia, a R24 pode atravessar um bloco de memória M-RAM que possui 589.824 RAM bits.

Em se tratando das colunas de conexão, estas operam de maneira similar às linhas de conexão, podendo ser classificadas como C4, C16 e conexões diretas entre ALMs. As conexões C4 e C16 se assemelham às conexões R4 e R24, sendo estas responsáveis por fazer a ligação entre blocos que pertençam à mesma coluna. Já as conexões diretas entre ALMs, permitem que somadores, registradores sejam

compartilhados entre ALMs de forma a implementar-se por exemplo, operadores aritméticos maiores, contadores, etc.

A Fig. 20 apresenta de forma simplificada as interconexões, apresentada no parágrafo anterior, entre as estruturas LABs dentro do FPGA.

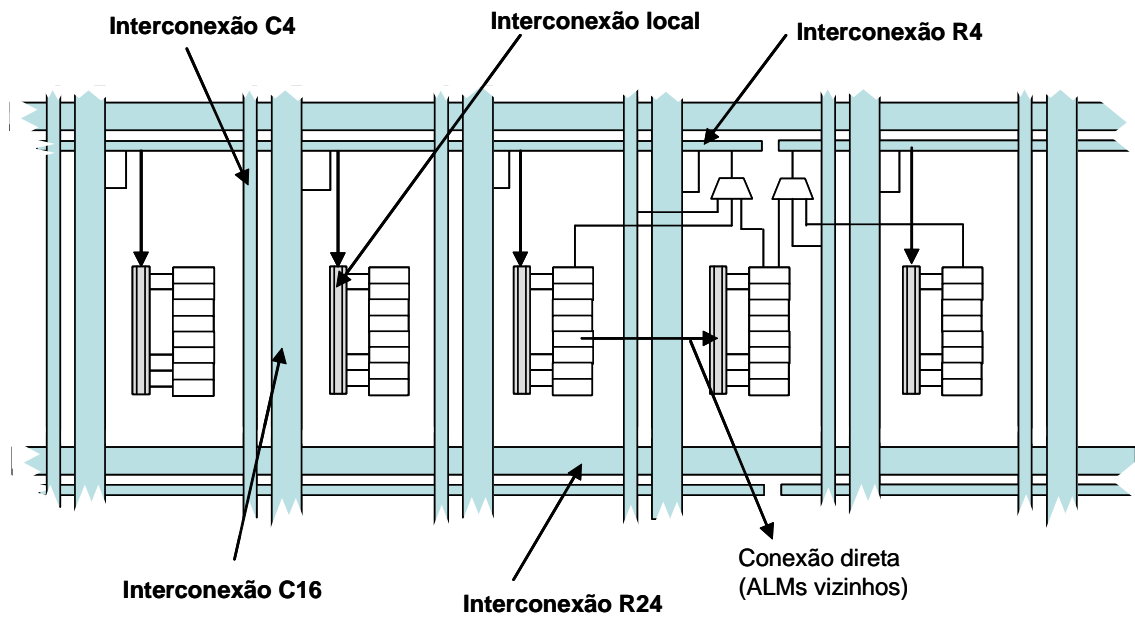


Figura 20 – Linhas de conexão dentro de dos dispositivos da Stratix II.

4 Somadores

Circuitos somadores se constituem em circuitos fundamentais que servem de base para a maioria das demais operações aritméticas. Assim sendo, o alto desempenho destes circuitos é essencial não apenas para circuitos somadores em si, mas também para subtratores, multiplicadores, divisores e outros operadores aritméticos que fazem uso da adição. Portanto, o desempenho de qualquer processador aritmético digital depende, de forma intensa, do desempenho dos circuitos somadores (OKLOBDZIJA, 2001).

A eficiência de um somador depende fundamentalmente da maneira como ele é projetado. Para ser mais preciso, seu desempenho depende primordialmente da forma como são computados os *carries* de saída dos estágios de soma.

Várias arquiteturas de somadores rápidos têm sido propostas com o intuito de reduzir o atraso da propagação dos carries. Dentre estas citam-se os somadores *Carry-Select*, *Carry Lookahead*, *Manchester* (HWANG, 1979; OKLOBDZIJA, 2001; PORTO, 2003). Tais arquiteturas buscam melhorar o desempenho manipulando as equações de *carry* e, no caso do somador *Carry-Select*, realizando modificações arquiteturais.

Nas seções seguintes são apresentadas algumas arquiteturas de somadores rápidos, bem como suas características e aplicabilidade. Em especial, apresenta-se uma nova arquitetura de somador rápido, a qual constitui-se em uma das principais contribuições deste trabalho.

4.1 Somadores *Ripple-Carry*

Dentre as arquiteturas de somadores existentes, o somador *Ripple Carry* (RCA) destaca-se como uma arquitetura de referência. Em geral, as arquiteturas de somadores fazem uso dos RCAs de uma forma ou de outra. Assim, uma perfeita compreensão da filosofia de funcionamento deste circuito é essencial para que mais adiante possam ser apresentadas de maneira mais intuitiva as arquiteturas de somadores rápidos.

A arquitetura de um somador RCA utiliza basicamente dois circuitos, um responsável por realizar o cálculo da soma de três bits de mesmo peso e outro para o cálculo do *carry* de saída. As entradas para estes dois cálculos são dois bits de mesmo peso (um de cada número a ser somado) e um *carry* de entrada. As equações utilizadas por um somador RCA podem facilmente ser extraídas de uma tabela-verdade, como mostra a Fig. 21.

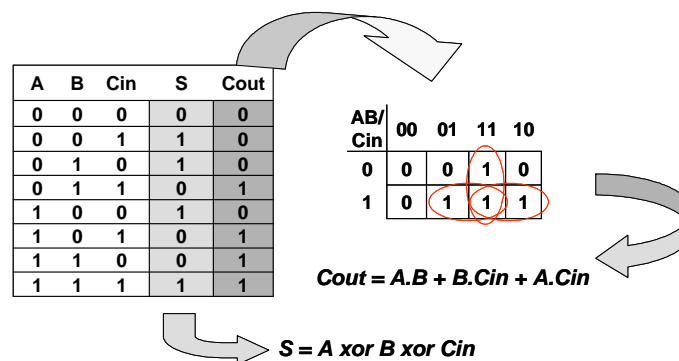


Figura 21 – Extração da arquitetura de um RCA através de sua tabela-verdade.

Observa-se na Fig. 21 que os resultados gerados para a saída S, com base nas entradas A e B, possuem uma característica de verificador de paridade. Desta forma, seu circuito lógico pode ser implementado utilizando-se portas XOR. Através das equações é possível obter-se o circuito lógico de um somador completo (*full adder*) de um bit, conforme mostra a Fig. 22.

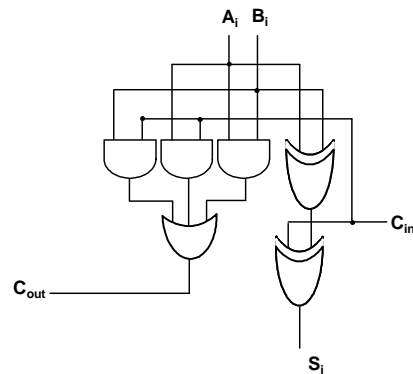


Figura 22 – Somador completo de 1 bit.

Somadores RCA podem ser construídos de maneira trivial, conectando-se o *carry* de saída de um somador completo ao *carry* de entrada do somador completo subsequente. A Fig. 23 apresenta um somador RCA de 4 bits.

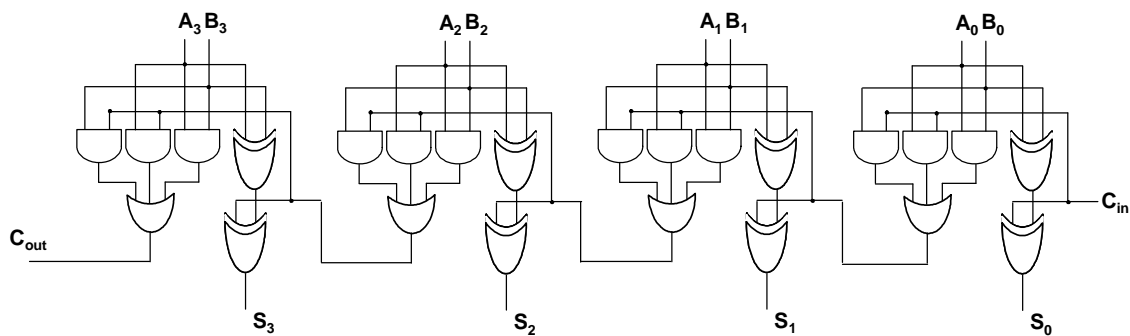


Figura 23 – Somador *Ripple Carry* de 4 bits.

Observando-se a Fig. 23, percebe-se que os somadores RCA não fazem uso de uma quantidade significativa de recursos. Por outro lado, somadores RCA apresentam um desempenho ruim. Note que o cálculo realizado por cada somador completo depende dos *carry* de saída do somador completo anterior. Daí resulta um atraso significativo na propagação do *carry*. A Fig. 24 apresenta o caminho crítico de um somador RCA de 4 bits.

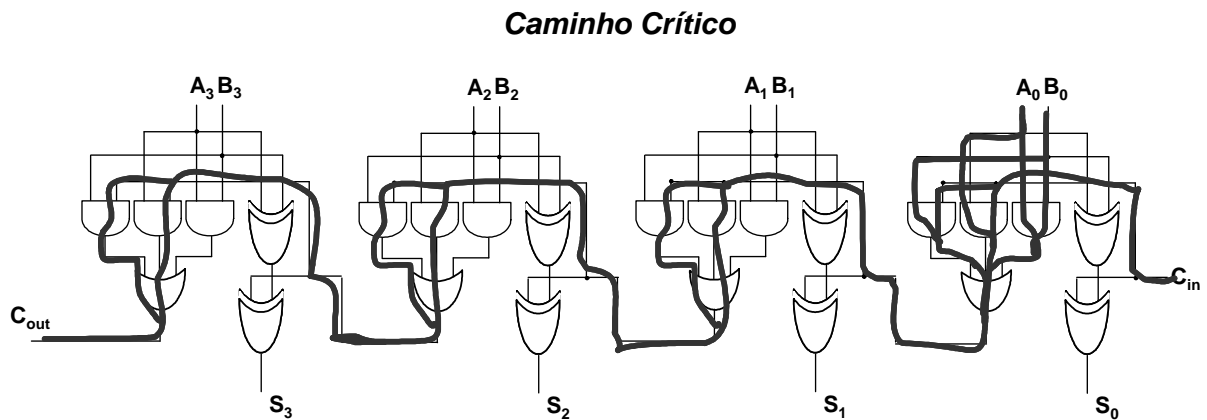


Figura 24 – Caminho crítico de um RCA de 4 bits.

Ressalta-se que quanto mais bits possuir o RCA maior será o atraso gerado por ele. Assim, se o objetivo em um projeto de somador é o desempenho, o uso de somadores RCA se justifica apenas para somadores com poucos bits. Esta característica poderá ser constatada pela análise dos resultados práticos apresentados deste trabalho.

Embora a arquitetura do RCA utilizada neste trabalho tenha sido a apresentada na Fig. 24, tais somadores podem ser construídos também a partir dos chamados meios somadores (Fig. 25). Embora utilize uma quantidade de hardware um pouco menor, quando comparada à arquitetura da Fig. 24, seu desempenho é praticamente o mesmo.

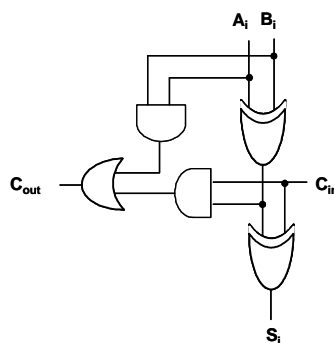


Figura 25 – Somador RCA de 1 bit construído a partir de dois meio-somadores (*half-adders*).

4.2 Somadores *Carry Select*

Os somadores *Carry-Select* (CSA) caracterizam-se por apresentar o melhor desempenho dentre a classe dos chamados somadores rápidos (PORTO, 2003). Porém, para que possam alcançar notável desempenho, fazem uso de um acréscimo de recursos de mais de 100% em relação aos RCAs. Com isto, estes tipos de somadores raramente são utilizados, uma vez que os projetos atuais precisam satisfazer a um compromisso entre desempenho e custo.

O princípio básico do funcionamento dos somadores CSA é dividir o cálculo de uma soma em n seções menores de m bits, possibilitando que estas possam calcular suas parcelas em paralelo, de forma independente umas das outras. Ao final, quando todas as seções tiverem calculado suas parcelas, os resultados são reunidos de forma a representarem o resultado final da soma. A fim de permitir o cálculo independente, cada seção faz uso de dois somadores, sendo um dos somadores responsável pelo cálculo de sua parcela com *carry* de entrada igual a 0, enquanto que o outro é responsável pelo mesmo cálculo, porém, com *carry* de entrada igual a 1. Desta forma, ao contrário do que acontece com somadores RCA, uma seção não necessita esperar o *carry* de saída da seção anterior para realizar a sua parcela do cálculo, pois os cálculos de ambas possibilidades de *carries* de entrada são realizados concomitantemente em cada seção. A implementação de somadores CSA implica, ainda, na utilização de um circuito multiplexador em cada seção, a fim de que seja escolhido qual, dentre os resultados calculados pelos blocos duplicados, será enviado para a saída. A Fig. 26 apresenta a arquitetura de um somador CSA de 8 bits.

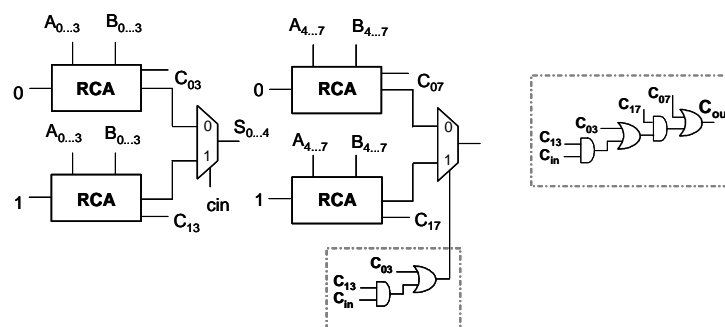


Figura 26 – Somador CSA de 8 bits.

O número de seções em que um somador CSA pode ser dividido, bem como o tipo de somador a ser utilizado em cada uma das seções depende fundamentalmente das restrições de projeto. A configuração mais utilizada dentre os autores é a divisão da soma em seções de 4 bits, assim como a utilização de somadores RCA em cada seção (HWANG, 1979). Vale lembrar que os somadores RCA, quando implementados com poucos bits, possuem um desempenho aceitável. Embora esta seja uma configuração muito utilizada, também podem ser implementados somadores CSA que utilizem outros tipos de arquiteturas de somadores, bem como seções com um número maior de bits.

Note na Fig. 26 que um circuito lógico é utilizado pelas entradas de controle dos multiplexadores em cada seção, a fim de definir qual o resultado correto a ser enviado para a saída. Abaixo são apresentadas as equações utilizadas pelos circuitos lógicos em cada seção, para um somador de 16 bits.

$$B_1 = C_{in} \quad (1)$$

$$B_2 = C_{03} \text{ or } (C_{13} \text{ and } C_{IN}) \quad (2)$$

$$B_3 = C_{07} \text{ or } (C_{17} \text{ and } (C_{03} \text{ or } (C_{13} \text{ and } C_{IN}))) \quad (3)$$

$$B_4 = C_{011} \text{ or } (C_{111} \text{ and } (C_{07} \text{ or } (C_{17} \text{ and } (C_{03} \text{ or } (C_{13} \text{ and } C_{IN})))))) \quad (4)$$

$$B_4 = C_{015} \text{ or } (C_{115} \text{ and } (C_{011} \text{ or } (C_{111} \text{ and } (C_{07} \text{ or } (C_{17} \text{ and } (C_{03} \text{ or } (C_{13} \text{ and } C_{IN}))))))) \quad (5)$$

Com relação às equações, entende-se por C_{03} , o *carry* de saída do RCA de 4 bits que realiza a adição ($A_{0...3} + B_{0...3}$) com *carry* de entrada igual a 0, enquanto que o sinal C_{13} corresponde ao *carry* de saída do RCA que realiza esta mesma adição, porém com *carry* de entrada igual a 1. Os outros sinais seguem este mesmo raciocínio.

A fim de que se possa melhor entender as equações é preciso observar algumas propriedades da soma ao se utilizar dois blocos somadores idênticos, diferenciando-se apenas pela utilização de *carries* de entrada diferentes.

Propriedade 1: *Dados dois números a serem somados, se o somador que realiza o cálculo com carry de entrada igual a 0 produzir um carry de saída igual a 1, pode-se afirmar com certeza que o somador que possui como carry de entrada o valor 1, também produzirá um carry de saída igual a 1.*

A utilização da propriedade 1 pode ser observada em todas as equações apresentadas anteriormente. Percebe-se que, caso os sinais C_{03} , C_{07} , C_{011} e C_{015} no início de cada equação forem iguais a 1, estes por si só definem, independente das outras variáveis da equação, qual dos resultados calculados pelos somadores duplicados em cada seção será enviado para saída.

Propriedade 2: *Dados dois números a serem somados, se o somador que realiza o cálculo com carry de entrada igual a 0 produzir como resultado um carry de saída igual a 0, é necessário que se verifique o resultado gerado pelo somador com carry de entrada igual a 1. Caso o resultado deste também venha a ser 0, então pode-se afirmar com certeza que o carry de saída da seção possui o valor 0. Caso contrário, para que se possa saber qual o carry de saída da seção é necessário que se verifique o carry de saída gerado pela seção antecessora a em questão.*

Esta característica pode ser observada mais claramente na equação (2) apresentada anteriormente. Caso os sinais C_{03} e C_{13} tenham valores iguais a zero, o carry de saída da seção terá certamente valor igual a 0. Caso estes venham a ter valores distintos, faz-se necessária a análise do carry de saída da seção antecessora, neste caso o C_{IN} .

Em relação ao atraso crítico dos somadores CSA, as arquiteturas com 4 e 8 bits possuem atrasos críticos referentes ao maior atraso de uma seção, ou seja, a soma do atraso de um somador RCA com o atraso do multiplexador da seção. A Fig. 27 apresenta um somador CSA de 8 bits com seu caminho crítico destacado. Note ainda que para um CSA de 8 bits o caminho que leva em conta as equações descritas anteriormente ainda não possui um atraso suficiente para o tornar parte do caminho crítico.

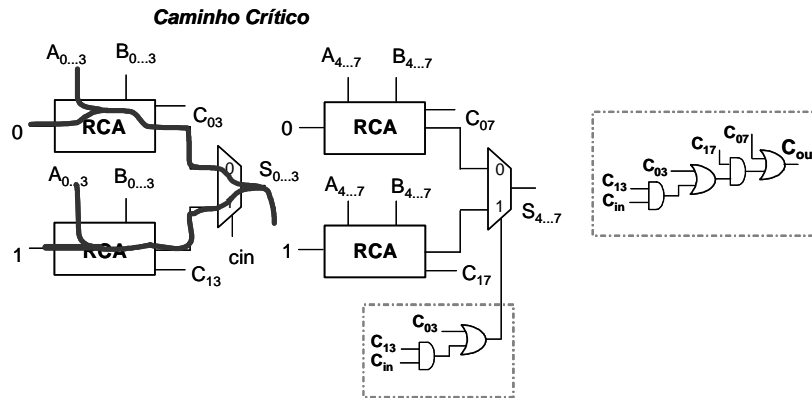


Figura 27 – Caminho crítico de um CSA de 8 bits.

É importante observar que, com relação à equação utilizada para escolha dos valores calculados em cada bloco de uma seção, o tamanho desta cresce conforme o tamanho do somador. Assim, a partir de um determinado número de bits, o atraso crítico dos somadores CSA passa a ser definido pela soma dos atrasos de um somador RCA, da maior equação de uma seção e o atraso do multiplexador. A Fig. 28 ilustra essa característica para um somador de 32 bits.

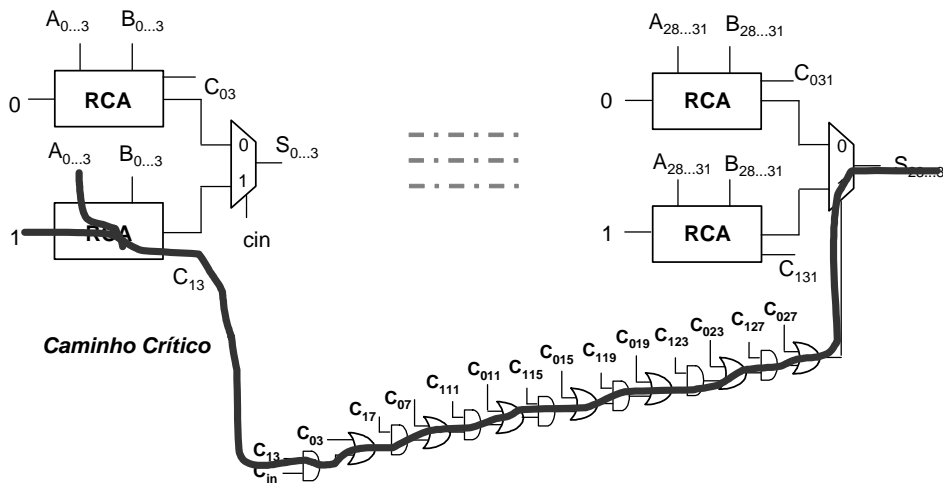


Figura 28 – Caminho crítico de um somador CSA de 32 bits.

4.3 Somadores RIC

O somador RIC (*Re-computing the Inverse Carry-in*) caracteriza-se por ser uma nova arquitetura de somadores rápidos. Embora a implementação de uma nova

arquitetura de somador não seja o foco deste trabalho, o desenvolvimento deste insere-se perfeitamente em seu contexto como uma importante contribuição.

A filosofia de funcionamento do somador RIC baseia-se nos mesmos princípios dos somadores CSA. Inicialmente, este trabalho objetivava realizar um aproveitamento da arquitetura duplicada dos somadores CSA para a aplicação de técnicas de tolerância à falhas, nas quais a redundância do hardware é imperativa. Porém, ao final do experimento, chegou-se a uma arquitetura de somador que embora possua algumas características em comum a um somador CSA, detém também características únicas que fazem deste circuito aritmético um somador inédito.

O somador RIC, ao invés de utilizar dois blocos RCA em cada seção, como no caso do CSA, faz uso de apenas um somador RCA mais um bloco denominado de RB – *Recomputing Block*. Este, por sua vez, realiza a mesma função de um RCA, utilizando porém uma quantidade de hardware menor quando comparado a um circuito somador RCA propriamente dito. A Fig. 29 apresenta a arquitetura de um somador RIC de 4 bits. Percebe-se que o RB é ligado em série com a saída do bloco somador com *carry* de entrada igual a 0, implicando em um pequeno acréscimo de atraso em cada seção do somador. O objetivo principal do circuito RB é converter o resultado gerado por um somador com *carry* de entrada igual a 0 para um resultado tal que este mesmo somador tivesse como *carry* de entrada o valor 1.

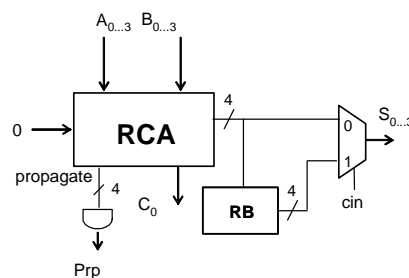


Figura 29 – Somador RIC de 4 bits.

A implementação do circuito RB é derivada de duas propriedades da adição binária (KUMAR, 2003).

Propriedade 1 : Dados dois números a serem somados, se ambos os números possuírem o bit menos significativo igual a 0, o resultado da soma destes números com carry de entrada igual a 1 é idêntico ao resultado da soma com carry de entrada igual a 0, exceto pela troca do valor do bit menos significativo do resultado. A Fig. 30 apresenta um exemplo da aplicação da propriedade anterior.

<table style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 10%;">Cin</td><td style="width: 10%; text-align: center;">0</td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td></tr> <tr><td>A</td><td style="text-align: center;">1 1 0 0</td><td style="text-align: right;">(12)</td><td></td><td></td><td></td></tr> <tr><td>B</td><td style="text-align: center;">0 1 0 0</td><td style="text-align: right;">(4)</td><td></td><td></td><td></td></tr> <tr><td>Output</td><td style="text-align: center;"><u>1 0 0 0</u></td><td style="text-align: right;">(16)</td><td></td><td></td><td></td></tr> </table>	Cin	0					A	1 1 0 0	(12)				B	0 1 0 0	(4)				Output	<u>1 0 0 0</u>	(16)				<table style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 10%;">Cin</td><td style="width: 10%; text-align: center;">1</td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td></tr> <tr><td>A</td><td style="text-align: center;">1 1 0 0</td><td style="text-align: right;">(12)</td><td></td><td></td><td></td></tr> <tr><td>B</td><td style="text-align: center;">0 1 0 0</td><td style="text-align: right;">(4)</td><td></td><td></td><td></td></tr> <tr><td>Output</td><td style="text-align: center;"><u>1 0 0 1</u></td><td style="text-align: right;">(17)</td><td></td><td></td><td></td></tr> </table>	Cin	1					A	1 1 0 0	(12)				B	0 1 0 0	(4)				Output	<u>1 0 0 1</u>	(17)			
Cin	0																																																
A	1 1 0 0	(12)																																															
B	0 1 0 0	(4)																																															
Output	<u>1 0 0 0</u>	(16)																																															
Cin	1																																																
A	1 1 0 0	(12)																																															
B	0 1 0 0	(4)																																															
Output	<u>1 0 0 1</u>	(17)																																															

Figura 30 – Exemplo de uma soma utilizando a propriedade 1 da soma.

