MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2018/03.06.14.06-TDI

MODEL CHECKING PROBABILÍSTICO PARA APOIAR A MITIGAÇÃO DE EVENTO DE FALTA ÚNICA EM FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

Viny Cesar Pereira

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Valdivino Alexandre de Santiago Júnior, aprovada em 28 de março de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3QLQMU2>

INPE
São José dos Campos
2018

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2018/03.06.14.06-TDI

MODEL CHECKING PROBABILÍSTICO PARA APOIAR A MITIGAÇÃO DE EVENTO DE FALTA ÚNICA EM FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

Viny Cesar Pereira

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Valdivino Alexandre de Santiago Júnior, aprovada em 28 de março de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34P/3QLQMU2>

INPE
São José dos Campos
2018

Dados Internacionais de Catalogação na Publicação (CIP)

Pereira, Viny Cesar.

P414m Model checking probabilístico para apoiar a mitigação de evento de falta única em Field Programmable Gate Arrays (FPGAs) / Viny Cesar Pereira. – São José dos Campos : INPE, 2018.

xx + 113 p. ; (sid.inpe.br/mtc-m21b/2018/03.06.14.06-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2018.

Orientador : Dr. Valdivino Alexandre de Santiago Júnior.

1. Verificação formal. 2. Model checking probabilístico. 3. Single Event Upset. 4. Field Programmable Gate Array. I.Título.

CDU 004.052.42

Aluno (a): *Viny Cesar Pereira*

Título: "MODEL CHECKING PROBABILÍSTICO PARA APOIAR A MITIGAÇÃO DE EVENTO DE FALTA ÚNICA EM FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)".

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em *Computação Aplicada*

Dr. Reinaldo Roberto Rosa

Presidente / INPE / SJCampos - SP

( ) Participação por Video - Conferência

Dr. Valdivino Alexandre de Santiago Júnior

Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Silvio Manea

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Roberto d' Amore

Convidado(a) / ITA / São José dos Campos - SP

( ) Participação por Video - Conferência

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 28 de março de 2018

AGRADECIMENTOS

Agradeço primeiramente à Deus, por me guiar no caminho correto e por nunca me desamparar. Aos meus pais, Agnaldo e Graça, pela educação que recebi, por me incentivar sempre nas minhas decisões e pelo amor incondicional. À toda minha família que, apesar das desavenças, torcem pelo meu sucesso e confiam no meu esforço. À minha namorada, Ana Ercilia, por me acompanhar durante todo o mestrado, suportando minhas reclamações e chatices, me dando carinho, atenção e amor. Amo absurdamente todos vocês, e sei que cada um tem uma parcela de contribuição nessa etapa que está se encerrando. Por fim, agradeço também ao CNPq pelo auxílio financeiro, ao meu orientador Valdivino e ao INPE.

RESUMO

O uso de dispositivos lógicos programáveis como *Field Programmable Gate Arrays* (FPGAs) em aplicações espaciais cresceu fortemente nos últimos anos devido à sua flexibilidade, custo de desenvolvimento e desempenho. Porém, existe uma exposição excessiva aos raios cósmicos presentes no ambiente e os efeitos da radiação podem causar erros transientes, quando partículas carregadas atingem a superfície dos componentes. Tais efeitos são chamados de *Single Event Effects* (SEEs), que, em sua forma não destrutiva conhecida como Evento de Falta Única (*Single Event Upset* (SEU)), pode atingir as células de memória e causar uma inversão no valor lógico armazenado, ou seja, um *bit flip*. Diversas técnicas de detecção, mitigação e correção de SEU surgiram no passado como forma de evitar falhas, mas a maioria dos testes presentes na literatura foram conduzidos após implementar as técnicas em FPGAs e simular os *upsets* com ferramentas de injeção de falhas, o que pode ser uma abordagem custosa, já que os resultados só são apresentados ao final das simulações. Por outro lado, modelos estocásticos/probabilísticos podem ser utilizados nos estágios iniciais de um projeto sem a necessidade de implementação, com grande potencial para amortizar o custo do projeto como um todo. Um dos métodos que lida com sistemas de comportamento estocástico é o *Model Checking* Probabilístico, o qual é capaz de garantir, de acordo com uma probabilidade especificada, a corretude do sistema. Essa dissertação de mestrado investiga a viabilidade, no contexto de aplicações espaciais, da utilização de *Model Checking* Probabilístico para determinar, dentre um conjunto de soluções, qual seria a melhor técnica de mitigação de SEU em FPGAs SRAM. Para isso, as técnicas de Scrubbing, TMR e código de Hamming foram modeladas via *Model Checking* Probabilístico com a ferramenta PRISM. Os modelos foram comparados em dois estudos de caso, com características de órbita e tempos de missão distintos, considerando uma Xilinx Virtex-5 em ambos os tipos de ionização, direta e indireta. Considerando três atributos de dependabilidade, disponibilidade, segurança e confiabilidade, as técnicas também foram implementadas e simuladas via ModelSim para obter comparações com os modelos probabilísticos desenvolvidos. As análises dos modelos mostram que, em termos de disponibilidade, o código de Hamming apresentou o melhor desempenho mantendo o sistema por mais tempo em modo operacional mesmo com a pior taxa de falhas. Na confiabilidade, o que mais afetou o Scrubbing foi o tamanho do intervalo entre as correções enquanto a segurança está mais relacionada com a taxa de cobertura. O TMR apresentou os piores resultados pois permite que os *upsets* se acumulem e deve ser combinado com alguma técnica de correção, como o código de Hamming ou Scrubbing. Já os resultados da comparação com a simulação funcional mostram que as condições de órbita e tempos de missão são fatores impactantes na precisão dos modelos e devem ser considerados. Por fim, é possível confirmar a viabilidade do uso de *Model Checking* Probabilístico pois, na maioria dos casos, tal abordagem apresentou resultados próximos aos obtidos com simulação funcional.

Palavras-chave: Verificação Formal. *Model Checking* Probabilístico. *Single Event Upset*. *Field Programmable Gate Array*.

PROBABILISTIC MODEL CHECKING TO SUPPORT THE MITIGATION OF SINGLE EVENT UPSET IN FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

ABSTRACT

The use of programmable logic devices such as Field Programmable Gate Arrays (FPGAs) in space applications has grown strongly in recent years due to its flexibility, development cost and performance. However, there is excessive exposure to the cosmic rays present in the environment and the effects of radiation can cause transient errors, when charged particles reach the surface of the components. These effects are called Single Event Effects (SEEs), which in their non-destructive form known as Single Event Upset (SEU) can reach the memory cells and cause a reversal in the stored logical value, i.e. a bit flip. Several techniques of detection, mitigation and correction of SEU have appeared in the past as a way of avoiding failures, but most of the tests in the literature were conducted after implementing the techniques in FPGAs and simulate upsets with fault injection tools, which can be a costly approach, since the results are only presented at the end of the simulations. On the other hand, stochastic/probabilistic models can be used in the early stages of design without the need of implementation, with great potential to amortize the cost of the design as a whole. One of the methods that deals with stochastic behavior systems is the Probabilistic Model Checking, which is able to guarantee, within a specified probability, the correctness of the system. This master's dissertation investigates the feasibility, in the context of space applications, the use of Probabilistic Model Checking to determine, among a set of solutions, what would be the best SEU mitigation technique in SRAM FPGAs. For this, the Scrubbing, TMR and Hamming code techniques were modeled using Probabilistic Model Checking within the PRISM tool. The models were compared in two case studies, with distinct orbit characteristics and mission times, considering a Xilinx Virtex-5 in both direct and indirect types of ionization. Considering three attributes of dependability, availability, safety and reliability, the techniques were also implemented and simulated via ModelSim to obtain comparisons with the developed probabilistic models. The analyzes of the models show that in terms of availability, the Hamming code presented the best performance by keeping the system longer in operational mode even with the worst failure rate. In reliability, what most affected Scrubbing was the size of the interval between corrections while safety is more related to the coverage rate. TMR showed the worst results because it allows upsets to accumulate and must be combined with some correction technique such as Hamming code or Scrubbing. The results of the comparison with the functional simulation show that the orbit conditions and mission times are impacting factors on the accuracy of models and should be considered. Finally, it is possible to confirm the feasibility of using Probabilistic Model Checking because, in most cases, this approach showed similar results to those obtained with functional simulation.

Keywords: Formal Verification. Probabilistic Model Checking. Single Event Upset. Field Programmable Gate Array.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

ASIC	–	*Application Specific Integrated Circuits*
BCH	–	Bose-Chaudhuri-Hocquenghem
BRAM	–	*Block RAM*
CITAR	–	Circuitos Integrados Tolerantes à Radiação
CLB	–	*Configurable Logic Block*
COTS	–	*Commercial Off-The-Shelf*
CREME	–	*Cosmic Ray Effects on Microelectronics*
CSL	–	*Continuous Stochastic Logic*
CTL	–	*Computation Tree Logic*
CTMC	–	*Continuous-Time Markov Chain*
DSCC	–	*Difference-Set Cyclic Code*
DSP	–	*Digital Signal Processor*
DTMC	–	*Discrete-Time Markov Chain*
ECC	–	*Error Correction Code*
ECSS	–	*European Cooperation for Space Standardization*
FPGA	–	*Field Programmable Gate Array*
HDL	–	*Hardware Description Language*
HEO	–	*Highly Elliptical Orbit*
ICAP	–	*Internal Configuration Access Port*
IOB	–	*Input/Output Block*
LET	–	*Linear Energy Transfer*
LTL	–	*Linear Temporal Logic*
LUT	–	*Look-Up Tables*
MBU	–	*Multiple Bit Upset*
MDP	–	*Markov Decision Process*
MTBF	–	*Mean Time Between Failure*
PCTL	–	*Probabilistic Computation Tree Logic*
PTA	–	*Probabilistic Timed Automata*
SAA	–	*South Atlantic Anomaly*
RTL	–	*Register Transfer Level*
SBU	–	*Single Bit Upset*
SEB	–	*Single Event Burnout*
SEE	–	*Single Event Effect*
SEFI	–	*Single Event Function Interrupt*
SEGR	–	*Single Event Gate Rupture*
SEL	–	*Single Event Latch-Up*
SET	–	*Single Event Transient*
SEU	–	*Single Event Upset*
SRAM	–	*Static Random Access Memory*
TMR	–	*Triple Modular Redundancy*

TS – *Transition System*
VANT – Veículo Aéreo Não Tripulado

SUMÁRIO

Pág.

1 INTRODUÇÃO

Garantir segurança, confiabilidade e disponibilidade em sistemas complexos tem sido um grande desafio para desenvolvedores, especialmente em aplicações onde falhas de hardware e software podem causar danos científicos e financeiros catastróficos (CLARKE; WING, 1996). Para circuitos eletrônicos utilizados em missões espaciais, existe uma exposição excessiva aos raios cósmicos presentes no ambiente e os efeitos da radiação podem causar faltas transientes quando partículas carregadas atingem a superfície do componente (STASSINOPOULOS; RAYMOND, 1988). Tais efeitos são chamados de Efeitos de Evento Único (*Single Event Effects* - SEEs), que, em sua forma não destrutiva é conhecido como **Evento de Falta Única** (*Single Event Upset* - SEU), e pode atingir as células de memória e causar uma inversão no valor lógico armazenado, ou seja, um *bit flip* (KASTENSMIDT, 2007). Na maioria dos casos, apenas uma célula de memória é afetada pelo pulso transiente, mas é possível que uma partícula energizada estimule duas ou mais células de memória adjacentes, causando um Evento de Falta em Múltiplos Bits (*Multiple Bit Upset* - MBU) (NEUBERGER et al., 2003).

O uso de dispositivos lógicos programáveis, tais como *Field Programmable Gate Arrays* (FPGAs), em aplicações espaciais cresceu fortemente nos últimos anos graças à sua flexibilidade, custo de desenvolvimento e desempenho frente a outras tecnologias, tais como Circuitos Integrados de Aplicação Específica (*Application Specific Integrated Circuits* - ASICs) (WANG et al., 1999). A seção A.1 do Apêndice A descreve o avanço tecnológico desses dispositivos. Dentre os modelos de FPGAs, as que mais se destacam são as baseadas em Memória Estática de Acesso Aleatório (*Static Random Access Memory* - SRAM) que, além de todas as vantagens citadas anteriormente, ainda permitem a reprogramação mesmo depois de operacionais. No entanto, FPGAs SRAM são sensíveis a radiação, e o grande desafio tem sido desenvolver técnicas para mitigar as consequências dos SEUs na memória de configuração e nos dados de programação das FPGAs (GRAHAM et al., 2003). Portanto, esse trabalho está no contexto de ocorrência de SEU em ambas as camadas estruturais das FPGAs de memória SRAM utilizadas em aplicações espaciais.

Técnicas que aumentam a imunidade das FPGAs contra os efeitos da radiação podem ser implementadas em diferentes níveis de abstração. Existem modelos de FPGAs chamados de *radiation hardened* (REBAUDENGO et al., 2002) que possuem uma resistência maior aos *upsets* devido ao processo de fabricação diferenciado dos componentes. Contudo, além da capacidade computacional reduzida em comparação a modelos

Commercial Off-The-Shelf (COTS), o preço é significativamente maior. Por outro lado, técnicas aplicadas no nível lógico podem ser implementadas em linguagens de descrição de hardware (*Hardware Description Languages* - HDLs) (KASTENSMIDT, 2007) e ainda garantir a confiabilidade quando se trata de tolerância a falhas.

1.1 Motivação

Diversas técnicas de detecção, mitigação e correção de SEU surgiram no passado como forma de evitar falhas em sistemas espaciais baseados em FPGAs (HEINER et al., 2009; HENTSCHKE et al., 2002; LIU et al., 2012; MORGAN et al., 2007). Nos últimos anos, pesquisadores propuseram melhorias para adaptar estas técnicas com o avanço no processo de fabricação dos circuitos integrados. As abordagens mais comuns envolvem a redundância de hardware, como a Redundância Modular Tripla (*Triple Modular Redundancy* - TMR) (ROLLINS et al., 2003) que consiste em triplicar os módulos do circuito que apresentem maior sensibilidade aos *upsets* e, por meio de um mecanismo de votação, decidir a saída correta baseado na maioria. Existem outras formas de redundância, como a redundância temporal (ELAKKUMANAN et al., 2006) que mascara a ocorrência de faltas pois obtém o sinal em diferentes instantes de tempo, salva os valores obtidos e, como no TMR, usa um votador para decidir qual saída é a correta.

Infelizmente, técnicas de redundância não são capazes de lidar com o acúmulo de *upsets* pois mascaram a falta mas não a corrigem de fato. Portanto, é necessário mecanismos que possam reconfigurar a FPGA nesses casos (OSTLER et al., 2009). Uma forma é utilizar técnicas de *Scrubbing* (BERG et al., 2008) para reescrever partes da memória que foram afetadas, corrigindo os *upsets*.

Outra categoria de técnicas conhecida como Código de Correção de Erro (*Error Correction Code* - ECC) é baseada na adição de informação (bits de paridade) junto aos dados de entrada para detectar e corrigir possíveis faltas (DUTTA; TOUBA, 2007). Existem diversos ECCs utilizados no contexto de SEUs tais como o código de Hamming (KUMAR; UMASHANKAR, 2007), código Reed-Solomon (ALMEIDA et al., 2007), e código Bose-Chaudhuri-Hocquenghem (BCH) (IONESCU et al., 2010). Verificando os bits de paridade, estas técnicas são capazes de reverter faltas de um bit e, em algumas implementações mais complexas, corrigir também faltas de dois ou mais bits (KASTENSMIDT, 2007).

Análises comparativas de técnicas de mitigação de SEU têm sido publicadas recentemente (SHULER et al., 2009; MORGAN et al., 2007; LIU et al., 2012; HENTSCHKE et

al., 2002). Muitos fatores devem ser considerados antes de decidir por uma técnica em particular ou até mesmo decidir pela combinação das técnicas. Alguns aspectos importantes como a área adicional requerida, impacto no desempenho, habilidade de corrigir faltas ou simplesmente mitigá-las, robustez contra *upsets* múltiplos podem interferir na decisão de qual técnica é mais interessante utilizar. Estudos como (EJLALI et al., 2006; SARI et al., 2013) sugerem combinar técnicas para obter melhores resultados, no entanto, garantir desempenho, confiabilidade e segurança pode se tornar um desafio dada a complexidade do problema. Para validar estas técnicas, a maioria dos trabalhos utilizam simulações e testes após projetar e implementar em FPGA, além de utilizar ferramentas de injeção de falhas para simular os *upsets* e, em seguida, avaliar o comportamento das técnicas. Estas abordagens não são capazes de avaliar de forma antecipada, durante o processo de desenvolvimento, todas as possíveis situações e podem deixar passar defeitos que serão custosos para detectar e corrigir posteriormente. A motivação desse trabalho se sustenta na investigação da possibilidade de, por meio de uma abordagem probabilística, obter indícios de quais técnicas apresentam os melhores resultados dentro das fases iniciais de um projeto. Em (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION., 2009) são estabelecidas sete fases de projeto padronizadas para aplicações espaciais, onde as três fases iniciais envolvem análise de missão/identificação de necessidades, viabilidade e definições preliminares. Obter resultados confiáveis sobre o uso de determinadas técnicas de mitigação de SEU em estágios precoces do processo de desenvolvimento de uma aplicação espacial é vantajoso para o desenvolvimento da missão como um todo.

Métodos de Verificação Formal (BAIER et al., 2008) têm sido utilizados em diversas aplicações em diferentes domínios. Em particular, *Model Checking* Probabilístico (KWIATKOWSKA et al., 2009) tem se mostrado uma técnica eficiente e robusta para modelagem de uma grande variedade de problemas em campos variados tais como na biologia, segurança, protocolos de comunicação e de rede, desempenho e confiabilidade entre outros. Modelos probabilísticos tais como Cadeias de Markov em Tempo Discreto (*Discrete-Time Markov Chain* - DTMC), Cadeias de Markov em Tempo Contínuo (*Continuous-Time Markov Chain* - CTMC) e Processos de Decisão Markovianos (*Markov Decision Process* - MDP) têm sido empregados nesse contexto.

Em (HOQUE et al., 2014; HOQUE, 2016), os autores analisaram a dependabilidade dos métodos de mitigação de SEU em FPGAs SRAM via *Model Checking* Probabilístico. Entretanto, a abordagem apresentada pelos autores lida com uma combinação

de duas técnicas (TMR e *Scrubbing*) que, quando comparadas, são colocadas em apenas duas situações: apenas *Scrubbing* e TMR com *Scrubbing*. Além de não ser considerado independentemente, o TMR foi analisado de uma maneira simplificada, apenas como componentes sobressalentes, ignorando detalhes importantes como o mecanismo de votação e os dados de entrada e saída. Além disso, os autores consideraram projetos com apenas dois somadores e dois multiplicadores. Por fim, código de Hamming, uma técnica bastante popular de correção de faltas, não foi considerado. Portanto, esse trabalho investiga com mais profundidade a viabilidade do uso de *Model Checking* Probabilístico como mecanismo de apoio de comparação de técnicas de mitigação de SEU em FPGAs.

Por fim, é importante mencionar a iniciativa do Instituto Nacional de Pesquisas Espaciais (INPE) no contexto de mitigar SEE/SEU em FPGAs. Atualmente, o INPE desenvolve o projeto Circuitos Integrados Tolerantes à Radiação (CITAR) que visa amadurecer a tecnologia de endurecimento à radiação de componentes eletrônicos e preparar a infraestrutura de testes de radiação no Brasil, promovendo a independência tecnológica nessa área de conhecimento (INPE, 2016).

1.2 Objetivo e Metodologia de Pesquisa

O objetivo dessa dissertação de mestrado é investigar a viabilidade, no contexto de aplicações espaciais, da utilização de *Model Checking* Probabilístico para determinar qual seria, dentre um conjunto de soluções, a melhor técnica de mitigação de SEU em FPGAs SRAM.

Para alcançar esse objetivo, inicialmente as técnicas de mitigação de SEU *Scrubbing* (BERG et al., 2008), TMR (KASTENSMIDT et al., 2005) e código de Hamming (KUMAR; UMASHANKAR, 2007) foram estudadas e modeladas via *Model Checking* Probabilístico e a ferramenta PRISM (KWIATKOWSKA et al., 2009). Os modelos foram comparados em dois estudos de caso, o primeiro considerando uma aplicação em Órbita Elíptica Alta (OEA) e um tempo de missão de 90 dias, e o segundo utilizando os dados orbitais do satélite CBERS-4A e um tempo de missão de 650 dias. A análise considerou três atributos de dependabilidade: disponibilidade, segurança e confiabilidade (AVIZIENIS et al., 2004). O modelo escolhido foi CTMC e as propriedades foram formalizadas via Lógica Estocástica Contínua (*Continuous Stochastic Logic* - CSL (BAIER et al., 2003)) estendida com Recompensas (*Rewards*).

Em seguida, para analisar a viabilidade da utilização dos modelos probabilísticos para mitigação de SEUs na memória de programação de FPGAs SRAM, as mes-

mas três técnicas foram implementadas em VHDL no ambiente Altera Quartus II (ALTERA, 2007) e simuladas por meio do Mentor Graphics ModelSim (MENTOR GRAPHICS,) para ambos os estudos de caso, a seção A.2 do Apêndice A descreve as ferramentas de software utilizadas nesse trabalho. Nesta fase, foram analisadas a disponibilidade e a confiabilidade das técnicas mantendo os mesmos critérios da análise probabilística desenvolvida anteriormente nesse trabalho. Por fim, foi possível comparar os resultados obtidos via *Model Checking* Probabilístico e aqueles resultantes da implementação e simulação para decidir qual técnica se adequou melhor em cada estudo de caso, e se é viável apoiar-se nos resultados probabilísticos obtidos antes mesmo de iniciar a etapa de implementação.

1.3 Contribuições e Limitações

As contribuições desse trabalho surgem no contexto de desenvolvimento de projetos espaciais. De acordo com (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION., 2009), o ciclo de vida pode ser dividido em sete fases: Fase 0, onde a missão como um todo é analisada e suas necessidades são identificadas; Fase A, onde a viabilidade da missão é considerada; Fase B, na qual são feitas as definições preliminares; Fase C, onde essas mesmas definições são detalhadas e especificadas; Fase D, que envolve a produção e qualificação dos componentes de hardware e software necessários; Fase E, onde todas as operações para efetuar o lançamento são finalizadas e, por fim, Fase F em que todos os recursos aplicados no projeto estão disponíveis e funcionais.

Com a etapa de desenvolvimento e teste das técnicas de mitigação de SEU em FPGAs deixadas para a Fase D, é possível que uma mudança causada por um resultado insatisfatório seja muito custosa dado que o projeto já está em um estágio avançado. Para evitar esses problemas, esse trabalho traz uma abordagem capaz de analisar e avaliar essas técnicas nas primeiras fases do projeto espacial, para suprimir as chances de um resultado inesperado surgir no futuro. Dessa forma, os custos podem ser reduzidos, o tempo de projeto diminuído e a qualidade operacional da missão progride.

Para o desenvolvimento de sistemas complexos como projetos espaciais é impensável que todo o processo de construção de hardware e software seja projetado, desenvolvido e testado diretamente nos componentes reais, pois os mesmos possuem um elevado custo e que podem ser danificados ou até mesmo perdidos durante o processo. O uso de uma abordagem baseada em modelos parte do princípio de que, caso exista uma etapa do desenvolvimento que possa ser abstraída em um modelo para

melhor entendimento do problema ou até mesmo para simulações e testes, isso pode aumentar a confiabilidade do resultado final e melhorar a produtividade de todos os envolvidos no processo de desenvolvimento, por tornar as soluções menos voltadas à implementação e mais próximas do domínio do problema, representado através dos modelos desenvolvidos (SELIC, 2003).

Conforme dito anteriormente, esse trabalho apresenta modelos probabilísticos construídos a partir da estrutura de funcionamento de técnicas de mitigação de SEU em FPGAs, e por serem modelos, as limitações surgem à medida que os detalhes são suprimidos ou não são abordados corretamente. Portanto, as taxas calculadas para cada estudo de caso, assim como os detalhes condicionados à cada modelo foram revisados e ajustados com a maior precisão possível. Para que os resultados sejam ainda mais fundamentados, outros estudos de casos devem ser desenvolvidos para aumentar a versatilidade e o detalhamento de cada modelo, buscando suprimir suas limitações. Por fim, a comparação entre os resultados provenientes de uma simulação funcional (ModelSim) traz restrições que seriam contornáveis no caso de implementações no hardware (FPGA) de fato, que devem ser desenvolvidas em trabalhos futuros.

1.4 Organização do texto

A estruturação para esse trabalho de pesquisa é descrita a seguir:

- Capítulo 2. Nesse capítulo, apresenta-se conceitos relacionados aos efeitos da radiação em circuitos integrados em aplicação aeroespacial incluindo SEU em FPGAs, os conceitos e o estado da arte de cada uma das técnicas de mitigação abordadas, e uma revisão sobre *Model Checking* Probabilístico;

- Capítulo 3. Esse capítulo apresenta uma descrição dos modelos CTMC desenvolvidos e das propriedades CSL abordadas, além de uma comparação dos resultados obtidos em cada uma das técnicas considerando os atributos de dependabilidade: disponibilidade, confiabilidade e segurança;

- Capítulo 4. Esse capítulo apresenta uma descrição das técnicas de mitigação de SEU implementadas em VHDL e simuladas no ModelSim, bem como a comparação dos resultados obtidos via modelagem probabilística e simulação funcional;

- Capítulo 5. As considerações finais são apresentadas nesse capítulo junta-

mente com os trabalhos futuros considerados relevantes para a evolução dessa pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta os conceitos relacionados aos efeitos da radiação em circuitos integrados no contexto de aplicações aeroespaciais, incluindo SEU em FPGAs, as definições e o estado da arte de cada uma das técnicas de mitigação abordadas, e uma revisão sobre *Model Checking* Probabilístico. Mas, primeiramente, será abordado o tema dependabilidade.

2.1 Dependabilidade

Dependabilidade surgiu como um termo geral para representar um conjunto de atributos desejáveis para sistemas cujo objetivo é a tolerância a falhas, mas o significado da palavra indica a habilidade de prover um serviço efetivamente qualificado, ou seja, capaz de evitar a ocorrência de uma quantidade de falhas além da aceitável (AVIZIENIS et al., 2004). Os principais atributos da dependabilidade incluem: disponibilidade, que é a prontidão no funcionamento do sistema; confiabilidade para garantir que o sistema permaneça funcional; segurança, que é a capacidade de prevenir adversidades; integridade, responsável pela ausência de alterações indesejadas no sistema e manutenibilidade, para possibilitar alterações e reparos.

No âmbito desse trabalho, é possível definir os atributos analisados da seguinte forma:

- Disponibilidade pode ser definida como a probabilidade do sistema estar operacional em um instante de tempo determinado ou como a relação de tempo em que o sistema se encontra operacional dentro de um intervalo de tempo.
- Confiabilidade é a capacidade de manter o sistema em funcionamento por um período de tempo determinado sem transgredir as condições de operação, dado que no início do período o sistema encontra-se operacional.
- Segurança é a probabilidade do sistema ou estar em funcionamento e se comportando da forma correta ou ter a capacidade de detectar que ocorreu um problema e interromper o funcionamento sem causar maiores danos.

De acordo com (AVIZIENIS et al., 2004), as ameaças aos atributos podem ser divididas em três categorias:

- Falha (*failure*) é a manifestação externa de um desvio no funcionamento

correto de um sistema. No caso de sistemas críticos, falhas devem ser evitadas pois podem causar danos econômicos e humanos severos.

- Erro (*error*) é o desvio no funcionamento correto do sistema, que pode ou não causar uma falha dependendo da quantidade e da relevância dos componentes degradados.

- Falta (*fault*) é a causa da ocorrência de um erro, que pode ser transiente ou permanente. Em sistemas críticos, faltas não mitigadas podem levar a erros, que consequentemente podem levar a falhas.

Para garantir a dependabilidade desejada em um sistema, existem diversos meios como a prevenção, tolerância, remoção e previsão de faltas. Esse trabalho aborda técnicas de tolerância a faltas capazes de detectar, mascarar e até corrigir suas ocorrências.

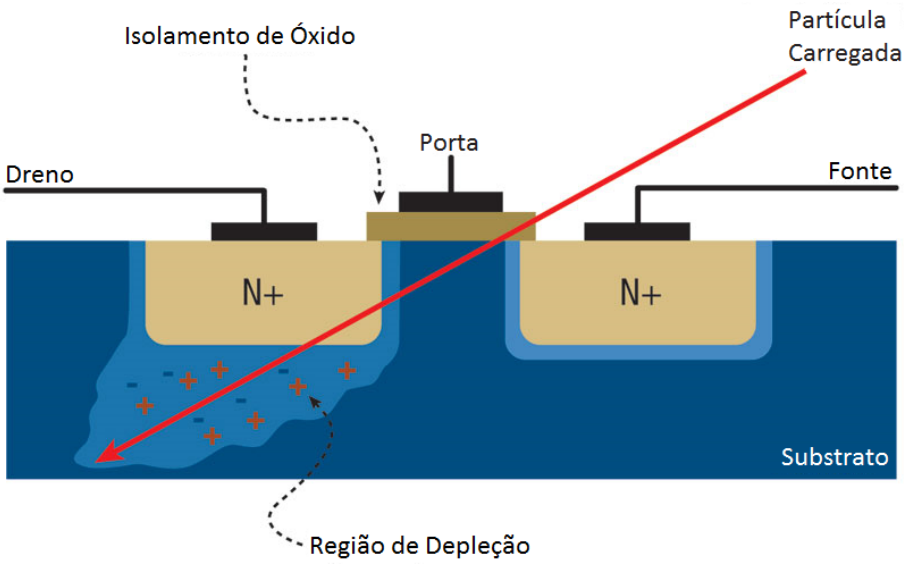
2.2 *Single Event Effects*

Geralmente, em dispositivos eletrônicos, os valores lógicos dos bits de informação são representados por um nível de tensão, ou seja, internamente cada elemento sensível pode ter seu potencial elétrico alterado pela injeção de cargas elétricas (GAILLARD, 2011). *Single Event Effects* (SEEs) ocorrem quando uma partícula de alta energia passa pelo substrato do silício de um componente eletrônico digital (Figura 2.1), onde é possível que parte da carga liberada pela partícula seja maior que a carga elétrica utilizada pelo dispositivo para alterar o valor lógico de cada elemento, provocando uma mudança involuntária de estado ou até danificando permanentemente o dispositivo (WHITE, 2012). Todo equipamento eletrônico desenvolvido em escala nanométrica é suscetível, em algum nível, a SEEs, seja no ambiente espacial, aeronáutico ou até mesmo em nível terrestre (NORMAND, 1996a; NORMAND, 1996b).

Para os efeitos causados pelos *upsets*, duas categorias gerais foram estipuladas: *soft errors*[1], que representa os eventos que causam faltas recuperáveis (temporários) e *hard errors*[1], que representa os eventos que causam faltas não-recuperáveis (permanentes). A Figura 2.2 ilustra os tipos de SEEs e a categoria na qual cada um deles pertence.

[1] Os termos *soft error* e *hard error* foram tratados como *soft fault* e *hard fault*, respectivamente.

Figura 2.1 - Partícula carregada atingindo o substrato de um componente.

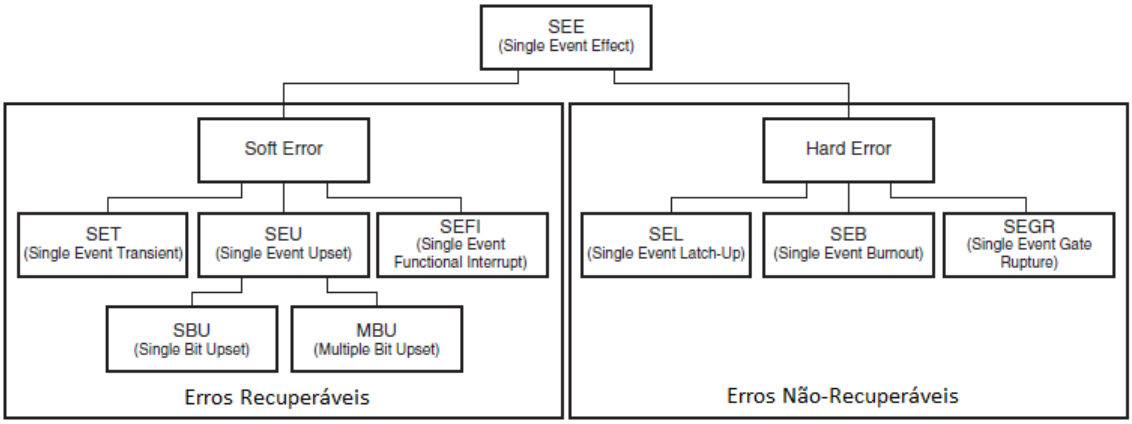


Fonte: adaptado de Sawant (2012).

Dentro do contexto de faltas recuperáveis (*soft errors*), tem-se os seguintes eventos:

- *Single Event Transients* (SETs) ocorrem quando uma partícula carregada atinge um bloco de lógica combinacional do dispositivo, gerando um pulso transiente que pode se propagar pelo circuito e ser interpretado como um sinal válido;
- *Single Event Upsets* (SEUs) ocorrem quando uma partícula carregada atinge um nó sensível de uma célula de memória e o pulso transiente gerado inverte o valor lógico armazenado, ou seja, ocorre um *bit flip*. SEUs podem afetar um ou mais bits de memória ao mesmo tempo, sendo chamado de *Single Bit Upset* (SBU) a inversão de um bit, e *Multiple Bit Upset* (MBU) a inversão de dois ou mais bits. SBU é o tipo mais comum de SEE encontrado em SRAM FPGAs;
- *Single Event Function Interrupts* (SEFIs), como o nome diz, são interrupções no funcionamento de dispositivos eletrônicos causadas por uma partícula ionizante que atingiu um sinal de controle.