Propriedade 2 : Se o resultado de uma adição entre dois números com carry de entrada igual a 0 apresenta o bit menos significativo igual a 1, o resultado da adição destes mesmos dois números, mas com carry de entrada igual a 1 é equivalente sendo necessária a inversão dos m bits do resultado, da esquerda para direita, até que seja encontrado o primeiro 0. Este último também deve ser invertido. A Fig. 31 apresenta um exemplo da aplicação da propriedade anterior.

<table style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 10%;">Cin</td><td style="width: 10%; text-align: center;">0</td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td></tr> <tr><td>A</td><td style="text-align: center;">0 0 0 1</td><td style="text-align: right;">(1)</td><td></td><td></td><td></td></tr> <tr><td>B</td><td style="text-align: center;">0 1 1 0</td><td style="text-align: right;">(6)</td><td></td><td></td><td></td></tr> <tr><td>Output</td><td style="text-align: center;"><u>0 1 1 1</u></td><td style="text-align: right;">(7)</td><td></td><td></td><td></td></tr> </table>	Cin	0					A	0 0 0 1	(1)				B	0 1 1 0	(6)				Output	<u>0 1 1 1</u>	(7)				<table style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 10%;">Cin</td><td style="width: 10%; text-align: center;">1</td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td><td style="width: 10%;"></td></tr> <tr><td>A</td><td style="text-align: center;">0 0 0 1</td><td style="text-align: right;">(1)</td><td></td><td></td><td></td></tr> <tr><td>B</td><td style="text-align: center;">0 1 1 0</td><td style="text-align: right;">(6)</td><td></td><td></td><td></td></tr> <tr><td>Output</td><td style="text-align: center;"><u>1 0 0 0</u></td><td style="text-align: right;">(8)</td><td></td><td></td><td></td></tr> </table>	Cin	1					A	0 0 0 1	(1)				B	0 1 1 0	(6)				Output	<u>1 0 0 0</u>	(8)			
Cin	0																																																
A	0 0 0 1	(1)																																															
B	0 1 1 0	(6)																																															
Output	<u>0 1 1 1</u>	(7)																																															
Cin	1																																																
A	0 0 0 1	(1)																																															
B	0 1 1 0	(6)																																															
Output	<u>1 0 0 0</u>	(8)																																															

Figura 31 – Exemplo de uma soma utilizando a propriedade 2 da soma.

Com base nas duas propriedades anteriores é possível montar-se uma tabela com todos os possíveis resultados gerados pela adição de dois números de m bits. Neste trabalho o circuito RB é utilizado em substituição a um bloco somador RCA de 4 bits. A tab.1 apresenta todos os resultados possíveis levando em conta a adição entre números de 4 bits. As primeiras quatro colunas referem-se aos possíveis resultados gerados a partir da soma entre dois números com carry de entrada igual a 0. As quatro colunas na seqüência se referem aos resultados desejados quando da adição dos mesmos dois números mas neste caso, com carry de entrada de igual a 1. E por fim, as

próximas colunas apresentam as funções que, com base na propriedades, indicam quais bits do resultado devem ser invertidos para que o resultado da adição com *carry* de entrada igual a 0 possa ser transformado em um resultado com *carry* de entrada igual a 1.

Tabela 1 – Conversão do resultado de uma adição com *carry* de entrada igual a 0 para o resultado com *carry* de entrada igual a 1.

S3S2S1S0	S3S2S1S0	f1	f2	f3	f4
0 0 0 0	0 0 0 1	0	0	0	1
0 0 0 1	0 0 1 0	0	0	1	1
0 0 1 0	0 0 1 1	0	0	0	1
0 0 1 1	0 1 0 0	0	1	1	1
0 1 0 0	0 1 0 1	0	0	0	1
0 1 0 1	0 1 1 0	0	0	1	1
0 1 1 0	0 1 1 1	0	0	0	1
0 1 1 1	1 0 0 0	1	1	1	1
1 0 0 0	1 0 0 1	0	0	0	1
1 0 0 1	1 0 1 0	0	0	1	1
1 0 1 0	1 0 1 1	1	0	0	1
1 0 1 1	1 1 0 0	0	1	1	1
1 1 0 0	1 1 0 1	0	0	0	1
1 1 0 1	1 1 1 0	0	0	1	1
1 1 1 0	1 1 1 1	0	0	0	1
1 1 1 1	0 0 0 0	1	1	1	1

De posse dos resultados da tab.1 é possível derivar um circuito mínimo para o bloco RB. Como forma de tornar ainda mais claro o processo de transformação da tab.1 no circuito em questão, a Fig. 32 apresenta todo o processo utilizado para a síntese manual do bloco RB. Observe que, além do circuito obtido através dos mapas de *Karnaugh*, é necessário, ainda, a utilização de 3 portas XOR ligadas à saída do circuito. Estas funcionam como inversores controlados, ou seja, caso haja a necessidade, alguns bits do resultado do RCA de 4 bits deverão ser invertidos. Caso contrário, as portas simplesmente deixam passar os valores para a saída. Em se tratando do bit menos significativo do resultado, o bit S0, como pode ser observado na tab.1, este sempre necessita ser invertido de acordo com propriedades algébricas.

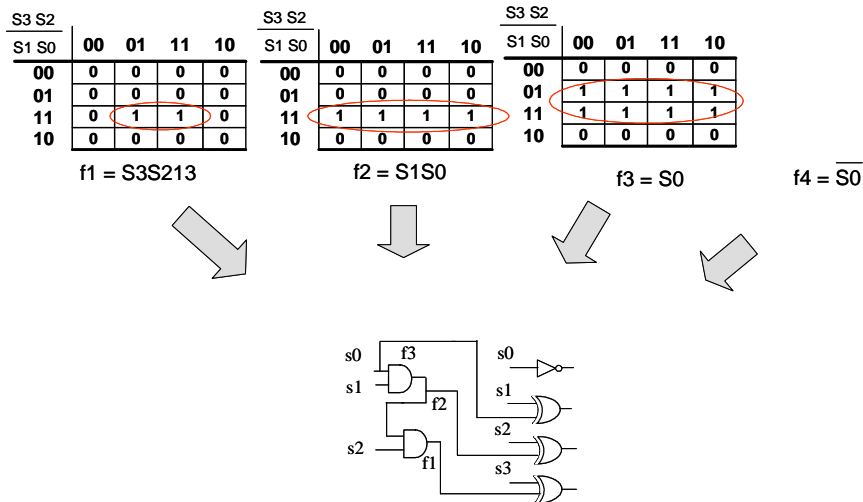


Figura 32 – Processo de extração do bloco RB do RIC.

Como pode ser observado na Fig. 32, o tamanho do bloco RB, quando comparado com um somador RCA de 4 bits, é significativamente menor. Enquanto um bloco RB possui 6 portas lógicas um RCA possui 24.

Considerando que cada seção de um somador RIC produz dois resultados, um referente à adição realizada pelo RCA e o outro referente à saída do RB, é necessário que se realize uma escolha entre estes dois valores a fim de que um deles seja enviado para a saída da seção. A forma como isto é realizado se constitui em uma das características únicas do RIC. Ao contrário do somador CSA, o RIC não possui dois *carries* de saída em cada seção, pois este utiliza apenas um somador por seção. Assim, as equações lógicas utilizadas para escolha dos resultados em cada seção necessitam levar em conta outro aspecto: o AND lógico entre os sinais *propagate* do RCA. Os sinais *propagate* correspondem à aplicação de uma porta XOR a cada par A_iB_i de operandos. Ressalta-se que, tratando-se de um somador RCA, nenhum hardware excedente é necessário para obtenção destes sinais, já que a própria construção de somadores RCA fornece tal estrutura, conforme pode ser observado na Fig. 23. Assim, é necessária apenas a utilização de uma porta AND de 4 entradas para obtenção do sinal desejado.

As equações utilizadas pelos multiplexadores em cada seção de um somador RIC são:

$$Sel0 = C_{in} \quad (1)$$

$$Sel1 = C_0 \text{ or } (P_0 \text{ and } C_{in}) \quad (2)$$

$$Sel2 = C_1 \text{ or } (P_1 \cdot (C_0 \text{ or } (P_0 \text{ and } C_{in}))) \quad (3)$$

$$Sel3 = C_2 \text{ or } (P_2 \text{ and } (C_1 \text{ or } (P_1 \text{ and } (C_0 \text{ or } (P_0 \text{ and } C_{in})))) \quad (4)$$

$$Cout = C_3 \text{ or } (p_3 \text{ and } (= C_2 \text{ or } (P_2 \text{ and } (C_1 \text{ or } (P_1 \text{ and } (C_0 \text{ or } (P_0 \text{ and } C_{in})))))) \quad (5)$$

Em relação às equações acima, o sinal C_0 corresponde ao *carry* de saída de um RCA de 4 bits que realiza a adição ($A_{0..3} + B_{0..3}$) com *carry* de entrada igual a 0. Já o sinal P_0 corresponde ao AND lógico aplicado aos sinais de *propagate* deste mesmo RCA ($(A_0 \text{ xor } B_0) \text{ AND } (A_1 \text{ xor } B_1) \text{ AND } (A_2 \text{ xor } B_2) \text{ AND } (A_3 \text{ xor } B_3)$). Os outros sinais das equações podem ser deduzidos seguindo este mesmo raciocínio.

Para que se chegasse à conclusão de que o AND lógico entre os sinais *propagate* dos RCAs de cada seção seria necessário para a formação das equações, foi necessária a identificação de uma outra propriedade da adição.

Propriedade 3: *Dois somadores idênticos recebendo os mesmos operandos de entrada, porém com carries de entrada distintos, exibem o mesmo carry de saída, exceto quando o AND lógico entre todos os sinais propagate dos somadores resulta em um valor igual a 1.*

Os sinais C_0 , C_1 , C_2 e C_3 (carries de saída dos RCAs de cada seção tendo como carries de entrada o valor 0) no início de cada equação acima, assim como nos somadores CSA, quando são iguais a 1, definem qual dos resultados calculados em cada seção deverá ser enviado para a saída da seção, independente das outras variáveis das equações. Dada a ausência de um RCA com *carry* de entrada igual a 1 em cada seção do RIC, se algum dos sinais C_0 , C_1 , C_2 , C_3 for igual a 0, se faz necessário que se saiba qual seria o *carry* de saída dos RCAs em questão caso estes tivessem como *carry* de entrada o valor 1. Tal resultado pode ser inferido utilizando-se a propriedade 3. Para isso, utiliza-se o resultado do AND lógico entre todos os *propagates* gerados pelos somadores RCA de cada seção. Caso o resultado desta porta AND seja 1, isto significa que se houvesse um outro RCA com *carry* de entrada igual a 1, o *carry* de saída deste somador seria 1 também.

Assim, as equações no somador RIC podem ser geradas da mesma forma como foram geradas as equações para os somadores CSA, necessitando apenas a troca dos sinais C_{ji} pelos sinais P_i .

Outra característica em comum deste somador com o *Carry Select* se dá quanto ao atraso do circuito. Somadores com 4 e 8 bits apresentam como atraso crítico o atraso de uma seção. Neste caso, este atraso refere-se à soma dos atrasos de um somador RCA, do circuito RB e do multiplexador, conforme ilustra a Fig. 33.

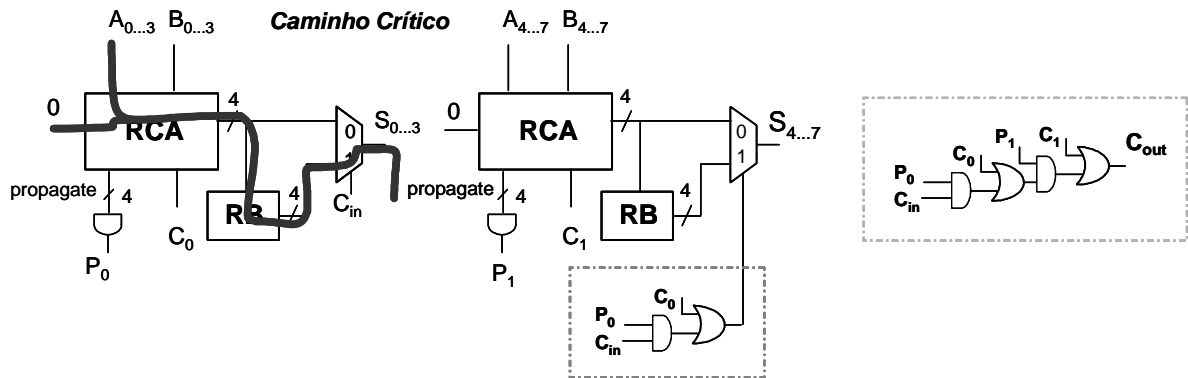


Figura 33 – Caminho crítico de um somador RIC de 8 bits.

Nota-se na Fig. 33 que, com relação aos CSAs com 4 e 8 bits, a diferença entre os atrasos dos somadores RIC e CSA é tão somente o atraso do RB. Portanto, é possível inferir que os somadores RIC de 4 e 8 bits apresentam um atraso superior aos somadores CSA de mesmo comprimento de dado.

Entretanto, em se tratando de mais bits, da mesma forma que os CSAs, o caminho crítico do RIC passa a ser influenciado pela equação lógica de cada seção. Nestes casos, o atraso dos somadores passa a ser a soma dos atrasos de um bloco RCA, da maior equação lógica de uma seção e do multiplexador. A Fig. 34 apresenta um RIC de 32 bits destacando seu caminho crítico.

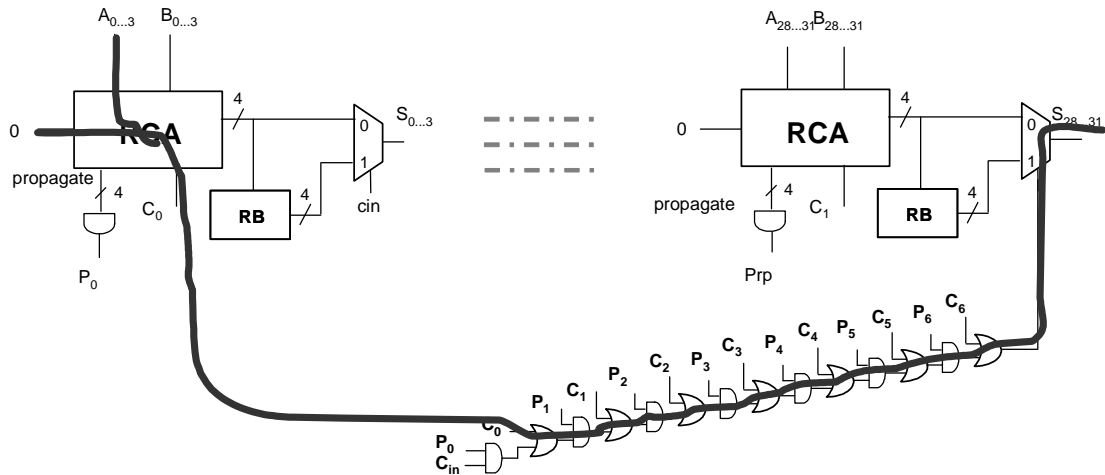


Figura 34 – Caminho crítico de um somador R1C de 32 bits.

Realizando-se um comparativo entre os somadores CSA e R1C de 32 bits, (Figs. 28 e 34 respectivamente), percebe-se que, com relação aos somadores R1C, o caminho crítico, a partir do momento que passar a contabilizar o atraso de uma equação lógica, possui uma porta a menos quando comparado ao caminho crítico do CSA. Deste modo, pode-se inferir que os somadores R1C, a partir de 16 bits, passam a ter um desempenho superior quando comparados aos somadores CSA com a mesma configuração.

4.4 Resultados comparativos entre as arquiteturas de somadores

Considerando-se que o somador R1C proposto neste trabalho é uma arquitetura de somador inédita, e portanto sendo assim uma contribuição importante, é imprescindível sua avaliação comparativamente com outras arquiteturas de somadores consolidadas. Desta maneira, foram descritas em VHDL – *VHSIC Hardware Description Language* (IEEE, 1994) todas as arquiteturas de somadores apresentadas nas seções anteriores. Para cada tipo de somador foram descritas versões com 4, 8, 16, 32, 64 e 128 bits. As arquiteturas foram sintetizadas para o dispositivo EP2S15F484C3 da família Stratix II da Altera e validadas através de simulação funcional com atrasos, utilizando a ferramenta Quartus II da Altera (2007). Foram analisados os impactos,

quando da implementação destas arquiteturas, com relação ao desempenho (atraso crítico topológico) e ao uso de recursos do FPGA (número de ALUTs).

O gráfico da Fig. 35 apresenta os resultados com relação ao atraso crítico de todos os somadores implementados (RCA, RIC e CSA).

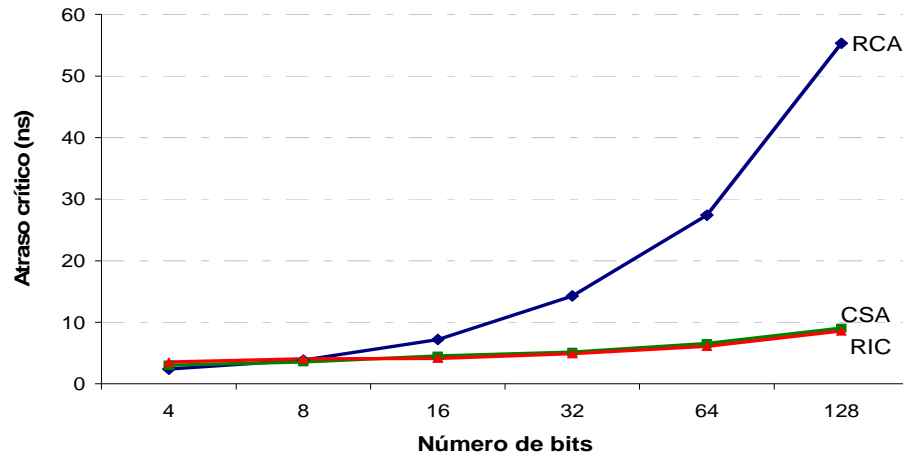


Figura 35 – Comparação entre os atrasos críticos das arquiteturas de somadores.

Como já se supunha, os somadores RCA apresentaram o pior desempenho. Já os somadores RIC e CSA apresentaram um desempenho muito semelhante. A fim de que estes últimos possam ser analisados mais claramente, a Fig. 36 apresenta o gráfico referente aos atrasos críticos apenas dos somadores RIC e CSA.

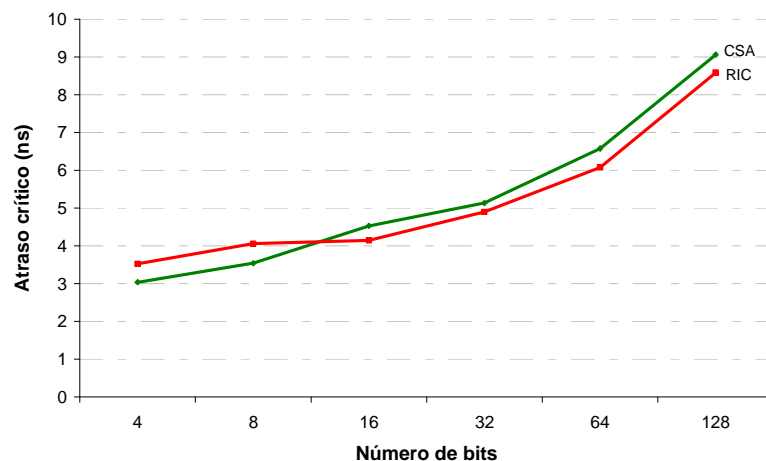


Figura 36 – Atrasos críticos dos somadores RIC e CSA.

Assinala-se que na Fig. 36 a partir de 16 bits o desempenho dos somadores RIC ficou melhor que o desempenho dos somadores CSA. Tal comportamento já foi previsto na seção 4.4.

Com o objetivo de melhor analisar os dados com relação ao atraso crítico dos somadores, a tab.2 apresenta todos os valores obtidos através da síntese dos somadores. A tabela demonstra ainda, nas últimas duas colunas, um comparativo percentual entre os atrasos dos somadores, tomando como referência o somador RCA.

Tabela 2 – Atrasos críticos dos somadores RCA, RIC e RCA.

Bits	Atraso crítico (ns)				
	RCA (1)	RIC (2)	$[(2)/(1)-1] \times 100$	CSA (3)	$[(3)/(1)-1] \times 100$
4	2.38	3.52	+48%	3.04	+28%
8	3.86	4.05	+5%	3.54	-8%
16	7.21	4.14	-43%	4.53	-37%
32	14.28	4.8	-66%	5.13	-64%
64	27.39	6.07	-78%	6.57	-76%
128	55.33	8.58	-84%	9.06	-84%

Através da tab.2 percebe-se que o somador RIC, quando comparado ao somador RCA, a partir de 16 bits apresenta atrasos críticos cada vez menores. Tal comportamento já era esperado, haja vista o paralelismo provido pelos somadores RIC ser maior à medida que crescem os somadores. Para os somadores com 128, o atraso crítico do RIC foi 84% menor que o do RCA. Esta diferença entre os atrasos dos RICs e RCAs tende a aumentar conforme aumenta o número de bits dos somadores.

Comportamento semelhante pode ser observado quando são comparados os somadores CSA e RCA. Neste caso, os CSAs apresentam menores atrasos a partir de 8 bits, quando comparados aos RCAs.

Percebe-se também que a partir de 16 bits os somadores RIC apresentam atrasos críticos menores que os atrasos críticos dos CSAs. Este decréscimo nos atrasos variou entre 5% (para o RIC com 128 bits) e 9% (para o RIC com 16 bits).

Com relação à quantidade de recursos exigidos pelas arquiteturas, a Fig. 37 apresenta um gráfico referente a todos somadores implementados (RCA, RIC e CSA).

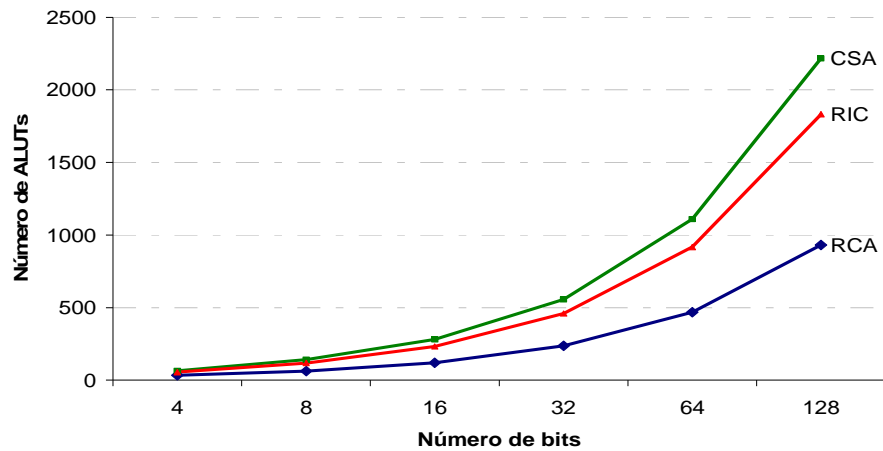


Figura 37 - Número de ALUTs utilizadas pelos somadores RCA, RIC e CSA.

Como já era esperado, pela análise da Fig. 37 percebe-se que o RCA é a arquitetura que utiliza menos recursos. Já o somador RIC se apresentou como uma arquitetura intermediária, no que diz respeito à utilização de recursos. O RIC, assim como fora explicado na seção 1.3, quando comparado ao CSA, utiliza um bloco somador a menos em cada seção, substituindo-o por um bloco RB, que possui uma complexidade inferior.

A fim de que possam ser analisados mais detalhadamente os resultados, a tab.3 apresenta todos resultados em relação a uso de recursos de todos somadores implementados. Nas 2 últimas colunas da tabela é apresentado um comparativo percentual, tomando o somador RCA como referência.

Tabela 3 - Comparação entre as arquiteturas de somadores, quanto à utilização de ALUTs.

Bits	ALUTs				
	RCA (1)	RIC (2)	$[(2)/(1)-1] \times 100$	CSA (3)	$[(3)/(1)-1] \times 100$
4	33	55	+67%	65	+97%
8	62	118	+90%	142	+129%
16	120	232	+93%	280	+133%
32	236	460	+95%	556	+136%
64	468	918	+96%	1110	+137%
128	932	1832	+97%	2216	+138%

Percebe-se, através da tab.3, que em todos os casos os somadores RIC e CSA utilizaram mais recursos que o somador RCA. Em relação ao RIC de 128 bits, quando comparado ao RCA, também de 128 bits, esse acréscimo de recursos foi de 97%. Já em relação ao CSA de 128 bits, o acréscimo de recursos foi de 138% quando comparado ao RCA com a mesma configuração.

Comparando-se os somadores RIC e CSA, percebe-se que em todos os casos o RIC utiliza uma quantidade de recursos menor que o CSA. Em média, o RIC utiliza 17% menos recursos que os somadores CSA.

Com base nos resultados é possível afirmar que o RIC é uma boa alternativa de somador rápido quando há restrições severas de uso de recursos. Além disto, como pôde ser observado, o RIC ainda apresenta ótimos resultados, em termos de desempenho.

5 Projeto de Sistemas Tolerantes a falhas

A preocupação com a construção de sistemas computacionais tolerantes a falhas não é um tema recente. Com o avanço da tecnologia CMOS, sistemas computacionais fabricados no silício passaram a controlar diversas aplicações, dentre elas as chamadas aplicações críticas, onde uma falha pode resultar em perda de vidas humanas.

Dois conceitos fundamentais necessitam ser levados em conta na construção de um sistema tolerante a falhas: a Intolerância a falhas e a Tolerância a falhas (LALA, 2001).

A Intolerância a falhas está aliada ao conceito de prevenção, tendo por objetivo aumentar a confiabilidade do projeto, eliminando, a priori, todas suas falhas. Como esta é uma tarefa impossível, o objetivo desta passa a ser então diminuir a probabilidade de um sistema falhar para valores aceitáveis. Por outro lado, a tolerância a falhas objetiva o mascaramento da falha para que o sistema continue operando. Para que esta tolerância a falhas seja alcançada, os sistemas devem fazer uso de algum acréscimo de recursos, dentre eles excedente de hardware (Redundância de hardware), excedente de informação (redundância de informação), excedente de tempo (redundância temporal), ou ainda uma combinação entre estes.

Em se tratando de tolerância a falhas é importante que se conheça três conceitos fundamentais: falha (*fault*), erro (*error*) e mau funcionamento (*failure*). Existe uma relação de causa e efeito entre estes termos; especificamente, falhas são as causas de erros, que são as causas do mau funcionamento. Entende-se como falha qualquer imperfeição ou desvio do funcionamento esperado de um componente, seja de hardware ou de software. Por sua vez, um erro é a manifestação de uma falha,

normalmente observado como um desvio na precisão ou correção de um resultado. Finalmente, se um erro fizer com que o sistema se comporte de maneira divergente de sua especificação, haverá a deflagração de um mau funcionamento do sistema (PRADHAN, 1996).

Esta relação de causa e efeito implicada pelos três conceitos acima apresentados remete à definição de dois importantes parâmetros: latência de falha e latência de erro. Latência de falha é o período de tempo entre a ocorrência de uma falha e sua manifestação como um erro. Latência do erro remete-se ao tempo entre a ocorrência de um erro e sua manifestação no sistema em forma de um mau funcionamento (PRADHAN, 1996).

Falhas podem ser resultados de uma variedade de fenômenos que ocorrem dentro dos componentes eletrônicos, resultados de efeitos externos a estes componentes ou até mesmo resultado da fase de projeto. Em relação a este último, duas são as causas principais: erros de especificação, tais como algoritmos ou hardware incorretos e erros de implementação, como uma má escolha de componentes ou erros na codificação do software. Outra causa de falhas são os componentes eletrônicos defeituosos. O motivo de defeitos em componentes varia desde problemas no processo de fabricação até mesmo ao desgaste destes devido ao tempo. Por fim, efeitos causados por eventos externos como a radiação, também podem resultar em falhas dos componentes.

Para descrever uma falha adequadamente, outras características, além de sua origem, são relevantes. Dentre elas podemos citar 4 atributos: a natureza, duração, extensão e valor. A natureza especifica o tipo de falha: de software ou hardware, por exemplo. A duração especifica o tempo que uma falha permanece ativa, sendo que neste sentido as falhas podem ser classificadas como permanentes, transientes ou intermitentes. Falhas permanentes são as que se mantêm indefinidamente até que algum método de correção seja aplicado. Falhas transientes se caracterizam por aparecerem e desaparecerem dentro um pequeno período de tempo. Já as falhas intermitentes possuem a característica de se manifestarem mais de uma vez. Em relação à extensão de uma falha, esta indica se os seus efeitos estão localizados em um determinado componente do sistema ou se afetam outros módulos de hardware e

software. Outro atributo de uma falha é o seu valor, podendo este ser determinado (o estado da falha mantém-se inalterado durante toda a sua ocorrência) ou indeterminado (o estado da falha varia em relação ao tempo) (PRADHAN, 1996).

5.1 Tolerância a falhas aplicada a Circuitos Integrados

A utilização de técnicas de tolerância a falhas no projeto de sistemas integrados pode ser visto como uma evolução natural das técnicas de projeto. À medida que os circuitos integrados se tornaram mais complexos, passou a ser necessário integrar no próprio circuito estruturas extras para viabilizar o teste, sendo esta última também chamada de auto-teste integrado ou *built-in self test* - BIST (BUSHNELL; AGRAWAL, 2000). De forma concorrente, novas estruturas integradas surgiram, propiciando aos circuitos detectarem e, por vezes, realizarem a correção na presença de um erro (*on-line testing* e *on-line correction*). Assim, a construção de sistemas tolerantes surgiu como uma evolução das técnicas de projeto empregadas por sistemas autotestáveis e autocorrigíveis.

Conforme já explicado no capítulo 2, o avanço da tecnologia CMOS estado-da-arte, bem como a constante diminuição das dimensões dos transistores, têm tornado o comportamento dos circuitos integrados cada vez mais imprevisível. Desta forma, a implementação de técnicas de tolerância a falhas nos projetos de sistemas digitais têm sido cada vez mais necessárias.

Possivelmente a primeira técnica empregada quando da construção de sistemas tolerantes a falhas transientes geradas por partículas carregadas tenha sido a blindagem (*shielding*). Porém, além desta não proteger completamente os circuitos quanto a efeitos de partículas carregadas, a blindagem aumentava demasiadamente o custo do projeto, sobretudo considerando-se as aplicações espaciais, militares e aeronáuticas, justamente onde a tolerância à radiação é uma condição *sine qua non*.

Atualmente, as técnicas empregadas na construção de circuitos integrados em silício podem ser divididas de duas formas: processo de fabricação específico ou aplicação de técnicas de proteção.

5.2 Processo de Fabricação Específico

Processos de fabricação diferentes do CMOS padrão, tais como CMOS epitaxial e silício sobre isolante (*silicon-on-insulator* - SOI) (DODD; MASSENGILL, 2003; BAUMANN, 2005), podem reduzir a níveis aceitáveis alguns dos efeitos da radiação. Entretanto, os processos de fabricação não-padrão disponíveis atualmente não são capazes de eliminar completamente os SEUs e SETs, principais preocupações dos projetos a serem fabricados em tecnologia VDSM. Além disto, processos de fabricação diferentes do CMOS padrão apresentam elevado custo, eventualmente podendo se justificar apenas para elevados volumes de produção. Resultados publicados recentemente por Irom et al. (2002) revelaram pequena redução da sensibilidade a SEUs em um microprocessador comercial implementado em tecnologia SOI. Por conseqüência, ainda que se faça uso de alguma tecnologia de fabricação não-padrão, técnicas de projeto dedicadas provavelmente serão necessárias.

5.3 Técnicas de proteção

Ao longo dos anos várias técnicas de proteção têm sido propostas com intuito de aumentar a confiabilidade e disponibilidade de sistemas computacionais. Toda e qualquer técnica de proteção está fortemente ligada ao conceito de redundância, seja ela de hardware, tempo ou informação.

Diante disto, tratando-se de projetos de circuitos integrados, é de fundamental importância a escolha de uma técnica apropriada quando da implementação de um novo projeto. Isto se deve principalmente aos impactos proporcionados, dada a aplicação de uma determinada técnica. Dentre eles podemos citar os impactos referentes à área de ocupação, consumo de potência, peso entre outros.

A maioria das técnicas de proteção já proposta baseia-se exclusivamente em redundância de hardware, redundância temporal, redundância de informação e, por fim, uma mescla de ambas. As subseções posteriores tratam de apresentar algumas das diversas técnicas utilizadas na construção de circuitos integrados protegidos contra

falhas transientes, bem como classificar as técnicas aplicadas de acordo com o tipo de proteção oferecida por cada técnica.

5.3.1 Técnicas de proteção baseadas em redundância de hardware

A redundância de hardware é provavelmente a técnica mais comum de proteção usada em sistemas (PRADHAN, 1996; LALA, 2001). Existem basicamente 3 formas básicas de redundância de hardware: redundância estática, que alcança a tolerância a falhas sem que para isto necessite detectar nenhuma falha; redundância dinâmica, que utiliza o conceito de detecção e posterior recuperação de falhas; e redundância híbrida, que utiliza características das duas.

5.3.1.1 Redundância estática

Técnicas de proteção estáticas, ou passivas, estão relacionadas ao conceito de mascaramento de falhas. Dada a ocorrência de um erro no sistema, essas técnicas têm como objetivo fazer com que este não venha a traduzir-se em um mau funcionamento do circuito. Vale observar que técnicas passivas não têm como objetivos detectar, nem tampouco recuperar falhas que ocorram no sistema.

A mais comum técnica de mascaramento de falhas é a chamada TMR – *Triple Module Redundancy*. A técnica foi proposta por Von Neumann (1956) e está representada na Fig. 38.

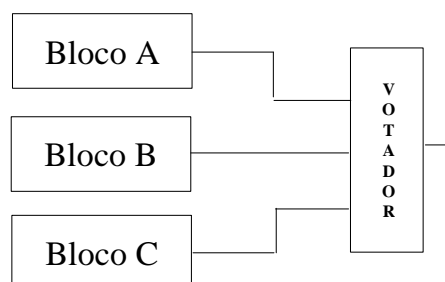


Figura 38 – Técnica de proteção TMR.

TMR consiste em utilizar três exemplares do hardware a ser protegido e utilizar um bloco de hardware extra, denominado votador (PRAHDAN, 1996; CARMICHAEL,

2001; LALA, 2001). Cada um dos blocos triplicados possui sua saída ligada a uma das entradas do votador. O objetivo da técnica é, diante de uma falha em um dos blocos triplicados, fazer com que o votador tenha em suas entradas sempre no mínimo dois valores corretos. Uma das vantagens da utilização de TMR é com relação ao desempenho dos circuitos protegidos. Note na Fig. 38 que cada um dos blocos triplicados executa suas funções em paralelo com os outros dois blocos. Assim, a queda de desempenho se dá apenas devido à inserção do circuito votador na saída dos blocos.

A utilização de TMR como técnica de proteção também implica em algumas desvantagens. A mais clara desvantagem reside no acréscimo de recursos: mais de 200%. Outra desvantagem é referente ao circuito votador, que se constitui no único ponto vulnerável a falhas, ou seja, dada uma falha neste circuito a técnica não garante mais a correção dos resultados. E por fim, outro problema que não pode ser ignorado diz respeito à possível ocorrência de uma falha em mais de um bloco por vez. Embora a probabilidade da ocorrência de tal situação seja pequena para as tecnologias atuais (LIMA, 2003), tal problema tende a se tornar importante para as tecnologias CMOS vindouras.

A fim de evitar a última desvantagem citada no parágrafo anterior, o conceito de TMR pode ser expandido para incluir qualquer número de blocos redundantes, dando origem, assim, aos chamados NMR – *N Modular Redundancy*.

5.3.1.2 Redundância Dinâmica

Outra categoria de técnica de proteção são as chamadas técnicas ativas, também conhecidas como técnicas dinâmicas. Estas, ao contrário das técnicas passivas, tem por objetivos a detecção de uma falha no sistema e, se necessário, a substituição de partes que não estejam funcionando corretamente. Em outras palavras, o sistema que faz uso de técnicas deste gênero deve estar apto a ser reconfigurado.

Um sistema que faz uso de redundância dinâmica utiliza vários módulos idênticos, porém apenas um operando a cada vez. Se uma falha é detectada no módulo que está operando, este é substituído por um módulo reserva. Assim, redundância

dinâmica requer consecutivas ações de detecção e recuperação de falhas. A Fig. 39 ilustra o conceito de redundância dinâmica.

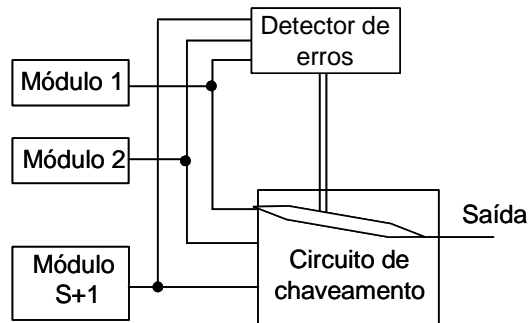


Figura 39 – Redundância dinâmica utilizando S módulos reservas.

Técnicas de proteção baseadas em redundância dinâmica normalmente implicam em uma interrupção do funcionamento do sistema. Assim, estas técnicas normalmente são empregadas em aplicações nas quais resultados errôneos são aceitáveis durante um breve período (PRADHAN, 96).

Um exemplo de mecanismo de detecção de falhas utilizado por técnicas baseadas em redundância dinâmica é a duplicação com comparação – DWC (*Duplication with comparison*). Esta técnica consiste em utilizar dois blocos de hardware idênticos, realizando a mesma computação em paralelo. Ao final da computação de ambos os blocos os resultados são comparados. Caso o detector perceba alguma discordância entre os resultados, uma mensagem de erro é gerada. Em sua forma mais básica, a DWC não pode tolerar nenhum tipo de falha, podendo apenas detectá-la.

Uma segunda técnica empregada na redundância dinâmica é a reposição dinâmica – *standby replace*. Neste método, um módulo permanece operacional enquanto uma ou mais réplicas encontram-se inativas (podendo inclusive ficar sem alimentação, técnica conhecida como *cold standby*). Vários esquemas de detecção são utilizados para determinar quando um módulo apresenta alguma falha permanente. Neste caso, após a localização da falha e a identificação do módulo responsável, a unidade defeituosa é retirada do sistema e este é reconfigurado.

A técnica de *cold standby* requer que, momentaneamente, o desempenho do circuito diminua até que a reconfiguração seja finalizada. Essa perda no desempenho pode ser minimizada utilizando o chamado *hot standby sparing* (módulo reserva ativo).

Neste caso um módulo reserva é sincronizado com os módulos em atividade no sistema. Em caso de falha em algum dos módulos, o módulo reserva está sempre pronto para substituí-lo (PRADHAN, 1996; LALA, 2001).

5.3.1.3 Redundância Híbrida

Técnicas de tolerância a falhas baseadas em redundância híbrida combinam características de técnicas estáticas e dinâmicas. Geralmente o mascaramento de falhas é usado como forma de prevenir o sistema de produzir resultados errôneos, enquanto que, a detecção, localização e recuperação de falhas são utilizadas a fim de reconfigurar o sistema na presença de uma falha.

Existem várias aplicações de redundância híbrida. A maioria destas sustentam-se no conceito de NMR com blocos reservas (*sparers*). A Fig. 40 apresenta um TMR com um bloco reserva.

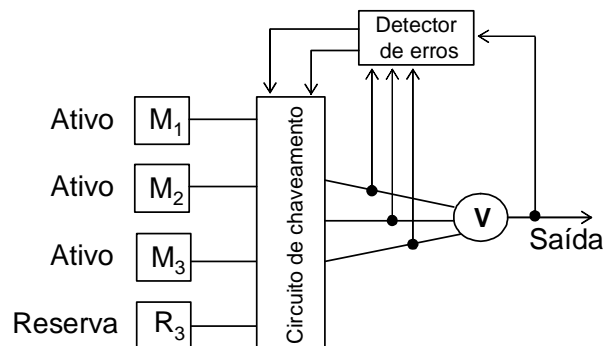


Figura 40– Redundância híbrida com um bloco reserva.

Quando um módulo do TMR falha, este é substituído por um módulo reserva e o TMR continua a funcionar normalmente. O detector de erros, através dos valores enviados para o votador de saída, verifica se existe alguma inconsistência em algum dos resultados gerados pelos blocos redundantes. Em caso positivo, o detector envia um sinal para o circuito de chaveamento que por sua vez substitui o bloco falho por um bloco reserva, com um estado perfeito. A grande desvantagem de técnicas desta natureza é a quantidade de hardware extra despendida. Note na Fig. 40 que, para a implementação de um TMR é necessário no mínimo 3 blocos redundantes extras, um

detector de erros, um circuito de chaveamento e mais um circuito votador (PRADHAN, 1996; LALA, 2001).

5.3.2 Redundância Temporal

A principal desvantagem da utilização de técnicas de proteção baseadas em redundância de hardware é o acréscimo de recursos exigido. Uma alternativa para amenizar tal desvantagem é a utilização de técnicas que façam uso de redundância temporal - TR (*Time Redundancy*). Ao invés de duplicar ou triplicar um bloco de hardware, por exemplo, técnicas baseadas em TR utilizam apenas um bloco e armazenam os resultados gerados por este bloco em tempos diferentes (PRADHAN, 1996; LALA, 2001; LIMA, 2003). A Fig. 41 mostra como seria a arquitetura de um bloco combinacional protegido com TR. A saída do bloco combinacional é ligada a três registradores, responsáveis por armazenar seu resultado em tempos diferentes. O primeiro registrador armazena o valor da saída do bloco em um tempo T , o segundo registrador e o terceiro armazenam o valor da saída em um tempo $T+d$ e $T+2d$, respectivamente. Assim como TMR, a técnica utiliza ainda um circuito votador que recebe como entradas as saídas dos registradores. Caso um dos registradores armazene algum valor incorreto, o votador pode enviar para a saída o valor correto, conforme os valores fornecidos pelos outros dois registradores.

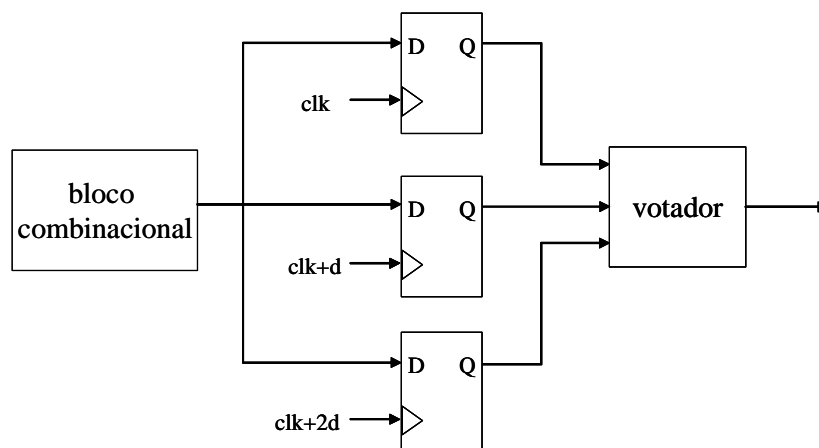


Figura 41 – Bloco combinacional protegido com Redundância Temporal.

A utilização de técnicas baseadas em RT, embora apresentem algumas vantagens quando comparadas à utilização de TMR, principalmente em relação a recursos extras exigidos, em se tratando de desempenho a RT apresenta algumas desvantagens. Dentre elas podemos citar a necessidade da utilização de um esquema de sincronização com três fases de relógio (ou a inserção proposital de um atraso conhecido no sinal de relógio), que certamente implica em uma maior complexidade do projeto, bem como uma redução no desempenho. Além disso, a técnica apresentada só é eficaz em virtude da ocorrência de falhas transientes. Na presença de uma falha permanente no bloco combinacional, o votador não terá condições de enviar um valor correto para saída do circuito, e os registradores sempre armazenarão valores incorretos.

5.3.3 Redundância de hardware combinada com Redundância temporal

Como pôde ser observado nas seções anteriores, a aplicação de uma técnica de proteção sempre implica em alguma desvantagem. Técnicas baseadas em redundância de hardware exigem uma quantidade de recursos extras significativa. Já as técnicas baseadas em redundância temporal implicam em quedas do desempenho. Assim, algumas técnicas têm sido propostas a fim de se reduzir as perdas de desempenho e ao mesmo tempo, controlando a quantidade de recursos extras.

A Duplicação com Comparação Combinada com Redundância Temporal (*DWC+TR - Duplication with comparison combined with time redundancy*) (LIMA, 2003) pertence à classe de técnicas mencionadas no parágrafo anterior. Esta técnica combina características de técnicas de detecção, tais como duplicação com comparação – DWC (PRADHAN, 1996), com técnicas baseadas em redundância temporal. O objetivo principal desta técnica é obter a mesma eficácia da técnica TMR, sem que para isso necessite triplicar o hardware a ser protegido. A Fig. 42 apresenta um bloco combinacional utilizando *DWC+TR*. Pela análise desta figura percebe-se que são utilizados apenas dois exemplares do bloco combinacional a ser protegido. A saída de cada exemplar é ligada diretamente às duas primeiras entradas do votador. Os registradores, ligados na saída de cada bloco, são responsáveis por armazenarem os

resultados gerados em cada exemplar, em tempos diferentes. Em um tempo t os registradores Reg0 e Reg1 armazenam os resultados fornecidos pelos exemplares 0 e 1 e, em um tempo $t+d$, então, os resultados dos mesmo exemplares são armazenados pelos registradores Reg0_d e Reg1_d. A técnica utiliza, ainda, um circuito comparador que recebe como entradas os valores fornecidos pelos registradores. Caso algum registrador capture algum valor incorreto, dada a ocorrência de uma falha em um dos blocos combinacionais, ou até mesmo no registrador, o comparador, através de um multiplexador, possibilita que o resultado calculado pelo bloco combinacional livre de falhas seja enviado para a terceira entrada do votador.

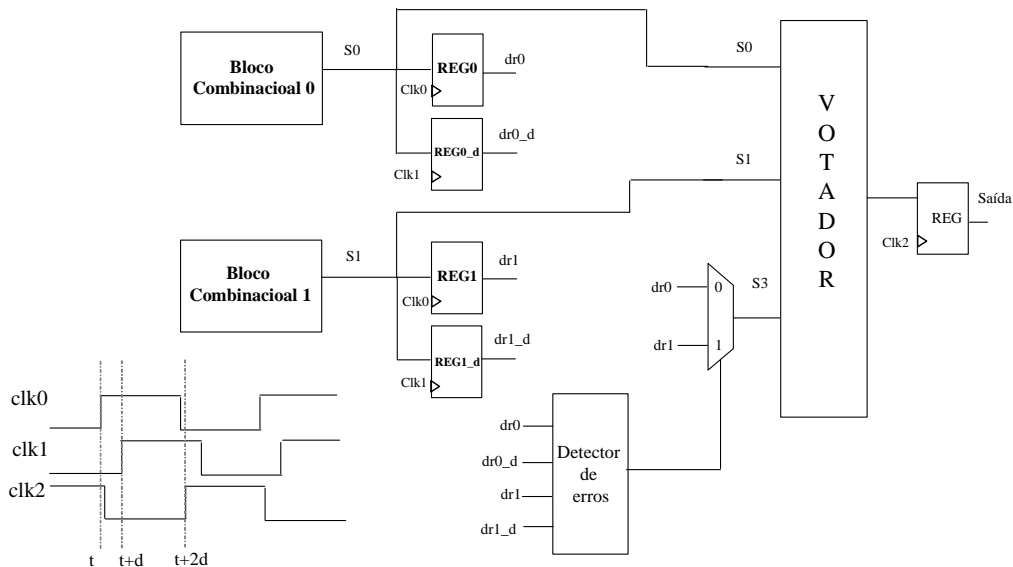


Figura 42 – Bloco combinacional protegido com DWC+TR.

A utilização de *DWC+TR*, de acordo com o exemplo apresentado na Fig. 42, embora utilize um bloco a menos que a técnica TMR, necessita que sejam utilizados alguns circuitos extras. São eles: os registradores, um multiplexador e um comparador. Logo, dependendo do tamanho do circuito a ser protegido, a utilização de *DWC+TR* pode não corresponder às expectativas em relação à utilização de recursos. Além do mais, a técnica faz uso de redundância temporal, necessitando que haja uma defasagem no relógio do circuito, o que certamente aumenta a complexidade do mesmo. Outra característica importante refere-se à defasagem entre os relógios. Esta

defasagem deve ser longa o suficiente para que, havendo uma falha em um dos blocos combinacionais, a falha não possa ser armazenada por dois registradores ao mesmo tempo, o que possibilitaria que um valor errado fosse enviado para saída do circuito (LIMA, 2003). Vale observar que esta técnica só é eficaz na presença de falhas transientes. Na presença de uma falha permanente em um dos blocos combinacionais, a técnica não pode detectá-la nem tampouco mascará-la.

5.3.4 Técnicas baseadas em Redundância de Informação

Técnicas baseadas em redundância de informação envolvem a adição de informação a dados, com o intuito de detectar, mascarar ou até mesmo corrigir possíveis inconsistências (PRADHAN, 1996; LALA, 2001).

Existem inúmeras técnicas baseadas em redundância de informação, mas em relação à projetos de circuitos integrados, podemos destacar, principalmente, os chamados códigos de detecção e correção de erros (EDAC - *error detection and correction codes*), utilizados principalmente na proteção de memórias (LIMA, 2003). Dentre eles podemos citar o código Hamming (HAMMING 1950, LIMA, 2003), e ainda uma combinação entre códigos *Hamming* e *Reed-Solomon*.