Figura 2.2 - Tipos de SEEs.



Fonte: adaptado de White (2012).

Já as faltas não-recuperáveis (*hard errors*) possuem as seguintes subclasses:

- *Single Event Latch-Up* (SEL) ocorre em CMOS quando uma partícula ionizante altera estruturas responsáveis pelo controle de corrente, causando um aumento considerável de corrente que pode danificar de forma permanente o componente;
- *Single Event Burnout* (SEB) é caracterizado pela queima de um componente quando uma partícula atinge um semicondutor aumentando, assim, a temperatura do circuito;
- *Single Event Gate Rupture* (SEGR) é, de forma simplificada, uma ruptura no óxido das portas causada por perturbações no campo elétrico existente ao redor da porta.

SEEs não ocorrem apenas no espaço pois os efeitos da radiação afetam, também, componentes eletrônicos em baixas altitudes e até ao nível do mar (NORMAND, 1996b). Durante décadas, os raios cósmicos vêm afetando negativamente o funcionamento de espaçonaves, foguetes e satélites geoestacionários mas, com a evolução do processo de fabricação dos circuitos integrados, satélites presentes na baixa órbita terrestre (entre 160 e 2000 km de altitude), aeronaves e até mesmo equipamentos utilizados em solo começaram a ser afetados pela radiação presente no cinturão de Van Allen, principalmente nas proximidades da Anomalia Magnética do Atlântico

Sul (SAA) (EDWARDS et al., 2004; DODD et al., 2003; BROSSER; MILH, 2014). Um dos motivos do aumento da ocorrência de SEEs em baixas altitudes é a diminuição do tamanho físico dos circuitos, tornando-os mais sensíveis à radiação. Devido a isso Sims et al. (1994) aplicaram técnicas de mitigação de SEU em equipamentos eletrônicos de aeronaves, Vinter et al. (2005) avaliaram a dependabilidade do hardware utilizado em Veículos Aéreos Não Tripulados (VANTs) com SEUs sendo criados via ferramentas de injeção de falhas.

No contexto da tecnologia empregada na arquitetura das FPGAs, existem três grandes categorias: *Antifuse*, *Flash* e SRAM. Os critérios mais relevantes a se considerar na escolha de uma tecnologia envolvem a programabilidade, custo e sensibilidade à radiação. As FPGAs *antifuse* foram largamente utilizadas em aplicações espaciais devido à sua imunidade perante SEEs e *Total Ionizing Dose* (TID) (WANG et al., 1997). Mas, com o passar do tempo, a necessidade de reconfigurar a memória das FPGAs se tornou algo indispensável para a manutenibilidade dos sistemas, e então as FPGAs com tecnologias *flash* e SRAM se tornaram mais procuradas. No entanto, a possibilidade de reprogramação as torna vulneráveis perante a radiação.

Entre as características das tecnologias, as FPGAs *Flash* possuem reconfiguração significativamente mais lenta e limitada quando comparadas às FPGAS SRAM. Mas por trabalhar com memória não-volátil as FPGAs *Flash* não são sensíveis à SEEs, sendo afetadas apenas pela dose total ionizante que ocorre à longo prazo (WANG, 2003). Dentre as três tecnologias, a única sensível à SEEs é a SRAM, pois utiliza memória volátil. As vantagens são que sua reconfiguração se torna mais eficiente e a dose total ionizante deixa de ter um efeito significativo (WANG et al., 1999).

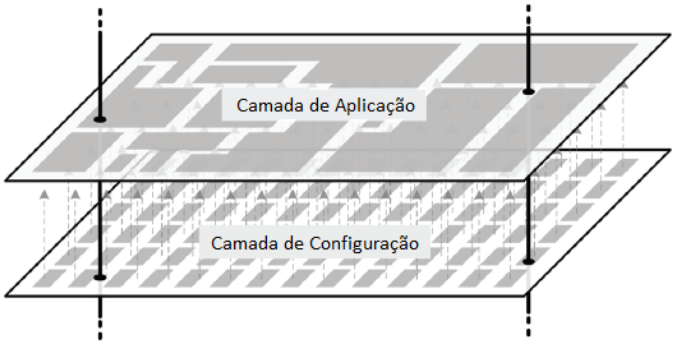
Esse trabalho está no contexto de SEU em memórias SRAM de FPGAs utilizadas em aplicações espaciais e, dessa forma, a próxima seção apresentará os detalhes desse fenômeno nessa plataforma específica.

2.3 *Single Event Upsets* em FPGAs SRAM

O uso de FPGAs em aplicações espaciais nas últimas décadas mostra que sua flexibilidade, desempenho e custo se destacam comparada a outras tecnologias disponíveis. A possibilidade de reprogramação remota dos modelos com memória SRAM se torna vantajosa para o uso em satélites, balões estratosféricos e espaçonaves em missões críticas (BERNARDI et al., 2004). Conceitualmente, as FPGAs podem ser divididas em duas camadas. A camada de aplicação possui os blocos lógicos e elementos de memória responsáveis pelo funcionamento do sistema desenvolvido pelo usuário. A

camada de configuração possui os blocos lógicos e elementos de memória responsáveis por configurar o funcionamento e controlar os recursos utilizados pela camada de aplicação (HERRERA-ALZU; LOPEZ-VALLEJO, 2013). A Figura 2.3 mostra as camadas conceituais em uma FPGA.

Figura 2.3 - Camadas conceituais em uma FPGA.

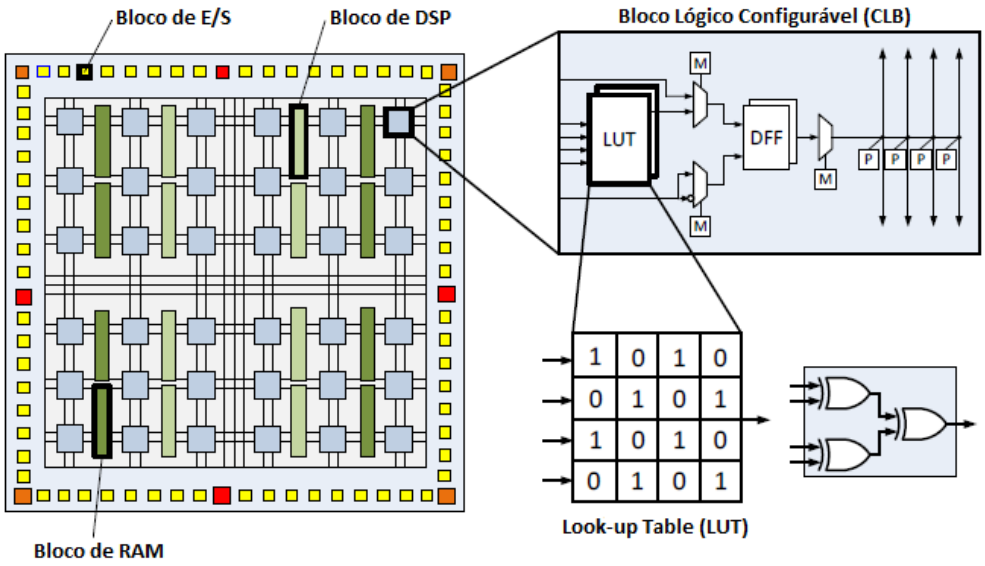


Fonte: adaptado de Herrera-Alzu e Lopez-Vallejo (2013).

A estrutura da camada de aplicação é composta, basicamente, por blocos lógicos configuráveis (CLBs) em sua grande maioria, pois são justamente os responsáveis pela lógica sequencial e combinacional do sistema por meio das *look-up-tables* (LUTs) e dos *flip-flops*. Existem também outros recursos como blocos de memória RAM (BRAM), de entrada e saída (IOB) e de processamento de sinais digitais (DSPs), e todos esses componentes trocam informações por meio das interconexões programáveis (BROSSER; MILH, 2014), como mostra a Figura 2.4.

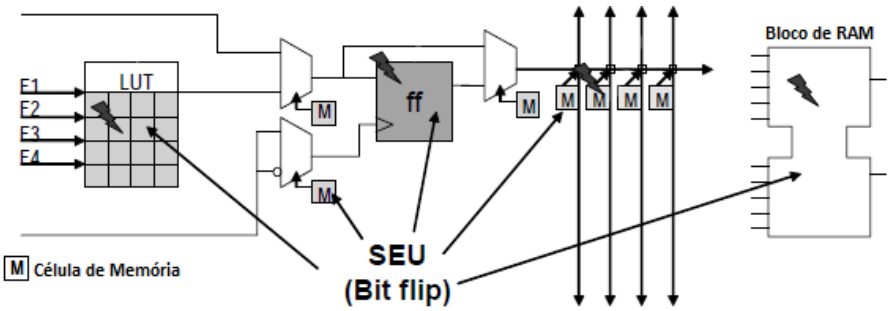
Com essa visão, é possível pressupor que FPGAs são sensíveis aos efeitos de radiação ionizante em ambas as camadas de aplicação e configuração (HERRERA-ALZU; LOPEZ-VALLEJO, 2013), especialmente os *bit-flips* causados pelo SEU nas células de memória. Apesar de blocos combinacionais não possuírem memória, em SRAM FPGAs, tanto a lógica combinacional quanto a sequencial são implementadas por CLBs, ou seja, por memória SRAM. Portanto, o que possivelmente seria apenas um pulso transiente não-prejudicial em outros dispositivos, em FPGAs essa partícula que atingiu um bloco combinacional pode causar uma inversão de bit na célula de memória de configuração responsável por aquele bloco (KASTENSMIDT,). A Figura 2.5 mostra os tipos de SEU que podem ocorrer em SRAM FPGAs.

Figura 2.4 - Arquitetura de uma FPGA.



Fonte: adaptado de Brosser e Milh (2014).

Figura 2.5 - Blocos afetados pelo SEU em SRAM FPGAs.



Fonte: adaptado de Kastensmidt ().

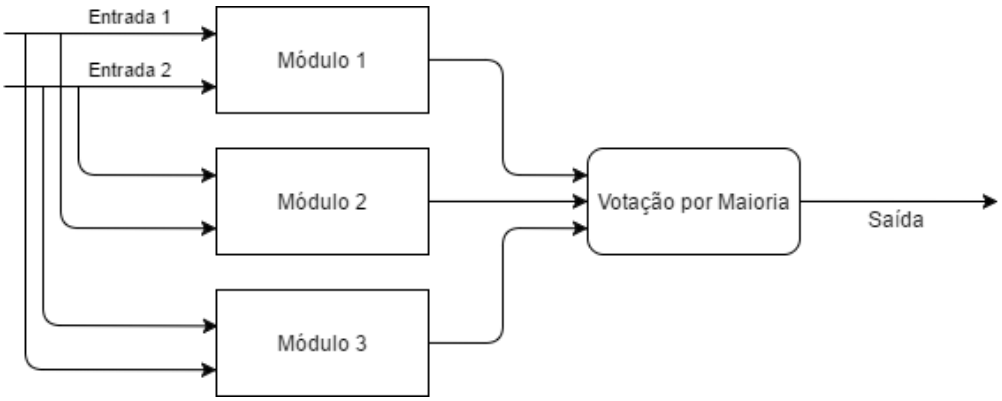
Para mitigar e corrigir os SEUs em SRAM FPGAs, diversas técnicas foram desenvolvidas e aperfeiçoadas ao longo dos anos, mas normalmente cada técnica depende de diversos fatores para ser considerada a mais adequada para cada aplicação. As próximas seções descrevem o funcionamento das técnicas abordadas nesse trabalho, assim como uma visão geral de trabalhos relacionados.

2.4 Técnicas de Mitigação de SEU em FPGAs

2.4.1 Redundância Modular Tripla (TMR)

O TMR é uma das técnicas mais utilizadas para prevenir faltas causadas por SEUs, especialmente SBUs (KASTENSMIDT et al., 2005). Em geral, a estratégia do TMR é usar três réplicas dos componentes mais sensíveis aos efeitos da radiação pois, em caso de *upset*, o mecanismo de votação será capaz de escolher, pela maioria, qual será a saída correta (GRAHAM et al., 2003), como representado na Figura 2.6.

Figura 2.6 - Diagrama que representa o funcionamento do TMR.



Fonte: produção do autor.

O mascaramento do componente atingido pela radiação é possível pois o TMR remove pontos individuais de falta por meio da redundância, porém, como em qualquer sistema redundante, existe um custo. Para garantir que a falta não atinja um ponto comum entre todas as réplicas e propague o defeito para a saída, cada componente redundante deve ter portas individuais de entrada e saída, causando um aumento de aproximadamente seis vezes na área de memória requerida (MORGAN et al., 2007). Portanto, decidir em qual nível lógico o TMR será aplicado é uma tarefa complexa, e a relação entre a área adicional e o ganho efetivo deve ser considerada.

Além do uso de área adicional, outra importante característica do TMR é que, como sua estratégia mascara o defeito ao invés de corrigir, o componente atingido continuará a se deteriorar, aumentando as chances de um acúmulo de *upsets* (KASTENSMIDT, 2007). O uso do TMR deve ser levado em consideração em aplicações

com uma baixa incidência de MBUs, pois não é capaz de lidar com mais de uma falta simultânea nos módulos replicados.

Em (MORGAN et al., 2007), os autores compararam o TMR com três técnicas adicionais de mitigação de SEU em FPGAs: lógica quadruplicada, codificação de máquina de estado e redundância temporal. Para simular o ambiente radioativo espacial, os autores utilizaram uma ferramenta de injeção de falhas Virtex da Xilinx e analisaram os resultados baseados na confiabilidade, sobrecarga da área e redução da sensibilidade das FPGAs. A conclusão foi que as técnicas apresentaram uma sobrecarga de área maior e uma redução na sensibilidade menor quando comparadas ao TMR, garantindo ao TMR uma maior confiabilidade. Em (SHULER et al., 2009), o TMR também foi comparado a outros métodos de mitigação como *flip-flops radiation-hardened* e lógica *dual-rail*. Considerando a eficiência no uso de área, o TMR apresentou o melhor resultado com 95% de eficiência.

2.4.2 *Scrubbing*

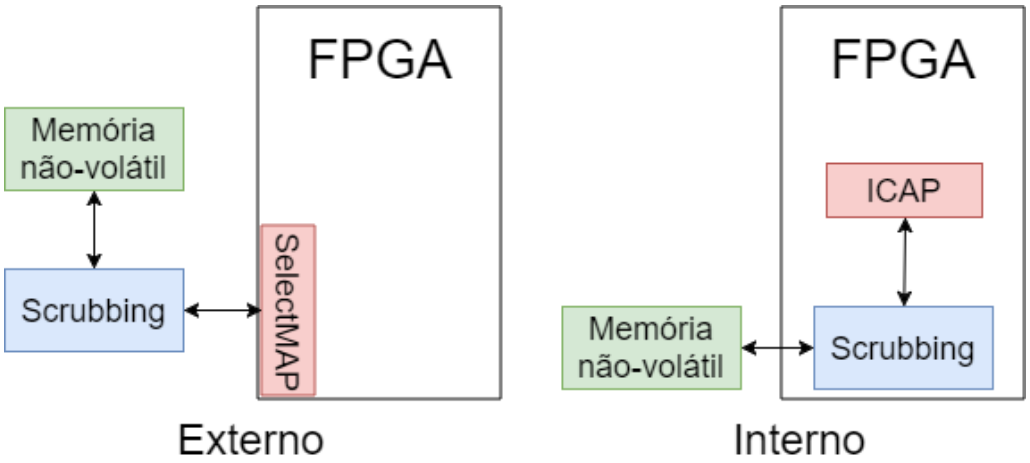
Dado o contexto do TMR, é necessário uma técnica capaz de corrigir faltas ao invés de apenas suprimí-las. O *Scrubbing* é um mecanismo que reconfigura a memória utilizando o bitstream com a configuração original, normalmente armazenada em uma memória que seja imune à radiação e SEEs (SARI et al., 2013). Uma das maiores vantagens do *Scrubbing* em relação a uma reconfiguração total do dispositivo é que o *Scrubbing* não interrompe completamente o modo operacional do sistema, apenas a parte atingida é atualizada em cada intervalo de *Scrubbing* (BERG et al., 2008).

A forma mais comum de *Scrubbing* é conhecida como *Blind Scrubbing*, pois não inclui um mecanismo de detecção de faltas, apenas uma cópia dos dados originais é utilizada para reescrever a configuração atual. Uma das decisões no *Blind Scrubbing* é o intervalo de tempo, ou seja, a frequência em que cada ciclo deve ocorrer. O intervalo de *Scrubbing* depende do nível de confiabilidade desejado para o sistema e a frequência com que os *upsets* ocorrem. Outras formas mais sofisticadas de *Scrubbing* como o *Readback Scrubbing*, utilizam internamente técnicas de detecção e são capazes de restaurar as configurações de memória somente quando uma falta é detectada, evitando que o *Scrubbing* seja executado sem necessidade (HEINER et al., 2009).

Existem duas arquiteturas principais para implementar o *Scrubbing* em FPGAs: utilizando um dispositivo interno ou externo para processar a técnica, como mostra a Figura 2.7. No caso da arquitetura externa, normalmente utiliza-se hardware *radiation-hardened* para garantir que o *Scrubbing* não fique vulnerável a *upsets*, au-

mentando a confiabilidade mas também o custo. Para o *Scrubbing* interno, existe uma porta de acesso à configuração interna (ICAP) que faz a comunicação entre a memória de configuração e a cópia contendo a configuração original. Apesar da arquitetura interna afetar menos o custo, a confiabilidade pode ficar comprometida, pois a implementação da técnica estará exposta à radiação (BERG et al., 2008; HOQUE, 2016).

Figura 2.7 - Duas formas de arquitetura do *Scrubbing*.



Fonte: produção do autor.

Em (BERG et al., 2008) os autores compararam a eficiência das duas arquiteturas de *Scrubbing*, sendo a primeira interna e aliada à uma técnica de detecção (*Readback Scrubbing*) e a segunda externa e sem técnica de detecção (*Blind Scrubbing*). Os resultados mostraram que, mesmo sendo mais sofisticado, o *Scrubbing* interno não teve eficiência superior ao *Scrubbing* externo e ficou com uma pequena vantagem em relação à FPGA sem *Scrubbing* implementado. O melhor resultado foi apresentado pelo *Scrubbing* externo, mas os autores sugerem adicionar também a reconfiguração total para melhorar o desempenho.

2.4.3 Código de Hamming

Código de Hamming é uma das técnicas de ECC utilizada em diversas aplicações para a detecção de faltas de 1 ou 2 bits e correção de faltas de 1 bit em códigos binários (DUTTA; TOUBA, 2007). Assim como o TMR, o código de Hamming é uma

técnica indicada para sistemas com baixa incidência de MBUs.

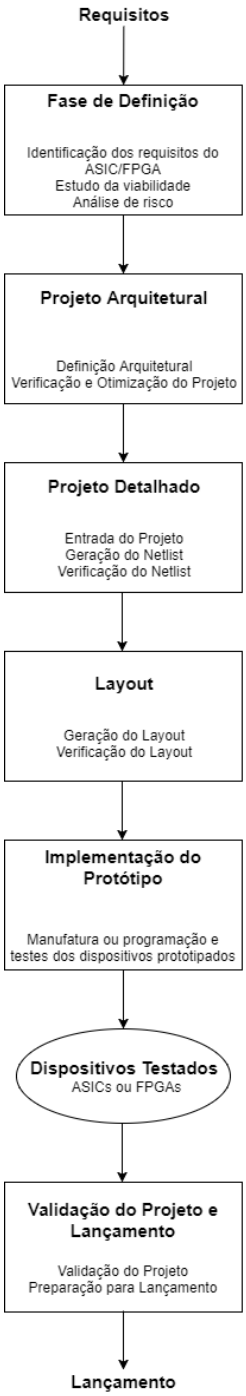
O funcionamento do código de Hamming é estruturado em dois módulos, um codificador e um decodificador. O codificador é responsável por calcular os bits de paridade baseado nos dados de entrada e adicioná-los ao fluxo de bits (KASTENSMIDT et al., 2005). O trabalho mais complexo fica por conta do decodificador, que deve checar os bits de paridade novamente e verificar se existe algum defeito. Em caso de divergência na paridade, o decodificador é responsável por encontrar qual bit foi invertido e efetuar a correção (LIU et al., 2012). Existem diversas variações de código de Hamming projetadas para aplicações com diferentes objetivos. Nesse trabalho, considerou-se o código de Hamming (7,4) que codifica 3 bits de paridade para cada 4 bits de dados.

Em (LIU et al., 2012), os autores compararam o código de Hamming com outro ECC, o código cíclico do conjunto diferença (*Difference-Set Cyclic Code* - DSCC). O objetivo foi analisar o desempenho, custo de projeto e taxa de falha. Os resultados mostraram que apesar do DSCC ser mais adequado para a arquitetura das FPGAs da Xilinx, na prática o código de Hamming mostrou uma taxa de falhas menor e uma economia de energia melhor, resultando em um custo de projeto mais factível. Uma comparação direta entre o TMR e o código de Hamming foi apresentada em (HENTSCHKE et al., 2002) analisando o impacto no desempenho e na área. Como esperado, os resultados mostraram vantagens e desvantagens em cada uma das técnicas. O TMR aumentou significativamente a área das células de memória mas funciona melhor pois tem um atraso menor, enquanto o código de Hamming tem apenas um pequeno aumento no uso das células de memória, mas, dependendo da quantidade de bits que precisa proteger, pode adicionar um atraso significativo ao sistema.

2.5 Qualificação de FPGAs para lidar com SEUs

A *European Cooperation for Space Standardization* (ECSS) apresentou em (EUROPEAN COOPERATION FOR SPACE STANDARDIZATION, 2008) um regulamento para o desenvolvimento de sistemas críticos em aplicações espaciais voltado especificamente para FPGAs e ASICs. De forma simplificada, as etapas de desenvolvimento estão ilustradas na Figura 2.8, na qual as etapas de gerenciamento e documentação foram omitidas.

Figura 2.8 - Fluxo de desenvolvimento.



Fonte: adaptado de EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (2008).

As etapas definidas na Figura 2.8 tem como propósito garantir que o sistema possua níveis elevados de confiabilidade. Desde a primeira fase são gerados relatórios, planos de controle e reuniões a cada iteração mas, no caso de um risco em potencial ser detectado, o protocolo define que uma nova iteração deva ser executada. De acordo com (BERNARDESCHI et al., 2015), existem requisitos adicionais para missões espaciais que possibilitam a ocorrência de faltas causadas pela radiação. Além dos procedimentos padrão, são necessários testes tanto em solo quanto em voo dos dispositivos, estratégias para lidar com possíveis erros e, por fim, provar analiticamente que a cobertura de faltas requerida é atingida durante os testes.

É possível notar que o desenvolvimento de uma FPGA para aplicações espaciais é complexo e extenso. Introduzir nas fases iniciais do desenvolvimento (e.g. Fase de Definição, Projeto Arquitetural) um método de análise de mitigação de SEUs baseado em modelos probabilísticos capaz de auxiliar no processo de elaboração do sistema como um todo, tornando as etapas menos custosas e diminuindo a possibilidade de ter que repetir o procedimento pela ocorrência de algo não previsto, é inovador em diversos sentidos, principalmente considerando a sensibilidade de FPGAs SRAM em relação à radiação. Além dos trabalhos recentes publicados por (HOQUE, 2016; HOQUE et al., 2014), essa abordagem não é encontrada na literatura e deve ser explorada, pois o intuito é facilitar e acelerar o fluxo de desenvolvimento baseado em FPGAs.

2.5.1 Determinação de taxas de SEU

Como dito anteriormente, grande parte do processo de desenvolvimento de um modelo recai na determinação das taxas, pois elas definem a corretude e a proximidade dos resultados do modelo em relação ao experimento factual. Nesse trabalho, as taxas utilizadas indicam o tempo médio entre as falhas (*Mean Time Between Failures* - MTBF), ou seja, uma previsão de quanto tempo o sistema estará funcional até que a próxima falha ocorra, mas o objetivo desta seção é demonstrar como as taxas foram calculadas. A definição de MTBF adotada parte do contexto de sistemas complexos reparáveis, ou seja, baseado em uma taxa de falha constante, normalmente utilizada para predições de atributos de dependabilidade, é possível estabelecer um tempo médio que é dado pelo valor esperado da função de densidade do tempo até que uma falha ocorra (BIROLINI, 2017). É importante enfatizar que, para que as taxas de falhas e os MTBFs calculados a partir delas sejam relevantes, é preciso considerar que todo o sistema está operando dentro de seu período de vida útil (parte constante da "curva da banheira"), ou seja, quando apenas as falhas de natureza

aleatória ocorrem, como é o caso dos SEUs.

De forma geral, os SEUs podem ser causados por dois tipos de ionização: direta e indireta (HOWE et al., 2005). Na ionização direta, as partículas carregadas como fótons e elétrons presentes no ambiente atingem a superfície do componente eletrônico e a carga liberada por essas partículas ultrapassa a carga limite suportada pela região atingida, causando um *upset*. Normalmente o cálculo da probabilidade de uma ionização direta ocorrer é baseado no corte transversal (*cross section*), que é o tamanho da área sensível analisada em relação à transferência de energia linear (*Linear Energy Transfer* - LET), ou seja, o quanto de energia pode ser transferida para o componente. Já no caso da ionização indireta, partículas desprovidas de cargas elétricas como nêutrons que, sozinhas, são incapazes de causar um *upset*, interagem com outras partículas presentes no silício e o produto dessa reação pode gerar cargas suficientes para ultrapassar os limites que a região sensível atingida suporta e, indiretamente, causar um *upset* (KOGA et al., 2004).

No início da década de 80, o Laboratório de Pesquisa Naval desenvolveu um modelo empírico capaz de estimar com precisão a ocorrência desses *upsets*, que ficou conhecido como CREME (*Cosmic Ray Effects on Microelectronics*) (ADAMS et al., 1981). Mais de 10 anos depois, uma atualização no modelo foi implementada para melhorar rotinas de transporte nuclear, transmissão geomagnética e as técnicas de cálculo de SEUs, renovar os parâmetros dos modelos de radiação cósmica e adicionar novas funcionalidades ao código, como a possibilidade de estimar as taxas para SEUs causados tanto por radiação direta quanto indireta, além de uma interface gráfica intuitiva (TYLKA et al., 1997). Algumas revisões surgiram mais tarde, mas o CREME96 ainda é largamente utilizado para estimativas de *upsets* para as mais diversas aplicações espaciais.

Atualmente, o CREME96 pode ser consultado por meio de uma página na *web*[2] mantida pela Escola de Engenharia da Universidade Vanderbilt. É preciso ter algumas respostas antes de começar os cálculos para que o os resultados sejam mais eficientes, como saber a órbita do satélite em questão, qual o tipo de ambiente de radiação ele enfrentará e qual o tipo de proteção contra radiação, interna ou externa, o satélite possui. Para o cálculo das taxas de SEEs, informações adicionais como as características do dispositivo e os valores de *cross-sections* tanto para ionização direta quanto indireta são essenciais.

[2] `https://creme.isde.vanderbilt.edu`

Para tornar os cálculos mais eficientes, o CREME96 foi dividido em módulos, onde cada módulo gera uma saída que deve ser a entrada do módulo seguinte. Abaixo estão elicitados os módulos e qual a função de cada um deles:

- GTRN: Avalia os efeitos da proteção geomagnética exercida pelo campo magnético da Terra contra algumas partículas energizadas. GTRN é indicado apenas para dispositivos em órbitas geossíncronas;
- TRP: Avalia o fluxo de prótons aprisionados no campo magnético da Terra, dado que estes são grandes causadores de SEEs em baixa órbita. Dessa forma, o TRP só é necessário para órbitas dentro da magnetosfera da Terra;
- FLUX: Avalia o ambiente de radiação ionizante na superfície externa do satélite antes que a radiação passe através dos mecanismos de proteção existentes. FLUX deve ser utilizado em órbitas geossíncronas, interplanetárias e até mesmo nas órbitas dentro da magnetosfera da Terra;
- TRANS: É responsável por transportar o fluxo de partículas através do nível de proteção do satélite que foi especificado inicialmente, considerando a perda de energia e a fragmentação nuclear. É o único módulo presente no cálculo de ambos os tipos de ionização, direta e indireta;
- LETSPEC: Transforma o fluxo de partículas em espectros das funções de Transferência de Energia Linear (LET). Esse módulo só é utilizado no cálculo das taxas de SEUs causados por ionização direta (*heavy ions*);
- HUP: É o módulo responsável por calcular a taxa de SEUs causados pela ionização direta, ou seja, por *heavy ions*. Nesta etapa, as características do dispositivo analisado devem ser introduzidas;
- PUP: É o módulo responsável por calculas a taxa de SEUs causados pela ionização indireta, ou seja, induzida por prótons. Assim como no módulo HUP, no PUP as características do dispositivo também são consideradas.

Para chegar no resultado final, uma sequência específica de módulos deve ser executada para cada tipo de ionização. Para o cálculo das taxas de SEUs causados por ionização direta, a sequência deve ser: GTRN, FLUX, TRANS, LETSPEC e então HUP. Já no caso da ionização indireta, a sequência é: TRP, TRANS e então PUP. Dessa forma, cada módulo (com exceção do primeiro) exige um arquivo de entrada

que é dado pelo módulo anterior, isso facilita e agiliza o processo pois é possível guardar um arquivo de saída de um módulo para executar modificações em módulos posteriores.

Para exemplificar o processo, será mostrado como foi feito o cálculo das taxas de SEU para ionização direta e indireta do satélite sino-brasileiro de recursos terrestres CBERS-4A, que foi um dos estudos de caso desse trabalho. Começando com a ionização direta, o primeiro módulo GTRN exige a introdução de características da órbita de interesse para o cálculo da função de transmissão geomagnética. As principais informações sobre a órbita do CBERS-4A estão disponíveis em sua página na *web*[3] e transcritas abaixo:

- Altitude orbital média: 628,614 km (Raio da Terra é 6371 km)
- Semieixo maior: 6999,614 km
- Excentricidade: 0,0012
- Inclinação: 97,8963 graus
- Argumento do perigeu: 90 graus
- Horal local do nodo descendente: 10:30 a.m.
- Período nodal: 97,2549 minutos
- Revoluções por dia: 14 + 25/31

Apesar de não estarem disponibilizados oficialmente, os dados de apogeu e perigeu da órbita podem ser calculados indiretamente por meio de um sistema de equações lineares. Dado que SMa é o semieixo maior, é possível obter a primeira equação:

$$SMa = \frac{Ra + Rp}{2} \tag{2.1}$$
$$6999,614 = \frac{Ra + Rp}{2}$$
$$Ra + Rp = 13999,228$$

A segunda equação utiliza a excentricidade e:

[3] http://www.cbers.inpe.br/sobre_satelite/orbita_cbers4A.php

$$e = \frac{Ra - Rp}{Ra + Rp} \tag{2.2}$$

$$0,0012 = \frac{Ra - Rp}{Ra + Rp}$$

$$Ra - Rp = 16,799$$

Por meio das Equações 2.1 e 2.2 é possível resolver o sistemas de duas equações e chegar nos valores de Ra e Rp. Por fim, com o valor do raio da Terra (Re), o apogeu pode ser determinado da seguinte forma:

$$Apogeu = Ra - Re \tag{2.3}$$

$$Apogeu = 7008,0135 - 6371$$

$$Apogeu = 637,0135$$

E o perigeu é obtido da mesma maneira:

$$Perigeu = Rp - Re \tag{2.4}$$

$$Perigeu = 6991,2145 - 6371$$

$$Perigeu = 620,2145$$

Com essas características de órbita definidas, é possível preencher os parâmetros do GTRN, submeter e obter o arquivo de saída com a extensão *.gtf*, que será a entrada do módulo FLUX no qual as informações sobre o ambiente de radiação ionizante são introduzidas. Para os módulos FLUX, TRANS (responsável pela caracterização do transporte nuclear dos dispositivos) e LETSPEC (módulo que considera a transferência linear de energia no cálculo das taxas) a recomendação é deixar as opções definidas por padrão e adicionar no módulo final (HUP) as características do dispositivo escolhido, que para esse estudo de caso foi a FPGA Virtex-5 XC5VFX130, considerando tanto os bits de configuração quanto os bits dos blocos de memória. Os dados específicos de cada bloco da FPGA como o *cross-section* e as dimensões das regiões sensíveis foram obtidos em (SWIFT; GREGORY, 2012), enquanto a saída final do CREME96 com as taxas de SEEs/bit/segundo causados por *heavy ions* para os bits de configuração (*Report No. 1*) e para os bits dos blocos de memória (*Report*

No. 2) estão transcritos na Figura 2.9.

Figura 2.9 - Saída do CREME96 com as taxas de SEEs/bit/segundo para os *heavy ions*.

```
REPORT NO.    1:   virtex5 T O D O
RPP Dimensions: X =    1.82000  Y =    1.82000  Z =    1.00000 microns.
Funnel length =    0.00000 microns.
CROSS-SECTION INPUT   4  WEIBULL FIT:
    ONSET   =     0.100 MeV-cm2/milligram
    WIDTH   =    10.860 MeV-cm2/milligram
    POWER   =     1.750 (dimensionless)
    PLATEAU =     3.330 square microns/bit
Number of bits =   1.00000E+00
Rates:      SEEs/bit/second     /bit/day    /device/second    /device/day
*****   1     1.28824E-12     1.11304E-07     1.28824E-12     1.11304E-07

REPORT NO.    2:   virtex5 T O D O
RPP Dimensions: X =    1.00000  Y =    1.00000  Z =    1.00000 microns.
Funnel length =    0.00000 microns.
CROSS-SECTION INPUT   4  WEIBULL FIT:
    ONSET   =     0.250 MeV-cm2/milligram
    WIDTH   =   100.000 MeV-cm2/milligram
    POWER   =     2.950 (dimensionless)
    PLATEAU =     1.000 square microns/bit
Number of bits =   1.00000E+00
Rates:      SEEs/bit/second     /bit/day    /device/second    /device/day
*****   2     1.53736E-16     1.32828E-11     1.53736E-16     1.32828E-11
```

Fonte: produção do autor.