Várias razões fazem do código *Hamming* um dos mais utilizados na proteção de memórias, como o fato deste não necessitar de grande quantidade extra de hardware para sua implementação, podendo esta variar de 10 a 40% do circuito. Pode-se citar, ainda, como razão o fato dos processos de codificação e decodificação implicarem em pequenos aumentos no atraso do circuito. Finalmente, o código de *Hamming* é capaz de detectar e corrigir de 60% a 70% das falhas que venham a ocorrer em uma memória (PRADHAN, 1996).

A implementação do código *Hamming* consiste em particionar os dados a serem protegidos em pequenos grupos de bits e utilizar um bit de paridade para cada subgrupo. Uma característica importante de cada bit de paridade extra é o fato destes não ficarem atrelados a apenas um grupo de bits. Cada bit carrega informações de paridade de subgrupos de bits posicionados a sua direita (desconsiderando os bits de

paridade). Tal característica possibilita que o código consiga localizar o bit errôneo, dada alguma inconsistência nos dados.

Embora o Código *Hamming* tenha uma eficácia considerável, ele não é capaz de detectar, nem tampouco corrigir, dados que possuam mais de um bit errôneo. Dado o constante avanço da tecnologia CMOS e o aumento da taxa de MBUs em memórias SRAM, algumas técnicas capazes de detectar e corrigir até dois bits errôneos por vez têm sido propostas. Dentre elas podemos citar uma técnica que combina códigos *Reed-Solomon* e *Hamming*. O objetivo principal da utilização de uma combinação entre ambas é diminuir o acréscimo de recursos necessário em comparação a circuitos protegidos apenas com *Reed-Soloman* (NEUBERGER; LIMA; CARRO; REIS, 2003).

Por fugirem ao escopo deste trabalho, os diversos códigos de detecção e correção de erros existentes não serão detalhados. Maiores informações referentes aos códigos *Hamming* e *Reed-Soloman* e diversos outros códigos podem ser encontrados em Pradhan (1996) e em Lala (2001).

5.3.5 Técnicas de tolerância a falhas aplicadas à FPGAs

A preocupação com os efeitos da radiação nos FPGAs não é recente, haja vista a grande utilização destes dispositivos em missões espaciais, onde a atividade cósmica é intensa. O principal motivo do uso de FPGAs com estes fins se deve principalmente à facilidade de reconfiguração destes dispositivos.

Um FPGA ideal para aplicações espaciais necessita de um QML (*qualified manufacturing line*) da classe V, que assegura alta disponibilidade dos dispositivos operando no espaço. Em se tratando da maioria das missões espaciais, um TID – *Total Ionization Dose*⁴ de 300 krad⁵ é suficiente. Para FPGAs que operam em situações críticas, a tolerância a SEU é imprescindível. Assim, uma tensão de *threshold* de pelo menos 37 eVcm²/mg é exigida para elementos de memória (bits de configuração), registradores e flip-flops da lógica do usuário (ROOSTA, 2004).

⁴ Degradação dos circuitos eletrônicos devido ao acúmulo de energia em matérias pertencentes aos Circuitos Integrados.

⁵ Rad - *radiation absorbed dose*: unidade de medida dos efeitos causados pela radiação nos materiais.

Considerando-se os FPGAs baseados em SRAM, conforme apresentado no capítulo 3, várias células de memória (SRAM) são utilizadas com o intuito de armazenar a configuração dos FPGAs. A ocorrência de uma colisão de uma partícula carregada em um desses bits de configuração pode ocasionar a inversão do valor armazenado na célula de memória. Neste caso, é necessário que sua configuração seja carregada novamente.

A Fig. 43 a seguir ilustra os pontos vulneráveis a falhas de um FPGA baseado em SRAM da família *Virtex* da Xilinx (XILINX, 2000). Embora este não seja o dispositivo utilizado neste trabalho, sua arquitetura interna se assemelha muito aos dispositivos da família Stratix II (ALTERA, 2005b).

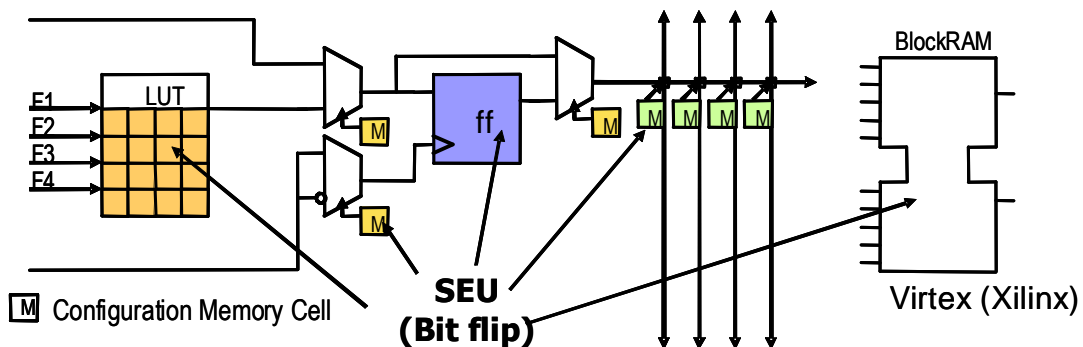


Figura 43 – Pontos vulneráveis de um FPGA da família Virtex da Xilinx.

Fonte: LIMA, 2003.

A fim de que possam ser reduzidas as taxas de suscetibilidade a falhas provocadas por radiação a valores tais quais os exigidos, empresas, tais como Xilinx e Actel, têm investido na construção de FPGAs protegidos para aplicações críticas. A proteção desses dispositivos se dá principalmente nas células de memória. A Actel desenvolveu recentemente uma família de FPGAs, RTSX-SU, baseada em tecnologia anti-fusível, 0.25 μm que provê proteção contra SEUs utilizando uma célula de memória especial. Tal célula, batizada de *SEU-Hardened D Flip-Flop*, utiliza a técnica de TMR em sua construção (ACTEL, 2006). A Fig. 44 apresenta em alto nível uma célula de memória *SEU-Hardened D Flip-Flop*.

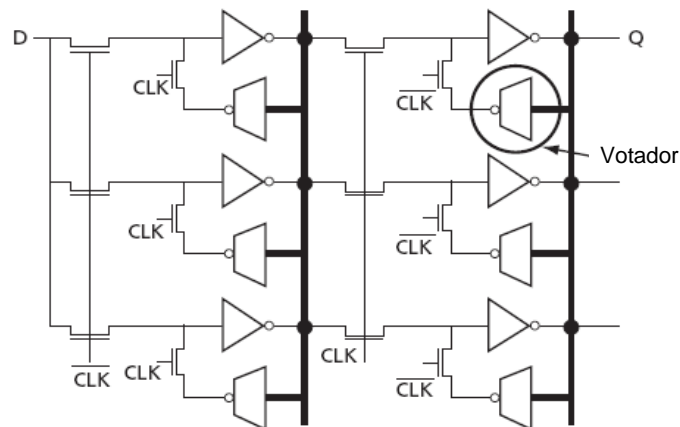


Figura 44 - *SEU-Hardened D Flip-Flop* do FPGA da Actel.

Fonte: ACTEL, 2006.

Através da Fig. 44 percebe-se que o *Flip-Flop* é constituído por três pares de circuitos mestre/escravo. Cada circuito mestre/escravo possui dois *latches*: um ativado pela borda ascendente e o outro pela borda descendente do relógio. Para cada par de circuitos mestre-escravo são utilizados dois circuitos votadores. Estes votadores recebem como entradas as saídas dos *latches* triplicados. Desta forma, na presença de uma falha em um dos *latches*, o votador, através das outras duas entradas, poderá sobrescrever o valor incorreto no *latch* falho. Note que esta técnica não será eficaz na presença de um MBU.

O alto custo exigido para implementação de FPGAs como os citados anteriormente, devido à necessidade de investimentos em desenvolvimento, teste e fabricação, faz com que o preço por peça de um FPGA protegido seja demasiadamente alto. Conforme já pôde ser observado no capítulo 2, devido ao avanço da tecnologia CMOS, cada vez mais os circuitos operando na superfície da terra estão se tornando vulneráveis a falhas provocadas pela radiação. Assim, soluções que dispensem a utilização de FPGAs protegidos, começam a ser estudadas a fim de proteger estes dispositivos operando ao nível do mar. Uma destas soluções é a aplicação de técnicas de mais alto nível às arquiteturas que serão implementadas dentro dos FPGAs.

A técnica mais utilizada, a fim de proteger os FPGAs da ação de SEUs, é uma combinação de TMR com *scrubbing* (CARMICHAEL; CAFFREY; SALAZAR, 2000; CARMICHAEL, 2001). A técnica de *scrubbing* (LIMA, 2003a) consiste em carregar o

frame de configuração do FPGA em um dado intervalo de tempo. Este intervalo varia conforme a taxa de SEUs esperada para determinada aplicação. Assim, TMR é responsável por proteger a lógica do usuário enquanto que os bits de configuração ficam protegidos pela técnica de *scrubbing*. Alguns FPGAs podem ser configurados parcialmente. Desta forma, se detectada uma falha em algum bit de configuração do FPGA, apenas o *frame* que possui um bit errôneo será carregado. Esta técnica é conhecida como reconfiguração parcial (CARMICHAEL; CAFFREY; SALAZAR, 2000).

O capítulo seguinte apresenta os resultados obtidos no projeto, síntese e validação dos somadores apresentados no capítulo 4, protegidos contra SETs.

6 Experimentos e Resultados Práticos

Tendo em vista a crescente preocupação com os efeitos causados pela colisão de partículas carregadas com as regiões sensíveis da lógica do usuário dentro dos FPGAs, os SETs (*Single Event Transients*), e dada a importância dos operadores aritméticos para os sistemas computacionais integrados, foram implementadas arquiteturas de somadores protegidas contra tais efeitos.

As arquiteturas implementadas correspondem a todas as arquiteturas de somadores já apresentadas no capítulo 4, ou seja, os somadores RCA (*Ripple Carry*), RIC (*Recomputing on Inverted Carry-in*) e os somadores CSA (*Carry Select*). Em se tratando das técnicas de proteção utilizadas, foram escolhidas três técnicas: TMR (*Triple Module Redundancy*), TR (*Time Redundancy*) e DWC+TR (*Duplication with Comparison Combined with Time Redundancy*).

Para cada técnica de proteção, foram descritos em linguagem VHDL somadores com 4, 8, 16, 32, 64 e 128 bits. As arquiteturas foram sintetizadas para o dispositivo EP2S15F484C3 da família Stratix II da Altera e posteriormente, validados através de simulação funcional com atrasos, utilizando a ferramenta Quartus II da Altera (2007). Nas seções seguintes serão apresentados os detalhes de implementação no que dizem respeito às arquiteturas de somadores protegidas implementadas.

A seção 6.1 apresenta os detalhes de implementação no que dizem respeito às arquiteturas de somadores protegidas que foram implementadas. A seção 6.2 apresenta uma diretiva de compilação utilizada neste trabalho a fim de inibir a ferramenta Quartus II de realizar otimizações no projeto. Por fim, seção 6.3 refere aos resultados obtidos quando da implementação das arquiteturas protegidas.

6.1 Arquiteturas de somadores Protegidas

6.1.1 Detalhes de implementação dos somadores protegidos com - TMR

Todas as arquiteturas de somadores protegidas com TMR (*Triple Module Redundancy*) foram implementadas a partir de blocos de 4 bits. Cada bloco destes é constituído por 3 exemplares do bloco a ser protegido e um circuito votador que recebe como entradas as saídas destes blocos. A Fig. 45 apresenta a arquitetura de um CSA de 8 bits protegido com TMR.

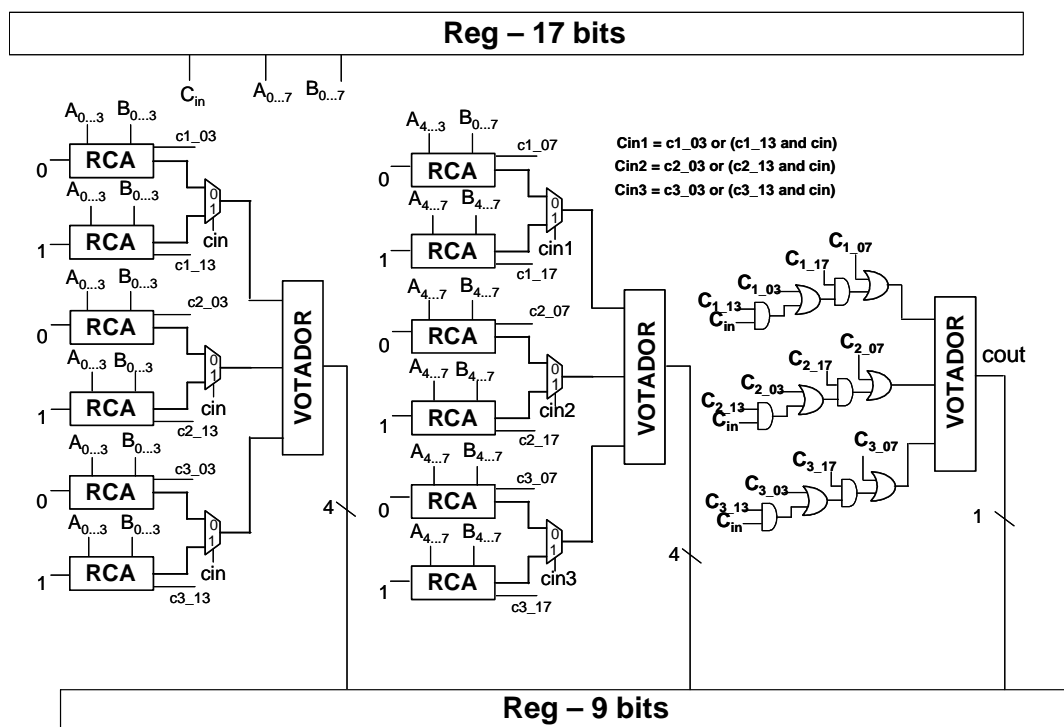


Figura 45 – CSA de 8 bits protegido com TMR.

Note na Fig. 45 que cada seção do CSA possui seu próprio votador. Observe-se ainda na Fig. 45 que sinais de *carry* gerados pelos CSAs triplicados em cada seção não passam por nenhum tipo de votação, sendo utilizados diretamente pelos CSA das seções seguintes. Com relação ao *carry* de saída do CSA, note na Fig. 45 que para a obtenção deste sinal é utilizado um circuito à parte. Como não poderia ser diferente, a

este foi aplicada a técnica TMR também, porém neste caso utilizando um circuito votador de 1 bit.

A arquitetura do circuito votador de 1 bit é bastante simples, conforme mostra a Fig. 46 (a). O valor da saída do votador corresponde ao valor que possui maioria em suas entradas. Essa característica do votador pode ser observada na tabela da Fig. 46 (b).

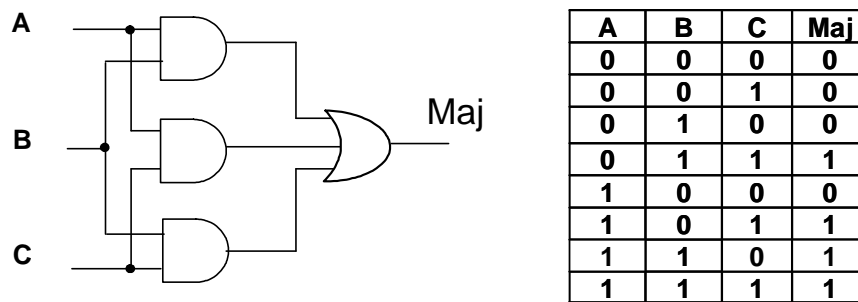


Figura 46 – (a) Votador de 1 bit e (b) tabela-verdade.

A implementação de votadores para vetores de bits é realizada instanciando-se mais de um circuito como este.

Com relação ao atraso das arquiteturas protegidas com TMR, note na Fig. 45 que, embora haja três somadores CSA, estes operam paralelamente e de forma independente uns dos outros. Assim, a única diferença entre as arquiteturas protegidas e as não-protegidas, com relação ao atraso, é a presença do votador. Levando-se em conta que o votador é o circuito responsável por enviar para os registradores de saída os valores corretos da computação realizada pelos somadores, este influencia o atraso crítico das arquiteturas protegidas.

Caso este trabalho tivesse focado na implementação das arquiteturas de somadores para fabricação com máscaras, o atraso crítico das arquiteturas protegidas com TMR poderia ser estimado com precisão somando-se o atraso do circuito votador ao atraso do CSA. Porém, como este trabalho visa a implementação das arquiteturas em FPGAs, esta análise não pode ser feita de maneira tão simplista. As características da arquitetura (granularidade dos blocos programáveis, disponibilidade de elementos de memória, estrutura da rede de interconexão programável etc) e os algoritmos de síntese lógica e mapeamento (*technology binding*) influenciam de sobremaneira nas

características das arquiteturas sintetizadas, tanto no que diz respeito ao desempenho (medido pelo atraso crítico ou pela frequência de operação) quanto no uso de recursos de hardware (medido pelo número de blocos programáveis, LUTs, ALUTs etc, conforme a arquitetura utilizada). Para que se tenha uma idéia do resultado da implementação de uma arquitetura dentro de um FPGA, a Fig. 47 apresenta um CSA de 16 bits protegido com TMR mapeado para um dispositivo EP2S15F484C3 da família Stratix II.

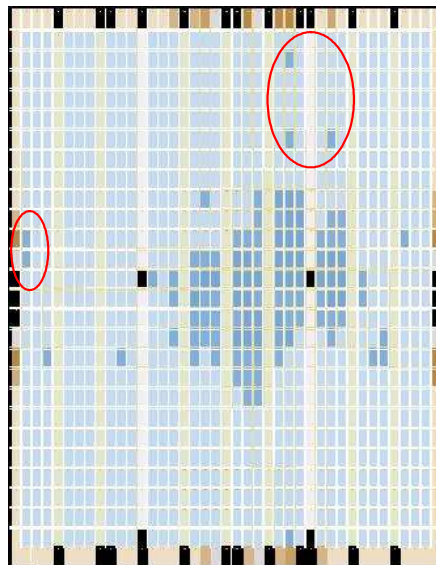
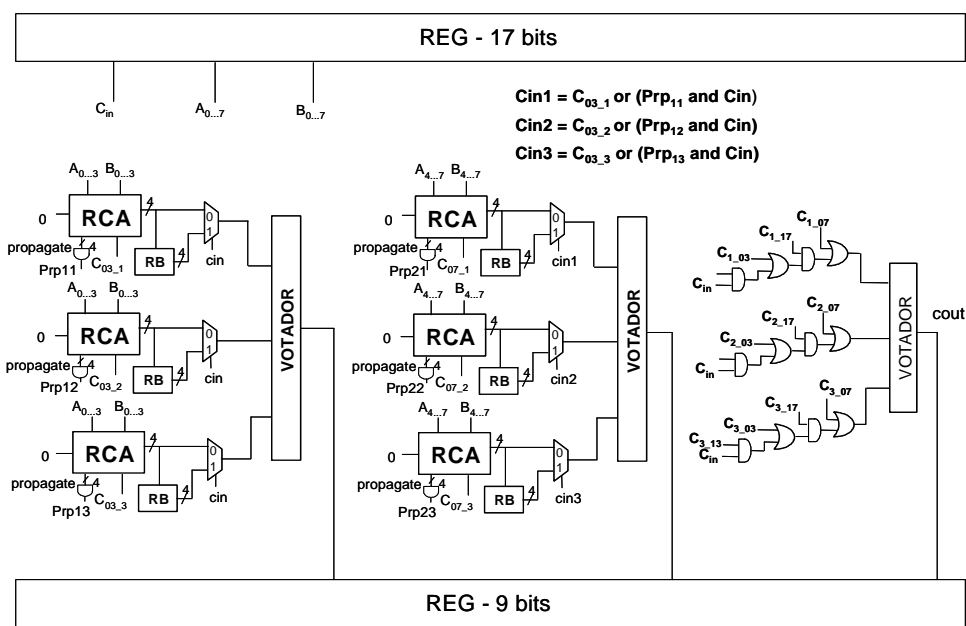


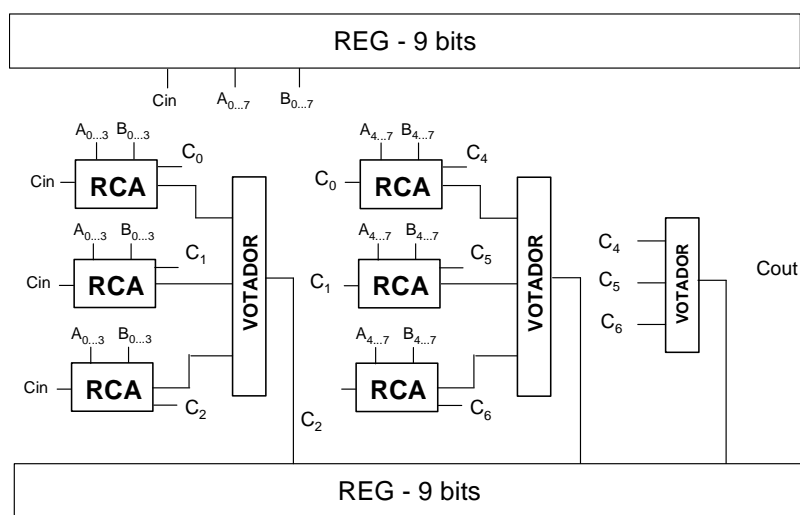
Figura 47 – Somador CSA de 16 bits mapeado para o FPGA da família Stratix II.

Com relação à Fig. 47, cada retângulo corresponde a uma estrutura ALM, já apresentada anteriormente. Os retângulos em cinza mais escuro, posicionados em maior número no centro do FPGA, correspondem às estruturas utilizadas pelo CSA TMR de 16 bits dentro do FPGA, mapeadas pela ferramenta de síntese. Note na Fig. 47 que a ferramenta de síntese mapeou alguns elementos da arquitetura do CSA para regiões um pouco afastadas do núcleo central onde se encontra a maioria dos ALMs em uso. Estes podem ser observados na Fig. 47 com círculo em volta. Tais estruturas correspondem a alguns bits dos registradores de entrada, informação esta retirada da própria ferramenta de síntese. Se por um acaso a saída de algum destes registradores fizesse parte do caminho crítico do somador, o atraso proveniente das estruturas de interconexão existentes entre esta saída e o centro do FPGA com certeza influenciaria no atraso crítico da arquitetura.

Os somadores RIC e RCA protegidos com TMR foram implementados da mesma forma que os somadores CSA protegidos. A Fig. 48 (a) e (b) apresenta os somadores RIC e CSA protegidos com TMR a fim de que possam ser comparadas às arquiteturas dos somadores CSA TMR.



(a)



(b)

Figura 48 - Somadores (a) RIC de 8 bits protegido com TMR (b) RCA de 8 bits protegido com TMR.

Observe na Fig. 48 (a) que a estrutura do somador RIC é tal que este não necessita que sejam utilizados 2 blocos somadores RCA por seção, assim como ocorre nos somadores CSA. Ao invés disso, o RIC faz uso de um bloco chamado RB, que utiliza no lugar de um RCA. Dessa forma, dada a aplicação de TMR ao RIC, o uso de recursos quando comparado aos somadores CSA resultam em uma economia de quase 3 somadores RCAS por seção.

Em se tratando do RCA protegido com TMR, percebe-se na Fig. 48 (b) que não diferente do que era esperado, este utiliza dentre os três tipos de somadores a menor quantidade de recursos. Também pode-se verificar na Fig. 48 (b) que, embora a arquitetura esteja dividida em seções, a característica do somador não foi alterada, haja vista que cada somador RCA de uma seção depende do *carry* de saída dos RCAs da seção anterior.

6.1.2 Detalhes de implementação dos somadores protegidos com TR

Conforme apresentado no capítulo 5, as técnicas baseadas em redundância temporal – TR (*Time Redundancy*) necessitam que sejam utilizados três sinais de relógio com um período de defasagem entre eles. Tal característica implica em um aumento da complexidade do circuito em função da necessidade de serem roteados 3 sinais de relógio. Assim, decidiu-se que, ao invés de utilizar 3 sinais de relógio, seria utilizada uma máquina de estados finitos – FSM (*Finite state machine*) com 4 estados (A, B, C e D) cadenciada por um sinal de relógio mestre.

Os estados A, B e C são utilizados como sinal de carga nos registradores responsáveis por realizar a redundância temporal. O estado D é utilizado como sinal de carga dos registradores de entrada e saída da arquitetura a ser protegida. A Fig. 49 apresenta um diagrama em blocos, dada a aplicação de TR a uma arquitetura de hardware hipotética (Fig. 49 (a)), utilizando uma FSM (Fig.49 (b)).

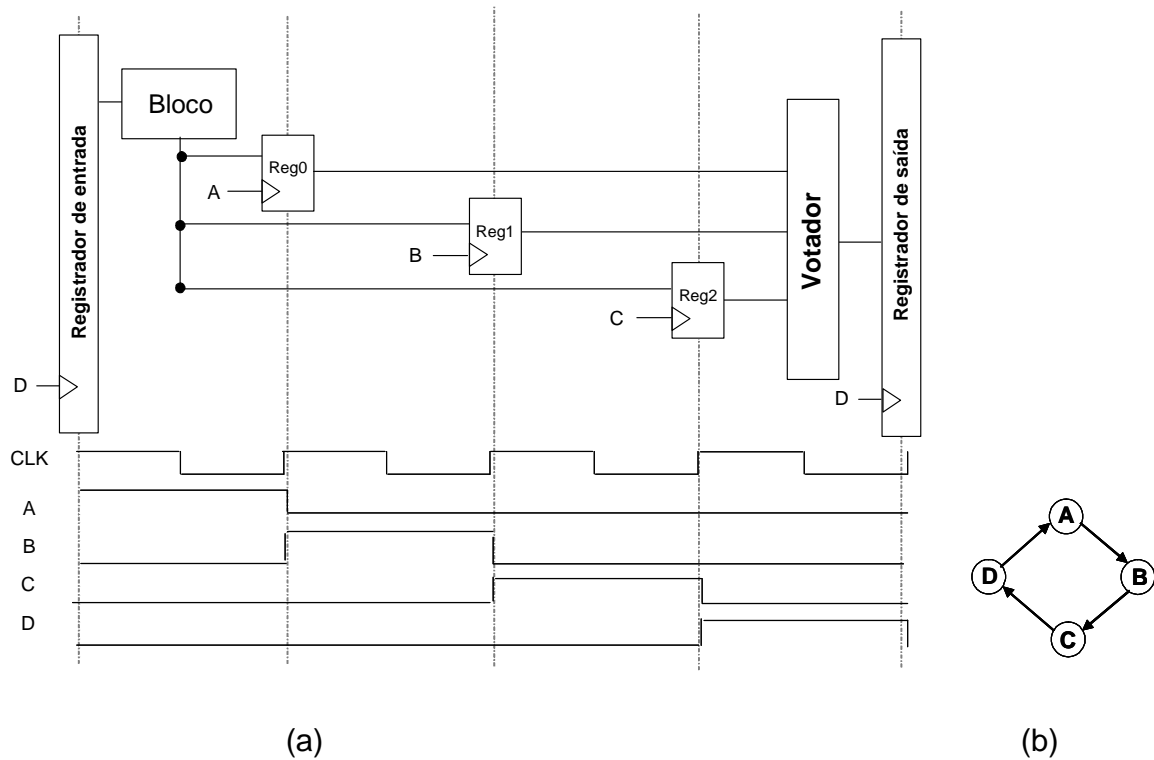


Figura 49 -(a) Diagrama em blocos de um módulo de hardware protegido com TR e (b) diagrama de estados do controle.

É possível perceber na Fig. 49 (a) que os sinais de controle enviados pela máquina de estados são controlados por um sinal de relógio principal. O período deste sinal de relógio principal deve ser no mínimo igual ao maior atraso verificado entre dois registradores da arquitetura. Verifica-se ainda na Fig. 49 (a) que existem 6 possíveis caminhos entre registradores: três referentes aos caminhos entre o registrador de entrada e cada um dos registradores intermediários (Reg1, Reg2, Reg3) e mais três entre os registradores intermediários e o registrador de saída. Uma vez encontrado o atraso máximo entre dois registradores, este será usado como período de relógio mestre (clk). Por conseguinte, o atraso total para a computação será 4 vezes o período do relógio.

A utilização de TR da forma como foi apresentada no parágrafo anterior implica em total segurança da arquitetura quanto à presença de uma falha transiente no bloco a ser protegido. Isto acontece porque o intervalo de tempo em que cada registrador intermediário armazena o resultado do bloco é no mínimo igual ao atraso do bloco a ser

protegido. Assim, caso ocorra uma falha transiente neste bloco e essa venha a ser armazenada por um dos registradores, esta mesma falha não poderá vir a ser armazenada por nenhum outro registrador, visto que quando um outro registrador estiver apto a armazenar o resultado a falha já terá se propagado pelo bloco e desaparecido do circuito.

Levando-se em consideração o que foi mostrado anteriormente foram então descritas em VHDL arquiteturas de somadores RCA, CSA e RIC de 4, 8, 16, 32, 64 e 128 bits, a partir de seções de 4 bits.

Cada seção de 4 bits possui uma parte do somador a ser protegido, 3 conjuntos de registradores e um votador. Além disso, foram utilizados registradores de entrada e saída em todas as arquiteturas descritas. A Fig. 50 apresenta um somador RIC de 8 bits protegido com TR. Note que neste caso, o período do ciclo de relógio é referente ao atraso crítico do somador RIC de 8 bits. Vale reforçar que este último ainda deve ser multiplicado por 4 para que o somador possa operar de forma correta.

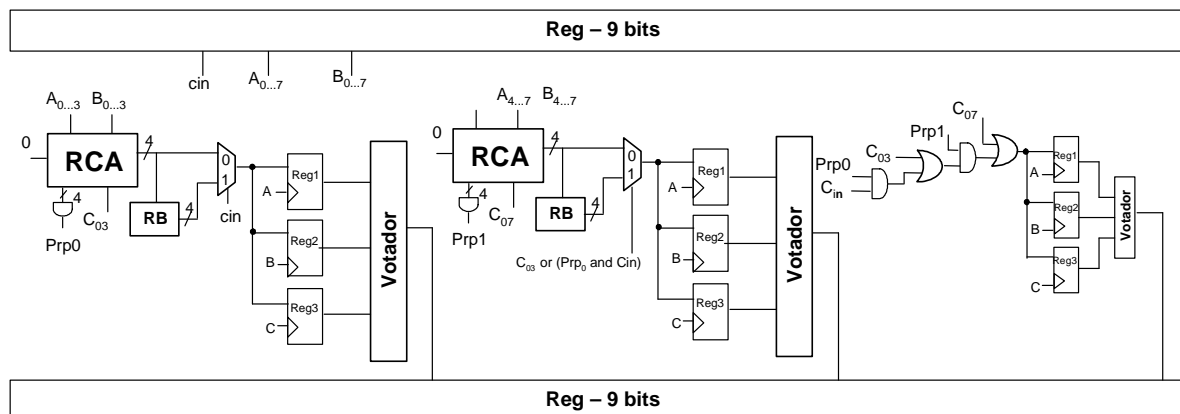


Figura 50 – RIC de 8 bits protegido com RT.

Em relação aos somadores CSA e RCA, as arquiteturas são praticamente idênticas à apresentada na Fig. 50, diferenciando-se apenas pelo tipo de somador a ser protegido.

6.1.3 Detalhes de implementação dos somadores protegidos com DWC+TR

A técnica de proteção DWC+TR – (*Duplication with Comparison Combined with Time Redundancy*), como o próprio nome explicita, faz uso da redundância temporal em sua implementação. Dadas as justificativas quanto à dificuldade de utilizarem-se vários sinais de relógio (apresentadas na subseção anterior), para a implementação de DWC+TR neste trabalho foi também utilizada uma máquina de estados. Como já foi mostrado no capítulo 5, a DWC+TR necessita de 3 sinais de relógio diferentes. A Fig. 51 apresenta o diagrama em blocos da aplicação de DWC+TR a um circuito a ser protegido, bem como a máquina de estados utilizada a fim de controlar os registradores responsáveis pela redundância temporal da técnica.

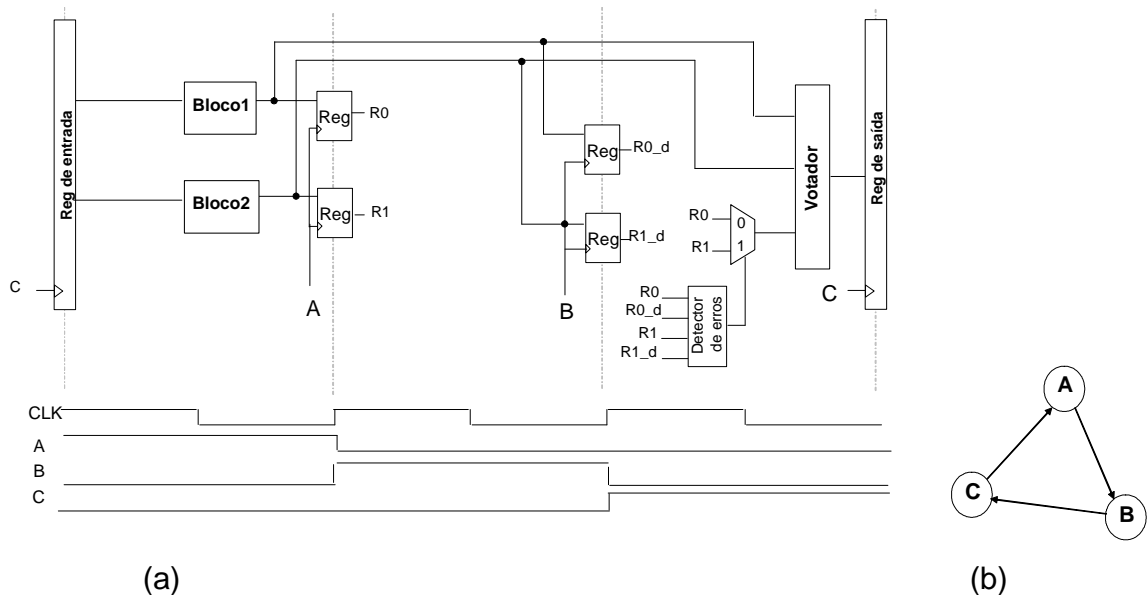


Figura 51– (a) Diagrama em blocos de um módulo de hardware protegido com DWC+TR e (b) diagrama de estados do controle.

Como pode ser visto na Fig. 51 (a), a máquina de estados utilizada pela DWC+TR possui 3 estados (A, B e C) e é controlada por um sinal mestre clk. Os estados A e B são utilizados como sinais de carga pelos registradores intermediários, conforme a Fig. 51 (a). Já o estado C é utilizado como sinal de carga pelos registradores de entrada e saída.

O período do sinal de relógio mestre clk é equivalente ao maior atraso entre dois registradores utilizados pela DWC+TR. Conforme pode ser observado na Fig. 51 (a), existem 10 caminhos possíveis: 4 referentes aos caminhos entre o registrador de entrada e cada registrador intermediário, 4 entre os registradores intermediários e o registrador de saída e mais 2 entre os registradores de entrada e saída.

Considerando-se que a técnica da forma como foi implementada necessita de 3 ciclos de relógio, o atraso crítico do circuito deve ser calculado multiplicando-se o período do relógio por 3. Com relação ao caminho direto entre os registradores de entrada e saída, note na Fig. 51 (a) que neste caso não é necessário um período de relógio só para a realização desta tarefa. Isto se deve ao atraso deste caminho ser menor quando comparado aos 3 ciclos de relógio utilizados para implementação da técnica. Assim, a computação realizada diretamente entre os registradores de entrada de saída pode ser efetuada em paralelo com as outras tarefas realizadas dada a aplicação da técnica.

Seguindo as características citadas nos parágrafos anteriores foram implementados os circuitos somadores RCA, RIC e CSA. A Fig. 52 apresenta a arquitetura de um somador RIC de 8 bits implementado com DWC+TR.

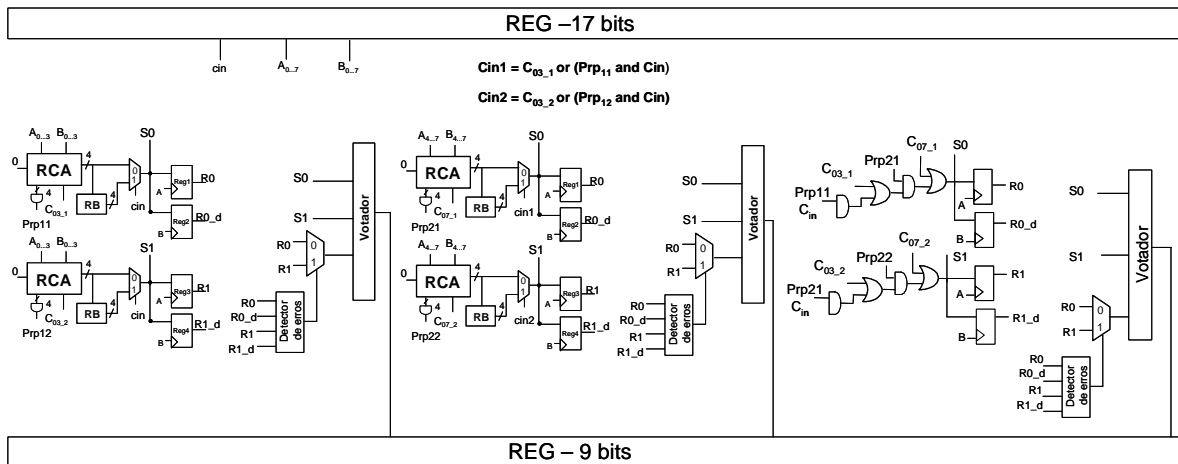


Figura 52 – RIC de 8 bits protegido com DWC+TR.

Assim como na aplicação de TR e TMR, as arquiteturas de somadores protegidas com DWC+TR foram implementadas a partir de blocos de 4 bits. Foi utilizado também neste caso um circuito a parte apenas para o cálculo do *carry* de

saída do circuito. Para este último, foram utilizados um votador, um multiplexador e um detector de erros, todos com um bit cada. É possível verificar na Fig. 52 que além do circuito votador, a técnica DWC+TR também utiliza um circuito detector de erros. Caso algum valor errôneo tenha sido armazenado por um dos registradores responsáveis por implementar a redundância temporal (Reg0, Reg1, Reg2 e Reg3), o detector de erros deve identificar qual bloco possui o valor errôneo e assim, através de um multiplexador enviar para a terceira entrada do votador o valor correto. A Fig. 53 mostra juntamente a tabela-verdade e a arquitetura deste bloco.

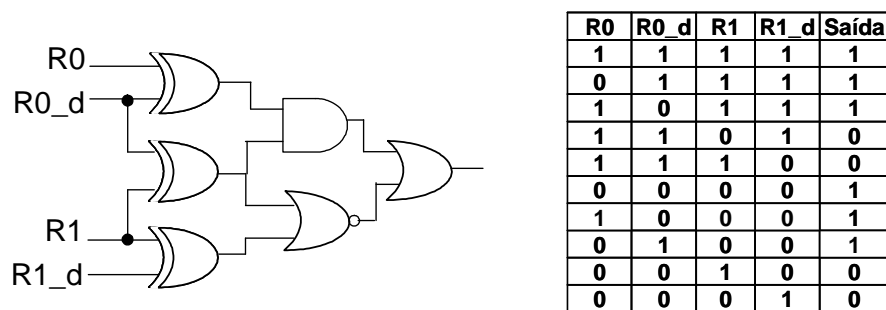


Figura 53 – (a) Circuito detector de erros e (b) sua tabela-verdade.

A tabela da Fig. 53 (b) apresenta todas as possibilidades de saída do detector de erros, dada a ocorrência de uma falha por vez nos blocos duplicados a serem protegidos. Caso contrário, o detector de erros não consegue detectar qual dos blocos duplicados apresentou a falha. O valor da saída do detector funciona como sinal de controle de um multiplexador. Este multiplexador recebe com entradas os resultados calculados pelos blocos duplicados armazenados nos registradores intermediários. Caso o detector de erros detecte uma falha em um dos blocos, este deve fazer com que o multiplexador envie para a terceira entrada do votador a entrada livre de falhas.

Como pode ser observado na Fig. 53, a quantidade de recursos despendida por um detector de erros é bastante significativa. Embora essa técnica tenha como objetivo enquadrar-se entre as técnicas TMR e TR, com relação a uso de recursos, quando utilizada para proteger circuitos de baixa complexidade pode apresentar resultados não satisfatórios. Esta tendência ficará mais clara nas seções seguintes onde apresentados os resultados, dada a aplicação de todas as técnicas aos somadores RIC, CSA e RCA.

6.2 Diretivas de compilação da ferramenta Quartus II

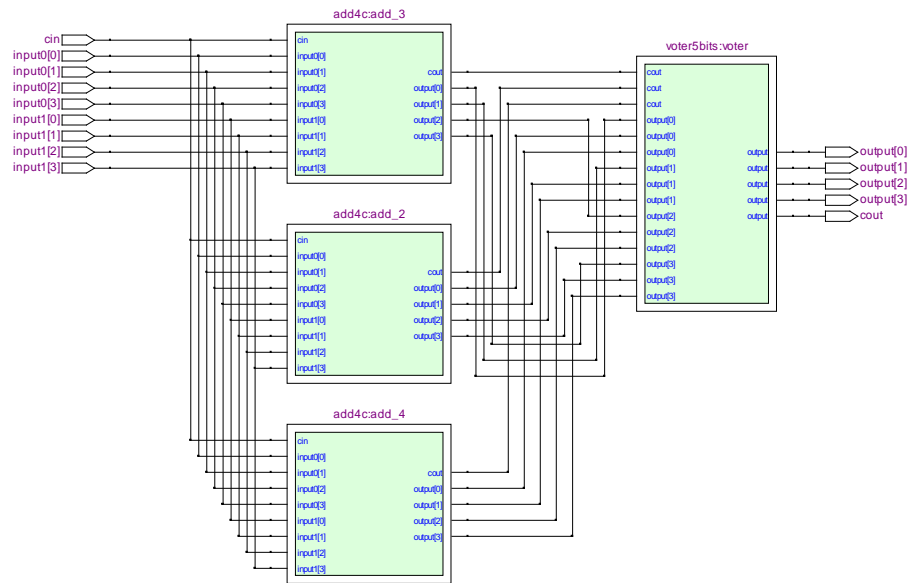
Ferramentas de síntese lógica e mapeamento para FPGAs tais como a ferramenta Quartus II da Altera possuem algoritmos capazes de detectar e eliminar blocos redundantes dentro da lógica criada pelo usuário. Levando em conta que o principal foco deste trabalho era a implementação de arquiteturas protegidas, era imprescindível evitar que a ferramenta realizasse tais otimizações.

A fim de evitar que a ferramenta Quartus II otimizasse qualquer lógica que lhe parecesse redundante, foi utilizada uma diretiva de compilação que inibe o compilador da ferramenta Quartus II a otimizar qualquer sinal protegido por ela. A sintaxe básica desta diretiva pode ser observada abaixo:

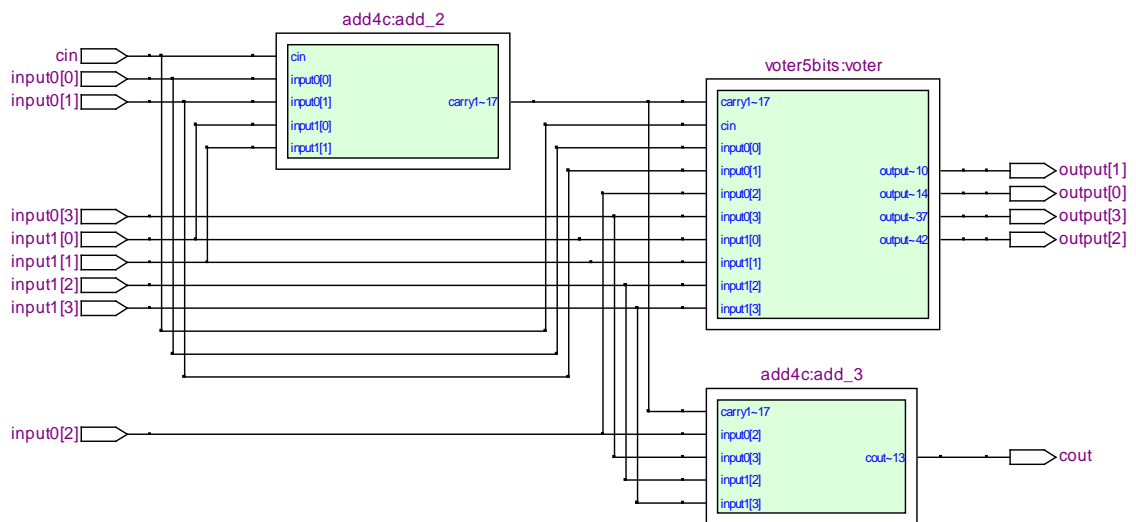
```
attribute syn_keep : boolean;  
signal exemplo : std_logic;  
attribute syn_keep of cout0 : signal is true
```

O código acima informa ao compilador que o sinal “exemplo” não deve substituído por nenhum outro sinal, mesmo que este seja idêntico a ele. A fim de exemplificar a eficácia destas diretivas, foram implementadas duas versões de um somador *Ripple Carry* de 4 bits protegido com TMR. Uma versão utilizando as diretivas e outra sem elas. A Fig. 54 (a) e (b) foi obtida com o uso da própria ferramenta Quartus II e corresponde aos somadores com diretivas e sem diretivas.

Note na Fig. 54 (a) que o compilador do Quartus II não realizou nenhum tipo de otimização com relação aos módulos redundantes. Já na Fig. 54 (b) percebe-se que o compilador eliminou um dos somadores triplicados e simplificou o circuito votador a fim de se adequar à nova arquitetura sintetizada. O anexo A apresenta um código VHDL de um somador RCA de 4 bits utilizando as diretivas de compilação.



(a)



(b)

Figura 54 – (a) RCA protegido com TMR utilizando as diretivas e (b) RCA protegido com TMR sem as diretivas.

6.3 Resultados de síntese para os Somadores Protegidos com TMR

A técnica TMR – *Triple Module Redundancy* caracteriza-se por ser, dentre as técnicas de proteção existentes, a mais eficiente e também possivelmente a pioneira. Porém, para que esta consiga alcançar tamanha eficiência, há um custo associado a sua implementação, tanto em termos de desempenho quanto em termos de recursos de hardware. Estas características poderão ser analisadas mais adiante, quando forem apresentados os resultados de síntese logo após terem sido descritas as arquiteturas de somadores protegidas.

O gráfico da Fig. 55 apresenta os resultados referentes ao desempenho das três arquiteturas de somadores (RCA, CSA e RIC) não-protegidas e protegidas com TMR.

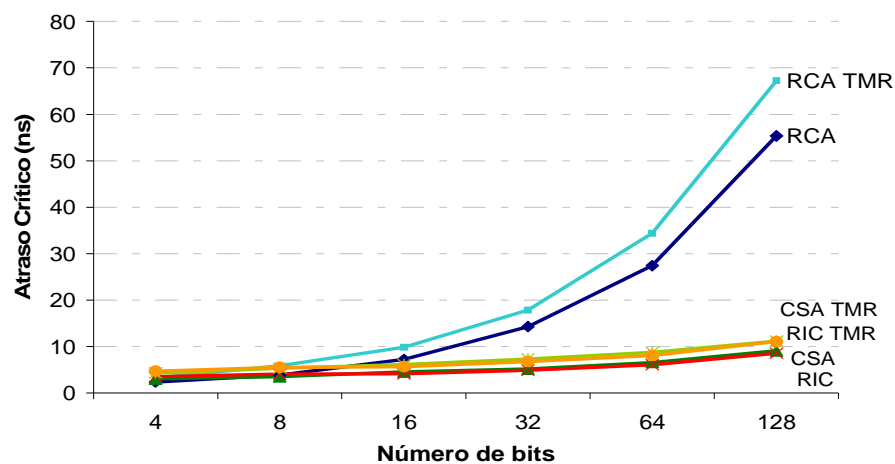


Figura 55 – Atraso crítico das arquiteturas não-protegidas e protegidas com TMR.

Assim como era esperado, as arquiteturas protegidas apresentaram atrasos críticos maiores quando comparadas com as arquiteturas não-protegidas. Isto se deve principalmente ao fato da presença de um circuito votador no caminho crítico de cada arquitetura, conforme já mostrado na seção anterior. Esta característica pode ser observada no gráfico da Fig. 55 quando são analisados comparativamente dois somadores de um mesmo tipo, sendo um protegido e o outro não-protegido. Nestes casos, percebe-se na Fig. 55 que é mantida uma linearidade em relação às diferenças

de atrasos entre as arquiteturas, independente da quantidade de bits dos somadores analisados.

Em se tratando mais precisamente das arquiteturas de somadores rápidos, a Fig. 56 apresenta com mais detalhes um gráfico comparativo entre o desempenho dos somadores RIC e CSA não-protegidos e protegidos com a técnica TMR.

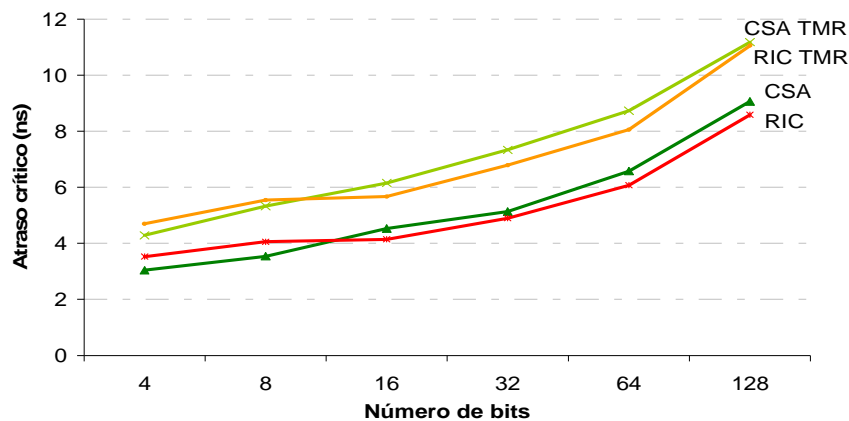


Figura 56 – Atraso crítico dos somadores rápidos não-protegidos e protegidos com TMR.