Para o cálculo das taxas de *upset* causadas pela ionização indireta, são necessários apenas os módulos TRP (responsável por calcular o espectro de prótons presos utilizando as informações de órbita citadas anteriormente), TRANS (único módulo presente em ambos os tipos de ionização) e o módulo final (PUP) que exige as informações de *cross-section* tal qual o módulo HUP. Como as características tanto de órbita quanto do dispositivo já haviam sido definidas, esses valores foram apenas mantidos para obter a saída final dos SEEs/bit/segundo induzidos por prótons nos bits de configuração (*Report No. 1*) e nos bits dos blocos de memória (*Report No. 2*) como mostra a Figura 2.10.

Figura 2.10 - Saída do CREME96 com as taxas de SEEs/bit/segundo para os prótons.

```
REPORT NO.    1:   virtex5 T O D O
CROSS-SECTION INPUT   4  WEIBULL FIT:
    ONSET   =     0.800 MeV
    WIDTH   =    12.000 MeV
    POWER   =     0.600 (dimensionless)
    PLATEAU =     0.047 x 10**-12 cm2/bit
Number of bits =   1.00000E+00
Rates:      SEEs/bit/second     /bit/day    /device/second    /device/day
*****   1     2.48063E-12     2.14327E-07     2.48063E-12     2.14327E-07

REPORT NO.    2:   virtex5 T O D O
CROSS-SECTION INPUT   4  WEIBULL FIT:
    ONSET   =     5.000 MeV
    WIDTH   =    50.000 MeV
    POWER   =     1.000 (dimensionless)
    PLATEAU =     0.000 x 10**-12 cm2/bit
Number of bits =   1.00000E+00
Rates:      SEEs/bit/second     /bit/day    /device/second    /device/day
*****   2     1.02846E-17     8.88590E-13     1.02846E-17     8.88590E-13
```

Fonte: produção do autor.

2.6 *Model Checking* Probabilístico

Sistemas complexos críticos utilizados em aplicações aeroespaciais, médicas, de comunicação, entre outras são muito sensíveis a falhas de software e/ou hardware. Atualmente, métodos clássicos de validação de sistemas incluem simulação, que é muito utilizada no nível de modelagem e, com o sistema já construído, são executados os ciclos de teste. Apesar de eficiente para a maioria dos problemas, quando se trata de sistemas críticos, apenas a simulação pode não ser suficiente para lidar com todas as possibilidades e garantir o nível desejado de segurança. Acidentes registrados no passado mostram que os custos financeiros e humanos de uma falha são incalculáveis (CLARKE et al., 1999). Com a evolução da tecnologia, sistemas embarcados estão cada vez mais presentes no dia-a-dia das pessoas, portanto, é crucial desenvolver técnicas capazes de aumentar a confiabilidade, segurança e disponibilidade desses dispositivos.

Para aumentar a precisão, Métodos Formais têm mostrado grande potencial para reduzir o tempo gasto e aumentar a eficácia do processo de modelagem aplicando rigor matemático para garantir a corretude do sistema (BAIER et al., 2008). *Model Checking* (CLARKE et al., 1999) é um método muito popular de Verificação Formal que explora exaustivamente o espaço de estados de uma maneira automatizada e é

capaz de encontrar defeitos em uma solução (*Transition System* - TS) como estados inalcançáveis ou *deadlocks*. Tipicamente, uma propriedade é formalizada via uma variedade de lógicas temporais como a *Linear Temporal Logic* (LTL) (CLARKE et al., 1986) e a *Computation Tree Logic* (CTL) (BAIER et al., 2003), e a abordagem verifica, sistematicamente, se o TS (modelo) satisfaz esta propriedade ($TS \models \Phi$).

No escopo de *Model Checking*, uma categoria que lida com sistemas que apresentam comportamento estocástico é o *Model Checking* Probabilístico (BAIER et al., 2008; KWIATKOWSKA et al., 2009; HOQUE et al., 2014). Nesse caso, ao invés da garantia absoluta da corretude dificilmente atingida pelo *Model Checking* tradicional/funcional, *Model Checking* Probabilístico é capaz de garantir, de acordo com uma probabilidade especificada e de uma maneira menos exigente, a corretude do sistema.

DTMCs e CTMCs são tipicamente utilizadas como modelo probabilístico, na qual o conjunto de estados representam possíveis configurações do sistema a ser modelado e as transições entre os estados constituem o avanço entre as condições estabelecidas em cada estado, que podem ocorrer em intervalos de tempo discretos no caso do DTMC e contínuos no caso do CTMC. Atrelado a cada transição, existem os valores probabilísticos de uma determinada transição ocorrer e que são dados por distribuições de probabilidade discretas no DTMC e contínuas no CTMC. Cadeias de Markov possuem uma propriedade muito característica conhecida como "sem memória", ou seja, para um transição ocorrer apenas informações provenientes do estado atual são consideradas. Mas a decisão sobre qual cadeia de Markov utilizar depende majoritariamente da aplicação pretendida, e no contexto desse trabalho, os modelos foram desenvolvidos com CTMCs, por serem mais indicados para aplicações que envolvam análise de dependabilidade.

Uma definição para CTMC é mostrada a seguir.

Definição 2.1. Um CTMC (rotulado) (KWIATKOWSKA et al., 2007) é uma tupla $C = (S, i, R, L)$, onde:

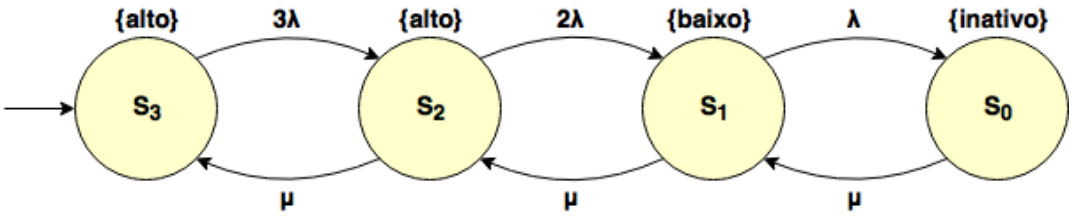
- S é um conjunto finito de estados;
- $i \in S$ é o estado inicial;
- $R : S \times S \longrightarrow \mathbb{R}_{\geq 0}$ é a matriz de taxas de transição;
- $L : S \to 2^{PA}$ é a função de rotulação que atribui para cada estado $s \in S$ o conjunto $L(s)$ de proposições atômicas que são válidas no estado ($PA =$

conjunto de proposições atômicas).

A matriz R atribui taxas para cada par de estados no CTMC (usados como parâmetros da distribuição exponencial).

Uma transição só pode ocorrer entre estados s e $s' \Rightarrow R(s, s') > 0$ e, nesse caso, a probabilidade desta transição ser desencadeada dentro de t unidades de tempo é igual a $1 - e^{-R(s,s')\cdot t}$ (função de distribuição acumulada - CDF). Um exemplo de CTMC é dado na Figura 2.11, na qual foi modelado um sistema com três máquinas onde cada máquina pode falhar independentemente. A taxa de falha é λ, portanto o tempo médio entre as falhas é de $\frac{1}{\lambda}$, a taxa de reparo de uma máquina é μ e o espaço de estados é $S = \{s_i\}_{i=0...3}$ onde s_i indica a quantidade i de máquinas operacionais.

Figura 2.11 - Exemplo de reparo de máquinas modelado em CTMC.



Fonte: produção do autor.

Tal qual o *Model Checking* funcional, o *Model Checking* Probabilístico utiliza lógicas temporais para descrever suas propriedades, e que nesse contexto são variações de CTL, como *Probabilistic Computation Tree Logic* (PCTL) (BAIER et al., 2008) para DTMC, e CSL (BAIER et al., 2003) para CTMC, que foi a lógica temporal utilizada para esse trabalho. Em todos os casos, a modelagem é um fator fundamental para a análise da dependabilidade do sistema, que depende diretamente do modelo estocástico e da especificação da lógica temporal abordada.

A definição formal da lógica CSL é dada a seguir:

Definição 2.2. A sintaxe das fórmulas de estado (Φ), sobre o conjunto PA de proposições atômicas, e fórmulas de caminho (φ) de CSL (SANTIAGO JÚNIOR, 2016) são de acordo com as seguintes gramáticas:

- $P_{est} \Rightarrow \Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}[\varphi] \mid \mathcal{S}_{\bowtie p}[\Phi]$
- $P_{cam} \Rightarrow \varphi ::= \mathcal{X}^{\mathcal{I}}\ \Phi \mid \Phi_1\ \mathcal{U}^{\mathcal{I}}\ \Phi_2$

onde:

- $a \in PA$;
- $\bowtie \in \{\leq, <, >, \geq\}$;
- $p \in [0, 1]$;
- $\mathcal{I}$ é um intervalo de $\mathbb{R}_{\geq 0}$;
- $\Phi, \Phi_1, \Phi_2 \Rightarrow$ fórmulas de estado;
- $\varphi \Rightarrow$ fórmula de caminho.

Os operadores $\mathcal{P}$ e $\mathcal{S}$ descrevem a probabilidade de caminho e a probabilidade limite, respectivamente. Já os operadores $\mathcal{X}$ e $\mathcal{U}$ indicam respectivamente 'próximo' (*next*) e 'até' (*until*). Os significados das fórmulas que utilizam os operadores são descritos de maneira informal abaixo:

- $\mathcal{P}_{\bowtie p}[\varphi] \Rightarrow$ a probabilidade para o conjunto de caminhos, começando em um estado s_i, que satisfaz a fórmula de caminho φ encontra o limite dado por $\bowtie p$;
- $\mathcal{S}_{\bowtie p}[\Phi] \Rightarrow$ a probabilidade limite de estar em um estado s_i satisfazendo a fórmula de estado Φ encontra o limite dado por $\bowtie p$;
- $\mathcal{X}^{\mathcal{I}}\ \Phi \Rightarrow$ uma transição é feita para o estado s_i que satisfaz a fórmula de estado Φ em algum instante de tempo $t \in \mathcal{I}$;
- $\Phi_1\ \mathcal{U}^{\mathcal{I}}\ \Phi_2 \Rightarrow$ a fórmula de estado Φ_2 é satisfeita em algum instante de tempo $t \in \mathcal{I}$ e que em todos os instantes de tempo precedentes a fórmula de estado Φ_1 é satisfeita.

Para exemplificar segue alguns exemplos de uso dos operadores nas propriedas e seus significados:

- $P = ?\ [\ F[0,t]fail_A\ ]$ - "A probabilidade do componente A falhar dentro de t instantes de tempo".
- $P < 0,01\ \ [\ \ X\ \ y = 1\ \ ]$ - "A probabilidade da expressão $y = 1$ ser verdadeira no próximo estado é menor que $0,01$".
- $P > 0,5\ \ [\ \ z > 2\ \ U\ \ z = 2\ \ ]$ - "A probabilidade de z ser eventualmente igual a 2 e que z permaneça menor que 2 até esse ponto, é maior que $0,5$".
- $S < 0,1\ \ [\ \ fail_A\ \ ]$ - "No longo prazo, a probabilidade do componente A falhar é menor que $0,1$".

É possível estender CSL com estruturas de Recompensas (*Rewards*) $R = (\rho, \iota)$ (KWIATKOWSKA et al., 2007) onde a função de recompensa do estado $\rho : S \to \mathbb{R}_{\geq 0}$ define a taxa em que cada recompensa é adquirida em um estado e a função de recompensa de transição $\iota : S \times S \to \mathbb{R}_{\geq 0}$ define a recompensa adquirida cada vez que uma transição ocorre. Um exemplo é dado abaixo:

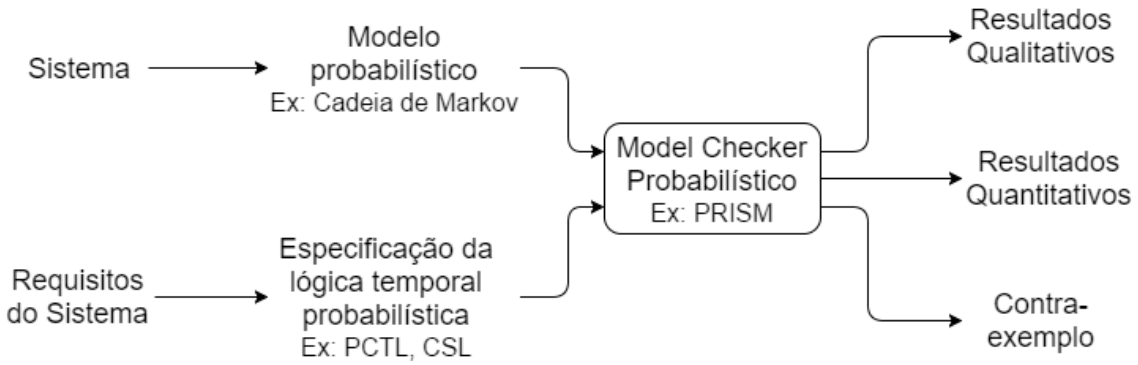
- $R\{\text{"}fail\text{"}\}=?\ [\ C < t\ ]$ - "O tempo esperado em estado de falha do sistema dentro do intervalo de tempo $[0, t]$".

Para a análise de dependabilidade desse trabalho foi utilizado o *Model Checker* (ferramenta) denominado PRISM (KWIATKOWSKA et al., 2009). O PRISM permite quatro tipos de modelos probabilísticos: DTMCs, CTMCs, MDPs e os autômatos probabilísticos temporizados (PTAs), além de duas linguagens para especificação das propriedades, PCTL e CSL. As estruturas de Recompensa podem ser utilizadas no PRISM e permitem a análise de diversas características do modelo por meio de valores reais associados aos estados ou transições. A Figura 2.12 mostra uma visão geral do processo relacionado a *Model Checking* Probabilístico.

2.7 Considerações finais sobre esse Capítulo

Esse capítulo apresentou a fundamentação teórica relacionada a essa dissertação de mestrado. Foram apresentados os conceitos e o estado da arte dos SEEs, mais especificamente os SEUs em FPGAs SRAM, tal qual as técnicas de mitigação de SEU em FPGAs: TMR, Scrubbing e código de Hamming. Em sequência foi mostrado o processo de desenvolvimento de FPGAs para o uso em ambiente espacial para lidar com SEUs e como o processo de determinação das taxas de *upsets* foi obtido para esse trabalho. Por fim, conceitos importantes de dependabilidade foram elicitados, tal qual os fundamentos principais de *Model Checking* Probabilístico.

Figura 2.12 - *Model Checking* Probabilístico.



Fonte: adaptado de Hoque (2016).

O próximo capítulo detalha como *Model Checking* Probabilístico foi utilizado para apoiar a mitigação de SEU em FPGAs.

3 MODEL CHECKING PROBABILÍSTICO PARA MITIGAÇÃO DE SEUS EM FPGAS

Esse capítulo apresenta a metodologia criada para analisar três técnicas de mitigação de SEU em FPGAs SRAM, TMR, *Scrubbing* e Código de Hamming, assim como os modelos CTMC desenvolvidos e as propriedades CSL abordadas (PEREIRA et al., 2017). A análise foi realizada considerando três atributos de dependabilidade, disponibilidade, confiabilidade e segurança, para dois estudos de caso da área espacial, os quais são descritos na seção 3.1.

3.1 Estudos de Caso

Para as análises foram definidos dois estudos de caso, o primeiro se baseou em (HOQUE, 2016) onde o autor utilizou uma Xilinx Virtex-5 em Órbita Elíptica Alta (OEA) para, por meio do CREME96, encontrar as taxas de SEUs/bit/segundo para executar os experimentos. Foi importante manter as características nominais de outro trabalho para garantir a integridade da comparação entre os resultados obtidos nos modelos implementados nesse trabalho.

E o segundo estudo de caso utilizou os parâmetros de órbita descritos na seção anterior para o satélite CBERS-4A[1] mantendo as características arquiteturais de uma FPGA Virtex-5 para o cálculo das taxas de SEUs/bit/segundo. Como já dito anteriormente, essa taxa é altamente dependente do dispositivo utilizado e, principalmente, da órbita de interesse, logo, os dois estudos de caso apresentam órbitas bem distintas e possibilitam uma variabilidade interessante para atingir o propósito desse trabalho.

3.2 Metodologia, Modelos CTMC e Propriedades CSL

Inicialmente é importante explicar a metodologia criada para alcançar o objetivo dessa pesquisa, a qual é apresentada na Figura 3.1. Primeiramente, foi feito um estudo sobre os princípios das técnicas de mitigação de SEU em FPGAs mais mencionadas na literatura (`Estudo das técnicas de mitigação de SEU em FPGAs`), no qual o objetivo principal era entender o problema, obter o estado da arte e verificar como a modelagem probabilística poderia se encaixar na solução. Após isso, foi feita a definição dos estudos de caso (`Definição dos estudos de caso`), etapa que engloba a definição de quais órbitas e dispositivos levar em consideração para estimar as taxas de SEU/bit/segundo e para quais técnicas o *Model Checking* Pro-

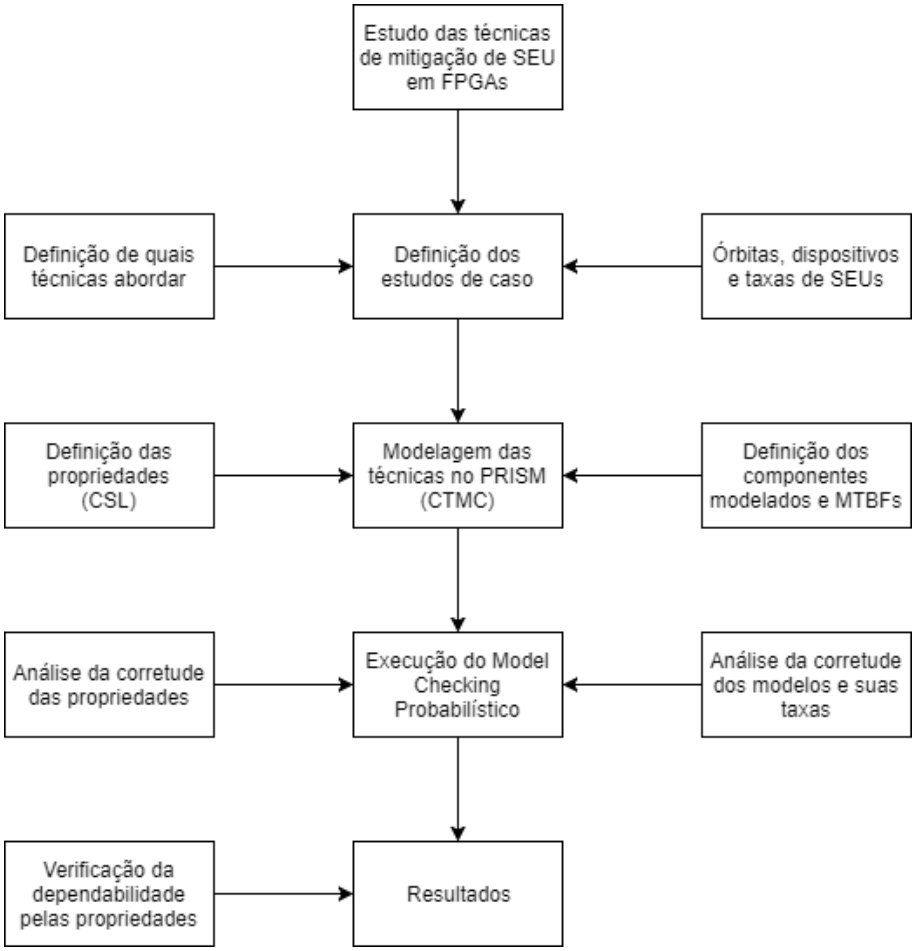
[1] `http://www.cbers.inpe.br/sobre_satelite/orbita_cbers4A.php`

babilístico deveria ser aplicado. Para definir o ambiente, o intuito foi cobrir duas órbitas bem distintas e verificar como a taxa de *upsets* varia conforme o tipo de radiação e características orbitais. Já o critério de definição das técnicas abordadas foi, além da frequência de uso na literatura, investigar técnicas com baixa similaridade ou que usem estratégias de mitigação distintas.

Com essas definições iniciais estabelecidas, o procedimento seguinte foi o de desenvolvimento dos modelos CTMC no PRISM (`Modelagem das técnicas no PRISM`), o que inclui a definição de quais componentes deveriam ser levados em consideração dentro do modelo, quais atributos de dependabilidade seriam aplicados nas propriedades entre outros detalhes como tempo de missão e intervalo de Scrubbing. Dada a relevância dessa etapa, o restante da seção trará mais detalhes de como essas características foram definidas e ajustadas para os estudos de caso.

Em seguida, após o desenvolvimento, os modelos foram executados para que o PRISM gerasse suas complexidades como o número de estados alcançáveis e transições (`Execução do Model Checking Probabilístico`), pois essa etapa é importante para encontrar problemas nos modelos como taxas incorretas e/ou estados inalcançáveis. Como já dito anteriormente, é imprescindível que os modelos desenvolvidos estejam dentro das características estipuladas para cada técnica, portanto o PRISM fornece recursos para a análise dos modelos como a navegação manual pelos estados (também é possível verificar as taxas de transição) e a exportação de uma versão gráfica do modelo que só é viável quando o número de estados é pequeno. Nesse mesmo estágio as propriedades foram colocadas em prática para verificar se os atributos de dependabilidade, descritos com detalhes mais à frente, estavam seguindo os conceitos corretos. Finalmente, após as correções necessárias, tanto nos modelos como nas propriedades, os resultados (`Resultados`) puderam ser gerados e analisados dentro do próprio PRISM através de experimentos, onde é possível gerar os gráficos de cada atributo de dependabilidade variando as características dos modelos.

Figura 3.1 - Metodologia de desenvolvimento aplicada no trabalho.



Fonte: produção do autor.

Essa etapa do trabalho se baseia em (HOQUE, 2016; HOQUE et al., 2014) onde os autores investigaram uma combinação do TMR com o *Scrubbing*. Os modelos CTMC representam as técnicas de mitigação de SEU em aplicações que usam somadores e multiplicadores, pois esses componentes são implementados na memória SRAM das FPGAs, portanto, sensíveis a *upsets*. Para o primeiro estudo de caso, as taxas de falha (λ) dos componentes foram estimadas em (HOQUE, 2016) por meio do CREME96 utilizando a taxa de *upset* por bit (Tu) para uma Xilinx Virtex-5 em uma Órbita Elíptica Alta (OEA), que é de $7,31\text{x}10^{-12}$ SEUs/bit/seg. Esta taxa é multiplicada pelo número de bits críticos (Nb) em cada componente e por 86400 segundos (1 dia) para converter a unidade de medida do tempo, como mostra a Equação 3.1. A Tabela 3.1 mostra uma descrição dos componentes utilizados nos

modelos TMR e *Scrubbing*, assim como os tempos médios entre as falhas (MTBFs) estimados para as taxas de falha. Consequentemente, é possível calcular o valor de λ como o inverso do MTBF para os somadores e multiplicadores, como mostra a Equação 3.2.

$$MTBF = Tu \times Nb \times 86400 \tag{3.1}$$

Tabela 3.1 - Caracterização dos componentes utilizados no primeiro estudo de caso (OEA).

Componente	**No. de LUT**s	**No. de bits**	**MTBF (dias)**
Multiplicador Wallace Tree	722	133503	11,85
Somador Kogge-Stone	183	41499	38,15

Fonte: Hoque (2016).

$$\lambda = \frac{1}{MTBF} \tag{3.2}$$
$$\lambda_{mult} = \frac{1}{11,85}$$
$$\lambda_{som} = \frac{1}{38,15}$$

As Equações 3.1 e 3.2 são utilizadas também no segundo estudo de caso (CBERS-4A), mas foi considerado relevante, para esse contexto, analisar não somente a ionização causada por *heavy ions* (HUP), mas também aquela causada pelos prótons (PUP). Por isso, a Tabela 3.2 indica dois MTBFs para cada componente, diferenciados pelo tipo de ionização. Nesse estudo de caso, a taxa de falha retornada pelo CREME96 para o HUP foi de $1,29\text{x}10^{-12}$ SEUs/bit/seg e para o PUP foi de $2,48\text{x}10^{-12}$ SEUs/bit/seg, considerando para ambos os casos o tempo de missão de 650 dias. É possível observar que o MTBF do componentes na órbita do CBERS-4A foi consideravelmente maior do que na OEA, já que o CREME96 atesta uma maior ocorrência de partículas ionizantes em órbitas mais altas.

Tabela 3.2 - Caracterização dos componentes utilizados no segundo estudo de caso (CBERS-4A).

Componente	**No. de bits**	**MTBF (HUP)**	**MTBF (PUP)**
Multiplicador Wallace Tree	133503	67,2 dias	34,96 dias
Somador Kogge-Stone	41499	216,2 dias	112,46 dias

3.2.1 Modelos CTMC

Essa seção descreve aspectos relacionados aos modelos CTMC criados para ambos os estudos de caso. Importante perceber que os códigos do *Model Checker* PRISM, que geram os modelos CTMC[2] para ambos os estudos de caso, possuem, basicamente, a mesma lógica mas com taxas de evento (λ) diferentes.

O uso de Cadeias de Markov para a modelagem de técnicas de mitigação de SEU em FPGAs de uso espacial é adequado pois, na prática, os eventos ocorrem de forma transiente, ou seja, não é possível prever quando nem quanto tempo vai levar até uma falha ocorrer. Essa característica de não-determinismo está presente em diversas aplicações práticas como sistemas de controle, protocolos de rede, reações químicas e interações biológicas. Portanto, para modelar esse tipo de aplicação é necessário que as transições do modelo ocorram em qualquer instante de tempo e que elas dependam apenas das informações contidas no estado atual, ou seja, não possuam memória. Nessas situações, os CTMCs são capazes de representar tais fenômenos, pois possuem tempo contínuo e são modelados utilizando distribuição exponencial, única distribuição contínua e sem memória existente.

Para o *Scrubbing*, o modelo foi disponibilizado em (HOQUE, 2016) mas, ao invés de analisar os resultados com somadores e multiplicadores limitados a três unidades de cada componente, o modelo desenvolvido lida com 10 somadores e 10 multiplicadores para avaliar a disponibilidade, segurança e confiabilidade do *Scrubbing* em uma aplicação mais complexa que requer mais componentes na configuração. Em resumo, o objetivo é confirmar se as conclusões apresentadas em (HOQUE, 2016) são válidas para aplicações mais complexas. No *Scrubbing*, existem apenas dois módulos, um para os somadores e outro para os multiplicadores, e a função desses módulos é controlar a quantidade de componentes operacionais e degradados. É importante

[2]De fato, assim como muitas ferramentas de *Model Checking*, o PRISM possui uma linguagem que permite escrever um código que, depois, é efetivamente transformado para uma representação computacional de um modelo formal probabilístico (DTMC, CTMC, ...). Nesse trabalho, o termo "modelo" é usado indistintamente onde pode significar tanto o código escrito na linguagem do *Model Checker* PRISM como a representação computacional do modelo probabilístico.

enfatizar que em (HOQUE, 2016), o modelo CTMC não é suficientemente detalhado para considerar os dados de entrada e saída dos somadores e multiplicadores: o modelo basicamente lida com o funcionamento ou o não-funcionamento dos elementos.

A Figura 3.2 demonstra um trecho do código PRISM para o Scrubbing onde é possível notar o módulo multiplicador, na qual segue-se a mesma lógica para o módulo somador. A constante `num_M` indica a quantidade inicial de multiplicadores, o `repair` indica a taxa de reparo do Scrubbing que pode ser 1, 4 ou 9 dias, a constante `c` representa a taxa de cobertura e o `lambda_M` é a taxa de falha estipulada para os multiplicadores.

Figura 3.2 - Módulo multiplicador para o modelo Scrubbing desenvolvido no PRISM.

```
const int num_M=10;
const double repair=9;
const double c=1;
const double lambda_M = 1/11.85;

module mult
  m : [0..num_M+1] init num_M;
  [] (m > 0 & (m < num_M+1)) -> c*m*lambda_M : (m'=m-1)
                              + m*(1-c)*lambda_M : (m'= num_M+1);
  [rep] (m >= 0) -> 1 : (m'= num_M);
endmodule
```

Fonte: adaptado de Hoque (2016).

Como o modelo aborda o *Blind Scrubbing*, não existe uma técnica de detecção de faltas e o reparo é efetuado independentemente do número de componentes operacionais. O intervalo de *Scrubbing* estipulado em (HOQUE, 2016) de 1, 4 e 9 dias ocorre sincronizadamente entre os somadores e multiplicadores. Para a análise do modelo, as condições dos componentes foram divididas em quatro categorias: operacional, onde todos os componentes estão funcionais; degradado, onde pelo menos um dos componentes falhou; falha segura, onde todos os somadores e multiplicadores sofreram um *upset* e não podem funcionar, mas todas as falhas foram corretamente detectadas; e falha insegura, onde pelo menos um dos componentes falhou mas o sistema não foi capaz de detectar, tornando todo o sistema vulnerável. Para classificar os estados dos modelos CTMC, o PRISM oferece o conceito de fórmulas, como

mostra a Figura 3.3. Para representar a probabilidade de detectar um componente degradado existe a taxa de cobertura dada por (HOQUE, 2016), que é definida como a probabilidade condicional C:

$C = P(detectar\ a\ falta | falta\ existe)$.

Figura 3.3 - Representação das fórmulas para o modelo Scrubbing desenvolvido no PRISM.

```
formula fail_unsafe = ((a=num_A+1) | (m=num_M+1));
formula fail_safe = ((a=0)|(m=0))& !fail_unsafe;
formula oper = (a=num_A) & (m=num_M);
formula degrade = !fail_safe & !fail_unsafe & !oper;
```

Fonte: adaptado de Hoque (2016).

O modelo CTMC em (HOQUE, 2016) não implementa, fielmente, um TMR. Em sua análise, o autor utilizou um modelo de *Scrubbing* e, para representar o TMR, adicionou componentes sobressalentes mas desprezou a estrutura e o funcionamento do TMR, visto que não incluiu o mecanismo de votação por maioria nem os dados de entrada e saída de cada componente. Além de uma modelagem superficial da técnica, o TMR não foi analisado individualmente, apenas combinado ao *Scrubbing*. Por isso, nesse trabalho foi desenvolvido um modelo TMR próprio e mais coerente com a definição conceitual dessa técnica.

No *Model Checker* PRISM, um modelo é composto por um ou mais módulos, onde cada módulo tem um conjunto de comandos responsáveis por fazer as transições necessárias que definem o modelo probabilístico, e podem conter diversas variáveis que representarão os estados do modelo. Uma transição de estado, representada pela atualização no valor de uma variável, é efetuada em uma certa taxa (`prob_-1..n`) constituída por um número real e positivo quando uma condição de guarda é satisfeita, como na Figura 3.4:

Figura 3.4 - Formato de um comando do PRISM para representar as transições de estado.

```
[] (guarda) -> prob_1 : (atualiza_1) + ... + prob_n : (atualiza_n);
```

Fonte: produção do autor.

O código desenvolvido no PRISM para o TMR tem oito módulos: seis representam as redundâncias triplas dos somadores e multiplicadores, e os restantes equivalem aos dois votadores, um para os somadores e outro para os multiplicadores. A Figura 3.5 ilustra o código PRISM para o primeiro somador, a mesma lógica foi aplicada aos multiplicadores. Observe que `a1` e `a2` representam as entradas de 4 bits, `outA1` representa a saída do módulo, `Na` é o número de somadores operacionais e `lambda_A` é a taxa de falha estipulada para os somadores. A linguagem do PRISM permite a sincronização entre módulos e esta função foi utilizada no modelo CMTC. Diferentemente do modelo desenvolvido em (HOQUE, 2016), esta solução é mais refinada pois representa explicitamente as entradas (4 bits para os somadores e multiplicadores) e saídas (5 bits para os somadores e 8 bits para os multiplicadores), assim como os módulos que realmente representam o funcionamento de um mecanismo de votação. O cálculo das taxas de evento em comandos sincronizados, mesmo que estejam em módulos diferentes, é o produto das taxas estipuladas para cada comando e, por isso, foi utilizada a abordagem passiva/ativa (*race condition*), ou seja, taxa λ em um módulo e 1 nos outros módulos para que conforme o modelo seja checado, as taxas não tenham seu valor estipulado alterado.

Figura 3.5 - Módulo somador para o modelo TMR desenvolvido no PRISM.

```
module adder1
  a1 : [0..15] init 0;
  a2 : [0..15] init 0;
  outA1 : [0..31] init 31;
  [input] (a1=0 & a2=0) -> 1: (a1'=5) & (a2'=7);
  [adder] ((a1 > 0 | a2 > 0) & (Na > 0)) -> Na*(1-lambda_A) :
  (outA1'= a1+a2) + Na*lambda_A : (outA1'= a1+a2-1);
endmodule
```

Fonte: produção do autor.

Inicialmente, todos os componentes estão operacionais e recebem dois valores de entrada de 4 bits. Baseado na taxa de falha estipulada, cada módulo pode funcionar de forma apropriada e atribuir o valor correto à saída ou ter um *upset* e retornar o valor incorreto. A taxa de falha é multiplicada pelo número de componentes operacionais, assim os componentes degradados não serão considerados no próximo ciclo. Com os valores de saída retornados pelos módulos, os votadores entram em ação e decidem qual será a saída final baseando-se no valor da maioria. Assim, o TMR é capaz de ignorar a ocorrência de um *upset* pois os outros dois módulos terão o valor correto de saída. No caso de dois *upsets* simultâneos, o votador considerará o valor errado de saída como maioria e consequentemente o resultado será incorreto.

O TMR desenvolvido foi dividido em três estados: operacional, onde todos os componentes estão funcionais; degradado, onde no máximo um componente entre somadores e multiplicadores sofreu um *upset*, mas a saída continua correta; e falho, onde mais de um componente sofreu *upset* e a saída gerada pelo votador está incorreta. As representações das fórmulas estipuladas para o TMR são mostradas na Figura 3.6:

Figura 3.6 - Representação das fórmulas para o modelo TMR desenvolvido no PRISM.

```
formula fail = ((outMf!=m1*m2) | (outAf!=a1+a2));
formula oper = ((outAf=outA1) & (outAf=outA2) & (outAf=outA3) &
                (outMf=outM1) & (outMf=outM2) & (outMf=outM3) &
                (outAf=a1+a2) & (outMf=m1*m2));
formula degrade = !fail & !oper;
```

Fonte: produção do autor.

Além do *Scrubbing* e TMR, uma terceira abordagem de mitigação de SEU foi considerada nesse trabalho. O modelo desenvolvido para o código de Hamming não está diretamente relacionado aos somadores e multiplicadores, pois pode ser implementado para detectar defeitos em qualquer componente de memória das FPGAs SRAM. Para representar seu funcionamento, dois módulos, um codificador e um decodificador foram criados, onde o codificador recebe uma entrada de 4 bits e, baseado nessas informações, calcula os 3 bits de paridade adicionais. Um trecho do módulo codificador está exposto na Figura 3.7, onde o bit de paridade recebe o valor 1 quando o número de bits de entrada com valor 1 for ímpar e 0 quando o número de bits de entrada com valor 0 for par. Já a detecção e correção de um possível defeito é executada pelo decodificador, que pode receber os bits inalterados do codificador ou inserir uma falta em qualquer bit baseado na taxa de falha, como no trecho de código mostrado na Figura 3.8.

Como se trata de um modelo genérico, a taxa de falha do código de Hamming é dependente da sensibilidade do modelo de FPGA e da arquitetura do componente protegido. Para uma comparação justa entre as técnicas e considerando que o código de Hamming está lidando com *upsets* nos mesmos componentes, as taxas de falha escolhidas seguem os modelos anteriores com um MTBF de 10 dias para o primeiro estudo de caso (OEA), 30 dias para o PUP e 60 dias para o HUP da órbita do CBERS-4A, dessa forma garantindo que os resultados representem um cenário onde a incidência de *upsets*, para o Código de Hamming, é maior que a estipulada para os somadores e multiplicadores no TMR e *Scrubbing* em seus respectivos estudos de caso.

Figura 3.7 - Trecho do módulo codificador para o modelo de código de Hamming desenvolvido no PRISM.

```
module encoder
  ib1 : [0..1] init 1;
  ib2 : [0..1] init 1;
  ib3 : [0..1] init 1;
  ib4 : [0..1] init 1;

  pb1 : [0..1] init 0;
  pb2 : [0..1] init 0;
  pb3 : [0..1] init 0;

  [] (ib1+ib2+ib4 = 1 | ib1+ib2+ib4 = 3) -> 1 : (pb1' = 1);
  [] (ib1+ib2+ib4 = 0 | ib1+ib2+ib4 = 2) -> 1 : (pb1' = 0);
endmodule
```

Fonte: produção do autor.

Figura 3.8 - Trecho do módulo decodificador para o modelo de código de Hamming desenvolvido no PRISM.

```
module decoder
  ob1 : [0..1] init 0;
  ob2 : [0..1] init 0;
  ob3 : [0..1] init 0;
  ob4 : [0..1] init 0;

  [] (ob1 >=0 & ob2 >= 0 & ob3 >= 0 & ob4 >= 0) -> (1-lambda_H) :
            (ob1' = ib1) & (ob2' = ib2) & (ob3' = ib3) & (ob4' = ib4)
                                                    + lambda_H :
            (ob1' = ib1) & (ob2' = ib2) & (ob3'= ib3) & (ob4' = 1-ib4);
endmodule
```

Fonte: produção do autor.

Quando o decodificador verifica os bits de paridade e de entrada e não detecta nenhuma alteração, o valor é passado para a saída normalmente, mas se os valores dos bits de paridade não correspondem com a entrada, o decodificador identifica

a posição do *upset* e restaura o valor correto do bit. Assim como no TMR, as fórmulas foram utilizadas para classificar os estados: operacional, indicando que os bits de saída do decodificador coincidem com os bits de entrada do codificador; degradado, que representa os estados que detectaram um defeito e estão em processo de correção; e falho, para o intervalo entre a ocorrência de um *upset* e a sua detecção pelo decodificador.

A Tabela 3.3 mostra as principais estatísticas geradas pelo PRISM para cada um dos modelos abordados, como a quantidade de estados alcançáveis, de transições e de nós na matriz de taxas.

Tabela 3.3 - Estatísticas geradas pelo PRISM dos modelos CTMC.

Técnica modelada	**No. de estados**	**No. de transições**	**No. de nós**
Scrubbing: 2A 2M	16	48	68
Scrubbing: 10A 10M	144	624	481
TMR	3233	28768	4275
Código de Hamming	21	24	343

3.2.2 Propriedades CSL

Para a análise de disponibilidade, o conceito de Recompensas dado pelo PRISM garante que a cada transição de estado será verificado em qual das fórmulas estipuladas o estado atual pertence e então será adicionado ao tempo total em cada uma das condições existentes. Por exemplo, se o estado atual do modelo pertence à fórmula operacional, uma recompensa de tempo operacional será somada ao modo correspondente. Dessa forma, ao final da execução do modelo, os tempos operacional, degradado e falho somados deverão resultar no tempo total de missão. Dado que a disponibilidade é o atributo que está relacionado à razão de tempo em que o sistema está operando corretamente e o tempo de missão, as Recompensas expressam de forma adequada o resultado dessa propriedade. Contudo, o retorno da análise de Recompensas cumulativa nesse trabalho é o número de dias, portanto uma forma adicional de analisar probabilisticamente a disponibilidade foi o uso da probabilidade limite. O operador S permite analisar o comportamento do modelo no longo prazo, nesse caso, o objetivo é obter a probabilidade do sistema falhar independente do tempo de missão.

Para formalizar os atributos de confiabilidade e segurança foi utilizado o operador P, um dos mais importantes da linguagem do PRISM. A função desse operador é

retornar a probabilidade de ocorrer um evento e que, no contexto desse trabalho, está limitado ao tempo t de missão. A diferença entre as propriedades que utilizam o mesmo operador mas descrevem atributos distintos está no uso das fórmulas, ou seja, a confiabilidade descreve a probabilidade de um sistema estar funcionando de forma adequada, portanto apenas as fórmulas de estado operacional ou degradado entram na descrição da propriedade. Já a segurança é definida como a probabilidade do sistema estar funcionando de forma adequada ou se falhar, conseguir detectar a ocorrência da falha e retirar o componente atingido de operação. Portanto, além das mesmas fórmulas operacional e degradado, também foi incluída a fórmula de falha segura, que ocorre justamente quando o sistema é capaz de detectar uma falha corretamente.

Cada propriedade utilizada está descrita na Tabela 3.4, onde percebe-se as fórmulas em CSL estendida com Recompensas (R), o atributo de dependabilidade relacionado, e seu significado de maneira informal. Observe que uma vez que apenas no *Scrubbing* existe a possibilidade de uma falta ocorrer e não ser detectada, apenas esta técnica teve sua segurança analisada. Também foi considerado que tanto no TMR quanto no código de Hamming, a ocorrência de uma falta será sempre detectada pelo votador e pelo decodificador, respectivamente. As definições das propriedades CSL e a relação de cada propriedade com seu atributo foi dada em (HOQUE, 2016), com exceção da probabilidade limite de falha que foi relacionada ao atributo disponibilidade e, dessa forma, é uma contribuição desse trabalho.

Tabela 3.4 - Propriedades formalizadas em CSL estendida com Recompensas

Atributo	**Propriedade**	**Significado**
Disponibilidade	$R\{\text{“}timeOperational\text{”}\}=?\ [\ C <= t\ ]$	Tempo acumulado no modo operacional dentro do intervalo $[0, t]$.
Disponibilidade	$R\{\text{“}timeDegraded\text{”}\}=?\ [\ C <= t\ ]$	Tempo acumulado no modo degradado dentro do intervalo $[0, t]$.
Disponibilidade	$R\{\text{“}timeFailure\text{”}\}=?\ [\ C <= t\ ]$	Tempo acumulado no modo falho dentro do intervalo $[0, t]$.
Disponibilidade	$S=?\ [\ fail\ ]$	A probabilidade limite do sistema falhar.
Confiabilidade	$P=?\ [\ G[0,t]\ oper \vert degrade\ ]$	A probabilidade do sistema estar operacional ou degradado nos primeiros t dias.
Segurança	$P=?\ [\ G[0,t]\ oper \vert degrade \vert fail_{safe}\ ]$	A probabilidade do sistema estar operacional, degradado ou em falha segura nos primeiros t dias.

Fonte: adaptado de Hoque (2016).

3.3 Resultados e Discussão

Essa seção apresenta os resultados obtidos via Model Checking Probabilístico para os dois estudos de caso exibidos nesse trabalho. Os resultados foram comparados entre mesmas técnicas com características diferentes mas também entre técnicas distintas. Os atributos de dependabilidade analisados foram: disponibilidade, confiabilidade e segurança.

3.3.1 Órbita Elíptica Alta

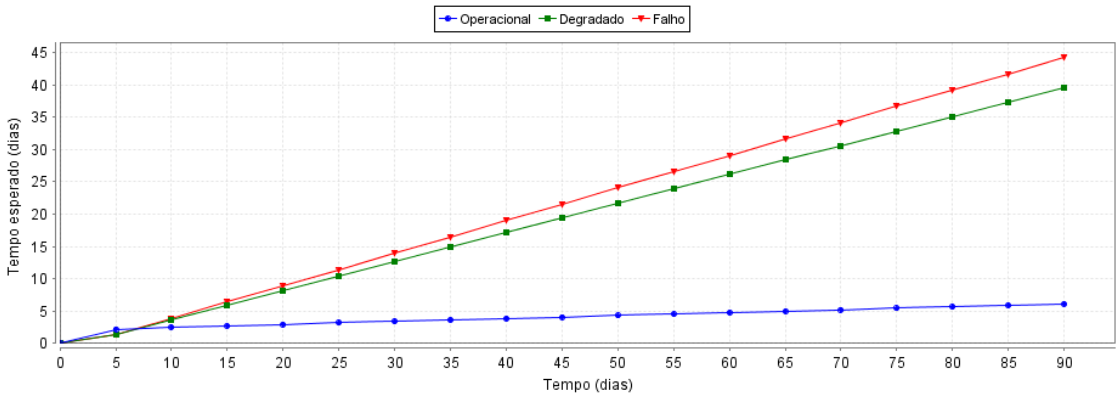
Esse estudo de caso apresenta as taxas de ocorrência de *upset* definidas em (HOQUE, 2016) para uma FPGA Virtex-5 em Órbita Elíptica Alta (OEA) com um tempo de missão de 90 dias (PEREIRA et al., 2017). As seções seguintes descrevem os resultados obtidos e uma análise dos mesmos considerando três categorias de funcionamento dos sistemas: operacional, degradado e falho.

3.3.1.1 Análise de Disponibilidade

Para esta análise, foi considerado um tempo de missão de 90 dias para verificar quanto tempo o modelo, relacionada a cada técnica de mitigação de SEU, estará em estado operacional, degradado e falho. A Figura 3.9 mostra os resultados de Recompensas obtidos com o TMR, onde, com aproximadamente metade do tempo de missão em estado de falha, demonstra que uma técnica de mitigação, implementada individualmente sem um método de correção capaz de restaurar o funcionamento dos componentes degradados, pode causar problemas em um curto período de tempo.

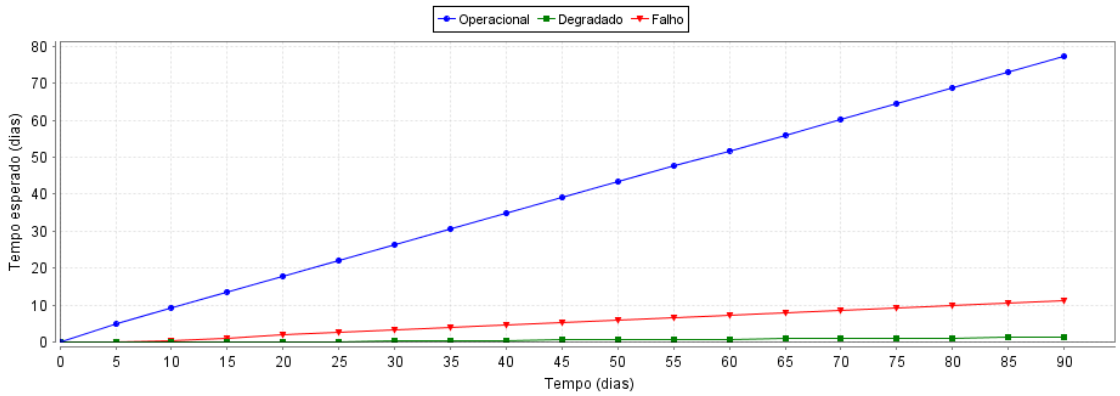
Analisando as mesmas propriedades para o modelo do código de Hamming, a Figura 3.10 ilustra um resultado melhor mesmo com a taxa de falha mais alta que a utilizada no TMR. Com mais de 80% do tempo total de missão em estado operacional, o código de Hamming se torna uma técnica de correção interessante para sistemas com baixa incidência de MBUs. Uma das suas desvantagens é que, se o número de bits protegidos é muito grande e diversos componentes de memória implementam os blocos de codificador e decodificador, o impacto no desempenho da FPGA pode ser considerável. Uma opção para contornar esse problema é implementar o TMR e o código de Hamming em conjunto, desta forma a correção só será executada em caso de acúmulo de *upsets*.

Figura 3.9 - Análise de disponibilidade do TMR: missão de 90 dias.



Fonte: produção do autor.

Figura 3.10 - Análise de disponibilidade do código de Hamming: missão de 90 dias.

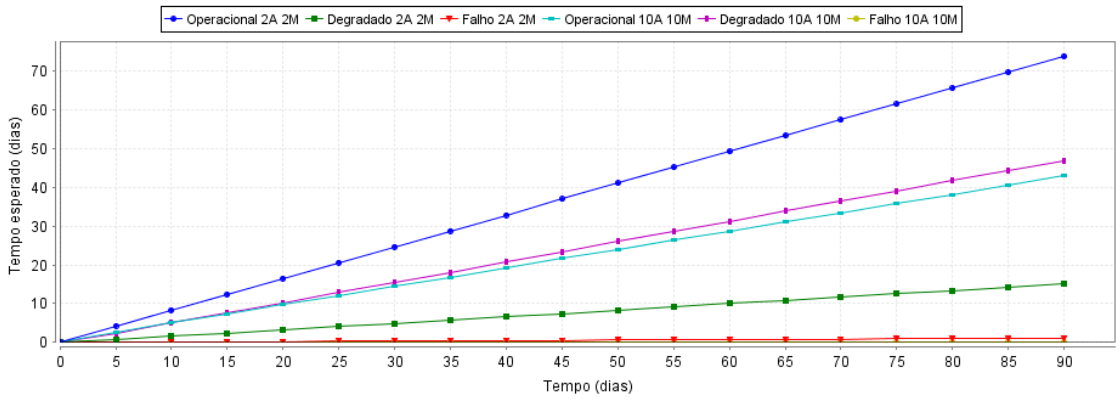


Fonte: produção do autor.

Os resultados de Recompensas obtidas com o *Scrubbing* utilizando 10 somadores e 10 multiplicadores (contribuição desse trabalho) foram comparados diretamente com os resultados apresentados em (HOQUE, 2016), que considerou 2 somadores e 2 multiplicadores. A Figura 3.11 mostra que, para o intervalo de 1 dia, os resultados apresentados pelo modelo com menos componentes ((HOQUE, 2016)) pode manter o sistema em estado operacional por um período de tempo maior, enquanto a configuração com 10 somadores e 10 multiplicadores passa mais tempo degradada do que operacional, mas se mantém funcional quase 100% do tempo. A taxa de cobertura

utilizada em ambos os modelos foi de 99%.

Figura 3.11 - Análise de disponibilidade do *Scrubbing* com intervalo de 1 dia: 2A 2M = apresentado em (HOQUE, 2016); 10A 10M = contribuição desse trabalho.

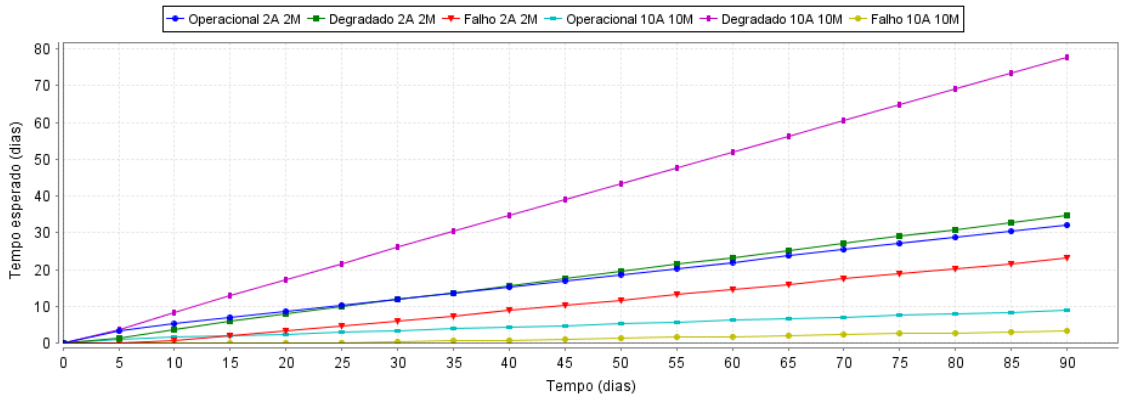


Fonte: produção do autor.

A situação se inverte quando o intervalo de *Scrubbing* é aumentado para 9 dias pois, mesmo degradada, a configuração de 10 somadores e 10 multiplicadores pode manter o sistema funcionando durante grande parte do tempo de missão. Para a configuração apresentada em (HOQUE, 2016), o tempo em que o sistema fica falho sobe consideravelmente, e pode se tornar inaceitável para sistemas que exigem alta disponibilidade, como mostra a Figura 3.12.

Com esses resultados, é possível inferir que em modelos com uma área menor, ou seja, com menos componentes suscetíveis a *upsets*, um intervalo de *Scrubbing* muito grande pode se tornar um problema, pois os *upsets* se acumularão e provavelmente nenhum componente permanecerá operacional. Com um número grande de componentes, um intervalo maior de *Scrubbing* se torna aceitável, pois mesmo se os *upsets* se acumularem, os componentes operacionais serão sobrecarregados mas, mesmo assim, capazes de manter o sistema funcional. O acúmulo de *upsets* ocorre quando mais de um componente é atingido dentro do mesmo intervalo de *Scrubbing*, e como não há detecção (*Blind Scrubbing*), as correções ocorrerão somente ao final do intervalo estipulado.

Figura 3.12 - Análise de disponibilidade do *Scrubbing* com intervalo de 9 dias: 2A 2M = apresentado em (HOQUE, 2016); 10A 10M = contribuição desse trabalho.



Fonte: produção do autor.

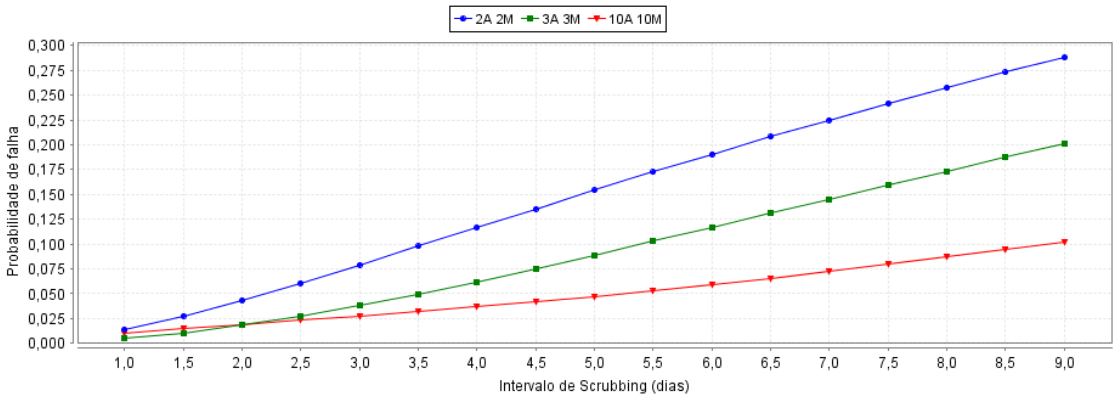
Considerando a outra perspectiva de disponibilidade, a probabilidade limite de falha das técnicas TMR e código de Hamming, a Tabela 3.5 mostra os valores obtidos em longo prazo. Novamente, devido à falta de uma técnica capaz de prevenir o acúmulo de *upsets*, o TMR apresentou uma probabilidade alta de falha.

Tabela 3.5 - Probabilidade limite de falha do TMR e do código de Hamming.

Técnica de mitigação de SEU	**Probabilidade de falha**
TMR	50.5%
Código de Hamming	13.5%

Para o *Scrubbing*, a probabilidade limite de falha depende diretamente do intervalo escolhido, como mostra a Figura 3.13. Para um intervalo pequeno, os resultados são muito similares entre as configurações mas, conforme o intervalo aumenta, a quantidade de componentes se torna inversamente proporcional à probabilidade de falha. Em outras palavras, quanto mais componentes no sistema, menor a chance de todos eles sofrerem um *upset* dentro do mesmo intervalo de *Scrubbing*.

Figura 3.13 - Probabilidade limite de falha de *Scrubbing*: 2A 2M e 3A 3M = apresentado em (HOQUE, 2016); 10A 10M = contribuição desse trabalho.



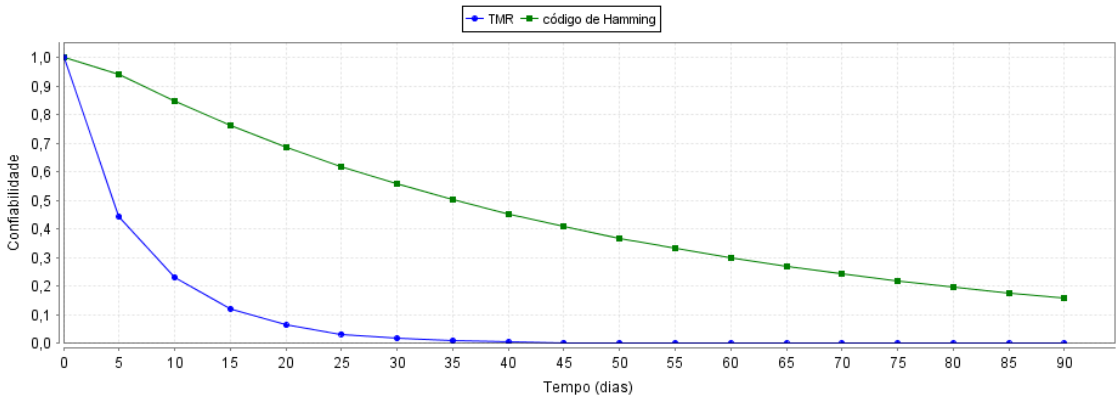
Fonte: produção do autor.

3.3.1.2 Análise de Confiabilidade e Segurança

Na análise da confiabilidade, que é a probabilidade do sistema estar operacional ou degradado nos primeiros 90 dias, o código de Hamming apresentou um resultado superior comparado ao TMR: em metade do tempo de missão, o TMR já estava sem nenhuma chance de permanecer sem faltas. A Figura 3.14 mostra os resultados obtidos na análise e, apesar da superioridade do código de Hamming, pode-se afirmar que ambos os resultados não são satisfatórios.

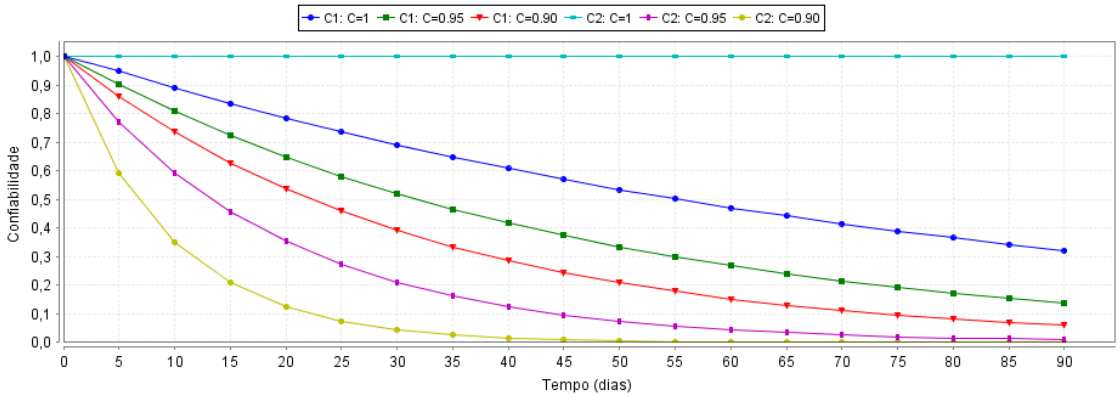
Assim como antes, a análise da confiabilidade do *Scrubbing* foi dividida em dois intervalos: 1 dia e 9 dias. De agora em diante, a configuração apresentada em (HOQUE, 2016) com 2 somadores e 2 multiplicadores será chamada de C1 enquanto a configuração de 10 somadores e 10 multiplicadores será C2. A Figura 3.15 representa a confiabilidade do *Scrubbing* para C1 e C2 variando apenas a taxa de cobertura entre 100% e 90% em uma missão de 90 dias, e com um intervalo de *Scrubbing* de 1 dia. A confiabilidade máxima foi obtida em C2 com 100% de taxa de cobertura, mas é possível observar que a redução da taxa de cobertura afetou mais drasticamente a confiabilidade de C2 em ambos os valores: 95% e 90%. Isso mostra que quanto mais componentes no sistema, se torna mais importante manter a cobertura mais próxima possível de 100%.

Figura 3.14 - Confiabilidade do TMR e do código de Hamming: missão de 90 dias.



Fonte: produção do autor.

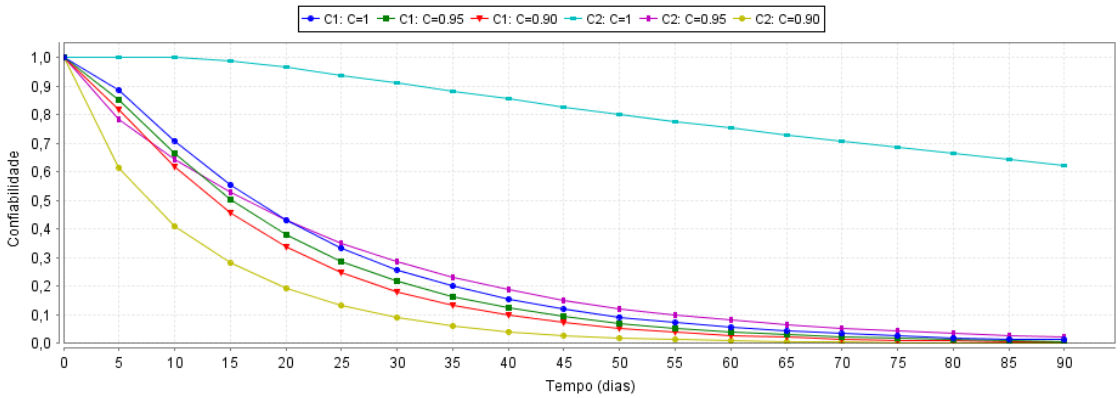
Figura 3.15 - Confiabilidade do *Scrubbing* com 1 dia de intervalo: missão de 90 dias.



Fonte: produção do autor.

Mantendo as mesmas propriedades mas aumentando o intervalo de *Scrubbing* para 9 dias, com exceção de C2 com 100% de taxa de cobertura, todas as outras configurações perderam sua confiabilidade até o fim dos 90 dias de missão, como mostra a Figura 3.16. Isto indica o quanto o intervalo de *Scrubbing* pode influenciar no funcionamento correto do sistema, especialmente em sistemas com menos componentes.

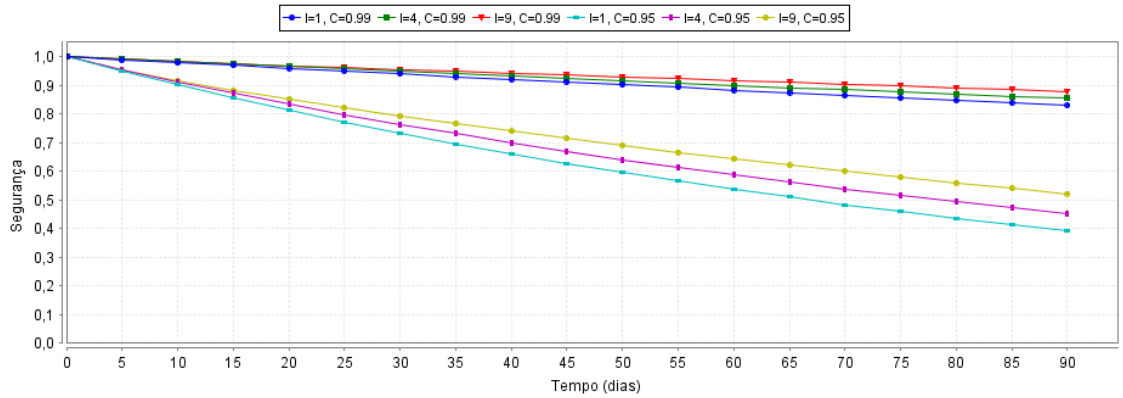
Figura 3.16 - Confiabilidade do *Scrubbing* com 9 dias de intervalo: missão de 90 dias.



Fonte: produção do autor.

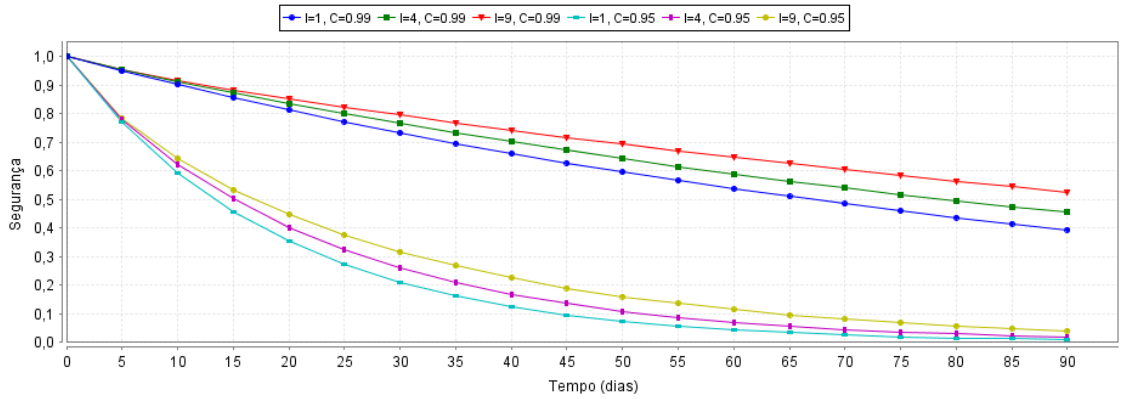
A segurança de um sistema que tem um método de mitigação como o *Blind Scrubbing* está relacionada com a probabilidade de detectar um componente defeituoso assim que a falta ocorre, ou seja, se relaciona com a taxa de cobertura. No modelo proposto por (HOQUE, 2016), um componente degradado que não foi retirado do conjunto de componentes operacionais leva imediatamente o sistema para o estado de falha insegura. As Figuras 3.17 e 3.18 mostram como a diminuição da cobertura de 99% para 95% reduziu drasticamente a segurança em ambas as configurações C1 e C2 mesmo variando o intervalo de Scrubbing (I) entre 1, 4 e 9 dias. Esse resultado indica que para sistemas que valorizam a segurança, garantir uma taxa de cobertura alta garante melhores resultados do que alterações no intervalo de *Scrubbing*. Além disso, C1 foi capaz de preservar melhor a segurança do que C2 em ambas as taxas de cobertura, mostrando que sistemas com uma quantidade menor de componentes diminuem as chances de faltas passarem despercebidas e tornarem o sistema inseguro.

Figura 3.17 - Segurança do *Scrubbing* com 2 somadores e 2 multiplicadores (C1): missão de 90 dias.



Fonte: produção do autor.

Figura 3.18 - Segurança do *Scrubbing* com 10 somadores e 10 multiplicadores (C2): missão de 90 dias.



Fonte: produção do autor.