Vale observar que, similarmente aos resultados encontrados para as arquiteturas de somadores não-protegidas, os somadores RIC protegidos com TMR apresentam um melhor desempenho a partir de 16 bits, quando comparados aos somadores CSA também protegidos com TMR. Isto se deve ao mesmo motivo já apresentado no capítulo 4. A aplicação de uma técnica de proteção a um somador não altera as características de funcionamento destas arquiteturas, exceto por um pequeno aumento dos atrasos críticos destes somadores decorrentes dos votadores da técnica presentes no caminho críticos das arquiteturas. Considerando-se o fato de que todas as arquiteturas protegidas fazem uso deste mesmo tipo de votador, de maneira intuitiva é fácil perceber que estes circuitos contribuirão de maneira similar para o aumento dos atrasos críticos de cada arquitetura de somador protegido.

A fim de permitir um exame mais detalhado dos resultados de atraso obtidos, a tab.4 mostra todos resultados referentes a desempenho, gerados através da implementação das arquiteturas não-protegidas, bem como os resultados obtidos para

as arquiteturas protegidas com TMR. A tabela mostra ainda o acréscimo percentual nos atrasos críticos provocado pela aplicação da técnica de TMR às arquiteturas de somadores investigadas.

Tabela 4 - Comparação entre os atrasos críticos dos somadores protegidos com TMR e não-protegidos.

Bits	Não-protegidas (1)			TMR (2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)
4	2.38	3.52	3.04	3.24	4.69	4.28	+36%	+33%	+41%
8	3.86	4.05	3.54	5.86	5.47	5.33	+52%	+35%	+51%
16	7.21	4.14	4.53	9.85	5.66	6.15	+37%	+37%	+36%
32	14.28	4.8	5.13	17.8	6.78	7.33	+25%	+41%	+43%
64	27.39	6.07	6.57	34.3	8.05	8.72	+25%	+33%	+33%
128	55.33	8.58	9.06	67.2	11.1	11.7	+21%	+29%	+29%

Analisando-se as 3 últimas colunas da tab.4 tem-se em evidência o fato de que a utilização da técnica de proteção TMR implica, em todos os casos, em uma degradação do desempenho, indicada pelo aumento do atraso crítico das arquiteturas de somadores consideradas. Em se tratando do somador RCA, o acréscimo do atraso crítico variou entre 21% (para 128 bits) e 52% (para 8 bits). No caso dos somadores RIC, o maior aumento do atraso crítico pode ser observado para a arquitetura com 32 bits (41%), enquanto que com relação aos somadores CSA, o maior aumento do atraso crítico se verificou para o somador com 8 bits (51%).

Considerando-se apenas as arquiteturas protegidas, percebe-se através da tab.4 que a relação de desempenho entre os somadores rápidos (CSA e RIC) e o RCA não foi alterada diante da aplicação da técnica TMR. Para somadores com poucos bits, o RCA obteve valores de atraso de crítico inferiores àqueles apresentados pelos somadores rápidos: cerca de 44% em relação ao RIC de 4 bits e 32% em relação ao CSA de 4bits. Em contrapartida, para arquiteturas de 128 bits, por exemplo, os somadores RIC e CSA foram 505 % e 474%, mais rápidos que o RCAs, respectivamente. Considerando-se ainda as arquiteturas protegidas, o RIC apresenta um atraso 9% maior que o CSA, levando-se em conta somadores com 4 bits, e 2%

maior que o CSA para somadores com 8 bits. A partir de 16 bits, o RIC apresenta atrasos 8% inferiores em relação aos somadores CSA de 16, 32 e 64 bits, e 5% inferior em se tratando de somadores com 128 bits.

Analisando-se os dados de forma mais genérica pode-se concluir que, dada a aplicação da técnica de proteção TMR, as arquiteturas tiveram um aumento médio de 35% em seus atrasos críticos, quando comparadas com as arquiteturas não-protegidas.

Com relação ao uso de ALUTs do FPGA, o gráfico da Fig. 57 mostra um comparativo entre somadores protegidos com TMR e os somadores não-protegidos.

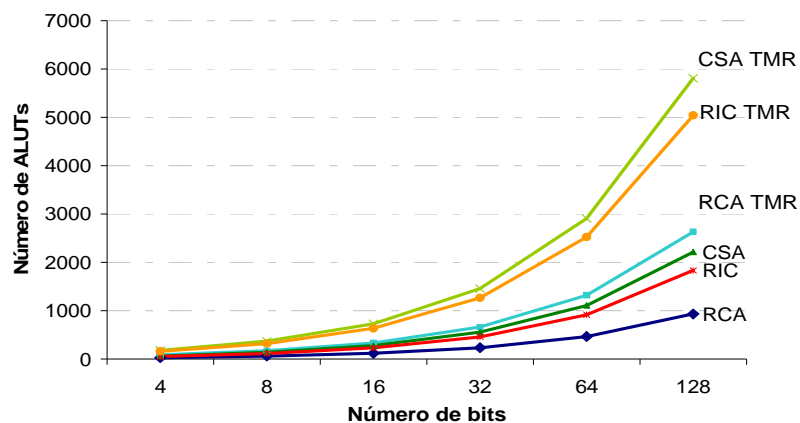


Figura 57 – Número de ALUTs utilizadas pelos somadores protegidos com TMR e não-protegidos.

Assim como já era esperado, pode-se verificar através do gráfico que as arquiteturas que fizeram uso da técnica TMR necessitaram mais recursos, quando comparadas com as arquiteturas não-protegidas. Isso se deve ao fato da necessidade da triplicação dos somadores e à utilização de um circuito votador. Ainda com relação ao gráfico da Fig. 57, e analisando-se apenas os somadores protegidos, percebe-se que o somador RCA é a arquitetura que necessita o menor número de ALUTs, como já era esperado. Assim como também já era esperado, o somador RIC aparece como uma arquitetura intermediária, quando se trata de utilização de recursos. Isto se deve ao fato deste fazer uso do bloco RB, ao invés de um outro somador como o CSA o faz.

Como pôde ser observado anteriormente, a técnica de proteção TMR remete a bons resultados com relação a desempenho das arquiteturas que venham a utilizá-la.

Porém, o gráfico da Fig. 57 mostra também que tal desempenho vem acompanhado de uso considerável de recursos extras. Assim, a fim de que sejam mais bem analisados os impactos decorrentes da utilização de TMR, a tab.5 apresenta todos resultados referentes à utilização de recursos tanto para as arquiteturas quanto para as protegidas com TMR. A tab.5 mostra ainda o acréscimo de recursos necessário para a proteção com TMR.

Tabela 5 - Comparação entre as arquiteturas de somadores protegidas com TMR e não-protegidas quanto à utilização de ALUTs.

Bits	Não-protegidos (1)			TMR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	33	55	65	87	160	184	+164%	+191%	+183%
8	62	118	142	173	323	371	+179%	+174%	+161%
16	120	232	280	337	637	733	+181%	+175%	+162%
32	236	460	556	665	1269	1461	+182%	+176%	+163%
64	468	918	1110	1321	2525	2909	+182%	+175%	+162%
128	932	1832	2216	2633	5042	5809	+183%	+175%	+162%

Observando-se as últimas colunas da tab.5, confirma-se a expectativa de que em todos os casos houve um acréscimo de recursos para que as arquiteturas pudessem ser protegidas TMR. Dadas algumas pequenas variações, percebe-se também um certo equilíbrio entre as arquiteturas, em se tratando da quantidade de recursos extras necessários, para que a técnica possa ser colocada em prática. No pior caso, referente à arquitetura com 128 bits, o somador RCA necessitou de 183% de recursos extras. Já o somador RIC precisou de 191%, em se tratando da arquitetura com 4 bits. Com valores não muito diferentes, a quantidade de recursos extras demandada para o somadores CSA variou entre 161% (CSA de 8 bits) e 183% (CSA de 4 bits)

Ainda em relação apenas às arquiteturas protegidas, o somador RIC, quando comparado ao somador RCA, o mais barato em termos de utilização de recursos, utiliza entre 83% (RIC com 4 bits) e 91% (RIC com 128 bits) mais recursos que o somador

RCA. Já o somador CSA, necessita entre 111% (para 4 bits) e 120% (para 128 bits) mais recursos que o somador RCA. Confrontando-se diretamente os resultados obtidos pelos somadores RIC TMR, CSA TMR e RCA TMR, é possível notar que o RIC utiliza em média 15% menos recursos de hardware que o CSA e 83% mais recursos que o RCA.

A fim de se obter um diagnóstico do impacto da aplicação da técnica TMR no uso de recursos, pode-se extrair da tab.5 que o uso desta técnica ocasionou um acréscimo médio de recursos em torno de 174%, considerando todos os resultados para os somadores experimentados.

De posse dos resultados apresentados anteriormente, confirmou-se a expectativa de que a utilização de TMR como técnica de proteção implica em uma demanda significativa de recursos extras. Porém, fica claro, através dos resultados, que a técnica, não degrada o desempenho de maneira muito significativa, sendo portanto uma escolha adequada, quando este quesito for o objetivo mais importante de projeto.

6.4 Resultados de síntese para somadores protegidos com TR

Como pôde ser observado na seção anterior, a construção de arquiteturas que façam uso da técnica de TMR tem como principal desvantagem o acréscimo de recursos necessário. Técnicas de proteção baseadas em redundância temporal (RT) têm o objetivo de amenizar os efeitos supracitados. Porém, para que sejam alcançados tais objetivos, TR acaba sacrificando o desempenho das arquiteturas. Assim, a fim de que possam ser investigados na prática os impactos da utilização desta técnica, todos os tipos de somadores apresentados neste trabalho foram também implementados utilizando TR.

O gráfico da Fig. 58 apresenta os resultados de atrasos críticos para as arquiteturas de somadores protegidas com TR, bem como para as arquiteturas não-protegidas.

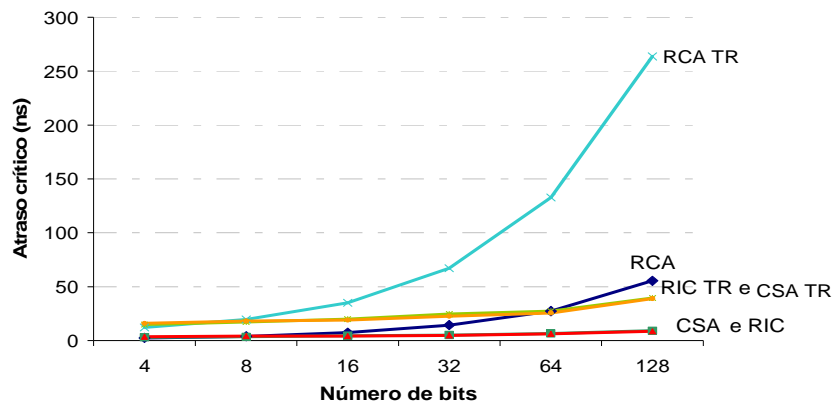


Figura 58 – Atraso crítico das arquiteturas protegidas com redundância temporal -TR e não-protégidas.

Como pode ser observado no gráfico da Fig. 58, o atraso das arquiteturas protegidas é significativamente maior que o atraso das arquiteturas não-protégidas. Isto se deve ao fato de que a utilização desta técnica, conforme já explicado na seção 6.1.2, faz uso de 4 ciclos de relógio. Assim, é natural que o atraso das arquiteturas correspondam a valores bem maiores. Em se tratando apenas das arquiteturas protegidas com TR, assim como já era esperado, a partir de 8 bits, o somador RCA apresentou os maiores atrasos quando comparados aos outros tipos de somadores.

Levando-se em conta somente os somadores rápidos protegidos, a Fig. 59 mostra em mais detalhes o comportamento destas arquiteturas.

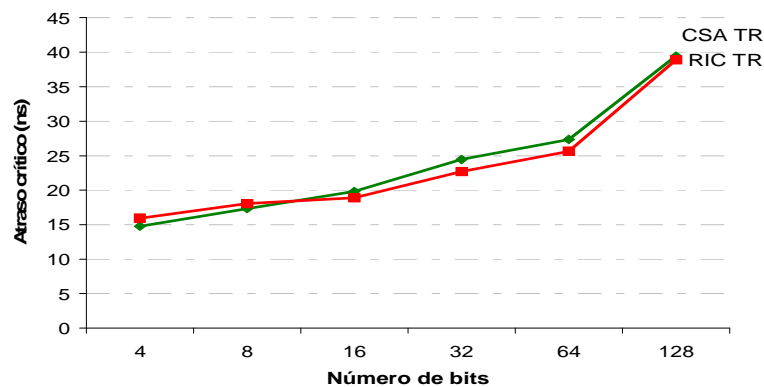


Figura 59 - Atraso crítico dos somadores rápidos protegidos com TR.

Note que o comportamento, já observado tanto nos somadores protegidos com TMR quanto nos somadores não-protegidos, aparece novamente quando os somadores são implementados com a técnica TR, qual seja: a partir de 16 bits os somadores RIC apresentam um melhor desempenho, quando comparados com os somadores CSA. O motivo pelo qual isso acontece já foi explicado anteriormente e o fato da utilização de uma técnica distinta, TR neste caso, não altera as características internas destes tipos de somadores.

De posse dos resultados das técnicas TMR e TR, já é possível realizar um comparativo entre elas. O gráfico da Fig. 60 mostra os atrasos críticos de todas as arquiteturas de somadores utilizando estas duas técnicas.

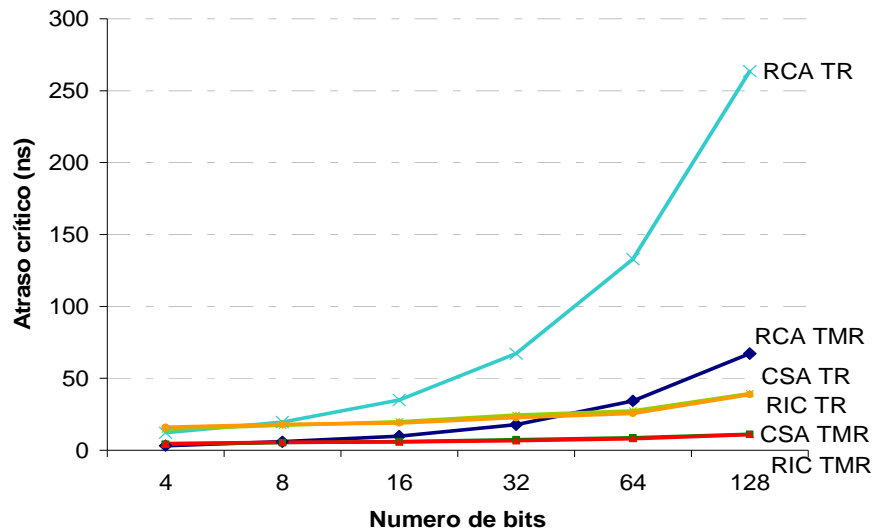


Figura 60 – Comparação entre os atrasos críticos dos somadores protegidos com TMR e TR.

É possível observar que em todos os casos os somadores que utilizam TMR apresentam atrasos menores quando comparados com os mesmos tipos de somadores, protegidos com a técnica TR.

A tab.6 apresenta os resultados de atrasos das arquiteturas protegidas com TR e não-protegidas, bem como o acréscimo de atraso em porcentagem decorrente da utilização de TR.

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

Bits	Não-protegidas (1)			TR (2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)
4	2.38	3.52	3.04	12.09	15.94	14.79	+408%	+353%	+387%
8	3.86	4.05	3.54	19.54	18.06	17.32	+406%	+346%	+389%
16	7.21	4.14	4.53	34.94	18.87	19.83	+385%	+356%	+338%
32	14.28	4.8	5.13	67.2	22.67	24.44	+371%	+372%	+376%
64	27.39	6.07	6.57	132.8	25.63	27.36	+385%	+322%	+316%
128	55.33	8.58	9.06	263.6	38.8	39.41	+376%	+352%	+335%

Observando-se as últimas 3 colunas da tab.6 pode-se perceber que em todos os casos a utilização da técnica TR implicou em um aumento do atraso crítico das arquiteturas de somadores. Em se tratando mais especificamente do somador RCA, esse aumento variou de 371 % até 408 % para as arquiteturas com 32 e 4 bits, respectivamente. Com relação aos somadores RIC e CSA, as maiores diferenças foram observadas para os somadores com 32 bits, em se tratando do RIC e 8 bits, em se tratando de CSA. Nestes casos as diferenças foram 372% e 389%, respectivamente.

Observando-se apenas os somadores rápidos RIC e CSA protegidos com TR, novamente os somadores RIC apresentaram menores atrasos a partir de 16 bits. Esta redução chegou a 7% em se tratando dos somadores com 32 bits. Já em relação aos somadores com 4 e 8 bits, o somador RIC apresentou atrasos 7% e 4% maiores que os somadores CSA.

De maneira geral, uma análise dos resultados mostra que, em média, as arquiteturas protegidas com TR apresentaram um aumento de 365% em seus atrasos críticos.

A tab.7 apresenta o acréscimo de atrasos para os somadores protegidos com TR, em relação aos somadores protegidos com TMR. Vale ressaltar aqui que o cálculo das diferenças entre os atrasos das arquiteturas, utilizando as duas técnicas até aqui apresentadas, foram realizados tomando-se por referencial as arquiteturas protegidas com TMR. Estes últimos podem ser observados nas três últimas colunas da tab.7.

Tabela 7 - Comparação entre os atrasos críticos dos somadores protegidos com TR e TMR.

Bits	TMR (1)			TR (2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA	RIC	CSA
4	3.24	4.69	4.28	12.09	15.94	14.79	+273%	+240%	+246%
8	5.86	5.47	5.33	19.54	18.06	17.32	+233%	+230%	+225%
16	9.85	5.66	6.15	34.94	18.87	19.83	+255%	+233%	+222%
32	17.8	6.78	7.33	67.2	22.67	24.44	+278%	+234%	+233%
64	34.3	8.05	8.72	132.8	25.63	27.36	+287%	+218%	+214%
128	67.2	11.1	11.7	263.6	38.8	39.41	+292%	+250%	+237%

Como já era esperado, em todos os casos, as arquiteturas que utilizaram TR obtiveram um desempenho inferior ao desempenho das mesmas arquiteturas utilizando TMR. Os somadores RCA utilizando TR, apresentaram, em média, atrasos 270% maiores quando comparados aos mesmos utilizando TMR. A maior variação foi apresentada pelos somadores RCA com 128 bits, com 292%. Já os somadores CSA apresentaram em média um aumento nos valores dos atrasos de 230%. Neste caso, os somadores com 4 bits apresentaram a maior variação: cerca de 246%. Por fim, com relação aos somadores RIC, o aumento dos atrasos foi de 234%, em média. Em se tratando deste último, a variação ficou entre 218% (para somadores com 64 bits) e 250% (para somadores com 128 bits).

O principal motivo da aplicação da técnica TR é a perspectiva de redução do uso de recursos, em comparação com a técnica TMR. Dessa maneira, foram analisadas todas as arquiteturas de somadores com relação à quantidade de ALUTs utilizadas. O gráfico da Fig. 61 apresenta os resultados obtidos através da síntese das arquiteturas protegidas com TR, bem como os resultados obtidos pelas mesmas arquiteturas não-protegidas. Conforme esperado, a utilização da técnica implicou em um aumento da utilização de recursos em todos os casos. Porém, como veremos mais adiante, tal aumento é significativamente menor quando comparado ao necessário para a aplicação de TMR.

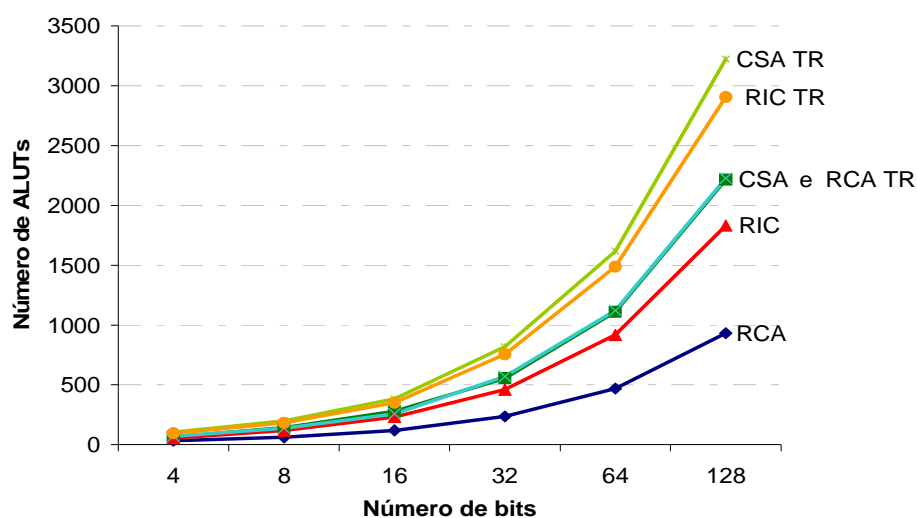


Figura 61 – Número de ALUTs utilizadas pelos somadores não-protetidos e protegidos com TR.

Ainda com relação à Fig. 61, e analisando-se apenas os somadores protegidos com TR, novamente podemos verificar que o somador RCA é a arquitetura que necessita da menor quantidade de recursos de hardware. É notório também que o somador RIC, assim como era esperado, apresenta-se novamente como uma arquitetura intermediária em relação ao uso de recursos.

Com relação às duas técnicas de proteção já apresentadas, pode-se observar, através do gráfico da Fig. 62, que as arquiteturas que fazem uso da técnica TR apresentam uma utilização de recursos significativamente menor quando comparadas às arquiteturas que fazem uso de TMR. Isto se deve ao fato da técnica de TR utilizar apenas um exemplar do bloco a ser protegido, posto que a redundância se dá através do armazenamento do resultado do bloco em registradores, em tempos diferentes, conforme já explicado anteriormente.

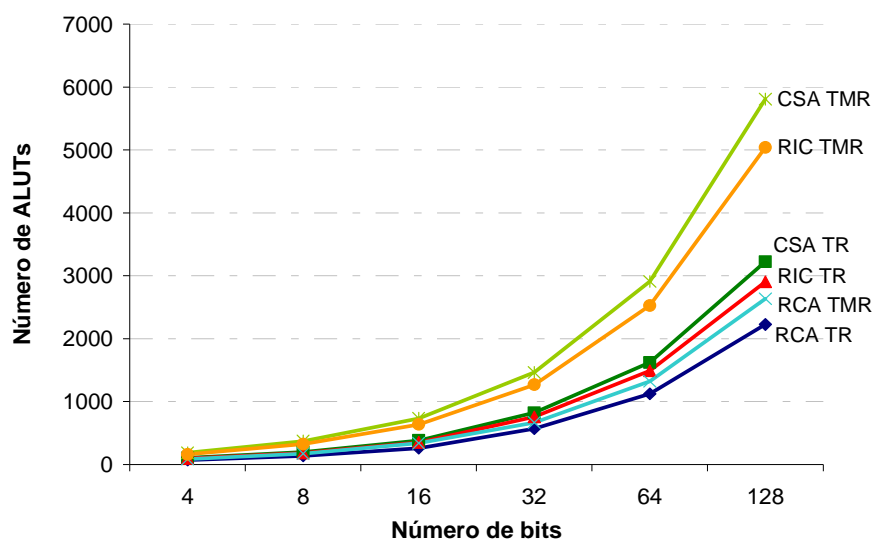


Figura 62 – Comparação entre os somadores protegidos com as técnicas TR e TMR, quanto à utilização de recursos.

A tab.8 apresenta de forma completa todos resultados de uso de ALUTs obtidos para as arquiteturas protegidas com TR, bem como os resultados para as arquiteturas não-protegidas. A tab.8 apresenta também, nas últimas colunas, valores correspondentes ao acréscimo de recursos em percentagem, dada a aplicação de TR em todos os somadores.

Tabela 8 - Comparação entre as arquiteturas de somadores protegidos com TR e não-protegidas, quanto à utilização de ALUTs.

Bits	Não-protegidas (1)			TR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	33	55	65	72	96	104	+118%	+75%	+60%
8	62	118	142	137	183	199	+121%	+55%	+40%
16	120	232	280	259	351	382	+116%	+51%	+36%
32	236	460	556	570	756	820	+142%	+64%	+47%
64	468	918	1110	1122	1489	1618	+140%	+62%	+46%
128	932	1832	2216	2226	2905	3223	+139%	+59%	+45%

Através dos resultados mostrados nas 3 últimas columnas da tab.8 é possível verificar-se que em todos os casos, a aplicação de TR implicou em um acréscimo de recursos utilizados. Seguindo a tendência anteriormente observada, o somador RIC manteve-se como uma arquitetura intermediária, dada a aplicação de TR. Confrontando-o diretamente com o somador CSA também protegido, o RIC utilizou em média 8% menos recursos. Com relação ao acréscimo de recursos exigido pela utilização da técnica TR, o somador RCA foi o tipo de arquitetura que apresentou os piores resultados percentuais. Em se tratando mais especificamente do RCA com 32 bits, o acréscimo de recursos chegou a 142%. Em contrapartida, o somador CSA foi a arquitetura que apresentou melhor adaptação à técnica, sendo o somador com 4 bits a arquitetura que mais demandou recursos extras: cerca de 60%. Percebe-se, através destes últimos resultados apresentados, uma tendência no comportamento das arquiteturas protegidas com TR. O gráfico da Fig. 63 apresenta esta tendência, dada a aplicação de TR às arquiteturas de somadores, bem como já aproveita para mostrar também o comportamento destas mesmas arquiteturas dada a aplicação de TMR.

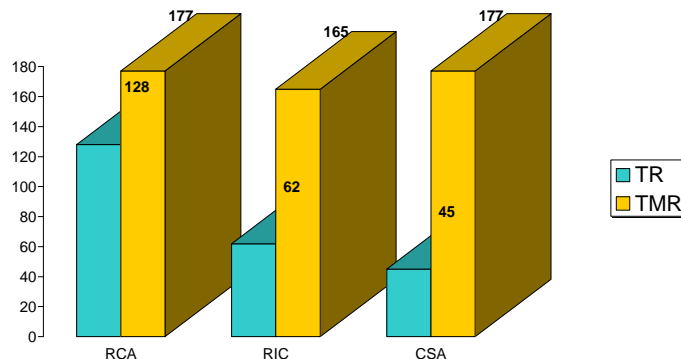


Figura 63 – Número médio de ALUTs demandado pelos somadores, dada a aplicação das técnicas TMR e TR.

É perceptível, através do gráfico da Fig. 63, que a aplicação da técnica TMR implica em uma média de utilização de recursos extras quase que invariável, independentemente da arquitetura protegida. Entretanto, em se tratando dos somadores protegidos com TR, a quantidade de recursos extras exigido para a utilização desta técnica, tende a tornar-se cada vez menos significativa a medida que

aumenta a quantidade de recursos necessários para se implementar o circuito a ser protegido.

Uma outra observação importante a ser feita é com relação ao uso de recursos dos somadores RIC e RCA, ambos protegidos com TR. Em média, o RIC apresenta um acréscimo de recursos de 33% em relação ao RCA, valor este significativamente pequeno levando em conta que o RIC é um somador rápido e o RCA o é mais lento dentre os somadores.

A fim de que seja feita uma análise comparativa mais detalhada entre todas arquiteturas de somadores protegidos com ambas as técnicas apresentadas até aqui, a tab.9 apresenta os resultados em termos de uso de recursos de todas arquiteturas de somadores, bem como as diferenças em percentagem de cada tipo de somador utilizando cada uma das técnicas. Os resultados das diferenças na tab.9 são calculados tomando por referência os somadores protegidos com TMR.

Tabela 9 - Comparação entre as arquiteturas de somadores protegidos com TR e TMR, quanto à utilização de ALUTs.

Bits	TMR (1)			TR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	87	160	184	72	96	104	-17%	-40%	-43%
8	173	323	371	137	183	199	-21%	-43%	-46%
16	337	637	733	259	351	382	-23%	-45%	-48%
32	665	1269	1461	570	756	820	-14%	-40%	-44%
64	1321	2525	2909	1122	1489	1618	-15%	-41%	-44%
128	2633	5042	5809	2226	2905	3223	-15%	-42%	-45%

Os resultados das últimas 3 colunas da tab.9 explicitam de forma clara que a utilização da técnica TR implica em uma economia em termos de uso de recursos, quando comparada à utilização da técnica TMR. A última coluna da tab.9 que apresenta os resultados referentes ao somador CSA, o maior somador experimentado neste trabalho, evidencia o que foi mostrado no gráfico da Fig. 63. A aplicação de TR a arquiteturas maiores implica em uma maior economia de recursos (ALUTs), quando comparada à utilização de TMR. Para o CSA de 16 bits, a economia de ALUTs ficou em torno de 48%, ou seja, quase metade dos recursos utilizados pelo mesmo somador

utilizando TMR como técnica de proteção. Outro aspecto importante é com relação ao somador RCA. Note que para este tipo de somador a maior economia ficou em torno de 23%, para os somadores com 16 bits. Considerando que o acréscimo nos atrasos dos somadores que utilizam a técnica TR é bastante significativo, esta por sua vez pode vir a não ser uma boa alternativa para este tipo de arquitetura. Isto se deve principalmente ao fato de que a perda em desempenho deste somador não é compensada com uma economia de recursos, quando comparada ao mesmo tipo somador protegido com TMR.

Com relação ao somador RIC protegido com TR, a redução de recursos utilizados, assim como o CSA, se mostrou ser significativa, quando comparada aos mesmos somadores protegidos com TMR. Neste caso, a variação ficou entre 40% (RIC de 4 e 32 bits) e 45% (RIC de 16 bits).

Embora o tipo de arquitetura a ser protegida seja de suma importância na escolha de uma técnica de proteção adequada, em se tratando do universo de arquiteturas experimentadas, consegue-se extrair da tab.9 também que as arquiteturas protegidas com TR consomem em média 35% menos recursos que as arquiteturas protegidas com TMR.

Diante dos resultados apresentados e as ponderações feitas com relação a estes, pode-se dizer que a aplicação da técnica de TR cumpre com a obrigação de obter um menor consumo de recursos. Seja o desejo de um projetista a implementação de um somador protegido com o menor acréscimo de área possível, a utilização de TR é com certeza uma eficiente alternativa.

6.5 Resultados de síntese para somadores protegidos com DWC+TR

A técnica de proteção DWC+TR (*Duplication with Comparison Combined with Time Redundancy*) visa a utilização de características de mais de uma técnica, com intuito de extrair os pontos positivos de cada técnica. Assim como já foi explicado no capítulo 5, esta técnica une características de técnicas de detecção, neste caso a Duplicação com comparação, com técnicas baseadas em redundância temporal. O objetivo principal é proteger uma arquitetura, sem que para isso seja necessária a

utilização de uma grande quantidade de recursos, e tampouco seja decorrente uma enorme perda de desempenho. Assim, em se tratando das três técnicas de proteção experimentadas neste trabalho, a DWC+TR objetiva encaixar-se em uma posição intermediária em relação às outras duas técnicas, tanto no que se refere a uso de ALUTs quanto no que diz respeito a atraso crítico.

Da mesma forma como foram implementados todos os tipos de somadores deste trabalho utilizando as técnicas até aqui apresentadas, a técnica DWC+TR também foi experimentada com o intuito de que pudessem ser analisadas suas características, quando aplicadas a arquiteturas de somadores.

A fim de que possam ser explorados os resultados de forma mais intuitiva, o gráfico da Fig. 64 apresenta os resultados comparativos entre as arquiteturas não-protegidas e as arquiteturas protegidas com DWC+TR. Como já era de se esperar, o gráfico deixa claro que em todos os casos, os somadores que fizeram uso da técnica de proteção DWC+TR tiveram uma perda de desempenho. Isto se deve principalmente ao fato desta técnica implicar em um uso de 3 ciclos de relógio a fim de que possa ser realizado um cálculo de soma. Tal característica já foi apresentada anteriormente na seção 6.1.3.

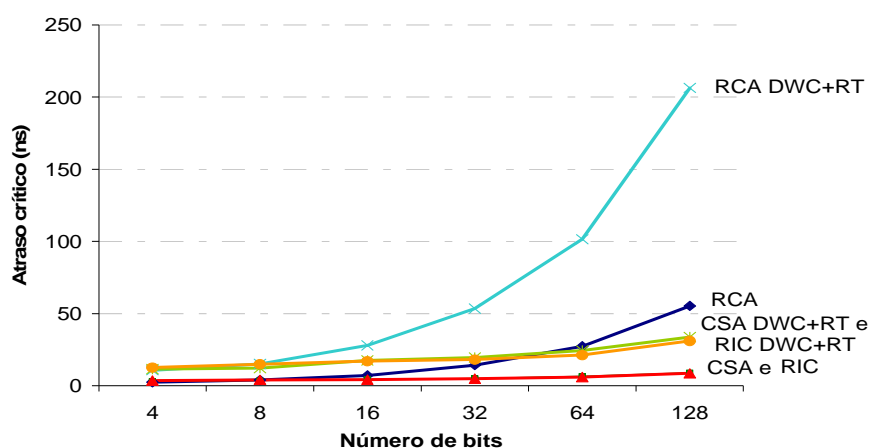


Figura 64– Atraso crítico das arquiteturas não-protegidas e protegidas com DWC+TR.

Assim como em todas as situações já apresentadas anteriormente, o RCA, como já era esperado, apresentou o pior desempenho dentre as arquiteturas protegidas. Com relação apenas aos somadores rápidos protegidos com DWC+TR,

novamente o comportamento revelado por estas arquiteturas apresentou as mesmas características. O somador RIC, a partir de 16 bits passou a ter um melhor desempenho que o somador CSA, conforme a Fig. 65. Novamente vale ressaltar que a aplicação da técnica em nada muda as características internas das arquiteturas.

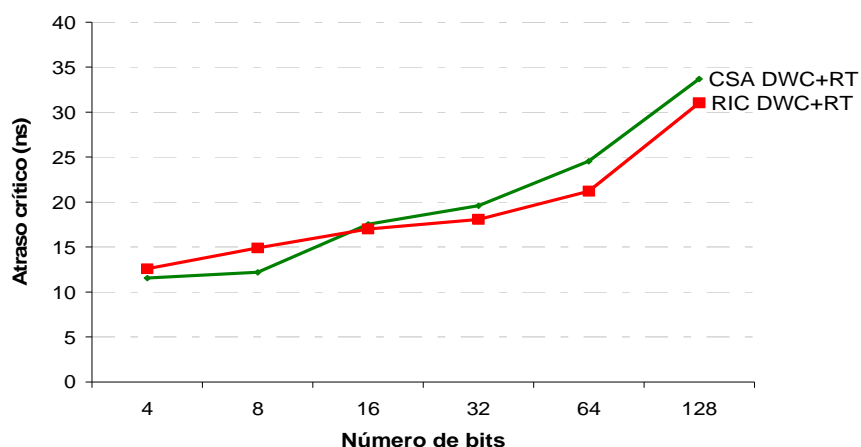


Figura 65 - Atraso crítico dos somadores rápidos protegidos com DWC+TR.

Fazendo-se um comparativo entre todas as arquiteturas protegidas, haja vista que já possuímos em mãos os resultados de 3 técnicas, podemos notar na Fig. 66 que a técnica DWC+TR apresenta-se com características intermediárias em relação a atraso crítico, quando comparadas com TMR e TR. Esse resultado era esperado já que esta técnica visa enquadrar-se entre as duas técnicas anteriormente apresentadas, em termos de desempenho.

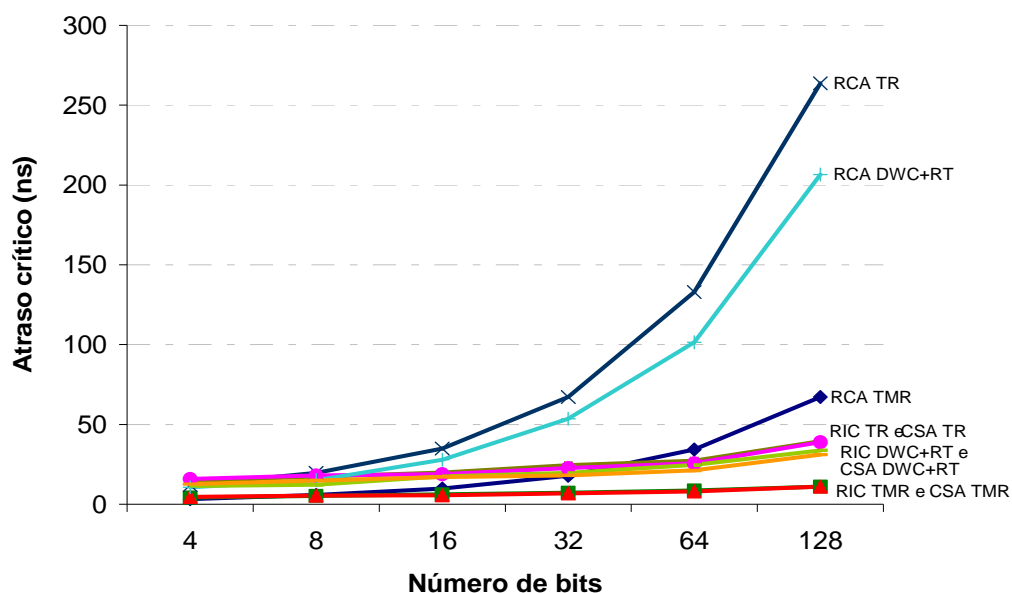


Figura 66 – Comparação entre os atrasos críticos de todos os somadores protegidos com TMR, TR e DWC+TR.

Com intuito de melhor compreender os resultados, a tab.10 apresenta em detalhes todos os resultados de atrasos referentes às arquiteturas protegidas com DWC+TR, bem como os resultados das arquiteturas não-protegidas. Assim como as outras tabelas apresentadas anteriormente, esta também apresenta os resultados referentes ao acréscimo de atraso, dada a aplicação da técnica DWC+TR às arquiteturas de somadores.

Tabela 10 - Comparação entre os atrasos críticos dos somadores protegidos com DWC+TR e não-protegidos.

Bits	Não-protegidas (1)			DWC+TR(2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA	RIC	CSA
4	2.38	3.52	3.04	10.72	12.58	11.54	+350%	+257%	+280%
8	3.86	4.05	3.54	15.19	14.89	12.19	+294%	+268%	+244%
16	7.21	4.14	4.53	27.9	17.01	17.53	+287%	+311%	+287%
32	14.28	4.8	5.13	53.51	18.08	19.57	+275%	+277%	+281%
64	27.39	6.07	6.57	101.5	21.2	24.53	+271%	+249%	+273%
128	55.33	8.58	9.06	206.4	31.02	33.67	+273%	+262%	+272%

Note que, dando prosseguimento ao mesmo comportamento já apresentado pelas outras técnicas, em todos os casos, os somadores apresentaram quedas de desempenho quando aplicada a técnica DWC+TR. A maior queda dentre todos os somadores foi observada para RCA de 4 bits, com um acréscimo do atraso crítico em 350%. O RIC apresentou acréscimo dos atrasos que variaram de 249% a 311% referentes aos RIC com 64 e 16 bits, respectivamente. Em se tratando do somador CSA, a maior diferença de atraso foi apresentada pelo CSA de 16bits, cerca de 287% superior.

Os somadores RIC, utilizando DWC+TR, seguindo a tendência, quando comparados ao CSAs, obtiveram um melhor desempenho, considerando as arquiteturas a partir de 16 bits. Essa melhora foi de 15% em se tratando do somador RIC com 64 bits.

A fim de que sejam feitas comparações entre o desempenho proporcionado por todas as técnicas até agora apresentadas, as tab.11 e tab.12 realizam, separadamente, um comparativo entre as 3 técnicas, duas a duas. A tab.11 compara os resultados gerados pelas técnicas DWC+TR e TMR e a tab.12 confronta os resultados para DWC+TR e TR.

Tabela 11 - Comparação entre os atrasos críticos dos somadores protegidos com TMR e DWC+TR.

Bits	TMR (1)			DWC+TR (2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA	RIC	CSA
4	3.24	4.69	4.28	10.72	12.58	11.54	+231%	+168%	+170%
8	5.86	5.47	5.33	15.19	14.89	12.19	+159%	+172%	+129%
16	9.85	5.66	6.15	27.9	17.01	17.53	+183%	+201%	+185%
32	17.8	6.78	7.33	53.51	18.08	19.57	+201%	+167%	+167%
64	34.3	8.05	8.72	101.5	21.2	24.53	+196%	+163%	+181%
128	67.2	11.1	11.7	206.4	31.02	33.67	+207%	+179%	+188%

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

Bits	TR (1)			DWC+TR (2)			Acréscimo de atraso [(2) / (1) - 1] * 100		
	RCA (ns)	RIC (ns)	CSA (ns)	RCA (ns)	RIC (ns)	CSA (ns)	RCA	RIC	CSA
4	12.09	15.94	14.79	10.72	12.58	11.54	-11%	-21%	-22%
8	19.54	18.06	17.32	15.19	14.89	12.19	-22%	-18%	-30%
16	34.94	18.87	19.83	27.9	17.01	17.53	-20%	-10%	-12%
32	67.2	22.67	24.44	53.51	18.08	19.57	-20%	-20%	-20%
64	132.8	25.63	27.36	101.5	21.2	24.53	-24%	-17%	-10%
128	263.6	38.8	39.41	206.4	31.02	33.67	-22%	-20%	-15%

De posse dos resultados de atrasos das arquiteturas de somadores protegidas apresentados nas duas tabelas anteriores, percebe-se que, conforme já era esperado, os somadores protegidos com a técnica DWC+TR apareceram em uma posição intermediária, quando comparados aos somadores protegidos com as técnicas TMR e TR. Tal conclusão pode ser derivada pela observação das últimas 3 colunas da tab.11 e tab.12. Os somadores protegidos com DWC+TR apresentaram em todos os casos, um acréscimo em seus atrasos quando comparados aos somadores protegidos com TMR, conforme a tab.11. Já em relação aos somadores protegidos com TR, as arquiteturas que utilizaram DWC+TR apresentaram um melhor desempenho, este último pode ser observado na tab.12.

A tab.11 permite uma comparação direta entre os somadores protegidos com TMR e somadores protegidos com DWC+TR, tomando estes últimos como referência. Percebe-se na tab.11 que, em se tratando dos RCAs, a arquitetura que apresentou o maior acréscimo do atraso foi o RCA de 4 bits: cerca de 231%. Já em relação aos somadores rápidos, o RIC com 16 bits e o CSA com 128 bits foram as arquiteturas que apresentaram as maiores diferenças de atrasos: cerca de 201% e 188%, respectivamente.

A tab.12 permite uma comparação direta entre os somadores protegidos com TMR e somadores protegidos com DWC+TR, tomando estes últimos como referência. Observa-se nesta que o somador RCA de 64 bits, dentre todos os somadores RCA foi o

que apresentou o melhor resultado, com uma redução do atraso crítico de 24%. Já o RIC apresentou melhoras no desempenho que variaram entre 10% e 21% para arquiteturas com 16 e 4 bits. Por fim, com relação ao somador CSA, a arquitetura com 8 bits foi a que apresentou melhor resultado: uma redução do atraso crítico em torno de 30%.

Uma análise quanto à utilização de recursos também foi realizada, o que não poderia ser diferente, haja vista a necessidade de investigar o acréscimo de recursos necessário para que esta técnica, DWC+TR, possa ser posta em prática.

Como não poderia ser diferente, inicialmente se faz necessário que sejam confrontadas as arquiteturas não-protegidas com as arquiteturas protegidas com DWC+TR. Note na Fig. 67 que da mesma forma com que se comportaram os somadores utilizando outras técnicas de proteção, neste caso as arquiteturas também fizeram uso de uma quantidade de recursos superior às arquiteturas não-protegidas. Da mesma forma, levando em conta somente as arquiteturas protegidas, também é possível notar que o somador RIC encontra-se novamente como uma arquitetura intermediária, quando se trata de utilização de recursos. O motivo pelo qual isso acontece é o mesmo já explicado anteriormente, haja vista que a utilização de uma outra técnica em nada influencia suas características arquiteturais internas.

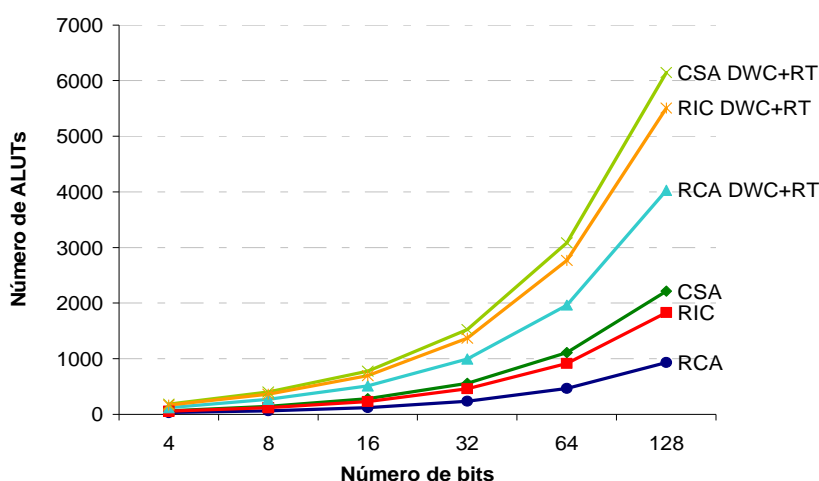


Figura 67 – Atraso crítico dos somadores não-protegidos e protegidos com DWC+TR.

Levando-se em consideração que até o presente momento já foram apresentados todos resultados com relação a desempenho dos somadores utilizando as três técnicas propostas, passemos então agora à análise comparativa da utilização de recursos (ALUTs). A Fig. 68 mostra que as arquiteturas que fizeram uso da técnica de DWC+TR apresentaram o maior uso de recursos entre todas as arquiteturas protegidas. Embora este não seja um resultado desejável quando se aplica tal técnica, este pode ser perfeitamente explicável. O acréscimo de recursos despendido para pôr esta técnica em uso, por exemplo, os circuitos multiplexadores e comparadores, juntos utilizam mais recursos que a própria arquitetura a ser protegida. Assim, embora a técnica vise apenas duplicar o bloco a ser protegido, esta não consegue obter um resultado melhor que o TMR porque a economia de um bloco logo é sobrepujada pelos recursos necessários para implementar os blocos auxiliares necessários à técnica DWC+TR.

É importante que se elucide aqui que esta técnica, embora neste caso não tenha correspondido às expectativas, quando utilizada como proteção de circuitos com maior número de componentes, tende a produzir resultados mais satisfatórios.

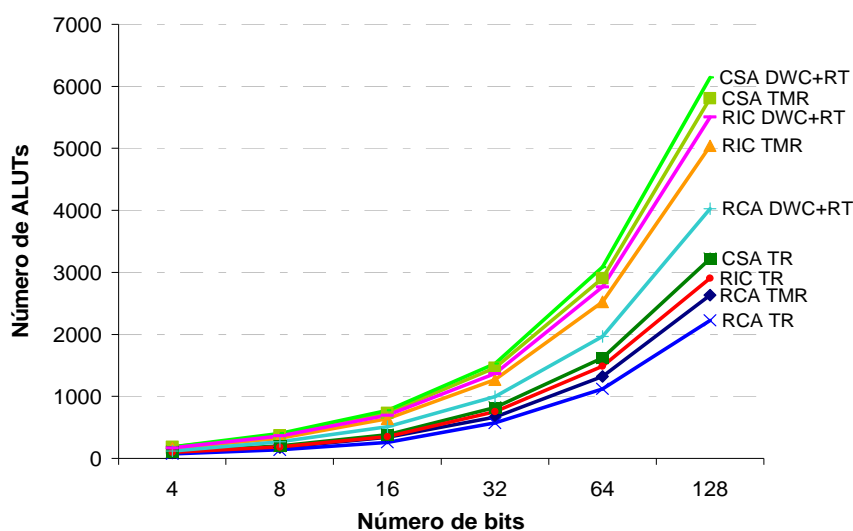


Figura 68 – Comparação entre o número de ALUTs de todos tipos de somadores protegidos com TMR, TR e DWC+TR.

Com intuito de analisar de forma mais detalhada os resultados obtidos através das arquiteturas protegidas com a técnica de DWC+TR, a tab.13 mostra os resultados,

em termos de utilização de recursos, para todas as arquiteturas de somadores não-protegidas e protegidas DWC+TR. A tab.13 faz referência também ao acréscimo de recursos necessário para que a técnica possa ser posta em prática.

Tabela 13 - Comparação entre as arquiteturas de somadores protegidos com DWC+TR e não-protegidos, quanto à utilização de ALUTs.

Bits	Não-protegidos (1)			DWC+TR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	33	55	65	120	167	183	+264%	+204%	+182%
8	62	118	142	269	361	401	+334%	+206%	+182%
16	120	232	280	511	695	775	+326%	+200%	+177%
32	236	460	556	995	1367	1527	+322%	+197%	+175%
64	468	918	1110	1965	2766	3086	+320%	+201%	+178%
128	932	1832	2216	4028	5506	6147	+332%	+201%	+177%

Da mesma maneira já evidenciada anteriormente quando se utiliza uma técnica de proteção, a técnica DWC+TR exige um acréscimo de hardware significativo. Tal comportamento pode ser observado nas 3 últimas colunas da tab.13. Em se tratando especificamente do RCA, observa-se também que este foi o somador que necessitou maior demanda percentual de recursos extras. No pior caso, RCA de 8 bits, esta demanda ficou em torno de 334%. Com relação ao somador RIC, a demanda de recursos extras variou entre 197% e 206%, valores estes referentes aos somadores RIC com 32 e 8 bits, respectivamente. Por fim, os somadores CSA, os maiores dentre todas arquiteturas de somadores experimentadas, foram as arquiteturas que apresentaram os menores acréscimos percentuais, quanto a recursos extras. Dentre elas, as arquiteturas que apresentaram os piores resultados foram os CSA com 4 e 8 bits, necessitando ambos de 182% de recursos extras.

Embora os resultados apresentados pelas arquiteturas protegidas com DWC+TR não sejam em sua totalidade satisfatórios, é necessário que se perceba uma tendência explícita nos resultados da tab.13. À medida que a arquitetura a ser protegida aumenta de tamanho, o acréscimo percentual de hardware necessário para pôr esta técnica em prática diminui. Isso fica evidenciado quando se analisam as 3 últimas

colunas da tab.13. O somador RCA necessita em média de 316% de recursos extras. Já o RIC, um somador com características de tamanho intermediárias em relação aos somadores RCA e CSA, necessita em média 202% de recursos extras. E por fim, o CSA, o maior dos somadores dentre todos do universo experimentado neste trabalho, necessita em média 179% de recursos extras.