3.3.2 CBERS-4A

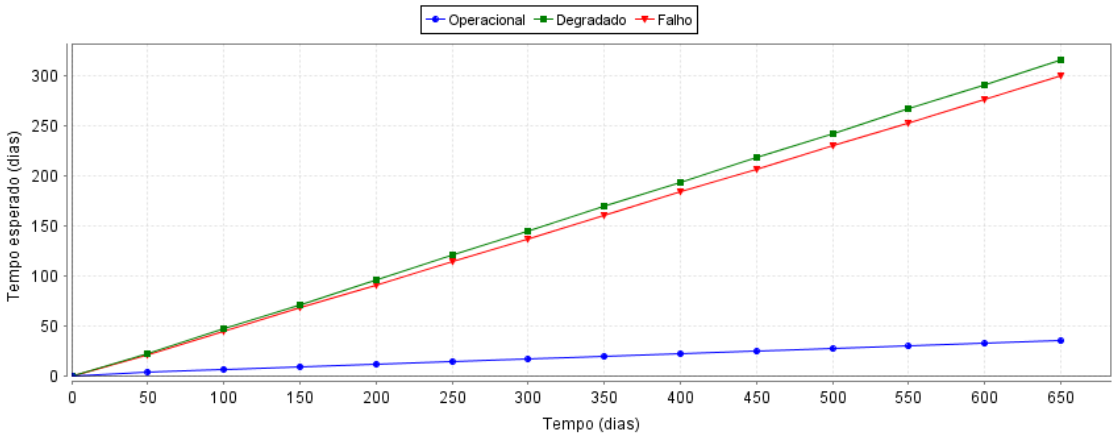
Para esse estudo de caso, foi considerado um tempo de missão de 650 dias e, novamente, analisados os tempos em que cada técnica modelada esteve operacional, degradada e em falha. Além da abordagem de comparação entre as técnicas que foi dada no estudo de caso anterior, nesse caso é interessante comparar também o desempenho das técnicas considerando ionização direta (HUP) e ionização indireta

(PUP). Por esse motivo, os gráficos que ilustram a mesma técnica submetida às mesmas condições de análise, variando apenas qual taxa foi utilizada, serão colocadas em sequência para facilitar o entendimento, haja vista que uni-las num mesmo gráfico perturbaria a visualização do mesmo.

3.3.2.1 Análise de Disponibilidade

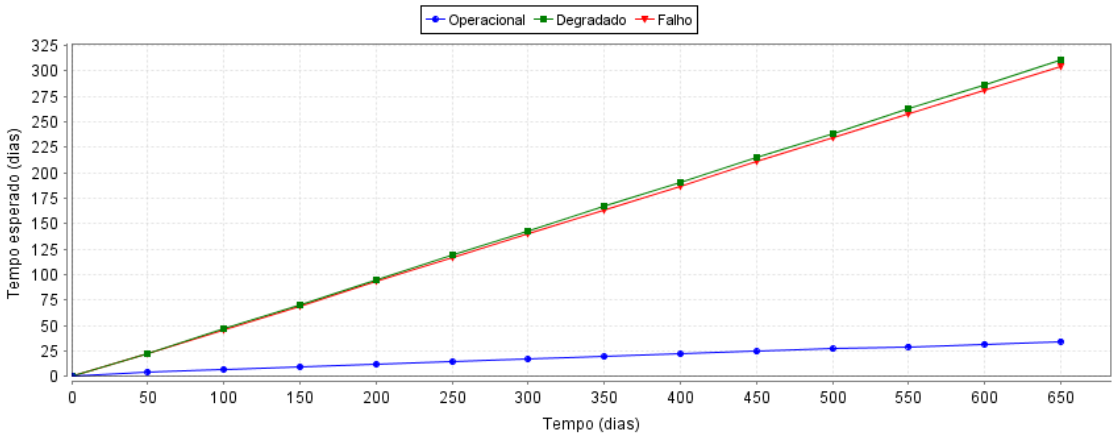
Assim como na seção anterior, a primeira análise envolve a disponibilidade das técnicas dividida em três categorias: operacional, degradado e falho. As Figuras 3.19 e 3.20 mostram os resultados obtidos com o TMR, onde novamente o desempenho foi insatisfatório tanto no HUP quanto no PUP. Isso mostra que mesmo com taxas de *upset* e tempos de missão diferentes, os resultados apresentados refletem a limitação de uma técnica de redundância perante o acúmulo de *upsets*.

Figura 3.19 - Análise de disponibilidade do TMR: missão de 650 dias (HUP).



Fonte: produção do autor.

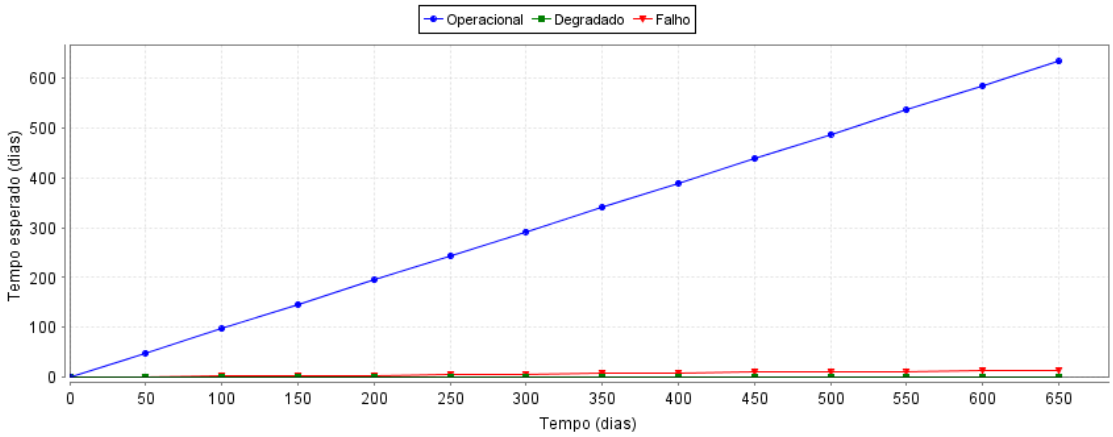
Figura 3.20 - Análise de disponibilidade do TMR: missão de 650 dias (PUP).



Fonte: produção do autor.

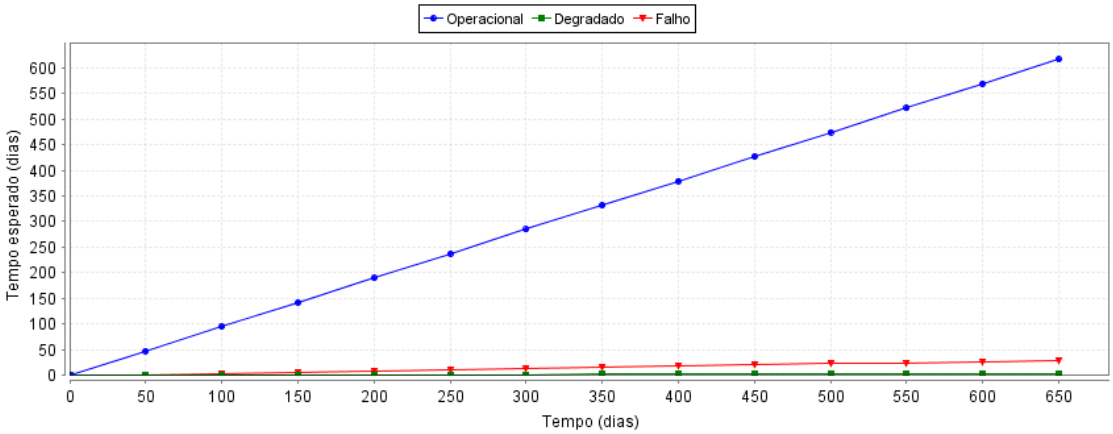
As Figuras 3.21 e 3.22 analisam o mesmo atributo no modelo do código de Hamming, onde os resultados também seguiram o que foi apresentado no estudo de caso anterior. Mesmo com taxas de falha maiores em relação ao TMR, mas como o tempo de missão foi muito maior, a proporção de tempo operacional também subiu consideravelmente. É importante ressaltar que diferente das outras técnicas de mitigação modeladas, o código de Hamming sempre detecta e corrige um *upset* desde que ele não ocorra em mais de um bit ao mesmo tempo. Portanto, o tempo degradado e falho representa os intervalos de tempo em que o modelo está em fase de detecção e correção do bit incorreto.

Figura 3.21 - Análise de disponibilidade do código de Hamming: missão de 650 dias (HUP).



Fonte: produção do autor.

Figura 3.22 - Análise de disponibilidade do código de Hamming: missão de 650 dias (PUP).

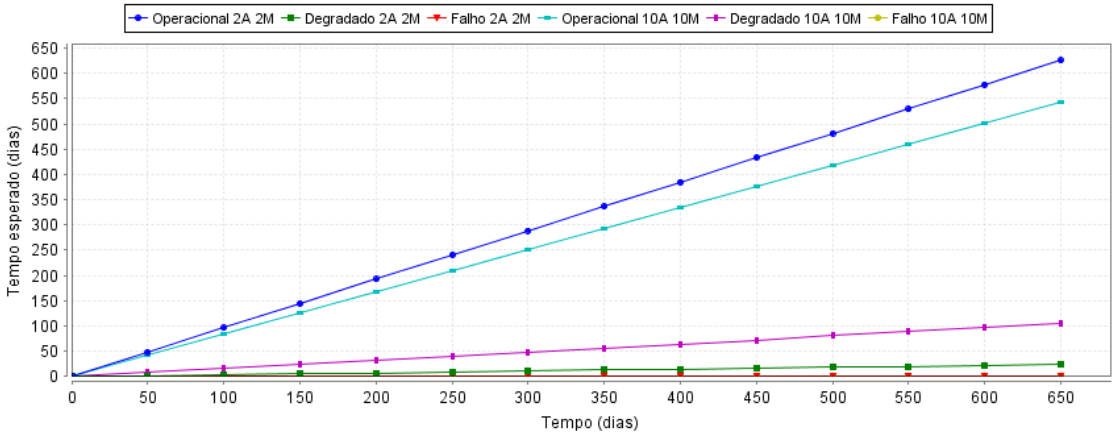


Fonte: produção do autor.

No caso do Scrubbing, as Figuras 3.23 e 3.24 ilustram as comparações entre as duas configurações estipuladas nesse trabalho, C1 com 2 somadores e 2 multiplicadores e C2 com 10 somadores e 10 multiplicadores. Pela primeira vez, foi possível notar uma mudança relevante entre os resultados apresentados em ambos os estudos caso, apesar de C1 continuar superior a C2 no estado operacional, C2 se destacou em relação ao resultado apresentado no estudo de caso anterior por chegar mais perto

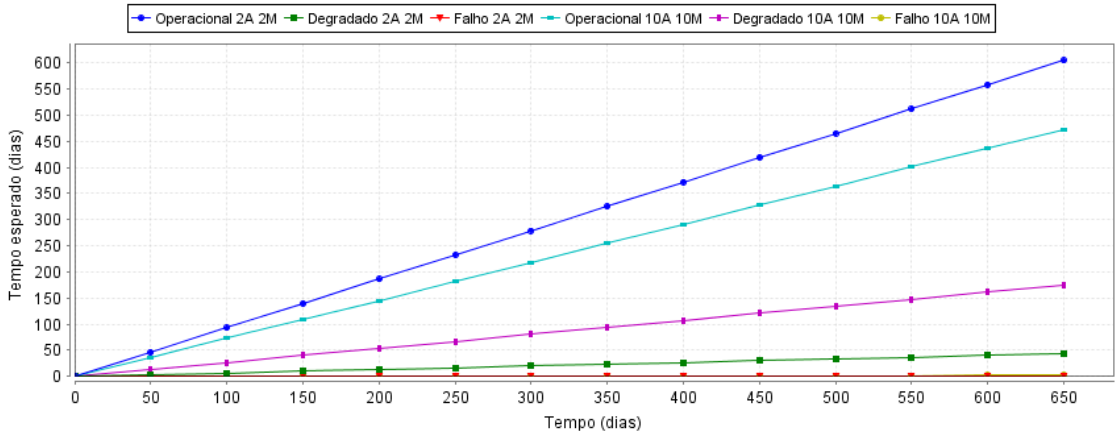
de C1 no tempo operacional. É possível atribuir essa mudança ao tempo de missão maior, que tornou a proporção do tempo degradado e falho menor, pois o intervalo de Scrubbing continua de 1 dia como no estudo anterior. Em uma comparação direta entre os resultados das diferentes formas de ionização, é possível notar como a taxa maior de *upsets* apresentada por PUP interfere no tempo operacional ao final dos 650 dias de missão.

Figura 3.23 - Análise de disponibilidade do *Scrubbing* com intervalo de 1 dia: 2A 2M e 10A 10M (HUP).



Fonte: produção do autor.

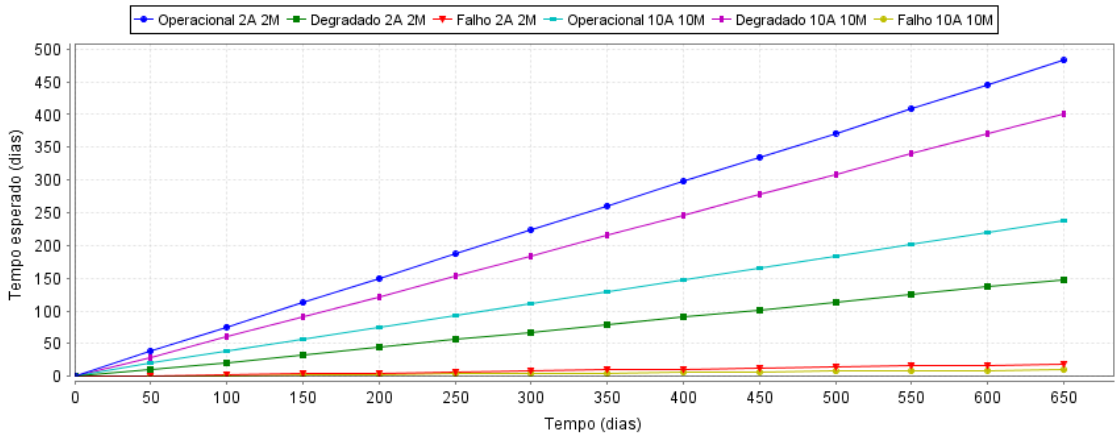
Figura 3.24 - Análise de disponibilidade do *Scrubbing* com intervalo de 1 dia: 2A 2M e 10A 10M (PUP).



Fonte: produção do autor.

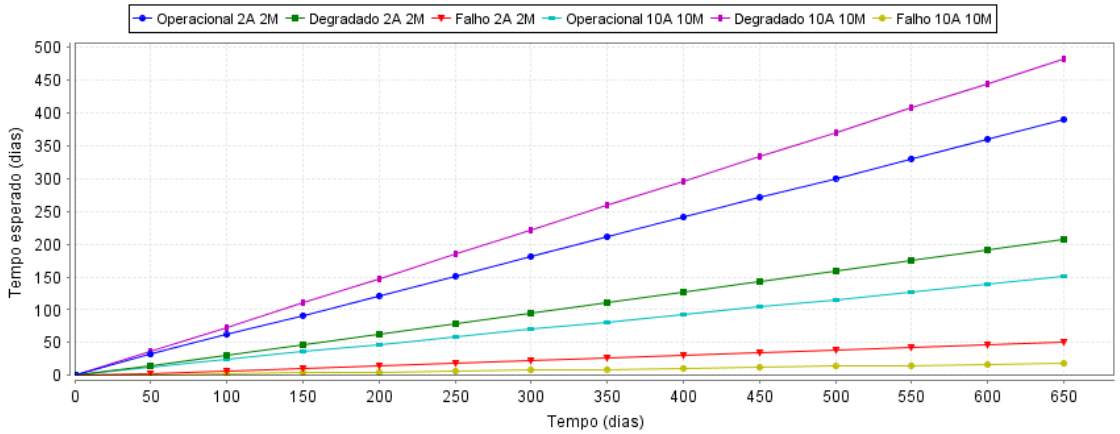
Quando o intervalo de Scrubbing subiu para 9 dias, C1 se manteve funcional por quase 100 dias a mais do que C2 no HUP, mas com a taxa de *upset* maior do PUP, a quantidade maior de componentes de C2 foi mais efetiva, como mostram as Figuras 3.25 e 3.26. Diferente do estudo de caso anterior, no qual a diferença entre os resultados apresentados por C1 e C2 era substancial, nesse caso, por conter um tempo de missão maior, o intervalo de Scrubbing alterou de forma significativa o resultado e deve ser fator decisivo em aplicações que possuam um tempo de missão longo mesmo com a variação da quantidade de componentes. O motivo principal dessa diferença foi o intervalo de Scrubbing, que mesmo considerando o de 9 dias, se tornou irrisório perto do tempo total de missão e da taxa de *upset*, impedindo um acúmulo demasiado de falhas.

Figura 3.25 - Análise de disponibilidade do *Scrubbing* com intervalo de 9 dias: 2A 2M e 10A 10M (HUP).



Fonte: produção do autor.

Figura 3.26 - Análise de disponibilidade do *Scrubbing* com intervalo de 9 dias: 2A 2M e 10A 10M (PUP).



Fonte: produção do autor.

Para os resultados obtidos na probabilidade limite de falha das técnicas TMR e código de Hamming, as conclusões da análise anterior se reforçaram, mostrando que mesmo com uma taxa de falhas bem diferente entre si (HUP tem o MTBF quase duas vezes maior que o PUP), com um tempo longo de missão os resultados se apro-

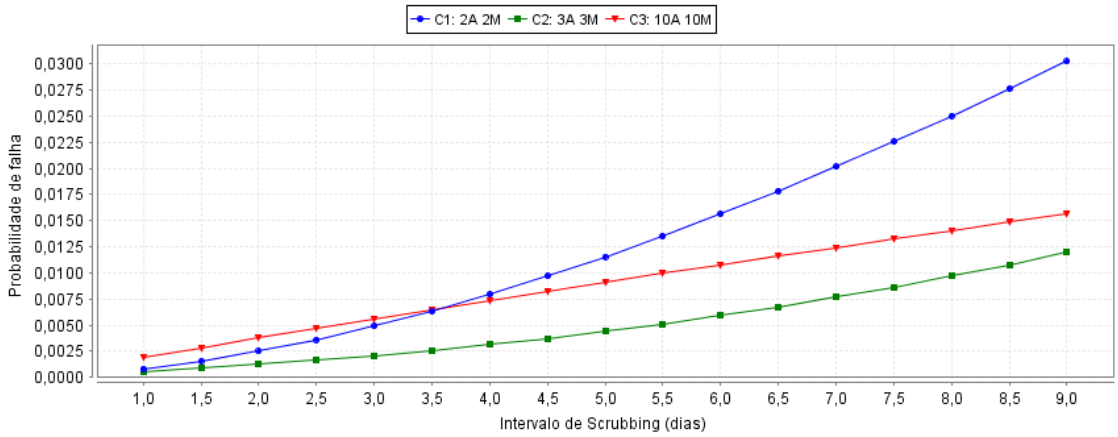
ximaram consideravelmente, como mostra a Tabela 3.6. Outro detalhe interessante para esse resultado foi que em ambas as técnicas, a probabilidade de falha diminuiu em relação ao estudo de caso anterior, mas a mudança mais drástica foi no código de Hamming, pois o tempo de correção de um *bit flip* se tornou desprezível perto do tempo de missão.

Tabela 3.6 - Probabilidade limite de falha do TMR e do código de Hamming para o segundo estudo de caso (CBERS-4A).

Técnica de mitigação de SEU	**Tipo de ionização**	**Probabilidade de falha**
TMR	HUP	$46,2\%$
TMR	PUP	$47,1\%$
Código de Hamming	HUP	$0,23\%$
Código de Hamming	PUP	$0,46\%$

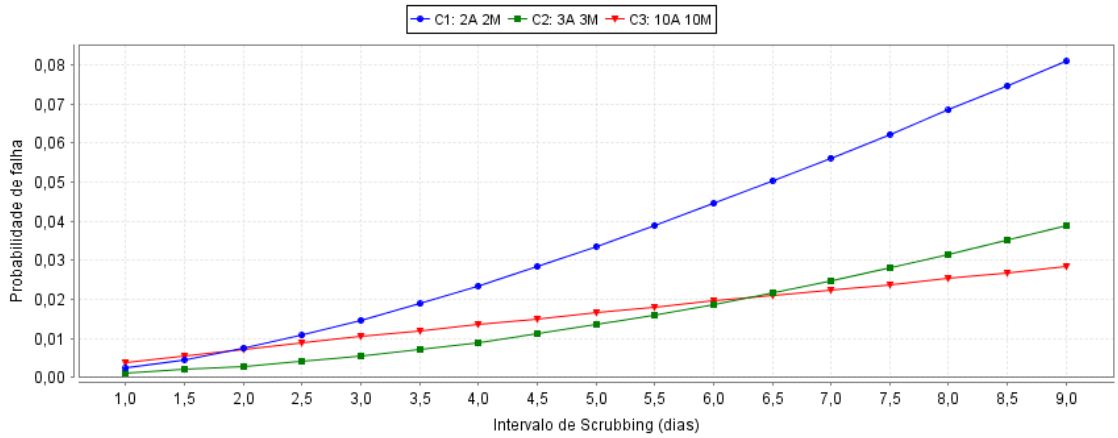
Na análise de probabilidade limite do Scrubbing com intervalo entre as correções variando de 1 até 9 dias, as Figuras 3.27 e 3.28 corroboraram o que foi apresentado nos resultados anteriores de disponibilidade para o Scrubbing, pois novamente a taxa de *upset* se tornou fator secundário quando o tempo de missão é maior. Quando os resultados abaixo são comparados com o estudo de caso anterior, é possível notar uma queda drástica na probabilidade limite de falha, já que o intervalo de Scrubbing foi mantido. O que permaneceu semelhante entre os estudos de caso foi a probabilidade de falha maior em C1 quando comparada à C2, mas ainda bem distantes dos valores apresentados na seção anterior.

Figura 3.27 - Probabilidade limite de falha de *Scrubbing*: 2A 2M, 3A 3M e 10A 10M (HUP)



Fonte: produção do autor.

Figura 3.28 - Probabilidade limite de falha de *Scrubbing*: 2A 2M, 3A 3M e 10A 10M (PUP).



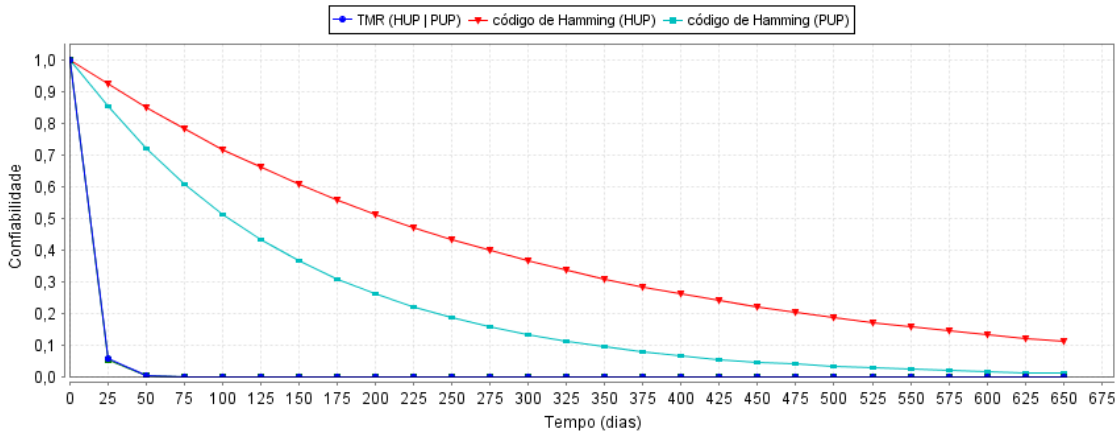
Fonte: produção do autor.

3.3.2.2 Análise de Confiabilidade e Segurança

Na análise de confiabilidade para o estudo de caso CBERS-4A, ao considerar apenas os estados operacional ou degradado nos 650 dias de missão, foram averiguados resultados distintos em relação ao estudo de caso anterior. A Figura 3.29 mostra que

os resultados de confiabilidade do código de Hamming e do TMR foram precários tanto em HUP quanto em PUP. Esses valores apresentados corroboram a afirmação de que uma técnica capaz de corrigir um *upset* deve acompanhar uma técnica de redundância para que a vulnerabilidade de uma técnica seja sobreposta pela competência da outra. Um detalhe interessante do TMR, é que a taxa de *upset* dos dois tipos de ionização não alteraram a confiabilidade, por isso foram consideradas de forma conjunta.

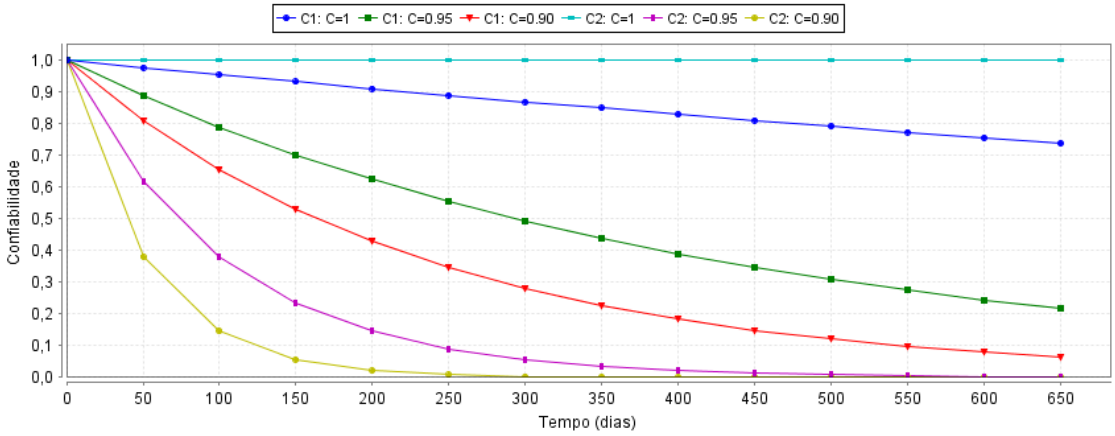
Figura 3.29 - Confiabilidade do TMR e do código de Hamming: missão de 650 dias (HUP e PUP).



Fonte: produção do autor.

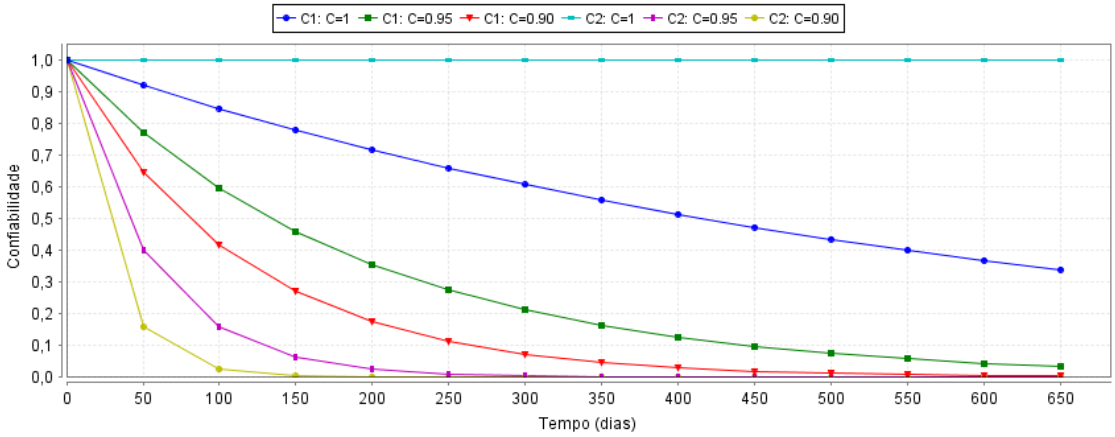
A confiabilidade do Scrubbing, assim como no estudo de caso anterior, foi dividida pelo intervalo de correção: 1 dia e 9 dias. As Figuras 3.30 e 3.31 apresentam os resultados de confiabilidade para o intervalo de 1 dia para o HUP e PUP, respectivamente. É possível notar que o tempo de missão maior não alterou tanto o resultado quando comparado ao estudo anterior, mas que a variação entre as duas taxas de *upset* foi fator dominante no resultado do Scrubbing para C1, que teve sua confiabilidade reduzida no PUP. Nessa análise, além quantidade de componentes, a taxa de cobertura também foi alterada para comparação, partindo de 100% até 90% com o mesmo tempo de missão de 650 dias.

Figura 3.30 - Confiabilidade do *Scrubbing* com 1 dia de intervalo: missão de 650 dias (HUP).



Fonte: produção do autor.

Figura 3.31 - Confiabilidade do *Scrubbing* com 1 dia de intervalo: missão de 650 dias (PUP).

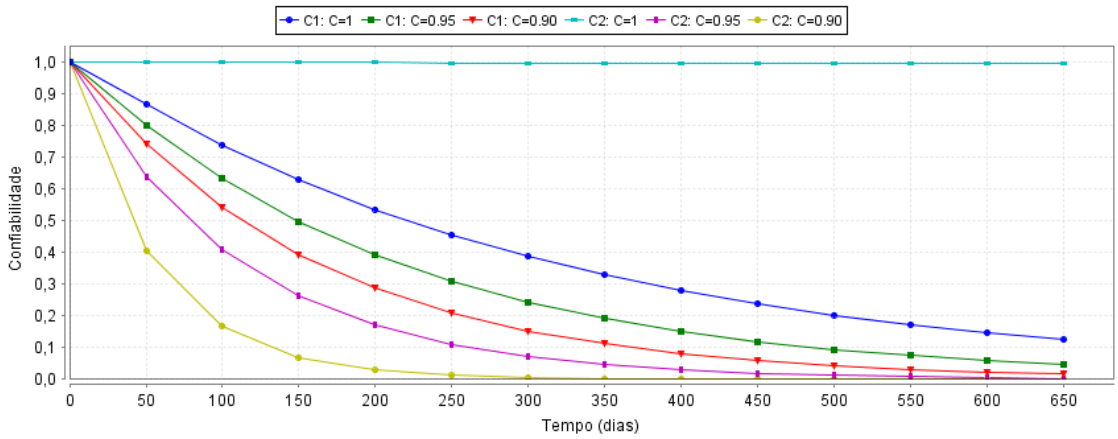


Fonte: produção do autor.

A segunda análise de confiabilidade do Scrubbing considerou o intervalo de 9 dias entre as correções e, assim como no estudo anterior, os resultados só foram adequados na configuração C2 com taxa de cobertura de 100%, como mostram as Figuras 3.32 e 3.33. Portanto, os resultados apresentados corroboram o estudo de caso anterior

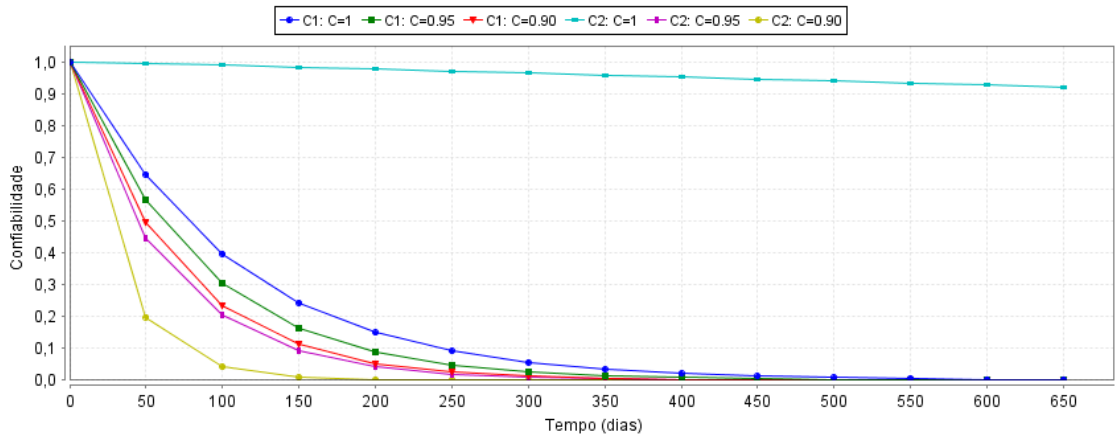
mostrando que o intervalo de Scrubbing, mesmo em uma missão de 650 dias afeta de forma crítica a confiabilidade do sistema, especialmente em sistemas com poucos componentes sobressalentes.

Figura 3.32 - Confiabilidade do *Scrubbing* com 9 dias de intervalo: missão de 650 dias (HUP).



Fonte: produção do autor.

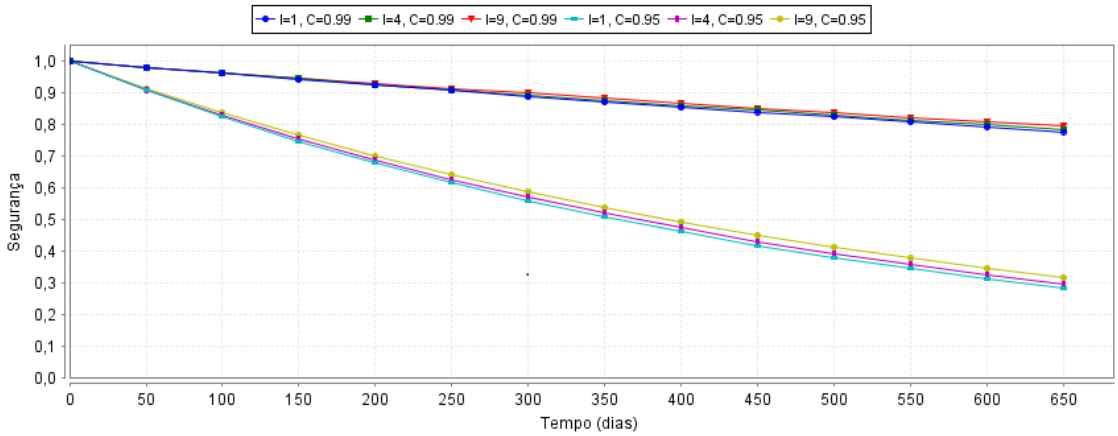
Figura 3.33 - Confiabilidade do *Scrubbing* com 9 dias de intervalo: missão de 650 dias (PUP).



Fonte: produção do autor.

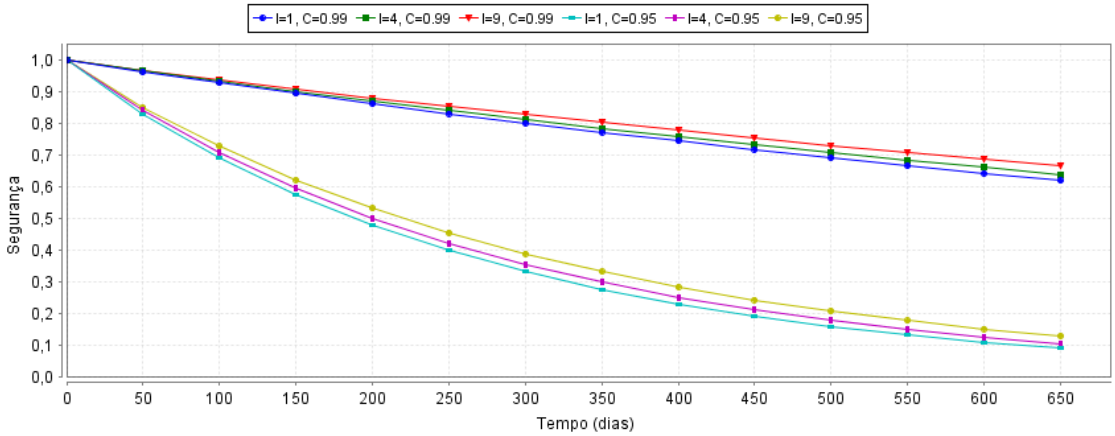
Para a análise da segurança, é importante ressaltar que apenas a técnica de Scrubbing foi testada por conter uma taxa de cobertura, ou seja, uma probabilidade variável de detectar um componente atingido por um *bit flip* dado que o mesmo ocorreu, conforme foi explicado anteriormente. As Figuras 3.34 e 3.35 ilustram a segurança de C1 com a taxas HUP e PUP com a taxa de cobertura de 99% e 95% para os três intervalos de correção e, assim como no estudo de caso anterior, a segurança está diretamente relacionada à taxa de cobertura, pois a variação no intervalo de Scrubbing não foi capaz de alterar de forma significativa os resultados em ambas as taxas HUP e PUP para C1.

Figura 3.34 - Segurança do *Scrubbing* com 2 somadores e 2 multiplicadores (C1): missão de 650 dias (HUP).



Fonte: produção do autor.

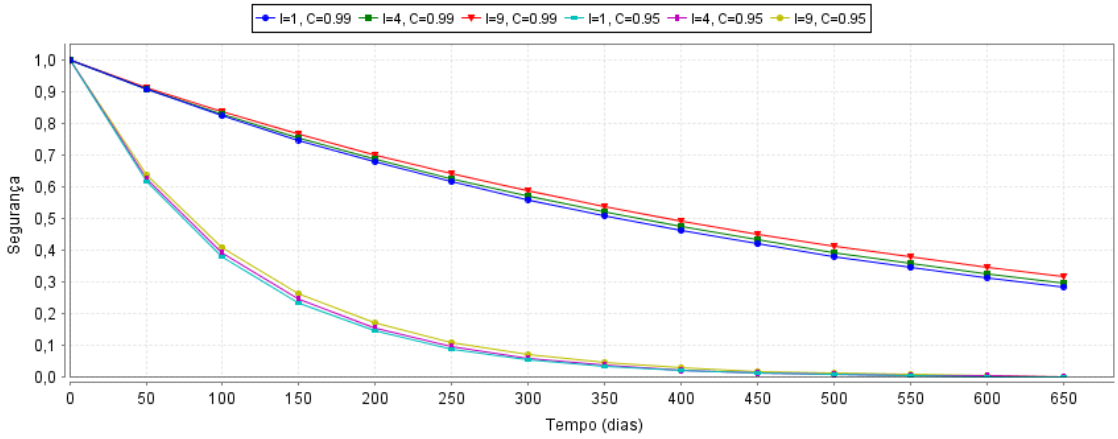
Figura 3.35 - Segurança do *Scrubbing* com 2 somadores e 2 multiplicadores (C1): missão de 650 dias (PUP).



Fonte: produção do autor.

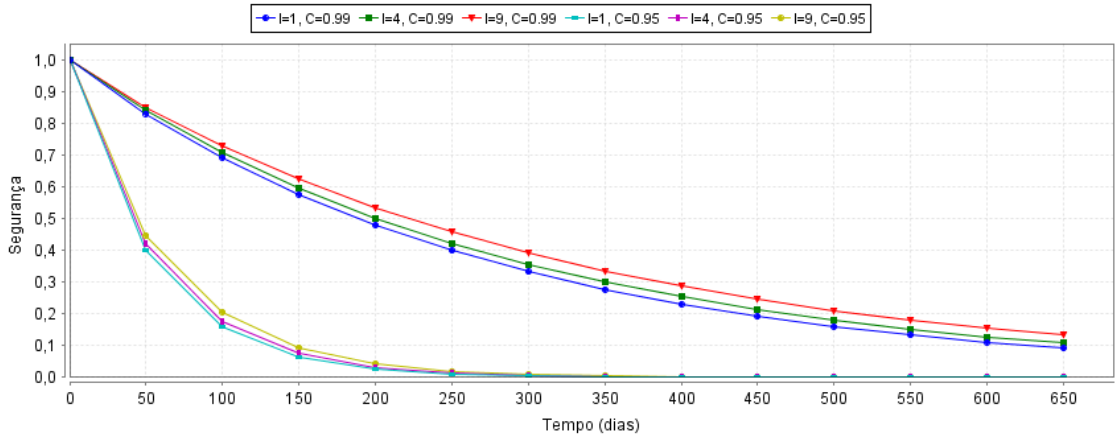
Quando a mesma análise é submetida à C2, as Figuras 3.36 e 3.37 mostram que a segurança apresentou resultado insatisfatório em ambas as taxas de cobertura, mas a superioridade da taxa maior (99%) valida a conclusão anterior. Por fim, é possível concluir que grande parte dos resultados corroboraram os obtidos no estudo de caso anterior e mostraram a consistência do *Model Checking* Probabilístico na análise das técnicas de mitigação de SEU em FPGAs.

Figura 3.36 - Segurança do *Scrubbing* com 10 somadores e 10 multiplicadores (C1): missão de 650 dias (HUP).



Fonte: produção do autor.

Figura 3.37 - Segurança do *Scrubbing* com 10 somadores e 10 multiplicadores (C1): missão de 650 dias (PUP).



Fonte: produção do autor.

3.4 Considerações finais sobre esse Capítulo

Esse capítulo apresentou os dois estudos de caso analisados nesta dissertação de mestrado e todo o processo de *Model Checking* Probabilístico desenvolvido para simular e analisar os critérios de dependabilidade na qual as técnicas de mitigação

de SEU foram submetidas. As fases de projeto foram elicitadas e detalhadas desde a elaboração, determinação das taxas, simulação e geração dos resultados. Por fim, os resultados apresentados em ambos os estudos de caso se alinharam nas conclusões e mostraram a consistência do *Model Checking* Probabilístico na análise das técnicas de mitigação de SEU em FPGAs SRAM mesmo em condições de órbita, taxas de falha e tempos de missão distintos.

Para a disponibilidade, o código de Hamming apresentou o melhor resultado e foi a técnica que conseguiu se manter mais tempo operacional seguida de perto pelo Scrubbing com intervalo de correção de 1 dia. Como dito anteriormente, o TMR apresentou o pior resultado pois acumulava as falhas por não haver técnica de correção, portanto técnicas de redundância precisam ser acompanhadas de algum mecanismo capaz de corrigir uma falha antes que ela interfira na votação por maioria. Para a confiabilidade, foi possível observar que o Scrubbing apresenta uma sensibilidade maior a variações no intervalo entre as correções do que quando a alteração ocorre na taxa de cobertura, e que a quantidade maior de componentes diminui as chances do sistema entrar em estado de falha. No entanto, para a segurança os resultados foram opostos, pois a taxa de cobertura foi justamente a característica que mais afetou o sistema enquanto as mudanças no intervalo entre as correções não alterou significativamente a segurança.

O capítulo seguinte detalha o desenvolvimento em VHDL e simulação funcional das mesmas técnicas analisadas via *Model Checking* Probabilístico para permitir uma comparação direta entre os resultados obtidos entre os modelos e as técnicas. Ambos foram comparados considerandos os critérios de disponibilidade e confiabilidade.

4 COMPARAÇÃO DE MODEL CHECKING PROBABILÍSTICO COM SIMULAÇÃO FUNCIONAL

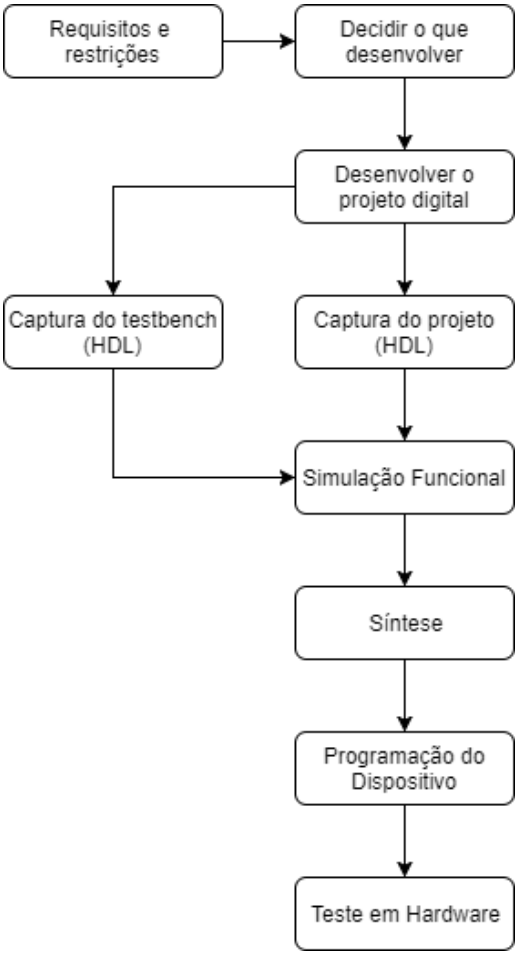
Esse capítulo trata da comparação entre os resultados obtidos via *Model Checking* Probabilístico, conforme metodologia apresentada no capítulo anterior, com Simulação Funcional. As etapas de desenvolvimento em VHDL no ambiente Quartus II e Simulação Funcional via ModelSim, considerando as técnicas de mitigação de SEU definidas para esse trabalho, são detalhadas para ser possível comparar os resultados obtidos via *Model Checking* Probabilístico. Ambas as abordagens foram comparadas considerandos os atributos de disponibilidade e confiabilidade.

4.1 Desenvolvimento e Simulação Funcional das Técnicas de Mitigação

Antes de apresentar os detalhes dos procedimentos de implementação e simulação das técnicas de mitigação, é importante que o processo de desenvolvimento em FPGA seja entendido de forma geral. As etapas de um projeto em hardware costumam seguir uma sequência padrão de passos, onde o objetivo é modularizar cada processo e tornar o desenvolvimento mais preciso. A Figura 4.1 foi adaptada de (FULKS; COFER, 2012) e apresenta etapas comuns para o desenvolvimento de projetos em hardware.

Como a etapa de requisitos e de decisão sobre qual deveria ser o desenvolvimento foi parte do processo de modelagem probabilística, a primeira etapa considerada nesse capítulo é a do desenvolvimento do projeto digital das técnicas de mitigação: Scrubbing, TMR e código de Hamming. Essa etapa envolve projetar o funcionamento de cada técnica antes de iniciar de fato a implementação, e pode ser feita por meio de rascunhos com diagramas ou com ferramentas mais sofisticadas que auxiliam a idealizar o sistema e como ele deveria funcionar. Como esta etapa foi inteira baseada em esboços pré-implementação, para ilustrar o processo, o Quartus II possui uma ferramenta de visualização que gera o RTL (*Register Transfer Level*) baseado no que foi implementado.

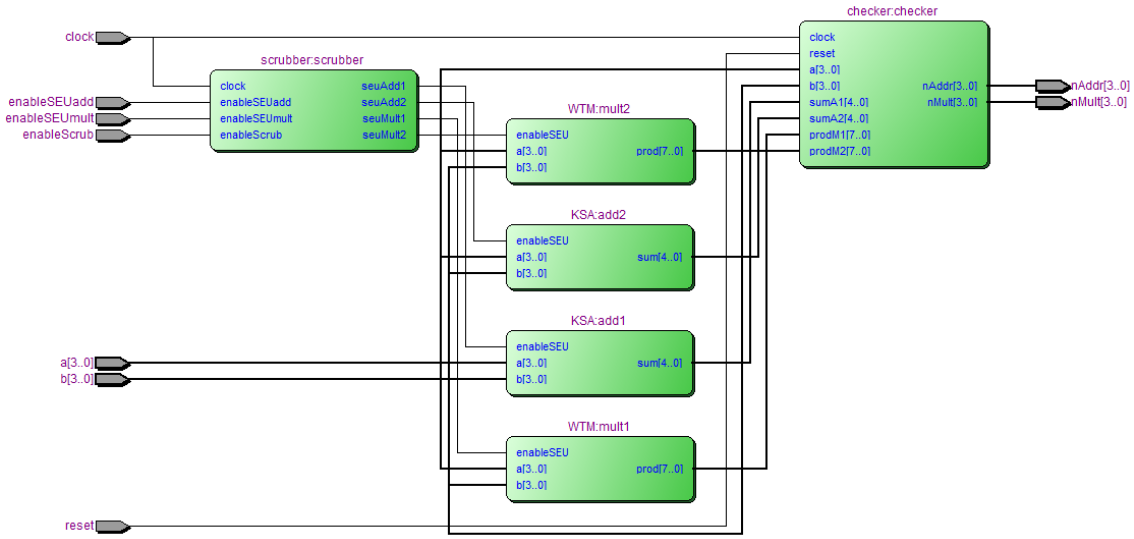
Figura 4.1 - Processo de desenvolvimento aplicado à FPGAs.



Fonte: adaptado de Fulks e Cofer (2012).

A Figura 4.2 ilustra como ficou o projeto digital desenvolvido para a técnica de Scrubbing. Inicialmente, a ideia foi que os componentes (somadores e multiplicadores) recebessem duas entradas de 4 bits e fizessem suas operações em paralelo, enquanto um módulo seria responsável por efetuar as correções (`scrubber`) dentro dos intervalos de tempo determinados e o último módulo (`checker`) verificaria as saídas de cada componente e retornaria quantos deles efetuou a operação de forma correta. Existem duas entradas que recebem o sinal para simular um *upset*, uma para somadores (`enableSEUadd`), outra para os multiplicadores (`enableSEUmult`) e também uma entrada que sinaliza a ocorrência de uma correção (`enableScrub`), e todos os componentes que estão gerando saídas incorretas passam a operar de forma funcional.

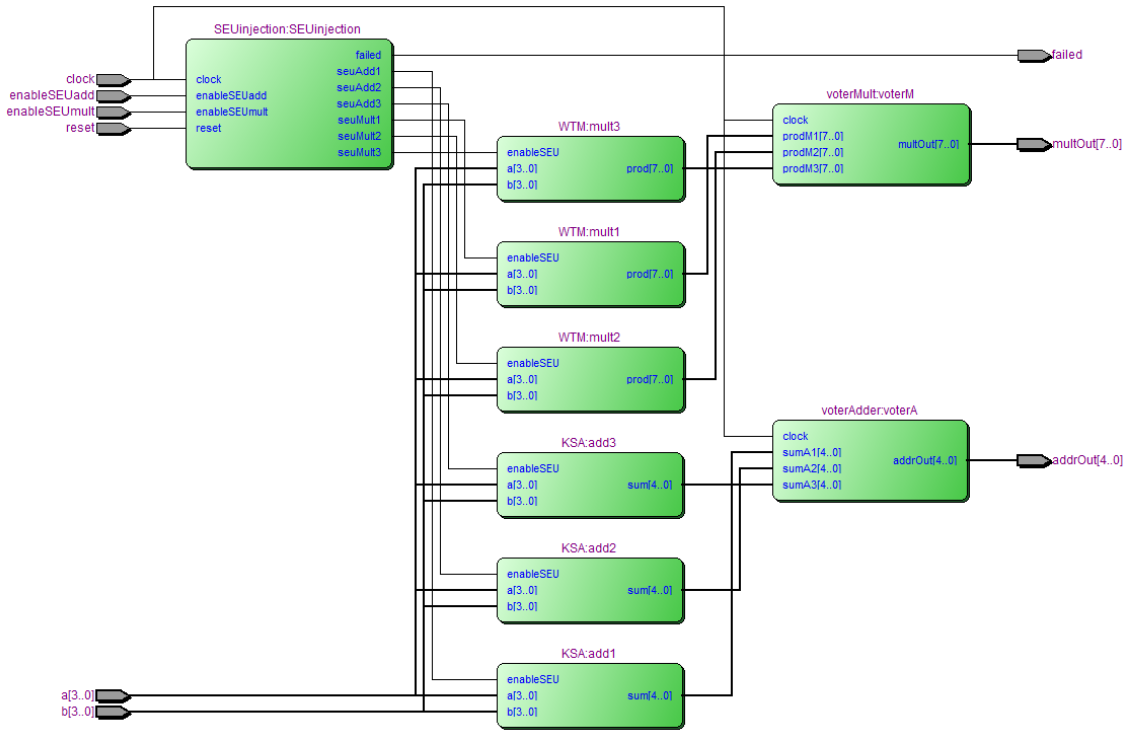
Figura 4.2 - Visualização do projeto digital (RTL) para o Scrubbing com 2 somadores e 2 multiplicadores.



Fonte: produção do autor.

Para o projeto do TMR, seu diagrama é bem conhecido e já foi apresentado no Capítulo 2, mas a Figura 4.3 ilustra o RTL gerado pelo Quartus II com mais detalhes. Nesse projeto, é possível identificar três módulos para os somadores (`KSA`) e três módulos para os multiplicadores (`WTM`), representando a redundância do TMR, além de um módulo responsável por simular a inserção de uma falha (`SEUinjection`) quando o sinal de entrada é acionado (`enableSEUadd` e `enableSEUmult`) e também os dois módulos votadores (`voterA` e `voterM`) que, baseado em uma votação por maioria, decidem qual será a saída final dos somadores e multiplicadores. Além das duas entradas de 4 bits (`a` e `b`) e das saídas geradas pelos votadores (`addrOut` e `multOut`), existe uma saída (`failed`) que indica quando mais de um módulo foi danificado e a saída gerada pelo votador está incorreta, mas essa saída foi utilizada somente para facilitar a verificação do funcionamento do TMR.

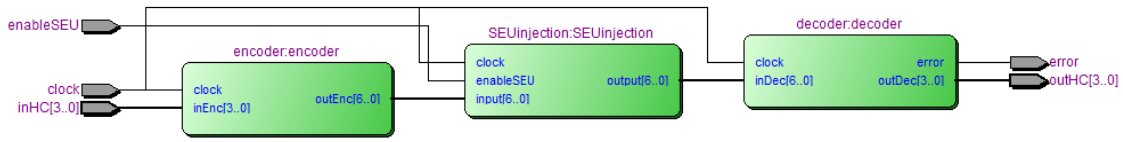
Figura 4.3 - Visualização do projeto digital (RTL) para o TMR.



Fonte: produção do autor.

No caso do projeto digital do código de Hamming, seu diagrama foi o mais compacto dentre os desenvolvidos, como mostra a Figura 4.4. Diferentemente dos outros dois modelos que consideravam duas entradas de 4 bits para que os componentes efetuassem suas operações, o código de Hamming, assim como nos modelos probabilísticos, não está relacionado aos somadores e multiplicadores, por isso pode ser implementado para qualquer componente que esteja na memória da FPGA. No contexto desse trabalho, para se aproximar ao modelo CTMC, o código de Hamming recebe uma entrada de 4 bits (`inHC`) e, no módulo codificador (`encoder`), adiciona 3 bits de paridade que são codificados junto aos 4 bits de entrada.

Figura 4.4 - Visualização do projeto digital (RTL) para o código de Hamming.



Fonte: produção do autor.

O segundo módulo (`SEUinjection`) é responsável por gerar um *bit flip* em qualquer um dos bits de entrada baseado no sinal de simulação de *upset*. Por fim, o módulo decodificador (`decoder`) recebe esse sinal de 7 bits, verifica sua integridade e em caso de *bit flip*, é capaz de detectar em qual posição ele ocorreu e efetuar a correção. Na saída do decodificador é passado o sinal de 4 bits inicial e uma *flag* chamada `error`, que indica se o decodificador precisou corrigir algum *bit flip* ou se a entrada codificada estava íntegra e não precisou de correção.

As etapas seguintes, categorizadas como "Captura do projeto" e "Captura do *testbench*", envolvem a implementação propriamente dita das técnicas de mitigação assim como as rotinas de *testbench*, que de forma simplificada são algoritmos que atribuem valores às entradas para que as ferramentas de simulação possam emular o funcionamento do sistema e gerar as saídas. Essas etapas foram desenvolvidas no ambiente Quartus II utilizando a linguagem de descrição de hardware VHDL, onde o foco foi manter o *modus operandi* mais próximo possível do que foi modelado no PRISM, para que a comparação dos resultados fosse possível.

Os componentes implementados para avaliar o funcionamento das técnicas, somador Kogge Stone (KSA) e multiplicador Wallace Tree (WTM) são utilizados tanto no Scrubbing quanto no TMR, portanto é importante explicar de forma geral o funcionamento de cada um. A Figura 4.5 apresenta trechos da implementação do KSA, que faz a soma de duas entradas de 4 bits, `a` e `b`. Um detalhe importante é que a ocorrência de um *bit flip* pode gerar uma alteração nos operadores utilizados para o cálculo da soma. Portanto, o processo implementado no fim da arquitetura considera o valor do sinal `enableSEU` para simular a ocorrência do *upset* por meio de uma troca da porta *xor* por uma porta *xnor*.

Figura 4.5 - Trechos da implementação do somador Kogge Stone (KSA) em VHDL.

```
architecture Behavioral of KSA is
begin
        Gin <= a and b;
        Pin <= a xor b;

        stg01:
                for i in 0 to 2 generate
                        G1(i+1) <= Gin(i+1) or (Gin(i) and Pin(i+1));
                        P1(i+1) <= Pin(i+1) and Pin(i);
                end generate;

        stg02:
                for i in 0 to 1 generate
                        G2(i+2) <= G1(i+2) or (G1(i) and P1(i+2));
                        P2(i+2) <= P1(i+2) and P1(i);
                end generate;

        process(sumInt,Pin,enableSEU)
        begin
                if(enableSEU='1') then
                        sumInt(0) <= '0' xnor Pin(0);
                else
                        sumInt(0) <= '0' xor Pin(0);
                end if;
        end process;
end Behavioral;
```

Fonte: adaptado de Mauri (2008).

Diferentemente do KSA, o WTM possui mais de uma entidade, pois utiliza internamente meio somador e somador completo para efetuar o produto entre duas entradas, `a` e `b` de 4 bits. A Figura 4.6 ilustra trechos da entidade principal do WTM onde, assim como no KSA, é possível observar a simulação de um *upset* por meio da troca de uma porta *and* por uma porta *or*, causando um *bit flip* na saída do módulo.

Figura 4.6 - Trechos da implementação do multiplicador Wallace Tree (WTM) em VHDL.

```
architecture Behavioral of WTM is
begin

process(a,b,enableSEU)
begin
    for i in 0 to 3 loop
        p0(i) <= a(i) and b(0);
        p1(i) <= a(i) and b(1);
        p2(i) <= a(i) and b(2);
                  if(enableSEU='1') then
                               p3(i) <= a(i) or b(3);
                  else
                               p3(i) <= a(i) and b(3);
                  end if;
    end loop;
end process;

ha11 : half_adder port map(p0(1),p1(0),s11,c11);
fa12 : full_adder port map(p0(2),p1(1),p2(0),s12,c12);
fa13 : full_adder port map(p0(3),p1(2),p2(1),s13,c13);
fa14 : full_adder port map(p1(3),p2(2),p3(1),s14,c14);
ha15 : half_adder port map(p2(3),p3(2),s15,c15);

end Behavioral;
```

Fonte: adaptado de Lal (2013).

Começando pelo Scrubbing, apesar do sistema com 10 somadores e 10 multiplicadores ter sido implementado em um projeto separado, os trechos de código apresentados valem para ambos, já que apenas a quantidade de componentes foi alterada. O trecho mais importante para ser apresentado se refere ao módulo de correção, que utiliza um somador e um multiplicador adicional para simular uma cópia imune à radiação, e a cada intervalo de tempo determinado, utiliza essa cópia adicional para reescrever os componentes e restaurar aqueles que estavam gerando a saída incorreta, como mostra a Figura 4.7. Assim como no modelo probabilístico, a técnica não depende da detecção da ocorrência de um *upset* para efetuar a correção, ou seja, ela age mesmo que nenhum componente tenha sido afetado (*Blind Scrubbing*).

Figura 4.7 - Trechos da implementação do módulo que efetua a correção no Scrubbing.

```
architecture Behavioral of scrubber is

component KSA is
    port (
        enableSEU : in std_logic;
        a : in std_logic_vector (3 downto 0);
        b : in std_logic_vector (3 downto 0);
        sum : out std_logic_vector (4 downto 0));
end component;

component WTM is
    port (
        enableSEU : in std_logic;
        a : in  std_logic_vector (3 downto 0);
        b : in  std_logic_vector (3 downto 0);
        prod : out  std_logic_vector (7 downto 0));
end component;

begin

KSAinterno : KSA port map(enableSEU,a,b,sumInt);

WTMinterno : WTM port map(enableSEU,a,b,prodInt);

end Behavioral;
```

Fonte: produção do autor.

Na implementação do TMR, além dos somadores e multiplicadores já exibidos acima, é importante apresentar o funcionamento dos votadores. Apesar de elementar, o modo com que o votador recebe as três saídas e é capaz de decidir baseado na maioria utilizando apenas portas *and* e *not* é exibido na Figura 4.8. O código exibido é utilizado no votador dos somadores, mas seu funcionamento é semelhante ao utilizado no votador dos multiplicadores.

Para controlar a ocorrência de *upsets*, o módulo injetor de falhas controla quais componentes estão operacionais e sinaliza com uma *flag* o próximo componente que será atingido. Assim como no modelo probabilístico, os *upsets* ocorrem um por vez, para que os resultados possam ser comparáveis.

Figura 4.8 - Trechos da implementação do módulo que efetua a votação por maioria para os somadores no TMR.

```
entity voterAdder is
    Port(clock : in std_logic;
        sumA1 : in std_logic_vector (4 downto 0);
        sumA2 : in std_logic_vector (4 downto 0);
        sumA3 : in std_logic_vector (4 downto 0);
        addrOut : out std_logic_vector (4 downto 0)
        );
end voterAdder;

architecture Behavioral of voterAdder is

begin

process (sumA1, sumA2, sumA3, sumInt1, sumInt2, sumInt3, clock)

begin
    if clock'event and clock='1' then
        sumInt1 <= sumA1 nand sumA2;
        sumInt2 <= sumA1 nand sumA3;
        sumInt3 <= sumA2 nand sumA3;
    end if;
    addrOut <= not (sumInt1 and sumInt2 and sumInt3);
end process;
end Behavioral;
```

Fonte: produção do autor.

Sobre a implementação do código de Hamming em VHDL, a Figura 4.9 ilustra trechos da codificação dos bits de paridade junto aos bits de entrada. Para isso, são utilizadas portas xor entre os bits para que a detecção da posição de possível *upset* ocorra no módulo decodificador. Para simular um *upset*, o módulo de injeção de falhas verifica o sinal `enableSEU` e, caso esteja ativo, um *bit flip* é inserido em qualquer um dos 7 bits codificados.

Figura 4.9 - Trechos da implementação do módulo que efetua a codificação dos bits no código de Hamming.

```
ENTITY encoder IS
    PORT(inEnc : in std_logic_vector (3 downto 0);
        clock : in std_logic;
        outEnc : out std_logic_vector (6 downto 0));
END encoder;

ARCHITECTURE Behavioral OF encoder IS
BEGIN

PROCESS(clock)
BEGIN
IF(clock'event and clock='1') THEN
    outEnc(6) <= inEnc(3);
    outEnc(5) <= inEnc(2);
    outEnc(4) <= inEnc(1);
    outEnc(3) <= inEnc(0);
    outEnc(2) <= inEnc(1) XOR inEnc(2) XOR inEnc(3);
    outEnc(1) <= inEnc(0) XOR inEnc(2) XOR inEnc(3);
    outEnc(0) <= inEnc(0) XOR inEnc(1) XOR inEnc(3);
END IF;
END PROCESS;

END Behavioral;
```

Fonte: adaptado de Choudhury e Podder (2015).

Com as técnicas implementadas juntamente aos seus módulos de injeção de falhas, foi preciso desenvolver *testbenchs* capazes de inserir uma falha a cada período de tempo estabelecido pelas taxas calculadas via CREME96. O procedimento implementado nos *testbenchs* são semelhantes em todas as técnicas com exceção do Scrubbing que, além dos ciclos de *upset*, também possui um ciclo de correção definido. A Figura 4.10 descreve quatro ciclos: o de *clock*, o de Scrubbing e os dois ciclos de injeção de falhas, para os somadores e para os multiplicadores. Para cada simulação, é necessário configurar os ciclos para que os intervalos atendam ao teste solicitado.

Figura 4.10 - Trechos da implementação do *testbench* que insere uma falha a cada período estabelecido.

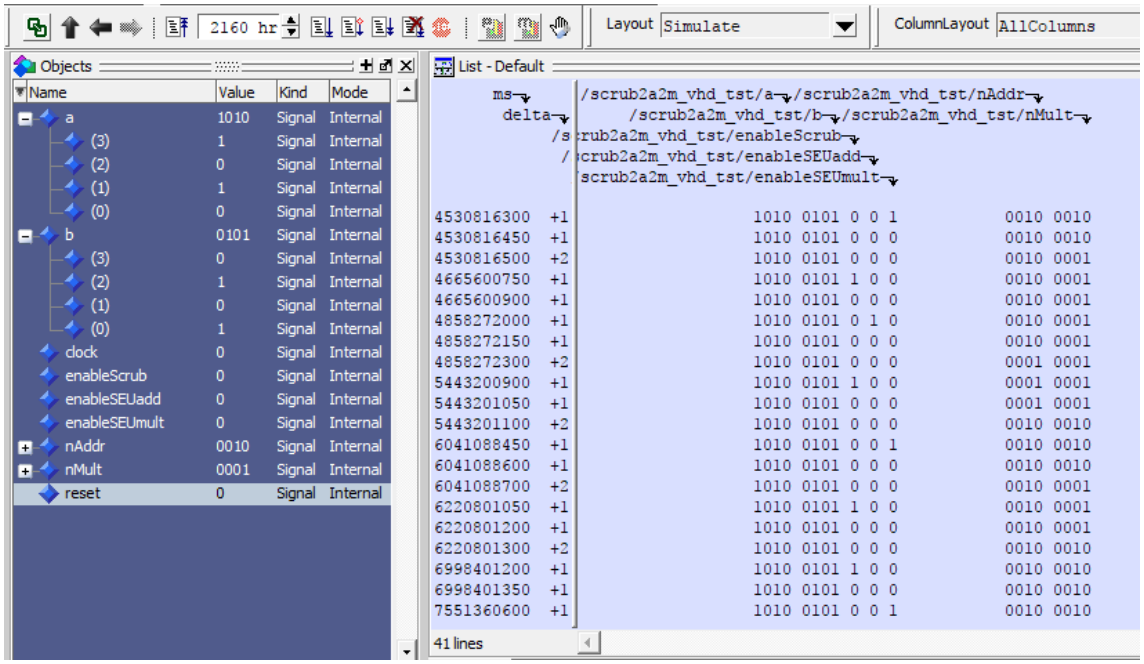
```
clk_proc : PROCESS
BEGIN
    clock <= '0';
    WAIT FOR 100 ms;
    clock <= '1';
    WAIT FOR 100 ms;
END PROCESS clk_proc;

scrub_proc : PROCESS
BEGIN
    enableScrub <= '0';
--  WAIT FOR 24 hr; -- 1 dia
--  WAIT FOR 96 hr; -- 4 dias
--  WAIT FOR 216 hr; -- 9 dias
    enableScrub <= '1';
    WAIT FOR 150 ms;
END PROCESS scrub_proc;

SEUm_proc : PROCESS
BEGIN
    enableSEUmult <= '0';
--  WAIT FOR 284.4 hr; -- 11,85 dias em horas
--  WAIT FOR 1612.8 hr; -- 67,2 dias em horas (HUP)
--  WAIT FOR 839.04 hr; -- 34,96 dias em horas (PUP)
    enableSEUmult <= '1';
    WAIT FOR 150 ms;
END PROCESS SEUm_proc;

SEUa_proc : PROCESS
BEGIN
    enableSEUadd <= '0';
--  WAIT FOR 915.6 hr; -- 38,15 dias em horas
--  WAIT FOR 5188.8 hr; -- 216,2 dias em horas (HUP)
--  WAIT FOR 2699.04 hr; -- 112,46 dias em horas (PUP)
    enableSEUadd <= '1';
    WAIT FOR 150 ms;
END PROCESS SEUa_proc;
```

Fonte: produção do autor.

O software selecionado para interpretar o *testbench* e executar as simulações foi o ModelSim, que é instalado como parte do Quartus II. De forma geral, a entrada do ModelSim exige que o usuário descreva o tempo total em que o projeto será simulado, que nesse caso foi o tempo de missão de 90 dias para o primeiro estudo de caso e 650 dias para o segundo. Normalmente, a saída mais comum utilizada pelo ModelSim é conhecida como *waveform*, onde é possível visualizar as saídas e entradas em uma linha do tempo, mas isso se tornou inviável devido ao longo período de execução das simulações.