Com base nos fatos citados no parágrafo anterior, é possível reafirmar de forma mais convicta o que foi dito anteriormente quanto à possibilidade da técnica DWC+TR possuir uma melhor eficácia em termos de utilização de recursos, quando utilizada em arquiteturas que necessitam maiores quantidades de ALUTs. Esta característica pode ser mais bem observada na Fig. 69, onde são apresentados os resultados, em termos de utilização média de recursos extras, de todas arquiteturas de somadores quando se aplicam as técnicas TMR e DWC+TR.

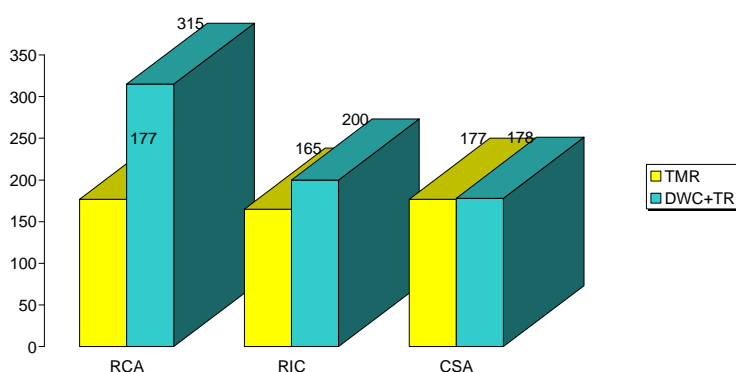


Figura 69 – Número médio de ALUTs demandado pelos somadores, dada a aplicação das técnicas TMR e DWC+TR.

Note que para todas as arquiteturas, o acréscimo de recursos exigido pela técnica de TMR tende a permanecer estável, independente de arquitetura. Já a tendência do uso de ALUTs dos somadores protegidos com DWC+TR segue o que foi dito no parágrafo anterior.

Para que sejam tiradas conclusões mais convincentes com relação às peculiaridades impostas por cada técnica aplicada aos somadores, as tab.14 e tab.15 apresentam resultados comparativos entre os somadores protegidos com DWC+TR e TMR e entre as arquiteturas de somadores protegidas com DWC+TR e TR. Da mesma

forma com que foram apresentados os resultados nas tabelas anteriores, esta também faz referência em percentagem das diferenças com relação à necessidade de utilização de recursos para as técnicas em questão.

Tabela 14 - Comparação entre as arquiteturas de somadores protegidos com TMR e DWC+TR, quanto à utilização de ALUTs.

Bits	TMR (1)			DWC+TR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	87	160	184	120	167	183	+38%	+4%	-1%
8	173	323	371	269	361	401	+55%	+12%	+8%
16	337	637	733	511	695	775	+52%	+9%	+6%
32	665	1269	1461	995	1367	1527	+50%	+8%	+5%
64	1321	2525	2909	1965	2766	3086	+49%	+10%	+6%
128	2633	5042	5809	4028	5506	6147	+53%	+9%	+6%

Tabela 15 - Comparação entre as arquiteturas de somadores protegidos com TR e DWC+TR, quanto à utilização de ALUTs.

Bits	TR (1)			DWC+TR (2)			Acréscimo de ALUTs [(2) / (1) - 1] * 100		
	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA (ALUTs)	RIC (ALUTs)	CSA (ALUTs)	RCA	RIC	CSA
4	72	96	104	120	167	183	+67%	+74%	+76%
8	137	183	199	269	361	401	+96%	+97%	+102%
16	259	351	382	511	695	775	+97%	+98%	+103%
32	570	756	820	995	1367	1527	+75%	+81%	+86%
64	1122	1489	1618	1965	2766	3086	+75%	+86%	+91%
128	2226	2905	3223	4028	5506	6147	+81%	+90%	+91%

Diante dos resultados apresentados pelas duas tabelas anteriores, é fácil perceber que a aplicação da técnica DWC+TR aos somadores é aquela que demanda mais recursos extras dentre todas as técnicas apresentadas neste trabalho. Analisando-se as duas tabelas, nota-se que em todos casos, com exceção do somador CSA de 4

bits mostrado na tab.14, as arquiteturas protegidas com DWC+TR utilizaram mais recursos quando comparadas às arquiteturas que fizeram uso das outras duas técnicas.

Em relação à tab.14, fica ainda mais clara idéia de que quanto maiores as arquiteturas a serem protegidas pela técnica DWC+TR, maior tende a ser a sua eficiência em termos uso de recursos. Considerando-se todas as arquiteturas de somadores deste trabalho protegidas com TMR e comparando-as a estes somadores protegidos com DWC+TR, percebe-se na tab.14 que o RCA utiliza em média 50% mais recursos. Já RIC utiliza em média 9% mais recursos, enquanto que o CSA gasta em média 5% mais recursos.

Em se tratando da tab.15, todos resultados ficaram dentro do esperado. Considerando-se que as arquiteturas que utilizam a técnica TR fazem uso apenas de um exemplar do bloco a ser protegido, é natural que estas utilizem menos recursos quando comparadas com as arquiteturas protegidas com DWC+TR. Isto pode ser confirmado observando-se as 3 últimas colunas da tab.15. Esta toma como referência os somadores protegidos com RT e os compara com os somadores protegidos com DWC+TR. Neste sentido, o RCA apresentou uma demanda de recursos no pior caso (RCA 16 bits) de 97%. Seguindo esta mesma linha de raciocínio, os somadores RIC apresentaram em média 88% mais recursos que quando protegidos com TR. Já o CSA demandou 103% mais recursos no pior caso (CSA de 16 bits).

Analisando-se de uma maneira geral todos os resultados gerados a partir da proteção dos somadores, percebeu que, com exceção dos resultados em termos de uso de ALUTs providos pela aplicação da técnica DWC+TR, todos eles apresentaram valores satisfatórios.

6.6 Análise da Proteção dos Somadores Contra SETs

A fim de validar as arquiteturas de somadores protegidas contra SETs propostas neste trabalho, foram realizadas campanhas de injeção de falhas por meio de simulação em nível lógico utilizando a ferramenta ModelSim da Mentor Graphics (2006), versão para Altera.

O método de injeção e simulação de falhas consiste em injetar uma falha por vez na lógica combinacional e analisar sua propagação através do circuito utilizando simulação de nível lógico com atrasos. A simulação deve levar em conta todas as possíveis combinações de vetores de entrada para cada falha analisada. A modelagem de uma falha é realizada utilizando-se o simulador da ferramenta ModelSim. Este permite que um sinal interno possa ser invertido (*forcing*) durante uma quantidade de tempo que corresponde à duração de um SET. O algoritmo para injeção das falhas utilizado neste trabalho pode ser observado abaixo.

```

1      Para cada falha
2          Para cada vetor de entrada
3              Simula o circuito com falha
4              Compara o resultado (circuito sem falha)
5              Se a falha propagou
6                  Incrementa a lista de falhas
7          Novo vetor de entrada
8      Nova falha

```

Para que pudesse ser realizada uma quantidade significativa de injeções de falhas nas arquiteturas propostas, foram desenvolvidos três *scripts* em TCL (OUSTERHOUT, 1994) a fim de automatizar o processo de injeção e simulação destas falhas. São eles: o Gerador de Arquivos de Simulação, o Gerador de Resultados e o Comparador. A ferramenta Quartus II permite o acesso aos sinais internos das arquiteturas mapeadas para o FPGA selecionado. Assim, o Gerador de Arquivos de Simulação, para cada combinação de um sinal interno e vetor de entrada, gera uma lista contendo os comandos de simulação utilizados pelo ModelSim. Uma vez gerados todos os arquivos de simulação, o *script* Gerador de Resultados controla a execução das simulações. Desta forma, para cada arquivo de simulação gerado, a ferramenta ModelSim, utilizando informações de atrasos extraídas pela ferramenta Quartus II, realiza uma simulação. Ao término de cada simulação é gerado um arquivo contendo os resultados da simulação. Quando todas as simulações tiverem sido concluídas, o *script* Comparador analisa os resultados gerados para uma dada falha e compara com os resultados gerados por um circuito sem falhas. Caso venha a ser detectado que alguma falha tenha se propagado até alguma saída primária, tendo sido capturada pelo

registrador de saída, um arquivo de falhas é incrementado. A Fig. 70 apresenta o fluxo completo de injeção e simulação de um circuito.

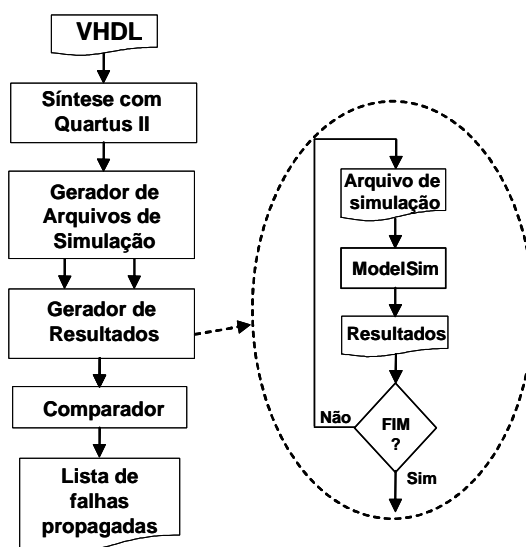


Figura 70 - Fluxo de injeção e simulação de falhas.

Considerando-se o demasiado tempo de execução necessário para que se possa analisar a robustez das arquiteturas utilizando o método acima apresentado, foram realizadas neste trabalho apenas injeções de falhas nas arquiteturas com 4 bits, conforme pode ser observado na tab.16. Ressalta-se que não foram injetadas falhas nos circuitos votadores das arquiteturas por se constituírem sabidamente em pontos vulneráveis.

Tabela 16 – Resultados oriundos da injeção de falhas nos circuitos somadores de 4 bits.

Somador	Técnica de proteção	# pontos de injeção	# Simulações	# falhas propagadas	Tempo de execução aproximado
CSA	Não-prot.	91	23296	1936	32 horas
	DWC+RT	242	61952	0	18 dias
	TMR	210	53760	0	13 dias
	TR	94	24064	0	8 dias
RIC	Não-prot.	74	18944	2368	27 horas
	DWC+RT	442	113152	0	11 dias
	TMR	162	41472	0	120 horas
	TR	79	20224	0	60 horas
RCA	Não-prot.	26	6656	192	2 dias
	DWC+RT	178	45568	0	18 dias
	TMR	78	19968	0	28 horas
	TR	47	12032	0	7 dias

Através da tab.16 pode-se verificar que nenhuma falha aplicada às arquiteturas protegidas conseguiu propagar-se para as saídas, o que comprova a eficácia das técnicas de proteção. Percebe-se também, através da tab.16, que as arquiteturas protegidas possuem uma quantidade maior de pontos passíveis de injeção de falhas quando comparadas às arquiteturas não-protegidas. Obviamente, isto se deve a quantidade de recursos extras exigida pela aplicação das técnicas de proteção.

A última coluna da tab.16 mostra os tempos de execução para as campanhas de injeção de falhas. Estes tempos servem tão somente como referência, visto que foram utilizados computadores com configurações bastante diversas, equipados com processadores de diferentes desempenhos (AMD Athlon, 1.26 GHz; Intel Pentium IV, 3.0 GHz; Intel Core Duo, 1.87 GHz; AMD Turion 64 Mobile, 2.0 GHz). Ainda assim, é possível constatar-se que a injeção de falhas em arquiteturas com mais bits torna-se inviável. Porém, conforme foi apresentada na seção 6.1, todas arquiteturas são implementadas a partir de somadores de 4 bits. Dessa forma, considerando o grau de proteção atingido, pode-se especular que as mesmas arquiteturas protegidas com mais bits não propagarão falhas.

7 Conclusão

Este trabalho teve como objetivo a análise dos impactos, em termos de atraso crítico e recursos utilizados, na implementação de arquiteturas de somadores rápidos utilizando-se dispositivos FPGAs da Altera. Além disso, no capítulo 4 foi apresentada uma nova arquitetura de somadores rápidos, batizada de RIC - Re-computing the Inverse Carry-in, a qual apresenta características próprias de desempenho e uso de recursos. Dentre as principais características dos somadores RIC destaca-se um desempenho comparável àquele exibido pelos somadores CSA (Carry-Select Adder), porém demandando uma menor quantidade de recursos. Através dos resultados pôde-se observar que o RIC utilizou em média 17% menos recursos que o CSA. Já em relação ao RCA, o RIC, no pior caso (128 bits) utilizou 97% mais recursos. Quanto ao atraso crítico, os somadores RIC, a partir de 16 bits, apresentaram atrasos críticos menores que os apresentados pelos somadores CSA. Esta diferença variou entre 5% (para o RIC com 128 bits) e 9% (para o RIC com 16 bits).

Quanto aos somadores protegidos com TMR (Triple Module Redundancy), TR (Time Redundancy) e DWC+RT (Duplication with Comparison combined with Time Redundancy), percebeu-se, através dos resultados, que em todas as situações, o somador RIC manteve-se como uma arquitetura de características intermediárias em relação aos somadores CSA e RCA, no que diz respeito à utilização de recursos. Em relação ao atraso crítico, as arquiteturas protegidas apresentaram comportamento semelhante às arquiteturas não protegidas, ou seja, dada a aplicação das técnicas, a partir de 16 bits o RIC apresentou atrasos críticos menores que os atrasos críticos do CSA.

Percebeu, através dos resultados, que a aplicação de TMR aos somadores implica, em média, em um acréscimo de recursos utilizados de 174%. Porém, quanto ao desempenho, os somadores apresentaram em média quedas de 35%. Já a aplicação de TR aos somadores implicou, em média, um gasto extra de recursos de 79%. Além disto, a utilização de TR implicou em uma queda de desempenho dos somadores de 365%, em média. E por fim, a aplicação de DWC+RT aos somadores resultou em um acréscimo de recursos de 232%, em média, e aumento dos atrasos críticos, também em média, de 278%.

Especificamente em relação à aplicação de RT, concluiu-se também que esta, quando aplicada a arquiteturas maiores, tende a apresentar resultados ainda mais satisfatórios quanto ao uso de recursos. Quando aplicada aos RCAs, menor somador experimentado neste trabalho, o acréscimo de recursos exigido foi de 128%, em média. Já quando aplicado aos CSAs, o acréscimo de recursos foi de apenas 46%.

Comparando-se os resultados obtidos somente pelos somadores protegidos com RT e TMR, conclui-se que as arquiteturas que utilizaram RT apresentaram aumentos nos atrasos de 243% (em média) em relação as que utilizaram TMR. Porém, a quantidade de recursos exigida pela técnica TR, quando aplicada aos somadores, foi, em média, 35% menor que a exigida pela técnica TMR. Conclui-se, ainda, que a aplicação de TR a arquiteturas pouco complexas pode não ser uma boa alternativa. Tal constatação decorre da análise dos resultados dos RCAs TMR e RCAs TR. No melhor caso, a maior economia de recursos obtida pela utilização de TR em relação a TMR foi de 23% para o RCA de 16 bits. Considerando o acréscimo de atraso resultante da aplicação de RT, a utilização desta técnica não é uma boa alternativa para arquiteturas de somadores RCA.

Quanto aos somadores protegidos com DWC+RT, estes, quando comparados aos somadores protegidos com TMR e RT, apresentaram valores de atrasos críticos intermediários. Porém, quanto à quantidade de recursos utilizados, estas arquiteturas apresentaram os piores resultados. Isto se deve principalmente pela quantidade de recursos necessários para pôr esta técnica em prática. Entretanto, através dos resultados, percebeu-se que a técnica, quando aplicada a arquiteturas mais complexas, tende a apresentar resultados melhores que TMR.

Considerando-se apenas as arquiteturas de somadores experimentadas e as técnicas de proteção utilizadas neste trabalho, concluiu-se que, se o objetivo do projetista é a construção de somadores protegidos com um alto desempenho, a técnica TMR é a mais adequada. Entretanto, se o objetivo estiver focado em uma economia de recursos, TR é uma boa alternativa.

Quanto aos somadores, conclui-se que o RIC é uma boa alternativa para os projetistas, dada a necessidade da construção de um somador com alto desempenho, mas que utilize uma quantidade de recursos moderada.

Considerando-se que este é um trabalho atual e que os problemas causados por partículas carregadas tendem a aumentar devido ao constante avanço da tecnologia CMOS, faz-se necessário que a investigação iniciada por este trabalho prossiga. Como trabalhos futuros, uma análise dos resultados de desempenho e recursos utilizados se faz necessária, utilizando-se uma ferramenta de síntese profissional como, por exemplo, a Simplify-Pro da empresa Synplicity (SYMPPLICITY, 2007). Outro trabalho de investigação importante é o mapeamento dos somadores aqui considerados para outros dispositivos FPGA da Altera fabricados com tecnologia nanométrica (Stratix III, por exemplo) e mesmo para dispositivos da Xilinx (Virtex 4), de modo a verificar se as tendências observadas se mantêm. Quanto ao RIC, a implementação deste somador em nível de transistor é essencial para que se possa realizar uma comparação mais precisa com outros tipos de somadores, tanto no que diz respeito aos atrasos críticos, quanto à quantidade de recursos necessários. Neste caso, algumas variações do RIC também podem ser investigadas, como por exemplo, adotando-se somadores CLA (*Carry Lookahead*) de quatro bits no lugar nos RCAs.

8 Referências

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

ACTEL. Actel Corporation. RTSX-SU RadTolerant FPGAs (UMC). Mar. 2006. Disponível em: <<http://www.actel.com>>.

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, pp. 99-107.

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.

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.; NICOLAIDIS, M. Cost Reduction and Evaluation of a Temporary Faults Detecting Technique. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2000, Paris. **Proceedings...** Los Alamitos: IEEE Computer Society, 2000. p. 591-598.

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.; SMITH, E. Neutron Induced Boron Fission as a Major Source of Soft Errors in Deep Submicron SRAM Devices. In: IEEE INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 38., 2000. **Proceedings ...** [S.l.]: IEEE Computer Society, 2000.

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.

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.

BAZE, M.; BUCHNER, S. Attenuation of Single Event Induced Pulses in CMOS Combinational Logic. **IEEE Transactions on Nuclear Science**, Vol. 44, No. 6, December 1997.

BORKAR, S. Designing Reliable System from Unreliable Components: The Challenges of Transistor Variability and Degradation. **IEEE MICRO**, [S.l.], v.25, n.6, p.10-16, Nov.-Dec. 2005.

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.

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.

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

- 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.
- GADLAGE, M. et al. Single Event Transient Pulsewidths in Digital Microcircuits. **IEEE Transactions on Nuclear Science**, v.51, n. 6, Dec. 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.
- HAMMING, R. W. Error Detecting Error Correcting Codes. In: Bell System Technical Journal, v.26, no. 2, p. 147-160, April 1950.
- 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.I.], 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.I.]: IEEE Computer Society, 2002. p.203-204.
- JOHNSTON, A. Scaling and Technology Issues for Soft Error Rates. In: RESEARCH CONFERENCE ON RELIABILITY, 4., 2000. **Proceedings...** Palo Alto: Stanford University, 2000.
- KARNIK, T.; HAZUCHA, P.; PATEL, J. Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes. **IEEE Transaction on Dependable and Secure Computing**, [S.1.], v.1, n.2, p. 128-143, Apr.-June 2004.
- KOLASINSKI, W. et al. Simulation of cosmic-ray induced soft-errors and latchup in integrated –circuit computer memories. **IEEE Transactions on Nuclear Science**, v. 26, p. 5087-5091, Dec, 1979.
- KUMAR, K.; LALA, P.; On-line Detection of Faults in Carry-Select Adders. In: International Test Conference, pp. 912, 2003.

LABEL, K. A. et al. A Roadmap for NASA's Radiation Effects Research in Emerging Microelectronics and Photonics. In: AEROSPACE CONFERENCE, 2000. **Proceedings...** [S.1.]: IEEE, 2000. p. 535-545.

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

LERAY, J. et al. Atmospheric Neutron Effects in Advanced Microelectronics, Standards and Applications. In: INTERNATIONAL CONFERENCE ON INTEGRATED CIRCUIT DESIGN TECHNOLOGY, ICICDT, 2004. **Proceedings...** [S.1.]: IEEE, 2004. p. 311-321.

LIMA, F. **Single Event Upser Mitigation Techniques for Programmable Devices.** 2000. 102 f. Qualifying Examination (Ph.D) – PPGC, Instituto de Informática, UFRGS, Porto Alegre.

LIMA, F. **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.

MAHESHWARI, A.; KOREN, I.; BURLESON, N. Techniques for Transients Fault Sensitivity Analysis Reduction in VLSI Circuitos. In: INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 18., 2003. **Proceedings...**[S.1.]: IEEE, 2003. p. 597-604.

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

MAXFIELD, C. The Design Warrior's guide to FPGAs Devices, Tools and Flows. Amsterdam: Elsevier, 2004.

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.

NEUBERGER, G.; LIMA, F.; CARRO, L.; REIS, R. A Multiple Bit Upset Tolerant SRAM Memory. **Transactions on Design Automation of Electronic Systems**, TODAES, New York, v.8, n.4, Oct. 2003.

NEUMANN, V. J. Probabilistic logics and synthesis of reliable organism from unreliable components. Automata Studies. In: Annals of Mathematical Studies no.34 (Eds.: C. E. Shannon and J. McCarthy), 43-98. Princeton University Press, 1956.

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.

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.

NIEUWLAND, A.; JASAREVIC, S.; JERIN, G. Combinational Logic Soft Error analysis and Protection. In: 12TH IEEE INTERNATIONAL ON-LINE TESTING SYMPOSIUM (IOLTS'06), pp. 99-104, 2006.

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.

O'BRYAN, M. et al. Single Event Effect and Radiation Damage Results for Candidate Spacecraft Electronics. In: RADIATION EFFECTS DATA WORKSHOP, 1998. **Proceedings...**[S.1.]: IEEE, 1998. p. 39-50.

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.I.]: 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., PAPASSO, G., ROSSI, D., METRA, C., 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.

PORTO, Roger Endrigo Carvalho. Estudo e Desenvolvimento de Arquiteturas de Somadores para Uso na Compressão de Imagens JPEG. Pelotas: Bacharelado em Ciência da Computação da UFPel, maio 2003. 48p. Trabalho de Conclusão de Curso.

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

ROOSTA, R. A Comparison of Radiation-Hard and Radiation - Tolerant FPGAs for Space Applications. NASA Electronic Parts and Packaging Program. Dec. 2004.

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.

SYMPPLICITY. Synplicity Corporate. Disponível em:

<<http://www.synplicity.com/products/synplifypro/>>. Acesso em Jan. 2007.

TOSAKA, Y. Measurement and Analysis Neutron-Induced Soft Errors in SUB-Half-Micro CMOS circuits. **IEEE Transactions on Electron Device**, [S1.1], v.45, n.7, p. 1453-1458, July 1998.

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.

XILINX, INC. Virtex®™ 2.5 V Field Programmable Gate Arrays: Datasheet DS003. USA, 2000.

WIRTH, G. et al. Single Event Transients in Combinatorial Circuits. In: 16th INTERNATIONAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 2005, Florianópolis. **Proceedings...** New York (USA): ACM: Association for Computing Machinery, 2005. p. 121-126.

MAXFIELD, C. The Design Warrior's guide to FPGAs Devices, Tools and Flows. Amsterdam: Elsevier, 2004.

ANEXO A – Código VHDL de um RCA de 4 bits Utilizando as Diretivas de Compilação.

```
library ieee;
use ieee.std_logic_1164.all;

-- SOMADOR RIPPLE CARRY --
-- DE 4 BITS COM CARRY OUT --
-- Ciência da Computação - UFPel --

entity add4c is
  port (
    cin      : in std_logic;
    input0   : in std_logic_vector(3 downto 0);
    input1   : in std_logic_vector(3 downto 0);
    output   : out std_logic_vector(3 downto 0);
    cout     : out std_logic
  );
end add4c;

architecture behavior of add4c is

  attribute syn_keep : boolean;

  signal carry0 : std_logic;
  attribute syn_keep of carry0 : signal is true;
  signal carry1 : std_logic;
  attribute syn_keep of carry1 : signal is true;
  signal carry2 : std_logic;
  attribute syn_keep of carry2 : signal is true;

  attribute syn_keep of output : signal is true;
  attribute syn_keep of cout : signal is true;

  attribute syn_keep of input0 : signal is true;
  attribute syn_keep of input1 : signal is true;
  attribute syn_keep of cin : signal is true;

begin

  carry0 <= (cin and input0(0)) or (cin and input1(0)) or (input1(0) and input0(0));
  carry1 <= (carry0 and input0(1)) or (carry0 and input1(1)) or (input1(1) and input0(1));
  carry2 <= (carry1 and input0(2)) or (carry1 and input1(2)) or (input1(2) and input0(2));
  cout  <= (carry2 and input0(3)) or (carry2 and input1(3)) or (input1(3) and input0(3));

  output(0) <= input0(0) xor input1(0) xor cin;
  output(1) <= input0(1) xor input1(1) xor carry0;
  output(2) <= input0(2) xor input1(2) xor carry1;
  output(3) <= input0(3) xor input1(3) xor carry2;

end behavior;
```