Portanto, foi necessário utilizar uma outra forma de avaliar as entradas e saídas do ModelSim, e a solução encontrada foi a de organizar os dados em formato de lista, já que a ferramenta permite exportar essas saídas em um arquivo "*.list*". A Figura 4.11 ilustra como é visualmente gerada a saída e o que cada coluna representa no projeto. O passo seguinte envolve exportar os arquivos de saída, inserí-los em planilhas e analisar, baseado no tempo em que cada mudança de sinal ocorreu, quais foram os resultados obtidos pelas técnicas dentro das configurações de quantidade de componentes, taxa de falha, e no caso do Scrubbing, intervalo de correção.

Figura 4.11 - Saída gerada pelo ModelSim para a simulação do Scrubbing com tempo de missão de 90 dias (2160 horas).



Fonte: produção do autor.

No total, foram executadas 24 simulações, sendo 18 simulações para as duas configurações de Scrubbing (C1 e C2) variando a taxa de correção (1, 4 e 9 dias) em ambos os estudos de caso. É importante relembrar que o primeiro estudo de caso (OEA) possui uma taxa de *upset* para os somadores e multiplicadores, e o segundo estudo (CBERS-4A) possui duas taxas (HUP e PUP) para os mesmos componentes. Por fim, foram realizadas três simulações para o TMR, com as três taxas de falha e mais três para o código de Hamming, respeitando sempre o tempo de missão estabelecido para cada estudo de caso.

Além da simulação funcional, cada uma das técnicas também foi implementada e executada em uma Xilinx Virtex-5 presente no Laboratório do Projeto CITAR (INPE, 2016), mas o equipamento não foi submetido à radiação portanto não foram obtidos resultados práticos desse desenvolvimento.

4.2 Resultados e Discussão

Essa seção apresenta os resultados obtidos via simulação funcional no *ModelSim* para os dois estudos de caso exibidos nesse trabalho. Os resultados foram diretamente comparados aos apresentados no capítulo anterior para discutir a viabilidade do uso do *Model Checking* Probabilístico nas etapas iniciais de desenvolvimento de um projeto espacial. Os atributos de dependabilidade analisados foram: disponibilidade e confiabilidade.

4.2.1 Órbita Elíptica Alta

Esse estudo de caso apresenta as taxas de ocorrência de *upset* definidas em (HOQUE, 2016) para uma FPGA Virtex-5 em Órbita Elíptica Alta (OEA) com um tempo de missão de 90 dias. As seções seguintes descrevem os resultados obtidos e uma análise dos mesmos considerando três categorias de funcionamento dos sistemas: operacional, degradado e falho, além da comparação com resultados obtidos com os modelos CTMC.

4.2.1.1 Análise de Disponibilidade

Para essa análise, a simulação considerou 2160 horas, ou seja, os 90 dias de tempo de missão. Os dados de saída foram reunidos e organizados por técnica e por categoria de funcionamento, da mesma forma como foi feito na análise dos modelos no PRISM. A Tabela 4.1 apresenta os resultados para a configuração C1 (2 somadores e 2 multiplicadores) do Scrubbing e intervalo de correção estabelecido em 1 dia, seguido da Tabela 4.2 contendo a mesma configuração C1 mas com intervalo de 4 dias e por

fim a Tabela 4.3 com intervalo de correção de 9 dias. Em todos os casos, o valor que aparece nas tabelas é relativo ao tempo final de 90 dias. O valor percentual expressa a diferença entre o valor de referência (ModelSim) e aquele obtido com o PRISM nos modelos. Quando o valor for negativo, significa que o PRISM estimou menos dias do que o esperado, e positivo, mais dias do que o esperado.

Na análise via modelos probabilísticos, existe uma segunda forma de analisar a disponibilidade utilizando a probabilidade limite, mas por ser um critério que não considera o tempo de missão, não foi possível reproduzi-lo na simulação funcional das técnicas implementadas no Quartus II. Portanto, nessa seção só serão discutidos e comparados os resultados obtidos com o primeiro critério de disponibilidade utilizado no PRISM.

Tabela 4.1 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 1 dia.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	73,81	73,99	-0,24
Degradado	15,10	16,00	-5,62
Falho	1,09	0,00	——

Tabela 4.2 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 4 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	48,60	52,67	-8,29
Degradado	31,90	37,32	-14,52
Falho	9,50	0,00	——

Tabela 4.3 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 9 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	32,10	39,55	-18,83
Degradado	34,90	36,57	-4,56
Falho	23,00	13,87	65,82

Para essa análise, é possível observar que os resultados estiveram mais próximos

quando a correção ocorreu de forma mais frequente (1 dia), principalmente quando o estado de falha é analisado. O PRISM apresentou resultados mais pessimistas em relação à simulação do ModelSim, mas que nas situações analisadas nesse trabalho, se mantiveram próximos mesmo no intervalo de correção de 9 dias. Mas quando a mesma situação é analisada para o Scrubbing com 10 somadores e 10 multiplicadores, nota-se uma aproximação ainda maior entre os resultados do PRISM e do ModelSim, principalmente no intervalo de 1 dia, como mostram as Tabelas 4.4, 4.5 e 4.6. Portanto, uma conclusão plausível permite dizer que, para sistemas mais complexos, a utilização da modelagem probabilística pode trazer resultados muito próximos aos de uma abordagem de simulação funcional, antes mesmo de iniciar o desenvolvimento em hardware da aplicação.

Tabela 4.4 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 1 dia.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	42,99	43,43	-1,01
Degradado	47,00	46,56	0,94
Falho	0,00	0,00	——

Tabela 4.5 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 4 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	17,22	13,49	27,65
Degradado	72,50	76,50	-5,22
Falho	0,27	0,00	——

Tabela 4.6 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 9 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	8,94	6,34	41,01
Degradado	77,70	77,79	-0,11
Falho	3,36	5,86	-42,66

Para o TMR, o resultado considerando a disponibilidade está exibido na Tabela 4.7. Assim como nos resultados apresentados pelo PRISM (*Model Checking* Probabilís-

tico), o TMR desenvolvido em VHDL e simulado no ModelSim também apresentou resultados insatisfatórios. Mas, em uma análise comparativa, é possível observar que os resultados não foram tão próximos nos estados operacional e degradado, e isso ocorre pois no TMR o que determina a distinção entre esses dois estados é a ocorrência do primeiro *upset*, já que se trata de uma redundância tripla, portanto essa diferença elevada pode ser aceitável.

Porém, considerando o estado funcional como um todo (degradado e operacional são estados funcionais), os resultados se aproximaram de forma interessante mostrando que, para o TMR, a análise probabilística também pode auxiliar no desenvolvimento de técnicas de mitigação de SEU.

Tabela 4.7 - Comparação de disponibilidade do TMR com tempo de missão de 90 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	5,60	11,85	-52,74
Degradado	33,00	23,70	39,24
Falho	51,40	54,44	-5,58

Por fim, a Tabela 4.8 apresenta os resultados de disponibilidade do código de Hamming retornados pelo PRISM e pela simulação do ModelSim lado a lado. Diferente das técnicas anteriores, o conceito de degradado e falho no código de Hamming é diferente, pois essa técnica não foi implementada baseando-se em somadores e multiplicadores, e sim na correção direta de *bit flips* em qualquer componente na memória. Portanto, os estados falho e degradado apresentam o tempo que o código de Hamming demorou para detectar a ocorrência de um *upset* e o tempo a partir da detecção até a correção do *upset*, respectivamente.

Tabela 4.8 - Comparação de disponibilidade do código de Hamming com tempo de missão de 90 dias.

Estado	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	77,28	89,99	-14,12
Degradado	1,42	$1,38 \times 10^{-5}$	$1,02 \times 10^{7}$
Falho	11,30	$1,73 \times 10^{-5}$	$6,53 \times 10^{7}$

A inferência direta desse resultado é que, para análises que prezem por uma avaliação precisa da velocidade de processamento de uma técnica de mitigação de SEU, ou

seja, quando o tempo é fator crítico para o funcionamento da técnica, a abordagem via Model Checking Probabilístico talvez não seja a melhor escolha, já que FPGAs são dispositivos de alta capacidade computacional. Portanto, uma comparação que avalie esses critérios não traria resultados muito coerentes.

4.2.1.2 Análise de Confiabilidade

Para a análise da confiabilidade, tal qual a propriedade CSL estipulada para os modelos CTMCs, o objetivo é identificar qual a porcentagem de tempo em que uma técnica de mitigação de SEU esteve operacional ou degradada, ou seja, em estado funcional em relação ao tempo total de missão estipulado, e comparar esse dado com o que foi obtido no PRISM via modelos probabilísticos.

Para a análise da confiabilidade, o intuito foi aproximar o resultado das simulações funcionais com a propriedade CSL utilizada nos modelos CTMC, que retornava a probabilidade do sistema estar operacional ou degradado ao fim do período de missão. Portanto, uma forma simples de comparar foi calcular, dentro do tempo total, a porcentagem de tempo em que as técnicas permaneciam operacionais ou degradadas. Dessa forma, foi possível fazer o mesmo para os modelos CTMC e obter uma comparação justa e que reflete a confiabilidade tanto dos modelos quanto das técnicas simuladas. Assim como na disponibilidade, esse estudo de caso considerou a confiabilidade ao final dos 90 dias de missão.

Começando com o Scrubbing de configuração C1 (2 somadores e 2 multiplicadores), a Tabela 4.9 mostra que os resultados apresentados pelo PRISM foram pessimistas em confiabilidade quando comparados aos resultados das simulações no ModelSim, mas, em todos os intervalos de correção estipulados, os resultados foram próximos, principalmente quando os intervalos de correção são mais curtos, como no caso do intervalo de 1 dia. Dado que o Scrubbing desenvolvido não possui uma taxa de cobertura variável, ou seja, toda falha será prontamente detectada e o componente atingido removido de forma segura, a maneira mais justa de comparar com os modelos probabilísticos foi utilizar a taxa de cobertura em 100%. Dessa forma, todos as comparações entre o modelo PRISM do Scrubbing e a implementação funcional não envolvem variações na taxa de cobertura.

Tabela 4.9 - Comparação de confiabilidade do Scrubbing com 2 somadores e 2 multiplicadores e tempo de missão de 90 dias.

Intervalo de Scrubbing	**PRISM**	**ModelSim**
1 dia	98,79%	100%
4 dias	89,44%	100%
9 dias	74,44%	84,58%

Para corroborar o resultado anterior, quando a configuração C2 (10 somadores e 10 multiplicadores) é submetida à mesma análise, os resultados ficam ainda mais próximos, como mostra a Tabela 4.10. Com a quantidade maior de componentes, a frequência com que o sistema fica em estado de falha diminui, logo as taxas de confiabilidade sobem e a modelagem probabilística se aproxima mais do resultado obtido via implementação e simulação. Portanto, é interessante considerar o uso do Model Checking Probabilístico para aplicações que exijam uma alta confiabilidade, pois o modelo se aproximará com mais precisão dos resultados práticos.

Tabela 4.10 - Comparação de confiabilidade do Scrubbing com 10 somadores e 10 multiplicadores e tempo de missão de 90 dias.

Intervalo de Scrubbing	**PRISM**	**ModelSim**
1 dia	100%	100%
4 dias	99,70%	100%
9 dias	96,27%	93,48%

Para o TMR, a confiabilidade foi um ponto crítico assim como a disponibilidade. A Tabela 4.11 mostra que os resultados da técnica foram abaixo do esperado, mas que novamente o modelo probabilístico desenvolvido se aproximou no resultado obtido no ModelSim, inclusive apresentando uma confiabilidade superior no TMR. Na mesma tabela é possível observar que o código de Hamming seguiu os resultados obtidos na disponibilidade e mostrou que a velocidade de detecção e correção de um *bit flip* na implementação é maior que no PRISM, interferindo diretamente no tempo total em estado funcional da missão.

Tabela 4.11 - Comparação de confiabilidade do TMR e código de Hamming com tempo de missão de 90 dias.

Técnica de mitigação	**PRISM**	**ModelSim**
TMR	42,89%	39,50%
Código de Hamming	87,44%	100%

Portanto, apesar do TMR apresentar a necessidade de implementação em conjunto com alguma técnica de correção e não só de mascaramento, o Model Checking Probabilístico se aproximou do valor obtido via implementação e simulação, já no código de Hamming a confiabilidade da técnica foi excelente, mas a relação entre os resultados via PRISM e ModelSim ficou abaixo do esperado e mostra a diferença de desempenho no processamento da técnica e do modelo.

Para esse primeiro estudo de caso, considerando que o objetivo dos modelos desenvolvidos no PRISM era de se aproximar o máximo possível dos resultados obtidos na implementação funcional das técnicas, é possível afirmar que o Model Checking Probabilístico demonstrou um bom desempenho, tanto para a disponibilidade quanto para a confiabilidade, nas técnicas de Scrubbing (ambas as configurações) e TMR. Para o código de Hamming, como já dito anteriormente, os resultados não apresentaram grande proximidade, mas foram importantes para avaliar a precisão de tempo e desempenho computacional dos modelos em técnicas que exigem tais características.

4.2.2 CBERS-4A

Nesse estudo de caso, assim como no capítulo anterior, foi considerado um tempo de missão de 650 dias e, da mesma forma como foi feito no primeiro estudo de caso, as técnicas foram divididas pela categoria de funcionamento e comparadas com o resultado obtido via Model Checking Probabilístico. Além disso, esse estudo de caso apresenta duas taxas de *upset* distintas, uma para ionização por *heavy ions* (HUP) e outra por prótons (PUP). Essa variedade de taxas auxiliará na análise de desempenho das técnicas e na conclusão dos resultados comparativos.

4.2.2.1 Análise de Disponibilidade

Para essa análise, a simulação considerou 15600 horas de execução, ou seja, uma missão com duração de 2 anos. Para facilitar a visualização, os resultados HUP e PUP de uma técnica sob os mesmos parâmetros serão unidos em uma mesma tabela. Começando com o Scrubbing de configuração C1 (2 somadores e 2 multiplicadores) e intervalo de correção de 1 dia, a Tabela 4.12 apresenta os resultados para ambas as

taxas de *upset* HUP e PUP. É possível notar que em ambas as situações os resultados apresentados pelo PRISM se mantiveram próximos dos obtidos pela simulação no ModelSim, e que em um primeiro momento é difícil concluir que a taxa de *upset* maior do PUP interferiu de alguma forma na proximidade do resultado. A maior diferença foi no estado degradado de ambos, no qual o resultado apresentado pelo PRISM foi mais que o dobro da simulação via ModelSim, mas é muito importante notar que, apesar de proporcionalmente discrepantes entre si, a diferença em dias é a mesma apresentada pelos estados operacionais, de aproximadamente 12 dias para o HUP e 15 dias para o PUP. Isso mostra a consistência do *Model Checking* Probabilístico, pois dentro da mesma configuração, a diferença em dias dos resultados entre estados de operação diferentes não aumentam conforme o tempo de missão.

Tabela 4.12 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 1 dia.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	625,60	638,28	-1,98
Degradado	HUP	24,10	11,71	105,80
Falho	HUP	0,30	0,00	——
Operacional	PUP	604,92	620,78	-2,55
Degradado	PUP	44,00	29,21	50,63
Falho	PUP	1,07	0,00	——

As Tabelas 4.13 e 4.14 apresentam os resultados de disponibilidade para Scrubbing em C1 mas com taxas de correção de 4 e 9 dias, respectivamente. Para intervalos de correção maiores, foi possível observar que os modelos probabilísticos não se aproximaram tão bem quanto antes, principalmente no tempo em estado de falha, que subiu consideravelmente no modelo probabilístico, mas continuou baixo na simulação funcional. Esse resultado não foi perceptível no estudo de caso anterior (OEA) e, portanto, a inferência clara é que um tempo de missão muito longo pode ser fator de relevância na precisão dos modelos probabilísticos considerando os resultados de forma geral. Além disso, as taxas diferentes aliadas ao intervalo de correção maior tornaram os resultados ainda mais distantes, como no caso do intervalo de 9 dias.

Tabela 4.13 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 4 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	562,75	587,29	-4,17
Degradado	HUP	83,00	62,70	32,37
Falho	HUP	4,25	0,00	——
Operacional	PUP	500,40	552,17	-9,37
Degradado	PUP	136,00	97,82	39,03
Falho	PUP	13,60	0,00	——

Tabela 4.14 - Comparação de disponibilidade do Scrubbing com 2 somadores e 2 multiplicadores e intervalo de correção de 9 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	482,50	544,49	-11,38
Degradado	HUP	150,00	105,50	42,18
Falho	HUP	17,50	0,00	——
Operacional	PUP	390,00	425,72	-8,39
Degradado	PUP	211,00	224,27	-5,91
Falho	PUP	49,00	0,00	——

Continuando na técnica de Scrubbing, a análise agora será da configuração C2 (10 somadores e 10 multiplicadores) para verificar se a quantidade de componentes dificulta a precisão nos resultados do modelo considerando a disponibilidade. A Tabela 4.15 traz os resultados considerando o intervalo de correção de 1 dia, e assim como na configuração C1, nesse intervalo os resultados do modelo se aproximaram de forma satisfatória em relação ao obtidos via simulação em ambos os tipos de ionização. Para C2, ocorreu algo semelhante à C1 em relação ao estado degradado, principalmente pela baixa quantidade de dias, o percentual de diferença ficou mais alto do que o esperado, mas ainda próximo ao apresentado pelo estado operacional com aproximadamente 40 dias de diferença para o HUP e 45 dias para o PUP.

Tabela 4.15 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 1 dia.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	544,00	582,23	-6,56
Degradado	HUP	106,00	67,76	56,43
Falho	HUP	0,00	0,00	——
Operacional	PUP	471,99	516,05	-8,53
Degradado	PUP	178,00	133,94	32,89
Falho	PUP	0,00	0,00	——

Tal qual C1, as Tabelas 4.16 e 4.17 apresentam os resultados de C2 com 4 e 9 dias, respectivamente. E, diferentemente de C1, os resultados com um intervalo maior de correção não afetaram de forma negativa a precisão dos modelos probabilísticos. É possível inferir dessa análise que a quantidade maior de componentes pode ajudar a tornar a modelagem mais precisa no caso de intervalos de correção muito grandes, e que novamente a taxa (HUP e PUP) não alterou de forma significativa o desempenho do *Model Checking* Probabilístico, já que os modelos se aproximaram dos resultados da simulação em ambas as taxas. No entanto, o resultado mais equilibrado foi C2 considerando 4 dias de intervalo de Scrubbing, onde a diferença percentual entre os resultados dentro de um mesmo estado de operação ficou abaixo dos 12%.

Tabela 4.16 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 4 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	367,00	391,35	-6,22
Degradado	HUP	283,00	258,64	9,41
Falho	HUP	0,00	0,00	——
Operacional	PUP	260,99	294,44	-11,36
Degradado	PUP	389,00	355,55	9,40
Falho	PUP	0,00	0,00	——

Passando para o TMR, os resultados de disponibilidade para os dois tipos de ionização estão presentes na Tabela 4.18. Diferente do estudo de caso anterior, onde o modelo probabilístico se aproximou de forma satisfatória da simulação, nesse caso o modelo apresentou resultado aquém do esperado, possivelmente pelo tempo longo de missão e pelo desempenho insatisfatório da técnica. As taxas de falha diferentes não interferiram de forma significativa, já que em ambos os resultados a imprecisão dos modelos esteve presente.

Tabela 4.17 - Comparação de disponibilidade do Scrubbing com 10 somadores e 10 multiplicadores e intervalo de correção de 9 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	237,97	245,12	-2,91
Degradado	HUP	412,00	404,87	1,76
Falho	HUP	0,02	0,00	——
Operacional	PUP	150,30	117,31	28,12
Degradado	PUP	499,00	532,68	-6,32
Falho	PUP	0,70	0,00	——

Tabela 4.18 - Comparação de disponibilidade do TMR com tempo de missão de 650 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	27,00	67,20	-59,82
Degradado	HUP	239,00	134,40	77,82
Falho	HUP	384,00	448,39	-14,36
Operacional	PUP	21,00	34,96	-39,93
Degradado	PUP	182,00	69,92	160,29
Falho	PUP	447,00	545,11	-17,99

No código de Hamming, assim como no estudo de caso anterior, a velocidade de processamento da detecção e correção dos bits via simulação de FPGA tornou o resultado apresentado pelo modelo probabilístico inferior e comprometeu de certa forma a precisão dessa comparação. A Tabela 4.19 apresenta os resultados do código de Hamming em ambos os tipos de ionização e, quanto maior a taxa de falha, mais discrepante vai ficar o resultado por ressaltar a diferença de velocidade de processamento entre as abordagens.

Tabela 4.19 - Comparação de disponibilidade do código de Hamming com tempo de missão de 650 dias.

Estado	**Ionização**	**PRISM (dias)**	**ModelSim (dias)**	**Diferença (%)**
Operacional	HUP	632,90	649,99	-2,62
Degradado	HUP	2,10	$1,41 \times 10^{-5}$	$1,48 \times 10^{7}$
Falho	HUP	15,00	$1,87 \times 10^{-5}$	$8,02 \times 10^{7}$
Operacional	PUP	616,10	649,99	-5,21
Degradado	PUP	4,10	$3,70 \times 10^{-5}$	$1,10 \times 10^{7}$
Falho	PUP	29,80	$4,68 \times 10^{-5}$	$6,36 \times 10^{7}$

4.2.2.2 Análise de Confiabilidade

Na análise de confiabilidade é possível analisar a proporção do tempo em que a técnica se manteve operacional ou degradada em relação ao tempo total de missão. Iniciando com o Scrubbing com configuração C1 e intervalos de correção de 1, 4 e 9 dias para as duas taxas de *upset*, HUP e PUP como mostra a Tabela 4.20. Nesse estudo de caso, os resultados foram superiores aos apresentados no estudo anterior (Tabela 4.9), onde a diferença chegou a mais de 10% para o intervalo de correção de 4 dias. Nesse caso, o modelo probabilístico se aproximou muito dos 100% retornados pela simulação no ModelSim, que em primeiro momento essa melhoria pode ser atribuída ao tempo de missão maior em relação à taxa de *upsets*, que mesmo na maior taxa (PUP) conseguiu apresentar bons resultados. Assim como na análise anterior, a taxa de cobertura foi fixada em 100% nos modelos probabilísticos para obter uma comparação justa com as técnicas simuladas no ModelSim.

Tabela 4.20 - Comparação de confiabilidade do Scrubbing com 2 somadores e 2 multiplicadores e tempo de missão de 650 dias.

Intervalo de Scrubbing	**Ionização**	**PRISM**	**ModelSim**
1 dia	HUP	99,95%	100%
4 dias	HUP	99,35%	100%
9 dias	HUP	97,31%	100%
1 dia	PUP	99,83%	100%
4 dias	PUP	97,91%	100%
9 dias	PUP	92,46%	100%

Como era de se esperar, assim como no estudo de caso anterior onde a configuração C2 conseguiu melhorar os resultados comparativos entre as duas abordagens, o mesmo ocorreu nesse estudo de caso, onde em praticamente todas as situações os modelos probabilísticos alcançaram os 100% de confiabilidade retornados pelas simulações no ModelSim, como mostra a Tabela 4.21. É possível concluir a análise do Scrubbing afirmando que a abordagem probabilística apresentou resultados muito precisos na grande maioria das configurações estipuladas nesse trabalho, e a aproximação maior ocorreu na configuração que possui mais componentes sobressalentes, ou seja, menos chance de entrar em estado de falha. O intervalo de Scrubbing deve ser levado em consideração no caso de aplicações que utilizem poucos componentes, onde intervalos menores podem melhorar os resultados dos modelos.

Tabela 4.21 - Comparação de confiabilidade do Scrubbing com 10 somadores e 10 multiplicadores e tempo de missão de 650 dias.

Intervalo de Scrubbing	**Ionização**	**PRISM**	**ModelSim**
1 dia	HUP	100%	100%
4 dias	HUP	100%	100%
9 dias	HUP	100%	100%
1 dia	PUP	100%	100%
4 dias	PUP	100%	100%
9 dias	PUP	99,89%	100%

Para o TMR, novamente a abordagem probabilística foi limitada pelo desempenho insatisfatório da técnica. Conforme a taxa de *upset* aumenta, o modelo probabilístico se distancia do resultado retornado pela simulação via ModelSim, como mostra a Tabela 4.22. Para a análise de confiabilidade do TMR, é possível concluir que a taxa de *upset* juntamente com o desempenho da técnica foram os fatores principais no resultado abaixo do esperado do modelo probabilístico de uma forma geral. Portanto, para utilizar o PRISM na análise do TMR, é importante verificar as taxas utilizadas, já que esse pode ser o fator limitante de desempenho. Isso fica claro nessa análise, pois o mesmo modelo submetido a taxas de *upset* distintas apresentou um resultado interessante no HUP (menos de 10% de diferença) e no PUP ficou abaixo do esperado (mais de 15%).

Tabela 4.22 - Comparação de confiabilidade do TMR com tempo de missão de 650 dias.

Ionização	**PRISM**	**ModelSim**
HUP	40,92%	31,02%
PUP	31,23%	16,14%

Por fim, para o código de Hamming, é possível observar resultados excelentes, ou seja, uma mudança com o que foi apresentado nas análises anteriores, pois para aplicações com tempos de missão maiores, as diferenças na velocidade de processamento podem se tornar desprezíveis, como mostra a Tabela 4.23. Apesar da diferença, em um contexto de 650 dias, o modelo probabilístico conseguiu se manter a menos de 5% de diferença em confiabilidade da implementação simulada no ModelSim, o que torna o tempo de missão um dos critérios mais relevantes para avaliar o código de Hamming no quesito confiabilidade.

Tabela 4.23 - Comparação de confiabilidade do código de Hamming com tempo de missão de 650 dias.

Ionização	**PRISM**	**ModelSim**
HUP	97,69%	100%
PUP	95,42%	100%

Para esse estudo de caso, considerando o atributo de disponibilidade, é possível concluir que no Scrubbing os resultados foram plausíveis, principalmente na configuração C2, no qual os intervalos de correção mais longos contribuíram para uma diferença menor entre os resultados apresentados por ModelSim e PRISM. Para C1, apesar do estado degradado ter despontado no percentual de diferença como já dito anteriormente, considerando o tempo total de missão, o resultado ainda é aceitável. O mesmo não ocorreu com o TMR e código de Hamming, onde a diferença entre os resultados do ModelSim e PRISM foi insatisfatória, situação claramente agravada em relação ao estudo de caso anterior devido ao tempo de missão maior e taxas de *upset* mais altas.

Para a disponibilidade, os resultados foram excelentes para o Scrubbing em ambas as configurações C1 e C2, mostrando que percentualmente o modelo PRISM foi capaz de se aproximar da confiabilidade da técnica simulada no ModelSim com precisão maior do que o primeiro estudo de caso. Já para o TMR, a análise comparativa mostrou que a diferença entre os resultados foi mais próxima na ionização direta do que na indireta, mostrando que a taxa de *upset* maior pode afetar o desempenho da técnica assim como o do modelo. Entretanto, o resultado surpreendente da disponibilidade foi o código de Hamming, que diferente do estudo de caso anterior, apresentou bons resultados tanto em HUP quanto em PUP. É plausível assumir que o tempo de missão maior foi fundamental para que a discrepância na velocidade de processamento da técnica e do modelo fosse desprezada.

No geral, o *Model Checking* Probabilístico manteve os bons resultados apresentados no primeiro estudo de caso, e mesmo que algumas situações tenham afetado negativamente os modelos, não houve uma comparação em que a ordem os resultados tenham se invertido, ou seja, em todas as análises, os modelos foram capazes de descobrir corretamente quais estados (operacional, degradado e falho) ocuparam a maior e a menor parte do tempo total de missão.

4.3 Considerações finais sobre esse Capítulo

Esse capítulo apresentou todo o processo de desenvolvimento e simulação funcional das técnicas de mitigação de SEU em FPGAS SRAM especificadas nesse trabalho. Além disso, foram apresentados os resultados dos dois estudos de caso analisados, assim como uma comparação desses resultados com os obtidos via *Model Checking* Probabilístico, com o intuito de avaliar o desempenho dos modelos CTMC frente as implementações em VHDL.

Uma síntese interessante dos resultados obtidos é que para o Scrubbing, o critério fundamental para a decisão de utilizar os modelos probabilísticos é avaliar a quantidade de componentes sobressalentes e que serão corrigidos em cada ciclo de Scrubbing, pois o desempenho dos modelos PRISM na configuração C2 foi maior que C1 em todos os testes. No TMR, o fator principal a ser levado em consideração é a taxa de *upsets*, pois quando a ocorrência de falhas é muito alta, o desempenho do TMR cai consideravelmente, tornando o modelo probabilístico impreciso. Por fim, o código de Hamming permitiu enxergar o abismo na diferença de velocidade de processamento entre uma técnica implementada em hardware e a mesma técnica modelada via PRISM, mas também é necessário considerar o tempo de missão, pois como dito anteriormente, para longos períodos essa diferença no tempo de processamento pode se tornar desprezível. Outras razões para a diferença entre os resultados do código de Hamming podem estar relacionadas com as características intrínsecas de cada implementação, pois o modelo é uma representação do funcionamento da técnica. É importante ressaltar que o segundo estudo de caso, por considerar um tempo de missão maior e duas taxas distintas de *upsets*, permitiu uma análise mais aprofundada dos resultados obtidos e das comparações, pois em algumas situações uma determinada técnica pode apresentar resultados divergentes, alterando apenas características de órbita e de radiação.

Por fim, considerando o desempenho do *Model Checking* Probabilístico, ou seja, sua proximidade em relação aos resultados apresentados pela simulação funcional do ModelSim, é possível afirmar que *Model Checking* Probabilístico apresentou um desempenho adequado e promissor. Além disso, como mencionado anteriormente, não houve situação em que, dentro de uma mesma análise, os modelos apresentassem resultados contrários aos obtidos via ModelSim. Claramente existiram técnicas que, dado o seu paradigma de concepção e funcionamento, foram capazes de manter um resultado próximo em praticamente todas as comparações. As Tabelas 4.24 e 4.25 resumem o desempenho do *Model Checking* Probabilístico na análise comparativa

realizada nesse capítulo, para o primeiro e o segundo estudo de caso, respectivamente. O desempenho foi dividido em três categorias: alto (na qual as comparações apresentaram diferença percentual absoluta abaixo dos 20%), médio (onde as comparações tiveram diferença percentual absoluta entre 20% e 70%) e baixo (as comparações apresentaram diferença percentual absoluta acima dos 70%), tomando como referência o resultado obtido via simulação no ModelSim.

Tabela 4.24 - Desempenho do Model Checking Probabilístico na análise comparativa: estudo de caso Órbita Elíptica Alta (OEA).

Atributo	**Scrubbing**	**TMR**	**Código de Hamming**
Disponibilidade	Alto	Médio	Baixo
Confiabilidade	Alto	Alto	Médio

Tabela 4.25 - Desempenho do Model Checking Probabilístico na análise comparativa: estudo de caso CBERS-4A.

Atributo	**Scrubbing**	**TMR**	**Código de Hamming**
Disponibilidade	Médio	Baixo	Baixo
Confiabilidade	Alto	Médio	Alto

O capítulo seguinte apresenta as considerações finais desse trabalho, assim como suas contribuições e detalhes importantes a serem desenvolvidos futuramente para aprimorar esse estudo.

5 CONCLUSÃO

Comparações realizadas para avaliar técnicas de mitigação de SEU em FPGA têm sido propostas na literatura (SHULER et al., 2009; MORGAN et al., 2007; LIU et al., 2012; HENTSCHKE et al., 2002). No entanto, a maioria desses trabalhos foram conduzidos após implementar as técnicas em FPGA e simular os *upsets* com ferramentas de injeção de falhas. Esse tipo de abordagem pode ser custosa, uma vez que apresenta os resultados apenas ao final das simulações e com o esforço para realizar a implementação em FPGA. Por outro lado, modelos estocásticos/probabilísticos podem ser utilizados nos estágios iniciais de um projeto para avaliar técnicas de mitigação de SEU em FPGA, e podem amortizar o custo relacionado a implementação. Eventualmente, uma abordagem baseada em modelos probabilísticos não significa, necessariamente, eliminar completamente as abordagens que se baseiam em implementação, mas os resultados de tal análise via modelos probabilísticos podem complementar as abordagens baseadas em implementação.

Análises probabilísticas via métodos de Verificação Formal ainda são pouco encontradas na literatura. Em (HOQUE et al., 2014) e (HOQUE, 2016), os autores apresentaram estudos de caso utilizando *Model Checking* Probabilístico para a análise de dependabilidade de técnicas de SEU em FPGA. Essa dissertação de mestrado é diferente desses trabalhos anteriores em diversas formas, começando com a análise individual das técnicas ao invés de um modelo único que combina duas técnicas. Além disso, o modelo TMR desenvolvido nesse trabalho não lida somente com componentes sobressalentes, mas também inclui o mecanismo de votação e os bits de entrada e saída. No Scrubbing, um cenário diferente foi abordado por meio do aumento da quantidade de componentes analisados. Utilizando *Model Checking* Probabilístico, é possível afirmar que os benefícios são altos, pois os resultados são obtidos nos estágios iniciais do desenvolvimento, sem os custos e os esforços gastos projetando e implementando em FPGA e com ferramentas de injeção de falhas.

Portanto, o objetivo dessa dissertação de mestrado é investigar a viabilidade, no contexto de aplicações espaciais, da utilização do Model Checking Probabilístico para determinar qual a melhor técnica de mitigação de SEU em FPGAs SRAM, dentre as soluções apresentadas. Para isso, inicialmente as técnicas de mitigação de SEU Scrubbing, TMR e código de Hamming foram estudadas e modeladas no *Model Checker* PRISM e, em seguida, os modelos foram comparados em dois estudos de caso, o primeiro considerando uma aplicação em Órbita Elíptica Alta (OEA) e um tempo de missão de 90 dias, e o segundo utilizando os dados orbitais do satélite

CBERS-4A e um tempo de missão de 650 dias. Ambas as análises consideraram três atributos de dependabilidade: disponibilidade, segurança e confiabilidade.

Em seguida, para analisar a viabilidade da utilização dos modelos probabilísticos, as mesmas três técnicas foram implementadas em VHDL no ambiente Altera Quartus II e simuladas por meio do ModelSim para ambos os estudos de caso. Nessa fase, foram analisadas a disponibilidade e a confiabilidade das técnicas mantendo os mesmos critérios da análise probabilística desenvolvida anteriormente. Diferente da análise anterior que buscava descobrir características das técnicas através da modelagem probabilística, o objetivo dessa análise é comparar os resultados dos modelos probabilísticos com as simulações das técnicas implementadas em VHDL.

Considerando inicialmente os resultados das análises do *Model Checking* Probabilístico, foi possível concluir que ambos os estudos de caso se alinharam nos resultados, mesmo com condições de órbita e tempos de missão distintos. Para a disponibilidade, o melhor resultado foi obtido pelo código de Hamming, pois foi a técnica que se manteve operacional por mais tempo, seguida pelo Scrubbing com intervalo de correção de 1 dia e 9 dias, enquanto o TMR apresentou resultados insatisfatórios e revelou a necessidade de uma técnica de correção em conjunto para evitar o acúmulo de *upsets*. Na confiabilidade, os melhores resultados surgiram do Scrubbing, que foi diretamente influenciado pelo tamanho do intervalo entre as correções, especialmente na configuração C1. Já na segurança, ocorreu o inverso, a taxa de cobertura foi o critério mais impactante nos resultados de ambas as configurações.

Para as análises comparativas entre os modelos probabilísticos e a simulação funcional, diferente dos resultados anteriores, os estudos de caso não se alinharam totalmente, e as condições de órbita e tempos de missão foram fatores impactantes nos resultados e devem ser levados em consideração. No geral, para o Scrubbing, a quantidade de componentes deve ser avaliada no uso de um modelo probabilístico, pois o desempenho apresentado pelo configuração C2 foi superior em ambos os estudos de caso. Para o TMR, é importante averiguar a taxa de *upsets*, pois as taxas mais elevadas distanciaram o modelo do resultado da simulação. Por fim, o código de Hamming foi afetado por um critério que não havia sido considerado anteriormente, que é o tempo de processamento necessário para a técnica obter seu funcionamento adequado. Nesse caso, o melhor resultado foi obtido no estudo de caso com maior tempo de missão, pois em longos períodos a diferença no desempenho do modelo pode se tornar desprezível. Portanto, no caso de técnicas que possuam características de funcionamento que exijam alta capacidade de processamento para que seu

desempenho não seja afetado, como é o caso do código de Hamming, as abordagens baseadas em modelos podem apresentar resultados insatisfatórios e por isso não são a melhor forma de avaliar o desempenho desse tipo de técnica de mitigação de SEU.

Dado o desempenho geral do *Model Checking* Probabilístico, principalmente considerando os resultados apresentados e as comparações com as simulações obtidas via ModelSim, é possível classificá-lo como promissor, já que foi adequado em grande parte das análises e, nas situações em que apresentou resultado insatisfatório, ainda foi capaz de indicar a tendência correta, pois não houve inversão entre os resultados obtidos em estados de funcionamento distintos em relação ao obtido na simulação funcional.

Portanto, dado o objetivo do trabalho, é possível confirmar a viabilidade do uso de técnicas de modelagem probabilísticas, em especial do *Model Checking* Probabilístico, para determinar qual técnica de mitigação de SEU em FPGAs SRAM apresenta melhor desempenho dentro dos critérios estabelecidos pelo estudo de caso considerado. Dessa forma, as contribuições dessa dissertação de mestrado surgem no contexto de desenvolvimento de projetos espaciais, como os desenvolvidos no Instituto Nacional de Pesquisas Espaciais (INPE), pois possibilitam a obtenção de resultados, em estágios iniciais de desenvolvimento, do comportamento das técnicas de mitigação de SEU em FPGAs, e como consequência a redução nos custos de projeto, afetados principalmente pelo valor dos dispositivos, e no tempo necessário para avaliar e testar seu desempenho sob radiação.

5.1 Trabalhos Futuros

Com o objetivo dessa dissertação de mestrado atingido, os próximos passos para a continuação desse projeto são:

a) Estudar e analisar o espaço de parâmetros e configurações utilizados no modelamento probabilístico e simulações, para tentar encontrar padrões nas técnicas de mitigação considerando diferentes perspectivas;

b) Implementar em FPGA SRAM as mesmas três técnicas consideradas nesse trabalho (TMR, *Scrubbing*, Código de Hamming) e avaliar seu comportamento sob radiação;

c) Considerar outros estudos de caso para obter mais resultados dos modelos probabilísticos desenvolvidos;

d) Desenvolver modelos probabilísticos para outras técnicas de mitigação ou até mesmo combinar mais de uma técnica em um único modelo para avaliar seus resultados.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, J.; SILBERBERG, R.; TSAO, C.-H. Cosmic ray effects on microelectronics. part 1. the near-earth particle environment. **Washington: Naval Reasearch Laboratory**, 1981. 95p. 22

ALMEIDA, G. M.; BEZERRA, E. A.; CARGNINI, L. V.; FAGUNDES, R. D. R.; MESQUITA, D. G. A reed-solomon algorithm for fpga area optimization in space applications. In: ADAPTIVE HARDWARE AND SYSTEMS, 2., 2007. **Proceedings...** Edinburgh: IEEE, 2007. p. 243–249. 2

ALTERA, Q. I. Version 7.2 handbook. **Altera Inc**, v. 1, 2007. 5, 112

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, 2004. 4, 9

BAIER, C.; HAVERKORT, B.; HERMANNS, H.; KATOEN, J.-P. Model-checking algorithms for continuous-time markov chains. **IEEE Transactions on Software Engineering**, v. 29, n. 6, p. 524–541, 2003. 4, 28, 29

BAIER, C.; KATOEN, J.-P.; LARSEN, K. G. **Principles of model checking**. [S.l.]: MIT press, 2008. 3, 27, 28, 29

BERG, M.; POIVEY, C.; PETRICK, D.; ESPINOSA, D.; LESEA, A.; LABEL, K. A.; FRIENDLICH, M.; KIM, H.; PHAN, A. Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: design, test, and analysis. **IEEE Transactions on Nuclear Science**, v. 55, n. 4, p. 2259–2266, Aug. 2008. ISSN 0018-9499. 2, 4, 17, 18

BERNARDESCHI, C.; CASSANO, L.; DOMENICI, A. SRAM-Based FPGA systems for safety-critical applications: a survey on design standards and proposed methodologies. **Journal of Computer Science and Technology**, v. 30, n. 2, p. 373, 2015. 21

BERNARDI, P.; REORDA, M. S.; STERPONE, L.; VIOLANTE, M. On the evaluation of seu sensitiveness in SRAM-based FPGAS. In: IEEE INTERNATIONAL ON-LINE TESTING SYMPOSIUM, 10., 2004. **Proceedings...** [S.l.]: IEEE, 2004. p. 115–120. 13

BIROLINI, A. **Reliability engineering: theory and practice**. [S.l.]: Springer, 2017. 21

BROESCH, J. **Practical Programmable Circuits: A Guide to PLDs, State Machines, and Microcontrollers**. [S.l.]: Elsevier Science, 2012. 110

BROSSER, F.; MILH, E. **Seu mitigation techniques for advanced reprogrammable fpga in space**. [S.l.]: Gothenburg: Chalmers University of Technology, 2014. 13, 14, 15

BROWN, S.; FRANCIS, R.; ROSE, J.; VRANESIC, Z. **Field-Programmable Gate Arrays**. [S.l.]: Springer US, 2012. 112

CHOUDHURY, D. R.; PODDER, K. Design of hamming code encoding and decoding circuit using transmission gate logic. **International Research Journal of Engineering and Technology**, v. 2, n. 7, p. 1165–1169, 2015. 78

CLARKE, E.; GRUMBERG, O.; PELED, D. **Model Checking**. [S.l.]: MIT Press, 1999. 27

CLARKE, E. M.; EMERSON, E. A.; SISTLA, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, v. 8, n. 2, p. 244–263, 1986. 28

CLARKE, E. M.; WING, J. M. Formal methods: state of the art and future directions. **ACM Computing Surveys (CSUR)**, v. 28, n. 4, p. 626–643, 1996. 1

DODD, P. E.; SHANEYFELT, M. R.; SCHWANK, J. R.; HASH, G. L. Neutron-induced latchup in srams at ground level. In: IEEE ANNUAL INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 41., 2003, Dallas, Texas. **Proceedings...** [S.l.]: IEEE, 2003. p. 51–55. 13

DUTTA, A.; TOUBA, N. A. Multiple bit upset tolerant memory using a selective cycle avoidance based sec-ded-daec code. In: IEEE VLSI TEST SYMPOSIUM, 25., 2007, Berkeley, CA. **Proceedings...** [S.l.]: IEEE, 2007. p. 349–354. 2, 18

EDWARDS, R.; DYER, C.; NORMAND, E. Technical standard for atmospheric radiation single event effects,(see) on avionics electronics. In: IEEE RADIATION EFFECTS DATA WORKSHOP, 2004, Atlanta, GA. **Proceedings...** [S.l.]: IEEE, 2004. p. 1–5. 13

EJLALI, A.; AL-HASHIMI, B. M.; SCHMITZ, M. T.; ROSINGER, P.; MIREMADI, S. G. Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, IEEE, v. 14, n. 4, p. 323–335, 2006. 3

ELAKKUMANAN, P.; PRASAD, K.; SRIDHAR, R. Time redundancy based scan flip-flop reuse to reduce ser of combinational logic. In: IEEE QUALITY ELECTRONIC DESIGN, 2006, San Jose, CA. **Proceedings...** [S.l.]: IEEE, 2006. p. 6–pp. 2

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-Q-ST-10C**: space product assurance: product assurance management. Noordwijk, 2008. 25p. 19, 20

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-M-ST-10C**: space project management: project planning and implementation. Noordwijk, 2009. 50p. 3, 5

FULKS, C.; COFER, R. Best fpga development practices. In: DESIGN WEST CONFERENCE, 2012. **Proceedings...** [S.l.]: Intuitive, 2012. 69, 70

GAILLARD, R. Single event effects: mechanisms and classification. In: **NICOLAIDIS, M. (Ed.). Soft errors in modern electronic systems**. [S.l.]: Springer, 2011. p. 27–54. 10

GRAHAM, P.; CAFFREY, M.; ZIMMERMAN, J.; SUNDARARAJAN, P.; JOHNSON, E. Consequences and categories of SRAM FPGA configuration SEUs. In: CITESEER. In: MAPLD INTERNATIONAL CONFERENCE, 6., 2003. **Proceedings...** Washington: NASA, 2003. 1, 16

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation**. [S.l.]: Elsevier Science, 2010. (Systems on Silicon). ISBN 9780080556017. 109

HEINER, J.; SELLERS, B.; WIRTHLIN, M.; KALB, J. FPGA partial reconfiguration via configuration scrubbing. In: IEEE FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 2009, Prague, Czech Republic. **Proceedings...** [S.l.]: IEEE, 2009. p. 99–104. 2, 17

HENTSCHKE, R.; MARQUES, F.; LIMA, F.; CARRO, L.; SUSIN, A.; REIS, R. Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy. In: IEEE SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 15., 2002, Porto Alegre, Brazil. **Proceedings...** [S.l.]: IEEE, 2002. p. 95–100. 2, 3, 19, 97

HERRERA-ALZU, I.; LOPEZ-VALLEJO, M. Design techniques for xilinx virtex fpga configuration memory scrubbers. **IEEE Transactions on Nuclear Science**, v. 60, n. 1, p. 376–385, 2013. 14

HOQUE, K. A. **Early dependability analysis of FPGA-based space applications using formal verification**. Tese (Doutorado) — Tese (Doutorado em Filosofia), Concordia University Montréal, Québec, Canada, 2016. xi, 3, 18, 21, 32, 33, 35, 36, 37, 38, 39, 40, 45, 46, 47, 48, 49, 50, 52, 81, 97

HOQUE, K. A.; MOHAMED, O. A.; SAVARIA, Y.; THIBEAULT, C. Probabilistic model checking based dal analysis to optimize a combined tmr-blind-scrubbing mitigation technique for fpga-based aerospace applications. In: ACM/IEEE CONFERENCE ON FORMAL METHODS AND MODELS FOR CODESIGN, 14., 2014. **Proceedings...** [S.l.]: IEEE, 2014. p. 175–184. 3, 21, 28, 35, 97

HOWE, C. L.; WELLER, R. A.; REED, R. A.; MENDENHALL, M. H.; SCHRIMPF, R. D.; WARREN, K. M.; BALL, D. R.; MASSENGILL, L. W.; LABEL, K. A.; HOWARD, J. W. et al. Role of heavy-ion nuclear reactions in determining on-orbit single event error rates. **IEEE Transactions on Nuclear Science**, v. 52, n. 6, p. 2182–2188, 2005. 22

INPE. **Projeto CITAR testa componente eletrônico tolerante à radiação para uso em sistemas espaciais**. 2016. Disponível em: http://www.inpe.br/noticias/noticia.php?Cod_Noticia=4115. Acesso em: 30 abr. 2017. 4, 81

IONESCU, L. M.; ANTON, C.; TUTANESCU, I.; MAZARE, A.; SERBAN, G. Hardware implementation of bch error-correcting codes on a fpga. **International Journal of Intelligent Computing Research (IJICR)**, v. 1, n. 3, p. 148–153, 2010. 2

KASTENSMIDT, F. L. **Designing single event upset mitigation techniques for large SRAM-based FPGA components**. Tese (Doutorado) — Tese (Doutorado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002. 14, 15

KASTENSMIDT, F. L. See mitigation strategies for digital circuit design applicable to asic and fpgas. In: IEEE NSREC SHORT COURSE, 44., 2007, Honolulu, Hawai. **Proceedings...** [S.l.]: IEEE, 2007. 1, 2, 16

KASTENSMIDT, F. L.; STERPONE, L.; CARRO, L.; REORDA, M. S. On the optimal design of triple modular redundancy logic for sram-based fpgas. In: IEEE

DESIGN, AUTOMATION AND TEST IN EUROPE, 2005, Munich, German. **Proceedings...** [S.l.], 2005. p. 1290–1295. 4, 16, 19

KOGA, R.; GEORGE, J.; SWIFT, G.; YUI, C.; EDMONDS, L.; CARMICHAEL, C.; LANGLEY, T.; MURRAY, P.; LANES, K.; NAPIER, M. Comparison of xilinx virtex-ii fpga see sensitivities to protons and heavy ions. **IEEE Transactions on Nuclear Science**, v. 51, n. 5, p. 2825–2833, 2004. 22

KUMAR, U.; UMASHANKAR, B. Improved hamming code for error detection and correction. In: INTERNATIONAL SYMPOSIUM ON WIRELESS PERVASIVE COMPUTING, 2., 2007, San Juan, Puerto Rico. **Proceedings...** [S.l.]: IEEE, 2007. 2, 4

KWIATKOWSKA, M.; NORMAN, G.; PARKER, D. Stochastic model checking. In: BERNARDO, M.; HILLSTON, J. (EDS.). **Formal methods for peformance evaluation: International School on Formal Methods for the Design of Computer, Communication and Software Systems.** [S.l.]: Springer, 2007. p. 220–270. 28, 31

_____. Prism: probabilistic model checking for performance and reliability analysis. **ACM SIGMETRICS Performance Evaluation Review**, v. 36, n. 4, p. 40–45, 2009. 3, 4, 28, 31, 113

LAL, V. **VHDL code for wallace tree multiplication**. 2013. Disponível em: <http://vhdlguru.blogspot.com.br/2013/08/vhdl-code-for-wallace-tree.html>. 75

LIU, S.; SORRENTI, G.; REVIRIEGO, P.; CASINI, F.; MAESTRO, J.; ALDERIGHI, M.; MECHA, H. Comparison of the susceptibility to soft errors of sram-based fpga error correction codes implementations. **IEEE Transactions on Nuclear Science**, v. 59, n. 3, p. 619–624, 2012. 2, 3, 19, 97

LOJEK, B. **History of Semiconductor Engineering**. [S.l.]: Springer Berlin Heidelberg, 2007. 109

MAURI, G. R. **8 bit kogge stone adder**. Kanpur, 2008. Project Report. 74

MAXFIELD, C. **The Design Warrior's Guide to FPGAs: Devices, Tools and Flows**. [S.l.]: Elsevier Science, 2004. 109, 110, 111, 112

MENTOR GRAPHICS. **Modelsim-advanced simulation and debugging**. [S.l.]: Wilsonville: Mentor Graphics Corporation, 2012. 5, 113

MORGAN, K. S.; MCMURTREY, D. L.; PRATT, B. H.; WIRTHLIN, M. J. A comparison of TMR with alternative fault-tolerant design techniques for fpgas. **IEEE transactions on nuclear science**, v. 54, n. 6, p. 2065–2072, 2007. 2, 3, 16, 17, 97

NAVABI, Z. **Digital Design and Implementation with Field Programmable Devices**. [S.l.]: Springer US, 2006. 111

NEUBERGER, G.; LIMA, F. D.; CARRO, L.; REIS, R. A multiple bit upset tolerant SRAM memory. **ACM Transactions on Design Automation of Electronic Systems (TODAES)**, v. 8, n. 4, p. 577–590, 2003. 1

NORMAND, E. Single-event effects in avionics. **IEEE Transactions on nuclear science**, v. 43, n. 2, p. 461–474, 1996. 10

_____. Single event upset at ground level. **IEEE transactions on Nuclear Science**, v. 43, n. 6, p. 2742–2750, 1996. 10, 12

OSTLER, P. S.; CAFFREY, M. P.; GIBELYOU, D. S.; GRAHAM, P. S.; MORGAN, K. S.; PRATT, B. H.; QUINN, H. M.; WIRTHLIN, M. J. SRAM FPGA reliability analysis for harsh radiation environments. **IEEE Transactions on Nuclear Science**, v. 56, n. 6, p. 3519–3526, 2009. 2

PEREIRA, V. C.; SANTIAGO JÚNIOR, V. A.; MANEA, S. SEU mitigation for SRAM FPGAs: a comparison via probabilistic model checking. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 35.; WORKSHOP DE TESTES E TOLERÂNCIA A FALHAS, 18., 2017, Belém, Pará. **Anais...** [S.l.]: SBC, 2017. 33, 46

REBAUDENGO, M.; REORDA, M. S.; VIOLANTE, M. Simulation-based analysis of seu effects on sram-based fpgas. In: GLESNER, M.; ZIPF, P.; RENOVELL, M. (EDS.). **Field-programmable logic and applications: reconfigurable computing is going mainstream: International Conference FPL, 2002.** Berlin: Springer, 2002. p. 607–615. 1

ROLLINS, N.; WIRTHLIN, M.; CAFFREY, M.; GRAHAM, P. Evaluating TMR techniques in the presence of single event upsets. In: ANNUAL INTERNATIONAL CONFERENCE ON MILITARY AND AEROSPACE PROGRAMMABLE LOGIC DEVICES, 6., 2003. **Proceedings...** [S.l.], 2003. p. 63. 2

SANTIAGO JÚNIOR, V. A. **Model checking probabilístico**. São José dos Campos: INPE, 2016. Notas de aula. 29

SARI, A.; PSARAKIS, M.; GIZOPOULOS, D. Combining checkpointing and scrubbing in fpga-based real-time systems. In: IEEE VLSI TEST SYMPOSIUM, 31., 2013, Berkeley, CA. **Proceedings...** [S.l.]: IEEE, 2013. p. 1–6. 3, 17

SAWANT, M. Single event effects complicate military avionics systems design. **Journal of Military Electronics and Computing**, 2012. 11

SELIC, B. The pragmatics of model-driven development. **IEEE Software**, v. 20, n. 5, p. 19–25, 2003. 6

SHULER, R. L.; BHUVA, B. L.; O'NEILL, P. M.; GAMBLES, J. W.; REZGUI, S. Comparison of dual-rail and tmr logic cost effectiveness and suitability for fpgas with reconfigurable seu tolerance. **IEEE Transactions on Nuclear Science**, v. 56, n. 1, p. 214–219, 2009. 2, 3, 17, 97

SIMS, A.; DYER, C.; PEERLESS, C.; JOHANSSON, K.; PETTERSSON, H.; FARREN, J. The single event upset environment for avionics at high latitude. **IEEE transactions on nuclear science**, v. 41, n. 6, p. 2361–2367, 1994. 13

STASSINOPOULOS, E.; RAYMOND, J. P. The space radiation environment for electronics. **Proceedings of the IEEE**, v. 76, n. 11, p. 1423–1442, 1988. 1

STAVINOV, E. **100 Power Tips for FPGA Designers**. [S.l.]: Evgeni Stavinov, 2011. 112

SWIFT, G.; GREGORY, A. **Virtex-5qv static seu characterization summary**. [S.l.]: Jet Propulsion Laboratory, 2012. 44 p. 25

TRIMBERGER, S. **Field-Programmable Gate Array Technology**. [S.l.]: Springer US, 2012. 112

TYLKA, A. J.; ADAMS, J. H.; BOBERG, P. R.; BROWNSTEIN, B.; DIETRICH, W. F.; FLUECKIGER, E. O.; PETERSEN, E. L.; SHEA, M. A.; SMART, D. F.; SMITH, E. C. Creme96: a revision of the cosmic ray effects on micro-electronics code. **IEEE Transactions on Nuclear Science**, v. 44, n. 6, p. 2150–2160, 1997. 22

VINTER, J.; HANNIUS, O.; NORLANDER, T.; FOLKESSON, P.; KARLSSON, J. Experimental dependability evaluation of a fail-bounded jet engine control system for unmanned aerial vehicles. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2005, Yokohama, Japan. **Proceedings...** [S.l.]: IEEE, 2005. p. 666–671. 13

WANG, J. **Radiation effects in FPGAs**. [S.l.]: Cern, 2003. 13

WANG, J.; KATZ, R.; SUN, J.; CRONQUIST, B.; MCCOLLUM, J.; SPEERS, T.; PLANTS, W. Sram based re-programmable FPGA for space applications. **IEEE Transactions on Nuclear Science**, v. 46, n. 6, p. 1728–1735, 1999. 1, 13

WANG, J.-J.; CRONQUIST, B. E.; SIN, B.; MORIARTA, J. J.; KATZ, R. B. **Antifuse fpga for space applications**. 1997. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.4445&rep=rep1&type=pdf. 13

WHITE, D. **Considerations surrounding single event effects in fpgas, asics, and processors**. 2012. Disponível em: https://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf. 10, 12

WIDMANN, D.; MADER, H.; FRIEDRICH, H. **Technology of Integrated Circuits**. [S.l.]: Springer, 2000. 109

ZEIDMAN, B. **Designing with FPGAs and CPLDs**. [S.l.]: Taylor & Francis, 2002. 111

APÊNDICE A

Esse apêndice apresenta a evolução dos circuitos digitais desde sua formação até os dias atuais, a importância da FPGA como dispositivo programável, e também uma breve descrição das ferramentas de modelagem e simulação utilizadas nesse trabalho.

A.1 A Tecnologia FPGA

Apesar de parecer uma tecnologia recente, a FPGA se tornou o resultado de inúmeras tecnologias anteriores, que a cada geração acrescentavam alguma melhoria, resultando assim no que conhecemos atualmente. Um dos primeiros passos ocorreu um 1947, quando físicos que trabalhavam na Bell Laboratories nos Estados Unidos desenvolveram o primeiro transistor. Este, por sua vez, veio a substituir as válvulas (LOJEK, 2007) que eram usadas até então, pois geravam menos calor e eram consideravelmente menores. O "efeito transistor" foi tão grande, que no início da década de 50 houveram muitos esforços para melhoria dos transistores, que se tornaram mais sofisticados, confiáveis e acessíveis.

Com o avanço dos transistores, se tornou interessante começar a uní-los e fabricar circuitos inteiros utilizando apenas uma peça condutora. Estes circuitos integrados (CI) eram compostos por diversos dispositivos (WIDMANN et al., 2000), como diodos, resistores e capacitores. Conforme os CIs evoluíam, projetos eletrônicos que eram considerados inviáveis, passaram a ser desenvolvidos, tanto para uso comercial quanto militar.

As décadas seguintes foram cruciais no desenvolvimento da área de circuitos integrados, com o surgimento da primeira DRAM (*Dynamic Random Access Memory*), anunciada pela Intel e a primeira SRAM (*Static Random Access Memory*), anunciada pela Fairchild, no ano de 1970 (MAXFIELD, 2004). Apesar de ambas serem voláteis, ou seja, perdem a informação caso a energia elétrica seja removida, o que basicamente as difere é a necessidade da atualização das informações contidas na DRAM periodicamente, já na RAM estática este procedimento não é necessário. Pouco tempo depois, a Intel anunciou o primeiro microprocessador do mundo, chamado de 4004 ele continha 2.300 transistores e era capaz de executar 60.000 operações por segundo.

Os microprocessadores e as SRAMs são de grande importância até os dias atuais, já que a maioria das FPGAs atuais utilizam estes circuitos como base da sua arquitetura (HAUCK; DEHON, 2010). Com o advento dessas novas tecnologias, conhecidas

como "*computer-on-a-chip*", o tamanho dos dispositivos diminuíram significativamente, e a capacidade de processamento de informações complexas e em massa se tornou uma realidade.

As soluções oferecidas pela lógica programável são inviáveis caso implementadas utilizando outras abordagens de projeto, por isso, foi introduzido o conceito de PLD (*Erasable Programmable Logic Device*), que são circuitos integrados que contém portas lógicas e são unidos eletronicamente via software (BROESCH, 2012). Os PLDs foram introduzidos inicialmente na memória PROM (*Programmable Read-Only Memory*), onde eram implementados por circuitos lógicos. Assim como transistores, as linhas de entrada eram usadas como entrada e as linhas de dados como saída.

Os PLDs são subdivididos em duas categorias, os SPLDs (*Simple Programmable Logic Device*) e os CPLDs (*Complex Programmable Logic Device*). Basicamente o que os diferencia é que um CPLD é um conjunto de SPLDs interconectados e passíveis de implementação contidos no mesmo *chip*. Dentro da categoria de SPLD existem diversos dispositivos com diferentes características, até hoje existe uma certa confusão para designar qual categoria pertence um determinado dispositivo, já que a diferença entre eles é sutil.

Apesar de todos os dispositivos citados anteriormente, o primeiro a ser desenvolvido especialmente para a implementação de circuitos lógicos foi o PLA (*Programmable Logic Arrays*) (BROESCH, 2012), que veio para resolver as limitações existentes nas PROMs causadas pela forma com que as portas de entrada eram dependentes do número de funções AND/OR, o que foi melhorado nos PLAs, tornando ambos os *arrays*, AND e OR, programáveis, apesar disso torná-los lentos quando comparados aos PROMs.

Mesmo sendo particularmente útil para grandes projetos devido à sua facilidade de computar soma e produto de termos usando múltiplas saídas, os PLAs ainda eram lentos. Para resolver esse problema de velocidade, foi proposto o PAL (*Programmable Array Logic*), que, ao contrário da PROM, possui apenas o *array* AND programável (MAXFIELD, 2004), sendo o *array* OR definido previamente. Apesar de mais limitados, os dispositivos PAL foram de grande importância para as arquiteturas posteriores. Tanto PAL, quanto PLA ou PROM caracterizam bem os SPLDs, pois possuem um mesmo objetivo de prover bom desempenho a baixos custos.

Já os CPLDs, surgiram no final da década de 70, devido à grande demanda de aparelhos cada vez mais rápidos e poderosos. A arquitetura do CPLD é composta por

diversos PLDs de menor capacidade, unidos para realizar tarefas que eram incapazes de realizar individualmente. A ideia foi resolver o problema de velocidade dos SPLDs, unindo diversos deles através de canais programáveis (NAVABI, 2006), possibilitando a implementação de circuitos lógicos mais complexos.

Com características bem peculiares, cada PLD apresenta um melhor uso dependendo da aplicação que se pretende implementar. Cabe ao projetista pesquisar qual dispositivo se encaixa melhor no projeto pretendido, decidir quais requisitos são mais relevantes e projetar o *hardware* para o dispositivo escolhido. Uma outra classe de circuitos integrados que cresceu em paralelo foi o ASIC (*Application-Specific Integrated Circuit*), que como o nome sugere, era utilizado para apenas uma aplicação específica, tinha grande capacidade de processamento, mas não era reprogramável (ZEIDMAN, 2002), ou seja, uma vez gravado, não poderia ser alterado ou removido. A grande vantagem dos ASICs é que seus circuitos internos são muito rápidos e possuem uma alta densidade, com custo relativamente baixo.

É possível notar a discrepância entre as duas classes principais de circuitos integrados, uma possui um baixo poder computacional, mas por ser reprogramável permite que o processo de desenvolvimento seja mais rápido, e o custo de modificação depois de iniciado é pequeno. A outra classe já era capaz de cálculos complexos e robustos, mas necessitava de um processo de desenvolvimento longo, e o custo de uma alteração era altíssimo, devido a sua não-reprogramabilidade.

Em 1985, a Xilinx percebeu que era preciso um dispositivo que pudesse atuar como intermediário, sendo flexível e barato como os PLDs, mas ao mesmo tempo robusto e passível de implementação de muitas funções lógicas como os ASICs (ZEIDMAN, 2002). Assim, surgiu a primeira FPGA (*Field Programmable Gate Array*) abordando a complexidade dos ASICs, mas altamente reprogramável como os PLDs (MAXFIELD, 2004). O custo de projeto em FPGA costuma ser menor do que utilizando ASIC, o tempo para desenvolver todo o projeto é bem menor, além de que modificar o projeto mesmo que pronto, é mais fácil.

Um FPGA consiste em um grande arranjo de células configuráveis contidos em único *chip*. Cada uma das células contém uma capacidade computacional para implementar diferentes funções lógicas (ZEIDMAN, 2002). A parte do nome FPGA que referencia "*Field Programmable*" menciona o fato da programação ocorrer "em campo", ou seja, no local do usuário, diferente dos dispositivos onde a programação é feita apenas pelo fabricante, como em sistemas embarcados. Logo, FPGAs podem ser tanto configuradas em laboratório, como modificadas em seus locais de operação

(MAXFIELD, 2004). O termo "*Gate Array*" está relacionado ao arranjo de unidade lógica, que traz a herança dos ASICs nessa parte de sua arquitetura (TRIMBERGER, 2012).

Os FPGAs atuais chamados de *high-end* possuem memória e núcleos de processamento embarcados em cada bloco, além de alta velocidade de entrada e saída (MAXFIELD, 2004). O uso dos FPGAs são os mais diversos possíveis (BROWN et al., 2012) e o FPGA mais adequado deve ser escolhido após encontrar a medida certa comparando a flexibilidade e a complexidade envolvendo os blocos lógicos e suas interconexões. Outro fator a ser levado em consideração é se a aplicação tem um propósito específico ou geral, se é de tempo real ou não, entre outras características.

O crescimento dos FPGAs é notório. O que alguns anos atrás era utilizado apenas como intermediário entre componentes eletrônicos e controladores de barramento (STAVINOV, 2011), atualmente, após se tornarem mais rápidos e maiores, o custo dos FPGAs caiu, os tornando uma ótima escolha de mercado. Os FPGAs são hoje confiáveis até para o desenvolvimento de aplicações críticas, seja no ramo espacial, militar, automotivo ou hospitalar.

A.2 Descrição das Ferramentas

A.2.1 Altera Quartus II

O Quartus II é um software produzido pela Altera para desenvolvimento em PLDs, fortemente voltado para os projetistas e desenvolvedores que trabalham com os FPGAs da própria Altera. Nele, é possível implementar nas duas principais HDLs, VHDL e Verilog, compilar, simular no ModelSim e ainda transferir o código para um dispositivo físico. O Quartus II também possui funcionalidades para projetar e testar circuitos lógicos, podendo gerar o código em alguma HDL a partir do seu esquemático. Mais informações sobre a ferramenta podem ser encontradas em (ALTERA, 2007).

A.2.2 Mentor Graphics ModelSim

O ModelSim é um software produzido pela Mentor Graphics que permite a simulação das principais HDLs. Além da simulação em forma de ondas (*Waveform*), o ModelSim ainda mostra os processos ativos no programa simulado e também os sinais declarados, sejam eles de entrada, saída ou internos. Caso necessário, é possível abrir o código do programa e usar o ModelSim como ambiente de desenvolvimento. Nesse trabalho, foi utilizado o ModelSim que acompanha o Quartus II, mas ele tam-

bém pode ser instalado independentemente. Mais informações sobre a ferramenta podem ser encontradas em (MENTOR GRAPHICS,).

A.2.3 PRISM

O PRISM é uma ferramenta de Model Checking Probabilístico utilizada para modelagem formal e análise de sistemas que possuam comportamento estocástico. Para desenvolver os modelos, o PRISM utiliza sua própria linguagem, que é baseada em estados. Através da análise automatizada de uma vasta variedade de propriedades quantitativas dos modelos, é possível analisar sistemas com aplicações em diversos domínios, como protocolos de comunicação e segurança, sistemas biológicos, espaciais entre outros. Para o desenvolvimento dos modelos probabilísticos, o PRISM suporta DTMCs (*Discrete-Time Markov Chains*), CTMCs (*Continuous-Time Markov Chains*), MDPs (*Markov Decision Processes*), PAs (*Probabilistic Automata*) e PTAs (*Probabilistic Timed Automata*) e a linguagem de especificação das propriedades incorpora as lógicas temporais PCTL, CSL e LTL, assim como suas extensões e Recompensas (*Rewards*). Mais informações sobre a ferramenta podem ser encontradas em (KWIATKOWSKA et al., 2009).